Experimental Design and Process Optimization with R

Gerhard Krennrich

2020-02-10

Contents

Chapter 1

Introduction

The present document is a short and elementary course on the Design of Experiments (DoE) and empirical process optimization with the open-source Software **R**. The course is self-contained and does not assume any preknowledge in statistics or mathematics beyond high school level. Statistical concepts will be introduced on an elementary level and made tangible with R-code and R-graphics based on simulated and real world data.
So, then, what is DoE and why should the reader become familiar with the concepts of DoE?
Very briefly, DoE is the science of varying **many** experimental parameters in a systematic way to gain insight on how to further improve and optimize these parameters. Chapter 2 will show how and why multidimensional DoE techniques are superiour to the classical "one-dimensional" optimization approach. Chapter 6 will demonstrate why and how DoE can be combined with optimization. Finally, the use of DoE and optimization will be practically demonstrated in chapter 7 for improving the performance of a catalytic system.
Historically, Experimental Design started as a branch of statistics in the early years of the 20^{th} century and has meanwhile grown into a mature method with a plethora of applications in the experimental sciences. Consequently, there are many good and comprehensive books available about DoE, some of which we will make frequent reference to in the present text, namely (George E.P. Box, Norman R. Draper 1987), (D.C. Montgomery 2013) and (G.E.P. Box, W.G. Hunter, J.S. Hunter 2005). A more recent text with emphasis on the use of R in conjuction with DoE is (John Lawson 2015). Linear models are comprehensively covered, e.g., by the text book (A. Sen, M. Srivastava 1990). A general, however fairly technical text on linear and nonlinear statistical model building is the excellent book (T. Hastie, R. Tibshirani, J. Friedman 2009). (J.G. Kalbfleisch 1985) is a smooth introduction into statistics, probability and statistical inference.
The present text draws on these books and on many years of experience as a statistical consultant in the chemical industry. Most examples in this course

are therefore taken from applications and optimization projects in the chemical sciences.
The primarily intended readers of this document are chemists and engineers entrusted with empirical optimization in research and development. However, the presented methods and concepts are fairly generic and scientist working in other areas such as biology or the medical sciences might benefit from the text.
As to software, R, probably together with Phyton, is the only open-source software which combines the whole spectrum of DoE and optimization with the flexibility of a powerful script language that allows any kind of data pre- and postprocessing within one software environment. That makes, in my opinion, R superior to many commercial GUI based tools which often buy userfriendlyness at the expense of flexibility.

1.1 How to install R

The R-software can be downloaded free of charge from the R repository CRAN

Download R

An IDE (**I**ntegrated **D**evelopment **E**nvironment) is reqired for smoothly working with R. An IDE allows editing, running and debugging of R code and managing programm in- and output.
In principle any IDE can be used but we recommend R-Studio as the de-facto standard.

Get R-Studio IDE

The R-introduction at CRAN is a concise introduction into the R-language.
A short R-introduction

1.2 Some remarks on how to read the present text

This document is not an introduction into the R language, rather the document follows the philosophy of "learning by doing". In this spirit the above mentioned text R-introduction is recommended as a first reference together with the present R examples on DoE and optimization. As it is usually easier to modify existing code than writing code from scratch, it is hoped that the R-examples in this course will help learning both R and DoE more rapidly.
The course is divided into seven chapters. There is, however, one stand-alone chapter, chapter 5, which can be skipped by those readers not explicitly dealing with mixture problems. The final chapter 7 is a published, (Siebert M., Krennrich G., Seibicke M., Siegle A.F., Trapp O. 2019), real-world example combining many elements of DoE and optimization for improving the performance of a catalytic system. This application should encourage readers to use these powerful methods for the sake of their own projects.

Chapter 2

Basics of empirical model building

Statistics is the mathematical science dealing with uncertainty and change. There are many fields of modern statistics, but the branch of empirical model building became particular popular with the advent of modern machine learning methods. Machine learning and empirical model buiding aims at deriving functional representations (mathematical models) of large multidimensional data sets which can then be used for deriving insight into the data and/or predicting new cases. The following two chapters will introduce into the basic concepts of empirical model building, the underlying assumptions and some problems associated with this field of application.

2.1 Model building with historical data

It is an everyday experience that everything in the world is in a state of constant flux and subject to permanent change, a state schematically depicted in figure 2.1

The changing entities (variables) in 2.1 can be conceptually divided into X-, Y- and Z-variables:

$$\Delta X : \{\Delta x_1, \Delta x_2 ... \Delta x_I\};$$

$$\Delta Y : \{\Delta y_1, \Delta y_2 ... \Delta y_J\};$$

$$\Delta Z : \{\Delta z_1, \Delta z_2 ... \Delta z_K\};$$

$$I, J, K \in \mathbb{N}$$

$$X, Y, Z \in \mathbb{R}$$

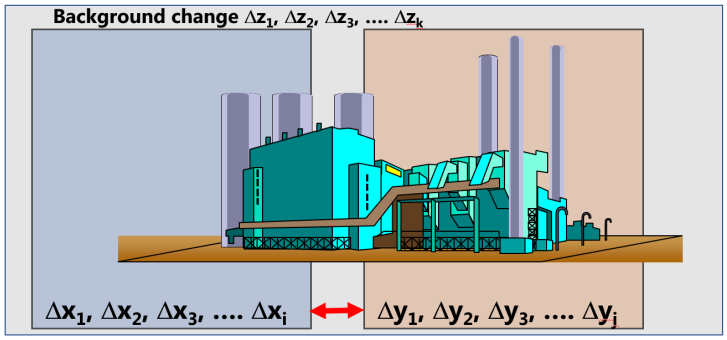

Figure 2.1: Analysing change in front of a changing background

In this notation the X-variables are considered to be the "independent" variables influencing the Y-variables (the responses ΔY responding to the changing X-variables, ΔX), while background variables ΔZ are assumed unkown and changing, too. Defining X,Y,Z for historical data is usually done based on scientific ground and knowledge of the system, because no statistical analysis of observational data can make such an assignment. In this model all variables are assumed changing, indicated by the prefix operator Δ.

Without loss of generality the relationship of one particular response Δy with ΔX, ΔZ and the link function $F()$ can be written:

$$\Delta y = F(\Delta X, \Delta Z) = F(\Delta x_1, \Delta x_2 ... \Delta x_I; \Delta z_1, \Delta z_2 ... \Delta z_K) \tag{2.1}$$

Under the assumption that background variables ΔZ do not affect foreground variables ΔX, equation (2.1) can be factorized into $f()$ and $g()$[1]

$$\Delta y = f(\Delta X) + g(\Delta Z) = f(\Delta x_1, \Delta x_2 ... \Delta x_I) + g(\Delta z_1, \Delta z_2 ... \Delta z_K) \tag{2.2}$$

Under the weak assumption that the background change ΔZ is sufficiently small and with a_k being the first order derivative of $g()$ at z_k, the second term in (2.2) can be simplified

[1] Factorization requires that all mixed partial derivaties must vanish, formally

$$\frac{\partial^k}{\partial z_1 ... \partial z_k} \left(\frac{\partial^i f(x_1, ... x_I; z_1, ... z_K}{\partial x_1 ... \partial x_i} \right) = 0; \forall\ i, k \leq I, K$$

, so background variables Z are not allowed to moderate foreground variables X thereby rendering the model time-invariant.

$$\Delta y = f(\Delta X) + \sum_{k=1}^{K} a_k \cdot \Delta z_k = f(\Delta x_1, \Delta x_2 ... \Delta x_I) + \sum_{k=1}^{K} a_k \cdot \Delta z_k \qquad (2.3)$$

By the Central Limit Theorem - sums of arbitrarily distributed random variables are asymptotically normal distributed, $\sum_{k=1}^{K} a_k \cdot \Delta z_k = \epsilon \sim N(0, \sigma^2)$[2] - (2.3) can be further simplified and becomes

$$\Delta y = f(\Delta X) + \epsilon = f(\Delta x_1, \Delta x_2 ... \Delta x_I) + \epsilon;\ \epsilon \sim N(0, \sigma^2) \qquad (2.4)$$

(2.4) is the formula usually found in text books on statistical learning and empirical model building: The observed response Δy is a function of the influential factors ΔX plus some random noise from a normal distribution with unknown variance σ^2. In deriving the base equation of empirical model building, $\Delta y = f(\Delta X) + \epsilon$, two strong assumptions were made, namely

1. Factorization of (2.1)
   - This is a very strong assumption and ensures that the model *f()* is invariant with respect to the backgound (space and time). It ensures generalisability of *f()* in space and time, an assumption usually taken for granted of the laws of nature.
2. The assumption of normality in (2.4)
   - Depending on the dimension K of Z and K being sufficiently large, and the variability of the background variables Z, the normality hypothesis might often be an assumption too tight. However, non-normal data can be handled within the framework of empirical model building, hence normality of the random term is not a crucial hypothesis.

However, these limitations are by far not the most crucial when working with historical data.
What can turn historical model building and inference[3] into a nightmare are the mutual dependencies between the X-variables which will render the model building process often difficult and inconclusive. From a more fundamental point of view, working with historical data suffers from not knowing the source of variance ΔX: There is change ΔX and ΔY in front of a changing background ΔZ, and it remains unclear from what source the change in ΔX has arisen[4] (and the same is, of course, true for ΔZ and ΔY). Compared with this problem, the

[2] see (J.G. Kalbfleisch 1985) p. 237

[3] Inference refers here to the process of separating significant from nonsignificant variables. The latter are usually excluded from model building as non-informative

[4] An example might be helpful at this point. Say we have some chemical process data consisting of *selectivity* and *temperature*. From first principles we know that temperature affects selectivity and it seems natural to set up the model *selectivity=f(temperature)* with selectivity being the response Y and temperature being the independent factor X. Here, the assumption is that the variability of the response results from the variability of X. However, if there is a controller in the plant controlling the temperature as a function of the selectivity our model assumption is wrong, because the variability of temperature results from the variability of the selectivity and not vice-versa. This is not something the data can tell. This information

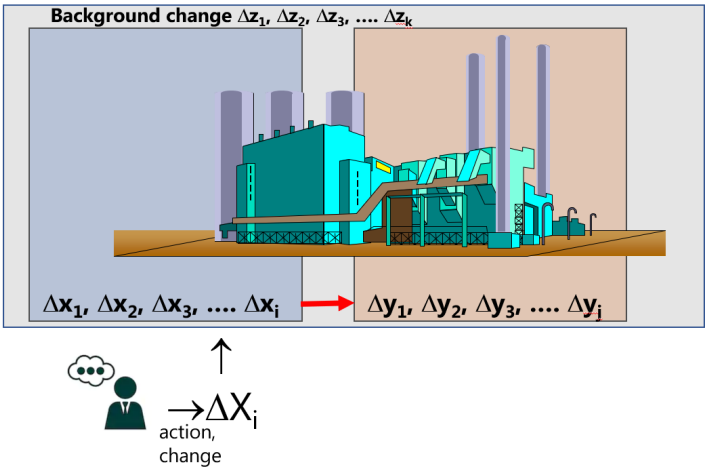

Figure 2.2: Introduction of variance by an agent with DoE

problem of finding an adequate functional representation $f()$ between Δy and ΔX is a minor one that can be solved with modern machine learning methods[5].

2.2 Model building based on well-designed data

In the previous chapter 2.1, the basic elements of empirical model building were introduced. All concepts derived there also apply to DoE with one fundamental difference: The source of variance ΔX in a well-controlled experiment is known with the experimenter becoming the indubitable source of that variance. This process is schematically depicted in figure 2.2.

The researcher selects from a set of potential variables X a subset, X', deemed relevant for the problem at hand and varies these parameters (that is $X' \rightarrow \Delta X'$) within carefully chosen boundaries Δ with an experimental design while trying to keep the background variables, if known, as constant as possible
($Z = const. \rightarrow \Delta Z = 0$).
The X-variables are deliberately and systematically varied to study, how this variation affects Y[6]. By carefully selecting the support point in a linear space $\mathbb{R}^N$ the effects ΔX can be separated from the background noise and quantified as an analytical expression $\Delta y = f(\Delta X) + \epsilon$

must come from outside, from the context of the data. In addition, the process might be affected by ambient conditions unknown to the plant operators, some "lurking" background variables Z.

[5] In chapter 7 the Random Forest as a flexible machine learning method will be used for analysing historical data.

[6] The concept of causality is based on the somewhat hidden assumption that change always results from and leads to change. In statistical terms: Without variation no co-variation. Knowledge of whether and how entities are related is based on change.

The science of designing and analysing multivariate experiments in N-dimensional space $\mathbb{R}^N$ with the R-software will be elucidated in the following chapters.

In essence, the process just described is the scientific method usually attributed to Bacon (1561-1626) who allegedly expressed this thought first. It boils down to the simple recipe: Bring together new conditions in a controlled experiment and see what condition(s) arise(s) from that action.

Chapter 3

Two introductory examples

Before discussing the statistical details of linear models, a model class with very wide use in the statistical sciences and the major tool of DoE, a few thoughts about the classical one-factor-at-the-time (OFAT) optimization are appropriate, as they can help introducing some concepts of statistical model building and optimization.

3.1 Example I: Optimization in one dimension

In this chapter a simple one-dimensional example will be used to introduce some helpful statistical concepts, namely

1. What is a statistical model?
2. Pure Error (PE) versus model bias (Lack of Fit, LoF)
3. Inter- and extrapolating statistical models and identifying directions of ascent/descent

Imagine a one-dimensional optimization project with the objective of finding the concentration, $conc^*$, maximizing *yield=f(conc)*. Design space (factor range), budget (number of experiements) and design (DoE) are given as follows

1. Chemically sensible range for *conc*
   - 1 - 3 mmol/l
2. Research budget
   - 9 runs
3. DoE1
   - spread runs equidistantly across range and measure *yield* at the nine concentration values

The results from runnning the equidistant one-factor design, DoE1, are listed in table 3.1 and depicted in figure 3.1 as simple *yield~conc* scatter plots with four different models (interpretations) superimposed.

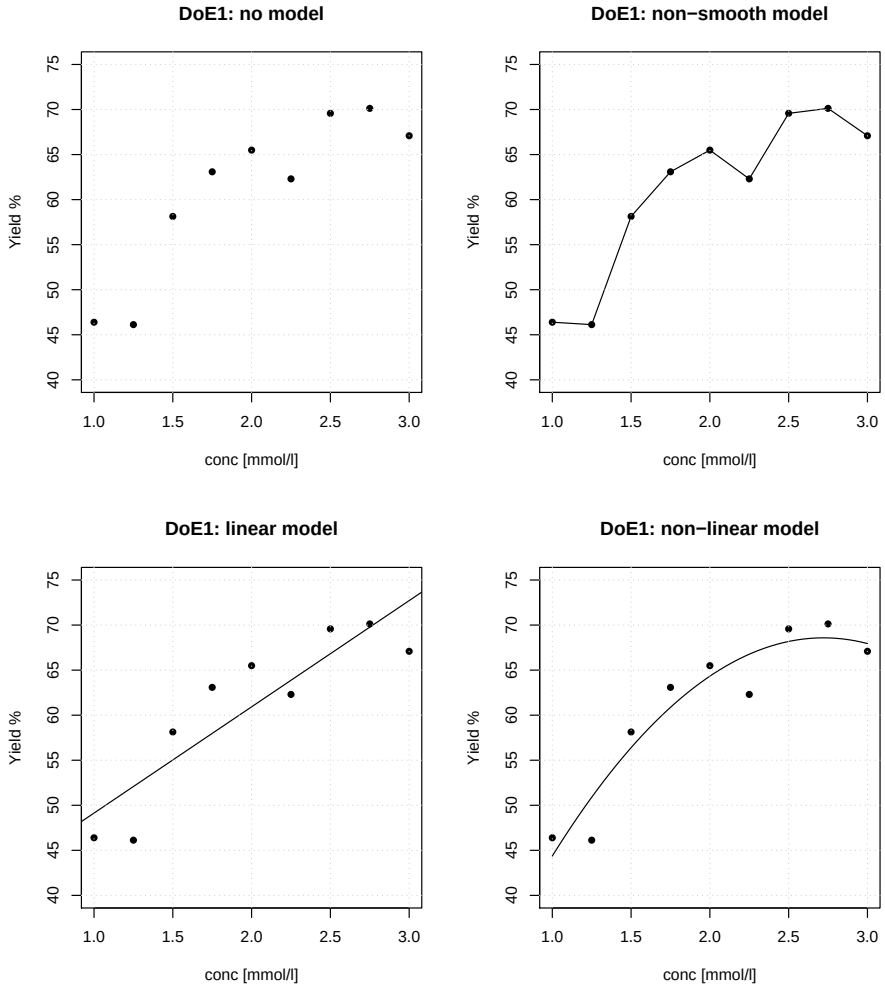

Figure 3.1: Different models (interpretations) of the data from table 3.1; DoE1

Table 3.1: Results of one-dimensional optimization of yield over concentration, DoE1; Max[*yield=f(conc)*]

Obsnr	conc	yield
1	1.00	46.39
2	1.25	46.13
3	1.50	58.13
4	1.75	63.09
5	2.00	65.49
6	2.25	62.30
7	2.50	69.57
8	2.75	70.13
9	3.00	67.09

Figure 3.1 interprets the raw data in four different way

1. No model
   - data is plotted "as is" without making any model assumption.
2. Non-smooth "model"
   - This model consists of a solid line connecting data points and helps the observer in recognizing patterns such as trends, minima and maxima. It ignores experimental uncertainty by assuming the measurements to be exact and and free from errors. The slope of the connecting line (the gradient) changes at each support point in a steplike manner, a behaviour unlikely to be found in nature[1].
3. Linear model
   - The data is being interpolated with a straight line so as to fit the data in a least square sense. Mathematically, the model line is represented by a linear equation, $yield = a_0 + a_1 \cdot conc$ with intercept and slope parameter a_0 and a_1, respectively.
4. Non-linear model
   - Here the data is being interpolated with a second order polynomial to account for non-linear effects, parametrically represented by the equation $yield = a_0 + a_1 \cdot conc + a_2 \cdot conc^2$

The regression models, #3 and #4, are clearly advantageous when it comes to interpolating, extrapolating and optimizing the models and can be used to identify directions of ascend/descend. The model property of revealing gradients information is even more important in higher dimensions as the number of directions in $\mathbb{R}^k$ grows exponentially ($\approx 2^k$). Finally, these smooth models account for experimental uncertainties (errors) as a natural part of the least squares fitting process as will be described later in more detail.

[1]Smoothness of nature finds expression in the Latin phrase "Natura non facit saltus": nature does not make jumps.

Table 3.2: Results of one-dimensional optimization of yield over concentration, $\max_{conc} yield = f(conc)$, with an alternative design DoE2 consisting of three equidistant triple replicates in the concentration range [1-3 mmol/l]

obsnr	conc	yield
1	1	46.39
2	1	46.21
3	1	47.07
4	2	59.25
5	2	65.49
6	2	68.25
7	3	65.63
8	3	60.43
9	3	67.09

However, figure 3.1 reveals some weaknesses of the "equidistant" design when it comes to discriminating between model #4 and model #3, i.e. deciding which model suits the data best given experimental uncertainty. While model #4 indicates non-linear saturation or a maximum in the concentration range [2-3 mmol/l], model #3 suggests a linear increase beyond the upper bound[2]. The difficulty in distinguishing #3 from #4 lies in the design, DoE1, which makes it hard to differentiate between **model bias** and **pure error**.
Table 3.2, shows an alternative design, DoE2, generated by the above model and consisting of three triple replicates at three different concentration values conc={1,2,3}. DoE2 is depicted in figure 3.2 as scatter plots with linear and a non-linear models superimposed.

DoE2 renders the task of discriminating between model #3 and model #4 easier: Figure 3.2 suggests the linear model to be biased, because the triple replicates at the center are not approriately described, whereas the non-linear model describes the data appropriately in the experimental concentration range, thereby supporting the saturation hypothesis from above. With $\hat{y}$ denoting the model predictions and $\bar{y}$ the mean average of the replicate triple, the model bias (=systematic error, **L**ack **o**f **F**it) is defined as the sum of squares (SS) $SS_{\mathrm{LoF}} = (\hat{y} - \bar{y})^2$ while the **P**ure **E**rror (statistical error, PE) is given by $SS_{\mathrm{PE}} = \sum_i (y_i - \bar{y})^2$.

Figure 3.3 gives a graphical overview of the statistical elements that differentiates **L**ack **o**f **F**it from **P**ure **E**rror.

[2]the difference between model #3 and #4 is especially important when it comes to finding conditions maximizing yield. #4 suggests a maximum in the present design space, while #3 points to a maximum somewhere outside this space.

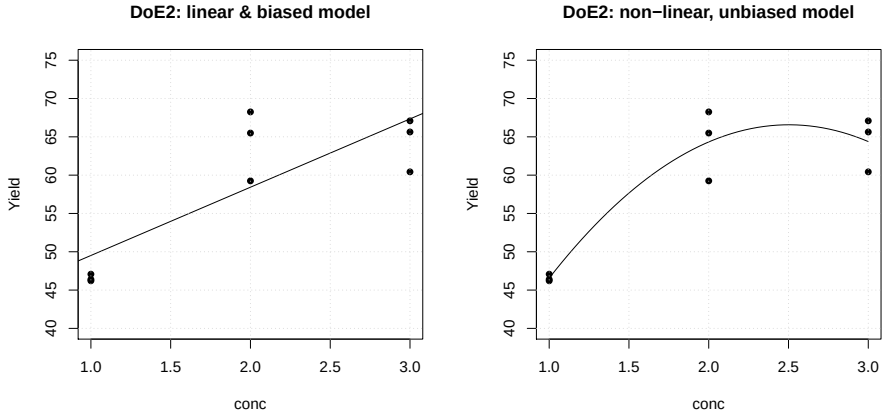

Figure 3.2: Linear biased and non-linear unbiased models of the design DoE2

3.2 Controlling nuisance parameters

There is a proverbial joke among statisticians about the relationship between the population size and storck numbers, allegedly observed in the German town Oldenburg between 1930-1936 (Ornithologische Monatsberichte, **44**, Nr. 2 (1936), **48**, Nr. 1 (1940), Berlin). The raw figures of this groundbreaking work are plotted in Figure 3.4 and reveal a strong and significant relationship between population and number of storcks.

Now the question arises whether figure 3.4 is in support of the hypothesis that the storcks are bringing the babies and thereby affecting the size of the population? The correlation in figure 3.4 is much too strong to be dismissed as a chance result, and so there must be a hidden confounder explaining the strong relationship. Indeed, this confounder is easily identified to be the time as figure 3.5 illustrates.

As figure 3.5 shows, human and storck population share a common trend in time and that, by transitivity, shows up as a strong correlation between both variables

$$Birth \leftrightarrow time \leftrightarrow Storck$$

$$\Rightarrow Birth \leftrightarrow Storck$$

Whenever two variables share a common trend in time, they will be correlated nescessarily. This property makes time[3] to the most dangerous confounder in the experimental sciences, and statisticians thought about ways and means to control this confounder. Fortunately, randomization is an effective way to eliminate

[3]of course time as such is not responsible for the observed correlation between storck and birth numbers. Living conditions of storcks and humans might have improved in the time range thereby creating the observed population trend.

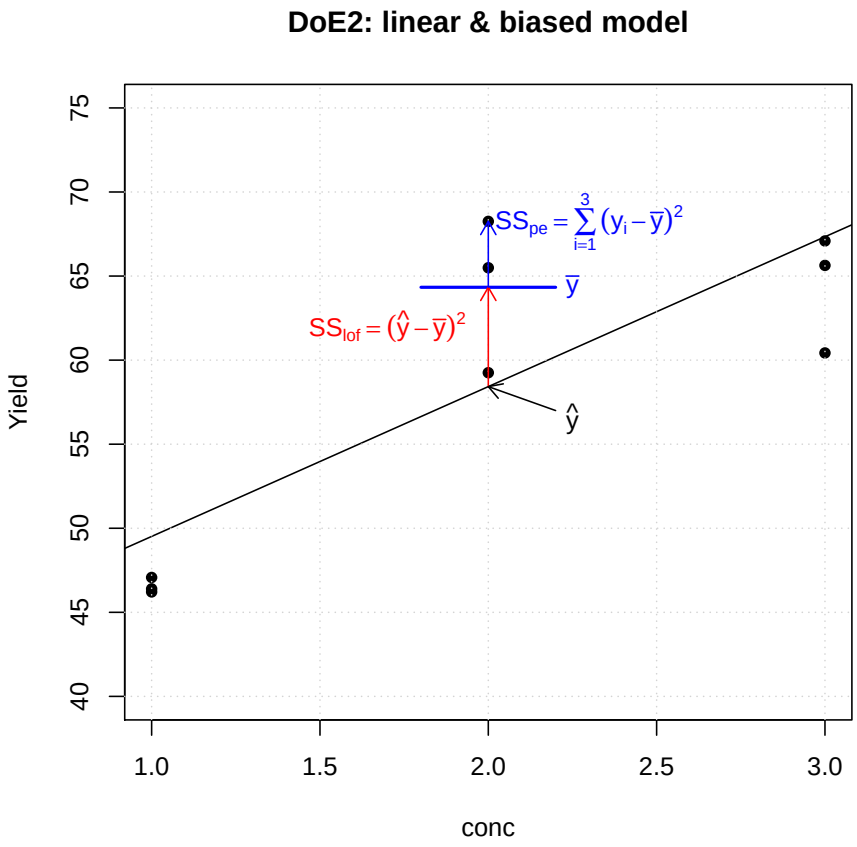

Figure 3.3: Lack of Fit (LoF, model bias) and Pure Error (PE)

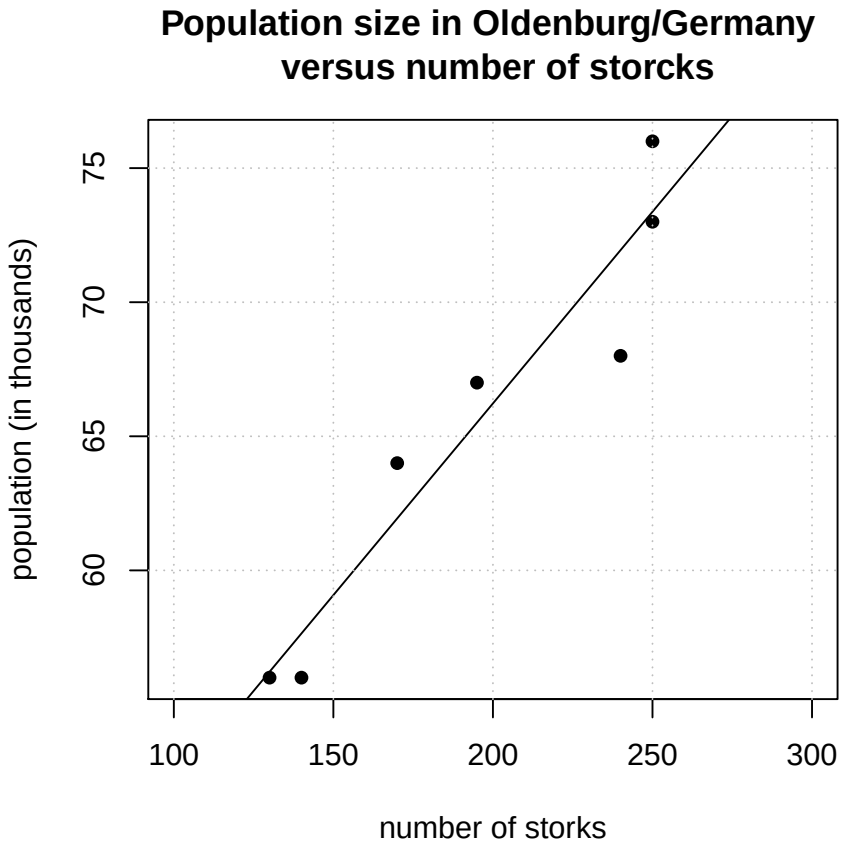

Figure 3.4: Population size in Oldenburg/Germany versus observed number of storcks

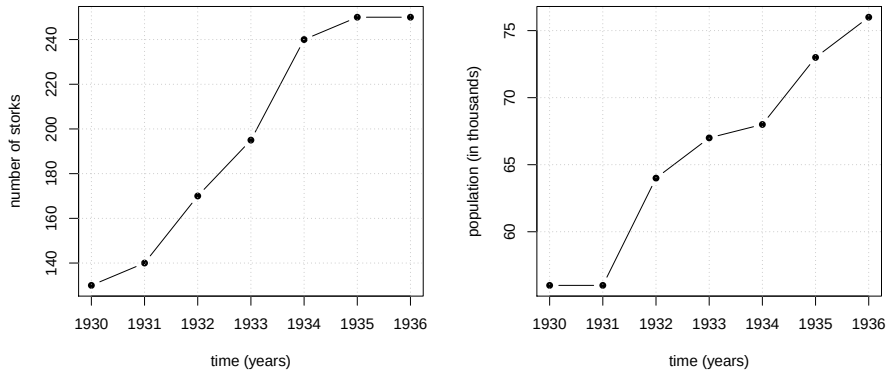

Figure 3.5: Number of storcks and population size plotted versus time in Oldenburg/Germany between 1930-1936

Table 3.3: OFAT optimization of yield over temperature, $\max_{temp}[yield = f(temp)]$, without randomization of the design

run.order	temp	yield
1	100	56.4
2	105	62.8
3	110	67.3
4	115	59.0
5	120	69.3
6	125	71.5

potential bias from serial effects.
Controlling time effects by randomization is best understood with a simple one-dimensional example: A researcher wants to investigates the effect of temperature on the yield of a chemical reaction and sets up a simple One-Factor-at-the-Time (OFAT) design consisting of the six trials listed in table 3.3.

"Eye-ball regression" of table 3.3 suggest a positive effect of temperature on yield. Unfortunately, this is not the only possible reading of the experimental results, and the outcome can be alternatively attributed to the positive effect of the run.order (or, here, time) on yield. Even worse, any linear combination of time and temperature, $a_1 \cdot \text{temp} + a_2 \cdot \text{run.order};\ a_1, a_2 \in \mathbb{R}$, is a valid reading of the outcome: The increase can be exclusively explained by time and temperature, however, any other "effect mixture" is similarily valid, and no statistical method

Table 3.4: OFAT optimization yield over temperature, $\max_{temp}[yield = f(temp)]$, with randomization of the design

run.order	temp	yield
1	125	71.5
2	100	56.4
3	120	69.3
4	110	67.3
5	115	59.0
6	105	62.8

can identify the individual coefficients a_1, a_2. In the design, table 3.3, time and temperature are perfectly confounded ($r_{temp,time} = +1$[4]), and thereby the individual effects become hidden. Confounding run.order and temp must be and can be avoided by "decorrelating" time and temperature by randomizing the experimental design as shown in table 3.4.

Randomization decorrelates time and temperature ($r_{temp,time} = -0.37$) and the regression model $yield = b_0 + b_1 \cdot temp + b_2 \cdot time$ can now be jointly identified as $yield = 10.7(\pm 30.2) - 0.18(\pm 1.25) \cdot time + 0.48(\pm 0.25) \cdot temp$. Randomization thus becomes an important instrument in DoE for controlling the influence of "confounders in time" and should be applied whenever possible. Usually, experimenters do not like randomization, as the logistics of the experimental program become more complex. However, the value of randomization exceeds by far the additional amount of logistics incurred. In chemistry, hidden time effects, such as temporal loss of catalytic activity or deterioration of analytical instruments over time, are realities not to be ignored. Nevertheless, there are situations where randomization cannot be applied such as running DoE on large scale plants[5]. The consequence of lacking randomization will be, that all effects will be temporally biased, provided, of course, that serial effects are present.

3.3 Example II: Optimization in two dimensions

Section 3.1 and 3.2 introduced some helpful statistical concepts in the process of statistical model buiding with a one-dimensional example.
In the present section (3.3) the one dimensional example from section 3.1 will be augmented by another dimension for discussing the shortcomings of the one-factor-at-the-time method (OFAT) in higher dimensions. The problem is

[4] r denotes the Pearson correlation coefficient. A value of +1 indicates that *temp* and *time* are perfectly colinear and their relationship can be expressed by the linear equation $temp = b_0 + b_1 \cdot time$ with $b_1 > 0$.

[5] Plant operators will certainly not be amused by suggesting to randomly increase and decrease the temperature over the course of the experimental programm. In addition, randomly varying the temperature can be a cause of catalytic activity loss.

Table 3.5: OFAT: vary conc|temp=142

temp	conc	yield
142	76	77.5
142	85	80.9
142	94	81.4
142	104	78.6

Table 3.6: OFAT: vary temp|conc=91

temp	conc	yield
138	91	76.0
142	91	81.6
148	91	85.4
152	91	84.8

stated as an optimization problem, namely finding optimal reaction conditions $conc^*$ and $temp^*$ maximizing yield.
Sensible and feasible factor ranges are given as follows

1. *conc*: {76 - 104 mmol/l}
2. *temp*: {138 - 152 ^{0}C}

Faced with such a problem, many researchers, not being familiar with multivariate methods, would intuitively turn to OFAT optimization by varying one variable first while keeping the other factor(s) constant.

In the present example, the researcher decided to start the OFAT process by keeping the temperature constant at *temp*=142[^{0}C] and varying the concentration at four different levels. This OFAT design is shown in table 3.5 and depicted in figure 3.6.

Figure 3.6 suggests a conditional yield optimum at $conc^*|(temp = 142[^0C])) \approx 91[mmol/l]$; $yield^* \approx 82[\%]$.
The next optimization round varies the temperature given the conditional OFAT optimum $conc^* \approx 91$ $[mmol/l]$ as shown in table 3.6 and figure 3.7.

Figure 3.7 reveals a maximum at $temp^* \approx 149[^0C]$ with an expected yield, $yield^* \approx 86[\%]$. At this stage the OFAT optimization process stops and reports the OFAT maximum

$$\boxed{conc^* = 91[mmol/l];\ temp^* = 149[^0C];\ yield^* \approx 86[\%]}$$

OFAT optimization is clearly a suboptimal optimization strategy as can be seen in figure 3.8 that shows the response surface of the model $y = f(conc, temp)$

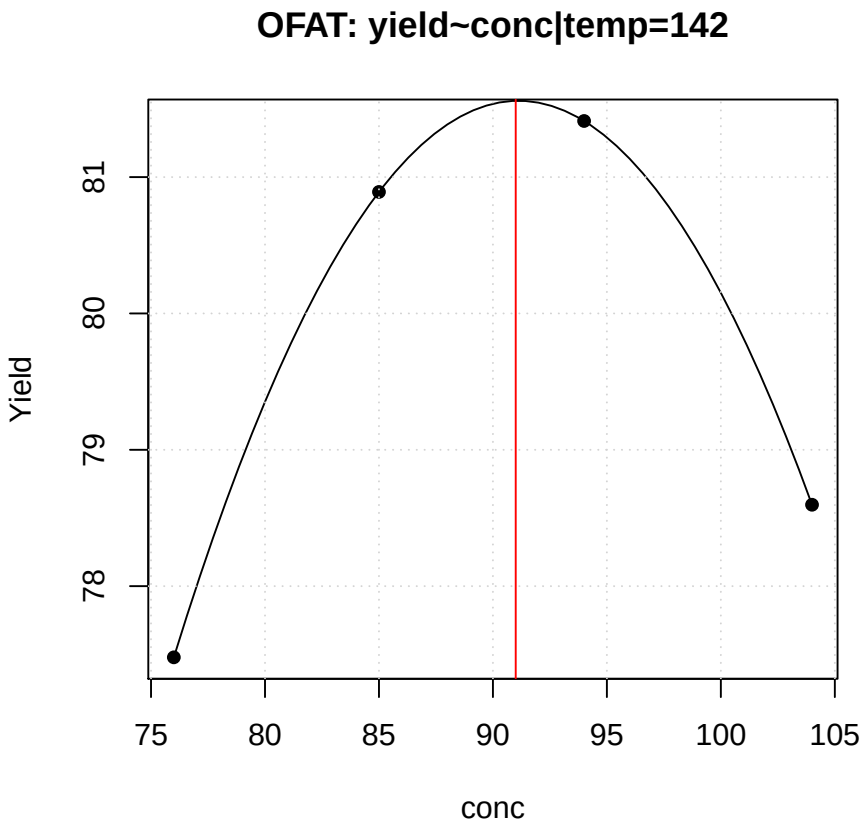

Figure 3.6: Conditional OFAT optimization $\max_{conc}[yield = f(conc|temp = 142)]$. The red line indicates the location of the OFAT maximum

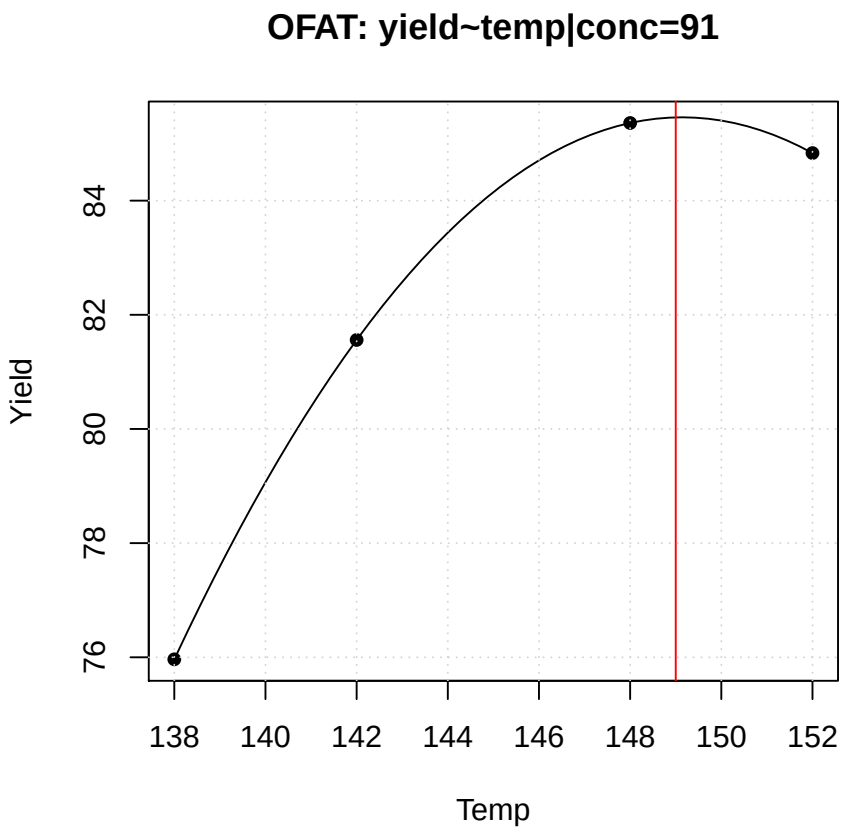

Figure 3.7: Conditional OFAT optimization $\max_{temp}[yield = f(temp|conc^* = 91)]$. The red line indicates the location of the OFAT maximum

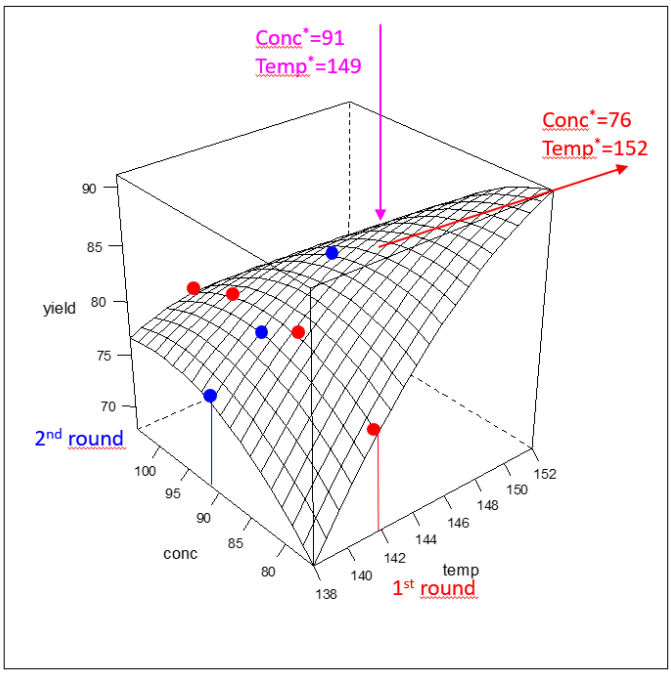

Figure 3.8: OFAT optimization seen from a multivariate perspective. The OFAT optimum is labelled magenta. OFAT trials are printed as blue and red dots. The direction of steepest ascent is indicated by the red arrow with the mutivariate optimum labelled red at the upper right corner of the design space

together with the overlayed OFAT optimization trials from tables 3.5 and 3.6. The "true" maximum within experimental space is found at the upper and lower bound of *temp* and *conc*, i.e. the true local optimum is

$$\boxed{conc^* = 76[mmol/l];\ temp^* = 152[^0C];\ yield^* \approx 91[\%]}$$

OFAT actually slices the experimental space thereby ignoring (because of not knowing) the local topology of the response surface. The local topology (response surface) depicted in figure 3.8 was obtained with a multivariate experimental design in the given factor ranges and suggests to increase the temperature and to decrease the concentration so as to further increase the yield beyond 91%.

In high-dimensional space the OFAT optimization method gets even more laborious, cumbersome and ambigious. Given a set of process factors $x_1, x_2, x_3, ...x_I$, the univariate OFAT process starts by ambiguously selecting one variable to be varied while keeping the other factors constant at specific values, formally

$$\Delta x_i | x_{j \neq i} = \text{const}_j$$

and then works its way throught the whole set of variables $x_1, x_2, x_3, ...x_I$. This process is tedious and error prone and will almost certaintly miss local optima with higher performance in high-dimensional space.

3.4 An introduction into linear statistical model building

In this chapter the basic principles of linear least squares models will be introduced, one of the most versatile statistical modeling class. Linear models have wide use in the empirical sciences and can be used for predictions, inference and optimization.

3.4.1 The elements of statistical inference

Statistical inference is, shortly stated, the method for making rational decisions in the presence of uncertainty and is an integral part of empirical model building and a prerequisite for understanding linear models. Fortunately, understanding one statistical test is sufficient for understanding all statistical tests, and there are many, as the logic of statistical testing turns out to be the same for all tests. As a start let us consider the normal distribution with its probability density function (*pdf*) given by the analytic expression (3.1)[6]

$$pdf(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.1}$$

The normal distribution has two free parameters:

1. The location parameter μ which determines the location of the maximum of the pdf.
2. The dispersion parameter σ which determines the spread (width) of the distribution.

The normal distribution, equation (3.1), can be abbreviated by $N(\mu, \sigma)$. Two examples of this well-known, bell-shaped distribution are depicted in figure 3.9.

Finite samples of size N from the normal distribution $N(\mu, \sigma^2)$ can be used as estimators of the true, however unknown parameters μ and σ^2. This process, estimating some true parameters, here μ and σ^2, from population parameters of finite size, here $\bar{x}$ and s^2, is called inference in statistics. Unbiased population estimators for the location and dispersion parameters of the normal distribution are the well-known sample mean $\bar{x}$, equation (3.2), and the sample variance s^2, equation (3.3), with asymptotic properties $\lim_{N\to\infty} \bar{X} = \mu$ and $\lim_{N\to\infty} s^2 = \sigma^2$.

[6] There are many probability distributions in statistics but the normal distribution is by far the most important one as a result of the Central Limit Theorem: Sums of arbitrary distributed random variables will be asymptotically normal distributed, see the arguments from section 2.1.

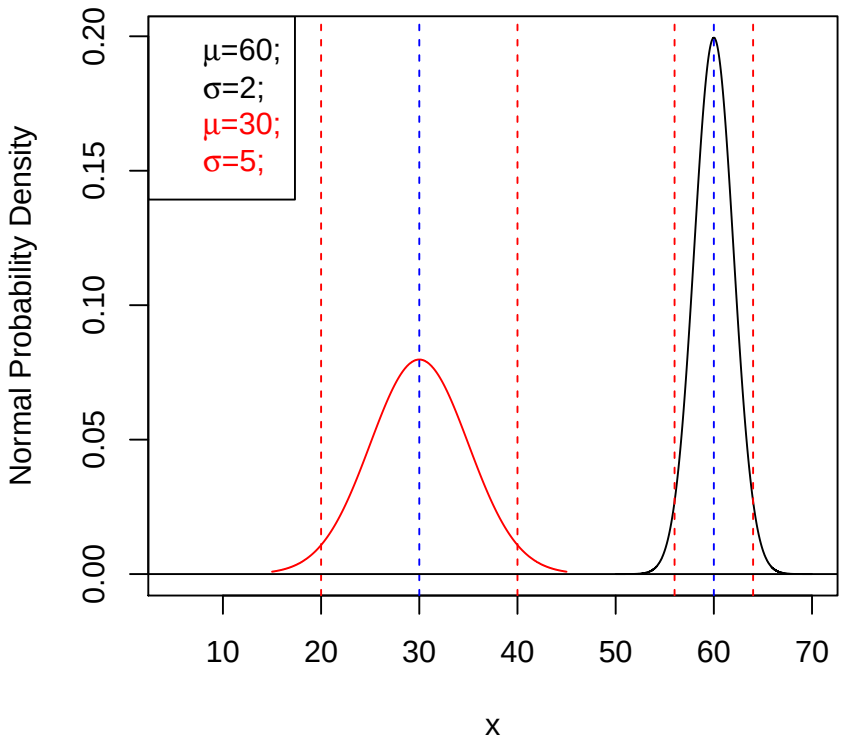

Figure 3.9: Normal probability density with two different location (blue) and dispersion parameters; $\pm 2\sigma$ limits are labelled red

Better known as the sample variance, s^2, is the standard deviation s, the square root of the variance s^2.

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3.2}$$

$$s^2(x) = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{3.3}$$

Sample averages and sample variances are themselve random variables, and they will always show up differently when repeating the sampling process. From their limiting properties, it is clear, that they will finally approach the true values, μ, σ^2, with increasing sample size N[7].

The essence of statistical inference can be understood with the following two sample experiment.
Suppose a researcher has developped a new process and wants to compare (test) the new method against the old standard. He or she produces ten specimen with the old and new method. There are ten samples N_{old}=10 and N_{new}=10 with average outcome, say, $\bar{X}_{old} \pm s_{old} = 133.08 \pm 4.98$ and $\bar{X}_{new} \pm s_{new} = 139.41 \pm 5.34$. Given these experimental results, the question is whether there is enough evidence for claiming both methods different, or alternatively to judge both outcomes the same within experimental error. This question can be answered by creating a test statistics from the experimental elements $N_{old}, N_{new}, \bar{X}_{old}, \bar{X}_{new}, s_{old}, s_{new}$, here the t-statistics. The recipe for calculating the t-statistic is shown in equation (3.4) with old, new indexed by 1 and 2.

$$s_p = \sqrt{\frac{(n_1-1)\,s_{X_1}^2 + (n_2-1)\,s_{X_2}^2}{n_1+n_2-2}}\, t = \frac{\bar{X}_1 - \bar{X}_2}{s_p \cdot \sqrt{\frac{1}{n_1}+\frac{1}{n_2}}} \tag{3.4}$$

It is worthwhile inspecting the equations (3.4) a little closer. The first part of the calculation consists in deriving a weighted pooled standard error s_p while ensuring that larger samples get more weight in the calculation. The second part, calculating the test statistics t, is in essence the distance between the two sample distributions expressed in units of the pooled standard error, a unit free number. Evaluating equation (3.4) with $N_{1,2}$, $\hat{x}_{1,2}$ and $s_{1,2}$ gives a t-value of $t_{1,2}$=-2.740403. The following R-snippet shows how the two samples were generated and how the t-value was calculated with the user function "t.val()" (note that both samples truly differ by $\Delta\mu = 5$ with "true" uncertainty $\sigma = 5$).

[7]For instance, it can be shown that the population mean of size N is normally distributed with $N(\mu, \frac{s}{\sqrt{N}})$. As a consequence, a doubling of the precision, $\frac{s}{2}$, requires quadrupling the experimental effort.

```
rm(list=ls())
set.seed(1234)
t.val <- function(pop1,pop2){
  x1    <- mean(pop1,na.rm=T)
  x2    <- mean(pop2,na.rm=T)
  n1    <- length(pop1[!is.na(pop1)])
  n2    <- length(pop2[!is.na(pop2)])
  s2.1  <- var(pop1,na.rm=T)
  s2.2  <- var(pop2,na.rm=T)
  s.pool <- sqrt( ( (n1-1)*s2.1 +  (n2-1)*s2.2  ) / (n1+n2-2))
  t      <- (x1-x2)/(s.pool*sqrt( 1/n1 + 1/n2  ) )
  return(t)
}
old    <- rnorm(10,135,5)
new    <-  rnorm(10,140,5)
cat("Old(N=10):",round( mean(old),2),"+/-", round(sd(old),2))
```

```
## Old(N=10): 133.08 +/- 4.98
```

```
cat("New(N=10):",round(mean(new),2),"+/-", round(sd(new),2))
```

```
## New(N=10): 139.41 +/- 5.34
```

```
cat("t-value=",t.val(old,new)) #
```

```
## t-value= -2.740403
```

As such, the value of t=-2.74 does not tell much, but it would tell more if the distribution of t were known under the assumption that both samples come from the same distribution. This distribution can easily be obtained by Monte Carlo simulation as shown with the following R-code. The code generates 100000 t-values by sampling from a normal distribution, N($\mu = 0, \sigma^2 = 1$), with sample size $N_{old,new}$=10 and shows the resulting distribution as histogram, figure 3.10. It then counts the number of t-values exeeding of falling short of $\pm$2.74 (that is N|abs(t)>2.74) and reports this as a percentage value, $\alpha = 0.01312$.

```
t.val <- function(pop1,pop2){
  x1    <- mean(pop1,na.rm=T)
  x2    <- mean(pop2,na.rm=T)
  n1    <- length(pop1[!is.na(pop1)])
  n2    <- length(pop2[!is.na(pop2)])
  s2.1  <- var(pop1,na.rm=T)
  s2.2  <- var(pop2,na.rm=T)

  s.pool <- sqrt( ( (n1-1)*s2.1 +  (n2-1)*s2.2  ) / (n1+n2-2))
  t      <- (x1-x2)/(s.pool*sqrt( 1/n1 + 1/n2  ) )
  return(t)
}
```

```
set.seed(1234)
t <- NULL
for (i in 1:100000) {
  old <- rnorm(10,0,1)
  new   <- rnorm(10,0,1)
  t   <- c(t,t.val(old,new))
}
hist(t, breaks=60,col="lightblue",main="")
abline(v=c(-2.740403,0,2.740403),
       lty=c(2,1,2),col=c("red","black","red"))
box()
```

```
prop.table(table(abs(t)>2.740403))
```

```
##
##   FALSE    TRUE
## 0.98688 0.01312
```

What does α mean/tell?
It is the likelihood that a value larger than +2.74 or smaller than -2.74 will be observed under the **Null-hypothesis H0**, that is when there are truly no effects and both mean values, $\bar{x}_{1,2}$, come from the same distribution.
α is also called the **False-Positive-Rate** or **Type-I Error Rate**, the chance of taking an effect for real when it is not. In the present case α is with $\alpha = 0.01312$ very small, $\approx 1\%$, and we tend to reject the **Null-Hypothesis H0** in favour of the **Alternative-Hypothesis H1** which states that there is a "true" underlying effect, and the underlying parameters μ_{old}, μ_{new} are truly different. However, it must be kept in mind that $\alpha = 0.01$ indicates a 1% chance that both processes are not different and potentially false-positive, so, ultimately, the test cannot and does not lead to certainty, but rather quantifies the certainty by reporting the **False-Positive-Rate** α. There are many tests in statistics such as, e.g., t-, F- and z-test, and they all come with the same logic:

1. One calculates from the data a statistics (a t-, F- or z-value, a real number).
2. One knows the distribution of the statistics under H0 and compares the statistics in hand with the distribution.
3. Depending on a preset decision level α one either rejects or accepts the **Null-Hypothesis H0** thereby accepting or rejecting the **Alternative Hypothesis H1**.

Usually there is no need to derive the test distribution by Monte Carlo simulation as virtually all important distributions are known in closed form and can be obtained analytically (however, Monte Carlo simulation has its place and is often the only method for deriving test distributions for non-normal data).
Figure 3.11 shows probability density plots of the t-distribution for different sample sizes. As a rule of thumb the t-distribution equals N(0,1) for N>30.

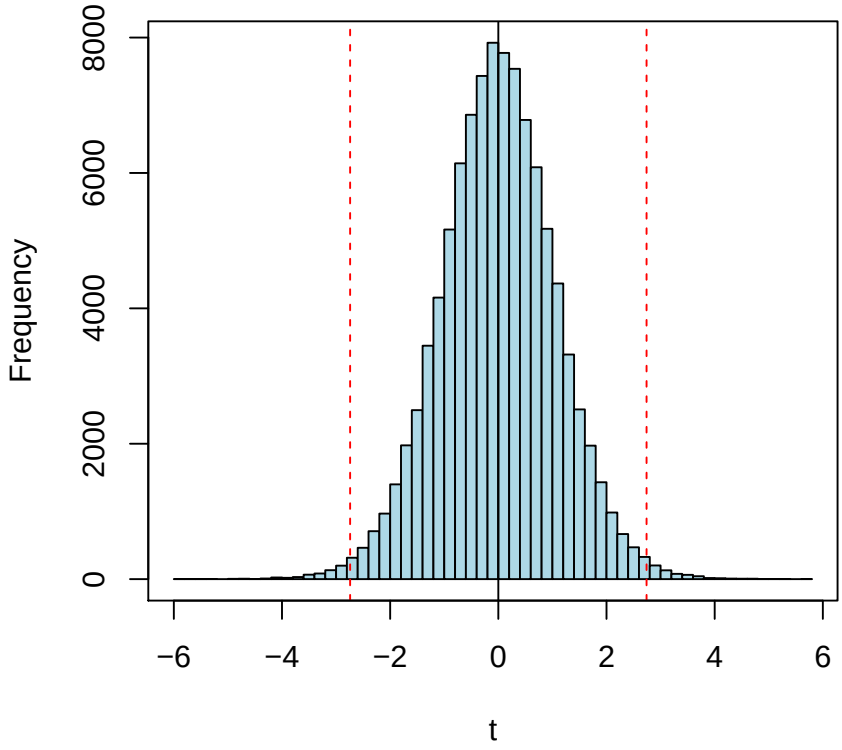

Figure 3.10: Monte Carlo t-distribution of sample size N=10 obtained by 100000 resampling steps from $\sim N(0,1)$; the test value ± 2.74 in the present example is marked red in the plot.

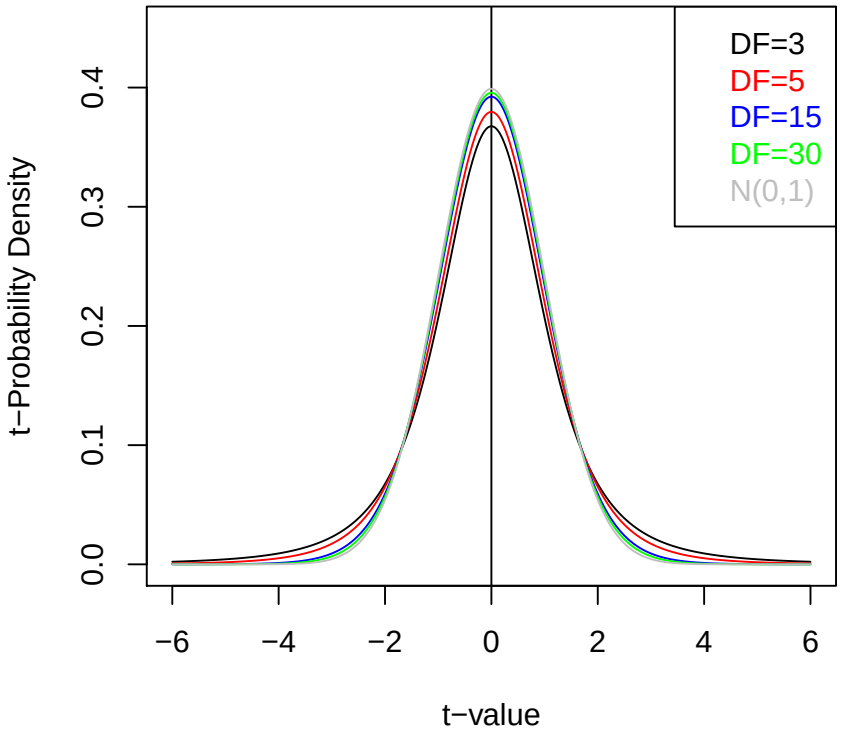

Figure 3.11: Probability density of the t-distribution, pdf(t|DF), for different degrees of freedom (DF)

Of course, there are ready-made functions for performing t-tests in all statistical software packages, in R, e.g., the base function t.test():

```
set.seed(1234)
old    <- rnorm(10,135,5)
new    <-  rnorm(10,140,5)
t.test(old,new, var.equal=T )
```

```
##
##  Two Sample t-test
##
## data:  old and new
## t = -2.7404, df = 18, p-value = 0.01344
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -11.173926  -1.475941
## sample estimates:
## mean of x mean of y
##  133.0842  139.4091
```

The t-tests reports an α-value[8] of 0.01344 in good agreement with the result obtained from Monte Carlo simulation. In addition the parameter **df** in the output reports the degrees of freedom, df=18, of the test. The t-test is based on two "internal" parameters, $\bar{X}_1$ and $\bar{X}_2$ and these two parameters must be deducted from the total sample size N=20, hence df=18[9].
By convention a test result is called significant in statistics when $0.01 \leq \alpha \leq 0.05$ and highly significant when $\alpha < 0.01$.
When a test result is jugded significant given α, H0 is rejected in favour of H1 while silently accepting that in $100 * \alpha\%$ of the cases H0 will be true.
Similar to the error rate under H0, another error rate refering to H1 can be defined: The **False-Negative-Rate** β or the **Type II Error Rate**. Both error rates, α and β, are related with each other as figure 3.12 shows for the two sample t-test.

The False-Negative-Rate β is the likelihood of rejecting the Alternative-Hypothsis H1 when it is true. For instance, by increasing t_{crit}[10] in 3.12 the likelihood α for rejecting H0 in favour of H1 becomes very small at the expense of increasing β. Small αs ($\Leftrightarrow$ large t_{crit}) will render decisions very conservative (rejecting most test results as nonresponders) to the detriment of accepting potential responders thereby inflating the likelihood β. From an innovative point of view, β is much more important than α, because researchers usually want to identify responders

[8] In statistics α-values are often simply called p-values.

[9] Informally speaking two data points were "consumed" for estimating $\bar{X}_1$ and $\bar{X}_2$ and are no longer available for the t-statistics.

[10] The value t_{crit} is the decision boundary of the test. In the two sample t-test we obtain a test statistics t. If $t t_{crit}$ we accept H1 and reject H0. By increasing t_{crit} α becomes small at the expense of making β large (and vice-versa). In a minute we will an example where this property will be exploited for determining the optimal sample size of a t-test

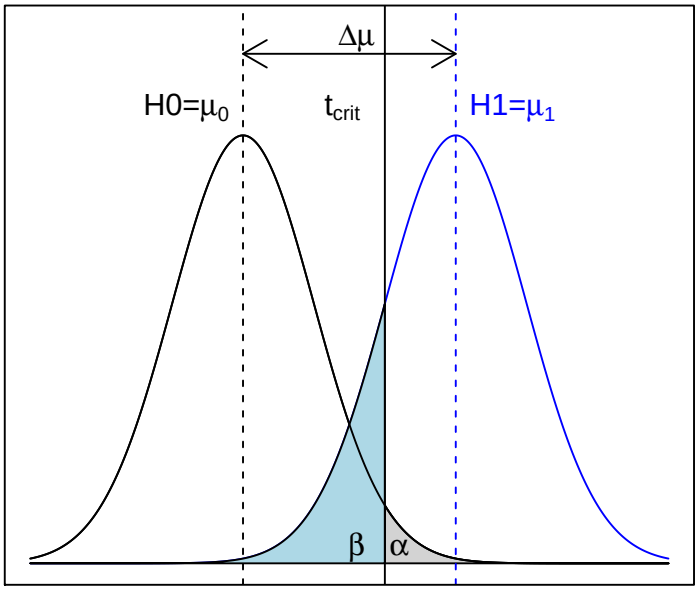

Figure 3.12: t-test and the relationship between α (grey shaded area) and β (blue shaded area)

rather than nonresponders. This naturally leads to the question whether it is possible to control both error rates simultaneously. It is, indeed, possible to control both error rates by appropriately choosing the sample size $N_{control}$ and N_{treat}, as this will render both distributions, H0 and H1, wider or slender.
However, controlling α and β requires knowledge about the effect size $\Delta\mu$ and an estimate of σ, see figure 3.12, a situation rarely given when analyzing historical data. Therefore, virtually all tests on happenstance data rely on α with β unknown.
This situation changes when it comes to designed experiments, and that can be understood by recasting the t-test example from above: A researcher wants to set up an experiment with the current standard being 135 ± 5. She considers an improvement of $\Delta d = +5$ units economically worthwhile and wants to know the number of specimen for each group to ascertain an effect Δd given α and β. The following R-code answers this question (note that the effect size d is specified in standardized unit, that is $d = \frac{\mu_2 - \mu_1}{s} = \frac{140-135}{5} = 1$; also note the definition power=1-β).

```
pwr::pwr.t.test(n = NULL, d = 1,sig.level = 0.05,
power = 0.9,alternative = c("greater"))
```

```
##
##      Two-sample t test power calculation
##
##               n = 17.84712
##               d = 1
##       sig.level = 0.05
##           power = 0.9
##     alternative = greater
##
## NOTE: n is number in *each* group
```

It takes 18 specimen in each group to ascertain an effect of $\Delta = 5$ with a pooled standard error of s=5 given $\alpha = 0.05$ and $\beta = 0.1$. If the test results after having performed the experiments turn out not significant ($\alpha > 0.05$) then the effect Δ is very likely smaller than 5, hence the new process can be discarded as ineffective given $\alpha = 0.05$, $\beta = 0.1$. Of course, higher certainties, that is smaller α and β, will require larger sample sizes.
Depending on the experimental situation, researchers can tune α and β according to their needs: If, for instance, the False-Positive-Rate hurts, which is typically the case in a later project phase with the product being closer to market, then α should be made small. Conversely, when in an early project (screening) phase the False-Negative-Rate hurts. α should be large and β comparatively small so as not to miss potential lead candidates.
Experimental designs simultaneously controlling α and β are called **powered designs** and are the gold standard in the biomedical sciences.

3.4.2 Linear models and their use

In section 2.1 the elements of empirical model building were discussed starting with the base equation (3.5)

$$\Delta y = f(\Delta X) + \epsilon = f(\Delta x_1, \Delta x_2 ... \Delta x_I) + \epsilon;\ \epsilon \sim N(0, \sigma^2) \tag{3.5}$$

Equation (3.5) creates the functional link between observed response Δy and the set of influential factors $\Delta x_1, \Delta x_2 ... \Delta x_I$ augmented with uncertainty ϵ, with the latter, for good mathematical reasons (see chapter 2.1), assumed to come from a normal distribution with unknown variance σ^2, that is $\epsilon \sim N(0, \sigma^2)$. Unfortuantely, the function *f()* is unknown, however, can be locally approximated by low order polynomials of increasing complexity based on Taylor series expansion[11].
Following this rationale, there are three levels of approximation of increasing complexity, namely the following three surrogate models

$$f(\Delta x_1, \Delta x_2 ... \Delta x_I) \approx a_0 + \sum_i a_i \cdot x_i \tag{3.6}$$

$$f(\Delta x_1, \Delta x_2 ... \Delta x_I) \approx a_0 + \sum_i a_i \cdot x_i + \sum_{j>i} a_{ij} \cdot x_i \cdot x_j \tag{3.7}$$

$$f(\Delta x_1, \Delta x_2 ... \Delta x_I) \approx a_0 + \sum_i a_i \cdot x_i + \sum_i a_{ii} \cdot x_i^2 + \sum_{j>i} a_{ij} \cdot x_i \cdot x_j \tag{3.8}$$

Surrogate model (3.6) is a linear model in x_i, whereas equation (3.7) denotes a bilinear model and (3.8) is a full second order model (often abbreviated RSM for **R**esponse **S**urface **M**odel).
In the terminology of statistics, these surrogates are linear parametric models, because they are linear in the regression parameters a_i,a_{ij} although non-linear in the variables $\Delta x_1, \Delta x_2 ... \Delta x_I$ (any model that can be written $y = \sum_j a_j \cdot g_j(x_1 \cdots x_I)$ is linear in the regression parameters a_j). Contrary to linear models, the model $\Delta y = a_0 \cdot e^{a_1 \cdot \Delta x}$ is non-linear in a_1 but can easily be linearized $ln(\Delta y) = ln(a_0) + a_1 \cdot \Delta x$. A non-linear parametric model which cannot be linearized is, e.g., the hyperbolic model $\Delta y = \frac{1}{a_0 + a_1 \cdot \Delta x}$. It is important to keep linear and non-linear models apart as parameter estimation for non-linear models is more complex and not the subject of the present document. The topology of the three surrogate models in two dimensions is depicted in the figures 3.13, 3.14 and 3.15.

Linear parametric models, equation (3.6), figure 3.13, describe planes and hyperplanes in $\mathbb{R}$ with the effects given by the slope parameter a_i. Linear models are

[11] Taylor series expansion requires the unknown function f() to be smooth and continuous

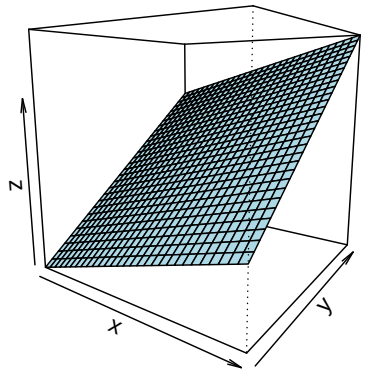

Figure 3.13: Linear parametric model $z = a_0 + a_1 \cdot x + a_2 \cdot y$.

often to simplistic by ignoring factor interactions, which, e.g., play an important role in the chemical sciences. Factor interactions can be accounted for by augmenting the linear model with bilinear model terms. As figure 3.14 shows, the effect of the factor x becomes dependent on y and vice versa. Given the bilinear equation $y = a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_1 \cdot x_2$, there are two potential readings, namely

1. $y = a_0 + (a_1 + a_3 \cdot x_2) \cdot x_1 + a_2 \cdot x_2$
2. $y = a_0 + a_1 \cdot x_1 + (a_2 + a_3 \cdot x_1) \cdot x_2$

In the first reading the slope parameter of x_1 becomes a linear function of x_2, while in the second reading the slope of x_2 is being moderated by x_1 via parameter a_3. In high-dimensional space, $\mathbb{R}$, bilinear models describe twisted planes or twisted hyperplanes.
Finally, response surface models can be used to describe local maxima, minima (see figure 3.15) or sattle points with the latter topology depicted in figure 3.16. Sattle points can make model interpretation and optimization more difficult as they show no clear direction of ascend or descend.

It should be noted that it is usually impossible to infer the topology of the response surface alone from the analytical expression of the parametric model. As will be discussed later, there are mathematical techniques for converting

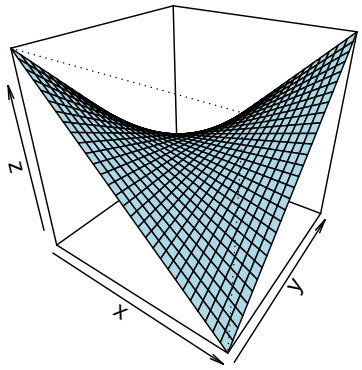

Figure 3.14: Bilinear parametric model $z = a_0 + a_1 \cdot x + a_2 \cdot y + a_3 \cdot x \cdot y$.

RS-models into more "readable" models (canonical analysis). Alternatively, the response surface can be plotted as conditional RSM plots. However, the capability of the human mind for grasping high-dimensional visual information is limited, and mathematical techniques such as the aforementioned canonical analysis or optimization can simplify the analysis of high-dimensional models.

3.4.3 Least Squares in two dimensions

The key ideas of linear model buiding can be beneficially introduced with some low-dimensional (here: two-dimensional) examples based on simulated data. This has the advantage that the "true model" is known and that both, data and model, can be depicted in one plot.
As a first example, figure 3.17 shows a realization of 30 samples of the function $y = 3 + 5 \cdot x + \epsilon; \epsilon \sim N(\mu = 0, \sigma = 1)$

The linear regression line in figure 3.17 is the model estimate $\hat{y}$ from a finite realization of the true, however unknown model $y = 3 + 5 \cdot x$. Model estimates from this realization can be obtained by drawing a straight line through the data with a ruler, so that the line fits the data "somehow well". Of course, this process of "eye-ball regression" is not unique, subject to personal bias and is of only historical relevance. An objective and optimal method for estimating the unknown parameters a_0 and a_1 is the least squares technique invented by the

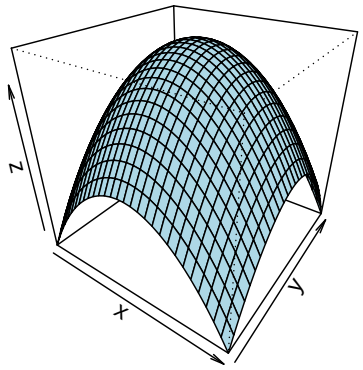

Figure 3.15: RSM model $z = a_0 + a_1 \cdot x + a_2 \cdot y + a_3 \cdot x \cdot y + a_4 \cdot x^2 + a_5 \cdot y^2$.

German mathematician C.F. Gauss (1777-1855).
Least squares works by defining the Least Square objective function SS(a_0,a_1), the sum of squares, equation (3.9)

$$SS(a_0, a_1) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{N} (y_i - [a_0 + a_1 \cdot x_i])^2 \tag{3.9}$$

and minimizing SS(a_0,a_1) over a_0,a_1, that is

$$\min_{a_0, a_1} SS(a_0, a_1)$$

The shape of SS(a_0,a_1) as a function of a_0,a_1 for the bivariate example above is shown in figure 3.18.

Figure 3.18 reveals a convex shaped response surface with a flat minimum at $a_0^* \approx 2.7$ and $a_1^* \approx 5$. These results can be obtained analytically by equating the partial derivates of SS(a_0,a_1) with zero and solving for a_0,a_1, that is

$$\frac{\partial SS(a_0, a_1|y, x)}{\partial a_0} = 0$$

$$\frac{\partial SS(a_0, a_1|y, x)}{\partial a_1} = 0$$

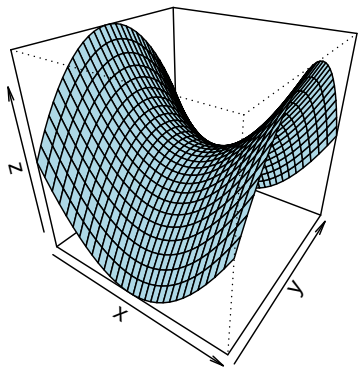

Figure 3.16: RSM model $z = a_0 + a_1 \cdot x + a_2 \cdot y + a_3 \cdot x \cdot y + a_4 \cdot (x^2 - y^2)$ revealing a sattle point.

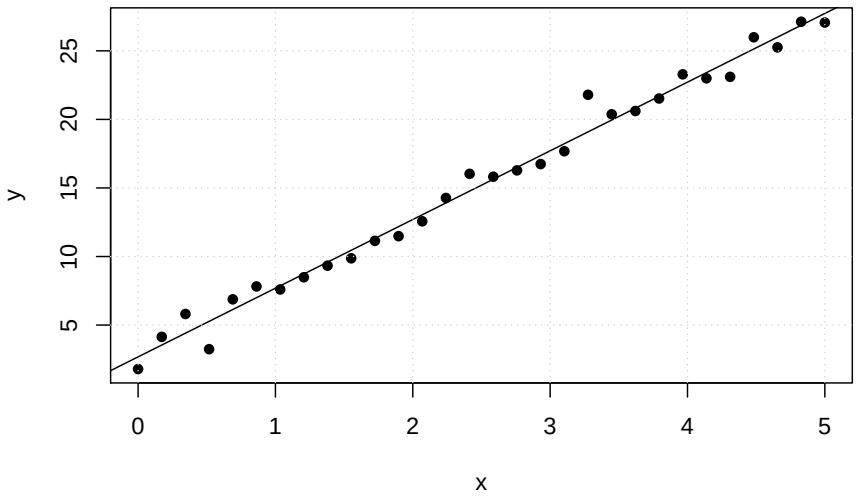

Figure 3.17: Simple scatter plot generated from the underlying function y = 3 + 5*x + N(0,1)

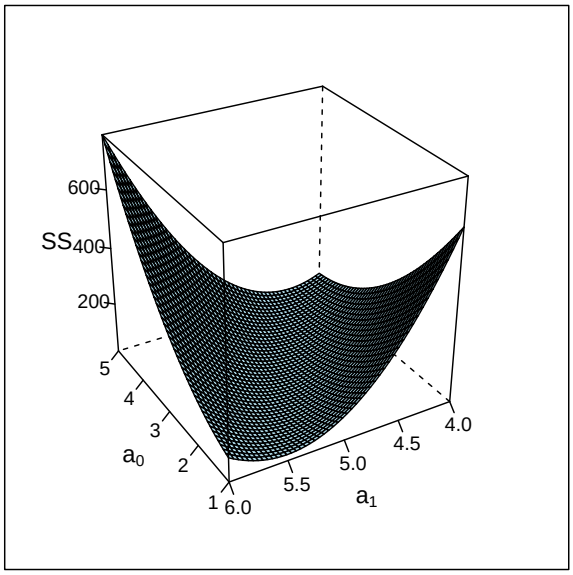

Figure 3.18: $SS(a_0, a_1)$ as a function of a_0, a_1 of sample size N=30 from $y = 3 + 5x + \epsilon; \epsilon \sim N(0, 1)$.

Solving the above normal equations gives the Least Squares solution for the bivariate case, namely

$$a_1^* = \frac{\sum_{i=1}^{N}(y_i - \bar{y}) \cdot (x_i - \bar{x})}{\sum_{i=1}^{N}(x_i - \bar{x})^2} = \frac{\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \bar{y}) \cdot (x_i - \bar{x})}{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} = \frac{Cov(y,x)}{Var(x)}$$

$$a_0^* = \bar{y} - a_1^* \cdot \bar{x}$$

The optimal slope parameter is the ratio of the covariance of x and y divided by the variance of x. The sample covariance, equation (3.10), is a bivariate criterion for measuring the degree of linear association between two random variables.

$$Cov(x,y) = \frac{1}{N-1}\sum_{i=1}^{N}(y_i - \bar{y}) \cdot (x_i - \bar{x}) \tag{3.10}$$

Being scale sensitive, the covariance is as such not very informative, but becomes more meaningful by scaling x and y appropriately, which leads to the well-known ordinary (or Pearson) correlation coefficient, equation (3.11) (the transformation $x_i^* = \frac{x_i - \bar{x}}{s_x}$ has wide use in statistics and is commonly known as standardization and can be done in R with the base function scale()).

$$Cor(x,y) = \frac{1}{N-1}\sum_{i=1}^{N}(\frac{y_i - \bar{y}}{s_y}) \cdot (\frac{x_i - \bar{x}}{s_x}) = \frac{1}{N-1}\sum_{i=1}^{N} y_i^* \cdot x_i^* = \frac{Cov(x,y)}{s_x \cdot s_y} \tag{3.11}$$

The Pearson correlation coefficient $r_{x,y}$ is a scale free value in the range $-1 \leq r_{x,y} \leq +1$ and tells how well the data fits the linear equation $y = \hat{y} + \epsilon = a_0 + a_1 \cdot x + \epsilon$. Values of $r_{x,y} = \pm 1$ indicate that the data fits the model $\hat{y}$ with either positive or negative slope and $\epsilon = 0$. Values close to zero, $|r_{x,y}| \approx 0$, indicate the absence of a linear relationship and suggest the random model $y = \epsilon \; \forall \; x$.

Least Squares problems can be solved conveniently with statistical software, making Ordinary Least Squares (OLS) to the most widely used statistical method in science. In R the lm() function provides all functionalities for solving OLS problems, e.g. with the R example code

```
###### plot SS
set.seed(1234)
x <- seq(0,5,length=30)
y <- 3 + 5*x + rnorm(length(x),0,1)
x.df <- data.frame(y,x) # make a data.frame
res <- lm(y~x,data=x.df) # do linear modeling
summary(res) # show results
```

```
##
## Call:
## lm(formula = y ~ x, data = x.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.0306 -0.5751 -0.2109  0.5522  2.7050
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.6800     0.3273   8.188 6.51e-09 ***
## x             5.0094     0.1124  44.562  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9189 on 28 degrees of freedom
## Multiple R-squared:  0.9861, Adjusted R-squared:  0.9856
## F-statistic:  1986 on 1 and 28 DF,  p-value: < 2.2e-16
```

```
a1 <- cov(x,y)/var(x)                # analytical OLS solution
a0 <- mean(y) - a1*mean(x)
c(a0=a0,a1=a1)                       # show OLS estimators a0,a1
```

```
##       a0       a1
## 2.680009 5.009426
```

```
cor(x.df)                            # shows the Pearson  r.xy
```

```
##           y         x
## y 1.0000000 0.9930235
## x 0.9930235 1.0000000
```

```
sqrt(sum((y - predict(res))^2)/28) # residual standard error
```

```
## [1] 0.9188509
```

```
cor(x.df$y,predict(res))^2           # R2-value #1: cor(y,y.hat)^2;
```

```
## [1] 0.9860958
```

```
ss <- anova(res)
names(ss)                            # what's in ss?
```

```
## [1] "Df"      "Sum Sq"  "Mean Sq" "F value" "Pr(>F)"
```

```
r2 <- ss[,2][1]/sum(ss[,2])
r2                                 # R2-value #2: SS.model/ss.total
```

```
## [1] 0.9860958
```

```
F.test <- ss[,3][1]/ss[,3][2]    # Calculate F-value
F.test
```

```
## [1] 1985.773
```

```
1-pf(F.test,ss[1,1],ss[1,2] )      # Prob(F>F.test)|H0
```

```
## [1] 0
```

The function lm() provides a lot of information and there are many ready-made auxiliary functions associated with objects of class "lm" such as plot.lm(), summary.lm(), predict.lm(),..., type "?lm" in the R console for more information.

3.4.3.1 OLS results and its interpretation

The output from summary(lm()) reports first OLS parameter estimates along with their corresponding standard errors (uncertainty), here $a_0 = 2.6800 \pm 0.3273$ and $a_1 = 5.0094 \pm 0.1124$. When the numerical range $\hat{a}_i \pm 2 \cdot \Delta a_i$ includes zero then, as a rule of thumb, the OLS parameters are noninformative and should be excluded from the model[12]. Underlying the two-sigma rule is a t-test with the statistics $t = \frac{\hat{a}_i}{\Delta a_i}$. The t-values in the example, t_0=8.2 and t_1=44.6, exceed by far the critical 5% quantile of the t-distribution with 28 degrees of freedom[13], here $Prob(T > (t_{critical} = 0.51)|DF = 28) = 0.025$, and the Null-Hypothesis, $a_0 = 0; a_1 = 0$ can safely be rejected in favour of H1, that is $a_0 \neq 0; a_1 \neq 0$. The t-test statistics reported in the OLS output is a two-sided test and reports the probability $Prob(|T| > t_2) = \alpha$, the probability of finding a t-value larger or smaller than $\pm t_2$ by chance under H0.
When the expected effect can be assumed positive (negative) a one-sided test, $Prob(T > t_1) = \alpha$, $(Prob(T < -t_1) = \alpha)$, is more powerful, as $t_1 < t_2$ always holds given DF and α. Figure 3.19 schematically depicts how the one-sided relates to the two-sided test.
In the example, for instance, the likelihood that the test statistics $t_0 = \pm 8.2$ is exceeded or fallen short of under H0 is next to zero, more precisely Prob(|T|>8.2)=6.5*10^{-9}.

Next, the summary() function reports the residual standard deviation which is given by the expression, equation (3.12)

$$s_{residual} = \sqrt{\frac{1}{N-p}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{3.12}$$

[12]In a linear model $\hat{y} = a_0 + \sum_i a_i \cdot x_i$ non-significant parameters a_k will only inflate the variance of the model $\hat{y}$, $Var(\hat{y}) = \sum_k x_k^2 \cdot Var(a_k)$ without significantly contributing any information to the model $\hat{y}$. Therefore, non-significant factors x_k should be excluded from the model

[13]The degrees of freedom, DF, are given by the equation DF=N-p, mit N being the number of observations and p being the number of estimated parameters. In the bivariate example a_0 and a_1 are being estimated from 30 observations, hence DF=28.

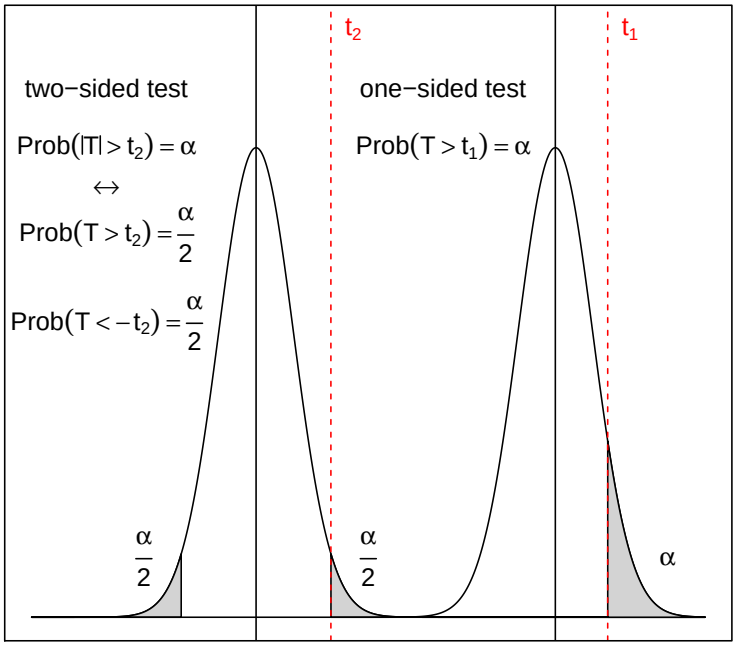

Figure 3.19: Two-sided and one-sided t-test given sample size N and test level α. Note that the decision threshold t_2 for the two-sided test is larger than t_1 rendering the two-sided test less powerful than the one-sided test

with N denoting the number of observations and p the number of model parameters. The contrast (N-p) denotes the degrees of freedom (DF) of the linear model, for the bivariate example DF=(30-2)=28. $s_{residual}$ is an unbiased estimate of the unknown standard error σ from the error term $\epsilon \sim N(0, \sigma = 1)$.
Next in the output follows the well-known R^2 value, which is defined in two different but mathematically equivalent ways.
The first definition is already suggested by the statistic's name according to which R^2 is defined as the squared correlation coefficient between $\hat{y}$ and y, that is $R^2 = Cor(\hat{y}, y)^2$. The second definition needs a little bit more elaboration and relies on the principle of partitioning the sum of squares, a principle that holds for all linear OLS models. According to this principle, the total sum of squares of the response y can be partioned into a model and an error part, see equation (3.13) and figure 3.20 for a graphical description.

$$\begin{aligned} \sum_{i=1}^{N}(y_i - \bar{y})^2 &= \sum_{i=1}^{N}(\hat{y}_i - \bar{y})^2 &&+ \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \\ SS_{total} &= SS_{model} &&+ SS_{error} \\ (N-1) &= (p-1) &&+ (N-p) \end{aligned} \tag{3.13}$$

With both terms SS_{total} and SS_{model} from equation (3.13), the second definition of R^2 is simply

$$R^2 = \frac{SS_{model}}{SS_{total}} = \frac{SS_{total} - SS_{error}}{SS_{total}} = 1 - \frac{SS_{error}}{SS_{total}}$$

R^2 expresses the Goodness-of-Fit as the percentage of variance explained by the model and lies in the domain $0 \leq R^2 \leq 1$. The elements of equation (3.13) give rise to another important statistics reported next in the lm-output, the F-statistics, named after the statistician R.A. Fisher.

$$F = \frac{explained\ variance}{unexplained\ variance} = \frac{\frac{SS_{model}}{DF_{model}}}{\frac{SS_{error}}{DF_{error}}} = \frac{\frac{SS_{model}}{p-1}}{\frac{SS_{error}}{N-p}} \tag{3.14}$$

The F-statistics, equation (3.14), is a signal-to-noise ratio in the range $0 \leq F \leq \infty$ and tests the global Null-hypothis against the global alternative H1,

$$\begin{aligned} H0 &: y = \epsilon \\ H1 &: y = \hat{y} + \epsilon \end{aligned} \tag{3.15}$$

The F-test follows the test logic already discussed: The two-parametric distribution of F is known under the Null-Hypothesis H0, and the F-value obtained

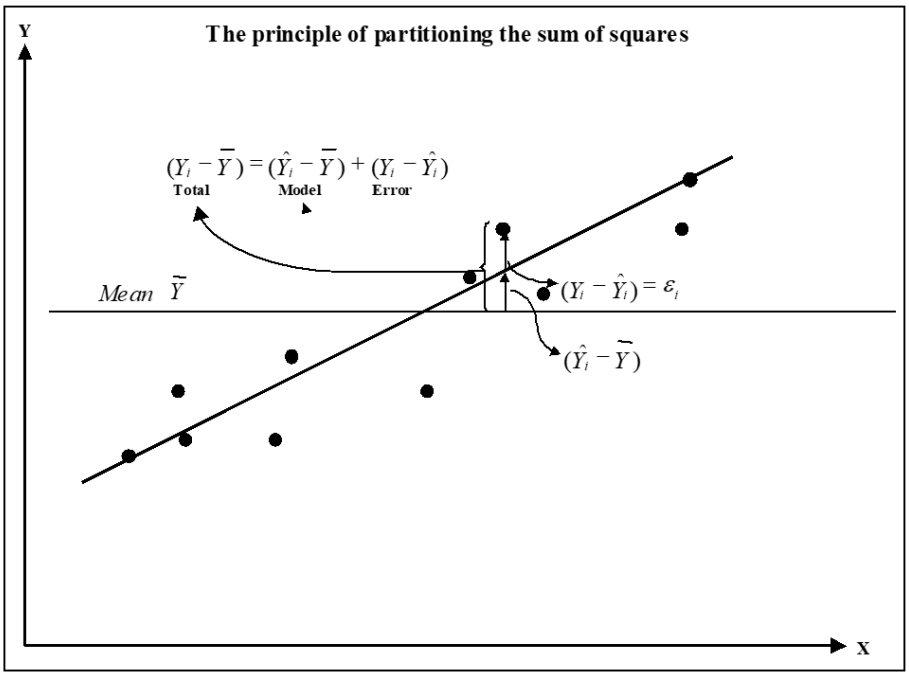

Figure 3.20: Partitioning the total sum of squares into model and error sum of squares

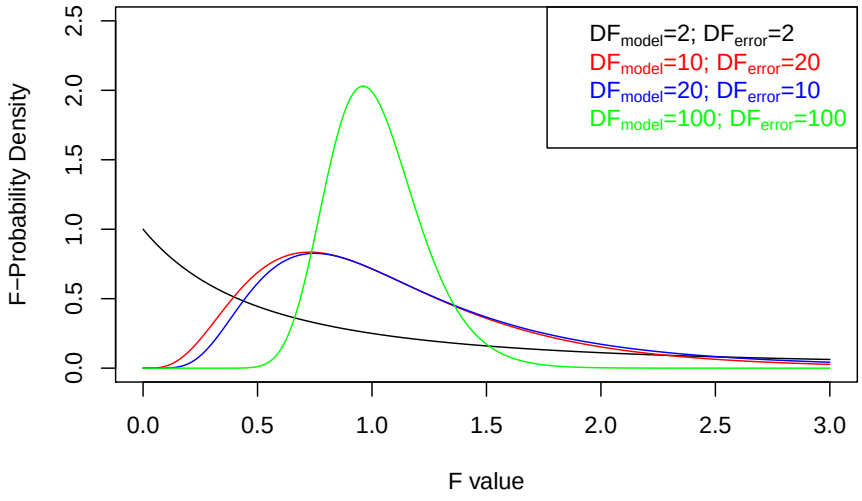

Figure 3.21: Probability density of the F-distribution for different parameters DF_{model}, DF_{error}.

from the data, f_{test}, is compared against the F-distribution under H0, i.e. it is checked whether Prob(F>f_{test}|DF_{model},DF_{error}) < 0.05 holds. If so, reject $y = \epsilon$ in favour of $y = f(X) + \epsilon$. Figure 3.21 shows the probability density of the F-distribution for different values of DF_{model} and DF_{error}.

The F-value in the present example is reported to be F_{test}=1986 and the probability of observing such a large value is virually zero and H1 is accepted, so SS_{model} significantly contributes to the variance of y.
There is a test hierarchy: The F-test checks whether the model $\hat{y}$ is different from 0 (tests, whether the model as a whole is significant), while the t-test checks whether the underlying regression parameters are significant. When the F-test turns out nonsignicant there is not much use checking the significance of the underlying regression parameters. However, there are cases where the F-test is significant with no regression parameter found significant. Such pathological cases can be informative in itself and will be discussed later.

3.4.3.2 Linear OLS for categorial data

In the above OLS example, response y and variable x were both of numerical data type and measured on a continuous, numerical scale. However, in the real world sciences there are a plethora of situations were some, if not all factors X

Table 3.7: t-test data casted as a data frame of dimension 20x2

obsnr	treat	y
1	old	128.96
2	old	136.39
3	old	140.42
4	old	123.27
5	old	137.15
6	old	137.53
7	old	132.13
8	old	132.27
9	old	132.18
10	old	130.55
11	new	137.61
12	new	135.01
13	new	136.12
14	new	140.32
15	new	144.80
16	new	139.45
17	new	137.44
18	new	135.44
19	new	135.81
20	new	152.08

belong to the categorial data type. For instance, a chemist might be interested in testing the performance of three different catalysts with the categorial levels Cat_1, Cat_2 and Cat_3 by setting up and analysing a categorial design.
It is in fact very easy to translate categorial variables into variables more amenable to numerical OLS analysis with the method called "dummy-coding". In order to understand this method, the t-test example from chapter 3.4.1 will be revisited and reformulated as a linear OLS model. The two samples from above are first converted into the data frame depicted in table 3.7 comprising 20 observations and two variables, namely the response "y" and the categorial treatment factor "treat" on the levels "old" and "new".

The idea of "dummy-coding" now consist in converting a factor with N categorial levels into N binary variables with numerical level 0/1. Table 3.8 shows the converted data frame together with the original data. Whenever the factor "treat" is on the level "old" then old=1; else old=0, and the same applies to the level "new".

The data frame, table 3.8, suggest to set up the linear OLS model $y = a_0 \cdot intercept + a_1 \cdot old + a_2 \cdot new$ However, this linear model is overparametrized

Table 3.8: Augmented t-test data with treat being dummy-coded as binary factors old, new

obsnr	treat	y	intercept	new	old
1	old	128.96	1	0	1
2	old	136.39	1	0	1
3	old	140.42	1	0	1
4	old	123.27	1	0	1
5	old	137.15	1	0	1
6	old	137.53	1	0	1
7	old	132.13	1	0	1
8	old	132.27	1	0	1
9	old	132.18	1	0	1
10	old	130.55	1	0	1
11	new	137.61	1	1	0
12	new	135.01	1	1	0
13	new	136.12	1	1	0
14	new	140.32	1	1	0
15	new	144.80	1	1	0
16	new	139.45	1	1	0
17	new	137.44	1	1	0
18	new	135.44	1	1	0
19	new	135.81	1	1	0
20	new	152.08	1	1	0

due to the equality constraint *intercept= old + new*=1. With the latter equality constraint the model can be rewritten

$$y = a_0 \cdot intercept + a_1 \cdot old + a_2 \cdot new;\ intercept = old + new \Rightarrow$$

$$a_0 \cdot intercept + a_1 \cdot (intercept - new) + a_2 \cdot new =$$

$$(a_0 + a_1) \cdot intercept + (a_2 - a_1) \cdot new = a_0^{'} \cdot intercept + \Delta a \cdot new$$

From comparision follows, that by omitting the variable *old* from the regression equation $y = a_0 \cdot intercept + a_1 \cdot old + a_2 \cdot new$ the coefficient a_2 becomes the contrast $a_2 - a_1$. Estimating and testing this contrast will directly answer the hypothesis at stake whether both treatments are the same or different.
The rule just derived holds for all categorial factors: Say, we have a 4-level factor with factor levels A_1, A_2, A_3, A_4: By dropping one level, A_2, all other levels turn into contrasts, here (A_1 - A_2), (A_3 - A_2) and (A_4 - A_2).
The factor levels to be dropped are typically the control (reference) levels of an experiment (in the t-test example the level "old"), and R by default assumes always the first level to be the reference (in the example the lexical order is "new", "old", hence the default reference is "new"). However, this default option can be overwritten by specifying the reference level explicitly with a reference option (see the R-example code).
The R-code for estimating and testing the contrast (new-old) with OLS then becomes

```
set.seed(1234)
rm(list=ls())
set.seed(1234)
old    <- rnorm(10,135,5)
new    <-  rnorm(10,140,5)

x <- data.frame(obsnr=1:20,treat=c(rep("old",10),
                       rep("new",10) ), y=round(c(old,new),2))
levels(x$treat) # wrong order with "new" being the reference
```

```
## [1] "new" "old"
```

```
x$treat <- relevel(x$treat,ref="old") #.. so relevel
levels(x$treat)
```

```
## [1] "old" "new"
```

```
summary(lm(y~treat,data=x))
```

```
##
## Call:
## lm(formula = y ~ treat, data = x)
##
## Residuals:
```

```
##    Min     1Q Median     3Q    Max
## -9.815 -3.365 -0.930  3.495 12.672
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  133.085      1.632  81.530   <2e-16 ***
## treatnew       6.323      2.308   2.739   0.0135 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.162 on 18 degrees of freedom
## Multiple R-squared:  0.2942, Adjusted R-squared:  0.255
## F-statistic: 7.502 on 1 and 18 DF,  p-value: 0.01348
```

The results are numerically the same as those obtained with the t-test, namely *t=2.7404* and *p-value=0.01344*, and the OLS t-value is estimated to be $t = 2.739$, which is significant at the 1.35% test level. The OLS-contrast is reported to be $a_{new} - a_{old} = 6.323 \pm 2.308$. Note, that the label of the OLS estimate *treatnew* denotes implicitly the contrast *treatnew-treatold* by convention.

3.4.3.3 OLS model diagnostic and violation of OLS assumptions

Model diagnostics, the process of comparing model predictions $\hat{y}$ with the actually observed values y, is an important step in the process of empirical model building. This seems trivial in the bivariate case y~f(x), where a simple scatter plot can depict both data $y \sim x$ and model $\hat{y} \sim x$ and thereby highlighting anomalies such as outliers or lack of fit. However, high-dimensional parametric models, $\hat{y} = f(x_1, x_2...x_I|(a_1, a_2...a_K))$ cannot be visualized together with the data underlying the models, and residual diagnostics become an important step for assessing Goodness-of-Fit and model quality.
From the parametric model equation

$$y = f(x_1, x_2...x_I|(a_1, a_2...a_K)) + \epsilon;\ \epsilon \sim N(0, \sigma)$$

follows directly the equation defining the residual

$$\epsilon = y - f(x_1, x_2...x_I|(a_1, a_2...a_K))$$

The difference between observed response y and model expectation $\hat{y}$,

$$r_i = y_i - \hat{y}_i$$

is the residual, which is assumed to come from a normal distribution. The purpose of residual analysis is to check whether this assumption is correct. Basically, there are three distributional assumptions of $N(0, \sigma)$, also known as the Gauss-Markov (GM) conditions, which needs to be checked:

1. The expectation $E()$ (the mean) of the error term should be zero, $E(\epsilon) = 0$, i.e. the residual should be free from outliers biasing the expectation of ϵ.

2. The variance of the error term should be constant, $Var(\epsilon) = \sigma_X^2 = const. \forall X$. The uncertainty should not be a function of the experimental space X, i.e. should be constant over experimental space. This property is called homoscedasticity in statistics.
3. $E(\epsilon_t \cdot \epsilon_{t+i}) = 0; \forall i > 0$, that is the error term ϵ should be serially independent, i.e. uncorrelated in time, and from knowing ϵ_{t_0} it should not be possible to forecast ϵ_{t_1} with $t_1 > t_0$.

Checking whether the GM conditions hold is usually done graphically with three different plots:

1. Plot of the residual $y - \hat{y}$ versus model expectation $\hat{y}$.
2. Plot of the residual $y - \hat{y}$ versus run order.
3. Plot of the autocorrelation function (ACF) $r_i = E(\epsilon_t \cdot \epsilon_{t-i})$

The first plot typically highlights Lack of Fit, outliers and inhomogeneous error variance, while the second and third plots aim at detecting seriell anomalies. Because the residuals are scale sensitive and on the same scale as the response y, they are usually standardized with an estimate of σ and the standardized residual follows $r.studentized = \frac{y-\hat{y}}{\sigma}$. From normal probability theory it is known that 95% (99%) of the residuals can be expected to lie within the range of ± 2 (± 3) standard deviation under H0. Working with standardized residuals simplifies residual diagnostics as potential outliers can be scrutinized in a statistical meaningful contexts.
The autocorrelation function (ACF), γ_τ, of the residuals should be zero within standard error under the assumption of serial independence for all i>0. With R being the residual vector of length N the autocorrelation is defined

$$\gamma_\tau = \sum_{i=1}^{N} \frac{(R_i - \mu) \cdot (R_{i+\tau} - \mu)}{\sigma_R^2}$$

with $\mu = E(R)$ and $\sigma_R^2 = Var(R)$.
The autocorrelation function is the correlation coefficient betweeen τ-lagged residual terms, $cor(R_t, R_{t+\tau})$ and can be expected to be zero under serial independence. Figure 3.22 shows a simulated example violating the first GM condition by two outliers. Outliers are a reality in empirical data and might reflect a lack of experimental control or alternatively model inadequacy, i.e. a non-linearity not approriately described by the empirical model. Depending on the objective of the project[14], outlier should be checked by repeating the outlying experiments.

```
set.seed(1234)
x <- seq(0,5,length=30)
XX <- as.matrix(cbind(1,x))
```

[14] if, say, the objective is maximizing y and there is an outlier positively exceeding all other observations, this experiment should be and must be repeated, as the outlier could have resulted from some advantageous conditions X otherwise going unnoticed. This is of course not nescessary if the project goal is minimization.

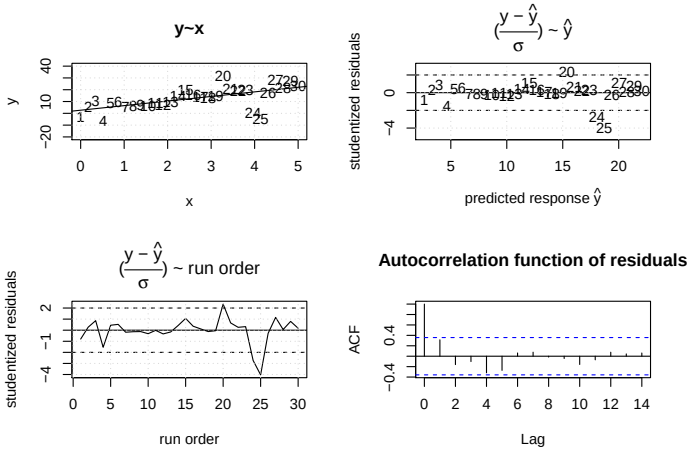

Figure 3.22: Effects of outlying observations on OLS estimation.

```
y <- 3 + 5*x + rnorm(length(x),0,5)
y[x > 3.8 & x < 4.2] <- y[x > 3.8 & x < 4.2] - 25
res <- lm(y~x)

layout(matrix(c(1,2,3,4), 2, 2, byrow = TRUE))
plot(x,y,type="n",pch=16, ylim=c(-20,40), main="y~x")
text(x,y,label=1:length(y))
abline(lm(y~x)$coef)
grid()
plot(predict(res), rstudent(res),xlab=expression( paste("predicted response ", hat(y)) ) ,ylim=c(
     ylab="studentized residuals",type="n",
     main=expression(paste("(",frac(paste("y - ", hat(y)),sigma), ") ~ ", hat(y)   )) )

text(predict(res), rstudent(res),label=1:length(y))
abline(h=c(-2,0,2),lty=c(2,1,2))
grid()
plot(1:length(y), rstudent(res),xlab="run order",ylab="studentized residuals",type="l",
     main=expression(paste("(",frac(paste("y - ", hat(y)),sigma), ") ~ run order"   )))
abline(h=c(-2,0,2),lty=c(2,1,2))
grid()
acf(rstudent(res), main="Autocorrelation function of residuals")
```

```
summary(res)
```

```
##
## Call:
## lm(formula = y ~ x)
```

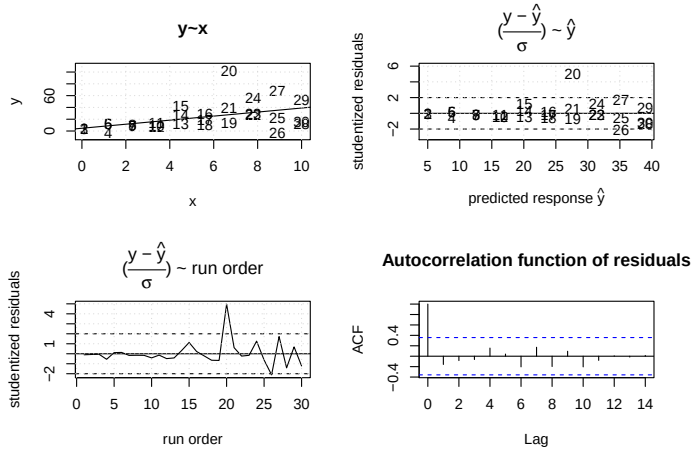

Figure 3.23: Effects of heterogenous variance on OLS

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -23.4949  -1.3215   0.6568   3.2538  16.0924
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.6366     2.6802   0.984 0.333666
## x             3.8858     0.9205   4.221 0.000232 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.524 on 28 degrees of freedom
## Multiple R-squared:  0.3889, Adjusted R-squared:  0.3671
## F-statistic: 17.82 on 1 and 28 DF,  p-value: 0.0002315
```

The outlying observations #24, #25 stick out by four standard deviations (#20 being a borderline case), which is under normality, H0, very unlikely. The slope parameter of the "true" model $y = 3 + 5 * x + N(0, 5)$ is estimated to be $a_1 = 3.9 \pm 0.9$, which is, despite the outliers, in good accordance with the true value. Hence, the slope bias incurred by the two outliers is in this example small. However, the true value $\sigma = 5$ of the error term ϵ is with $s_{residual} = 7.5$ significantly overestimated.

The next simulation, figure 3.23, will demonstrate the effects of inhomogeneous variance on OLS model predictions (the R-code is skipped for brevity and only lm-results are being printed). The underlying model is $y = 3 + 5 \cdot x + N(0, 4 \cdot x)$ and severely heteroscedastic with the variance proportional to x, $\sigma = 4 \cdot x$.

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -39.247  -9.574  -3.125   4.152  73.598
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.670      7.074   0.660   0.5146
## x              3.429      1.188   2.888   0.0074 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 20.55 on 28 degrees of freedom
## Multiple R-squared:  0.2295, Adjusted R-squared:  0.202
## F-statistic: 8.339 on 1 and 28 DF,  p-value: 0.007402
```

While the OLS parameter estimates agree very well with the true values $a_0 = 3$; $a_1 = 5$ within standard error, the scatter plot y~x reveals the model prediction to be biased. The high precision observations in the lower x-range {0,4} are not properly met by the regression line with the regression line being above the data points. This results from the fact that the OLS objective function $\sum_i (y_i - \hat{y}_i)^2$ gives each observation equal weight. However, observations with small standard deviation should get higher weight, while high variance observation should receive a lower weight in the least square functional.

Unbiased OLS estimates can be obtained by using the standard deviation of each replication triple s_i for defining a weighted least squares function, $\sum_i \frac{(y_i - \hat{y}_i)^2}{s_i^2}$.

```
##
## Call:
## lm(formula = y ~ x, weights = wls)
##
## Weighted Residuals:
##     Min      1Q  Median      3Q     Max
## -1.5571 -0.7522  0.2036  0.8851  1.9438
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.2341     0.2727   11.86 1.96e-12 ***
## x             2.6506     0.1229   21.57  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9832 on 28 degrees of freedom
```

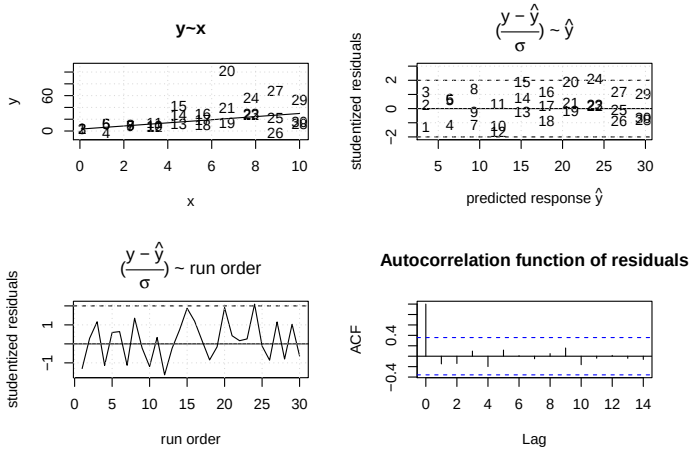

Figure 3.24: WLS results with weights inversely equal to the variance of the observation, $w_i = \frac{1}{s_i^2}$

```
## Multiple R-squared:  0.9432, Adjusted R-squared:  0.9412
## F-statistic: 465.1 on 1 and 28 DF,  p-value: < 2.2e-16
```

The residuals in figure 3.24 agree very well with the pattern expected under normality, and the WLS model now describes the data within error unbiasedly. Heterogeneity of variance is often observed in the real sciences. In chemistry, for instance, the accuracy of instruments are often specified as a percentage of the measured response thereby rendering the variance non-constant over measurement range. WLS is a versatile instrument for dealing with heterogeneous variance. Alternatively, a logarithmic transformation of the the response can sometimes stabilize the variance. The log-transformation will stretch small values and compress large values thereby giving small values higher and large values lower weight.

The third GM condition refers to the assumption of serial independence, that is $E(\epsilon_t \cdot \epsilon_{t+i}) = 0; \forall i > 0$. The most serious violation of this assumption is obtained by artficially replicating (copying) observations. By creating copies, say duplicates, the corresponding error terms ϵ_1, ϵ_2 will be identical thus creating "fake" precision. Rather than two there is in reality only one degree of freedom causing the OLS test statistics to become very liberal (reporting significance by underestimating the standard error when in reality there is no significance). The residual plot, r ~(run order), in figure 3.25 reveals the residuals to be associated with its predecessor in time, here positively autocorrelated[15]. The data was

[15]Informally, positive autocorrelation is indicated by a "lack of scattering" and smoothness in the residual plots. Negative autocorrelation of the error term renders the residual plot very "spiky" and non-smooth. In chemistry, negative autocorrelation can result from carry-over-effects: A positive deviation (error) is followed by a negative deviation and so on. However,

created from the model $y = 3 + 5 \cdot x + \epsilon_t;\ \epsilon_t = a_1 \cdot \epsilon_{t-1} + \epsilon_0;\ \epsilon_0 \sim N(0, \sigma = 5)$
In the real world positive autocorrelation often results from oversampling a process by increasing the sampling rate beyond reasonable limits, in this way creating lots of data, essentially copies, with little information content.
The method of generalized least squares (GLS) can deal with autocorrelated data and yield unbiased estimates and test statistics. However, GLS is beyond the scope of this document and will not be considerred any further here. A simple trick for dealing with autocorrelations is to mean aggregate dependent observations. By the Central Limit Theorem, the resulting mean values will be closer to normality and serial independence. In DoE autocorrelation is seen rarely as the sample size is usually too small for rendering these effects detectable. Serial effects in DoE often indicate systematic effects in time such as loss of catalytic activity or deterioration of analytical instruments.

```
set.seed(1234)
x <- seq(0,5,length=30)
y <- 3 + 5*x + 5*arima.sim(list(ar=c(0.95)),30)
res <- lm(y~x)
layout(matrix(c(1,2,3,4), 2, 2, byrow = TRUE))
plot(x,y,type="n",pch=16, ylim=c(0,50), main="y~x")
text(x,y,label=1:length(y))
abline(lm(y~x)$coef)
grid()
plot(predict(res), rstudent(res),xlab=expression( paste("predicted response ", hat(y)) ) ,ylim=c(
     ylab="studentized residuals",type="n",
     main=expression(paste("(",frac(paste("y - ", hat(y)),sigma), ") ~ ", hat(y)   )) )
text(predict(res), rstudent(res),label=1:length(y))
abline(h=c(-2,0,2),lty=c(2,1,2))
grid()
plot(1:length(y), rstudent(res),xlab="run order",ylab="studentized residuals",type="l",ylim=c(-3,
     main=expression(paste("(",frac(paste("y - ", hat(y)),sigma), ") ~ run order"   )))
abline(h=c(-2,0,2),lty=c(2,1,2))
grid()
acf(rstudent(res), main="Autocorrelation function of residuals")
```

```
summary(res)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.3957  -5.2302  -0.2657   4.9561  14.1255
##
```

positive is far more common then negative autocorrelation.

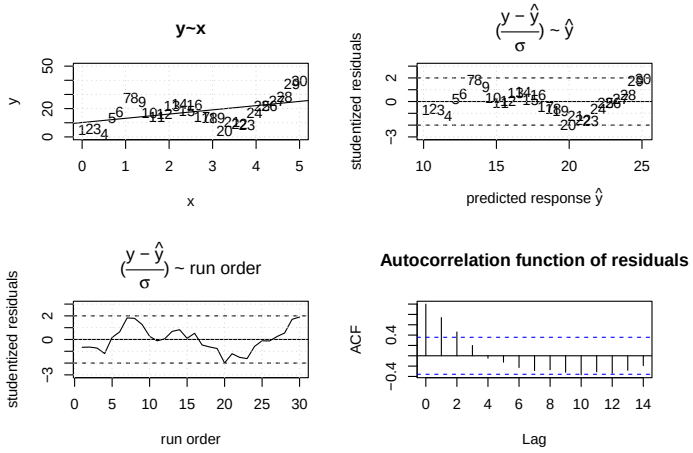

Figure 3.25: OLS results with positively autocorrelated error

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   10.152      2.960   3.430  0.00189 **
## x              2.988      1.017   2.939  0.00653 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.31 on 28 degrees of freedom
## Multiple R-squared:  0.2357, Adjusted R-squared:  0.2084
## F-statistic: 8.637 on 1 and 28 DF,  p-value: 0.006531
```

The examples considered so far all refer to anomalies in the error term ϵ of $f(x_1, x_2...x_I|a_1, a_2...a_K) + \epsilon$ which show up as non-normality in the residuals. However, anomalies in the residuals can also arise from misspecifying the functional part *f()* in $f(x_1, x_2...x_I|a_1, a_2...a_K) + \epsilon$.
The example data depicted in figure 3.26 is a realization of the model $y = 3 + 5 \cdot x + 3 \cdot x^2 + \epsilon;\ \epsilon \sim N(0, 5)$ analyzed with the parametric OLS model $y = a_0 + a_1 \cdot x$.

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -9.218 -5.559 -2.161  6.512 14.259
##
## Coefficients:
```

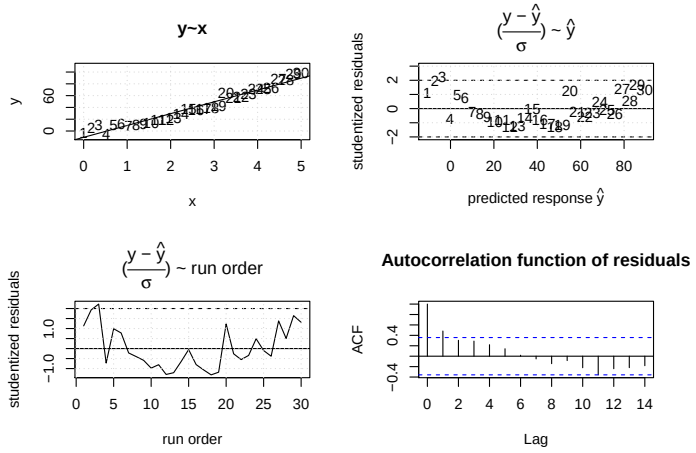

Figure 3.26: Linear, $\hat{y} = a_0 + a_1 \cdot x + \epsilon$, OLS results from non-linear data, $y = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \epsilon$.

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -10.6689     2.5772   -4.14 0.000289 ***
## x            20.0471     0.8852   22.65  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.235 on 28 degrees of freedom
## Multiple R-squared:  0.9482, Adjusted R-squared:  0.9464
## F-statistic: 512.9 on 1 and 28 DF,  p-value: < 2.2e-16
```

The residuals in figure 3.26 indicate that the model (the horizontal line in the residual plots at r_i=0) underestimates the data at the lower and upper tail, while overestimating the response y in the middle range. This is an indication of some non-linearity not accounted for by the linear model.
Remodeling the data with a quadratic RSM model $y = a_0 + a_1 \cdot x + a_2 \cdot x^2$ gives the result depicted in figure 3.27

```
##
## Call:
## lm(formula = y ~ x + I(x^2))
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -9.691 -3.045 -1.324  3.308 13.084
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

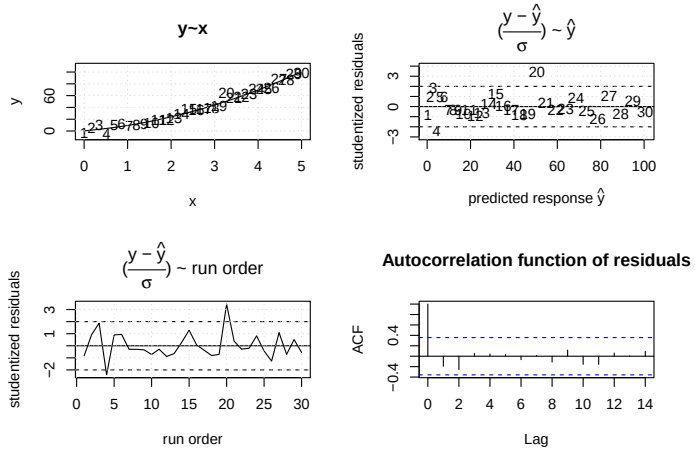

Figure 3.27: RSM results from non-linear data, $y = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \epsilon$.

```
## (Intercept)   0.3093     2.3829   0.130  0.89770
## x             6.4028     2.2062   2.902  0.00729 **
## I(x^2)        2.7289     0.4264   6.400 7.42e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.644 on 27 degrees of freedom
## Multiple R-squared:  0.9794, Adjusted R-squared:  0.9779
## F-statistic:   643 on 2 and 27 DF,  p-value: < 2.2e-16
```

Figure 3.27 does not reveal anomalies anymore and shows at the same time how the residuals should ideally look like, a homogeneous band with no systematic pattern in the range of $\pm 3 \cdot \sigma$. However, figure 3.27 should also be taken as a warning not to adhere too strictly to the $\pm 3\sigma$ rule as the "outlier", #20, is a chance result. So, unlikely things nevertheless happen.

3.4.3.4 OLS in many dimensions

In this section the low dimensional examples considered so far will be extended to higher dimensions. Given a data matrix of influential factors $\boldsymbol{X}$ with dimension Nxp (N times p) and a response vector of length N, the problem can be concisely expressed by equation (3.16)

$$\mathbf{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}; \epsilon \sim N(0, \sigma^2) \tag{3.16}$$

with the individual elements $\boldsymbol{X}$, $\boldsymbol{y}$ and $\boldsymbol{\beta}$ given in (3.17)

$$\boldsymbol{X} = \begin{bmatrix} 1; X_{11}; X_{12}; \cdots; X_{1p} \\ 1; X_{21}; X_{22}; \cdots; X_{2p} \\ \vdots \ddots \vdots \\ 1; X_{n1}; X_{n2}; \cdots; X_{np} \end{bmatrix}, \qquad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{3.17}$$

Note that the data matrix has been augmented by a column of 1's corresponding to the intercept parameter β_0. Also note, that the data matrix $\boldsymbol{X}$ also includes higher order model terms, here bilinear and/or quadratic terms, if these terms are specified in the formula expression of lm(). In R the model matrix is generated from the variables in a data frame and a symbolic formula (see "?formula") by the function model.matrix(). When running lm() this function is implicitly called prior to parameter estimation.
The OLS solution $\hat{\beta}$ can be derived by minimizing the quadratic norm, the function $S(\beta)$ over β as shown in equation (3.18), something that can be done analytically by equating the partial derivatives with zero, $\frac{\partial S}{\partial \beta_i} = 0$ and solving for β.

$$S(\boldsymbol{\beta}) = \left\| \mathbf{y} - \mathbf{X}\boldsymbol{\beta} \right\|^2 \hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} S(\boldsymbol{\beta}) \tag{3.18}$$

Skipping the mathematical details here, the OLS solution is finally found by the equation (3.19)

$$\boxed{\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}} \tag{3.19}$$

The computational burden of equation (3.19) is in essence the inversion of the covariance matrix $\boldsymbol{X^T \cdot X}$, something that can be done in real time with modern computers. The OLS solution, equation (3.19), is a function of the random variable $\mathbf{y}$ and thereby becomes itself a random variable with expectation $\hat{\beta}$ and variance, $Var(\hat{\beta}) > 0$.
It can be shown[16] that $Var(\hat{\beta})$ is given by equation (3.20) which is the fundamental equation of experimental design.

$$\boxed{\mathrm{Var}(\hat{\beta}) = \sigma^2(\boldsymbol{X^T X})^{-1}} \tag{3.20}$$

In equation (3.20) the parameter σ^2 affecting the error term is controlled by nature and basically not under the control of the experimenter (or only remotely under her control by keeping the ambient conditions as constant as possible, $\rightarrow \Delta Z = 0$). However, the design matrix $\boldsymbol{X}$ affecting the variance of $\hat{\beta}$ is under the control of the researcher.

[16] see, for instance, (A. Sen, M. Srivastava 1990)

Experimental design then becomes, very broadly stated, the process of designing $\boldsymbol{X}$ in such way that $Var(\hat{\beta})$, equation (3.20), attains a minimum within the experimental uncertainty, σ^2, set by nature.

Chapter 4

Design of Experiments (DoE)

This chapter introduces experimental design as an essential part of OLS modeling, Many important design classes will be discussed together with the associated OLS models for analysing these designs. It will be outlined that collinearity, due to a poorly designed matrix **X**, is the driving force for the methology of experimental design (**D**esign **o**f **E**xperiments, DoE).

4.1 OLS and collinearity

In chapter 3.4.3.4 DoE was loosely defined as the process of minimizing the objective function $Var(\hat{\beta})$ over $\boldsymbol{X}$, formally

$$\min_{\boldsymbol{X}} \sigma^2 (\boldsymbol{X^T X})^{-1}$$

The expression $Var(\hat{\beta})$ is a matrix and must first be converted into a scalar function $g(X) = f(Var(\hat{\beta}))$ which can then be optimized as a function of the design $\boldsymbol{X}$. The optimization problem thus becomes

$$\min_{\boldsymbol{X}} \sigma^2 f\Big((\boldsymbol{X^T X})^{-1}\Big)$$

with the scalar functional f() belonging to the class of the so called alphabetic functions seeking different kinds of optimality (A-,C-,D-,E-,G-optimality to name but a few)[1].
Alphabetic optimality aims at minimizing the uncertainty of the OLS estimates

[1]It should be noted that from a practical point of view the different optimization criteria are all similar and will give "good" designs. From a theoretical point of view they differ by giving different weight to certain statistical properties of the design, such as minimizing

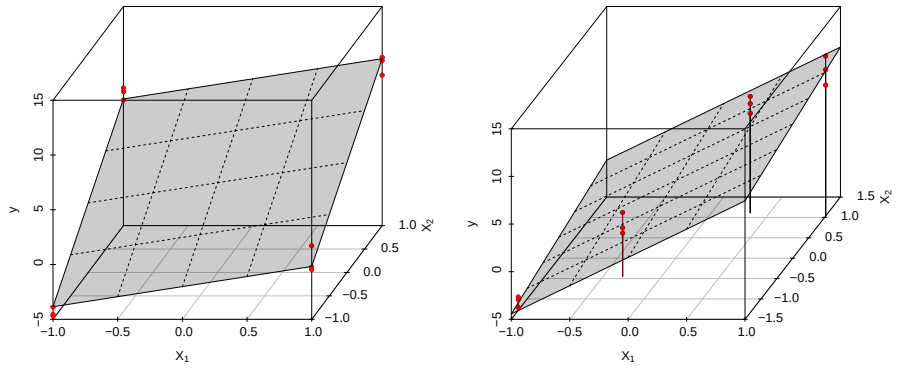

Figure 4.1: Linear OLS model $y = a_0 + a_1 \cdot x_1 + a_2 \cdot x_2$ based on orthogonal $design_1$ (left panel) and collinear $design_2$ (right panel)

Table 4.1: Comparision of OLS parameter estimates obtained from orthogonal $design_1$ and collinear $design_2$.

	design1	design2
(Intercept)	3.194 +/- 0.282	3.154 +/- 0.312
x1	1.84 +/- 0.282	5.933 +/- 1.164
x2	5.207 +/- 0.282	1.105 +/- 0.856

$\hat{\beta}$ over the design matrix $\mathbf{X}$ or to render the design $\mathbf{X}$ maximal robust and informative given the experimental uncertainty σ^2.
Now, in order to understand the mechanism of how different designs $\mathbf{X}$ can affect the outcome of OLS, let us consider two different designs for identifying the "true" model $y = 3 + 2 \cdot x_1 + 5 \cdot x_2 + \epsilon; \epsilon \sim N(0, 1)$

1. An orthogonal 2x2 full factorial design as a triple replicate with Pearson-r, $r(x_1,x_2) = 0$
2. A collineare design with $x_1 = x_2 + N(0,0.5)$ with Pearson-r, $r(x_1,x_2) = +0.94$

Both designs along with the estimated regression planes are depicted in figure 4.1.

the average error of the estimates (A) or minimizing the maximum prediction error of the model (G-optimality). A good compromise over these properties is given by the D-criterion which seeks to minimize the determinant of $(\boldsymbol{X^T X})^{-1}$. Geometrically, this is equivalent to minimizing the confidence (hyper-)ellipsoid of the parameter estimates.

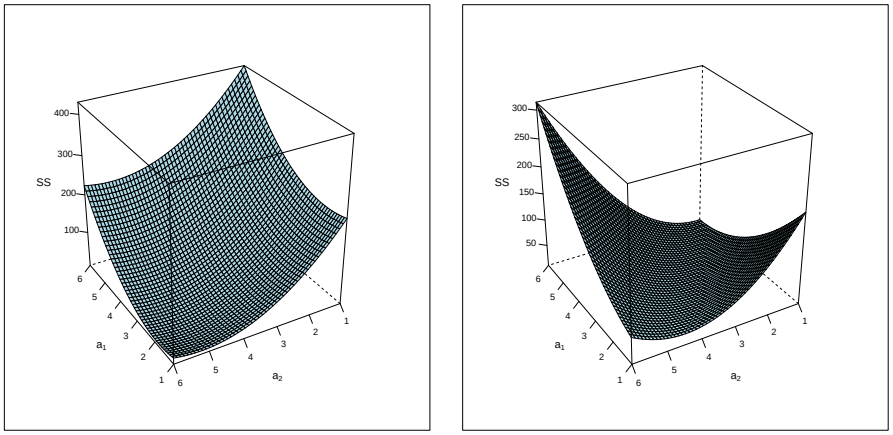

Figure 4.2: $SS = f(a_1, a_2|a_0)$ given $a_{0,design1} = 3.194$ (left panel) and $a_{0,design2} = 3.154$ (right panel)

According to figure 4.1 and table 4.1, both estimates differ significantly although being based on the same "true" model $y = 3 + 2 \cdot x_1 + 5 \cdot x_2 + \epsilon; \epsilon \sim N(0, 1)$. The estimates from the collinear design, *design₂*, are clearly biased, whereas the estimates from the orthogonal design, *design₁*, agree well with the true parameters within standard error. In addition, the standard errors from *design₁* are small and equal for all parameters, but large and unbalanced for the collinear design. This is a common observation and results from the data distribution of the designs. *Design₁* reveals a spherical, symmetrical data distribution in x_1,x_2 rendering the empirical evidence about $\mathbf{y}$ in all direction of experimental spaces approximately equal. In *design₂* the data is distributed close to the x_1, x_2-diagonal, that is elliptical in two-dimensional and hyperelliptical in high-dimensional space thus leaving the direction perpendicular to the densly populated diagonal unpopulated. The effects of ill-balanced data on parameter estimation can be understood by plotting the Sum of Squares SS(a_0,a_1,a_2) for both models, figure 4.2, as a function of a_1 and a_2 given a_0 at the OLS optimum taken from table 4.1.

While SS(a_1,a_2) of *design₁* reveals a distinct minimum close to the true values at a_1=2 and a_2=5, the SS(a_1,a_2) topology of *design₂* turns out degenerated in one direction but convex and non-degenerated perpendicular to that direction. The degenerated direction in parameter space, figure 4.2, corresponds to directions unpopulated by data in figure 4.1 with the consequence of making the location of the OLS minimum uncertain. More generally, a flat topology in SS(a_1,a_2) always indicates areas of uncertainty with large standard error, while sharp and convex minima in SS give rise to higher certainty about the location of the minimum thus shrinking the standard error.
While *design₂* is ill-conditioned but still estimable, the limiting case shown in fig-

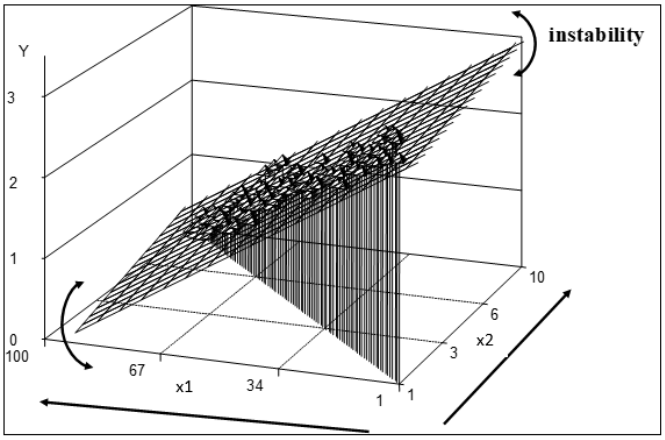

Figure 4.3: Degenerated design for estimating the OLS model $\hat{y} = a_0 + a_1 \cdot x_1 + a_2 \cdot x_2$.

ure 4.3 is singular and non-estimable. Here, x_1 is an exactly linear function of x_2, formally $x_1=b_0 + b_1x_2$ with the consequence of rendering the regression equation $y=a_0 + a_1x_1 + a_2x_2$ non-estimable. Parameter estimation with a singular design shown in figure 4.3 amounts to the attempt of placing a regression plane onto a one-dimensional data edge which is clearly impossible. Formally, the regression equation $y=a_0 + a_1x_1 + a_2x_2$ collapses with the contraint $x_1=b_0 + b_1x_2$ into $y=a_0 + a_1(b_0 + b_1x_2) + a_2x_2$, and the two-dimensional problem degenerates into a one-dimensional regression problem $y=c_0 + c_1x_2$ with $c_0=a_0+a_1b_0$ and $c_1=a_2+a_1b_1$.

Trying to estimate a linear OLS problem $y=a_0 + a_1x_1 + a_2x_2$ is equivalent to asking for the independent effects of x_1 and x_2. When, due to collinearity, no information about the independent effects is available these attempts must fail. Confounding of process factors x_i with x_k is afflicted with the fundamental problem that in the singular limiting case any linear combination $a_1 \cdot x_1 + a_2 \cdot x_2;\ a_1, a_2 \in \mathbb{R}$ is a valid solution of the OLS problem. There are, in other words, an infinite number of solutions for the OLS problem and the collinear data cannot tell one solution from the other. The problem of collinearity becomes worse with increasing number of dimensions and when higher order model terms, such as interactions x_ix_j or square terms x_i^2, are involved. For instance, it is easy to see that an interaction term x_ix_j will be obscured by collinearity and show up as a quadratic term x_i^2 due to the confounding constraint $x_j \approx a_i \cdot x_i$. For these reasons it is virtually impossible to estimate interactions from historical data as there will always be linear dependencies in the data invalidating such attemps. Confounding can also be the source of paradoxes leading to contradictions between t- and F-test. Consider therefore the following ill-conditioned case with the factors x_1,x_2 being highly counfounded ($r_{x1,x2} = +0.987$).

```
rm(list=ls())
set.seed(12556)
x <- data.frame(x1=seq(-1,1,length=4))
x$x2 <- x$x1 + rnorm(nrow(x),0,0.1)
x <- rbind(x,x,x)
x$y <- 3 + 2*x$x1 + 5*x$x2 + rnorm(nrow(x),0,1)
summary(lm(y~x1+x2,data=x))
```

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2375 -0.6452  0.2270  0.6547  1.1735
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.0239     0.3479   8.691 1.14e-05 ***
## x1            0.7088     2.2483   0.315   0.7598
## x2            6.2079     2.4621   2.521   0.0327 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9281 on 9 degrees of freedom
## Multiple R-squared:  0.9721, Adjusted R-squared:  0.9659
## F-statistic:   157 on 2 and 9 DF,  p-value: 1.006e-07
```

In this pathological example the F-test is found highly significant with an F-value of F=157 and DF_1=2, DF_2=9, however hardly any parameter a_1, a_2 can be judged significant based on the t-test. On the F-level the model is significant, however on the parameter level it is not significant. This ostensible contradiction can be resolved by remembering that OLS aims at estimating independent effects. However, when there is no information about the independent effects, then the estimates become arbitrary and there are arbitrarily many parameter combinations explaining the response with small error (note R^2=0.97).
The problems just discussed are solid reasons motivating DoE, as DoE will avoid confounding in the design phase and thereby render the parameter estimates $\hat{\beta}$ maximal informative[2]. Confounding problems can be identified by analysing the correlation structure of the design matrix $\mathbf{X}$. For instance, if two or more columns in X are highly correlated, then OLS estimates will likely be unstable, and the standard error of the collinear parameters will become inflated.
Analysing the confounding structure of large design matrices $\mathbf{X}$ with bivariate

[2]A scientific experiment can be conceived as a question asked to nature. The answer and the precision of her answer is not independent from the question being asked. DoE, to borrow from this simile, amounts to asking most precise questions.

correlation analyses can be difficult and tedious, especially when there are multicollinear dependencies[3] in the design. The condition number κ is a quality score of the design based on the eigenvalue decomposition of the covariance matrix $\boldsymbol{X^T \cdot X}$ and works even in the presence of multicollinearities. The condition number κ is defined as the ratio of the largest divided by the smallest eigenvalues of $\boldsymbol{X^T \cdot X}$, $\kappa = \frac{\lambda_{max}}{\lambda_{min}}; \lambda_{min} > 0$, i.e. the ratio of the largest divided by the smallest spread of data in k-dimensional space. Informally, it is a measure of the hyperellipticity or conversely the design's deviation from hypersphericity (see figure 4.4 for a two-dimensional example). By requiring $\lambda_{min} > 0$, κ measures closeness to singularity and not singularity, and that property can make κ misleading when applied to model matrices with n.col>n.row (see the section on screening designs below for an example). κ is a positive number in the range $\{1, \infty\}$. $\kappa = 1$ indicates a perfectly orthogonal, hypersherical design, while $\kappa = \infty$ signals a degenerated and almost singular design.
The following R-code shows some examples of how κ can be used for checking design and model quality. Note that κ should always be calculated based on scaled data, and the R-function model.matrix(...,data=data.frame(scale(...))) is the preferred method for doing so (see the R-code below). Figure 4.4 illustrates κ in two dimensions.

```
set.seed(123)
x <- expand.grid(x1=seq(-1,1,length=5),
    x2=seq(-1,1,length=5)) # orthogonal 5x5 full-factorial
kappa(model.matrix(~x1 + x2 ,
    data=data.frame(scale(x)))) # perfect kappa=1
```

```
## [1] 1.01384
```

```
x$x3 <- x$x1 + x$x2 + rnorm(nrow(x), 0,0.1)
kappa(model.matrix(~x1 + x2 + x3,
    data=data.frame(scale(x)))) # collinear kappa=38
```

```
## [1] 37.7213
```

```
x$y = 5 + 3*x$x1 + 5*x$x2 - 10*x$x3 +
  rnorm(nrow(x), 0,1)     # make response

summary(res <- lm(y~x1+x2+x3, data=x) )
```

```
##
## Call:
## lm(formula = y ~ x1 + x2 + x3, data = x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

[3] When one column in the design matrix, say x_i, can be approximately expressed as a linear function of other columns x_j, $x_i = \sum_{j,j \neq i} a_j \cdot x_j + \epsilon$.

```
## -1.6026 -0.6389 -0.1808  0.6001  1.4534
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.11465    0.16693  30.640  < 2e-16 ***
## x1          -0.08703    1.92135  -0.045  0.96430
## x2           1.25674    1.94548   0.646  0.52529
## x3          -6.24715    1.99908  -3.125  0.00512 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.834 on 21 degrees of freedom
## Multiple R-squared:  0.9809, Adjusted R-squared:  0.9781
## F-statistic:   359 on 3 and 21 DF,  p-value: < 2.2e-16
```

```
kappa(res)    # kappa = 46
```

```
## [1] 45.80286
```

```
kappa(model.matrix(~x1 + x2 + x3,
  data=data.frame(scale(x))) ) # kappa = 38;
```

```
## [1] 37.7213
```

```
round(cor(x),2)
```

```
##       x1    x2    x3     y
## x1  1.00  0.00  0.70 -0.77
## x2  0.00  1.00  0.71 -0.61
## x3  0.70  0.71  1.00 -0.98
## y  -0.77 -0.61 -0.98  1.00
```

4.2 Mixed level full factorial designs

Mixed level full factorial designs are not an overwhelmingly important design class in chemistry, but they are nevertheless of value as the basis of other important design classes such as the 2^2-full factorial designs or as candidate designs for optimal designs, an important design class discussed later.
In the following examples mixed-level factorial designs will be used to introduce the ANOVA[4] and ANCOVA[5] model class, models with either all X-variables entirely categorial or of mixed categorial-numerical data type.
Then, what is a mixed-level full factorial design?
Given factor x_1, $x_2 \ldots$ x_I on l_1, l_2 $\ldots$ l_I factor levels, a mixed factorial design is created by combining all factor levels with $N = l_1 \cdot l_2 \cdot \ldots l_I = \prod_{i=1}^{I} l_i$ combinations in total. This defines a grid in I-dimensional space $\mathbb{R}^{\mathbb{I}}$, hence these designs are

[4] **AN**alysis **O**f **VA**riance
[5] **AN**alysis of **COVA**riance

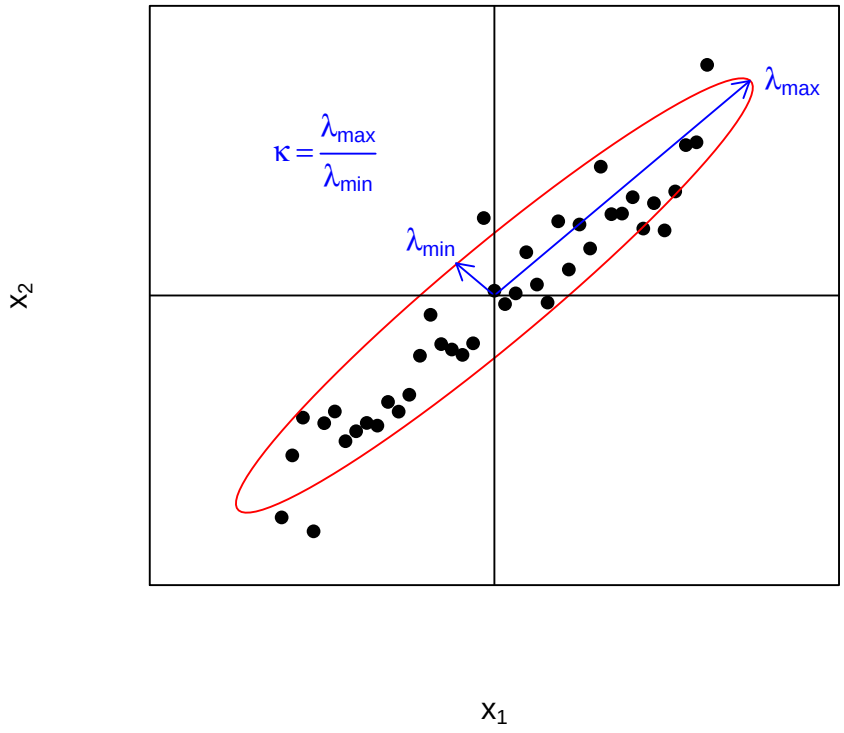

Figure 4.4: The condition number κ as a measure of the (hyper)ellipticity of the data illustrated in two dimensions x_1 and x_2

alternatively called grid designs. In R such designs can be easily created with base function expand.grid().
As a first example of a mixed-level factorial design consider the following ANOVA problem: A researcher wants to study the performance (here labelled **activity**) of three different catalysts, cat={A,B,C} and two co-catalyst, co.cat={co1,co2}. The corresponding 3x2 grid is replicated three times making a total of 18 runs. With the indicator function I(TRUE)=1, I(FALSE)=0, the activity is assumed to relate to cat and co.cat by the assumed example model

$$activity = 1 \cdot I(cat = A) + 2 \cdot I(cat = B) + 3 \cdot I(cat = C) - 1 \cdot I(co.cat = co1)$$

$$+1 \cdot I(co.cat = co2) + 5 \cdot I(cat = A \ \& \ co.cat = co1) + \epsilon; \epsilon \sim N(0, \sigma = 1)$$

The R-code creating design, model, and model diagnostics is as follows

```
rm(list=ls())
set.seed(1234)
x <- expand.grid(cat=c("A","B","C"),
                 co.cat=c("co1","co2")) # factorial 3x2 design
x <- rbind(x,x,x) # make a triple replicate
x <- x[order(runif(nrow(x))),]  # randomize
x$activity <- round( 1*(x$cat=="A") + 2*(x$cat=="B") +
3*(x$cat=="C") - # marginal cat effects
1*(x$co.cat=="co1") +1*(x$co.cat=="co2")   +
# marginal co.cat effects
5*( (x$cat=="A")&(x$co.cat=="co1")) + # interaction
rnorm(nrow(x),0,1) ,4 ) # error term; sigma=1
sapply(x,class)
```

```
##       cat    co.cat  activity
##  "factor"  "factor" "numeric"
```

```
# check x
rownames(x) <- NULL
knitr::kable(
  x, booktabs = TRUE,row.names=T,align="c",
  caption = 'A 3x2 mixed level factorial design as basis of
             a two-way ANOVA model with interaction'
)
```

```
summary(res  <- lm(activity~(cat+co.cat)^2 , data=x))  # lm model
```

```
##
## Call:
## lm(formula = activity ~ (cat + co.cat)^2, data = x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6625 -0.2989 -0.0466  0.4401  1.7517
```

Table 4.2: A 3x2 mixed level factorial design as basis of a two-way ANOVA model with interaction

	cat	co.cat	activity
1	A	co1	4.1100
2	A	co1	4.5228
3	B	co1	0.0016
4	C	co2	3.2237
5	A	co1	5.0645
6	B	co2	3.9595
7	C	co1	1.8897
8	A	co2	1.4890
9	C	co2	3.0888
10	C	co1	1.1628
11	B	co1	3.4158
12	A	co2	2.1341
13	C	co2	3.5093
14	C	co1	1.5595
15	B	co2	3.4596
16	A	co2	1.3063
17	B	co2	1.5518
18	B	co1	1.5748

```
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)      4.5658     0.5339   8.551 1.89e-06 ***
## catB            -2.9017     0.7551  -3.843 0.002340 **
## catC            -3.0284     0.7551  -4.011 0.001728 **
## co.catco2       -2.9226     0.7551  -3.871 0.002226 **
## catB:co.catco2   4.2489     1.0679   3.979 0.001830 **
## catC:co.catco2   4.6592     1.0679   4.363 0.000923 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9248 on 12 degrees of freedom
## Multiple R-squared:  0.6836, Adjusted R-squared:  0.5517
## F-statistic: 5.185 on 5 and 12 DF,  p-value: 0.009165
```

```
anova(res) # ANOVA analysis of the model terms
```

```
## Analysis of Variance Table
##
## Response: activity
##            Df  Sum Sq Mean Sq F value   Pr(>F)
## cat         2  2.1973  1.0986  1.2846 0.312229
## co.cat      1  0.0098  0.0098  0.0115 0.916403
## cat:co.cat  2 19.9649  9.9824 11.6721 0.001532 **
## Residuals  12 10.2629  0.8552
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# interpretation of OLS
ols.est <- c(
  intercept =  round(mean(x$activity[x$cat=="A" & x$co.cat=="co1"] ), 4),
  catB      =  round(mean(x$activity[x$cat=="B" & x$co.cat=="co1"] )-
               mean(x$activity[x$cat=="A" & x$co.cat=="co1"] ), 4),
  catC      =  round(mean(x$activity[x$cat=="C" & x$co.cat=="co1"] )-
                       mean(x$activity[x$cat=="A" & x$co.cat=="co1"] ),4),
  co.catco2=  round(mean(x$activity[x$cat=="A" & x$co.cat=="co2"] )  -
               mean(x$activity[x$cat=="A" & x$co.cat=="co1"] ), 4),
  catB_co.catco2= round((mean(x$activity[x$cat=="B" & x$co.cat=="co2"])-
                   mean(x$activity[x$cat=="A" & x$co.cat=="co2"] ))-
                   (mean(x$activity[x$cat=="B" & x$co.cat=="co1"] )-
                mean(x$activity[x$cat=="A" & x$co.cat=="co1"] ) ),4),
  catC_co.catco2= round((mean(x$activity[x$cat=="C" & x$co.cat=="co2"] )-
                mean(x$activity[x$cat=="A" & x$co.cat=="co2"] )) -
               (mean(x$activity[x$cat=="C" & x$co.cat=="co1"] )  -
               mean(x$activity[x$cat=="A" & x$co.cat=="co1"] ) ),4)
)
```

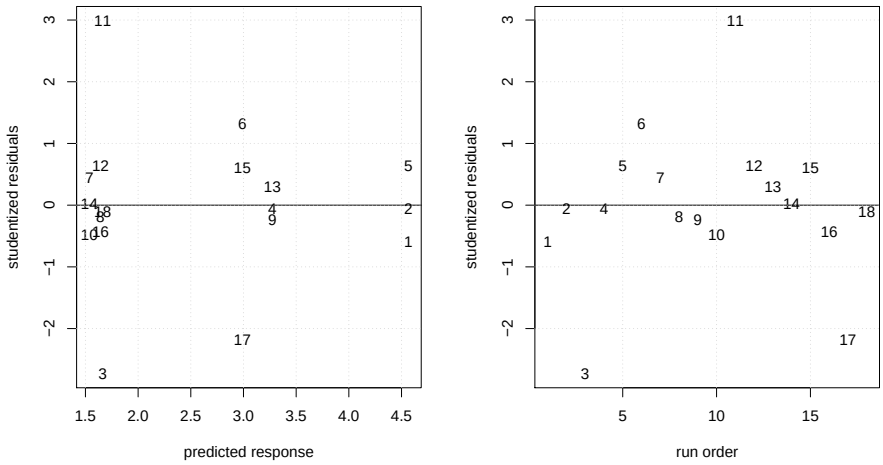

Figure 4.5: Some model diagnostics of the ANOVA model written in R-formula notation as activity (cat + co.cat)$\hat{2}$.

```
data.frame(ols.est) # print
```

```
##                ols.est
## intercept       4.5658
## catB           -2.9017
## catC           -3.0284
## co.catco2      -2.9226
## catB_co.catco2  4.2489
## catC_co.catco2  4.6592
```

```
par(mfrow=c(1,2))
plot(predict(res),rstudent(res),type="n",
     ylab="studentized residuals", xlab="predicted response")
text(predict(res),rstudent(res),label=1:nrow(x))
abline(h=0)
grid()
plot(1:nrow(x),rstudent(res),type="n",
     ylab="studentized residuals", xlab="run order")
text(1:nrow(x),rstudent(res),label=1:nrow(x))
abline(h=0)
grid()
```

The six OLS parameters from the two-way **ANOVA** can be interpreted as

follows

1. **Intercept**: It is the mean average of the reference group, mean(activity|cat="A" & co.cat="co1")=4.5658.
2. **catB**: It is the mean contrast B-A with co.cat at level "co1", i.e. mean(activity|cat="B"&co.cat="co1")- mean(activity|cat="A"&co.cat="co1")=-2.9017.
3. **catC**: Likewise follows mean(activity|cat="C"&co.cat="co1") - mean(activity|cat="A"&co.cat="co1")=-3.0284.
4. **co.catco2**: It is the mean constrast co2-co1 given cat at reference A, namely mean(activity|cat="A"&co.cat="co2") - mean(activity|cat="A"&co.cat="co1")= -2.9226.
5. **catB:co.catco2**: The interaction term can be interpreted as the contrast (B-A|co2) - (B-A|co1)= 4.2489 and so quantifies the heterogeneity of the first order contrast B-A over co.cat.
6. **catC:co.catco2**: Analog to #5 follows the interaction contrast (C-A|co2) - (C-A|co1)= 4.6592.

Model interpretation here is already quite complex with the OLS estimators expressed as conditional contrasts of the response **activity**. However, these conditional contrasts do not nescessarily agree with the contrasts of scientific interest, and it is therefore recommended to refrain from parameter interpretation of ANOVA models and to specify the contrasts of interest directly with some auxiliary functions in R.
For instance, the expected value of the six treatment cells cat*co.cat can be estimated together with test statistics, and from these so called Least-Squares-Means (LSM) all $\frac{6\cdot5}{2} = 15$[6] treatment constrast can be calculated. Least-Squares-Means, contrasts and LSM-plots can be obtained with the following r-code based on the OLS model.

```
library(plotrix)
x <- data.frame(lsmeans::lsmeans(res,~cat*co.cat)  ) # lsmeans
pairs(emmeans::emmeans(res,~cat*co.cat)) # all contrasts
```

```
##  contrast      estimate    SE df t.ratio p.value
##  A,co1 - B,co1   2.9017 0.755 12  3.843  0.0221
##  A,co1 - C,co1   3.0284 0.755 12  4.011  0.0167
##  A,co1 - A,co2   2.9226 0.755 12  3.871  0.0211
##  A,co1 - B,co2   1.5755 0.755 12  2.086  0.3544
##  A,co1 - C,co2   1.2918 0.755 12  1.711  0.5500
##  B,co1 - C,co1   0.1267 0.755 12  0.168  1.0000
##  B,co1 - A,co2   0.0209 0.755 12  0.028  1.0000
##  B,co1 - B,co2  -1.3262 0.755 12 -1.756  0.5243
##  B,co1 - C,co2  -1.6099 0.755 12 -2.132  0.3339
##  C,co1 - A,co2  -0.1058 0.755 12 -0.140  1.0000
```

[6] There are six cat-co.cat combinations (cat and co.cat are on three and two levels, respectively) leading to a total of 6x5/2 contrasts.

```
##  C,co1 - B,co2  -1.4530 0.755 12 -1.924  0.4337
##  C,co1 - C,co2  -1.7366 0.755 12 -2.300  0.2655
##  A,co2 - B,co2  -1.3472 0.755 12 -1.784  0.5089
##  A,co2 - C,co2  -1.6308 0.755 12 -2.160  0.3218
##  B,co2 - C,co2  -0.2836 0.755 12 -0.376  0.9988
##
## P value adjustment: tukey method for comparing a family of 6 estimates
```

```
x <- x[order(x$cat,x$co.cat),]
x
```

```
##   cat co.cat   lsmean        SE df  lower.CL upper.CL
## 1   A    co1 4.565767 0.5339293 12 3.4024347 5.729099
## 4   A    co2 1.643133 0.5339293 12 0.4798014 2.806465
## 2   B    co1 1.664067 0.5339293 12 0.5007347 2.827399
## 5   B    co2 2.990300 0.5339293 12 1.8269680 4.153632
## 3   C    co1 1.537333 0.5339293 12 0.3740014 2.700665
## 6   C    co2 3.273933 0.5339293 12 2.1106014 4.437265
```

```
p.x <- rep(c(0.9,1.1),3)+ rep(0:2,each=2)
plot(p.x,x$lsmean,type="p",pch=16,col=rep(c("red","blue"),3),axes=F,
     xlab="cat",ylab="Least-Square-Means", ylim=c(min(x$lower.CL),max(x$upper.CL)   ))
axis(1,at=1:3,labels=LETTERS[1:3])
axis(2)
plotCI(p.x,x$lsmean,ui=x$upper.CL,li=x$lower.CL,
       err="y",add=TRUE,pch=" ",col=rep(c("red","blue"),3), gap=0)
grid()
box()
legend("top",c("co.cat","co1","co2"),text.col=c("black","red","blue"))
```

The combination cat=“A” and co.cat=“co1” sticks out significantly thereby indicating a positive synergism between cat=“A” and co.cat=“co1”.
Historically, problems with all predictors being categorial and responses continuous are called **AN**alysis **O**f **VA**riance (**ANOVA**) and deal with estimating and comparing many mean values. Multiway-ANOVA (in the example: two-way ANOVA) can be conceived as a multidimensional generalization of the two-sample t-test within the framework of OLS.
Between multiple regression with all X-variables and responses continuous and ANOVA lies the **AN**alysis of **COVA**riance (**ANCOVA**) method with predictors being a mix of continuous and categorial factors. As the following example will show, ANCOVA problems can be treated conveniently within the framework of OLS. In the above example, a strong and significant synergism between cat=“A” and co.cat=“co1” was observed. In the ANOVA set-up the co.cat-concentration was held constant at $\text{co.cat}_{\text{co1}}$= 1% for all cats. In a follow-up experiment the $\text{co.cat}_{\text{co1}}$ concentration is varied in the range co1={1,3%} on 5 levels for the three catalysts, cat={A,B,C} and the activity of the 3x5 factorial design was measured. For the purpose of demonstration the “true” model is given by the

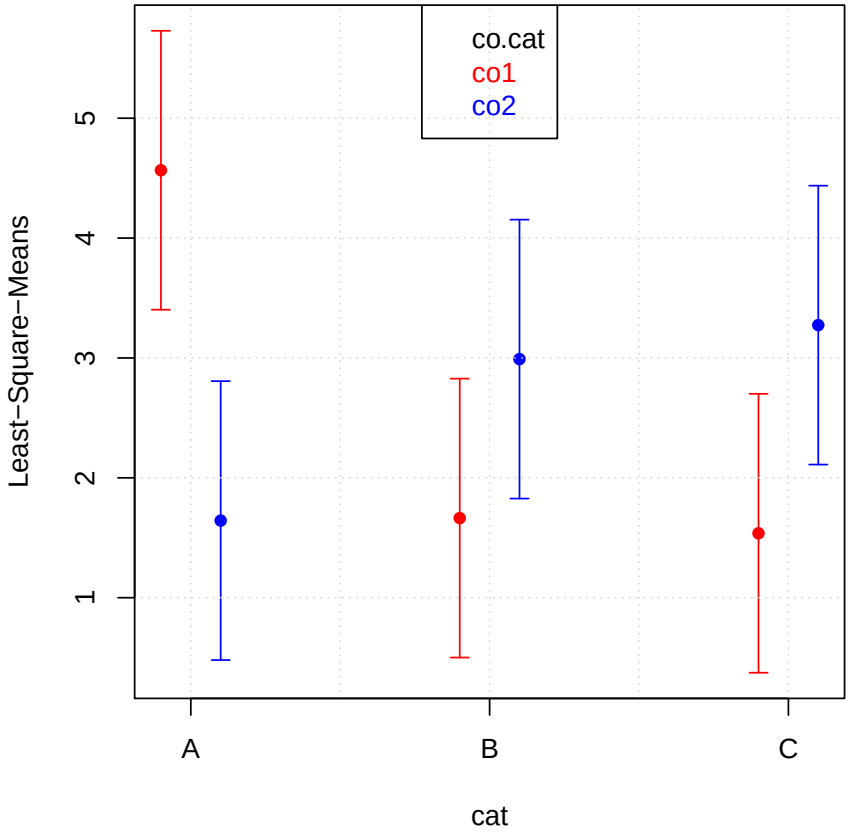

Figure 4.6: Interaction plot cat|co.cat

linear model

$$activity = 5 \cdot I(cat = A) - 1 \cdot I(cat = B) - 2 \cdot I(cat = C)$$

$$-1 \cdot I(cat = A) \cdot co.cat + 2 \cdot I(cat = B) \cdot co.cat + 3 \cdot I(cat = C) \cdot co.cat + \epsilon;$$

$$\epsilon \sim N(0, 1)$$

The data is depicted in figure 4.7 and listed in table 4.3 along with the outcome of the joint ANCOVA OLS-analysis activity ~ (cat+co.cat)^2.[7]

```
rm(list=ls())
set.seed(123)
library(ggplot2)

x <- data.frame(cat=rep(LETTERS[1:3],each=5),
            co.cat=rep(seq(1,3,length=5), 3)  )
x  <- x[order(runif(nrow(x))), ]
rownames(x) <- NULL
x$activity <- 5*(x$cat=="A")  -1*(x$cat=="B") +
       -2*(x$cat=="C") - 1*(x$cat=="A")*x$co.cat +
       2*(x$cat=="B")*x$co.cat +  3*(x$cat=="C")*x$co.cat +
  rnorm(nrow(x),0,1)
knitr::kable(
  x, booktabs = TRUE,row.names=T, digits=3,align="c",
  caption = '3x5 cat x co.cat design for estimating the
  joint model activity ~ (cat+co.cat)\\^2 with OLS')
```

```
ggplot(data = x, aes(x = co.cat, y = activity, colour = cat)) +
    geom_smooth(method="lm", se=F) + geom_point() + theme_bw()
```

```
summary(res <- lm(activity~(cat + co.cat)^2, data=x))
```

```
##
## Call:
## lm(formula = activity ~ (cat + co.cat)^2, data = x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6559 -0.7640  0.2762  0.6139  1.6092
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7.0439     1.6255   4.333   0.0019 **
## catB         -6.8868     2.2988  -2.996   0.0151 *
## catC         -6.9193     2.2988  -3.010   0.0147 *
```

[7] Note that in R the formula expression (...)^k will expand all factors in the argument list into main effects plus k-way interactions.

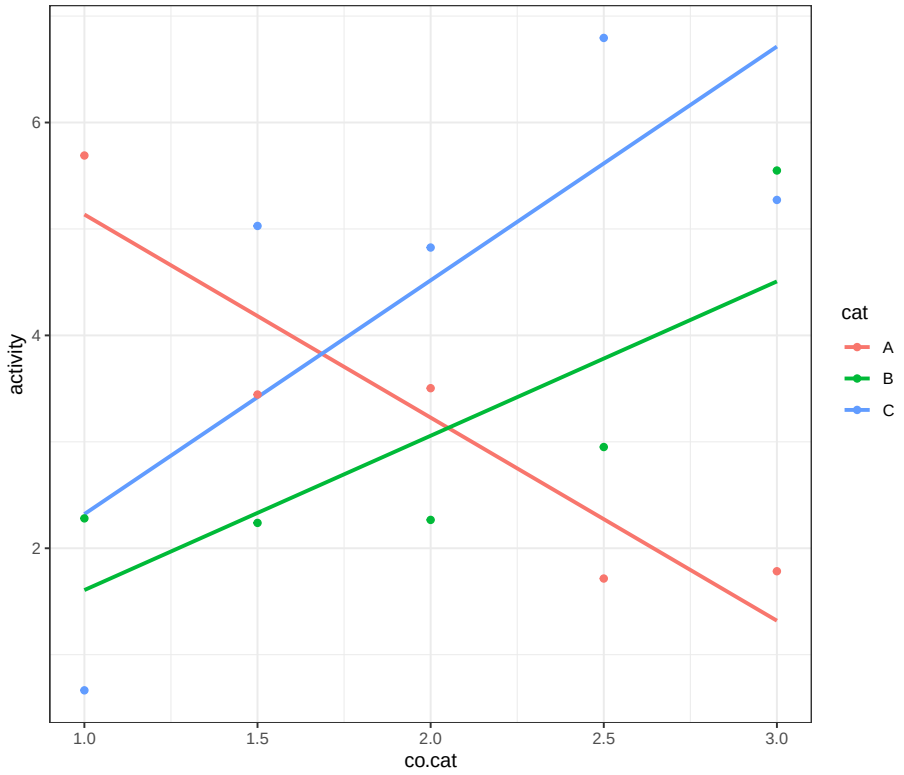

Figure 4.7: Plot of activity co.cat by cat of the ANCOVA example

Table 4.3: 3x5 cat x co.cat design for estimating the joint model activity (cat+co.cat)$\hat{2}$ with OLS

	cat	co.cat	activity
1	B	1.0	2.281
2	C	3.0	5.273
3	A	1.0	5.690
4	A	2.0	3.504
5	C	1.5	5.028
6	B	3.0	5.549
7	B	1.5	2.238
8	B	2.5	2.951
9	C	2.5	6.795
10	C	2.0	4.826
11	A	1.5	3.444
12	A	2.5	1.716
13	B	2.0	2.266
14	A	3.0	1.784
15	C	1.0	0.665

```
## co.cat       -1.9082     0.7663  -2.490   0.0344 *
## catB:co.cat   3.3582     1.0837   3.099   0.0127 *
## catC:co.cat   4.1045     1.0837   3.788   0.0043 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.212 on 9 degrees of freedom
## Multiple R-squared:  0.7128, Adjusted R-squared:  0.5533
## F-statistic: 4.468 on 5 and 9 DF,  p-value: 0.02527
```

With factor level A being the reference, the reported estimates are contrasts against the reference, here $\Delta a_i = a_i - a_A$. Keeping this in mind, the regression model reads

$$activity = a_0 + \Delta a_{0,B} \cdot I(cat = B) + \Delta a_{0,C} \cdot I(cat = C) + a_1 \cdot co.cat$$

$$+ \Delta a_{1,B} \cdot I(cat = B) \cdot co.cat + \Delta a_{1,C} \cdot I(cat = C) \cdot co.cat$$

From the above OLS estimates unbiased estimates of, say, B can be obtained with

$$activity_B(co.cat) = a_0 + \Delta_{0,B} + (a_1 + \Delta a_{1,B}) \cdot co.cat$$

$$= 7.0439 - 6.8868 + (-1.9082 + 3.3582) \cdot co.cat$$

$$= 0.1571 + 1.45 \cdot co.cat$$

Table 4.4: OLS parameter from stratified regression analysis of the ANCOVA example

Parameter	cat	Estimate	Std. Error	t value	Pr(>|t|)
Intercept	A	7.0439	0.9342	7.5400	0.0048
co.cat	A	-1.9082	0.4404	-4.3329	0.0227
Intercept	B	0.1571	1.3108	0.1199	0.9122
co.cat	B	1.4500	0.6179	2.3466	0.1006
Intercept	C	0.1246	2.3100	0.0539	0.9604
co.cat	C	2.1963	1.0889	2.0170	0.1371

which is within standard error in agreement with the model having generated the data[8].
It is worthwhile comparing the joint ANCOVA analysis with the results from OLS analysis stratified by cat[9]. OLS results of linear regression analysis stratified by cat are summarized in table 4.4. Stratified analysis yields the same location parameter as the joint ANCOVA analysis, however the ANCOVA test statistics is different and more powerful, as there are more degrees of freedom for estimating the error variance (DF_{ANCOVA}=9; $DF_{stratified}$=3). This will render the ANCOVA error variance smaller compared to the stratified results, hence smaller effects can be detected with the joint ANCOVA model. Joint model analysis of data is usually superior to stratified data analysis in terms of power and usually less laborious than stratified analysis. Joint modeling should therefore be the method of first choice when analysing complex designs.

4.3 Full factorial 2^K designs

Full factorial 2^K designs are a subclass of the above grid designs with all K-factors on two levels only. 2^K factorials are an important design class, because from these designs many other designs can be derived such as screening designs, fractional factorial designs, and central composite designs, all to be discussed later in more detail. The levels of the factors x_1, x_2 ... x_K are usually given in coded units, {-1,+1} with -1 and +1 denoting the lower and upper bounds of the K-factors. A simple 2^3 full factorial for the factors x_1, x_2 and x_3 is depicted in figure 4.8

[8]
$$activity = 5 \cdot I(cat = A) - 1 \cdot I(cat = B) - 2 \cdot I(cat = C)$$
$$-1 \cdot I(cat = A) \cdot co.cat + 2 \cdot I(cat = B) \cdot co.cat + 3 \cdot I(cat = C) \cdot co.cat + \epsilon;$$
$$\epsilon \sim N(0, 1)$$

[9]Stratified analysis means: Take the data of each cat individually and do OLS with each data stratum A,B,C separately.

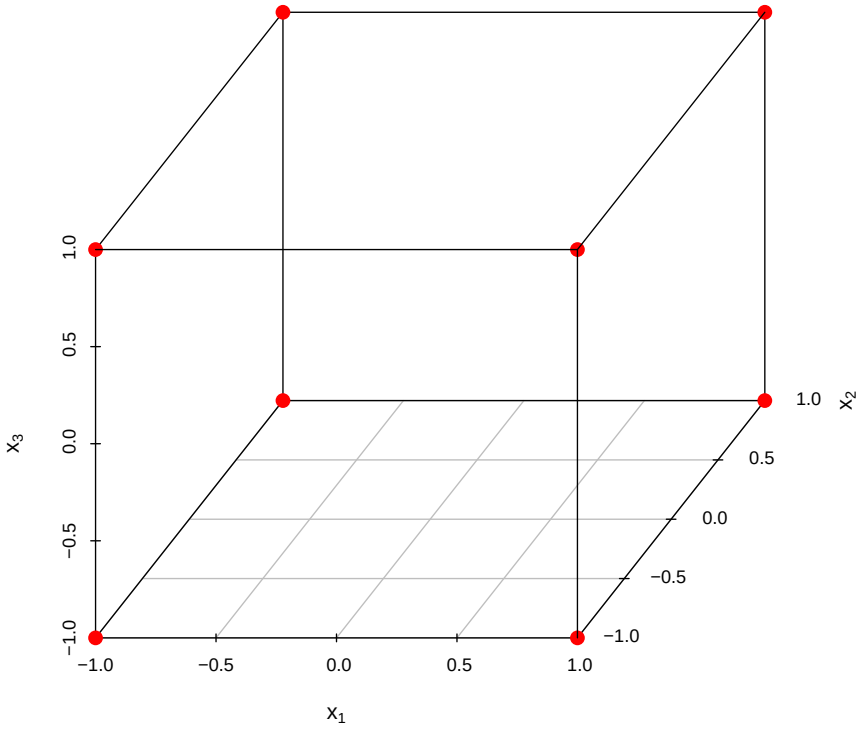

Figure 4.8: 2^3 full factorial design with factorial points labelled red

Full factorial 2^{K} designs span a K-dimensional sphere $\mathbb{R}^{\urcorner}$, these are squares in two, cubes in three and hypercubes in K>3 dimensions, with 2^{K} vertex points of the K-cube.
It can be shown that full factorial designs provide support for estimating all linear effects and all K-way interactions, schematically written

$$y = \sum (\text{1-way}), (\text{2-way}), (\text{3-way}), ...(\text{K-way})$$

with 1-way interaction denoting main effects $\mathrm{x_i}$, 2-way bilinear terms $\mathrm{x_i x_j}$, 3-way trilinear terms $\mathrm{x_i x_j x_k}$ etc. up to K-way term $\mathrm{x_1 x_2 x_3 \ldots x_K}$. Given K factors the number of k-way interaction terms can be calculated with the well known combinatorial expression (4.1).

$$\binom{K}{k} = \frac{K!}{k!(K-k)!} \tag{4.1}$$

Summing up over all interactions is equal to the number of support points of the 2^{K} design and given by equation (4.2)

$$\sum_{k=1}^{K} \binom{K}{k} = 2^K \tag{4.2}$$

The linear model supported by a 2^4 design can thus be expressed by the lengthy expression

$$y = a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3 + a_4 \cdot x_4 +$$
$$a_{12} \cdot x_1 \cdot x_2 + a_{13} \cdot x_1 \cdot x_3 + a_{14} \cdot x_1 \cdot x_4 + a_{23} \cdot x_2 \cdot x_3 + a_{24} \cdot x_2 \cdot x_4 + a_{34} \cdot x_3 \cdot x_4 +$$
$$a_{123} \cdot x_1 \cdot x_2 \cdot x_3 + a_{124} \cdot x_1 \cdot x_2 \cdot x_4 + a_{234} \cdot x_2 \cdot x_3 \cdot x_4 + a_{134} \cdot x_1 \cdot x_3 \cdot x_4 +$$
$$a_{1234} \cdot x_1 \cdot x_2 \cdot x_3 \cdot x_4 + \epsilon$$

There are four main effects, ten 2-way interactions, four 3-way interactions and one 4-way interaction being supported by 16 (2^4) design points, so there are 16 model parameters to be estimated from 16 runs. When the number of runs N equals the number of regression parameters p, the design is called saturated. Such designs are estimable, however they have no degrees of freedom for estimating the error variance, hence OLS will always fit the data with R^2=1 similar to fitting a line with two support points only.
Based on the Taylor series expansion, these complex expressions can be simplified. Given any smooth and continuous function f(), the higher order terms of the Taylor series expansion will vanish, and this fundamental property is being exploited by assuming the higher order terms >2-way to be zero or close to zero. With this assumption, all interaction terms>2 ≈ 0, the model equation from above becomes

$$y = a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3 + a_4 \cdot x_4 +$$

Table 4.5: Redundancy R(K) for different 2^K designs under bilinear nodel assumption

K	Redundancy
2	1.00
3	1.14
4	1.45
5	2.00
6	2.91
7	4.41
8	6.92
9	11.13
10	18.29

$$a_{12} \cdot x_1 \cdot x_2 + a_{13} \cdot x_1 \cdot x_3 + a_{14} \cdot x_1 \cdot x_4 + a_{23} \cdot x_2 \cdot x_3 + a_{24} \cdot x_2 \cdot x_4 + a_{34} \cdot x_3 \cdot x_4 + \epsilon;$$
$$\epsilon \sim N(0, \sigma^2)$$

usually written in short notation

$$y = a_0 + \sum_{i=1}^{4} a_i \cdot x_i + \sum_{j>i}^{4} a_{ij} \cdot x_i \cdot x_j + \epsilon$$

In this formula ϵ stands either for the effects of the hidden confounders Δz_K and/or the sum of the neglected higher order terms. At any rate, the Central Limit Theorem will "work" towards normalizing these terms into something close to $\epsilon \sim N(0, \sigma^2)$. Now, the above model, after omitting higher order interactions, becomes a bilinear surrogate model of the true, however unknown function f(), that is

$$f(\Delta x_1, \Delta x_2 ... \Delta x_I) \approx a_0 + \sum_{i} a_i \cdot x_i + \sum_{j>i} a_{ij} \cdot x_i \cdot x_j$$

with p=11 regression parameters, hence there are N-p=5 degrees of freedom for estimating σ^2.
The redundancy factor R can be defined as the number of trials N divided by the number of model parameters p, $R = \frac{N}{p}$. Assuming a bilinear model structure, redundancies R(K) can be calculated for the full factorial designs as a function of the dimension K according to

$$R(K) = \frac{2^K}{1 + K + \frac{K \cdot (K-1)}{2}}$$

some of which are listed in table 4.5.

According to table 4.5, a five factor design, 2^5, has already a redundancy of 2, meaning, that one half of the experimental effort (16 trials) is spend on estimating

the error variance. A redundancy of two is probably a number too large for making a 2^5 design an economical choice under bilinear model assumption.
As a **rule of thumb**, R_K should be in the range $1 \leq R_K \leq 1.5$ with the lower bound indicating a saturated design and the upper value admitting one third of the experimental effort for estimating the error variance. Given this rule, the number of factors suitable for full factorial analysis are restricted to K=2,3,4 with K=4 being the ideal number. The two factor design K=2 is a saturated design requiring additional trials for estimating the error variance. For this and other reasons explained below, full factorial or fractional factorial designs should always be augmented by replicates, usually triple replicates, located at the center of the design because

1. Placing replicates in the center will not disturb the balance (orthogonality) of the design while providing an unbiased estimate of the pure replication error of the experimental setup.
2. Replicates at the center serve as a check for the presence of convex effects (minima, maxima) and are a base for estimating the sum of the quadratic model terms $\sum_i a_{ii}$, the "global convexity".

Figure 4.9 shows a 2^3 design augmented by triple replicates at the center, (0,0,0).

The discussed concepts become transparent with the following simulation example involving six steps, namely

1. A full factorial coded design + three center point is created using the FrF2 function of the library "FrF2". Note that FrF2 excludes the center points from randomization by placing them at the beginning, middle and end of the design. This optimally supports estimation of serial effects.
2. The design is uncoded with a function from the DoE library "rsm" with the linear transformation $x_{uncoded} = halfway \cdot x_{coded} + center$
3. κ is being calulated for the coded, uncoded and standardized uncoded design showing κ to be highly scale sensitive. Hence, unit variance scaling ("standardization") is mandatory when using kappa on uncoded data.
4. Response y is realized from the assumed true model $y = 10 + 2 \cdot x_1 + 3 \cdot x_2 + 5 \cdot x_1 \cdot x_2 + 2 \cdot x_3^2 + N(0, 1)$
5. The following three OLS models are given in R notation[10] and estimated from the data:
   - M1: y~(x1+x2+x3+x4)^2;
   - M2: y~(x1+x2+x3+x4)^2 + I(x1^2);
   - M3: y~(x1+x2+x3+x4)^2 + I(x4^2);
6. Residuals for M1 and M2 are depicted in figure 4.10

```
rm(list=ls())
set.seed(12556)
x <- FrF2::FrF2(nruns=16,nfactors=4,
```

[10]Note that a square term in R formula notation must be specified as "I(x1^2)" with the "as.is" function I(); see "?lm" and the R-introduction for more details on how to specify formula objects in R

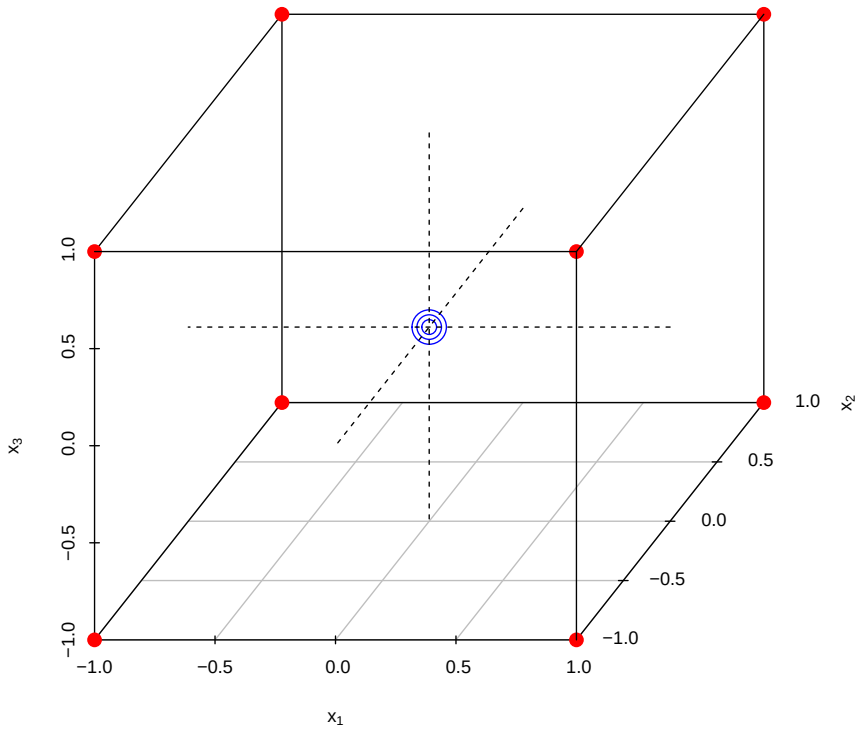

Figure 4.9: 2^3 full factorial design with triple replicates marked blue in the center

```
     factor.names=paste("x",1:4,sep=""), ncenter=3, randomize=F)
```

```
## creating full factorial with 16 runs ...
```

```
x <- as.data.frame(x)
x <- x[order(runif(nrow(x))),]
x.uncoded <- data.frame(rsm::coded.data(x,temp~ 50*x1+100,
    cat~4*x2+4, stirrer~100*x3+100, conc~10*x4+10) )# uncode
cbind(x,x.uncoded)  # plot coded + uncoded design
```

```
##    x1 x2 x3 x4 temp cat stirrer conc
## 10  1 -1 -1  1  150   0       0   20
## 2   1 -1 -1 -1  150   0       0    0
## 12  1  1 -1  1  150   8       0   20
## 15 -1  1  1  1   50   8     200   20
## 19  0  0  0  0  100   4     100   10
## 5  -1 -1  1 -1   50   0     200    0
## 4   1  1 -1 -1  150   8       0    0
## 18  0  0  0  0  100   4     100   10
## 3  -1  1 -1 -1   50   8       0    0
## 6   1 -1  1 -1  150   0     200    0
## 14  1 -1  1  1  150   0     200   20
## 13 -1 -1  1  1   50   0     200   20
## 8   1  1  1 -1  150   8     200    0
## 7  -1  1  1 -1   50   8     200    0
## 9  -1 -1 -1  1   50   0       0   20
## 11 -1  1 -1  1   50   8       0   20
## 17  0  0  0  0  100   4     100   10
## 16  1  1  1  1  150   8     200   20
## 1  -1 -1 -1 -1   50   0       0    0
```

```
kappa(model.matrix(~(x1+x2+x3+x4)^2, data=x))   # kappa coded
```

```
## [1] 1.082183
```

```
kappa(model.matrix(~(temp +cat +stirrer+ conc)^2,
                   data=x.uncoded))           # kappa uncoded
```

```
## [1] 53062.32
```

```
kappa(model.matrix(~(temp +cat +stirrer+ conc)^2,
  data=data.frame(scale(x.uncoded))))
```

```
## [1] 1.02579
```

```
     # kappa on standardized uncoded data

set.seed(1234)
x$y  <- 10 + 2*x$x1 + 3*x$x2 + 5*x$x1*x$x2 +
```

```
       5*x$x3^2 + rnorm(nrow(x),0,1)
round(cor(model.matrix(~x1+x2+x3+x4+I(x1^2)+I(x2^2)
    +I(x3^2)+I(x4^2),data=x)[,-1]),2) # note confounding X^2
```

```
##         x1 x2 x3 x4 I(x1^2) I(x2^2) I(x3^2) I(x4^2)
## x1       1  0  0  0       0       0       0       0
## x2       0  1  0  0       0       0       0       0
## x3       0  0  1  0       0       0       0       0
## x4       0  0  0  1       0       0       0       0
## I(x1^2)  0  0  0  0       1       1       1       1
## I(x2^2)  0  0  0  0       1       1       1       1
## I(x3^2)  0  0  0  0       1       1       1       1
## I(x4^2)  0  0  0  0       1       1       1       1
```

```
summary(res1 <- lm(y~(x1+x2+x3+x4)^2, data=x))
```

```
##
## Call:
## lm(formula = y ~ (x1 + x2 + x3 + x4)^2, data = x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.3662  0.1089  0.5638  1.0582  1.8634
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 13.81952    0.64954  21.276 2.50e-08 ***
## x1           1.99071    0.70782   2.812   0.0228 *
## x2           2.90832    0.70782   4.109   0.0034 **
## x3          -0.30349    0.70782  -0.429   0.6794
## x4          -0.07570    0.70782  -0.107   0.9175
## x1:x2        5.23157    0.70782   7.391 7.68e-05 ***
## x1:x3       -0.02585    0.70782  -0.037   0.9718
## x1:x4        0.13227    0.70782   0.187   0.8564
## x2:x3       -0.17196    0.70782  -0.243   0.8142
## x2:x4        0.02173    0.70782   0.031   0.9763
## x3:x4       -0.37889    0.70782  -0.535   0.6070
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.831 on 8 degrees of freedom
## Multiple R-squared:  0.9091, Adjusted R-squared:  0.7955
## F-statistic:     8 on 10 and 8 DF,  p-value: 0.00354
```

```
summary(res2 <- lm(y~(x1+x2+x3+x4)^2 + I(x1^2), data=x))
```

```
##
```

```
## Call:
## lm(formula = y ~ (x1 + x2 + x3 + x4)^2 + I(x1^2), data = x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.20396 -0.33077 -0.08242  0.50941  1.10800
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.79049    0.54768  17.876 4.23e-07 ***
## x1           1.99071    0.23715   8.394 6.70e-05 ***
## x2           2.90832    0.23715  12.264 5.50e-06 ***
## x3          -0.30349    0.23715  -1.280    0.241
## x4          -0.07570    0.23715  -0.319    0.759
## I(x1^2)      4.78447    0.59682   8.017 9.00e-05 ***
## x1:x2        5.23157    0.23715  22.060 9.94e-08 ***
## x1:x3       -0.02585    0.23715  -0.109    0.916
## x1:x4        0.13227    0.23715   0.558    0.594
## x2:x3       -0.17196    0.23715  -0.725    0.492
## x2:x4        0.02173    0.23715   0.092    0.930
## x3:x4       -0.37889    0.23715  -1.598    0.154
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9486 on 7 degrees of freedom
## Multiple R-squared:  0.9911, Adjusted R-squared:  0.977
## F-statistic: 70.63 on 11 and 7 DF,  p-value: 4.318e-06
```

```
summary(res3 <- lm(y~(x1+x2+x3+x4)^2 + I(x4^2), data=x))
```

```
##
## Call:
## lm(formula = y ~ (x1 + x2 + x3 + x4)^2 + I(x4^2), data = x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.20396 -0.33077 -0.08242  0.50941  1.10800
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.79049    0.54768  17.876 4.23e-07 ***
## x1           1.99071    0.23715   8.394 6.70e-05 ***
## x2           2.90832    0.23715  12.264 5.50e-06 ***
## x3          -0.30349    0.23715  -1.280    0.241
## x4          -0.07570    0.23715  -0.319    0.759
## I(x4^2)      4.78447    0.59682   8.017 9.00e-05 ***
```

```
## x1:x2        5.23157    0.23715  22.060 9.94e-08 ***
## x1:x3       -0.02585    0.23715  -0.109    0.916
## x1:x4        0.13227    0.23715   0.558    0.594
## x2:x3       -0.17196    0.23715  -0.725    0.492
## x2:x4        0.02173    0.23715   0.092    0.930
## x3:x4       -0.37889    0.23715  -1.598    0.154
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9486 on 7 degrees of freedom
## Multiple R-squared:  0.9911, Adjusted R-squared:  0.977
## F-statistic: 70.63 on 11 and 7 DF,  p-value: 4.318e-06
```

```
colcol       <- rep("black",nrow(x))
colcol[apply(abs(x[,1:4]),1,sum)==0]  <- "red"
par(mfrow=c(1,2))
plot(predict(res1), rstudent(res1),
xlab=expression( paste("predicted response ", hat(y)) )    ,
ylab="studentized residuals",type="n",
main=expression(paste("(",frac(paste("y - ",
               hat(y)),sigma), ") ~ ", hat(y)   )) )
text(predict(res1), rstudent(res1),label=1:nrow(x), col=colcol)
abline(h=c(-2,0,2),lty=c(2,1,2))
plot(predict(res2), rstudent(res2),
xlab=expression( paste("predicted response ", hat(y)) )    ,
ylab="studentized residuals",type="n",
main=expression(paste("(",frac(paste("y - ",
               hat(y)),sigma), ") ~ ", hat(y)   )) )
text(predict(res2), rstudent(res2),
     label=1:nrow(x), col=colcol)
abline(h=c(-2,0,2),lty=c(2,1,2))
```

The example is instructive by revealing all four square terms x_i^2 confounded with each other (see the correlation analysis of the expanded design matrix). This explains why all models $y \sim (x_1 + x_2 + x_3 + x_4)^2 + x_i^2;\ i = 1, 2, 3, 4$ will report the same square estimates $a_{ii} = 4.39 \pm 0.37$[11]. These modeling results suggest some convexity not being accounted for by the bilinear model M1. The residual plot, figure 4.10, tells a similar story: The residual plot of M1 (left panel) reveals the center points #5,#8,#17 to deviate significantly from the model (the horizontal line) within replication error, thereby indicating the presence of convex effects in the data. On the right panel of figure 4.10 the convex effect is being absorbed (erroneously) by the term x_1^2 and #5,#8,#17 are now unbiasedly predicted by M2.
The ambiguities regarding a_{ii}[2] can be resolved by augmenting the factorial design

[11] Note that the estimates of the square terms, I(x1^2), I(x4^2), are the same, despite x1 and x2 being different.

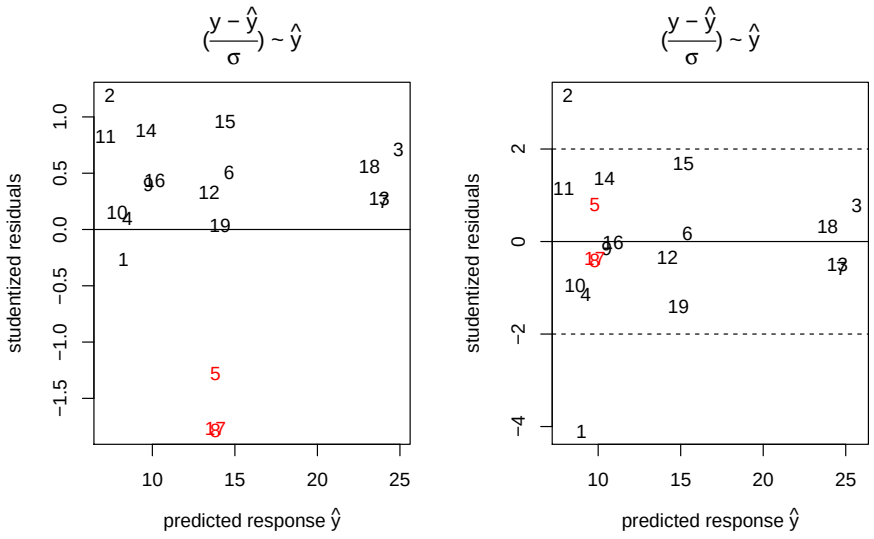

Figure 4.10: Residuals based on the parametric models $\hat{y} = (x_1 + x_2 + x_3 + x_4)^2$ (left panel) and $\hat{y} = (x_1 + x_2 + x_3 + x_4)^2 + x_1^2$ (right panel) with center points labelled red

with star points, a class of trials that will be discussed in more detail in the chapter on RSM designs below.

4.4 Fractional factorial 2^{K-P} designs

Full factorial designs are of limited use for estimating bilinear surrogate models in higher dimension as the design redundany R(K) grows rapidly with dimension K>4. Augmented by center points, full factorial designs can at best be used to economically investigate the effects of two, three or four factors.
High redundancies result from the fact that full factorial designs support higher order effects, here interactions of order >2, which are assumed small and of no scientific interest. Hence, the question arises whether it is possible to omit those points in the design supporting higher order interaction while leaving points supporting 1- and 2-way interactions untouched?
This is in fact possible with a class of designs known as fractional factorial designs, which fill the gap by economically supporting bilinear models of factor dimension K>4.
Fractional factorial designs can be derived analytically from full factorial designs by reassigning higher order interactions to main effects. This will be explained in the following by deriving a five factor resolution 5 design (a $2^{(5-1)}$ design) from a full factorial 2^4 design. Table 4.6 lists a 2^4 design along with its largest

Table 4.6: Coded full factorial 2^4 design with fourway interaction term in R-notation x1:x2:x3:x4

x1	x2	x3	x4	x1:x2:x3:x4
-1	-1	-1	-1	1
1	-1	-1	-1	-1
-1	1	-1	-1	-1
1	1	-1	-1	1
-1	-1	1	-1	-1
1	-1	1	-1	1
-1	1	1	-1	1
1	1	1	-1	-1
-1	-1	-1	1	-1
1	-1	-1	1	1
-1	1	-1	1	1
1	1	-1	1	-1
-1	-1	1	1	1
1	-1	1	1	-1
-1	1	1	1	-1
1	1	1	1	1

possible interaction term the 4-way interaction x1:x2:x3:x4[12], which is simply the product of the four main effects x1-x4. In total, there are 11 interaction terms in a 2^4 design which are all orthogonal (i.e. uncorrelated with $r_{\text{Pearson}}=0$) to each other. Fractional factorial designs make use of this property by assigning a new factor, here x5, to the highest contrast x1:x2:x3:x4. The confounding properties of this new design can be evaluated easily. From the identity $x_5 = x_1 \cdot x_2 \cdot x_3 \cdot x_4$ follows by multiplying with x_5 the important identity

$$x5 \cdot x5 = I = x1 \cdot x2 \cdot x3 \cdot x4 \cdot x5$$

with I denoting the 1-column. The identity $\boxed{I = x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5}$ is called the **defining contrast** of the design with the length (or the shortest length when there are several defining contrasts) of the defining contrast(s) being called design resolution (here: resolution=5).

With the defining contrast being known, the whole confounding structure of the design can be derived. For instance, in the above $2^{(5\text{-}1)}$ design the main effect of, say, x_1 is confounded with the four way term $\mathbf{x_1} = x_1 \cdot I = x_1 \cdot x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5 = I \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5 = \mathbf{x_2 \cdot x_3 \cdot x_4 \cdot x_5}$, while, say, the interaction term $x1 * x2$ is confounded with the three way term $x1 * x2 * I = x1 * x2 * x1 * x2 * x3 * x4 * x5 =$

[12] In R formula language the interaction term between x_1 and x_2 is written $x_1 : x_2$, rather than $x_1 \cdot x_2$.

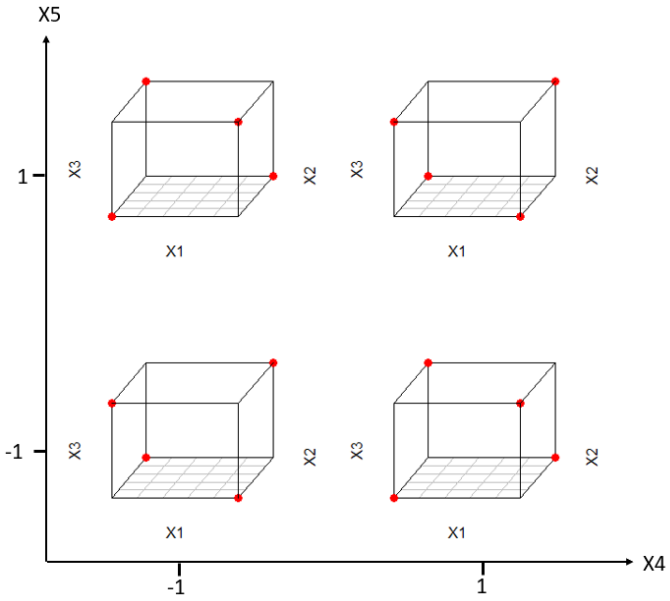

Figure 4.11: Resolution-5 design for five factors $x_1 - x_5$ depicted as five-dimensional trellis plot

$x3 * x4 * x5$. The design resolution as a single number is a short term for expressing the counfounding structure of a fractional factorial design and can be summarized in the rule: **In a resolution R design all k-factor interactions are confounded with (R-k)-factor interactions or higher interactions.** For a resolution 5 (R=5) design, e.g., all main effects (1-way interactions) are confounded with 4-way interactions or higher, 2-way interactions are confounded with 3-way interactions of higher. From that property we can conclude that resolution V designs are ideal candidates for identifying bilinear models, as all model effects of interest (1- and 2-way interactions) are confounded with higher order effects assumed zero. Figure 4.11 gives some insight into the anatomy of a $2^{(5-1)}$ resolution V design. Table 4.7 is an overview of the available resolution V designs as a function of the dimension K>4 along with the corresponding redundancy factors.

According to table 4.7 the relationship between R and K is not monotone anymore, so sometimes larger K leads to lower R(K) and higher dimensional designs can sometimes become more economical than lower dimensional designs in terms of R(K). However, the number of trials of resolution V designs grows rapidly simply by the fact that the number of two way interactions grows approximately quadratic with K according to $\frac{K \cdot (K-1)}{2}$.
This again brings up the question whether further points of the full factorial designs can be omitted so as so render more parsimonious models estimable.

Table 4.7: List of available resolution V designs with redundancy under bilinear model assumption

K	N.trials	Redundancy
5	16	1.00
6	32	1.45
7	64	2.21
8	64	1.73
9	128	2.78
10	128	2.29
11	128	1.91
12	256	3.24
13	256	2.78
14	256	2.42
15	256	2.12

Table 4.8: Expanded designs matrix of a 2^3 full factorial design

x1	x2	x3	x1:x2	x1:x3	x2:x3	x1:x2:x3
-1	-1	-1	1	1	1	-1
1	-1	-1	-1	-1	1	1
-1	1	-1	-1	1	-1	1
1	1	-1	1	-1	-1	-1
-1	-1	1	1	-1	-1	1
1	-1	1	-1	1	-1	-1
-1	1	1	-1	-1	1	-1
1	1	1	1	1	1	1

This is possible and will be demonstrated by constructing a $2^{(7-4)}$ fractional factorial design from a 2^3 full factorial design. The construction process starts by fully expanding a 2^3 design as shown in table 4.8

All interactions (in R notation) are know identified with new factor names according to

1. x4 = x1:x2
   - I = x1:x2:x4
2. x5 = x1:x3
   - I = x1:x3:x5
3. x6 = x2:x3
   - I = x2:x3:x6
4. x7 = x1:x2:x3

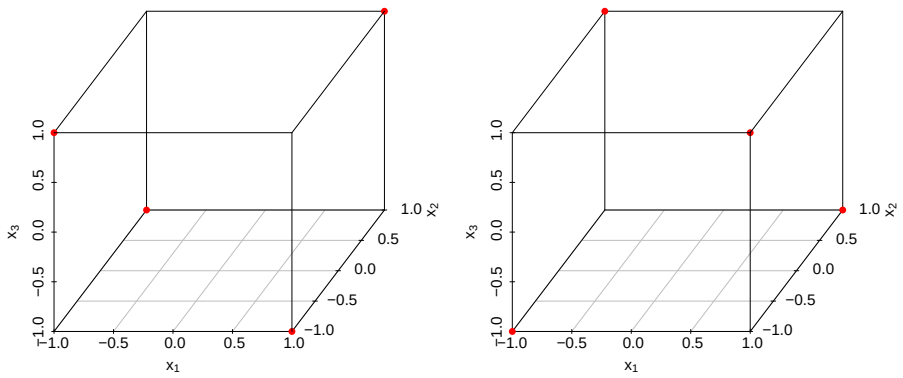

Figure 4.12: Fractional factorial resolution III design (left panel) and corresponding complementary design shown on the right panel.

- I = x1:x2:x3:x7

The shortest length of the defining contrasts in the list is three, hence the resolution of the thus constructed design is R=3. In this design all main effects are confounded with 2-way interactions or higher interactions and will be biased by these high order term (provided these terms are different from zero).
Fractional factorial resolution III designs belong to the class of screening designs and can be used for estimating the main effects of a large number of factors under the assumption that factor interactions can be neglected. With a $2^{(7-4)}$ design seven main effects can be investigated with eight runs. The design is saturated and requires replicates at the design center for an estimate of the error variance.
In a similar way a $2^{(3-1)}$ resolution III design can be constructed from a 2^2 factorial by the identity x3=x1:x2 and that allows estimating three main effects (x1,x2,x3) from four runs. This design and its complementary design are depicted in 4.12 with the complement being the part omitted from the full factorial design in creating the fractional factorials. Note that by combining a fractional factorial design with its complement, the original full factorials design is reconstructed.

Finally, a $2^{(4-1)}$ fractional factorial resolution IV design will be derived from the expanded 2^3 design, table 4.8, by reassigning the 3-way contrast x1:x2:x3 to the factor x4. The defining contrast I = x1:x2:x3:x4 is of length 4, hence the designs is of resolution 4. In the so constructed resolution IV design, main effects are confonded with 3-way interactions (or higher), while 2-way interactions are confounded with 2-way interactions. Put differently, a resolution IV design can only identify a half fraction of the 2-way interactions with the other half being

confounded. For the $2^{(4-1)}$ resolution IV design it can be easily derived from the generator that the following three confounding conditions hold

1. x1:x2=x3:x4
2. x1:x3=x2:x4
3. x1:x4=x2:x3

Resolution IV designs can play an important role in the sequential use of DoE. In experimental projects there is often uncertainty about the significance of interactions. In such a situation, it might be a good choice to estimate only a half fraction of the interaction terms in the first place. If, say, x1:x2 turns out significant in the outcome, the resolution IV design can be complemented easily to render the confounded pair x1:x2 and x3:x4 subsequentially estimable. Sequential strategies can reduce the experimental effort significantly. The FrF2 function aliasprint() reports the confounding structure of fractional factorial designs.

```
x <- FrF2::FrF2(nfactors=4,resolution=4,
    factor.names=paste("x",1:4,sep=""), ncenter=0, randomize=F)
FrF2::aliasprint(x)
```

```
## $legend
## [1] A=x1 B=x2 C=x3 D=x4
##
## $main
## character(0)
##
## $fi2
## [1] AB=CD AC=BD AD=BC
```

4.5 Screening designs

When there are many potential factors, say K>8, of which only a small fraction can be assumed to exert large effects, screening design can be an option for identifying these high leverage candidates. As shown above, fractional factorial resolution III designs belong to the class of screening designs used for estimating K main effects with $N \approx K$ runs. For instance, with the $2^{(7-4)}$ resolution III design from above, we can screen the effects of seven parameters with 8 runs.
There is another class of non-regular fractional factorial screening designs, the Plackett-Burman designs (PB). The number of runs in a PB design is always a multiple of four as opposed to the fractional factorial designs, where the number of runs is always a multiple of two.
Table 4.9 gives an overview of available PB and resolution III designs for K=4, 5...19. It should be noted that PB and resolution III designs coincide for K=4,5,6,7. The list shows that factor number 11,15,19 all reveal redundancies of 1 for the PB designs making PB the design of choice for these factor number.

Table 4.9: List of available **Placket Burman** (PB) and **Resolution III** (R3) designs along with their redundancy assuming a main effect model with K+1 regression parameters

K	N.PB	R.PB	N.FR3	R.FR3
4	8	1.60	8	1.60
5	8	1.33	8	1.33
6	8	1.14	8	1.14
7	8	1.00	8	1.00
8	12	1.33	16	1.78
9	12	1.20	16	1.60
10	12	1.09	16	1.45
11	12	1.00	16	1.33
12	16	1.23	16	1.23
13	16	1.14	16	1.14
14	16	1.07	16	1.07
15	16	1.00	16	1.00
16	20	1.18	32	1.88
17	20	1.11	32	1.78
18	20	1.05	32	1.68
19	20	1.00	32	1.60

Recently (2011) a new class of screening designs was introduced by Jones and Nachtsheim, the Definite Screening Designs (DSD). DSDs are available for K=4-12 factors on three levels with a total of 2K+1 designs runs. DSDs have the property of not confounding main with square effects which can be useful when screening many potentially non-linear factors. The following R-code analyses the confounding properties of a DSD with K=5 by plotting the correlation matrix of all model terms up to second order in figure 4.13. There is virtually no correlation between the main effects and the square terms and the (exact) condition number $\kappa = 8.9$ naively suggests estimability of the full RSM model. **Caveat: Never use kappa on a model matrix with n.col>n.row!** As already mentioned, κ measures closeness to singularity by requiring $\lambda_{min} > 0$ in $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$ and does not signal singularities from rank deficiency. In the present case with n.col(21)>n.row(11), there are ten zero eigenvalues[13].

```
x <- daewr::DefScreen(m=5,c=0)
z <- model.matrix(~ (A + B + C + D + E)^2 +
                    I(A^2) + I(B^2)+ I(C^2)+ I(D^2)+ I(E^2) ,data=x)[,-1]
kappa(x.model  <- model.matrix(~ (A + B + C + D + E)^2 +
                                 I(A^2) + I(B^2)+ I(C^2)+
                                 I(D^2)+ I(E^2) ,data=data.frame(scale(x))),exact=T)
```

```
## [1] 8.85372
```

```
dim(x.model)
```

```
## [1] 11 21
```

```
sv <- svd(x.model) # singular value decomposition
sv$d[1]/sv$d[length(sv$d)] # kappa from SVD
```

```
## [1] 8.85372
```

```
colnames(z)  <- stringi::stri_replace_all_fixed(colnames(z),LETTERS[1:5],
                                                paste("x",1:5,sep=""), vectorize_all =
corrplot::corrplot(cor(z))
```

Despite this singularity, the power of DSDs for identifying non-linear effects under screening conditions (here 2K+1 rather than 1+2K+K(K-1)/2 design runs for RSM models) can be quite remarkable as the following simulation will demonstrate. A DSD for K=8 factors is created and a sample of the true model $y = 3x_1 + 2x_2 + x_4 + x_5 + 5x_4 \cdot x_5 + 3x_3^2 + 4x_1^2 + 2x_6 + x_7 + 5x_7^2 + \epsilon; \epsilon \sim N(0, 1)$ is realized. The data is subsequently analyzed using a hierarchical step forward regression method from the library **daewr** with the user function step.forward(). It selects in a forward manner the terms maximizing the correlation between y and the partial residuals in a hierarchical model building process[14]. Model

[13] A model matrix of dimension 11x21 allows estimation of 11 linear independent parameters. The remaining 10 parameters are linear functions of these independent parameters. Hence, 10 out of 21 eigenvalues will be zero.

[14] Hierarchical model building is the process of including higher order model terms, such as

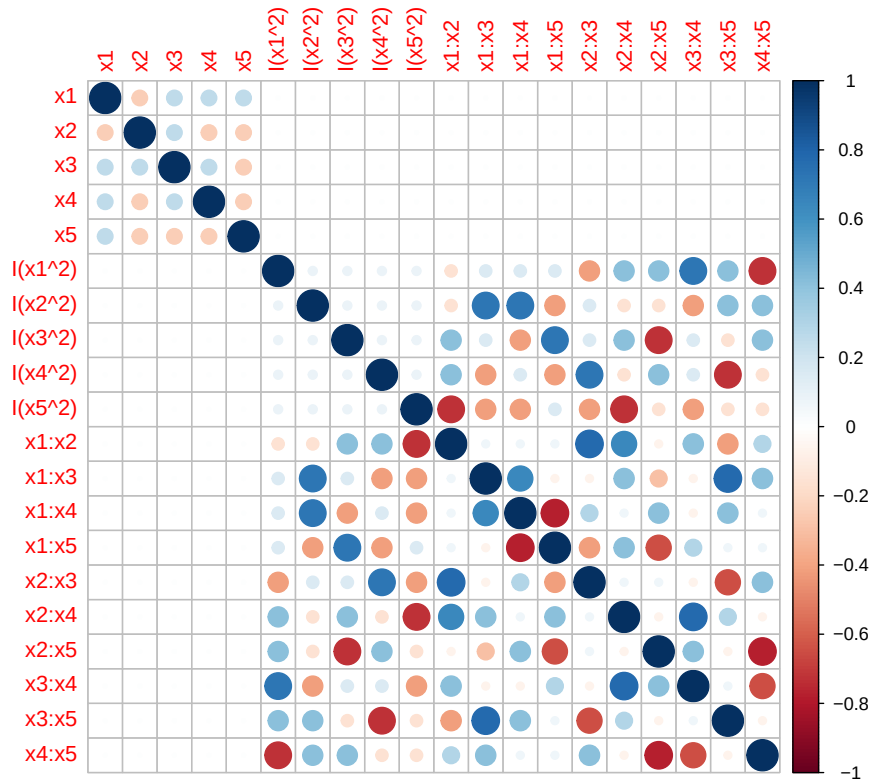

Figure 4.13: Pearson-r of all model terms up to second order of a Definite Screening Design for K=5

hierarchy is important as the significance of higher order terms can only be assessed with the lower order terms already present in the model. The results from eight forward steps are summarized in table 4.10 and the corresponding R^2 values of each step are plotted in figure 4.14.

Results from eight forward, hierarchical RSM regression steps

```
## step.forward
## RSM forward regression keeping model hierarchy
##    y: Response vector
##    x: Data frame of x-variable
## step: Number of forward steps requested
## require(daewr)
rm(list=ls())
step.forward <- function(y,x,step)  {
  require(daewr)
  any.col       <- colnames(x)
  letter.col    <- LETTERS[1:ncol(x)]
  colnames(x)   <- letter.col
  xx            <- x
  xx$y          <- y

  cc <- sum(sapply(x,function(x) nlevels(as.factor(x)) ==2        ))
  mm <- sum(sapply(x,function(x) nlevels(as.factor(x)) ==3        ))


  suppressWarnings(invisible(capture.output(res<- ihstep(y,x,m=mm,c=cc)))    )
  f         <- paste("y~", paste(res,collapse="+"),sep="")
  f2        <- paste("y ~",paste(stringi::stri_replace_all_fixed(res,letter.col ,
                                                                 any.col, vectorize_al
  collect   <- data.frame(formula=f2, R2= round(summary(lm(formula(f),data=xx) )$r.squa
                          ,stringsAsFactors = F     )

  for (i in (2:step)) {
    suppressWarnings( invisible(capture.output(res <- fhstep(y,x,m=mm,c=cc,res)))  )
    f          <- paste("y~", paste(res,collapse="+"),sep="")
    f2        <- paste("y ~",paste(stringi::stri_replace_all_fixed(res,letter.col ,
                                                                   any.col, vectorize_
    collect    <- rbind(collect,data.frame(formula=f2, R2= round(summary(lm(formula(f)
                                           stringsAsFactors = F))
  }
  return(collect)
}
set.seed(1234)
x           <- daewr::DefScreen(m=8,c=0)
```

square and interaction terms, not before their corresponding main effects have been included.

```
colnames(x) <- paste("x",1:8,sep="")
x$y         <-  3*x$x1 + 2*x$x2 + x$x4 + x$x5 +
  5*x$x4*x$x5 + 3*x$x3^2 +
  4*x$x1^2 + 2*x$x6 + x$x7 +
  5*x$x7^2 + rnorm(nrow(x),0,1)
z <- step.forward(x$y,x[,-9], 8 )
# show best model #4
summary(lm(formula(as.character(z[4,1])),data=x) )
```

```
##
## Call:
## lm(formula = formula(as.character(z[4, 1])), data = x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1103 -0.7592 -0.2003  0.6360  1.3022
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.5909     1.1303   0.523 0.619877
## I(x7^2)       5.1030     1.0770   4.738 0.003198 **
## x7            1.2834     0.3732   3.439 0.013824 *
## x2            2.0694     0.3732   5.545 0.001453 **
## x6            1.9827     0.3732   5.313 0.001808 **
## I(x1^2)       5.7737     1.0770   5.361 0.001727 **
## x1            2.9089     0.3732   7.794 0.000235 ***
## x4            1.2095     0.3732   3.241 0.017673 *
## x5            1.3971     0.3732   3.743 0.009583 **
## x2:x6        -0.7809     0.4937  -1.582 0.164816
## x4:x5         4.4906     0.4584   9.796 6.51e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.396 on 6 degrees of freedom
## Multiple R-squared:  0.9875, Adjusted R-squared:  0.9666
## F-statistic: 47.24 on 10 and 6 DF,  p-value: 6.647e-05
```

```
z$formula <- gsub("^", "\\^", z$formula, fixed = T)

knitr::kable(
  z, booktabs = TRUE, row.names=T,align="l",
  caption = 'Results from eight forward,
  hierarchical RSM regression steps
  created from DSD data with the true model
  $y=3x_{1} + 2x_{2} + x_{4} + x_{5} + 5x_{4} \\cdot x_{5} + 3x_{3}^2 + 4x_{1}^2 + 2x_{6} + x_{7}
```

Table 4.10: Results from eight forward, hierarchical RSM regression steps created from DSD data with the true model $y = 3x_1 + 2x_2 + x_4 + x_5 + 5x_4 \cdot x_5 + 3x_3^2 + 4x_1^2 + 2x_6 + x_7 + 5x_7^2 + \epsilon; \epsilon \sim N(0, 1)$

	formula	R2
1	y ~ x4+x5+x4:x5	0.562
2	y ~ I(x1\^2)+x1+x4+x5+x4:x5	0.755
3	y ~ x2+x6+x2:x6+I(x1\^2)+x1+x4+x5+x4:x5	0.916
4	y ~ I(x7\^2)+x7+x2+x6+x2:x6+I(x1\^2)+x1+x4+x5+x4:x5	0.987
5	y ~ x2+x5+x2:x5+I(x7\^2)+x7+x6+x2:x6+I(x1\^2)+x1+x4+x4:x5	0.995
6	y ~ x8+x2+x5+x2:x5+I(x7\^2)+x7+x6+x2:x6+I(x1\^2)+x1+x4+x4:x5	0.998
7	y ~ x3+x8+x2+x5+x2:x5+I(x7\^2)+x7+x6+x2:x6+I(x1\^2)+x1+x4+x4:x5	1.000
8	y ~ x5+x6+x5:x6+x3+x8+x2+x2:x5+I(x7\^2)+x7+x2:x6+I(x1\^2)+x1+x4+x4:x5	1.000

```
plot(1:nrow(z),z$R2,type="b", xlab="model index #",
     ylab=expression(paste("model R" ^ 2) ) )
grid()
```

The marked shoulder of model #4 in figure 4.14 recommends this model as local solution, and this model correctly identifies nine out of ten model terms unbiasedly (the term x2:x6 was erroneously selected, the term $x3^2$ was missed). This result clearly shows that DSDs should be an option when dealing with many, potentially non-linear factors in a screening situation.

4.6 Response surface model designs (RSM designs)

RSM designs aim at identifying all model terms in a full 2^{nd} order model of the general parametric form

$$f(\Delta x_1, \Delta x_2 ... \Delta x_I) \approx a_0 + \sum_i a_i \cdot \Delta x_i + \sum_i a_{ii} \cdot \Delta x_i^2 + \sum_{j>i} a_{ij} \cdot \Delta x_i \cdot \Delta x_j$$

Central Composite (CC) designs are a versatile and flexibel class of designs of first choice for optimally identifying response surface models. As the name already suggests, they are a composite of a factorial (or fractional factorial) part and a so called star part. The CC designs come in two subclasses, namely the Central Composite-Circumscribed (CCC) designs and the Central Composite-Face centered (CCF) designs. Figure 4.15 shows examples of CCC and CCF designs in $\mathbb{R}^{\not\vdash}$ with the factorial and star part labelled red and blue, respectively. In the CCC designs the stars protrude beyond the cube faces, while in a CCF design the star points lie on the cube surface. Consequently, in a CCC design

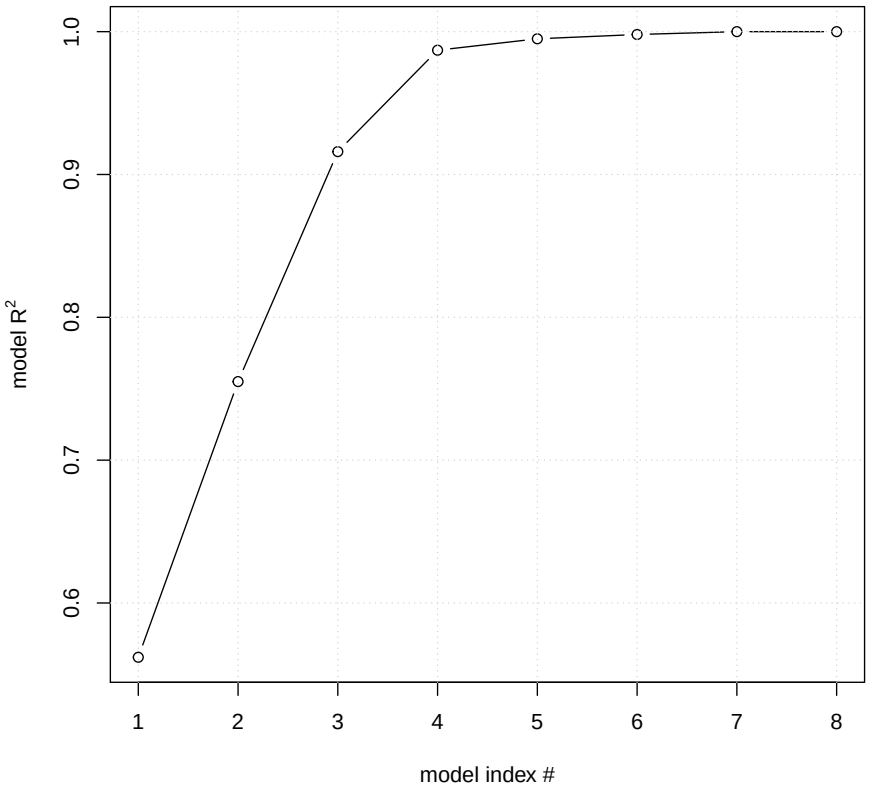

Figure 4.14: Results from eight forward, hierarchical RSM regression steps with the true model $y = 3x_1 + 2x_2 + x_4 + x_5 + 5x_4 \cdot x_5 + 3x_3^2 + 4x_1^2 + 2x_6 + x_7 + 5x_7^2 + \epsilon; \epsilon \sim N(0, 1)$

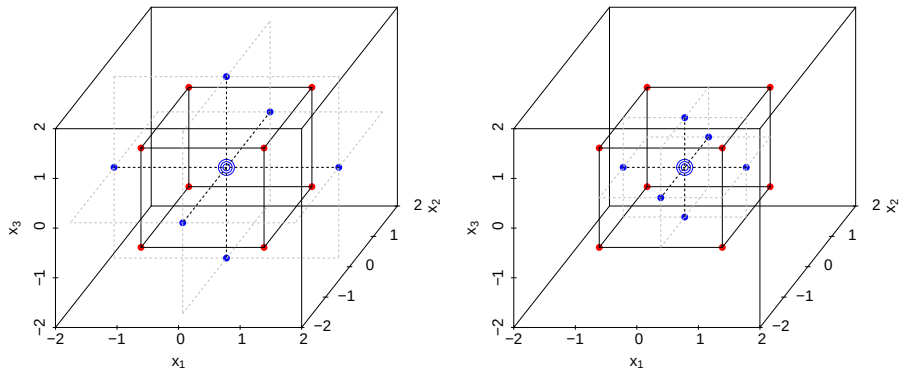

Figure 4.15: Central-Composite-Circumscribed (CCC, left panel) and Central-Composite Face-centered (CCF, right panel) designs for K=3 in coded units

the factors are on five levels, namely and in coded units: $-\alpha, -1, 0, 1, \alpha$, while in a CCF designs the factors are on three levels only, that is -1,0,1. Logistically, CCF designs are easier to handle in the laboratory at the expense of having somewhat lower statistical properties compared to CCC designs.

The extension of the star part α affects the statistical properties of the CCC designs. For a rotateable CCC design with N_{fac} points in the factorial part, α is simply given by the expression $\alpha_{rotateable} = \sqrt[4]{N_{fac}}$ which will render the variance function equal ([hyper]spherical) in all direction of experimental space. Alternatively, α can be choosen to make the star and factorial part orthogonal, i.e. is a block variable labelling both parts with {0,1} will be maximal unconfounded with the RSM model parameters. This property can be important when using CC designs sequentially by starting with a factorial design first and augmenting the factorial part by star points later. The following R-code shows a typical analysis of a RSM design using functions from the library "rsm" and the library "lattice".

```
rm(list=ls())
library(lattice)
set.seed(123)
x             <- data.frame(rsm::ccd(3,alpha="orthogonal",randomize=F,
                    n0=c(3,0)))[,-c(1,2)]
colnames(x)   <- c(paste("x",1:3,sep=""),"Block")
x             <- x[order(runif(nrow(x))),]
rownames(x)   <- NULL
knitr::kable(
```

Table 4.11: CCC design with 3 center points and orthogonal blocking

	x1	x2	x3	Block
1	1.000	-1.000	1.000	1
2	0.000	1.477	0.000	2
3	0.000	0.000	1.477	2
4	-1.000	-1.000	-1.000	1
5	-1.000	1.000	-1.000	1
6	-1.477	0.000	0.000	2
7	0.000	0.000	0.000	1
8	-1.000	1.000	1.000	1
9	0.000	0.000	0.000	1
10	0.000	-1.477	0.000	2
11	1.477	0.000	0.000	2
12	1.000	-1.000	-1.000	1
13	1.000	1.000	-1.000	1
14	1.000	1.000	1.000	1
15	0.000	0.000	-1.477	2
16	-1.000	-1.000	1.000	1
17	0.000	0.000	0.000	1

```
  x, booktabs = TRUE,row.names=T, digits=3,align="c",
  caption = 'CCC design with 3 center points and
             orthogonal blocking')
```

```
x$y <- 20*(x$Block==2)  + 3*x$x1 + 5*x$x2 +
  6*x$x3 + 5*x$x1^2 + 8*x$x2^2 + 5*x$x3^2 + rnorm(nrow(x),0,1)
              # Block=0: factorial; Block=1:star
summary(res <- rsm::rsm(y~Block + SO(x1,x2,x3), data=x))
```

```
##
## Call:
## rsm(formula = y ~ Block + SO(x1, x2, x3), data = x)
##
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  0.28242    0.63342  0.4459 0.6713339
## Block2      19.89145    0.55681 35.7241 3.208e-08 ***
## x1           2.28752    0.31202  7.3314 0.0003291 ***
## x2           5.31461    0.31202 17.0330 2.619e-06 ***
## x3           5.94880    0.31202 19.0656 1.346e-06 ***
## x1:x2        0.31363    0.38789  0.8086 0.4496323
## x1:x3        0.09441    0.38789  0.2434 0.8158084
```

```
## x2:x3        0.52927    0.38789  1.3645 0.2213735
## x1^2         4.79514    0.38155 12.5675 1.553e-05 ***
## x2^2         8.29488    0.38155 21.7398 6.186e-07 ***
## x3^2         4.78696    0.38155 12.5460 1.569e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.9977, Adjusted R-squared:  0.9939
## F-statistic:   263 on 10 and 6 DF,  p-value: 4.098e-07
##
## Analysis of Variance Table
##
## Response: y
##                 Df  Sum Sq Mean Sq   F value    Pr(>F)
## Block            1 1536.13 1536.13 1276.2107 3.208e-08
## FO(x1, x2, x3)   3  851.44  283.81  235.7896 1.297e-06
## TWI(x1, x2, x3)  3    3.10    1.03    0.8583    0.5117
## PQ(x1, x2, x3)   3  775.20  258.40  214.6783 1.714e-06
## Residuals        6    7.22    1.20
## Lack of fit      4    4.53    1.13    0.8431    0.6060
## Pure error       2    2.69    1.34
##
## Stationary point of response surface:
##         x1         x2         x3
## -0.2228815 -0.2969119 -0.6027428
##
## Eigenanalysis:
## eigen() decomposition
## $values
## [1] 8.321983 4.814304 4.740696
##
## $vectors
##           [,1]        [,2]        [,3]
## x1 -0.04529810  0.79991654 -0.59839921
## x2 -0.99614083 -0.08124721 -0.03320141
## x3 -0.07517663  0.59458592  0.80050987
```

```
# RSM plot within rsm
#par(mfrow=c(1,2))
#persp(res,~x1+x2,at=list(x3=-1.477098),zlab="y",zlim=c(-15,60))
#persp(res,~x1+x2,at=list(x3=1.477098),zlab="y",zlim=c(-15,60))

# rsm plot trellis plot with lattice
x.pred <- expand.grid(x1 = seq( -1.477098,  1.477098, length=15),
                      x2 = seq(-1.477098,  1.477098, length=15),
                      x3  = seq(-1.477098,  1.477098, length=3),
```

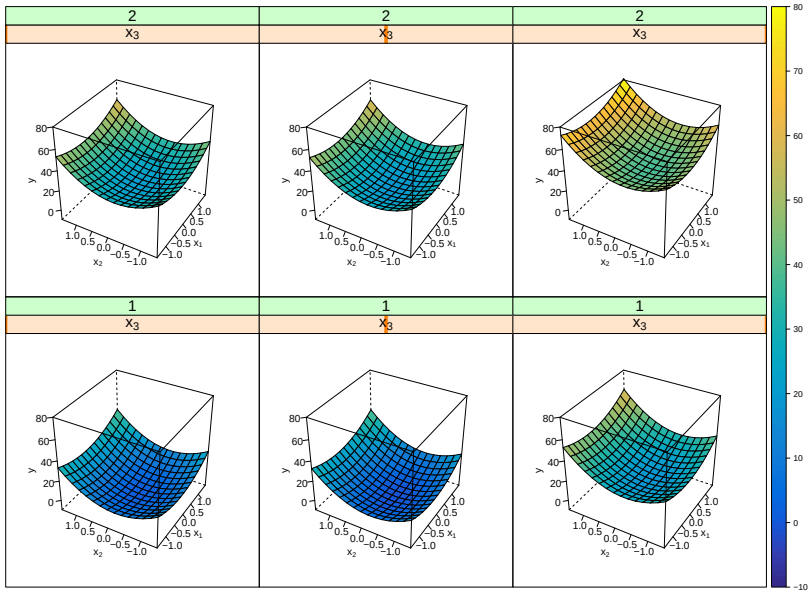

Figure 4.16: RSM trellis plot $\hat{y} = f(x_1, x_2, x_3, Block)$ including block effect Block=1(labelling the factorial part) and Block=2(corresponding to the star part). The variable Block is on the outer y-axis (on two levels 1,2), x 3 on the outer x-axis, and x 1 and x 2 are on the inner x- and y-axes.

```
                      Block    = c(1,2)  )
x.pred$Block <- factor(x.pred$Block)
x.pred$y1 <- predict(res,x.pred)
           wireframe(y1 ~ x1*x2|x3*Block, data=x.pred,
                zlab=list(label="y",cex=1,rot=90),
                xlab=list(label=expression(x [1]),cex=1,rot=0),
                ylab=list(label=expression(x [2]),cex=1,rot=0),
                zlim=c(-10,80),
                drape=TRUE,
                at=lattice::do.breaks(c(-10,80),100),
                col.regions = pals::parula(100),
                par.strip.text=list(cex=1.5),
                strip=strip.custom(var.name=expression(x [3])),
                scales = list(arrows = FALSE,
                              x=c(cex=1),
                              y=c(cex=1),
                              z=c(cex=1)) ,
                screen = list(z = 60, x = -55, y=0))
```

In addition to the known OLS results, the rsm function reports stationary points (max, min or sattle point). The nature of the stationary point is indicated by the eigenvalues λ_i in the output section "eigenanalysis" with $\lambda_i > 0\ \forall i \rightarrow min; \lambda_i < 0\ \forall i \rightarrow max$; *else sattle*. In the present case all eigenvalues are strictly positive thus indicating a minimum that is reported at $x_1^* = -0.332;\ x_2^* = -0.361;\ x_3^* = -0.696$. Furthermore, the function rsm decomposes the response variance into five contributing terms, namely

1. Variance contribution (VC) of the block variable **Block**
2. VC arising from the linear terms (abreviated FO(...))
3. VC from bilinear interactions, TWI(...)
4. VC from square terms, PQ(...)
5. VC of the error term, the residuals, with the residual term further decomposed into
   - Lack of Fit
   - Pure Error

F-test results reveal no evidence of interactions or lack of fit and the replicates at the center are correctly predicted by the model within the replication (or pure) error. Lack of Fit, e.g., may result from systematic shifts between factorial and star points and can be accommodated by including a block variable as shown in the example above (rerunning the analysis with Block excluded leads to biased estimates indicated by p_{LoF}=0.00048). The topology of the response surface can be visualized by plotting the model predictions over a grid of points with conditional 3D plots (trellis plots) y~f(x1,x2)|x3,x4 with x3 and x4 on the outer axes.
Figure 4.16 depicts the model y=f(x1,x2,x3,Block) as a 5D-trellis plot with two inner axes **x1**,**x2** and two outer x-axes **x3** and **Block**. The trellis plot reveals a minimum close to the center (0,0,0) with the exact location of the minimum in the plot remaining somewhat vague. Here, rsm helps by reporting the numerical solution of the minimum, $x_1^* = -0.223;\ x_2^* = -0.297;\ x_3^* = -0.603$.
As the list of available CC design shows, see table 4.12, the design size N=f(K) grows large rapidly with dimension K, prohibiting the use of CC designs with K>4 in a normal lab (low-throughput) setup.

Box-Behnken designs (BBD) as a class of reduced 3^3 designs are a parsimonious alternative to CC designs for estimating quadratic models. Table 4.13 lists the available designs and figure 4.17 shows a Box-Behnken design in three dimensions. Comparision of table 4.13 with 4.12 shows that CC and Box-Behnken designs are complementary in terms of model redundancy, and economical RSM designs may be found with the selection criterion $\min(R_{BBD}, R_{CC})$.

4.7 Optimal designs

The designs discussed so far are classical designs, i.e. given complexity - linear, bilinear or quadratic - and dimensionality K the number of design runs and the

Table 4.12: CCC design size N for different dimensions K with three center replicates n 0 along with optimal α for orthogonality and rotability plus model redundancy

K	N	orthogonal	rotateable	R
2	11	1.069	1.414	1.83
3	17	1.477	1.682	1.70
4	27	1.835	2.000	1.80
5	45	2.138	2.378	2.14
6	79	2.394	2.828	2.82
7	145	2.615	3.364	4.03

Table 4.13: List of available Box-Behnken designs with three center points and model redundancy under quadratic model assumption

K	N	R
3	15	1.50
4	33	2.20
5	46	2.19
6	51	1.82
7	59	1.64

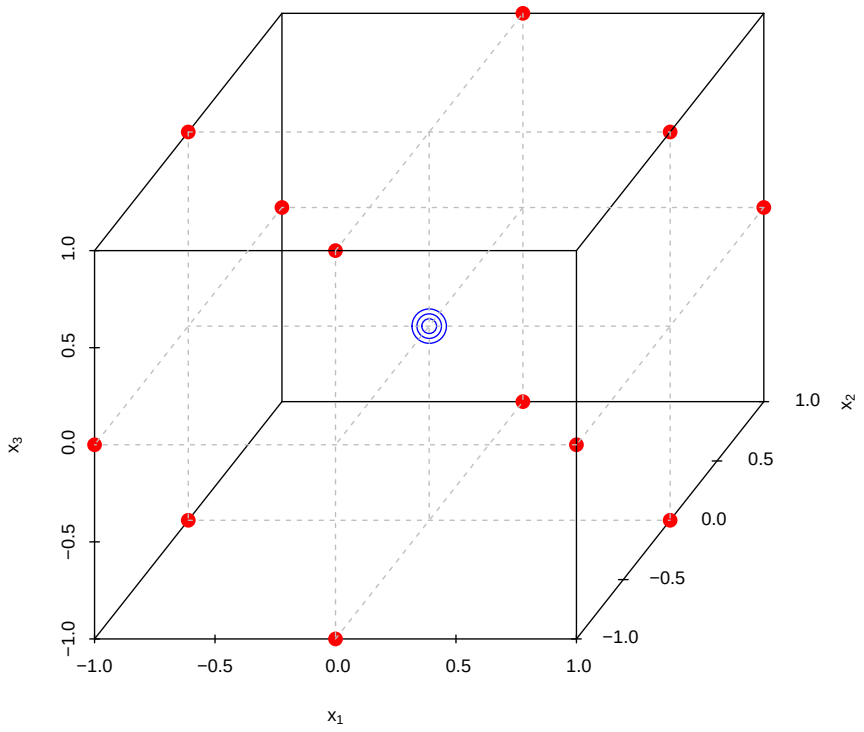

Figure 4.17: Three dimensional coded Box-Behnken design with three center points

parametric model are fixed and set by the design class. A fractional factorial resolution V design, as an example, will always support **all** bilinear models terms with a fixed number of runs irrespective of whether all interactions are physically possible.
Different from the classical designs, optimal or computer algorithmic designs define a more flexible class of designs able to handle almost all sort of experimental constraints, namely

1. Any linear model can be specified by the user, for instance a three factor design x1,x2,x3 with only one possible interaction can be specified in R-notation by ~ x1+x2+x3+x1:x2.
2. Any number of design runs with R(K)>1 can be specified allowing flexible and parsimonious experimentation.
3. Complex constraints can be handled by excluding infeasible runs and regions from the candidate set prior to design generation.
4. Existing data can be augmented to render a parametric model estimable in a process similar to augmenting a factorial design by star points. However, optimal augmentation does nor require the data to be of factorial and fractional factorial nature. In principle, any non-singular data set, historical or from DoE, can be augmented.

Optimal designs are created from a set of potential candidate points, typically 3^K full factorial designs (or higher). When there are constraints, they are applied to the candidate set (often linear constraints such as $LB_i \leq \sum_i a_i \cdot x_i \leq UB_i$) and, given model and number of desired runs, the optimal design is obtained by optimally selecting designs points from the candidate set using an alphabetic optimization objective. There are many scalar objective functions available, such as A-, G- or D-optimality to name but a few with the latter, D-optimality, being the most common seeking to minimize the determinant (indicated by |.|) of the inverse information matrix, i.e.

$$\min_X |(\boldsymbol{X^T X})^{-1}|$$

A visual overview of the process of generating optimal (constrained and unconstrained) designs from a list of candidate points is depicted in figure 4.18.

It should be noted that optimal designs are not necessarily superiour to classical design as the following example will show. The example code juxtaposes a classical CCF design of size N=17 with a D-optimal RSM design of the same size, see figure 4.19, and reports some design criteria, namely

1. **CCF design**: $\kappa = 3.03$, D-optimality=0.413
2. **D-optimal design**: $\kappa = 4.03$, D-optimality=0.458

The CCF design is better than the D-optimal design both in terms of κ and D-criteria. Figure 4.19 reveals the poor balance of the D-optimal design compared to the CCF design with the consequence being that the accuracy of the design becomes dependent on the direction in experimental space. Generating optimal

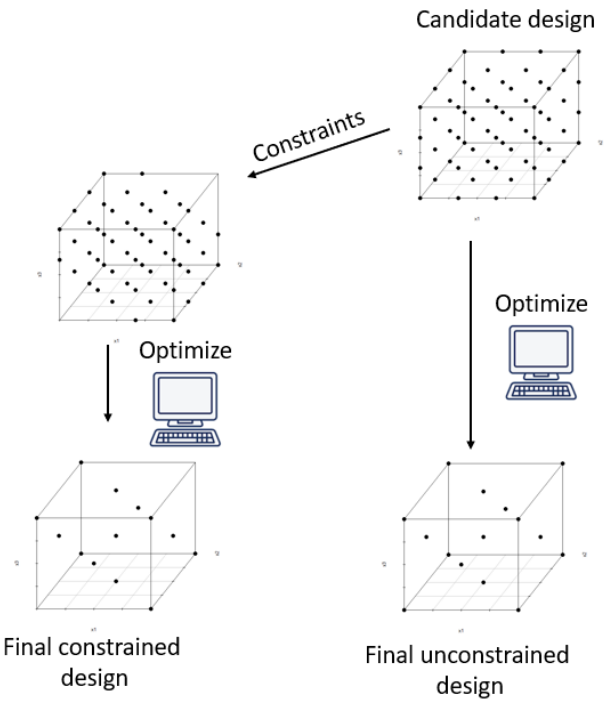

Figure 4.18: Process of generating an optimal designs from a set of candidate points

designs is computationally difficult as there are many local minima of the objective function and, as the example showed, there is no guarantee that the optimizer will find the global optimum (the CCF design in the example).

```
rm(list=ls())
candidate <- expand.grid(x1=c(-1,0,1),  # 3^3 candidate design
                         x2=c(-1,0,1),
                         x3=c(-1,0,1) )
set.seed(1234)
doe <- AlgDesign::optFederov(~ (x1+x2+x3)^2+
                               I(x1^2) + I(x2^2) + I(x3^2),
                     data=candidate,center=T,
                     criterion="D", nTrials=17)$design
                     # d-optimal design with 17 runs
kappa(model.matrix(~ (x1+x2+x3)^2+ I(x1^2) +
            I(x2^2) + I(x3^2), data=doe)) # kappa doe
```

```
## [1] 4.02569
```

```
kappa(model.matrix(~ (x1+x2+x3)^2+ I(x1^2)
          + I(x2^2) + I(x3^2),
          data=rsm::ccd(3,  n0=c(3,0),
          randomize=F,alpha=1)[,3:5])) # kappa CCF design
```

```
## [1] 3.025454
```

```
AlgDesign::eval.design(~ (x1+x2+x3)^2+ I(x1^2)
          + I(x2^2) + I(x3^2),doe) # D=0.4583859
```

```
## $determinant
## [1] 0.4583859
##
## $A
## [1] 3.751222
##
## $diagonality
## [1] 0.78
##
## $gmean.variances
## [1] 2.436736
```

```
AlgDesign::eval.design(~ (x1+x2+x3)^2+
           I(x1^2) + I(x2^2) + I(x3^2),
            rsm::ccd(3,  n0=c(3,0), randomize=F
            ,alpha=1)[,3:5]) # D=0.4129647
```

```
## $determinant
## [1] 0.4129647
```

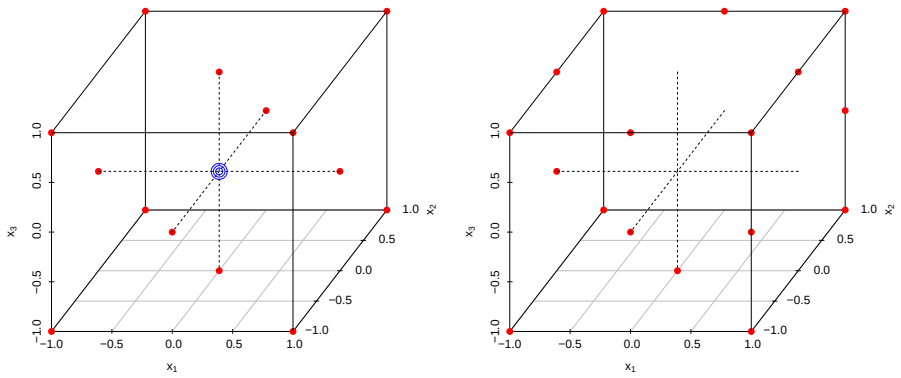

Figure 4.19: CCF design with three center points and N=17 (left panel) compared to D-optimal RSM design with 17 runs (right panel)

```
##
## $A
## [1] 3.362289
##
## $diagonality
## [1] 0.778
##
## $gmean.variances
## [1] 2.840631
```

The difference in model accuracy can be visualized with the variance dispersion graph (with the R function Vdgraph::Vdgraph(...)) shown in figure 4.20, depicting the average model variance as a function of the coded design radius. Direction dependence is indicated by the upper and lower limits of the variance (note that for a rotateable CCC design max, min and average variance coincide).

Model accuracy as a function of the radius is better for the CCF design compared with the D-optimal design as expected. However, this "lack of optimality" of optimal designs is often outweighted by their higher flexibility and there are practitioner exclusively working with these designs, something not recommended here. Whenever possible, classical designs are to be preferred over optimal designs for their better statistical properties.
The following example is a typical application of an optimal design and borrowed from chapter 7. In an optimization project aiming at maximizing catalytic performance **ton.dmm** over a set of 7 influential factors, a subspace comprising 9 runs was identify from a historical record with the condition ton.dmm>400.

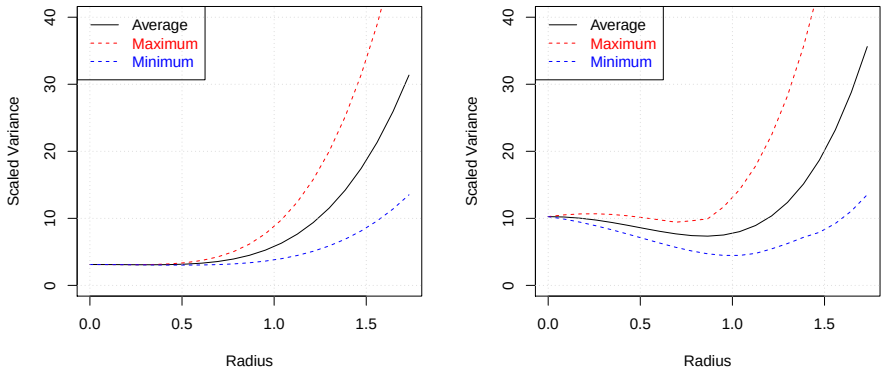

Figure 4.20: Variance dispersion graphs for the K=3, N=17, CCF design (left panel) compared with the K=3, N=17 D-optimal design (right panel)

In this subset **T**, **p.h2**, **p.co2** and **t.h** were found constant at T=90, p.h2=90, p.co2=20 and t.h=18 thereby suggesting to conditionally optimze **m.kat**, **v.ml** and **m.add** first while keeping the remaining 4 variables constant at T=90, p.h2=90, p.co2=20 and t.h=18 for the time being. A full factorial 3x3x3 (3^3) design was created as a set of candidate points and the nine runs from the historical data were augmented by another set of six runs optimally selected from the candidate set so as to render a full second order design of the factors **m.kat**, **v.ml**, **m.add** estimable. The following R-code does the augmentation and plots the augmented data set.

```
rm(list=ls())
m.kat <- scan(text=" 0.18750  0.3750  0.750  0.3750  0.375
              0.375  0.3750  0.375  0.3750")
v.ml  <- scan(text="0.50000  0.5000  0.500  0.5000  0.500
              0.500  0.5000  0.500  0.2500")
m.add <- scan(text=" 0.78125  1.5625  3.125  0.9375  1.250
              1.875  2.1875  2.500  1.5625")
T     <- scan(text="90.00000 90.0000 90.000 90.0000 90.000
              90.000 90.0000 90.000 90.0000")
p.h2  <- scan(text="90.00000 90.0000 90.000 90.0000 90.000
              90.000 90.0000 90.000 90.0000")
p.co2 <- scan(text="20.00000 20.0000 20.000 20.0000 20.000
              20.000 20.0000 20.000 20.0000")
t.h   <- scan(text="18.00000 18.0000 18.000 18.0000 18.000
              18.000 18.0000 18.000 18.0000")
```

```
x.promising <- data.frame(m.kat, v.ml, m.add, T,p.h2, p.co2, t.h)

sapply(x.promising,range) # report ranges
```

```
##       m.kat v.ml   m.add  T p.h2 p.co2 t.h
## [1,] 0.1875 0.25 0.78125 90   90    20  18
## [2,] 0.7500 0.50 3.12500 90   90    20  18
```

```
round(cor(x.promising[,1:3]),2) # ...and Pearson r
```

```
##       m.kat v.ml m.add
## m.kat  1.00 0.05  0.79
## v.ml   0.05 1.00  0.09
## m.add  0.79 0.09  1.00
```

```
candidate <- expand.grid(m.kat=seq(min(x.promising$m.kat),
                         max(x.promising$m.kat),length=3),
                         v.ml=seq(min(x.promising$v.ml),
                         max(x.promising$v.ml) ,length=3),
                         m.add=seq(min(x.promising$m.add),
                         max(x.promising$m.add),length=3) )
candidate  <- rbind(x.promising[,1:3],candidate)
              # rowbind x.promising and candidate set
kappa(model.matrix(~ (m.kat + v.ml + m.add)^2 +
                     I(m.kat^2) + I(v.ml^2) + I(m.add^2) ,
                   data=data.frame(scale(x.promising)   )   ) )
```

```
## [1] 3.863113e+17
```

```
set.seed(12345)
doe.cat <- AlgDesign::optFederov(~ (m.kat + v.ml + m.add)^2+
                 I(m.kat^2) + I(v.ml^2) + I(m.add^2),
                 data=candidate,center=T,augment=T,rows=(1:9),
                 criterion="D", nTrials=15)$design
                              # d-optimal cat design
# comprising 9 historical
# and 6 additional candidates from candidate set
rownames(doe.cat )   <- NULL
kappa(model.matrix(~ (m.kat + v.ml + m.add)^2 +
                     I(m.kat^2) + I(v.ml^2) + I(m.add^2) ,
                   data=data.frame(scale(doe.cat)   )   ) )
```

```
## [1] 8.519979
```

```
round(cor(doe.cat),2)
```

```
##       m.kat  v.ml m.add
## m.kat  1.00 -0.20  0.07
```

```
## v.ml  -0.20  1.00 -0.27
## m.add  0.07 -0.27  1.00
```

```
s3d <- scatterplot3d::scatterplot3d(doe.cat, type="p",
      highlight.3d=F,tick.marks=T,xlab="m.kat",
      ylab="v.ml",zlab="m.add",
      angle=55, scale.y=1, pch=16 ,
      color=c(rep("red",9), rep("blue",6)) ,
      cex.symbol=1.5, cex.lab = 0.8 )
```

Note how the comparatively large correlation between m.kat and m.add ($r_{m.add,m.kat}$=0.79) prior to augmentation is counterbalanced by the augmentation trials into $r_{m.add,m.kat}$=0.07. All effects up to 2^{nd} order, which were formerly non-estimable in the historical data, are now estimable with the augmented data ($\kappa = 8.5$). Figure 4.21 shows the historical core design (labelled red) along with the augmentation trials (labelled blue).

4.7.1 Example on the sequential use of RSM designs

CC and optimal designs both offer the possibility to build up models sequentially by starting with designs of low complexity which will then be gradually augmented to render more complex models estimable if indications of higher order effects arise in the course of the project. The data of the following example is taken from (George E.P. Box, Norman R. Draper 1987) pp. 306-322, and the aim of the project was finding conditions maximizing the elasticity of a polymer blend[15]. Three influential factors were identified by the practitioners with the following initial ranges

1. Concentration of the first component conc1 [%] = {15,21}
2. Concentration of the second component conc2 [%] = {2.3,3.1}
3. Temperature Temp [° C] = {135,155}

A 2^3 full factorial design without center points was set-up straddling the "so far best conditions" derived from previous experiments. There are many ways of creating a 2^3 design in R, here the library **rsm** is used as the package supports sequential buildup of CC designs from factorial designs. The designs are specified in coded variables, here abbreviated x1,x2,x3, from which the uncoded variables were obtained by the linear transformation $var_{coded} = \frac{var_{uncoded} - center}{midrange}$.

```
rm(list=ls())
set.seed(12345)
fac <-rsm::cube(~x1+x2+x3,n0=0, randomize=TRUE,
coding=list(x1~ (conc1-18)/3 ,x2~(conc2-2.7)/0.4,
```

[15] "The objective of this investigation was to explore reaction conditions yielding close to maximal elasticity for a certain polymer. At this stage of the enquiry, the variables of importance had been narrowed down to three. ...." (George E.P. Box, Norman R. Draper 1987) pp. 306-309

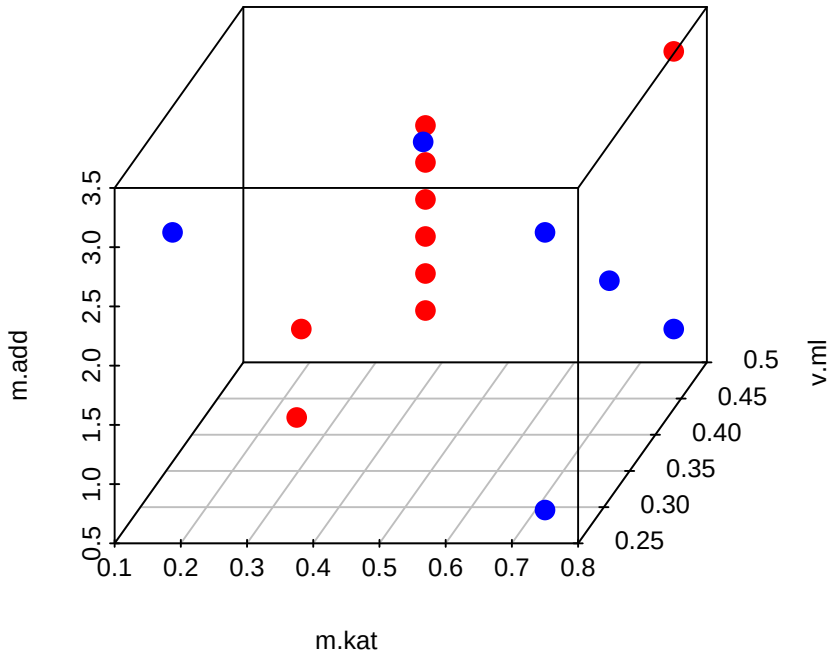

Figure 4.21: Historical trials *m.kat*, *v.ml*, *m.add* (red) augmented by six augmentation trials (blue)

```
            x3~(Temp-145)/10))
fac
```

```
##   run.order std.order conc1 conc2 Temp
## 1         1         6    21   2.3  155
## 2         2         5    15   2.3  155
## 3         3         7    15   3.1  155
## 4         4         8    21   3.1  155
## 5         5         3    15   3.1  135
## 6         6         1    15   2.3  135
## 7         7         2    21   2.3  135
## 8         8         4    21   3.1  135
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ (conc1 - 18)/3
## x2 ~ (conc2 - 2.7)/0.4
## x3 ~ (Temp - 145)/10
```

The data is internally stored in coded units, however uncoded variables are printed to be performed in the laboratory in randomized order. The factorial results were analysed with the rsm() function and visualized with a trellis response surface plot.

```
rm(list=ls())
conc1 <- scan(text="15  21  15  21  15  21  15  21")
conc2 <- scan(text="2.3 2.3 3.1 3.1 2.3 2.3 3.1 3.1")
Temp <- scan(text="135  135 135 135 155 155 155 155")
y    <- scan(text="25.74    48.98   42.78   35.94   41.5
             50.1   46.06   27.7")
x <- data.frame(conc1,conc2,Temp,y)
summary(res <- rsm::rsm(y~FO(conc1,conc2,Temp)+
                          TWI(conc1,conc2,Temp) , data=x))
```

```
##
## Call:
## rsm(formula = y ~ FO(conc1, conc2, Temp) + TWI(conc1, conc2,
##     Temp), data = x)
##
##               Estimate Std. Error  t value Pr(>|t|)
## (Intercept) -815.51125   53.87307 -15.1376  0.04199 *
## conc1         32.12417    2.08330  15.4199  0.04123 *
## conc2        201.58750   15.33108  13.1489  0.04832 *
## Temp           3.95375    0.35437  11.1571  0.05691 .
## conc1:conc2   -5.94167    0.32500 -18.2821  0.03479 *
## conc1:Temp    -0.10900    0.01300  -8.3846  0.07557 .
## conc2:Temp    -0.68250    0.09750  -7.0000  0.09033 .
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.998,  Adjusted R-squared:  0.9858
## F-statistic: 82.06 on 6 and 1 DF,  p-value: 0.0843
##
## Analysis of Variance Table
##
## Response: y
##                          Df Sum Sq Mean Sq F value Pr(>F)
## FO(conc1, conc2, Temp)    3  47.22  15.738  12.934 0.2010
## TWI(conc1, conc2, Temp)   3 551.86 183.954 151.178 0.0597
## Residuals                 1   1.22   1.217
## Lack of fit               1   1.22   1.217
## Pure error                0   0.00
##
## Stationary point of response surface:
##      conc1      conc2       Temp
##  18.173752   2.890566 137.150362
##
## Eigenanalysis:
## eigen() decomposition
## $values
## [1]  2.98466894  0.01235354 -2.99702248
##
## $vectors
##              [,1]        [,2]        [,3]
## conc1  0.70320084 -0.11402556 -0.70178825
## conc2 -0.70772462 -0.01774831 -0.70626543
## Temp   0.06807676  0.99331927 -0.09317933
```

```
x.pred <- expand.grid(conc1 = seq( min(x$conc1),
                                   max(x$conc1), length=15),
                      conc2 = seq( min(x$conc2),
                                   max(x$conc2), length=15),
                      Temp  = seq( min(x$Temp),
                                   max(x$Temp), length=3) )
x.pred$y <- predict(res,x.pred)
lattice::wireframe(y~conc1*conc2|Temp, data=x.pred,layout=c(3,1),
                   zlab=list(label="y",cex=1.3,rot=90),
                   xlab=list(label=expression(conc [1]),cex=1.3,rot=55),
                   ylab=list(label=expression(conc [2]),cex=1.3,rot=0),
                   drape=TRUE,
                   par.strip.text=list(cex=1.5),
                   col.regions = pals::parula(100),
                   strip=T,
                   scales = list(arrows = FALSE,
```

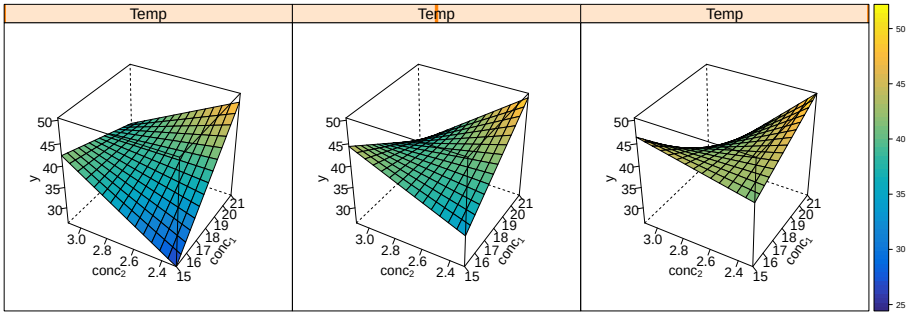

Figure 4.22: Response Surface trellis plot $\hat{y} = f(conc_1, conc_2, Temp)$ of factorial polymer elasticity $\hat{y}$

```
                                x=c(cex=1.3),
                                y=c(cex=1.3),
                                z=c(cex=1.3)) ,
                    screen = list(z = 60, x = -55, y=0))
```

The trellis plots, figure 4.22, as well as rsm-output show, that the system is dominated by interactions. In the rsm-output for example, interactions TWI() explain a large portion, SS=552, of the variance compared with minor linear contributions FO(), SS=47, and, as a consequence, the response surface trellis plot resembles a twisted plain. Given this nonlinear topology and lacking center point(!), it would be dangerous to take the suggested path of further ascend in figure 4.22 seriously, namely increase Temp and conc1 while decreasing conc2. It seems more reasonable to suspect that there are strong nonlinear effects underlying the system which the bilinear model (data) cannot appropriately describe.
This was also the conclusion of the researchers, and they decided to augment the data for rendering a full RSM model estimable. As all factors $x_{1,2,3}$ could be physically extended beyond the present upper and lower limits, the factorial data was subsequently augmented with six star points and two center points with $\alpha = 2$. In this investigation more than a week ellapsed between the first and second set of eight runs and some shifts between the two blocks cannot be excluded. Therefore, a numerical block indicator on two levels {0,1} was included in the analysis and indeed found significant (F=8.9; p=0.03).

```
rm(list=ls())
fac <-rsm::cube(~x1+x2+x3,n0=0, randomize=TRUE,
         coding=list(x1~ (conc1-18)/3 ,
                     x2~(conc2-2.7)/0.4, x3~(Temp-145)/10))
```

```
star <-  rsm::star(fac,n0=2, alpha=2)
ccd  <-  rsm::djoin(fac,star)
ccd
```

```
##    run.order std.order conc1 conc2 Temp Block
## 1          1         3    15   3.1  135     1
## 2          2         8    21   3.1  155     1
## 3          3         5    15   2.3  155     1
## 4          4         7    15   3.1  155     1
## 5          5         6    21   2.3  155     1
## 6          6         1    15   2.3  135     1
## 7          7         2    21   2.3  135     1
## 8          8         4    21   3.1  135     1
## 9          1         6    18   2.7  165     2
## 10         2         3    18   1.9  145     2
## 11         3         2    24   2.7  145     2
## 12         4         1    12   2.7  145     2
## 13         5         4    18   3.5  145     2
## 14         6         5    18   2.7  125     2
## 15         7         8    18   2.7  145     2
## 16         8         7    18   2.7  145     2
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ (conc1 - 18)/3
## x2 ~ (conc2 - 2.7)/0.4
## x3 ~ (Temp - 145)/10
```

The augmentation trials in Block=2 were realized in the the lab and the results were analysed with the R-code

```
rm(list=ls())
conc1 <- scan(text="15  21  15  21  15  21  15  21
                    18  18  12  24  18  18  18  18")
conc2 <- scan(text="2.3 2.3 3.1 3.1 2.3 2.3 3.1 3.1
                    2.7 2.7 2.7 2.7 1.9 3.5 2.7 2.7")
Temp <- scan(text="135  135 135 135 155 155 155 155
                   145  145 145 145 145 145 125 165")
y <- scan(text="25.74   48.98   42.78   35.94   41.5    50.1
                46.06   27.7    57.52   59.68   35.5
                44.18   38.58   28.46   33.5    42.02")
x <- data.frame(conc1,conc2,Temp,y)
x$block <- c(rep(0,8),rep(1,8))
summary(res <- rsm::rsm(y~block + SO(conc1,conc2,Temp) , data=x))
```

```
##
## Call:
## rsm(formula = y ~ block + SO(conc1, conc2, Temp), data = x)
```

```
##
##                Estimate  Std. Error  t value  Pr(>|t|)
## (Intercept) -2.3552e+03  1.3870e+02 -16.9810 1.296e-05 ***
## block        2.5800e+00  8.6401e-01   2.9861 0.0305856 *
## conc1        5.1108e+01  3.6900e+00  13.8502 3.524e-05 ***
## conc2        4.1220e+02  2.8075e+01  14.6820 2.649e-05 ***
## Temp         1.9095e+01  1.3696e+00  13.9416 3.413e-05 ***
## conc1:conc2 -5.9417e+00  5.0913e-01 -11.6703 8.115e-05 ***
## conc1:Temp  -1.0900e-01  2.0365e-02  -5.3523 0.0030584 **
## conc2:Temp  -6.8250e-01  1.5274e-01  -4.4685 0.0065887 **
## conc1^2     -5.2111e-01  4.8001e-02 -10.8563 0.0001151 ***
## conc2^2     -3.9188e+01  2.7000e+00 -14.5137 2.803e-05 ***
## Temp^2      -5.2100e-02  4.3201e-03 -12.0600 6.920e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:   0.99,  Adjusted R-squared:   0.97
## F-statistic:  49.5 on 10 and 5 DF,  p-value: 0.0002279
##
## Analysis of Variance Table
##
## Response: y
##                          Df Sum Sq Mean Sq F value    Pr(>F)
## block                     1  26.63  26.626  8.9166 0.0305856
## FO(conc1, conc2, Temp)    3 161.01  53.669 17.9732 0.0041420
## TWI(conc1, conc2, Temp)   3 551.86 183.954 61.6038 0.0002273
## PQ(conc1, conc2, Temp)    3 738.75 246.249 82.4658 0.0001117
## Residuals                 5  14.93   2.986
## Lack of fit               4  12.60   3.149  1.3501 0.5620374
## Pure error                1   2.33   2.333
##
## Stationary point of response surface:
##      conc1      conc2       Temp
##  19.381044   2.514217 146.509245
##
## Eigenanalysis:
## eigen() decomposition
## $values
## [1]  -0.04585026  -0.29742296 -39.41743789
##
## $vectors
##                [,1]        [,2]         [,3]
## conc1 -1.137565e-01  0.99058480 -0.076165710
## conc2 -2.767632e-05 -0.07666652 -0.997056791
## Temp   9.935087e-01  0.11341956 -0.008748729
```

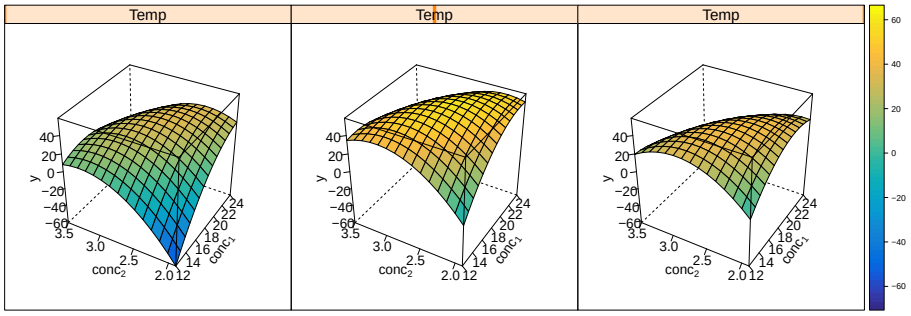

Figure 4.23: Response surface trellis plot $\hat{y} = f(conc_1, conc_2, Temp)$ of the final CCC design of the polymer data

```
x.pred <- expand.grid(conc1 = seq( min(x$conc1),  max(x$conc1),
                                   length=15    ),
                      conc2 = seq( min(x$conc2),  max(x$conc2),
                                   length=15    ),
                      Temp  = seq( min(x$Temp),  max(x$Temp),
                                   length=3    ) ,
                            block=0.5)
x.pred$y <- predict(res,x.pred)
lattice::wireframe(y~conc1*conc2|Temp, data=x.pred,layout=c(3,1),
                   zlab=list(label="y",cex=1.3,rot=90),
                   xlab=list(label=expression(conc [1]),cex=1.3,rot=55),
                   ylab=list(label=expression(conc [2]),cex=1.3,rot=0),
                   drape=TRUE,
                   col.regions = pals::parula(100),
                   par.strip.text=list(cex=1.5),
                   strip=T,
                   scales = list(arrows = FALSE,
                                 x=c(cex=1.3),
                                 y=c(cex=1.3),
                                 z=c(cex=1.3)) ,
                   screen = list(z = 60, x = -55, y=0))
```

Response Surface Trellis plot, figure 4.23, and rsm-output clearly reveal a degenerated maximum at conc1=19.4 ; conc2=2.5 ; Temp=146.5, close to the center of the design space. The degree of degeneration (flatness) is indicated by a small first eigenvalue suggesting a flat (degenerated) maximum along the first eigenvector. Almost perpendicular to that direction the maximum drops sharply with the direction indicated by the third eigenvector. Figure 4.24 shows

a response surface plot of this maximum with Temp kept at the optimal value and block=0.5, y=f(conc1,conc2)|(temp* = 146.5;block=0.5).

```
##
## Call:
## rsm(formula = y ~ block + SO(conc1, conc2, Temp), data = x)
##
##                Estimate  Std. Error  t value  Pr(>|t|)
## (Intercept) -2.3552e+03  1.3870e+02 -16.9810 1.296e-05 ***
## block        2.5800e+00  8.6401e-01   2.9861 0.0305856 *
## conc1        5.1108e+01  3.6900e+00  13.8502 3.524e-05 ***
## conc2        4.1220e+02  2.8075e+01  14.6820 2.649e-05 ***
## Temp         1.9095e+01  1.3696e+00  13.9416 3.413e-05 ***
## conc1:conc2 -5.9417e+00  5.0913e-01 -11.6703 8.115e-05 ***
## conc1:Temp  -1.0900e-01  2.0365e-02  -5.3523 0.0030584 **
## conc2:Temp  -6.8250e-01  1.5274e-01  -4.4685 0.0065887 **
## conc1^2     -5.2111e-01  4.8001e-02 -10.8563 0.0001151 ***
## conc2^2     -3.9188e+01  2.7000e+00 -14.5137 2.803e-05 ***
## Temp^2      -5.2100e-02  4.3201e-03 -12.0600 6.920e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:   0.99,  Adjusted R-squared:   0.97
## F-statistic:  49.5 on 10 and 5 DF,  p-value: 0.0002279
##
## Analysis of Variance Table
##
## Response: y
##                          Df Sum Sq Mean Sq F value    Pr(>F)
## block                     1  26.63  26.626  8.9166 0.0305856
## FO(conc1, conc2, Temp)    3 161.01  53.669 17.9732 0.0041420
## TWI(conc1, conc2, Temp)   3 551.86 183.954 61.6038 0.0002273
## PQ(conc1, conc2, Temp)    3 738.75 246.249 82.4658 0.0001117
## Residuals                 5  14.93   2.986
## Lack of fit               4  12.60   3.149  1.3501 0.5620374
## Pure error                1   2.33   2.333
##
## Stationary point of response surface:
##      conc1      conc2       Temp
##  19.381044   2.514217 146.509245
##
## Eigenanalysis:
## eigen() decomposition
## $values
## [1]  -0.04585026  -0.29742296 -39.41743789
##
```

```
## $vectors
##                 [,1]        [,2]         [,3]
## conc1 -1.137565e-01  0.99058480 -0.076165710
## conc2 -2.767632e-05 -0.07666652 -0.997056791
## Temp   9.935087e-01  0.11341956 -0.008748729
```

According to figure 4.24 a decrease in conc1 can be compensated by an increase of conc2 without significantly loosing optimality. However, jointly increasing or decreasing conc1 and conc2 will result in a substantial loss in terms of the objective function $\mathbf{y}$.
Informations about the sensitivity of an optimum in factor space can be very helpful for running a process safe and stable at optimal process conditions.

4.8 Design blocking

Often an experimental design cannot be realized at one stretch and must be divided into temporally disjoint parts. In the sequential use of Central Composite designs for instance, the factorial part is nescessarily realized first and, depending on the outcome, later augmented by star points. Such time gaps can become a source of bias which can, thanks to factorial and star part being orthogonal, controlled by blocking as has been shown in the example, figure 4.16.
Other reasons naturally calling for blocking are experimental heterogeneities introduced by different lab equipment, different staff or different raw material batches.
Many classical designs such as (fractional) factorial and Central Composite designs have the capability of blocking, and more information on how to create blocked factorial and fractional factorial designs, e.g., in R can be found with the function FrF2::blockpick().
Blocking can also be achieved algorithmically with the package *AlgDesign* and the function optBlock() therein. The following example code demonstrates this functionality by splitting a classical Box-Behnken designs with 16 runs into four blocks of size four. In the simulation with the model $y = 10B_2 + 15B_3 + 20B_4 + 3x_1 + 3x_2 + 3x_3 + 5x_1 * x_2 + 5x_1^2 - 5x_2^2 + 8x_3^2 + \epsilon;\ \epsilon \sim N(0,1)$ the OLS estimates agree well with the true model within standard error.

```
rm(list=ls())
set.seed(1234)
x <- as.data.frame(rsm::bbd(3,randomize=F))[,-c(1,2)]
x <- x[order(runif(nrow(x))),]
rownames(x)  <- NULL
colnames(x) <- paste("x",1:3,sep="")
res <- AlgDesign::optBlock(~ quad(.), x,
      blocksize=rep(4,4), criterion="Dpc", nRepeats=1000)
x       <- res$design
x$Block <- factor(rep(paste("B",1:4,sep=""),each=4))
```

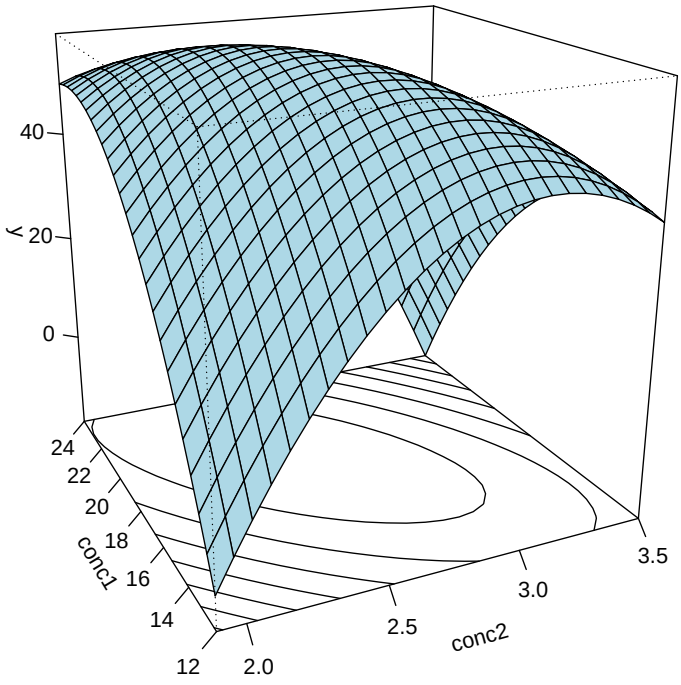

Figure 4.24: Conditional response surface plot $\hat{y} = f(conc_1, conc_1, Temp)|(Temp = 146.5; block = 0.5)$ of the CCC polymer data

```
kappa(model.matrix(~quad(x1,x2,x3), data=x))
```

```
## [1] 2.272431
```

```
kappa(model.matrix(~Block + quad(x1,x2,x3), data=x))
```

```
## [1] 6.004096
```

```
x$y <- 10*(x$Block=="B2") + 15*(x$Block=="B3") +
       20*(x$Block=="B4") +  3*x$x1 +
        3*x$x2 + 3*x$x3 + 5*x$x1*x$x2 +
        5*x$x1^2 -5*x$x2^2 +8*x$x3^2 +
        rnorm(nrow(x),0,1)
# OLS estimates agree very well with the true parameters
summary(lm(y~Block + (x1+x2+x3)^2 + I(x1^2)+
             I(x2^2)+ I(x3^2), data=x))
```

```
##
## Call:
## lm(formula = y ~ Block + (x1 + x2 + x3)^2 + I(x1^2) + I(x2^2) +
##     I(x3^2), data = x)
##
## Residuals:
##        7       10       15       16        2       11       13       14        1
##  0.21677  0.02782  0.24459 -0.48918 -0.02782 -0.09726 -0.06945  0.19453  0.09726  0
##        3        5        6        9
## -0.24459 -0.13890  0.06945  0.31404
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.65031    0.42418  -1.533 0.222789
## BlockB2     11.15129    0.55539  20.078 0.000270 ***
## BlockB3     16.01074    0.50700  31.580 6.98e-05 ***
## BlockB4     20.55500    0.50700  40.543 3.30e-05 ***
## x1           2.86765    0.24049  11.924 0.001269 **
## x2           3.73879    0.24049  15.547 0.000578 ***
## x3           3.29287    0.19636  16.770 0.000462 ***
## I(x1^2)      4.63003    0.27769  16.673 0.000470 ***
## I(x2^2)     -4.74465    0.27769 -17.086 0.000437 ***
## I(x3^2)      8.30608    0.27769  29.911 8.21e-05 ***
## x1:x2        4.70504    0.32065  14.673 0.000687 ***
## x1:x3       -0.12349    0.34010  -0.363 0.740600
## x2:x3       -0.04465    0.34010  -0.131 0.903859
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5554 on 3 degrees of freedom
```

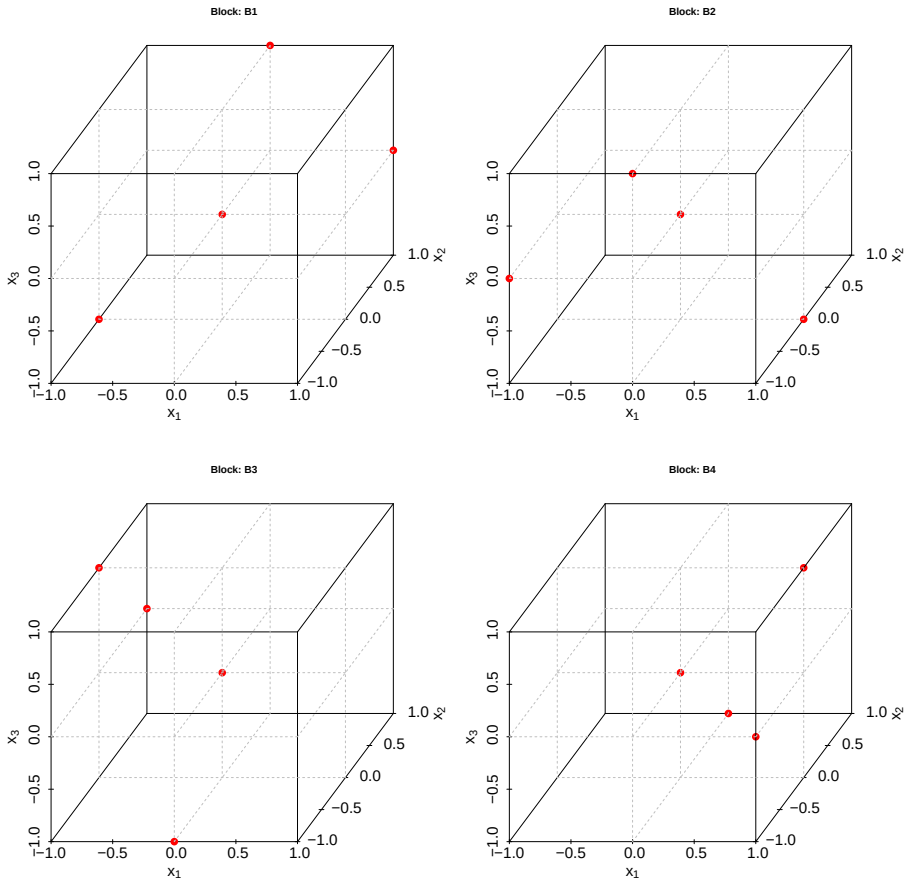

Figure 4.25: Three dimensional Box-Behnken design in four orthogonal blocks

```
## Multiple R-squared:  0.9993, Adjusted R-squared:  0.9967
## F-statistic: 375.5 on 12 and 3 DF,  p-value: 0.0002009
```

4.9 Variance component analysis

All models and designs discussed to this point aim at estimating the true, however unknown effects β_i in a linear parametric model of the general form $y = \sum_i \beta_i \cdot g_i(x_1, x_2, ...x_K) + \epsilon$ with $\mathrm{g_i}()$ belonging to a linear, bilinear or quadratic class of transformations. Because the true parameters are estimated from a finite sample, the estimates $\hat{\beta}_i$ become itself uncertain with good designs minimizing this uncertainty. The parameters β_i are also called "fixed effects" and the residuals are the so-called "random effects", and OLS estimates both,

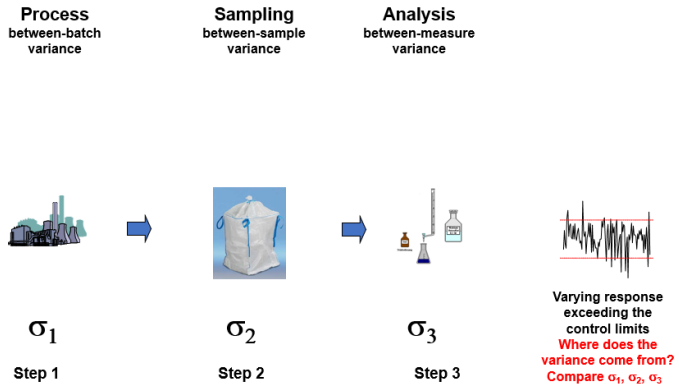

Figure 4.26: Variance components of a three step process with variance component $\sigma_1, \sigma_2, \sigma_3$

random and fixed effects, with least squares. In this process the residuals, the random effects, are only a by-product, required for judging the reliability of the fixed estimates $\hat{\beta}_i$. OLS is a member of the so called mixed models. These are models comprising fixed and random effects, although, for historical reasons, OLS is usually subsumed as a linear least squares method. It is not the place here to discuss mixed models, rather the special method of Variance Component Analysis (VCA) will be discussed, as VCA has very wide use in the applied sciences for detecting the sources of variance in process chains.
The elements of VCA can be understood from figure 4.26 which shows a three step process consisting of process, sampling and (analytical) analysis. Each steps is a source of variance contributing to the final variance $\sigma^2_{total} = \sum_{i=1}^{3} \sigma^2_i$. Now, when a process is running out-of-spec with σ^2_{total} too large[16], VCA can be exceedingly helpful in identifying the major sources of variance in the process chain, that is the step presumably causing the problems.

We shall introduce the key ideas of VCA with a simulated examples, thereby making the data generating process fully transparent.
The random effect model generating the data y_{ijk}[17] may be written

$$y_{ijk} = \beta_0 + \alpha_i + \beta_{ij} + \epsilon_{ijk}$$

$$\alpha_i \sim N(0, \sigma_1)$$

$$\beta_{ij} \sim N(0, \sigma_2)$$

$$\epsilon_{ijk} \sim N(0, \sigma_3)$$

[16]Often and unjustified, the staff of the last stage is taken responsible for the process variability. Here, VCA can help to settle the debate by quantifying the individual variance components of the different process steps

[17]The indices i,j,k refer to batch (process), sample and analysis in figure 4.26, so, e.g., y_{213} denotes the third analysis of the first sample from the second batch.

The following R-code generates a sample of (N_{batch}=3 x N_{sample}=3 x $N_{analysis}$~=3)=27 observations with variance components $\sigma_{batch} = 15; \sigma_{sample} = 5; \sigma_{analysis} = 1$. The result is a **nested random effect design** with the factor levels of *sample* nested within *batch* and *analysis* nested within *sample*. The model hierarchy is reflected by the three loops generating the data with the outermost loop, *batch*, representing the highest and the innermost loop, *analysis*, representing the lowest hierarchical level. The simulated data set is listed in table 4.14.
Figure 4.27 shows the model hierarchy of a 2^3 design.

```
rm(list=ls())
set.seed(123)
sigma1 <- 15
sigma2 <- 5
sigma3 <- 1
x <- NULL
for (batch in (1:3)) {
  alpha <- rnorm(1,0,sigma1)
  for (sample in (1:3)) {
    beta <- rnorm(1,0,sigma2)
    for (analysis in (1:3)) {
      epsilon <- rnorm(1,0,sigma3)
     x <- rbind(x,data.frame(batch,sample,
                analysis,y=alpha+beta+epsilon))
    }
  }
}
x  <- data.frame(obs=1:nrow(x),x)
knitr::kable(
  x, booktabs = TRUE,row.names=F,digits=2,align="c",
  caption = 'Nested random effect design on $3^3$ levels')
```

The simulated example from above becomes in reality a design problem consisting of three random sampling steps, namely

1. From the batch process select three batches at random and record the batch ID, i=1,2,3.
2. From each batch i=1,2,3 take three samples j=1,2,3 at random and record the ID i,j.
3. Analyse each of the nine samples from above three times k=1,2,3 and record the obtained values y_{ijk}.

The experimental data thus obtained will be similar to table 4.14, and the objective is now to estimate the variance components σ_1,σ_2 and σ_3 from the data. In R there are many packages and functions for doing variance component analysis. Skipping any theory here[18], the package lme4 and VCA will be used

[18]The basic idea for solving parametric models in statistics is finding the parameters under

Table 4.14: Nested random effect design on 3^3 levels

obs	batch	sample	analysis	y
1	1	1	1	-8.00
2	1	1	2	-9.49
3	1	1	3	-9.43
4	1	2	1	0.63
5	1	2	2	-1.10
6	1	2	3	-0.52
7	1	3	1	-9.41
8	1	3	2	-10.28
9	1	3	3	-10.23
10	2	1	1	0.67
11	2	1	2	-0.62
12	2	1	3	-3.09
13	2	2	1	4.69
14	2	2	2	4.10
15	2	2	3	4.95
16	2	3	1	-4.20
17	2	3	2	-4.09
18	2	3	3	-5.16
19	3	1	1	12.20
20	3	1	2	14.59
21	3	1	3	13.76
22	3	2	1	11.99
23	3	2	2	11.97
24	3	2	3	11.91
25	3	3	1	16.56
26	3	3	2	15.95
27	3	3	3	15.70

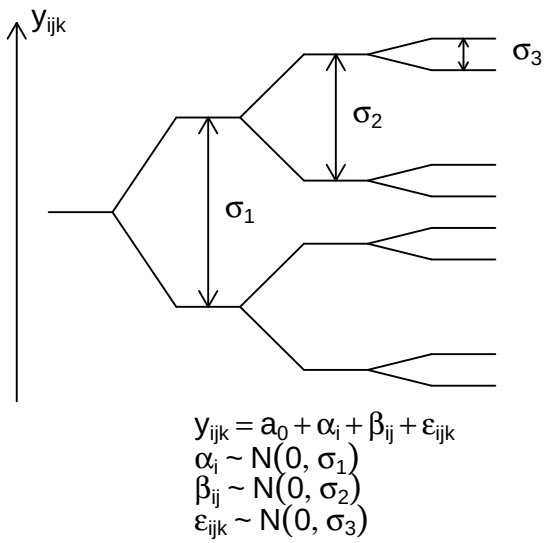

Figure 4.27: Hierarchy of a nested 2^3 design with variance components $\sigma_1 > \sigma_2 > \sigma_3$.

with the two functions lme4::lmer() and VCA::anovaVCA(). In both functions the nested effect structure is specified by ~batch/sample (read "sample nested within batch"), and the last factor (analyse) is omitted for being identical with the residual of the model, ϵ_{ijk}. The generic R-function confint() gives 95% confidence intervalls of the variance components.

```
library(lme4)
summary(res  <- lmer(y~ 1 +  (1|batch/sample), data=x ))
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: y ~ 1 + (1 | batch/sample)
##    Data: x
##
## REML criterion at convergence: 111.6
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.2756 -0.4212 -0.0073  0.4479  1.8189
##
## Random effects:
##  Groups       Name        Variance Std.Dev.
##  sample:batch (Intercept)  17.5050  4.1839
##  batch        (Intercept) 102.1885 10.1088
##  Residual                   0.8404  0.9167
## Number of obs: 27, groups:  sample:batch, 9; batch, 3
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)    2.373      6.003   0.395
```

```
confint(res)
```

```
##                    2.5 %    97.5 %
## .sig01         2.5641739  8.440086
## .sig02         3.4991394 24.955135
## .sigma         0.6827736  1.322468
## (Intercept) -11.3129630 16.058027
```

```
VCA::anovaVCA(y~batch/sample,x)
```

```
##
##
## Result Variance Component Analysis:
## ----------------------------------
##
```

which the data in hand is the most likely realization. With D denoting the data and a_i the model parameters, the optimization problem is $\max_{a_i} Prob(D|a_i)$, a method called Maximum Likelihood (ML). It can be shown that OLS belongs to this class.

```
##   Name          DF       SS          MS          VC          %Total    SD         CV[%]
## 1 total         2.475824                       120.537577  100       10.978961  462.751894
## 2 batch         2        1946.169529 973.084764  102.192132  84.78031  10.109012  426.08447
## 3 batch:sample  6        320.133475  53.355579   17.505067   14.522498 4.183906   176.347321
## 4 error         18       15.126806   0.840378    0.840378    0.697192  0.916721   38.638863
##
## Mean: 2.372537 (N = 27)
##
## Experimental Design: balanced  |  Method: ANOVA
```

The variance components are estimated to be $\sigma_{batch} = 10.1(15)$, $\sigma_{sample} = 4.2(5)$ and $\sigma_{analyse} = 0.9(1)$ with true values given in parenthesis. The confidence intervals are very wide, namely $0 \leq \sigma_{sample} \leq \infty$, $3.5 \leq \sigma_{batch} \leq 25$ and $0.7 \leq \sigma_{analyse} \leq 1.3$. The large uncertainty is a result of the small sample size. Nevertheless, the estimates are of the right order of magnitude. Based on these simulated results, the conclusion is to scrutinize the process step (labelled "batch") more closely, as this step is responsible for 85% of the variance of y_{ijk}.

Chapter 5

Mixture experiments

All designs discussed in the previous chapter 4 belong to the class of orthogonal designs in which all factors are varied independently. However, in many applications and especially in the chemical sciences there are cases where the factors cannot and should not be varied independently. If, say, there are three compounds x1,x2,x3, all in the range of 0-1 with 1 denoting the pure component, the design from blending these components must necessarily obey the mixture rule $\sum_i x_i = 1$. Strictly speaking, all problems in chemistry are mixture problem, but in most cases the solvent is treated as an inert "filler" not contributing to the mixture constraint and thereby allowing the remaining concentration factors to vary independently.
Due to the mixture constraints, mixture problems require special parametric models and corresponding to these models special designs. Mixture models and mixture designs will be introduced in the following sections and their use in R will be demonstrated with some examples borrowed from the literature.
A comprehensive overview on mixture experiments and a good source for further reading is the book (John A. Cornell 1990). (John Lawson, Cameron Willden 2016) gives an overview of the package **mixexp**, a collection of functions for creating and analyzing mixture designs in R.

5.1 Mixture models

Trying to identify the parametric OLS model $\hat{y} = a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3$ given the mixture constraint $\sum_{i=1}^{3} x_i = 1$ will fail because the mixture constraint renders one variable redundant. One way of dealing with the singularity is to drop one variable, say x_3, and proceed with the reduced model $\hat{y} = a_0^{'} + a_1 \cdot x_1^{'} + a_2 \cdot x_2^{'}$ and the auxiliary condition $x_3 = 1 - x_1 - x_2$. This is called the slack variable approach with the slack variable, here x_3, usually choosen to be the most inert component. Note that any other variable could have been excluded without affecting model statistics, as all slack variable models are statistically degenerated

and will give the same predictions.
A more natural way of dealing with the linear constraints is to integrate the mixture contraints $\sum_i x_i = 1$ into the regression equation, leading to the so called Scheffe models, named after the scientist having first derived these equations. From the parametric model

$$\hat{y} = a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3$$

and the mixture contraint

$$x_1 + x_2 + x_3 = 1$$

follows by substitution

$$\hat{y} = a_0(x_1 + x_2 + x_3) + a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3 \Leftrightarrow$$

$$\hat{y} = (a_0 + a_1) \cdot x_1 + (a_0 + a_2) \cdot x_2 + (a_0 + a_3) \cdot x_3$$

the Scheffe main effect model

$$\hat{y} = a_1^{'} \cdot x_1 + a_2^{'} \cdot x_2 + a_3^{'} \cdot x_3$$

which extends for K mixture components to the **Scheffe main effect model**, equation (5.1)

$$y = \sum_{i=1}^{K} a_i^{'} \cdot x_i + \epsilon \tag{5.1}$$

From equation (5.1) and the mixture constraint follows the identity $\hat{y}_{x_i=1} = a_i^{'}$, so the regression coefficients are the expected values $\hat{y}_i$ of the pure components x_i with intermediate mixtures simply becoming weighted sums of the pure components. In the linear Scheffe model, the components do not interact and this might be an assumtion too tight for many applications thus motivating the quadratic Scheffe models.
Starting with the quadratic model $\hat{y} = a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3 + a_4 \cdot x_1^2 + a_5 \cdot x_2^2 + a_6 \cdot x_3^2 + a_7 \cdot x_1 \cdot x_2 + a_8 \cdot x_1 \cdot x_3 + a_9 \cdot x_2 \cdot x_3$ by integrating the mixture constraint the quadratic Scheffe model is obtained

$$\hat{y} = a_1^{'} \cdot x_1 + a_2^{'} \cdot x_2 + a_3^{'} \cdot x_3 + a_4^{'} \cdot x_1 \cdot x_2 + a_5^{'} \cdot x_1 \cdot x_3 + a_6^{'} \cdot x_2 \cdot x_3$$

with the general **quadratic Scheffe model** for K components given by equation (5.2)

$$y = \sum_{i=1}^{K} a_i^{'} \cdot x_i + \sum_{j>i}^{K} a_{ij}^{'} \cdot x_i \cdot x_j + \epsilon \tag{5.2}$$

For any binary mixture x_i=0.5, x_j=0.5 follows from equation (5.2) the equality

$$\hat{y}_{x_i=0.5;x_j=0.5} = \frac{a_i + a_j}{2} + \frac{a_{ij}}{4}$$

which is depicted in figure 5.1 graphically for two binary mixtures with $a_{ij}>0$ (synergism) and $a_{ij}<0$ (antagonism). The straight line indicates the linear Scheffe model with $a_{ij}=0$.

Accounting for higher order effects such as ternary interactions can be achieved by augmenting the quadratic Scheffe model with higher order interaction terms as in the **full cubic Scheffe model**, see equation (5.3)

$$y = \sum_{i=1}^{K} a_i \cdot x_i + \sum_{j>i}^{K} a_{ij} \cdot x_i \cdot x_j + \sum_{j>i}^{K} \delta_{ij} \cdot x_i \cdot x_j \cdot (x_i - x_j) + \sum_{k>j>i}^{K} a_{ijk} \cdot x_i \cdot x_j \cdot x_k + \epsilon \quad (5.3)$$

Figure 5.2 shows a binary mixture plot of a full cubic model, which now can account for synergistic and antagonistic effects in mixture space. The cubic model underlying figure 5.2 is of the parametric form $\hat{y} = a_1 \cdot x_1 + a_2 \cdot x_2 + a_{12} \cdot x_1 \cdot x_2 + \delta_{12} \cdot x_1 \cdot x_2 \cdot (x_1 - x_2)$ with parameters set to a_1=50, a_2=100, a_{12}=0 and $\delta_{12} = -200$.

The full cubic model, formula (5.3), is very flexible due to the large number of regression parameters, hence it tends to overfit the data easily. Often the **reduced (or special) cubic model**, equation (5.4), is sufficiently complex to adequately describe the experimental situation.

$$y = \sum_{i=1}^{K} a_i \cdot x_i + \sum_{j>i}^{K} a_{ij} \cdot x_i \cdot x_j + \sum_{k>j>i}^{K} a_{ijk} \cdot x_i \cdot x_j \cdot x_k + \epsilon \quad (5.4)$$

The mixture models described above - linear, quadratic, full and the reduced (special) cubic model - are sufficient for modeling most mixture sytems appropriately and can be used as building blocks for modeling mixture-process designs, in which K mixture components $x_1, x_2, \ldots x_K$ are combined with P process factors $z_1, z_2, \ldots z_P$. The first mixture-process model results from fully crossing the quadratic Scheffe model with a factorial (bilinear) process model, i.e.

$$y = \left(\sum_{i=1}^{K} a_i \cdot x_i + \sum_{j>i}^{K} a_{ij} \cdot x_i \cdot x_j \right) \cdot \left(\alpha_0 + \sum_{l=1}^{P} \alpha_l \cdot z_l + \sum_{m>l}^{R} \alpha_{lm} \cdot z_l \cdot z_m \right) + \epsilon$$

and from product expansion follows directly the parametric model, equation (5.5)

$$\begin{aligned} y = & \sum_{i=1}^{K} b_i \cdot x_i + \sum_{j>i}^{K} b_{ij} \cdot x_i \cdot x_j + \sum_{i=1}^{K} \sum_{l=1}^{P} b'_{il} \cdot x_i \cdot z_l + \sum_{j>i}^{K} \sum_{l=1}^{P} b'_{ijl} \cdot x_i \cdot x_j \cdot z_l + \\ & \sum_{i}^{K} \sum_{m>l}^{P} b'_{ilm} \cdot x_i \cdot z_l \cdot z_m + \sum_{j>i}^{K} \sum_{m>l}^{P} b'_{ijlm} \cdot x_i \cdot x_j \cdot z_l \cdot z_m + \epsilon \end{aligned} \quad (5.5)$$

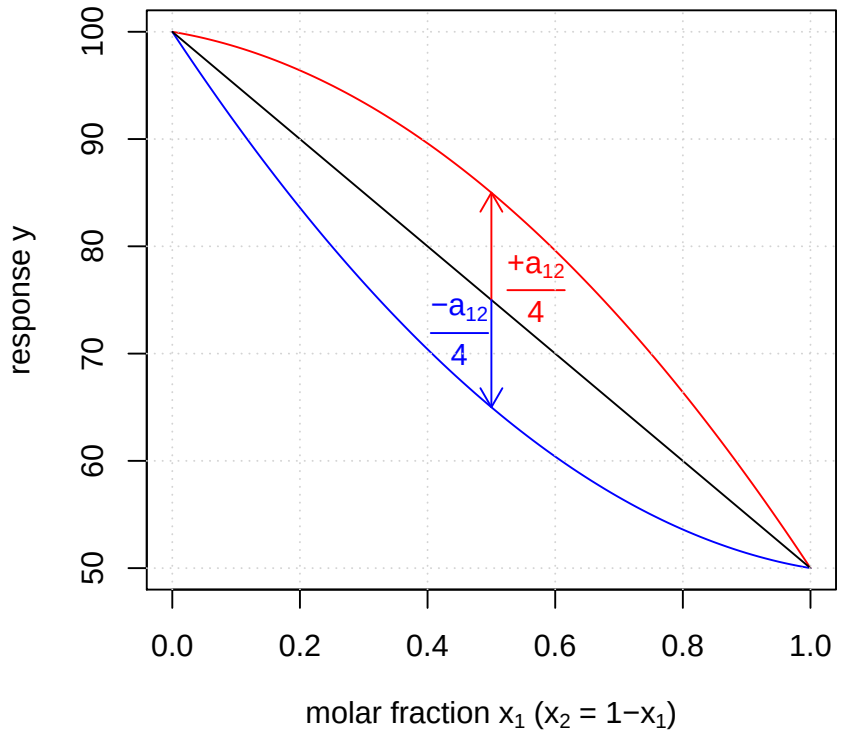

Figure 5.1: Response plot of a quadratic Scheffe model for a binary mixture x 1 ,x 2 with synergistic and antagonistic effects labelled red and blue, respectively

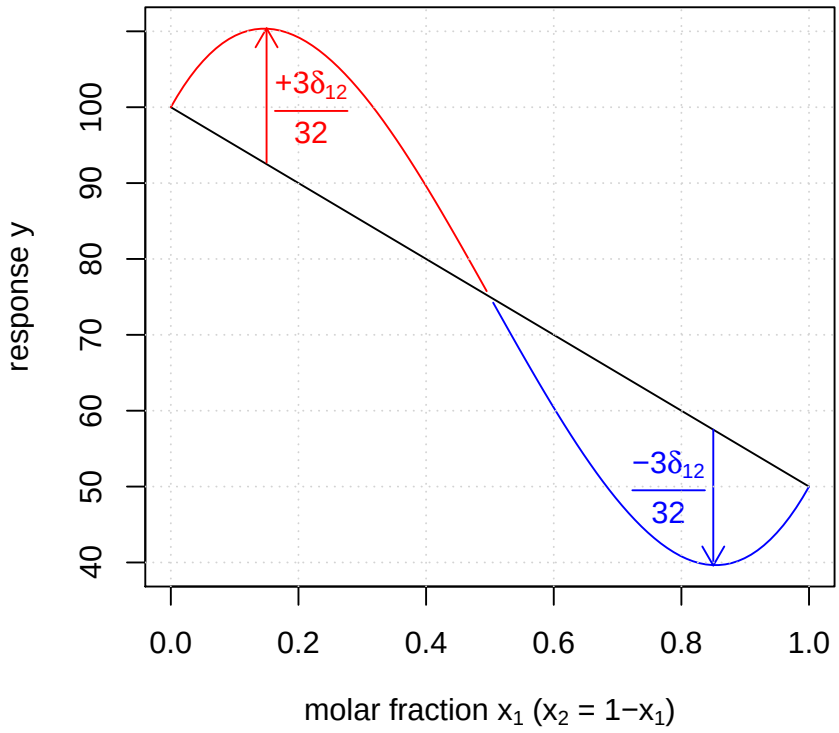

Figure 5.2: Response plot of a cubic Scheffe model for a binary mixture x 1 ,x 2 with synergistic and antagonistic effects in mixture space

The number of regression coefficient in formula (5.5) as a function of K and P grows large rapidly making crossed designs of limited use for most applications. A more parsimonious model is obtained by specifically linking a quadratic Scheffe model with an RSM model leading to equation (5.6)

$$y = \sum_{i=1}^{K} b_i \cdot x_i + \sum_{j>i}^{K} b_{ij} \cdot x_i \cdot x_j + \sum_{i=1}^{K} \sum_{l=1}^{P} b'_{il} \cdot x_i \cdot z_l + \sum_{m>l}^{P} b'_{lm} \cdot z_l \cdot z_m + \sum_{l=1}^{P} b_{ll} \cdot z_l^2 + \epsilon \quad (5.6)$$

The four mixture models, equations #1:(5.1), #2:(5.2), #3:(5.3), #4:(5.4) and the two mixture-process designs, fully and partially crossed mixture-process designs, equations #5:(5.5) and #6:(5.6), can be conveniently referenced by the model argument in the R-function mixexp::MixModel(..., model=#) with # denoting the model number in the above list.

When analyzing mixture experiments all principles of OLS model validation apply such as statistical testing and residual analysis as will be demonstrated with some example data provided with the R package **mixexp** from (R.H. Myers, D.C. Montgomery 2002). The aim of the experiment was to find an optimal blend of the components x_1,x_2,x_3 maximizing the etching rate of silicon wafers (response **erate**). The design of this experiment was a partially replicated Simplex-Centroid Design (explained later) augmented by some interior points. The design maximally supports a reduced cubic model and will serve here as an example to demonstrate some functions from **mixexp**. Different from orthogonal models, it is usually not possible in mixture models to individually remove non-significant model terms as this will violate the mixture constraints and render the model singular. It is usually sufficient to choose between the models #1-#4 to appropriately describing the data within the model domain[1].
The example data is shown in figure 5.3 as a ternary plots and supports either model #1, #2 or #4. Replicates are marked by circles and well balanced over the design space so as not to disturb the orthogonality of the design.

Following the principle of parsimony, the data is first analysed with a quadratic Scheffe model and next with a reduced cubic model, and residuals are being compared in figure 5.4 (note that R^2 in the lm-output of the no-intercept models are biased and should be ignored. R^2 values reported by MixModel are correct and should be used instead).

```
library(mixexp)
data("etch")
summary(lm(erate ~ -1 + x1 + x2 + x3 +
             x1:x2 + x1:x3 + x2:x3,
               data = etch))
```

[1]More precisely it is sufficient to reduce the cubic model #4 to the quadratic model #2 if third order effects are not significant, or to reduce #2 to #1 if quadratic effects turn out not significant.

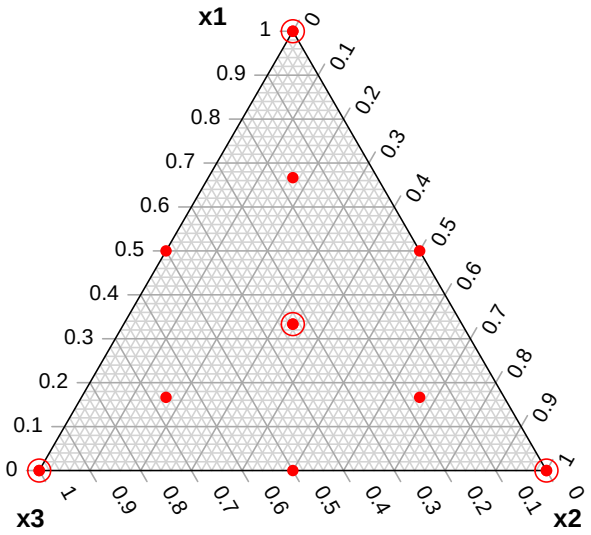

Figure 5.3: Ternary plot of the etching data with replicates marked by circles

```
##
## Call:
## lm(formula = erate ~ -1 + x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3,
##     data = etch)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -157.68  -30.13   11.62   38.04  177.90
##
## Coefficients:
##       Estimate Std. Error t value Pr(>|t|)
## x1      534.64      83.35   6.414 0.000206 ***
## x2      329.16      83.35   3.949 0.004241 **
## x3      252.73      83.35   3.032 0.016254 *
## x1:x2  1343.11     469.58   2.860 0.021145 *
## x1:x3   644.53     469.58   1.373 0.207133
## x2:x3   711.68     469.58   1.516 0.168101
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 120.3 on 8 degrees of freedom
## Multiple R-squared:  0.9721, Adjusted R-squared:  0.9511
## F-statistic: 46.38 on 6 and 8 DF,  p-value: 8.741e-06
```

```
# note: R2 is wrong because when there is no intercept
# then SS.model is not adjusted by the mean
# Normally, the intercept a0 ensures mean adjustment

(res1 <- MixModel(etch, "erate",
  mixcomps = c("x1", "x2", "x3"), model = 2))
```

```
##
##       coefficients   Std.err  t.value        Prob
## x1         534.6383  83.34866 6.414481 0.000205933
## x2         329.1622  83.34866 3.949220 0.004240658
## x3         252.7336  83.34866 3.032245 0.016253810
## x2:x1     1343.1061 469.58404 2.860204 0.021144784
## x3:x1      644.5346 469.58404 1.372565 0.207132664
## x2:x3      711.6775 469.58404 1.515549 0.168101200
##
## Residual standard error:  120.261  on  8 degrees of freedom
## Corrected Multiple R-squared:  0.7583617
```

```
##
## Call:
## lm(formula = mixmodnI, data = frame)
##
```

```
## Coefficients:
##     x1      x2      x3   x1:x2   x1:x3   x2:x3
##  534.6   329.2   252.7  1343.1   644.5   711.7
```

```
# quadratic Scheffe  R2 correct
(res2 <- MixModel(etch, "erate",
      mixcomps = c("x1", "x2", "x3"), model = 4))
```

```
##
##          coefficients   Std.err     t.value         Prob
## x1         550.199515  23.22446 23.69051468 6.067419e-08
## x2         344.723325  23.22446 14.84311192 1.509476e-06
## x3         268.294753  23.22446 11.55224716 8.203511e-06
## x2:x1      689.537037 146.51489  4.70625916 2.192427e-03
## x3:x1       -9.034392 146.51489 -0.06166193 9.525557e-01
## x2:x3       58.108466 146.51489  0.39660451 7.034720e-01
## x2:x3:x1  9243.333333 940.85336  9.82441444 2.404146e-05
##
## Residual standard error:  33.43177  on  7 degrees of freedom
## Corrected Multiple R-squared:  0.9836603
```

```
##
## Call:
## lm(formula = mixmodnI, data = frame)
##
## Coefficients:
##       x1        x2        x3     x1:x2     x1:x3     x2:x3  x1:x2:x3
##  550.200   344.723   268.295   689.537    -9.034    58.108  9243.333
```

```
# reduced cubic
kappa(res1)
```

```
## [1] 9.353397
```

```
kappa(res2)
```

```
## [1] 57.53584
```

```
par(mfrow=c(1,2))
colcol                    <- rep("black")
colcol[duplicated(etch[,1:3])]<- "red" #color label replicates
par(mfrow=c(1,2))
plot(predict(res1), rstudent(res1),ylim=c(-4,3),
     xlab=expression( paste("predicted response ", hat(y)) ),
     ylab="studentized residuals",type="n",
     main=expression(paste("(",frac(paste("y - ",
            hat(y)),sigma), ") ~ ", hat(y)   )) )
text(predict(res1), rstudent(res1),label=1:nrow(etch), col=colcol)
abline(h=c(-2,0,2),lty=c(2,1,2))
```

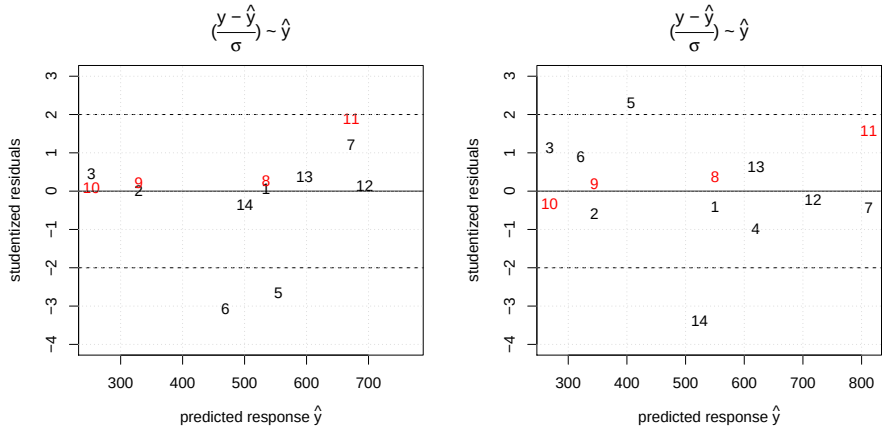

Figure 5.4: Studentized residuals versus model predictions from quadratic (left panel) and reduced cubic Scheffe models of the etching data. The replicate pairs (3,10), (2,9), (8,1) and (7,11) are better described within relication error by the cubic model (right panel) compared with the quadratic model (left panel).

```
grid()

# check residuals of reduced cubic model
plot(predict(res2), rstudent(res2),ylim=c(-4,3),
     xlab=expression( paste("predicted response ", hat(y)) ),
     ylab="studentized residuals",type="n",
     main=expression(paste("(",frac(paste("y - ",
          hat(y)),sigma), ") ~ ", hat(y)   )) )
text(predict(res2), rstudent(res2),label=1:nrow(etch),col=colcol)
abline(h=c(-2,0,2),lty=c(2,1,2))
grid()
```

The corrected R^2 value of the quadratic Scheffe model, R^2=0.76 is small compared with the cubic model, R^2=0.98, indicating that the term x1:x2:x3 contributes substantially to the variance of the response. Similar conclusions can be drawn from figure 5.4 which reveals the quadratic Scheffe model to be biased by, e.g., not appropriately describing the replicate pair #7,#11. In the cubic Scheffe model the ternary term x1:x2:x3 is found significant and the residuals of the cubic model are now in better accordance with the normality assumption of the error term ϵ.
The cubic model can be visualized with a ternary contour plot shown in figure 5.5. The maximum lies close to the centroid (x_1=0.33,x_2=0.33,x_3=0.33)[2].

[2]The precise location of the optimum remains vague in contour plots. With the aid of

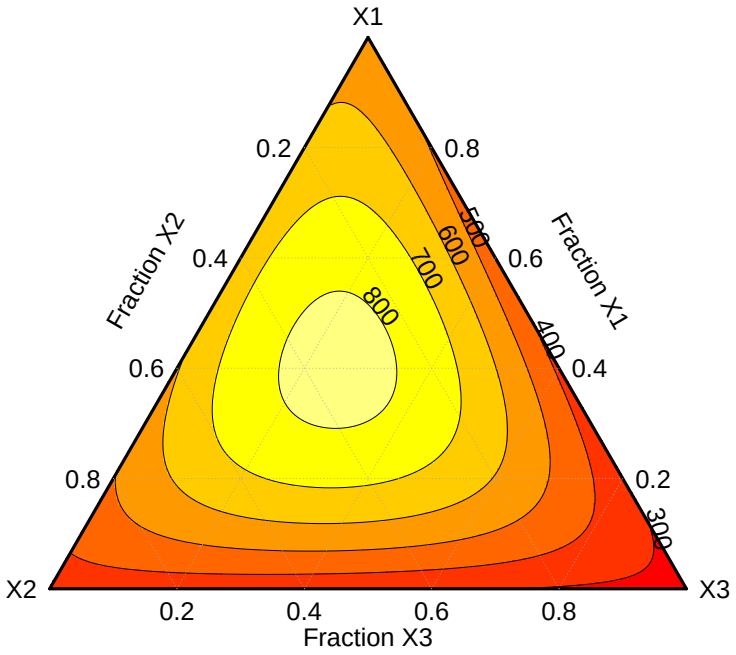

Figure 5.5: Contour plot of the reduced cubic model of the etching data

```
library(mixexp)
data("etch")
invisible(capture.output(res <- MixModel(etch, "erate",
            mixcomps = c("x1", "x2", "x3"), model = 4)) )
# plot cubic mixture model as ternary contour plot
ModelPlot(model = res,
          dimensions = list(x1 = "x1", x2 = "x2", x3 = "x3"),
          contour = TRUE, cuts = 6, fill = TRUE)
```

Contour plots can also be used for dimensions higher than K=3 by conditioning

optimization, the subject of chapter 6, the maximum can be located to be $x_1^* = 0.41, x_2^* = 0.34, x_3^* = 0.25$ with an expected etching rate at this point of $\hat{y} = 832$.

(slicing) the plots on variables set constant. However, this can become confusing when there are many slices to compare, and therefore another popular, and less confusing way of visualizing high dimensional mixture models are the Cox effect plots. First, the Cox direction is defined as the line that connects the centroid point with the vertices (see figure 5.6) of the mixture space. When moving in the Cox direction of, say, x_1 the ratio of the remaining mixture components stays constant, here $\frac{x_2}{x_3} = 1$, and this is next to equivalent to an "independent" effect of x1 in mixture space. The Cox effects are the traces obtained from sclicing the response surface along the Cox directions. Figure 5.6 shows the Cox direction in three dimensional mixture space along with a Cox effect plot of the reduced cubic model of the etching data.

```
rm(list=ls())
library(mixexp)
library(Ternary)
data(etch)
invisible(capture.output(res <- MixModel(etch, "erate",
       mixcomps = c("x1", "x2", "x3"), model = 4)) )
# plot cubic mixture model as ternary contour plot
par(mfrow=c(1,2))
TernaryPlot(atip="x1", btip="x2", ctip="x3", axis.cex=1.2,lab.cex = 1.3,
            axis.labels=seq(0,1,0.1))
TernaryArrows(c(0,0.5,0.5),c(1,0,0),col="red",length=0.15, lwd=2)
TernaryArrows(c(0.5,0,0.5),c(0,1,0),col="red",length=0.15, lwd=2)
TernaryArrows(c(0.5,0.5,0),c(0,0,1),col="red",length=0.15, lwd=2)
AddToTernary(points, list(x1=rep(0.33,3),x2=rep(0.33,3),
                          x3=rep(0.33,3)),pch=16,
                          col="black",cex=1.5)
title(main="Cox direction",cex.main=1.5)

ModelEff(nfac = 3, mod = 4, dir = 2, ufunc = res,
             dimensions = list("x1", "x2", "x3"))
```

5.2 Mixture designs

The previous section introduced six different parametric models for modeling mixture and mixture process systems. Associated with each model are mixture and mixture process designs optimally supporting the chosen model. Models and designs are two sides of the same coin: The parametric model determines the design and the design determines and limits the model which can be estimated from the data.

Mixture designs can be broadly divided into **regular designs** in which the design covers the whole mixture space ($0 \leq x_i \leq 1$) and **irregular (or constrained) designs** in which only a subspace, $LB_i(> 0) \leq x_i \leq UB_i(< 1)$, of the whole mixture space is populated. Figure 5.7 shows examples of a regular and irregular

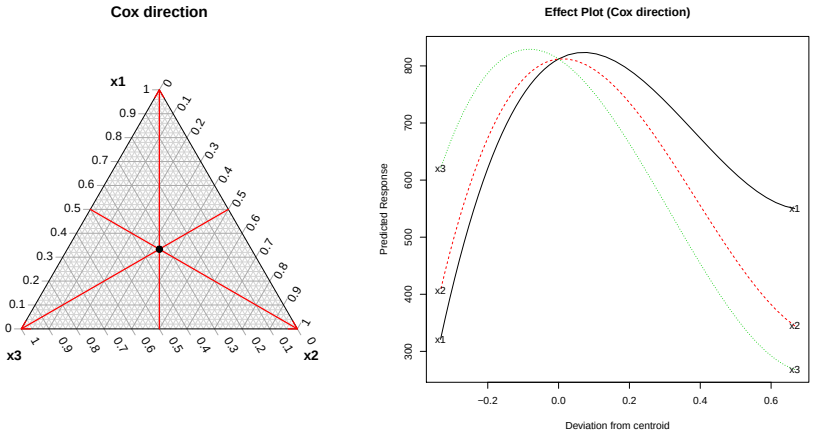

Figure 5.6: Cox direction in 3D mixture space (left panel) and effect plot along the Cox direction of the reduced cubic etching model (right panel)

mixture design in three dimensions.

Linear and quadratic Scheffe models (model #1, #2) can be estimated with (regular) Simplex Lattice Designs (SLD) which are shown for K=3 in figure 5.8 and created with the mixexp R-function SLD(fac,lev). The agument *fac* denotes the number of factors and *lev* specifies the number of levels with $x_i>0$.

The pure components (the vertex points in figure 5.8) are sufficient for estimating the linear Scheffe model. The quadratic Scheffe model is supported by augmenting the former design with all binary mixtures at the center of the three edges. Both designs are saturated[3] and need therefore additional runs for estimating the error variance and to check for the presence of higher order effects.
The reduced cubic model (model #4) can be estimated with Simplex Centroid Designs (SCD). Again, the SCD is saturated and need additional runs for estimating error variance and Lack of Fit.
Finally, three level Simplex Lattice Designs can be used for estimating the full cubic model (model #3). All designs can be created conveniently with the functions mixexp::SLD() and mixexp::SCD() and examples for K=3 of the former and the latter are depicted in the figures 5.8 and 5.9.

Classical mixture models and designs are rigid in terms of model structure and design size similar to what has been said about classical orthogonal designs. A second order Simplex-Lattice Design for instance will render all two-way interaction estimable, irrespective of whether they are of interest or physically possible. Model structure and design size can be specified more flexibly with optimal designs as shown with the following R-code. Here the aim was to

[3]The number of runs equal the number of regression. This will give a perfect fit with R^2=1.

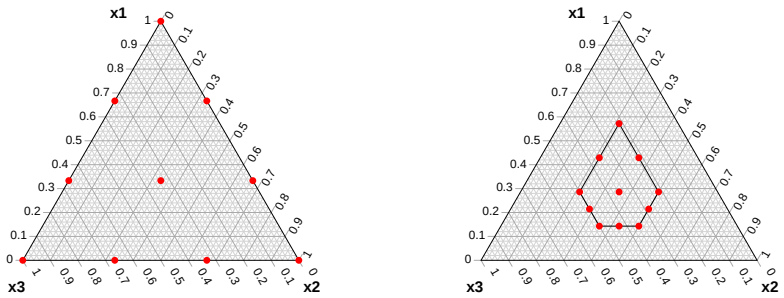

Figure 5.7: Examples of regular (left panel) and irregular (constrained) designs (right panel)

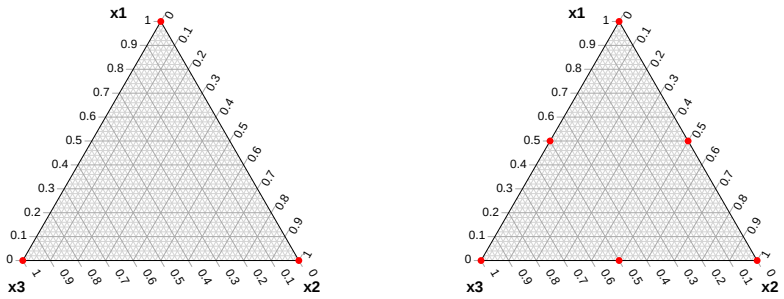

Figure 5.8: Simplex Lattice Designs for estimating linear (SLD(3,1), left panel) and quadratic Scheffe models (SLD(3,2), right panel)

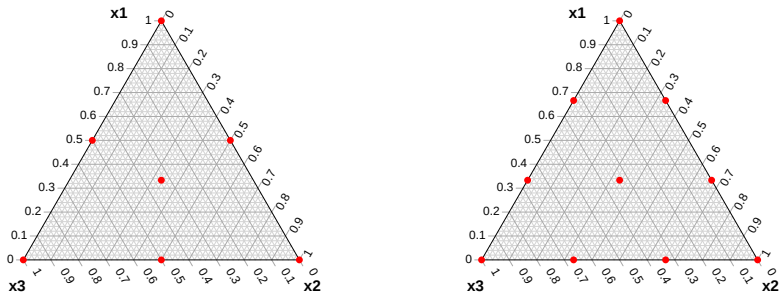

Figure 5.9: Simplex Centroid Design SCD(3) (left panel) for reduced cubic model and three level Simplex Lattice Design SLD(3,3) (right panel) for the full cubic Scheffe model

parsimoniously identify a non-standard Scheffe model, in R-notation ~ -1 + x1 + x2 + x3 + x1:x2 + x1:x2:x3, with six runs from a candidate set of 60 regular mixture candidates. Figure 5.10 depicts the D-optimal design points in mixture space comprising pure components (referring to main effects x1,x2,x3), the binary mixture x1,x2 (for the interaction term x1:x2) and two points close to the centroid for identifying the three way interaction x1:x2:x3.

```
rm(list=ls())
library(AlgDesign)
library(Ternary)
set.seed(1234)
xx <- expand.grid(x1=seq(0,1,0.1),
                  x2=seq(0,1,0.1),
                  x3=seq(0,1,0.1) ) # grid design
x <- xx[apply(xx,1,sum)==1,] # sum_x.i=1 only
mix <- optFederov(~ -1 + x1 + x2 + x3 +
            x1:x2 + x1:x2:x3, data=x,nTrials=6,
                  nRepeats=1000)$design
TernaryPlot(atip="x1", btip="x2", ctip="x3",
          axis.labels=seq(0,1,0.1), lab.cex=1)
AddToTernary(points, mix,pch=16,col="red",cex=1)
```

Designs for the fully crossed mixture-process model (#5), equation (5.5), results from crossing mixture and process designs. Let *fac* be a full or fractional factorial design with N_1 rows and P columns and *mix* a mixture design of dimension

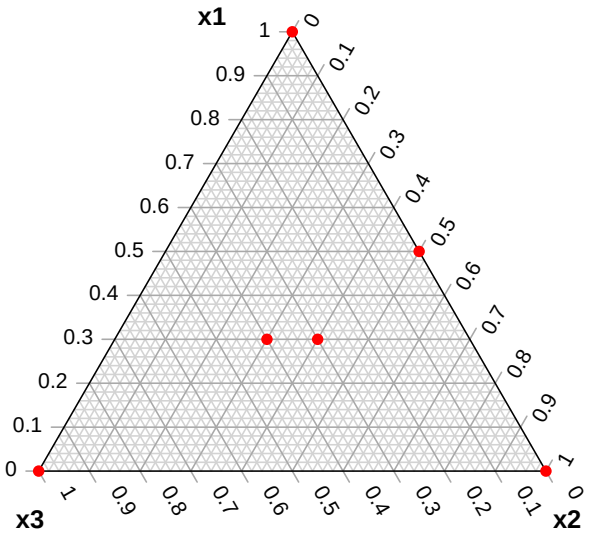

Figure 5.10: D-optimal mixture design of the model -1 + x1 + x2 + x3 + x1:x2 + x1:x2:x3 written in R-formula notation

(N_2)x(K), the Cartesian product of dimension (N_1 * N_2)x(P+K) supports the fully crossed mixture-process design as the following R-code shows for K=3 and P=3, respectively

```
rm(list=ls())
set.seed(1234)

fac <- FrF2::FrF2(8,3,randomize=F)  # 2^3 full factorial design; N=8
mix <- mixexp::SLD(3,3) # SLD design N=10
                        # 3vertex+3x2edges+centroid
fac <- data.frame(sapply(fac,function(x)
  as.numeric(as.character(x))))
# convert factor to numerical
colnames(fac) <- paste("z",1:3,sep="")
x <- merge(mix,fac) # Cartesian product
dim(x)
```

```
## [1] 80  6
```

```
x$y <- rnorm(nrow(x))
(res <- mixexp::MixModel(x, "y",
         mixcomps = c("x1", "x2", "x3"),
         procvars = c("z1", "z2", "z3"),
         model = 5)) # Fully crossed mixture-process model #5
```

```
##
##             coefficients   Std.err     t.value       Prob
## x1            0.05990870 0.3082853  0.19432872 0.84695368
## x2           -0.57818736 0.3082853 -1.87549413 0.06842376
## x3           -0.20611270 0.3082853 -0.66857769 0.50780621
## x2:x1         0.05555502 1.3647251  0.04070785 0.96774194
## x3:x1        -1.00098144 1.3647251 -0.73346746 0.46777458
## x2:x3         0.85963536 1.3647251  0.62989636 0.53253190
## x1:z1        -0.17240678 0.3082853 -0.55924415 0.57927649
## x1:z2         0.48903461 0.3082853  1.58630507 0.12095767
## x1:z3         0.12188272 0.3082853  0.39535684 0.69479036
## x2:z1         0.22176294 0.3082853  0.71934310 0.47633042
## x2:z2         0.26971744 0.3082853  0.87489542 0.38712724
## x2:z3        -0.03665631 0.3082853 -0.11890384 0.90597790
## x3:z1         0.18778266 0.3082853  0.60911964 0.54606984
## x3:z2         0.04506181 0.3082853  0.14616916 0.88456052
## x3:z3        -0.12555989 0.3082853 -0.40728467 0.68608424
## x2:x1:z1     -1.82763803 1.3647251 -1.33919867 0.18846181
## x2:x1:z2     -0.75822632 1.3647251 -0.55558905 0.58174831
## x2:x1:z3      0.06996290 1.3647251  0.05126520 0.95938274
## x3:x1:z1      0.13423785 1.3647251  0.09836256 0.92216144
## x3:x1:z2     -0.90892644 1.3647251 -0.66601431 0.50942517
```

```
## x3:x1:z3      1.30367180 1.3647251  0.95526330 0.34548243
## x2:x3:z1     -0.72218119 1.3647251 -0.52917705 0.59976052
## x2:x3:z2     -1.33291497 1.3647251 -0.97669119 0.33489917
## x2:x3:z3      1.88583297 1.3647251  1.38184092 0.17509079
## x1:z1:z2      0.26702234 0.3082853  0.86615321 0.39184221
## x1:z1:z3     -0.58110294 0.3082853 -1.88495153 0.06710090
## x1:z2:z3     -0.14089678 0.3082853 -0.45703367 0.65024895
## x2:z1:z2     -0.13425104 0.3082853 -0.43547654 0.66568022
## x2:z1:z3      0.01472591 0.3082853  0.04776715 0.96215199
## x2:z2:z3     -0.17024018 0.3082853 -0.55221625 0.58403377
## x3:z1:z2     -0.30992865 0.3082853 -1.00533047 0.32109681
## x3:z1:z3     -0.69175568 0.3082853 -2.24388117 0.03074588
## x3:z2:z3      0.45376014 0.3082853  1.47188357 0.14928650
## x2:x1:z1:z2  -0.45968552 1.3647251 -0.33683378 0.73809548
## x2:x1:z1:z3   0.93480706 1.3647251  0.68497829 0.49751482
## x2:x1:z2:z3   2.83909884 1.3647251  2.08034487 0.04428803
## x3:x1:z1:z2  -0.64270562 1.3647251 -0.47094146 0.64037490
## x3:x1:z1:z3   3.11414546 1.3647251  2.28188481 0.02818482
## x3:x1:z2:z3   2.41847375 1.3647251  1.77213255 0.08439084
## x2:x3:z1:z2  -1.50904885 1.3647251 -1.10575299 0.27578515
## x2:x3:z1:z3   3.01676321 1.3647251  2.21052813 0.03316260
## x2:x3:z2:z3  -1.35284331 1.3647251 -0.99129365 0.32781272
##
## Residual standard error:  0.926512  on  38 degrees of freedom
## Corrected Multiple R-squared:  0.5948409
```

```
##
## Call:
## lm(formula = mixmod, data = frame)
##
## Coefficients:
##          x1           x2           x3        x2:x1        x3:x1        x2:x3
##     0.05991     -0.57819     -0.20611      0.05556     -1.00098      0.85964
##       x1:z3        x2:z1        x2:z2        x2:z3        x3:z1        x3:z2
##     0.12188      0.22176      0.26972     -0.03666      0.18778      0.04506
##    x2:x1:z2     x2:x1:z3     x3:x1:z1     x3:x1:z2     x3:x1:z3     x2:x3:z1
##    -0.75823      0.06996      0.13424     -0.90893      1.30367     -0.72218
##    x1:z1:z2     x1:z1:z3     x1:z2:z3     x2:z1:z2     x2:z1:z3     x2:z2:z3
##     0.26702     -0.58110     -0.14090     -0.13425      0.01473     -0.17024
##    x3:z2:z3  x2:x1:z1:z2  x2:x1:z1:z3  x2:x1:z2:z3  x3:x1:z1:z2  x3:x1:z1:z3
##     0.45376     -0.45969      0.93481      2.83910     -0.64271      3.11415
## x2:x3:z1:z3  x2:x3:z2:z3
##     3.01676     -1.35284
```

```
kappa(res)
```

```
## [1] 15.35277
```

Fully crossed designs and models are very bulky and hardly feasible in ordinary laboratory setups, and the partially crossed designs, model #6 (equation (5.6))), might be a more realistic alternative for most projects. Partially crossed and parsimonious mixture-process designs can be generated by D-optimally selecting points from a fully crossed candidate set with AlgDesign::optFederov() as shown by the following code

```
rm(list=ls())
set.seed(1234)
rsm  <- rsm::bbd(3, n0=1,randomize=F)[,-(1:2)] # Box-Behnken design;N=13
sld  <- mixexp::SLD(3,3) # cubic Simplex-lattice design;N=10
colnames(rsm) <- paste("z",1:3,sep="")
candidate <- merge(sld,rsm) # Cartesian product dim=(130 x 6)
mix <- AlgDesign::optFederov(~ -1 + (x1+x2+x3)^3 +
                               x1:z1 + x1:z2 + x1:z3 +
                               x2:z1 + x2:z2 + x2:z3 +
                               x3:z1 + x3:z2 + x3:z3 +
                               z1:z2 + z1:z3 + z2:z3 +
                               I(z1^2) + I(z2^2) + I(z3^2) ,
                             data=candidate,nTrials=30,
                             criterion="D", nRepeats=1000)$design
mix$y <- rnorm(nrow(mix))
(res <- mixexp::MixModel(mix, "y",
         mixcomps = c("x1", "x2", "x3"),
         procvars = c("z1", "z2", "z3"),
         model = 6))
```

```
## [1] "Warning, when using Model 6 the design in the process variables allow fitting the full qu
##
##         coefficients   Std.err    t.value      Prob
## x1         1.1568064 1.1661825  0.9919600 0.3471356
## x2         1.4896200 1.0769369  1.3832007 0.1999491
## x3         0.5302152 1.2014390  0.4413168 0.6693956
## I(z1^2)   -0.4855809 0.5771805 -0.8412983 0.4219706
## I(z2^2)   -0.2406845 0.5947270 -0.4046974 0.6951494
## I(z3^2)   -0.2628178 0.6226630 -0.4220868 0.6828656
## x1:x2     -3.0211103 3.3953217 -0.8897862 0.3967412
## x1:x3      0.3875330 3.5732353  0.1084544 0.9160147
## x2:x3     -2.6004735 3.4484189 -0.7541060 0.4700549
## x1:z1     -0.5310433 0.5583275 -0.9511321 0.3663794
## x1:z2     -0.4529325 0.5666571 -0.7993062 0.4446950
## x1:z3     -0.2559378 0.4999943 -0.5118815 0.6210569
## x2:z1     -0.3321763 0.5764581 -0.5762367 0.5785786
## x2:z2     -0.3683906 0.5352710 -0.6882320 0.5086556
## x2:z3     -0.2032059 0.5336694 -0.3807712 0.7122050
## x3:z1      0.4478906 0.6118973  0.7319702 0.4828108
```

```
## x3:z2      0.4189375 0.6242892  0.6710632 0.5190298
## x3:z3     -0.8799695 0.5389427 -1.6327700 0.1369515
## z1:z2      0.1701386 0.4439679  0.3832228 0.7104494
## z1:z3      0.4408563 0.4035429  1.0924644 0.3030079
## z2:z3      0.1727396 0.4084885  0.4228750 0.6823111
##
## Residual standard error:  1.188226  on  9 degrees of freedom
## Corrected Multiple R-squared:  0.5440434
```

```
##
## Call:
## lm(formula = mixmod, data = frame)
##
## Coefficients:
##      x1       x2       x3  I(z1^2)  I(z2^2)  I(z3^2)    x1:x2    x1:x3    x2:x3    x
##  1.1568   1.4896   0.5302  -0.4856  -0.2407  -0.2628  -3.0211   0.3875  -2.6005  -0
##   x2:z1    x2:z2    x2:z3    x3:z1    x3:z2    x3:z3    z1:z2    z1:z3    z2:z3
## -0.3322  -0.3684  -0.2032   0.4479   0.4189  -0.8800   0.1701   0.4409   0.1727
```

```
kappa(res)
```

```
## [1] 24.00369
```

Constrained or irregular designs can be constructed with the extreme vertices algorithm implemented in the function Xvert() from the package ***mixexp*** by specifying dimension and constrained region $(LB_i \geq 0) \leq x_i \leq UB_i \leq 1$ with LB_i and UB_i denoting lower and upper bounds of x_i, respectively. Consistency requires the sum of the upper bounds to exceed 1, that is $\sum_i UB_i > 1$.
Regular and irregular designs are often sparse in supporting higher order terms. Additional interior points can be created with the function Fillv() by averaging and thereby creating midpoints between all possible pairs.
Conversely, if a constrained design has too many runs given project objectives, it can be reduced by D-optimally selecting a subset of design points with the R-function AlgDesign::optFederov().
Augmentation of a constrained design with interior points is demonstrated with the following R-code, and figure 5.11 shows the design prior and after augmentation.

```
rm(list=ls())
x1 <- mixexp::Xvert(3,uc=c(.3,.3,1),lc=c(0,0,0),ndm=1,plot=F)[,-4]
x2 <- mixexp::Fillv(3,x1)
par(mfrow=c(1,2))
TernaryPlot(atip="x1", btip="x2", ctip="x3",axis.cex=1.2,lab.cex = 1.5,
          axis.labels=seq(0,1,0.1))
AddToTernary(points, x1,pch=16,col="red",cex=1.5)
TernaryPlot(atip="x1", btip="x2", ctip="x3",axis.cex=1.2,lab.cex = 1.5,
            axis.labels=seq(0,1,0.1))
AddToTernary(points, x2,pch=16,col="red",cex=1.5)
```

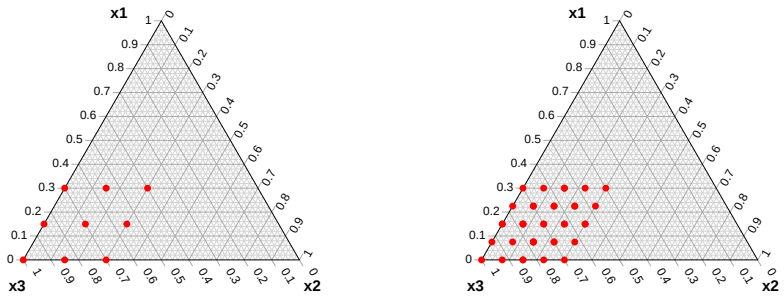

Figure 5.11: Constrained design $x_1 < 0.3; x_2 < 0.3; \sum_i x_i = 1$ (left panel) and corresponding design augmented by interior points (right panel);

Table 5.1 gives an overview of the so far discussed mixture models and designs combined with short hints on how to create these designs in R.

Finally and similar to orthogonal designs, mixture designs can be blocked if there are nuisance parameters such as different raw material batches, technicians or lab equipment. Although there are a couple of optimally blocked mixture designs derived from first principles (for details see (John A. Cornell 1990)), algorithmic blocking with the R-function AlgDesign::optBlock() will suffice for most practical purposes. The following example code will block a third order Simplex-lattic design with ten runs into two blocks and depict both blocks in figure 5.12. In addition the example code shows a few work-arounds to circumvent a couple of problems not covered by the standard functionality. First, define the variable x_{ij}=x_i-x_j for specifying the cubic term x_i:x_j:(x_i-x_j) in the formula expression. Second, define a new variable according to block2=as.integer(block==“B2”) as a unique block indicator. This is required because lm() by default drops the first level of a factor (here B1 in block={B1,B2}) **when the model has an intercept**. However, when there is **no intercept** both blockB1 and blockB2 are kept in the model matrix thus rendering the model singular because of blockB1+blockB2=1 and x1+x2+x3=1. This problem is circumvented with the new block variable block2=as.integer(block==“B2”). Then the location parameters of the full cubic lm-model agree well with the true model despite the fact that no term is found significant. However, this is not surpsising as there is only 1 degree of freedom rendering the test statistics very conservative.

Table 5.1: Mixture models and designs along with hints on how to create the designs in R. The numbers (1-6) in the list directly refer to the model argument in the R-function mixexp::MixModel(...,model=(1-6))

#	Model	Design	R-function
1	linear Scheffe	one-level Simplex-lattice	SLD(K,1)
2	quadratic Scheffe	two-level Simplex-lattice	SLD(K,2)
3	full cubic Scheffe	three-level Simplex-lattice	SLD(K,3)
4	special cubic Scheffe	Simplex-centroid	SCD(K)
5	fully crossed mix-process	Cartesian(mix*fac)	merge(mix,fac)
6	partially crossed mix-process	D-optimal from (mix*rsm)	optFederov(merge(mix,rsm))
7	constrained mix design	extreme vertices	Xvert(K,LB,UB)

```
rm(list=ls())
set.seed(12345)

x    <- mixexp::SLD(3,3)
x$x12  <- x$x1-x$x2       # first workaround
x$x13  <- x$x1-x$x3
x$x23  <- x$x2-x$x3

res  <- AlgDesign::optBlock(~ -1 +(x1+x2+x3)^3 +
              x1:x2:x12 + x1:x3:x13 + x2:x3:x23 , x,
              c(6,6),criterion="Dpc", nRepeats=1000)
x       <- res$design
x$block <- factor(c(rep("B1",6),rep("B2",6) ))

x$y      <- 20*(x$block=="B2") + 3*x$x1 + 2*x$x2 +
            5*x$x3 + 10*x$x1*x$x2*x$x12 + rnorm(nrow(x),0,0.1)
x$block2 <- as.integer(x$block=="B2") # second workaround

# that won't work as blockB1+blockB2=1
# => confounded with x1+x2+x3=1
summary(res <- (lm(y ~ -1 + block +(x1+x2+x3)^3 +
                x1:x2:x12 + x1:x3:x13 + x2:x3:x23  ,data=x)))
```

```
##
## Call:
## lm(formula = y ~ -1 + block + (x1 + x2 + x3)^3 + x1:x2:x12 +
##     x1:x3:x13 + x2:x3:x23, data = x)
##
## Residuals:
##          1          2          4          5          6          7          3
```

```
##  1.251e-17 -1.431e-17  2.339e-17  1.079e-01 -1.079e-01 -1.002e-17 -7.683e-18 -1.079e-01  1.079
##          9         10
##  6.674e-18 -1.067e-17
##
## Coefficients: (1 not defined because of singularities)
##           Estimate Std. Error t value Pr(>|t|)
## blockB1     5.0703     0.3052  16.613   0.0383 *
## blockB2    24.9924     0.2158 115.811   0.0055 **
## x1         -2.0946     0.3738  -5.604   0.1124
## x2         -2.9911     0.3738  -8.002   0.0791 .
## x3              NA         NA      NA       NA
## x1:x2      -0.3166     1.0857  -0.292   0.8194
## x1:x3       0.3885     0.9403   0.413   0.7505
## x2:x3      -0.5591     0.9711  -0.576   0.6674
## x1:x2:x3    1.5672     5.7401   0.273   0.8303
## x1:x2:x12  10.8875     2.6148   4.164   0.1501
## x1:x3:x13   0.3372     1.9270   0.175   0.8897
## x2:x3:x23  -1.3537     2.3787  -0.569   0.6706
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2158 on 1 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:  0.9998
## F-statistic:  6666 on 11 and 1 DF,  p-value: 0.009553
```

```
kappa(res) # 1.90344e+17 singular design
```

```
## [1] 1.90344e+17
```

```
## but this trick will help
x$block2 <- as.integer(x$block=="B2")
summary(res <- (lm(y ~ -1 + block2 +(x1+x2+x3)^3 +
              x1:x2:x12 + x1:x3:x13 + x2:x3:x23  ,data=x)))
```

```
##
## Call:
## lm(formula = y ~ -1 + block2 + (x1 + x2 + x3)^3 + x1:x2:x12 +
##     x1:x3:x13 + x2:x3:x23, data = x)
##
## Residuals:
##          1          2          4          5          6          7          3         51
## -1.739e-17  2.315e-17 -1.655e-17  1.079e-01 -1.079e-01  9.307e-18 -7.520e-18 -1.079e-01  1.079
##          9         10
##  6.300e-18 -3.478e-17
##
## Coefficients:
##           Estimate Std. Error t value Pr(>|t|)
```

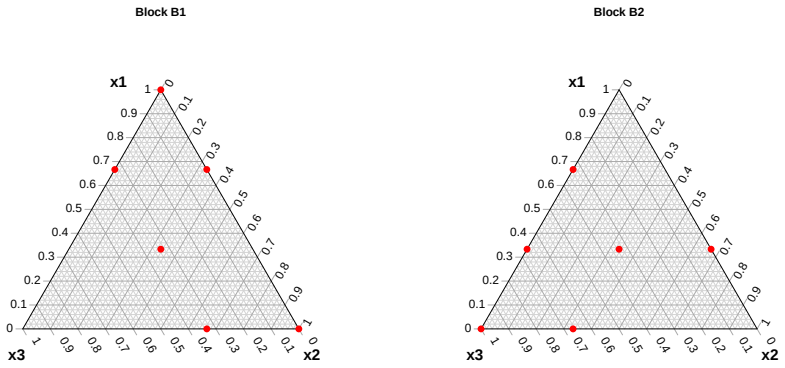

Figure 5.12: Full cubic mixture design in two blocks B1 (left panel) and B2 (right panel)

```
## block2     19.9221     0.2158  92.316   0.0069 **
## x1          2.9757     0.2158  13.789   0.0461 *
## x2          2.0792     0.2158   9.635   0.0658 .
## x3          5.0703     0.3052  16.613   0.0383 *
## x1:x2      -0.3166     1.0857  -0.292   0.8194
## x1:x3       0.3885     0.9403   0.413   0.7505
## x2:x3      -0.5591     0.9711  -0.576   0.6674
## x1:x2:x3    1.5672     5.7401   0.273   0.8303
## x1:x2:x12  10.8875     2.6148   4.164   0.1501
## x1:x3:x13   0.3372     1.9270   0.175   0.8897
## x2:x3:x23  -1.3537     2.3787  -0.569   0.6706
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2158 on 1 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:  0.9998
## F-statistic:  6666 on 11 and 1 DF,  p-value: 0.009553
```

```
kappa(res) # 47.5957
```

```
## [1] 47.5957
```

Chapter 6

Optimization

Many, if not all projects in applied science and industry can be stated as constrained optimization problems. Given a K-dimensional cost function cost=f(x_1,x_2,...x_K) and some functionality, product or customer requirements y_j=g_j(x_1,x_2,...x_K), y_l=g_l(x_1,x_2,...x_K) the goal is finding optimal solutions (conditions) $X^* = x_1^*, x_2^*, ...x_K^*$ satisfying the functionality, product or customer requirements at minimal costs. In the optimization task, formula (6.1), LB_j and UB_j denote the lower and upper bounds of the quality targets while C_l are the targets that must be met exactly in the optimal solution.

$$\begin{gathered} \min_X \left(cost = f(x_1, x_2, ...x_K)\right) \\ subject\ to \\ LB_j \leq y_j = g_j(x_1, x_2, ...x_K) \leq UB_j \\ y_l = g_l(x_1, x_2, ...x_K) = C_l \end{gathered} \tag{6.1}$$

With all functional elements in the optimization problem, (6.1), analytically known, the problem can be solved using a mathematical optimizer - a piece of sophisticated software - as schematically depicted in figure 6.1. Of course, there is no guarantee that a solution will be found, as the problem might be infeasible[1] and conditions meeting the constraints cannot be found.

A major obstacle in many applied projects is that analytical expressions g_j(x_1,x_1,...x_K) are not available or very laborious and expensive to obtain based on first principles, which makes a direct optimization approach inaccessible. However, DoE can be used to empirically derive surrogates of the true, however unknown functions y_j=g_j(x_1,x_2,...x_K) in the domain X and to derive

[1]Projects fail because in the domain X of the optimization problem the constraints are nowhere met. When this happens the optimization problem is called infeasible.

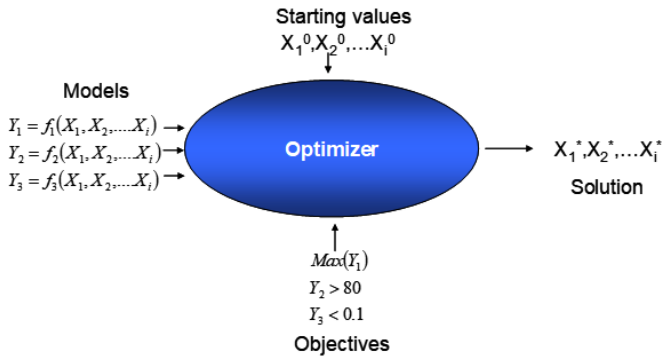

Figure 6.1: Solving a constrained optimization problem with mathematical optimization

local directions of ascend or descend depending on the optimization task. Optimization provides many techniques for navigating through high-dimensional space, some of which will be outlined in the following sections.

6.1 Non-linear functions

Non-linear and multidimensional functions can be divided into convex and non-convex functions with the former comparatively "easy" and the latter "hard" to optimize. Informally, a function is called convex, when all pairs of points on the function's graph can be connected by a straight line without intersecting the function's graph. When this property is lacking the function is called non-convex. This property is schematically depicted in figure 6.2.

Convex functions have, as a direct consequence of these properties, one and only one extremum (or put differently: are twice continuously differentiable) while non-convex functions have several extrema (e.g., three in fig. 6.2, two minima and one maximum). This situation extends to high-dimensional space as the two-dimensional example function, figure 6.3, may indicate. Again, there can be several extrema (maxima, minima and additionally sattle points) and the function may reveal non-smooth domains with the function value changing abruptly in a phase transition like manner (of course, this can also apply to one-dimensional functions y=f(x)). Trying to optimize such functions with global optimization methods can be a nightmare and there is no guarantee that the gobal extremum (e.g. in figure 6.3: the maximum of all maxima) is found[2].

From the figures 6.2 and 6.3 can be concluded that non-convex functions have convex domains which can be locally approximated by convex functions such

[2]For an overview of global optimization methods in R see, e.g., (K.M. Mullen 2014)

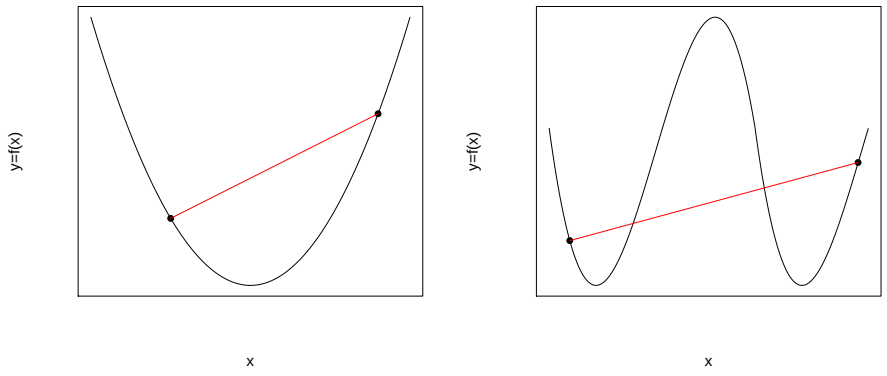

Figure 6.2: Convex (left panel) and non-convex (right panel) functions. Non-convexity on the right panel always imply some pairs of points on the graph which cannot be connected without intersecting the functions graph.

as linear, bilinear and quadratic DoE surrogates. Consequentially, non-convex functions can be optimized by repeatedly solving locally convex optimization problems without requiring the non-linear function $g(x_1,x_2,\ldots x_i)$ to be known analytically.

6.2 Relaxation and sequential optimization

DoE surrogate functions - linear, bilinear and quadratic parametric functions - are convex and can be used to locally probe the landscape of any smooth non-convex function. The individual steps of this approach can be outlined as follows

1. Create an experimental design in a local domain X rendering all linear effects (linear gradients) and some higher order effects estimable with the latter terms providing information about the non-linearity (curvature) of the local domain.
2. Depending on the outcome - linear or non-linear topology - follow either path of action
   - Linear topology: Ascend (or descend depending on objectives) along the gradient by relaxing the local domain in discrete step thereby creating a sequence of promising conditions in term of the optimization goal. Realize these conditions in the laboratory and proceed relaxing as long as there is improvement.
   - Non-linear topology: Augment the design to render all second order

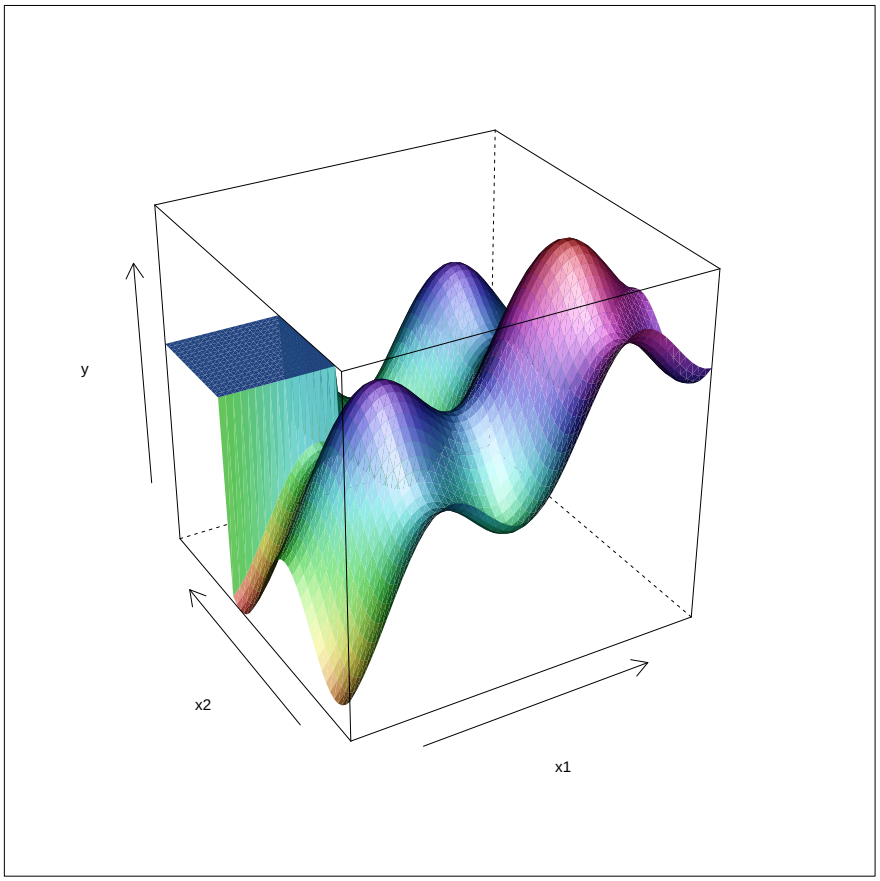

Figure 6.3: Non-linear function with smooth and non-smooth domains in two dimensions

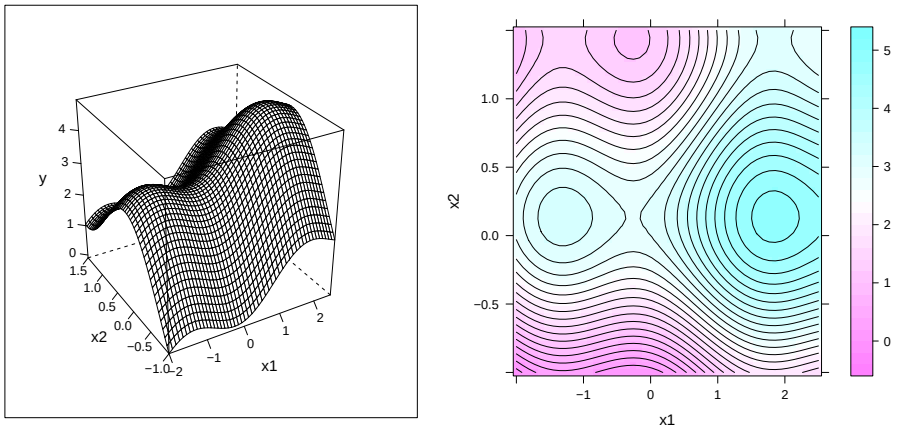

Figure 6.4: Wireframe (left panel) and contour (right panel) plot of a non-convex example function to be "empirically" maximized in the depicted domain x 1 ,x 2 .

effects estimable. Realize the augmentation trials in the lab, check for local extrema and, if possible, ascend (descend) by relaxing the local domain. Realize the relaxation trials in the lab and proceed as long as there is progress.

3. Use the best relaxation trial from the above sequence as a reference point for an new designs and start repeating the above optimization sequence.

This three step procedure will rapidly and with minimal number of experiments approach the next local extremum. The ideas just described will be illustrated by sequentially optimizing the non-linear function depicted in figure 6.4 as both wireframe and contour plot.

Figure 6.4 discerns three local extrema in the domain of interest, that is a marked sattle point between the global maximum in the east and a local maximum in the west. Suppose the aim is finding conditions x_1^*, x_2^* maximizing the response y and further suppose that the project locally starts in the south of the domain as depicted by the solid rectangle in figure 6.5.

As a reasonable start design[3], a full factorial 2x2 design plus center point was created, see the solid rectangle in the south of figure 6.5. Almost certainly the regression model $y = a_0 + a_1 \cdot x_1 + a_1 \cdot x_2 + \epsilon$ will miss the small non-linearities$<\epsilon$ and will report the OLS solution a_1=0 and a_2>>0 thereby suggesting to ascend to the north of x_2. This process called *hypercubical relaxation* can be made in silico by solving a constrained optimization problem in k=1,2,3...N discrete steps

[3] A start design must support all linear effects (the gradient) and should at least support some interactions for providing information on 2_{nd}-order effects in the domain. These requirements naturally suggest full factorial or fractional factorial designs as starting candidates.

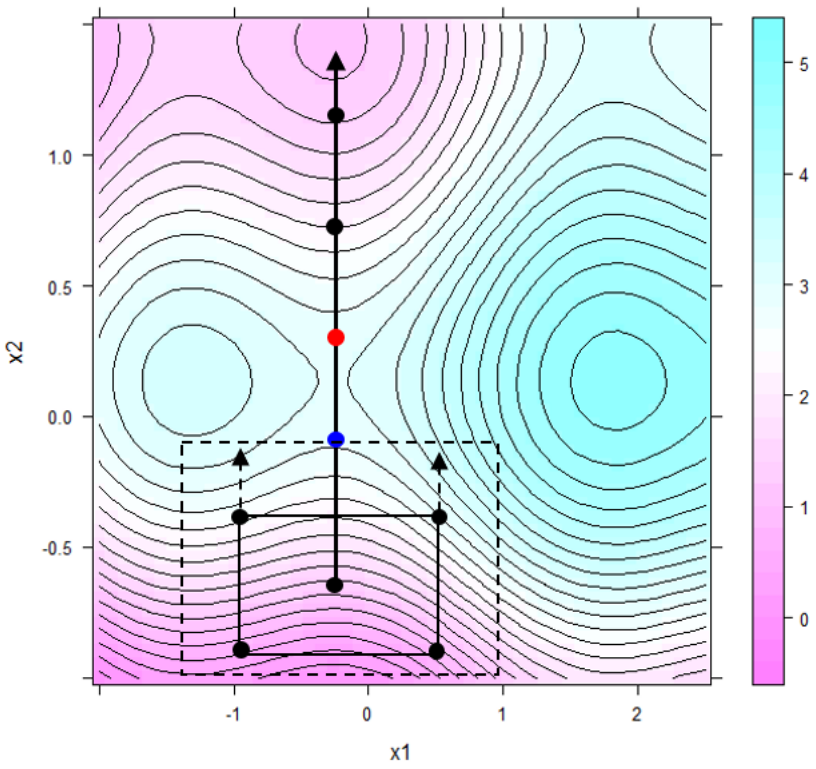

Figure 6.5: Steepest ascend derived from the local topology marked by the solid rectangle in the south. The first relaxation step with the respective relaxation trial marked blue is indicated by the dashed rectangle. The best relaxation trial is labelled red

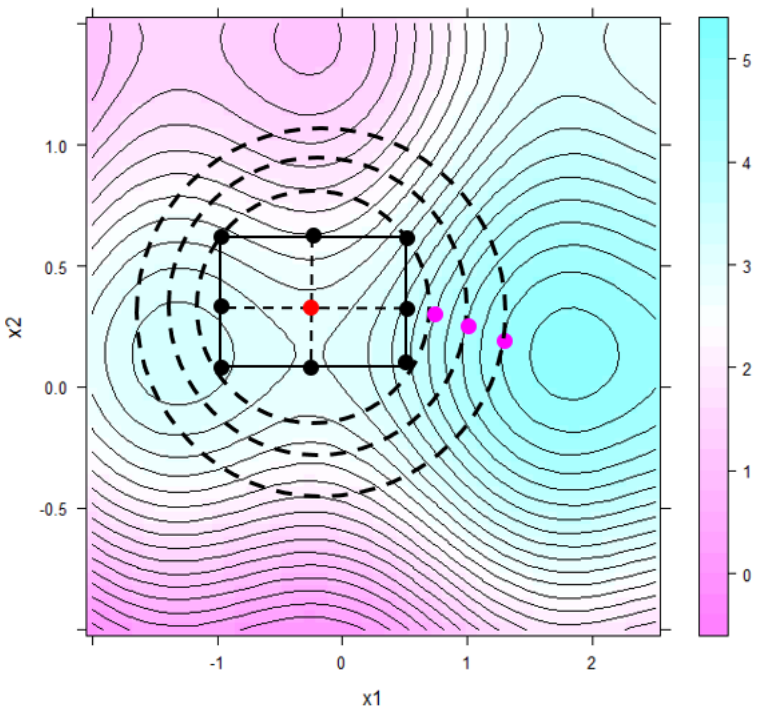

Figure 6.6: Ridge trace relaxation approaching the local optimum (see text for more explanations)

thus generating a sequence of relaxation trials, $X_1^*, X_2^*, ...X_N^*$. With stepsize $\Delta X = \Delta x_1, \Delta x_2$, the lower and upper bounds $LB = [min(x_1), min(x_2)]$ and $UB = [max(x_1), max(x_2)]$, respectively, the k relaxation steps can be formulated as a constrained optimization problem

$$\max_X \Big(a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 \Big)$$

subject to

$$LB - k \cdot \Delta X \leq X \leq UB + k \cdot \Delta X$$

Setting k=1,2,3,4 in figure 6.5 will generate a sequence of four relaxation trials in the direction of and traversing the sattle point from which the results start dropping again.
In order to make further progress, a second factorial around the best relaxation trial is set up. This time, the non-linear topology underlying the second design will render the interaction term significant, and the design will therefore be augmented by star points with the resulting CC design shown in figure 6.6.

The local response surface model will indicate an ascending ridge stretching to the east in figure 6.6. Again, relaxation trials can be generated using hypercubical relaxation as described above or, alternatively, *hyperspherical relaxation* can be used to further exploit the local topology. Rather than relaxing box constraints, hyperspherical relaxation works by relaxing a hypersphere in discrete step. With R denoting the radius of the sphere to be relaxed and stepsize Δr, the relaxation process can be written concisely

$$\max_X \left(a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 + a_{12} \cdot x_1 \cdot x_2 + a_{11} \cdot x_1^2 + a_{22} \cdot x_2^2 \right)$$

subject to

$$\sum_{i=1}^{2} x_i^2 \leq (R + k \cdot \Delta r)^2$$

Optimization with hyperspherical constraints is also known in the literature as *ridge trace analysis* and can be done with the R function steepest() in the package *rsm*. Usually the results from hypercubical and hyperspherical relaxation are similar with the former being located at the vertices of the domain and the latter more centered.
All concepts discussed so far are put together in the following R-code, namely the following four parts

1. The first code segement #1 creates a response surface with an ascending ridge stretching South-West from the model y = 90 - $8x_1$ - $40{x_1}^2$ - $8x_2$ - $80{x_2}^2$ + $80x_1x_2$ + N(0,2).
2. Then the ridge trace of steepest ascend (= set of hyperspherical relaxation trials) is calculated with the function steepest() from the package *rsm*.
3. Part #3 uses the convex optimizer solnp() from the R-package *Rsolnp* for creating (here) 20 hypercubical relaxation trials with a step size of (here) 2.5% of the initial X-space. The function solnp() can be extended by any smooth equality and inequality constraints if nescessary. The package can also be used for optimizing non-convex functions by randomly restarting the optimizer at different locations in X so as to increase the chance of finding the global optimum.
4. Finally, the hypercubical and hyperspherical traces are plotted along with the contour lines of the model in figure 6.7.

```
rm(list=ls())
library(lattice)
library(rsm)
library(pals)
set.seed(123)
############ 1: create a simple ascending ridge in x1,x2
#############################################################
x.grid <- expand.grid(x1= seq(0,1,length=10),
```

```
                  x2= seq(0,1,length=10)  )
x.grid$y <- 90  - 8*x.grid$x1 - 40*x.grid$x1^2 -
            8*x.grid$x2 - 80*x.grid$x2^2 +
                  80*x.grid$x1*x.grid$x2 +
            rnorm(nrow(x.grid),0,2)
res.lm <- lm(y ~ x1 +x2 + I(x1^2) + I(x2^2) + x1:x2, data=x.grid)

########## 2: rsm modeling + ridge trace analysis with rsm()
############################################################
x.coded  <- coded.data(x.grid, x1.c~(x1-0.5)/0.25,
                       x2.c ~ (x2-0.5)/0.25)
res.rsm  <- rsm(y ~ SO(x1.c, x2.c ),   data=x.coded)
ridge <- steepest(res.rsm, dist = seq(0,3,0.1),
          descent=FALSE) #descent=F: max
```

```
## Path of steepest ascent from ridge analysis:
```

```
head(ridge)
```

```
##   dist   x1.c   x2.c |      x1      x2 |   yhat
## 1  0.0  0.000  0.000 | 0.50000 0.50000 | 72.153
## 2  0.1 -0.021 -0.098 | 0.49475 0.47550 | 73.321
## 3  0.2 -0.050 -0.194 | 0.48750 0.45150 | 74.415
## 4  0.3 -0.088 -0.287 | 0.47800 0.42825 | 75.442
## 5  0.4 -0.134 -0.377 | 0.46650 0.40575 | 76.411
## 6  0.5 -0.187 -0.464 | 0.45325 0.38400 | 77.332
```

```
############################## 3: hypercubical relaxation
############################## using Rsolnp

library(Rsolnp)

x.var    <- c("x1","x2")
y.var    <-  "y"
x.mean   <- sapply(x.grid[,x.var],mean,na.rm=T)
x.lb     <-  sapply(x.grid[,x.var],min,na.rm=T) # LB,UB
x.ub     <-  sapply(x.grid[,x.var],max,na.rm=T)
delta    <- (x.ub-x.lb)/40 # stepsize 2.5% of X

df.collect  <- NULL

objective   =function(x) {
  xx               <- data.frame(rbind(x))
  colnames(xx)     <- x.var
  return( -predict(res.lm,xx)  ) # max(x) = min(-x)
}
```

```
for (i in (1:20))   {

  xx.lb <- c(0.5,0.5) - i*delta
  xx.ub <- c(0.5,0.5) + i*delta

  res.nlp  <- solnp(fun=objective , LB=xx.lb,UB=xx.ub,
                pars=x.mean, control=list(trace=0))

  solution           <- rbind(res.nlp$pars)
  colnames(solution) <- x.var
  df.collect         <- rbind(df.collect, data.frame(relax.step=i,
                        y.at.solution = -objective(solution) ,
                        rbind(solution) )  )
}
rownames(df.collect)  <- NULL
head(df.collect)
```

```
##   relax.step y.at.solution        x1    x2
## 1          1      73.51149 0.4750001 0.475
## 2          2      74.82308 0.4500000 0.450
## 3          3      76.08742 0.4250001 0.425
## 4          4      77.30450 0.4000001 0.400
## 5          5      78.47434 0.3750001 0.375
## 6          6      79.59693 0.3500001 0.350
```

```
# 4: plot hyperspherical + hypercubical trace together
# on 2D contour plot with 3 constraints superimposed

x.pred <- expand.grid(x1= seq(0,1,length=50),
                      x2= seq(0,1,length=50)  )
x.pred$y <- predict(res.lm,x.pred)
p  <- contourplot(y ~ x1*x2, data=x.pred, cuts=50,
                  labels=F,pretty=T,region=T)
update(p, panel = function(...){
  panel.contourplot(...)
  panel.xyplot(0.5,0.5, pch=16,col="black", cex=1)
  panel.abline(v=0.5,h=0.5, col="black", lty=2)
  panel.xyplot(df.collect[,3],df.collect[,4] ,
               col="blue", pch=16)
  panel.xyplot(ridge[-1,5],ridge[-1,6], col="red", pch=16)
  panel.lines(ridge[-1,5],ridge[-1,6], col="red")
  panel.lines(df.collect[,3],df.collect[,4] , col="blue")
  panel.xyplot(0.5,0.5, pch=1,col="red", cex=4.5)
  panel.xyplot(0.5,0.5, pch=1,col="red", cex=23)
  panel.xyplot(0.5,0.5, pch=1,col="red", cex=39)
  panel.lines(c(0.4,0.4,0.6,0.6,0.4),
```

```
              c(0.4,0.6,0.6,0.4,0.4) , col="blue")
  panel.lines(c(0.35,0.35,0.65,0.65,0.35),
              c(0.35,0.65,0.65,0.35,0.35) , col="blue")
  panel.lines(c(0.225,0.225,0.775,0.775,0.225),
              c(0.225,0.775,0.775,0.225,0.225) , col="blue")
})
```

As revealed in figure 6.7, the ridge trace (in red) is always perpendicular on the curved contour lines of the model and becomes thus curved, different from the straigth line (in blue) of the hypercubical relaxation trace. Generally, the difference between hypercubical and hyperspherical relaxation is often small with the former considered more extreme and the latter more cautious. However, choosing an appropriate stepsize is often more important than choosing the relaxation method, as the stepsize directly affects the degree of extrapolation into an unknown domain. Here, the following rules of thumb apply

1. If the model to be relaxed is linear, bold steps, say $\Delta = 20$ *or* 30% (or even larger), can be taken after having checked the feasibility of the stepsize with the scientists and technicians.
2. If the model is non-linear (bilinear or quadratic), stepsizes should be choosen smaller (say 1, 2... i% of the initial design space) due to detrimental edge effects of higher order polynomials.

The optimization method just described is in essence a sequential quadratic programming method: The unknown and potentially non-convex objective function is approximated locally by a linear polynom and, if needed, updated by higher order terms and subsequentially relaxed as long as there is progress in terms of the optimization goals. This process comes to a stop when the next local optimum is reached. Global optimality can only be checked by restarting the process just described at a different location in the domain and see whether the solution will converge to a better solution. In empirical optimization projects local optimality in the strict sense is often not nescessary and practical optimality will be sufficient. Relaxation trials meeting the requirements are, to be sure, mathematically not optimal but of practical optimality. Unfortunately, the opposite is also true: Local optima with vanishing first derivatives may not meet the practical optimization goals. When a local optimum misses the practical requirements, it is best to look for new degrees of freedom as new levers to further optimize the system.

6.3 Constrained optimization

So far one objective function was optimized subject to box constraints with the latter being subsequently relaxed in discrete steps as an effective way of multidimensional hill-climbing.
However, optimization has more to offer than just optimizing one objective function subject to box constraints. It can be used to study more complex

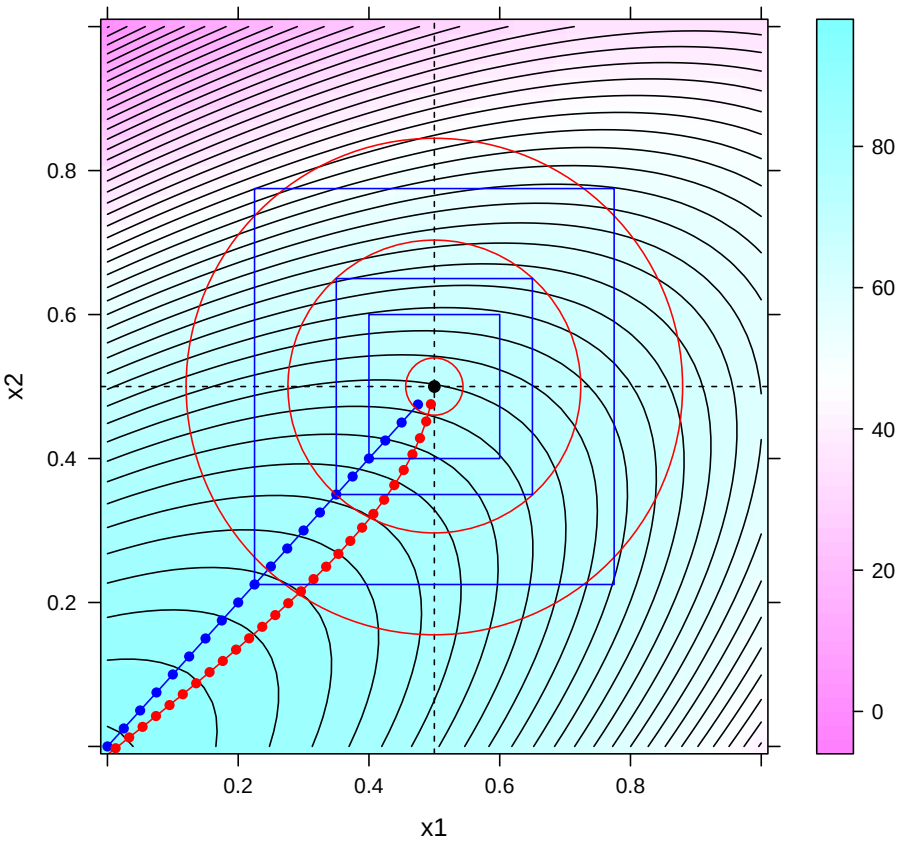

Figure 6.7: Hyperspherical (red) and hypercubical (blue) relaxation traces on an ascending ridge with three hypercubical and hyperspherical constraints superimposed

multidimensional problems hard to oversee otherwise.
The following mixture example is taken from (John A. Cornell 1990), p. 80, and aims at demonstrating the use of constrained optimization for analysing the cost and taste structure of a fruit punch. The design consists of 10 different mixtures and is depicted on the right panel in figure 6.8. The taste of each mixture as the response is scored by three different test persons thus making a total of 30 runs. The data can be analyzed easily using functionalities from the package *mixexp* (see chapter 5 for more details)., see figure 6.8 for a response surface plot and the modeling details in the R output. Lack of fit[4], p_{lof}=0.04, is hardly significant, and the model can be asssumed to unbiasedly describe the data within replication error.

```
rm(list=ls())
library(Ternary)
library(mixexp)

melon  <- scan(text="1  1   1   0.5 0.5 0.5 0   0   0   0   0   0
              0 0   0   0.5 0.5 0.5 0.33    0.33    0.33
              0.72  0.72    0.72    0.14    0.14    0.14    0.14    0.14    0.14")
pine  <- scan(text="0   0   0   0.5 0.5 0.5 1   1   1   0.5 0.5
                   0.5 0   0   0   0   0   0   0.33    0.33    0.33    0.14
                   0.14    0.14    0.57    0.57    0.57    0.29    0.29    0.29")
orange <- scan(text="0  0   0   0   0   0   0   0   0   0.5 0.5 0.5 1
                   1   1   0.5 0.5 0.5 0.33    0.33    0.33    0.14
                   0.14    0.14    0.29    0.29    0.29    0.57    0.57    0.57")
taste <- scan(text="4.8 4.3 4.7 6.1 6.3 5.8 6.3 6.5 6.2
                   6.1 6.2 6.2 7.4 6.9 7   5.9 6.1 6.5 6.4 6   5.8
                   6.6 5.4 5.8 5.6 5.7 5   6.4 5.2 6.4")

x <- data.frame(melon,pine,orange,taste)

res <- MixModel(x, "taste",
      mixcomps = c("melon", "pine", "orange"), model = 2)
```

```
##
##              coefficients   Std.err    t.value         Prob
## melon            4.773348 0.2383417 20.0273285 2.220446e-16
## pine             6.266015 0.2473754 25.3299816 0.000000e+00
## orange           7.107707 0.2473754 28.7324717 0.000000e+00
## pine:melon       2.155899 1.1380880  1.8943163 7.029560e-02
## orange:melon     1.105927 1.1380880  0.9717414 3.408717e-01
## pine:orange     -3.529823 1.0195672 -3.4620799 2.023323e-03
##
## Residual standard error:  0.4351875  on  24 degrees of freedom
```

[4]Note that the Lof statistics is obtained by using the function rsm::rsm() on the slack variable model

```
## Corrected Multiple R-squared:  0.6714546
```

```
   # reduced cubic

a <- summary(rsm::rsm(taste~ FO(melon,orange) + TWI(melon,orange)
          + PQ(melon,orange), data=x))
a$lof # p.lof=eport lof : p.lof=0.042
```

```
## Analysis of Variance Table
##
## Response: taste
##                     Df Sum Sq Mean Sq F value    Pr(>F)
## FO(melon, orange)    2 6.1983 3.09915 16.3765 3.271e-05 ***
## TWI(melon, orange)   1 0.1531 0.15307  0.8089  0.377393
## PQ(melon, orange)    2 2.9414 1.47071  7.7715  0.002499 **
## Residuals           24 4.5419 0.18924
## Lack of fit          4 1.7152 0.42880  3.0340  0.041640 *
## Pure error          20 2.8267 0.14133
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
par(mfrow=c(1,2))
plot(predict(res), rstudent(res),
     xlab=expression( paste("predicted response ", hat(y)) )    ,
     ylab="studentized residuals",type="n",
     main=expression(paste("(",frac(paste("y - ",
          hat(y)),sigma), ") ~ ", hat(y)   )) )
colcol <- rep("black",nrow(x))
colcol[duplicated(x[,1:3])]  <- "red"
text(predict(res), rstudent(res),
     col=colcol, label=(1:nrow(x)))
abline(h=c(-2,0,2),lty=c(2,1,2))
grid()

TernaryPlot(atip="melon", btip="pineapple", axis.cex=1.2,lab.cex = 1.3,isometric=F,
            ctip="orange",axis.labels=seq(0,1,0.1))
FunctionToContour <- function (a, b, c) {
  xx <- data.frame(a,b,c)
  colnames(xx) <- c("melon", "pine", "orange")
  return(predict(res,xx))
  }
values            <- TernaryPointValues(FunctionToContour,
                     resolution=24)
ColourTernary(values)
TernaryContour(FunctionToContour, resolution=36)
AddToTernary(points, x[,1:3],pch=16,col="red",cex=1.2)
```

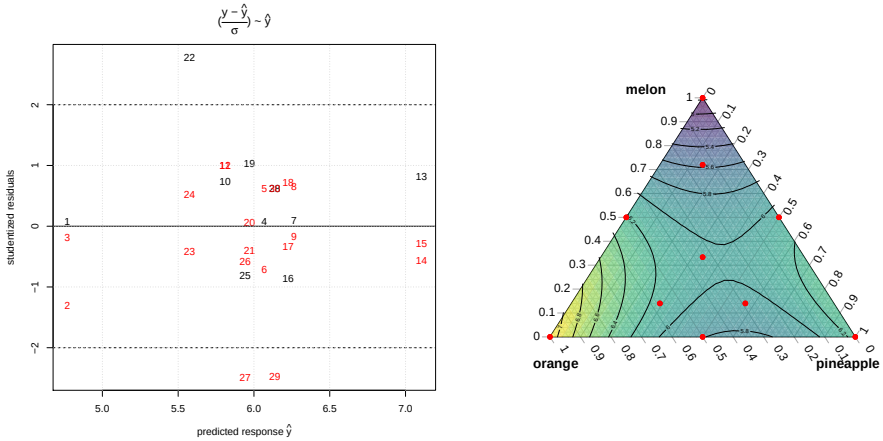

Figure 6.8: Residual analysis (left panel) and contour plot (right panel) from Cornells fruit punch data after being modelled with a quadratic Scheffe model

According to the modeling results, pure orange juice scores best followed by pineapple and watermelon. Pineapple and orange juice reveal an antagonistic interaction rendering binary pineapple orange mixtures less tasty. These conclusions will be further detailed by taking the cost function of the three components into account and setting up a conditional optimization problem. With $c_{i=1,2,3} = 1, 2, 3$ denoting the cost of the three components $x_i, i = 1, 2, 3$ (melon, pineapple, orange), the conditional optimization problem becomes

$$\min_X \sum_{i=1}^{3} (c_i \cdot x_i)$$

$$subject\ to$$

$$\sum_i x_i = 1$$

$$taste.LB_k \leq taste = f(x_i) \leq 10$$

The following R-code is an implementation of this optimization problem with taste.LB_k being varied in the range of 5.5-7.1. Results are listed in table 6.1, and the optimal results are traced in figure 6.9.

```
library(Rsolnp)

x.var    <- c("melon", "pine", "orange")
y.var    <-  "taste"
x.mean   <- sapply(x[,x.var],mean,na.rm=T)
```

```
x.lb     <-  sapply(x[,x.var],min,na.rm=T) # LB,UB
x.ub     <-  sapply(x[,x.var],max,na.rm=T)
delta    <-  seq(5.5,7.1,0.2) # sequence of lower bounds for taste

df.collect  <- NULL

taste   = function(x) {
  xx               <- data.frame(rbind(x))
  colnames(xx)     <- x.var
  return( predict(res,xx)  ) # max(x) = min(-x)
}

cost = function(x) {
  return(x[1]+ 2*x[2]+ 3*x[3] )
}


for (i in (1:length(delta)) )   {

  res.nlp     <- solnp(fun=cost , LB=x.lb,UB=x.ub,
                       pars=x.mean, eqfun=function(x) {sum(x)},eqB=1,
                      ineqfun=taste, ineqLB=delta[i],
                      ineqUB=10,control=list(trace=0))
  solution     <- (res.nlp$pars)
  df.collect   <- rbind(df.collect, data.frame(step=i,
                           cost.opt = round(cost(solution),3),
                           taste.LB=delta[i],
                           round(rbind(solution),3),
                           taste.opt=round(taste(solution),3),
                           RC=res.nlp$convergence )  )
}
rownames(df.collect)  <- NULL

knitr::kable(
  df.collect, booktabs = TRUE,row.names=F,
  caption = 'Results from taste-constrained cost
  minimization of the fruit juice model')
```

```
TernaryPlot(atip="melon", btip="pineapple", ctip="orange",axis.cex=0.7,lab.cex = 0.8,
            axis.labels=seq(0,1,0.1))
AddToTernary(points, df.collect[,4:6],pch=16,col="red", cex=1)
TernaryArrows(df.collect[1,4:6],   df.collect[3,4:6],
              col="red",lty=1,lwd=2,length=0.1)
TernaryArrows(df.collect[3,4:6],   df.collect[4,4:6],
              col="red",lty=1,lwd=2,length=0.1)
```

Table 6.1: Results from taste-constrained cost minimization of the fruit juice model

step	cost.opt	taste.LB	melon	pine	orange	taste.opt	RC
1	1.229	5.5	0.771	0.229	0.000	5.5	0
2	1.308	5.7	0.692	0.308	0.000	5.7	0
3	1.402	5.9	0.598	0.402	0.000	5.9	0
4	1.895	6.1	0.553	0.000	0.447	6.1	0
5	2.064	6.3	0.468	0.000	0.532	6.3	0
6	2.250	6.5	0.375	0.000	0.625	6.5	0
7	2.457	6.7	0.271	0.000	0.729	6.7	0
8	2.697	6.9	0.151	0.000	0.849	6.9	0
9	2.992	7.1	0.004	0.000	0.996	7.1	0

```
TernaryArrows(df.collect[4,4:6],   df.collect[9,4:6],
              col="red",lty=1,lwd=2,length=0.1)
FunctionToContour <- function (a, b, c) { a + 2*b + 3*c  }
values            <- TernaryPointValues(FunctionToContour, resolution=24)
ColourTernary(values)
TernaryContour(FunctionToContour, resolution=36)
```

The optimal trace plotted in figure 6.9 describes the cheapest mixtures with an expected taste of at least taste.LB_k. The mixtures found are entirely binary in nature: For taste.LB<5.9 pineapple-melon mixtures are cheapest and better tasting candidates are entirely made of orange-watermelon with watermelon subsequently replaced by orange in the further optimization course. In all nine solutions the lower bound condition is active, i.e. at the optimal solution the equality $taste.LB_k = taste.LB_k^*$ holds, and taste actively constrains the cost function. Put differently: By relaxing the lower taste constraint, that is here, lowering taste.LB, the objective function will further decrease.

6.4 Multiresponse optimization

In real-world projects, there are often many responses, formally $y_j=g_j(x_i)$, and scientists tend to state their project goals as a multiresponse task such as $\max(y_1,y_2,\ldots y_k)$, $\min(y_{k+1},y_{k+2},\ldots y_K)$. In a strict mathematical sense, such statements are not proper optimization statements because a solution X_i^* maximizing $y_1 = f(X_i^*)$ does not nescessarily minimize y_2 at the same location X_i^*. However, the above multiresponse optimization statements are acceptable by convention and can be tweaked into proper optimization problems.
Prior to implementing a multiresponse optimization problem, the joint correlation structure of the raw data $y_1,y_2,\ldots y_k$ should be analyses thoroughly. For

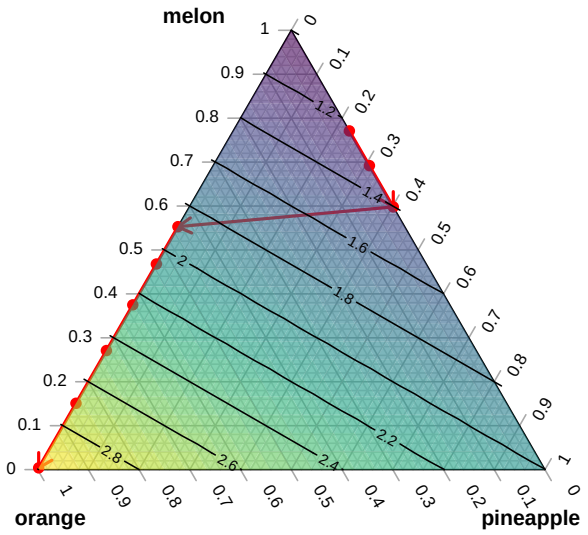

Figure 6.9: Contour plot at minimal costs as a function of increasing taste with direction of increase indicated by arrows

instance, if y_1 and y_2 turn out to be negatively correlated and the aim is joint maximization of both responses, any attempt to maximize both y_1 and y_2 over X **will fail**. Whatever the optimizer does for maximizing y_1 will inevitably minimize y_2, and including such conflicting responses in a joint optimization problem is not a good advice. Conflicts of aims have there cause in the underlying degrees of freedom (DF) (the X-factors). When there are conflicts of aims it is usually better to stop working with the present DFs and to look for other, more suitable DFs. The ideal DF in a multiresponse optimization problem is a factor affecting one response while leaving the other responses unaffected.
However, if the outcome of exploratory data analysis, here analysis of the joint distribution of y_1 and y_2, is similar to figure 6.10, the "outliers" labelled red should be replicated, as they may indicate conditions in accordance with the joint maximization goal. In this case, rather than trying to optimize $y_{1,2}=f_{1,2}(x_i)$, the X-conditions of the "outliers" should be taken as a reference (center) point for a new design which may now lead to a potentially feasible optimization problem.

```
rm(list=ls())
set.seed(123334)
x <- seq(1,10,length=10)
y <- -1*x + rnorm(length(x),0,1)
plot(x,y,type="p",axes=F,pch=16,
xlab=expression(plain(y) [1]),
ylab=expression(plain(y) [2]))
abline(lm(y~x)$coef)
points(c(10,9,9.5),c(-2,-2.5,-3),col="red",pch=16)
box()
```

Now, assume for the time being, that conflicting responses are not presents, and the responses are sufficiently independent to render the optimization problem feasible.
One way of optimizing a multiresponse system is similar to the technique described in the previous section and boils down to rewritting the multipresponse problem into a constrained optimization problem with the constraint boundaries made flexible. For instance, the simple statement $\max(y_1,y_2,y_3)$ could be rewritten into

$$\max_X(y_1 = f_1(X))$$

$$subject\ to$$

$$y_2 = f_2(X) > LB_2$$

$$y_3 = f_3(X) > LB_3$$

By successively increasing LB_i and LB_k, the optimization will be driven to a direction simultaneously maximizing y_1,y_2,y_3 until the solutions become infeasible. When joint minimization is the aim, the problem simply becomes

$$\min_X(y_1 = f_1(X))$$

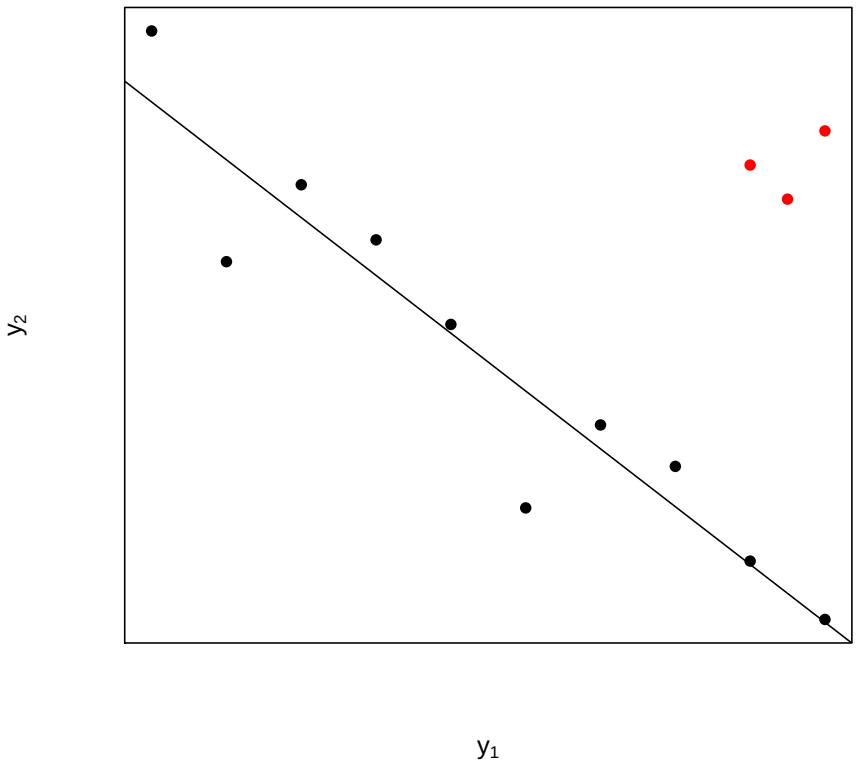

Figure 6.10: Joint distribution y_1, y_2 with promising "outliers" labelled red given goal $\max_X [y_1 = f_1(X), y_2 = f_2(X)]$.

$$\textit{subject to}$$

$$y_2 = f_2(X) < UB_2$$

$$y_3 = f_3(X) < UB_3$$

to the effect that the solution is driven to areas in X-space jointly minimizing y_1,y_2,y_3 When there are many responses to optimize this approach can becomes awkward and a more convenient method is obtained by using convex penalties. The idea is simple and expressed in formula (6.2)

$$z(x_1, x_2, ...x_i) = \sum_{j=1}^{K} \Big(\frac{y_j - T_j}{UB_j - LB_j} \Big)^2 = \sum_{j=1}^{K} \Big(\frac{f_j(x_1, x_2, ...x_i) - T_j}{UB_j - LB_j} \Big)^2 \qquad (6.2)$$

with T_j being the desired target values for response y_j and LB_j,UB_j the corresponding lower and upper acceptance boundaries. Depending on the optimization problem, the penalty,(6.2), can be made half-sided with the right sided penalty $z()^+$ given by

$$z(x_1, x_2, ...x_i)^+ = \sum_{j=1}^{K} max\Big(0, \frac{f_j(x_1, x_2, ...x_i) - T_j}{UB_j - LB_j}\Big)^2$$

and the corresponding left sided version $z()^-$

$$z(x_1, x_2, ...x_i)^- = \sum_{j=1}^{K} min\Big(0, \frac{f_j(x_1, x_2, ...x_i) - T_j}{UB_j - LB_j}\Big)^2$$

Figure 6.11 shows two symmetric penalty terms with narrow and wide acceptance ranges and two half-sided penalties, $z()^{+,-}$. Note how the width of the acceptance range affects the shape and the penalty incured by deviating from the target.

With the objective, formula (6.2), the multiresponse optimization simply becomes

$$\min_X z(x_{1,}, x_2, ...x_i)$$

Sums of convex functions are again convex, and $z(x_{1,}, x_2, ...x_i)$ can therefore be optimized with any convex solver. Here we demonstrate the use of the quasi-global solver gosolnp() from the package *Rsolnp* for optimizing three scenarios of the fruit punch example from above[5], namely

[5] n.sim=10000 creates 10000 points in mixture space and evaluates the objective function. The optimizer is then initialized with the X set thus found. The option n.restarts=5 repeats this process five times, so in total 50000 function evaluation were performed to generate "good" local starting values. Strictly, gosolnp() is not necessary for the objective $z(x_{1,}, x_2, ...x_i)$ (solnp() would be sufficient for z()) but shown here as an example for approaching global optimization problems.

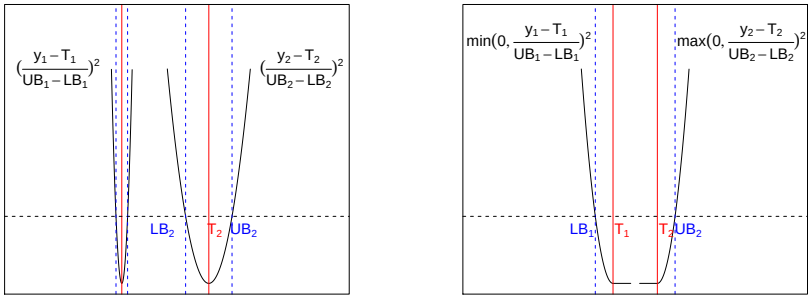

Figure 6.11: Convex penalty functions with narrow and wide acceptance range UB i -LB i (left panel) and half-sided convex penalties (right panel)

1. Scenario #1: T_{cost}=1.2 +/- 0.001 ; T_{taste}=7 +/- 0.1
2. Scenario #2: T_{cost}=1.2 +/- 0.2 ; T_{taste}=7 +/- 0.0001
3. Scenario #3: T_{cost}=2 +/- 1 ; T_{taste}=2 +/- 1

The scenarios differ by putting different weight to the individual terms of the objective function. In scenario #1, e.g., much weight is put on the cost target, cost=1.2 and, as a consequence, the desired target is realized in the solution (see R output from scenario #1).

```
rm(list=ls())

melon  <- scan(text="1  1   1   0.5 0.5 0.5 0   0   0   0   0   0
             0 0   0   0.5 0.5 0.5 0.33    0.33    0.33
             0.72  0.72    0.72    0.14    0.14    0.14    0.14    0.14    0.14")
pine  <- scan(text="0   0   0   0.5 0.5 0.5 1   1   1   0.5 0.5
                  0.5 0   0   0   0   0   0   0.33    0.33    0.33    0.14
                  0.14    0.14    0.57    0.57    0.57    0.29    0.29    0.29")
orange <- scan(text="0  0   0   0   0   0   0   0   0   0.5 0.5 0.5 1
                  1   1   0.5 0.5 0.5 0.33    0.33    0.33    0.14
                  0.14    0.14    0.29    0.29    0.29    0.57    0.57    0.57")
taste <- scan(text="4.8 4.3 4.7 6.1 6.3 5.8 6.3 6.5 6.2
                  6.1 6.2 6.2 7.4 6.9 7   5.9 6.1 6.5 6.4 6   5.8
                  6.6 5.4 5.8 5.6 5.7 5   6.4 5.2 6.4")

x <- data.frame(melon,pine,orange,taste)
```

```
x.var    <- c("melon", "pine", "orange")
y.var    <-  "taste"
x.mean   <-  sapply(x[,x.var],mean,na.rm=T)
x.lb     <-  sapply(x[,x.var],min,na.rm=T) # LB,UB
x.ub     <-  sapply(x[,x.var],max,na.rm=T)

res <- lm( taste~ (melon + pine + orange)^2,  data=x)

taste   = function(x) {
  xx               <- data.frame(rbind(x))
  colnames(xx)     <- x.var
  return( as.numeric(predict(res,xx)) )
}

cost = function(x) {
   return(as.numeric(x[1]+ 2*x[2]+ 3*x[3]))
}

penalty <- function(x) {
  sum( ( (c(cost(x),taste(x) ) -t)/(ub-lb) )^2 )
}

######### scenario 1
lb <- c(1.1999 , 6.9)
# min(cost)(<=>cost=1.2) given taste
# =>  melon=0.8, pine=0.2
t  <- c(1.2 , 7)
ub <- c(1.2001 , 7.1)

system.time(res.nlp <- Rsolnp::gosolnp(fun=penalty ,
                     LB=x.lb,UB=x.ub, par=x.mean,
                     eqfun=function(x) {sum(x)},eqB=1,  n.restarts = 5,
                     n.sim = 10000, rseed=12345,control=list(trace=0)))
```

```
##    user  system elapsed
##   29.93    0.66   31.22
```

```
res.nlp$pars
```

```
##        melon         pine       orange
## 7.999958e-01 2.000042e-01 5.638882e-11
```

```
cost(res.nlp$pars)
```

```
## [1] 1.200004
```

```
taste(res.nlp$pars)
```

```
## [1] 5.409904
```

```
################################### scenario 2
lb <- c(1 , 6.9999)
# max(taste)(<=>taste=7) given cost;
# => melon=0.08, orange=0.92,
t  <- c(1.2 , 7)
ub <- c(1.4 , 7.0001)

system.time(res.nlp <- Rsolnp::gosolnp(fun=penalty ,
                      LB=x.lb,UB=x.ub, par=x.mean,
                      eqfun=function(x) {sum(x)},eqB=1,  n.restarts = 5,
                      n.sim = 10000, rseed=12345,control=list(trace=0)))
```

```
##    user  system elapsed
##   29.67    0.61   30.58
```

```
res.nlp$pars
```

```
##        melon         pine       orange
## 8.070163e-02 1.263921e-10 9.192984e-01
```

```
cost(res.nlp$pars)
```

```
## [1] 2.838597
```

```
taste(res.nlp$pars)
```

```
## [1] 7
```

```
################################### scenario 3

lb <- c(1 , 5)
# max(taste)(<=>taste=7) given cost;
# => melon=0.33, pine=0.33, orange=0.33,
t  <- c(2 , 6)
ub <- c(3 , 7)

system.time(res.nlp <- Rsolnp::gosolnp(fun=penalty ,
                      LB=x.lb,UB=x.ub, par=x.mean,
                      eqfun=function(x) {sum(x)},eqB=1,  n.restarts = 5,
                      n.sim = 10000, rseed=12345,control=list(trace=0)))
```

```
##    user  system elapsed
##   29.33    0.67   30.41
```

```
res.nlp$pars
```

```
##     melon      pine    orange
## 0.3319041 0.3361917 0.3319042
```

```
cost(res.nlp$pars)
```

```
## [1] 2
```

```
taste(res.nlp$pars)
```

```
## [1] 6
```

The conclusion from these optimization scenarios agree with the results from constrained optimization: Tasty mixtures are expensive and are made primarily of (expensive) orange juice with watermelon acting as a cheap filler.
Cheap mixtures are less tasty blends of cheap watermelon with cheap pineapple.
Overall, in the fruit punch example there are no degrees of freedom to simultaneously minimize cost and maximize taste, because the expensive component is also the one required for good taste. An average mixture balancing cost and taste is the centroid point suggested by the third optimization scenario.

6.5 Optimization of derived responses

The previous sections discussed joint and multidimensional optimization problems over a set of measured responses. However, in the physical and chemical sciences optimization problems are often stated in terms of derived responses, these are functions of measured responses, $z=f(y_1, y_2, \ldots y_j)$. For instance, in chemistry, the goal can often be stated as: "Find conditions X^* maximizing the sum of the main product while minimizing the sum of the by-products", formally

$$\max_X \sum_{j=1}^{N_1} mp_j(x_1, x_2 ... x_i)$$

$$and$$

$$\min_X \sum_{j=1}^{N_2} bp_j(x_1, x_2 ... x_i)$$

It is a widespread habit among experimentally working scientists, especially in times of readily available spreadsheet programms, to functionally convert the raw measured responses y_j into derived responses $z_k=f_k(y_j)$ and treat these variables subsequently as if they were measured responses. However, empirical model building and optimization should never be based on derived responses, rather it should always be based on measured responses. There are two good reasons for this advice, namely

1. The measured values y_j are random variables with variances σ_j^2 as a consequence of $y_j = f_j(x_i) + \epsilon_j;\ \epsilon_j \sim N(0, \sigma_j^2)$. According to the laws of statistics,[6] aggregation of several responses into a derived response will

[6] see (J.G. Kalbfleisch 1985), Vol. I, pp. 182-187.

inflate the error of the new response. For instance, simply adding responses, $\sum_i y_i$, amounts to adding the variance of the individual responses, $Var(\sum_{j]} y_j) = \sum_j \sigma_j^2$ (see[7]). Now, the variances of the models $\hat{y}_j = f_j(x_i)$ are much smaller than $\mathrm{Var(y_j)}$[8], therefore summing $\hat{y}_j$ is to be prefered over summing y_j.

2. When $y_j = f_j(x_i) + \epsilon_j$ is linear in $\mathrm{x_i}$, transformation $\mathrm{z=f(y_j)}$ will likely render z non-linear in $\mathrm{x_i}$ in this way making empirical building more rather then less complex. It is therefore adviced to do the model building step with the measured responses only and use the parametric surrogates $\hat{y}_j = g_j(x_i)$ for defining the derived responses $z(x_i) = f(g_j(x_i))$. Irrespective of whether z is convex or non-convex, and it can become non-convex esily, optimizers such as gosolnp() can deal with such non-linearities and find reliable solutions.

The following R-simulation makes the above arguments transparent. First two responses, $y_{1,2} = f_{1,2}(x_1, x_2) + \epsilon_{1,2}$; $\epsilon_{1,2} \sim N(0, \sigma_{1,2}^2) = 1$; $Cov(\epsilon_1, \epsilon_2) = 0$, are defined with both being linear in $\mathrm{x_i}$. Then a realization of the derived response $\mathrm{z(x_1,x_2)}$ is created from the "experimental" values with the transformation $z = 100 - \frac{y_1}{y_2}$. The "true" z, here called Z, with $Z = 100 - \frac{f_1(x_1,x_2)}{f_2(x_1,x_2)}$ is shown in figure 6.12.

```
rm(list=ls())
set.seed(123)
x <- expand.grid(x1=seq(0.1,1,length=50), x2=seq(0.1,1,length=50) )
x.var    <- c("x1", "x2")
x.mean   <-  sapply(x[,x.var],mean,na.rm=T)
x.ub     <-  sapply(x[,x.var],max,na.rm=T)
x.lb     <-  sapply(x[,x.var],min,na.rm=T)
x$y1 <- 10  -3*x$x1 + 2*x$x2   + rnorm(nrow(x),0,1)
x$y2 <- 20 +  50*x$x1 - 20*x$x2 + rnorm(nrow(x),0,1)
x$z  <- 100 - (x$y1/x$y2)
sapply(x,range)
```

```
##       x1  x2        y1        y2        z
## [1,] 0.1 0.1  5.290225  3.466876 96.14665
## [2,] 1.0 1.0 13.798293 68.373127 99.91605
```

[7] the variance of a weighted sum of random variables $\mathrm{y_j}$ is given by $Var(\sum_j a_j y_j) = \sum_j a_j^2 \cdot Var(y_j) + 2\sum_{j>i} a_i a_j \cdot Cov(y_i, y_j)$ with $Var(y_j) = Var(\epsilon_j) = \sigma_j^2$ and $Cov(y_i, y_j) = Cov(\epsilon_i, \epsilon_j) = 0$.

[8] An example might be helpful here: If we take a N-sample from a normal distribution with variance σ^2 than each observations has variance σ^2. The mean of this sample can be written as the least squares estimate a_0 of the linear regression model, $\hat{y} = \bar{y} = a_0$, which has variance $\frac{\sigma^2}{N}$. More generally, for a linear parametric model $\hat{y} = f(x_0|a_i)$ at x-location x_0 the variance is given by $Var(\hat{y}) = [x_0^T \cdot (X^T \cdot X)^{-1} \cdot x_0] \cdot \sigma^2$. $Var(\hat{y}(x_0))$ gives a number in the order of magnitude $\approx \frac{\sigma^2}{N}$, and that explains why the model error is so much smaller than the sampling error. Note again, how the design X affects the model error by the term $(X^T \cdot X)^{-1}$.

```
x$yy <- 100 - (10  -3*x$x1 + 2*x$x2  ) / (20 +  50*x$x1 - 20*x$x2)
lattice::wireframe(yy ~ x1*x2, data=x , scales = list(arrows = F),drape=F,zlab="Z",
                   screen = list(z = 30, x = -60),zlim=c(97.5,100))
```

As expected Z is non-linear hyperbolic, and so the realization z=f(x_1,x_2) can be expected non-linear, too. This is in fact the case as the empirical model building step shows, and z needs to be modelled with a quadratic regression model, z=rsm(x_i). However, residual diagnostics shows (not included in the code and left as an excercise) that the model is still biased and does not appropriately describe the data. Figure 6.13 compares the response surface of the unbiased model $\hat{z}.derived = 100 - \frac{\hat{y}_1}{\hat{y}_2}$ with the biased RSM model $\hat{z}.rsm \sim x_1 + x_2 + x_1 \cdot x_2 + x_1^2 + x_2^2$. The relationship $100 - \frac{\hat{y}_1}{\hat{y}_2}$ depicted on the left of figure 6.13 cannot be described unbiasedly by the RSM model, and the quadratic model creates an artificial ridge in an attempt to fit the hyperbolic relationship. As a consequence the "true" maximum at $x_1^* = 1; x_2^* = 0.1$ is missed and an artifical maximum is created at $x_1^* = 0.74; x_2^* = 0.34$.

```
        res1  <- lm(y1~(x1+x2) ,data=x)
        res2  <- lm(y2~(x1+x2) ,data=x)
summary(res.z <- lm(z~(x1+x2)^2 + I(x1^2)+ I(x2^2),data=x))
```

```
##
## Call:
## lm(formula = z ~ (x1 + x2)^2 + I(x1^2) + I(x2^2), data = x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.62157 -0.03968 -0.00667  0.04400  0.30168
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 99.46314    0.01556 6392.63   <2e-16 ***
## x1           1.40641    0.04084   34.44   <2e-16 ***
## x2          -0.60462    0.04084  -14.81   <2e-16 ***
## I(x1^2)     -1.24237    0.03326  -37.36   <2e-16 ***
## I(x2^2)     -0.38020    0.03326  -11.43   <2e-16 ***
## x1:x2        1.18976    0.02973   40.02   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1044 on 2494 degrees of freedom
## Multiple R-squared:  0.8398, Adjusted R-squared:  0.8395
## F-statistic:  2615 on 5 and 2494 DF,  p-value: < 2.2e-16
```

```
x$z.derived   <- (100 - predict(res1,x)/predict(res2,x)  )
x$z.rsm       <- predict(res.z,x)
```

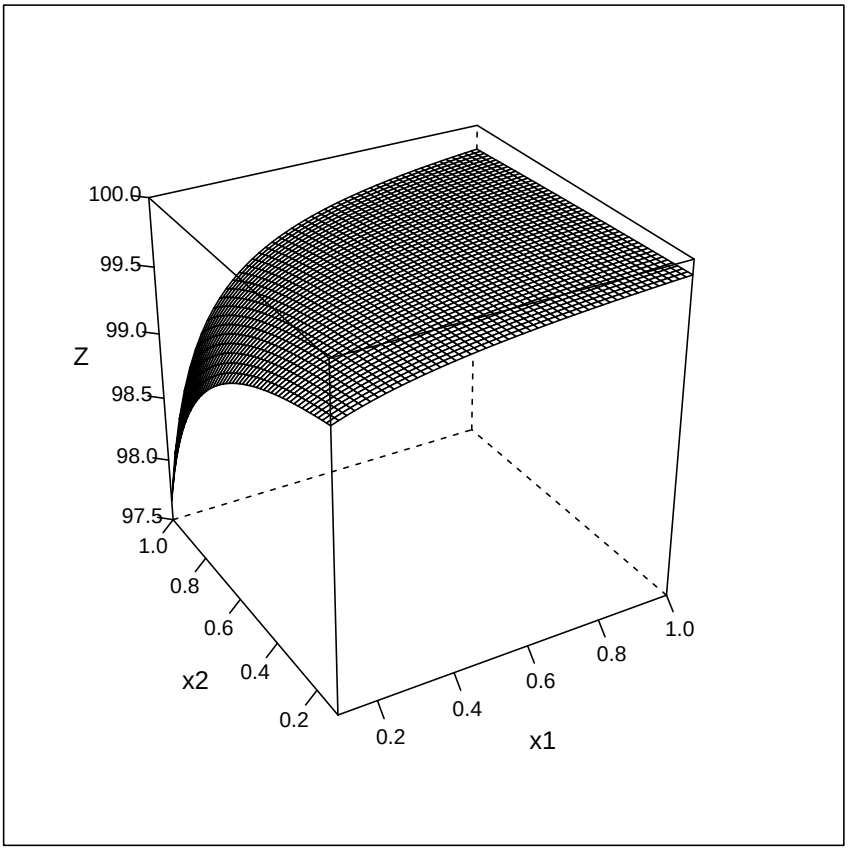

Figure 6.12: Wireframe plot of the "true" derived response $Z = 100 - \frac{f_1}{f_2}$ as a function of x_1, x_2.

```
p1  <- lattice::wireframe(z.derived ~ x1*x2, data=x ,
       scales = list(arrows = FALSE,
                                  x=c(cex=1.1),
                                  y=c(cex=1.1),
                                  z=c(cex=1.1)) ,
       drape=F,zlim=c(97.5,100),
       screen = list(z = 30, x = -60),
       zlab=list(label="z.derived",rot=90),
        xlab=list(label=expression(x [1]),rot=0,cex=1.1),
        ylab=list(label=expression(x [2]), rot=0,cex=1.1) )
p2  <- lattice::wireframe(z.rsm ~ x1*x2, data=x ,
       scales = list(arrows = FALSE,
                                  x=c(cex=1.1),
                                  y=c(cex=1.1),
                                  z=c(cex=1.1)) ,
       drape=F,zlim=c(97.5,100),
       screen = list(z = 30, x = -60),
       zlab=list(label="z.rsm",rot=90),
       xlab=list(label=expression(x [1]),rot=0,cex=1.1),
        ylab=list(label=expression(x [2]), rot=0,cex=1.1) )
plot(p1,split=c(1,1,2,1),more=T)  # split = c(col,row,ncol,nrow)
plot(p2,split=c(2,1,2,1),more=T)
```

```
##definition objective functions to be maximized

# the function 100 - (y1.hat/y2.hat) (taken negative for max)
obj.y1.y2 <- function(x) {
    xx               <- data.frame(rbind(x))
    colnames(xx)     <- x.var
    return(-(100 - predict(res1,xx)/predict(res2,xx)  ) )  # min
  }

# the function z.hat
obj.z <- function(x) {
  xx               <- data.frame(rbind(x))
  colnames(xx)     <- x.var
  return(- predict(res.z,xx)  )
}

# scenario 1: max(100 - (y1.hat/y2.hat)) ;
# correct maximum at x1=1;x2=0.1
system.time(res.nlp1 <- Rsolnp::gosolnp(fun=obj.y1.y2 ,
                        LB=x.lb,UB=x.ub, par=x.mean,
                        n.restarts = 1,
                        n.sim = 10000, rseed=12345,
```

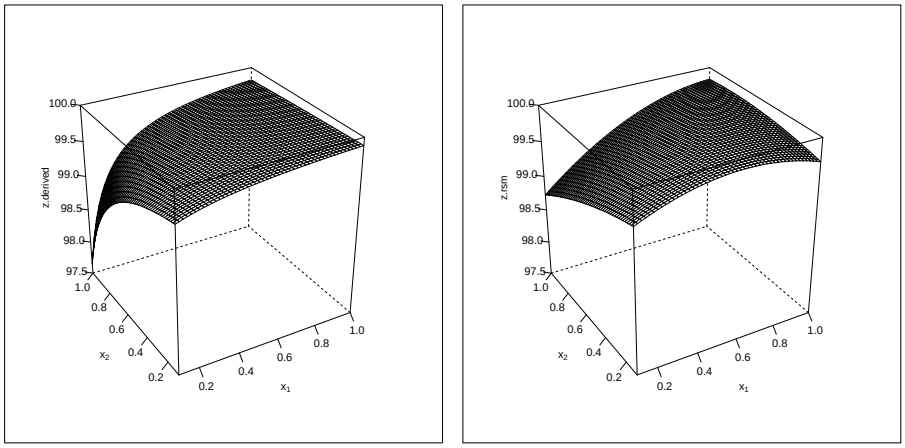

Figure 6.13: Wireframe plot of $\hat{z}.derived = 100 - \frac{\hat{f}_1}{\hat{f}_2}$ (left panel) and estimated $\hat{z}.rsm = rsm(100 - \frac{y_1}{y_2})$ (right panel) as a function of x_1, x_2.

```
                        control=list(trace=0)))
```

```
##    user  system elapsed
##    9.05    0.15    9.25
```

```
round(res.nlp1$pars,3)
```

```
##  x1  x2
## 1.0 0.1
```

```
res.nlp1$convergence # 0
```

```
## [1] 0
```

```
# scenario 2: max(z.hat) ;
# erroneous maximum at x1=0.739,x2=0.361
system.time(res.nlp2 <- Rsolnp::gosolnp(fun=obj.z,
                        LB=x.lb,UB=x.ub, par=x.mean,
                        n.restarts = 1,
                        n.sim = 10000, rseed=12345,
                        control=list(trace=0)))
```

```
##    user  system elapsed
##    7.01    0.11    7.18
```

```
round(res.nlp2$pars,3)
```

```
##    x1    x2
## 0.739 0.361
```

```
res.nlp2$convergence # 0
```

```
## [1] 0
```

The example impressively demonstrated how arithmetics with measured responses can turn a simple model building problem into a hard modeling problem, which cannot be handled anymore with simple polynomial surrogate functions. So, the habit of doing arithmetics with measured responses should better become a habit of bygone times.

Chapter 7

Application: Optimizing catalysis conditions

The following example is a detailed description on how modeling, DoE & optimization has helped to substantially improve the reaction conditions of a catalytic system for converting CO_2 to formaldehyde [for details (Siebert M., Krennrich G., Seibicke M., Siegle A.F., Trapp O. 2019)]. The experimental data for reproducing the modeling results with the R code below can be downloaded here. The R-code combines elements from machine learning (here: the Random Forest), experimental design (here: D-optimal design), optimization (here: augmented Lagrange) and relaxation (here: hypercubical relaxation). A visual overview of the optimization course is given in figure 7.1.

For the optimization project, seven different factors (X-variables) are considered - namely:

1. the temperature (**T** [°C]),
2. the partial pressure of hydrogen gas (**p.h2** [bar]),
3. the partial pressure of carbon dioxide gas (**p.co2** [bar]),
4. the time (**t.h** [h]),
5. the amount of catalyst (**m.kat** [μmol]),
6. the amount of additive $Al(OTf)_3$ (**m.add** [μmol])
7. and the volume of the catalysis solution (**v.ml** [ml]).

There are two responses (Y-variables) to be maximized over the seven process factors above - namely:

1. the turnover number of the main product dimethoxymethane (DMM), abbreviated **ton.dmm**
2. the turnover number of the by-product methyl formate (MF), abbreviated **ton.mf**

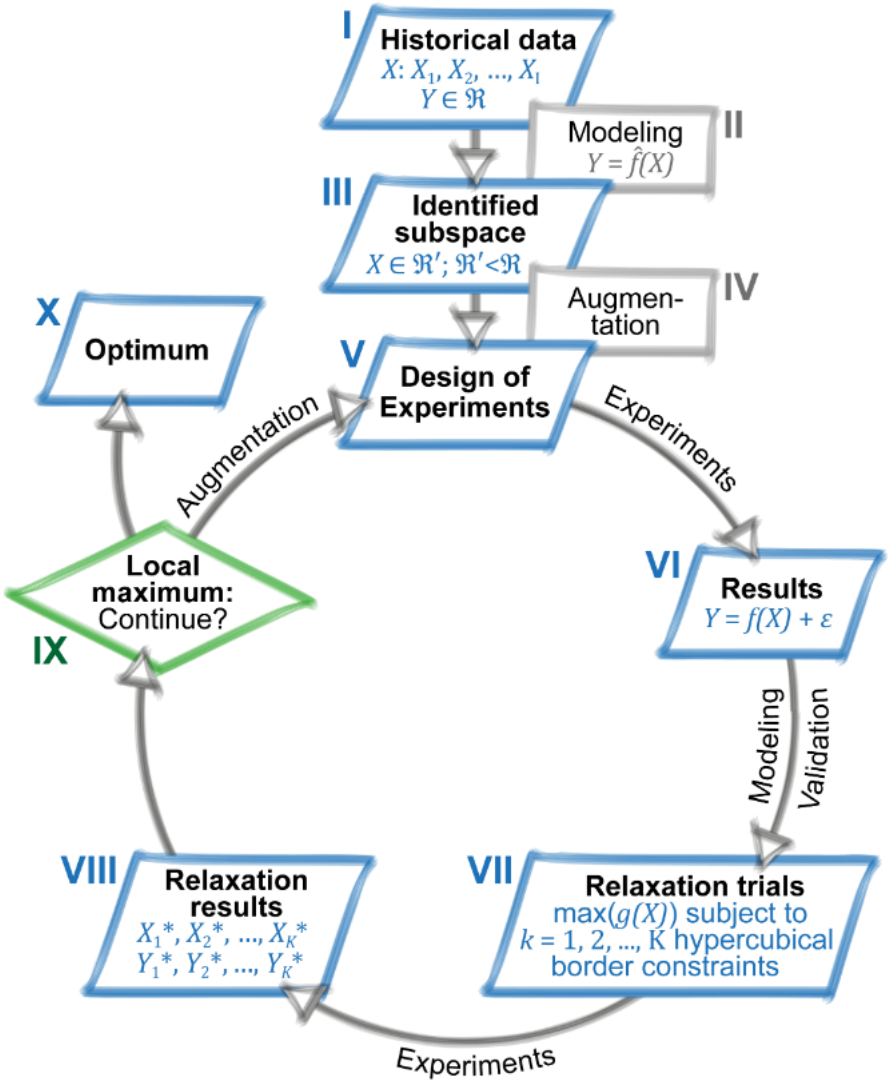

Figure 7.1: Outline of the optimization project

In the following, only the abbreviations mentioned in bold in the brackets are used to refer to the process parameters and responses. The use of units is omitted for the sake of clarity.

The R-code can be run by pasting code chunks (in the grey boxes) to and excecuting them in R-Studio.
Sequential run order is important as subsequent code might depend on objects created by previous code.

7.1 Modelling historical data with the Random Forest (I - III in figure 7.1)

File location: Must be adjusted by the user

```
input         <- file.path("C:","users","myname","data","r.example","c9sc04591k1.xlsx")
```

Next load the required libraries and identify the names of the X- and Y-variables, respectively

```
library(lattice)
library(randomForest)
library(pals)
library(readxl)
set.seed(12345)
##Read historical data 144 x 10 into data frame x
x.historical <- as.data.frame(read_excel(input,
              sheet="historical.data"))
dim(x.historical) # 144 x 10
```

```
## [1] 144  10
```

```
x.var <- colnames(x.historical)[2:8]
y.var <- colnames(x.historical)[9:10]
xx.historical <- round(x.historical[,c("obsnr",x.var,y.var)],3)
```

X-variables are located at position 2-8, Y-variables at position 9-10 (step I in figure 7.1).
Do Random-Forest modeling (see (T. Hastie, R. Tibshirani, J. Friedman 2009, 587–604)) and store models as list **rf.models()**.
Report Goodness-of-fit parameter R^2

```
rf.models <- list()
R2        <- NULL
for (i in (1:length(y.var)) )   {
# rf modeling; add models to list rf.models
  rf.models[[i]] <- randomForest(
    formula(paste(y.var[i],"~",paste(x.var,collapse="+"),
            sep="")),data=x.historical)
```

Table 7.1: First 10 observations of the historical record comprising 49 unique trials in factor space X=m.kat, v.ml, m.add, T, p.h2, p.co2, t.h as triple replicate (N=144) and responses Y=ton.dmm, ton.mf.

	obsnr	m.kat	v.ml	m.add	T	p.h2	p.co2	t.h	ton.dmm	ton.mf
1	1	1.5	0.5	6.25	20	60	20	18	3.856	48.735
2	2	1.5	0.5	6.25	20	60	20	18	7.377	45.773
3	3	1.5	0.5	6.25	20	60	20	18	12.826	58.683
4	4	1.5	0.5	6.25	70	60	20	18	154.421	164.816
5	5	1.5	0.5	6.25	70	60	20	18	155.679	162.916
6	6	1.5	0.5	6.25	70	60	20	18	181.080	155.651
7	7	1.5	0.5	6.25	80	60	20	18	268.602	108.201
8	8	1.5	0.5	6.25	80	60	20	18	289.141	114.908
9	9	1.5	0.5	6.25	80	60	20	18	318.818	113.902
10	10	1.5	0.5	6.25	85	60	20	18	314.123	96.297

```
  R2     <- c(R2, round(cor(predict(rf.models[[i]]),
                    x.historical[,y.var[i]])^2,2))
}
names(R2) <- y.var
R2 # show fit in terms of R2
```

```
## ton.dmm  ton.mf
##    0.83    0.74
```

The Random Forest fits the data well with $R^2_{ton.dmm}$=0.83 and $R^2_{ton.mf}$=0.74. Next report some graphical diagnostics by plotting observed versus predicted values of **ton.dmm** with the observation# used as plot label, see figure 7.2.

```
plot(predict(rf.models[[1]]), x.historical$ton.dmm,type="n",
     xlab="predicted ton.dmm", ylab="observed ton.dmm")
text(predict(rf.models[[1]]),
     x.historical$ton.dmm,labels=x.historical$obsnr)
abline(0,1,col="red")       # diagonal in red
abline(lm(x.historical$ton.dmm~predict(rf.models[[1]]))$coef)
legend("topleft",legend=substitute(R^2 == a, list(a=R2[1])   )   )
# linear fit in black
grid()
```

The Random Forest describes the data well within replication error except for a few outlying no-responders (note: the hyperparameter **mtry** has been validated by another script to be 2 based on 10-fold crossvalidation with consequtive blocks).

The next program chunk creates a 4-dimensional grid (**m.kat** x **m.add** x **p.co2**

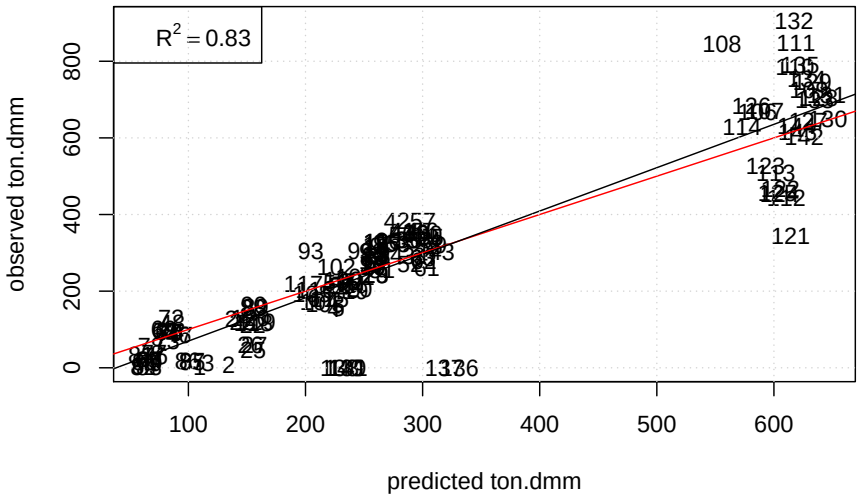

Figure 7.2: Plot observed ton.dmm versus Random Forest predictions with obs# as plot label. The red and black line denote the diagonal and regression line, respectively.

x **T**) given **t.h**, **p.h2** and **v.ml** at median values and plots it as 5-D Trellis plot, figure 7.2.

```
x.pred <- expand.grid(
    m.kat = seq(min(x.historical$m.kat,na.rm=T),
                max(x.historical$m.kat,na.rm=T), length=15),
    m.add = seq(min(x.historical$m.add),
                max(x.historical$m.add), length=15    ),
    p.co2 = seq(min(x.historical$p.co2),
                max(x.historical$p.co2), length=3    ),
    T     = seq(min(x.historical$T),
                max(x.historical$T), length=3  ),
    t.h   =median(x.historical$t.h),
    p.h2 =median(x.historical$p.h2),
    v.ml =median(x.historical$v.ml))
x.pred$y1 <- predict(rf.models[[1]],x.pred)
wireframe(y1 ~ m.kat*m.add|p.co2*T, data=x.pred,
                zlab=list(label="dmm",cex=0.7,rot=90),
                xlab=list(label="m.kat",cex=0.7,rot=0),
                ylab=list(label="m.add",cex=0.7,rot=0),
                drape=TRUE,
                at=do.breaks(c(50,600),100),
                col.regions = parula(100),
                strip=TRUE, pretty=TRUE,
                scales = list(arrows = FALSE,
                              x=c(cex=0.7),
                              y=c(cex=0.7),
                              z=c(cex=0.7)) ,
                screen = list(z = 300, x = -55, y=0)   ,
                par.strip.text=list(cex=1.2),
                layout=c(3,3),
                cuts=1000,
                par.settings = list(superpose.line = list(lwd=3))
  )
```

The trellis-plot, figure 7.3, reveals four different domains of activity in the **m.kat** x **m.add** subspace. This suggests to select all trials with condition ton.dmm>400 as potential candidates for further optimization (step III in figure 7.1).

7.2 First augmentation (DoE1; step IV & V in figure 7.1)

The following code chunk selects nine high-performing candiate points from the historical record with the condition ton.dmm>400. In this subset **T**, **p.h2**, **p.co2** and **t.h** were found constant at T=90, p.h2=90, p.co2=20 and t.h=18

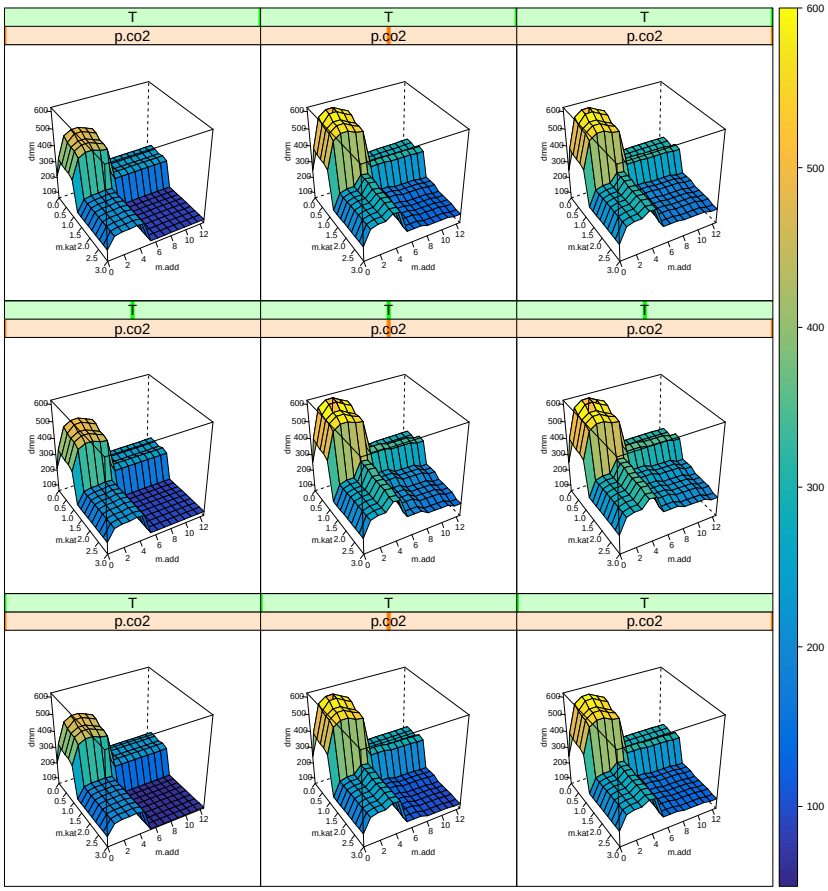

Figure 7.3: 5-D trellis plot ton.dmm=f(m.kat, m.add, p.co2, T) given t.h, p.h2, v.ml at median values. The outer x-axis is p.co2 and the outer y-axis refers to T.

thereby suggesting to conditionally optimize **m.kat**, **v.ml** and **m.add** first while keeping the remaining 4 variables constant at T=90, p.h2=90, p.co2=20 and t.h=18 for the time being.
A full factorial 3^3 grid design is created as a set of potential candidate points and the nine runs from above were augmented by another set of six runs optimally selected from the candidate set so as to render a full second order design of the factors **m.kat**, **v.ml**, **m.add** estimable (step III in figure 7.1).

```
library(AlgDesign)
# select promising subspace (9 unique runs)
# based on RF analysis and augment D-optimally
x.promising<- unique(x.historical[x.historical$ton.dmm>400,x.var])
dim(x.promising) # 9 x 7
```

```
## [1] 9 7
```

```
sapply(x.promising,range)
```

```
##       m.kat v.ml   m.add  T p.h2 p.co2 t.h
## [1,] 0.1875 0.25 0.78125 90   90    20  18
## [2,] 0.7500 0.50 3.12500 90   90    20  18
```

```
# show ranges; note T, p.h2, p.co2, t.h being constant
# create full factorial candidate set 3 x 3 x 3
candidate <- expand.grid(
  m.kat=seq(min(x.promising$m.kat),
            max(x.promising$m.kat)     ,length=3),
  v.ml=seq(min(x.promising$v.ml),
           max(x.promising$v.ml)     ,length=3),
  m.add=seq(min(x.promising$m.add),
            max(x.promising$m.add)     ,length=3) )
candidate  <- rbind(x.promising[,1:3],candidate)
# rowbind x.promising and candidate set
set.seed(12345)
doe.cat <- optFederov(~ (m.kat + v.ml + m.add)^2+
                        I(m.kat^2) + I(v.ml^2) + I(m.add^2),
          data=candidate,center=T,augment=T,rows=(1:9),
                     criterion="D", nTrials=15)$design
# d-optimal cat design
# comprising 9 historical
# and 6 additional candidates from candidate set
rownames(doe.cat )   <- NULL
## check condition number of augmeneted design,
#  1 indicates a perfectly hypersherical design,
#  large positive number >>1 indicate poor designs
kappa(model.matrix(~ (m.kat + v.ml + m.add)^2 +
                  I(m.kat^2) + I(v.ml^2) + I(m.add^2) ,
```

Table 7.2: D-optimal RSM design in (m.kat, v.ml, m.add) comprising historical high responders 1-9 and D-optimal augmentation trials 10-15

obsnr	m.kat	v.ml	m.add
1	0.18750	0.500	0.781250
2	0.37500	0.500	1.562500
3	0.75000	0.500	3.125000
4	0.37500	0.500	0.937500
5	0.37500	0.500	1.250000
6	0.37500	0.500	1.875000
7	0.37500	0.500	2.187500
8	0.37500	0.500	2.500000
9	0.37500	0.250	1.562500
10	0.75000	0.250	0.781250
11	0.75000	0.500	0.781250
12	0.75000	0.375	1.953125
13	0.18750	0.250	3.125000
14	0.75000	0.250	3.125000
15	0.46875	0.375	3.125000

```
                    data=data.frame(scale(doe.cat)    )    ) )
```

```
## [1] 8.519979
```

```
doe.cat <- data.frame(obsnr=1:15,doe.cat)
knitr::kable(
  doe.cat, booktabs = TRUE,row.names=F,align="c",
  caption = 'D-optimal RSM design in (m.kat, v.ml, m.add)
  comprising historical high responders
  1-9 and D-optimal augmentation trials 10-15')
```

The augmentation trials #10-#15 were run in the laboratory (results for #1-#9 are already available) and DoE1 was analyzed with stepwise OLS.

7.3 Modeling DoE1 (step VI & VII in figure 7.1)

The analysis of well-designed experiments is relatively straightforward. Because the data has been designed with an underlying parametric model assumption (here: a full second order RSM design), all model terms are unconfounded (orthogonal) and can be easily estimated with stepwise OLS.
DoE1 is available in the Excel sheet "DoE1" and is of dimension 44x10. X-variables are at position# 2-4 Y-variables are at position 9-10

7.4 Analysis of DoE1

The code reads the data from DoE1, assigns X- and Y-variables, does the OLS modeling and reports R^2 values of the fit.

```
library(MASS)
doe1 <- as.data.frame(read_excel(input,
                sheet="DoE1"))
dim(doe1)  # 44 x 10
```

```
## [1] 44 10
```

```
x.var <- colnames(doe1)[2:4]
y.var <- colnames(doe1)[9:10]
ols.models <- list()
R2         <- NULL
form <- "~ (m.kat + v.ml + m.add)^2 + I(m.kat^2) +
           I(v.ml^2) + I(m.add^2) "# parametric formula for
                                    # linear model
for (i in (1:length(y.var)) )   {# step-OLS modeling;
                                  # add models to list ols.models
ols.models[[i]] <- stepAIC(lm(formula(paste(y.var[i],form, sep="")),
                        data=doe1,x=T,y=T),trace=0,
                        k =   log(nrow(doe1))) # BIC criterion
  R2   <- c(R2, round(cor(predict(ols.models[[i]]),
                          doe1[,y.var[i]])^2,2))

}
names(R2) <- y.var
R2 # show fit in terms of R2
```

```
## ton.dmm  ton.mf
##    0.91    0.97
```

Simple graphical model diagnostics, figure 7.4, here scatter plot observed values versus OLS-predictions, reveal an unbiased fit, with the model depicted as trellis plot in figure 7.5

```
plot(predict(ols.models[[1]]), doe1$ton.dmm,type="n",
     xlab="predicted ton.dmm", ylab="observed ton.dmm")
text(predict(ols.models[[1]]), doe1$ton.dmm,labels=doe1$obsnr)
abline(0,1,col="red")     # diagonal in red
abline(lm(doe1$ton.dmm~predict(ols.models[[1]]))$coef,lty=2)
legend("topleft",legend=substitute(R^2 == a, list(a=R2[1])   )   )
# linear fit in black
grid()
```

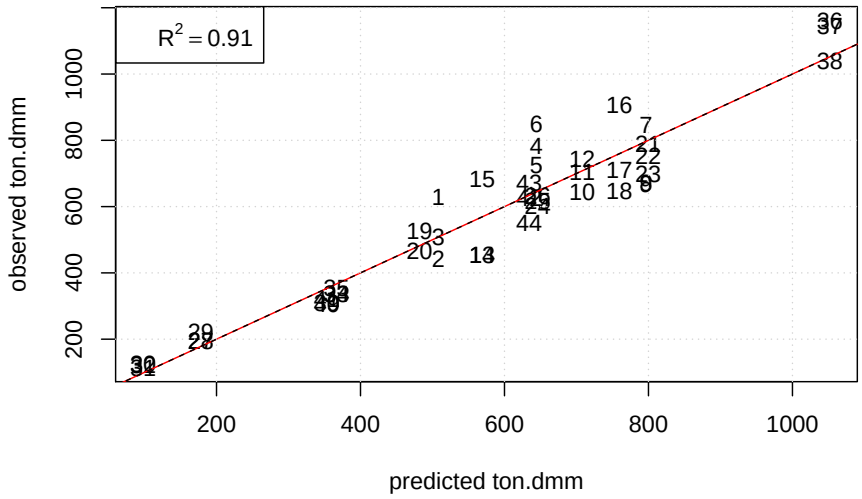

Figure 7.4: Observed ton.dmm versus OLS predictions of the RSM design with obs# being used as plot label. The red and black line denote the diagonal and regression line, respectively.

```
## show ton.dmm = g3(m.kat,m.add,v.ml) as trellis plot
x.pred <- expand.grid(m.kat = seq(min(doe1$m.kat),
                                  max(doe1$m.kat), length=15    ),
                      m.add = seq(min(doe1$m.add),
                                  max(doe1$m.add), length=15    ),
                      v.ml  = seq(min(doe1$v.ml),
                                  max(doe1$v.ml), length=3    )  )
x.pred$y1 <- predict(ols.models[[1]],x.pred)
wireframe(y1 ~ m.kat*m.add|v.ml, data=x.pred,
                zlab=list(label="dmm",cex=1.2,rot=90),
                xlab=list(label="m.kat",cex=1.2,rot=0),
                ylab=list(label="m.add",cex=1.2,rot=0),
                drape=TRUE,
                at=do.breaks(c(100,1200),100),
                col.regions = parula(100),
                strip=T,
                scales = list(arrows = FALSE,
                              x=c(cex=1.2),
                              y=c(cex=1.2),
                              z=list(cex=1.2) ),
                screen = list(z = 300, x = -60, y=0),
                par.strip.text=list(cex=1.2),
                layout=c(3,1)
)
```

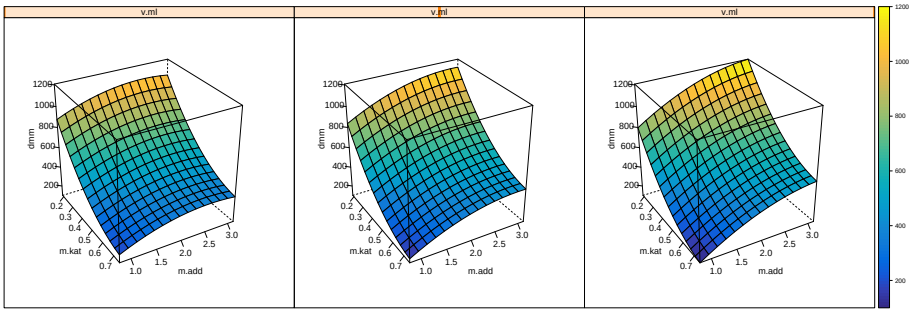

Figure 7.5: 4-D Trellisplot ton.dmm=f(m.kat,m.add,v.ml) given T=90, p.h2=90, p.co2=20 and t.h=18

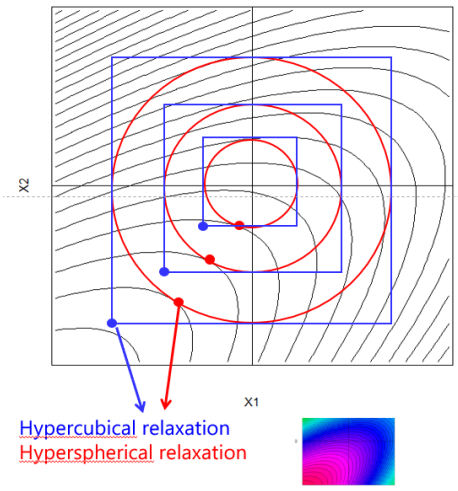

Figure 7.6: Two-dimensional example of hypercubical and hyperspherical relaxation

7.5 Optimization and relaxation of DoE1 (step VIII in figure 7.1)

In the optimization step the above OLS model is maximized subject to box-constraints and the box-constraints are subsequently relaxed in discrete steps with stepsize 10, 20 and 30% of the initial space thereby creating a sequence of three relaxation trials to be realized in the laboratory.

The rationale of relaxation as an exploratory tool for making rapid progress was discussed in detail in chapter 6 and is again graphically sketched in figure 7.6.

With ΔX being the stepsize of the relaxation, in the code chunk taken to be 10% of the initial design space, that is $\Delta X = 0.1 \cdot (UB - LB)$, and LB and UB denoting the lower and upper bounds of the process factors, respectively, the optimization and relaxation problem can be stated as follows

$$\max_X f(X)$$

subject to **hypercubical constraints:**

$$(LB - k \cdot \Delta X) \leq X \leq (UB + k \cdot \Delta X)$$

or alternatively subject to **hyperspherical constraints:**

$$\sum_{i=1}^{I} x_i^2 \leq (R + k \cdot \Delta r)^2$$

with R denoting the radius to be relaxed with stepsize Δr.

By varying k=1,2,3... and running the optimization, a sequence of relaxation trials is generated in the direction of steepest ascend of **ton.dmm** to be realized in laboratory.
Note that in the present code hypercubical rather than hypersperical constraints are being applied.

```
library(Rsolnp)
x.var          <- colnames(doe1)[2:4]
x.mean         <- sapply(doe1[,x.var],mean,na.rm=T)
names(x.mean)  <- x.var
x.lb           <-  sapply(doe1[,x.var],min,na.rm=T)
x.ub           <-  sapply(doe1[,x.var],max,na.rm=T)
delta          <- (x.ub-x.lb)/10 # stepsize 10% of initial design

df.collect  <- NULL

##########################
# given predictors x the function returns
# OLS predictions y=g(x) for each
y.val   = function(x) {
  xx               <- data.frame(rbind(x))
  colnames(xx)     <- x.var
  collect          <- NULL
  for (iii in (1:length(y.var)) )   {
    collect <- c(collect,predict(ols.models[[iii]],xx))
  }
  return( collect  )
}

objective   =function(x) {
  xx               <- data.frame(rbind(x))
  colnames(xx)     <- x.var
  return( -predict(ols.models[[1]],xx)  )#note: -obj <=> max(obj)
}
###
for (i in (1:3))   {

  xx.lb <- x.lb - i*delta
  xx.ub <- x.ub + i*delta

  tt <- system.time(res  <- solnp(fun=objective,
                            pars=x.mean,
                            LB=xx.lb,UB=xx.ub,
                            control=list(trace=0)))
```

```
  x.solution          <- res$pars
  y.opt               <- y.val(x.solution)
  names(y.opt)        <- y.var


  df.collect  <- rbind(df.collect,
                       data.frame(relax.step=i*10,
                       target="max(ton.dmm)",
                       rbind(x.solution),rbind(y.opt),
                       stringsAsFactors =FALSE,
                       t.min=round(tt[3]/60,2) )  )

}
rownames(df.collect)  <- NULL

df.collect
```

```
##   relax.step       target   m.kat      v.ml    m.add  ton.dmm   ton.mf t.min
## 1         10 max(ton.dmm) 0.13125 0.5249999 3.359373 1363.439 1080.125     0
## 2         20 max(ton.dmm) 0.07500 0.5499999 3.575959 1529.786 1303.163     0
## 3         30 max(ton.dmm) 0.01875 0.5749999 3.659436 1707.569 1544.425     0
```

Table 7.3: Relaxation of DoE1 model in 10,20,30% steps

relax.step	target	m.kat	v.ml	m.add	ton.dmm	ton.mf	t.min
10	max(ton.dmm)	0.13	0.52	3.36	1363.44	1080.13	0
20	max(ton.dmm)	0.08	0.55	3.58	1529.79	1303.16	0
30	max(ton.dmm)	0.02	0.57	3.66	1707.57	1544.42	0

The reported response values for **ton.dmm** and **ton.mf** in the result table 7.3 from relaxation are model predictions and **not** measured values.
In the laboratory the following results, table 7.4, were experimentally realized

Table 7.4: Experimentally obtained results from relaxation trials in table 7.3

Relaxation%	m.kat	v.ml	m.add	ton.dmm	ton.mf
10	0.13	0.52	3.36	1373.56	1559.56
20	0.08	0.55	3.58	1374.59	2761.87
30	0.02	0.57	3.66	179.56	1039.71

Based on these results, table 7.4, the second trial was selected as a candidate for further optimization. Note the sharp drop of the third relaxation trial (30%)

compared to the model predictions. This likely indicates that a local maximum was exceeded in the course of the relaxation.
Given this finding **m.kat**, **v.ml** and **m.add** are considered locally optimal (hereafter referred to as **m.kat**$^{\#}$, **v.ml**$^{\#}$, **m.add**$^{\#}$) and further optimization will focus on the factors **T**, **p.h2**, **p.co2**, **t.h** so far kept constant.

7.6 Linear DoE2 (step IX & V in figure 7.1)

A small linear saturated design was created comprising five unique runs in the parameters **T**, **p.h2**, **p.co2**, **t.h** given **m.kat**$^{\#}$, **v.ml**$^{\#}$ and **m.add**$^{\#}$.

```
library(AlgDesign)
## full factorial candidate set
candidate <- expand.grid(T=seq(80,  100 ,length=2),
                         p.H2=seq(80,  100 ,length=2),
                         p.CO2=seq(15,  25 ,length=2),
                         t.H=seq(16,  20 ,length=2) )
set.seed(12345)
doe2 <- optFederov(~ T + p.H2 + p.CO2 + t.H ,
                     data=candidate,center=T,
                     criterion="D", nTrials=5)$design
# d-optimal cat design
(kappa <- kappa(model.matrix(~ T + p.H2 + p.CO2 + t.H ,
                   data=data.frame(scale(doe2)   )   ) ))
```

```
## [1] 1.499816
```

```
# 1.499816
rownames(doe2)  <- NULL
doe2 <- data.frame(obsnr=1:5,doe2)
knitr::kable(
  doe2, booktabs = TRUE,row.names=F,digits=2,format="pandoc",
  caption = 'Linear D-optimal design with
  20% relaxation trial from table 7.4 as centerpoint')
```

Table 7.5: Linear D-optimal design with 20% relaxation trial from table 7.4 as centerpoint

obsnr	T	p.H2	p.CO2	t.H
1	80	80	15	16
2	100	100	25	16
3	100	100	15	20
4	100	80	25	20
5	80	100	25	20

Note that the design is next to orthogonal with κ=1.5!
Analysis and relaxation of DoE2 is similar to the steps already described.

7.7 Analysis of DoE2 (step VI & VII in figure 7.1)

```
doe2 <- as.data.frame(read_excel(input,
                  sheet="DoE2"))
dim(doe2)  # 21 x 10
```

```
## [1] 21 10
```

```
x.var <- colnames(doe2)[5:8]
y.var <- colnames(doe2)[9:10]
ols2.models <- list()
R2          <- NULL
form <- "~ T +  p.h2 +  p.co2 + t.h "
# parametric formula for linear
# models in R notation

for (i in (1:length(y.var)) )   {    # add OLS models to list ols2.models
ols2.models[[i]] <- stepAIC(lm(formula(paste(y.var[i],form, sep="")),
                    data=doe2,x=T,y=T),trace=0,
                    k = log(nrow(doe2)))
  R2   <- c(R2, round(cor(predict(ols2.models[[i]]),
                          doe2[,y.var[i]])^2,2))
}
names(R2) <- y.var
R2 # show fit in terms of R2
```

```
## ton.dmm  ton.mf
##    0.92    0.78
```

```
# show fit as plot observed versus predicted (only ton.dmm shouns)
plot(predict(ols2.models[[1]]), doe2$ton.dmm,
     type="n",xlab="predicted ton.dmm", ylab="observed ton.dmm")
text(predict(ols2.models[[1]]), doe2$ton.dmm,labels=doe2$obsnr)
abline(0,1,col="red")       # diagonal in red
abline(lm(doe2$ton.dmm~predict(ols2.models[[1]]))$coef,lty=2)
# linear fit in black
grid()
legend("topleft",legend=substitute(R^2 == a, list(a=R2[1])   )   )
```

```
###### trellis plot of OLS2 effects
```

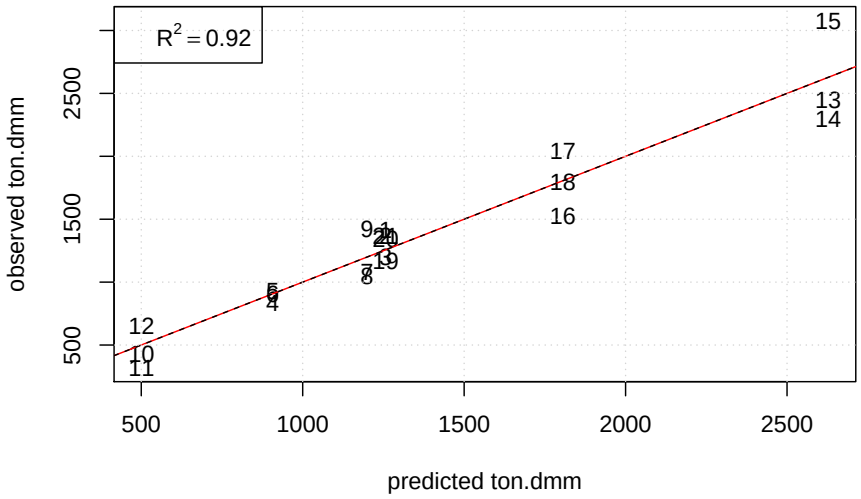

Figure 7.7: Observed ton.dmm versus OLS predictions ton.dmm=f(T, p.h2, p.co2,t.h) given m.kat=0.075, v.ml=0.55, m.add=3.576. The red and black line denote the diagonal and regression line, respectively.

```
x.pred <- expand.grid(     T = seq(min(doe2$T),
                                   max(doe2$T), length=15),
                           p.h2 = seq(min(doe2$p.h2),
                                      max(doe2$p.h2), length=15 ),
                           p.co2  = seq(min(doe2$p.co2),
                                        max(doe2$p.co2), length=3),
                           t.h    = seq(min(doe2$t.h),
                                        max(doe2$t.h), length=3))
x.pred$y1 <- predict(ols2.models[[1]],x.pred)
wireframe(y1 ~ T*p.h2|p.co2*t.h, data=x.pred,
                zlab=list(label="dmm",cex=1,rot=90),
                xlab=list(label="T",cex=1,rot=0),
                ylab=list(label="p.h2",cex=1,rot=0),
                drape=TRUE,
                at=do.breaks(c(-50,2600),100),
                col.regions = parula(100),
                strip=T,
                scales = list(arrows = FALSE,
                              x=c(cex=1),
                              y=c(cex=1),
                              z=c(cex=1)) ,
                screen = list(z = 60, x = -55, y=0), layout=c(3,3) ,
                 par.strip.text=list(cex=1.2)  )
```

7.8 Optimization and relaxation of DoE2 (step VIII in figure 7.1)

```
x.var          <- colnames(doe2)[5:8]
x.mean         <- sapply(doe2[,x.var],mean,na.rm=T)
names(x.mean)  <- x.var
x.lb           <-  sapply(doe2[,x.var],min,na.rm=T)
x.ub           <-  sapply(doe2[,x.var],max,na.rm=T)
delta          <- (x.ub-x.lb)/4
# stepsize 25% of initial design

df.collect  <- NULL

##########################
# given predictors x the function returns
# OLS predictions y.j=g.j(x) for each y.j
y.val   = function(x) {
  xx               <- data.frame(rbind(x))
  colnames(xx)     <- x.var
```

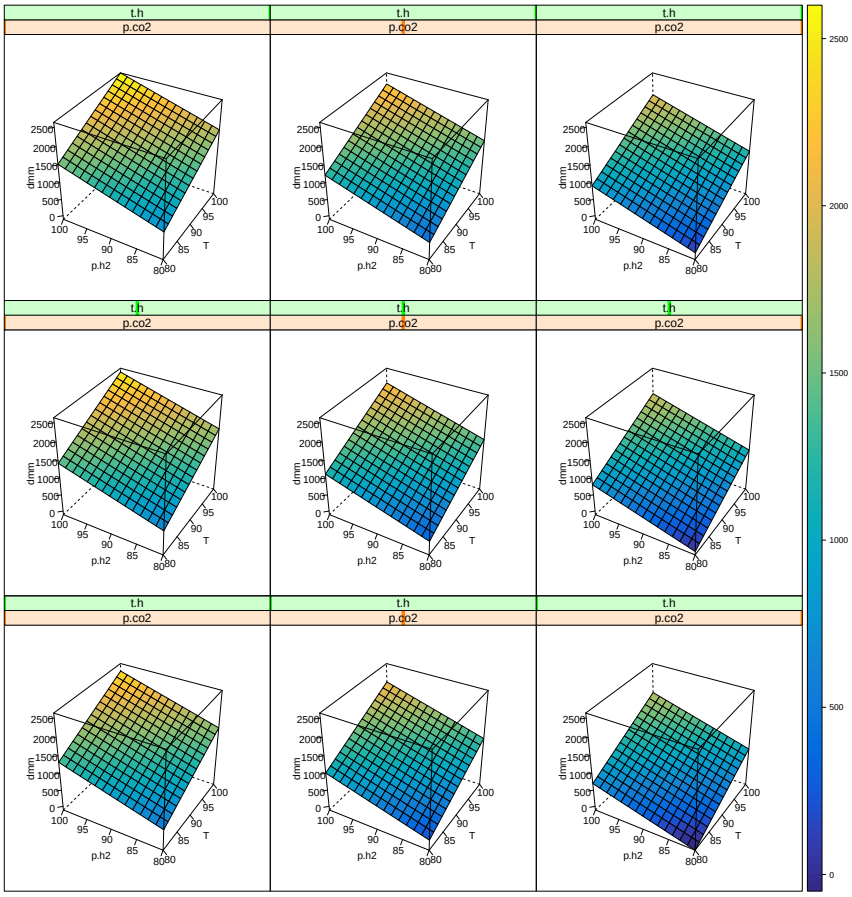

Figure 7.8: 5-D Trellis plot of ton.dmm=f(T, p.h2, p.co2, t.h) given m.kat=0.075, v.ml=0.55, m.add=3.576

```
  collect          <- NULL
  for (iii in (1:length(y.var)) )   {
    collect <- c(collect,predict(ols2.models[[iii]],xx))
  }
  return( collect  )
}
objective   =function(x) {
  xx               <- data.frame(rbind(x))
  colnames(xx)     <- x.var
  return( -predict(ols2.models[[1]],xx)  )
  # note: -obj <=> max(obj)
}
###
for (i in (0:2))   {

  xx.lb <- x.lb - i*delta
  xx.ub <- x.ub + i*delta
  tt <- system.time(res  <- solnp(fun=objective, pars=x.mean,
                            LB=xx.lb,UB=xx.ub,
                            control=list(trace=0)))

  x.solution    <- res$pars
  y.opt         <- y.val(x.solution)
  names(y.opt)  <- y.var
  df.collect    <- rbind(df.collect, data.frame(relax.step=i*25,
                         target="max(ton.dmm)",rbind(x.solution),
                         rbind(y.opt), stringsAsFactors =FALSE,
                         t.min=round(tt[3]/60,2) )  )

}
rownames(df.collect)  <- NULL
df.collect
```

```
##   relax.step       target   T p.h2 p.co2 t.h  ton.dmm   ton.mf t.min
## 1          0 max(ton.dmm) 100  100  15.0  20 2627.845 2407.657     0
## 2         25 max(ton.dmm) 105  105  12.5  21 3313.269 2262.819     0
## 3         50 max(ton.dmm) 110  110  10.0  22 3998.693 2117.981     0
```

Table 7.6: Relaxation trials obtained by hypercubical relaxation of OLS models from DoE2 with 25% step size.

relax.step	target	T	p.h2	p.co2	t.h	ton.dmm	ton.mf	t.min
0	max(ton.dmm)	100	100	15.0	20	2627.85	2407.66	0
25	max(ton.dmm)	105	105	12.5	21	3313.27	2262.82	0
50	max(ton.dmm)	110	110	10.0	22	3998.69	2117.98	0

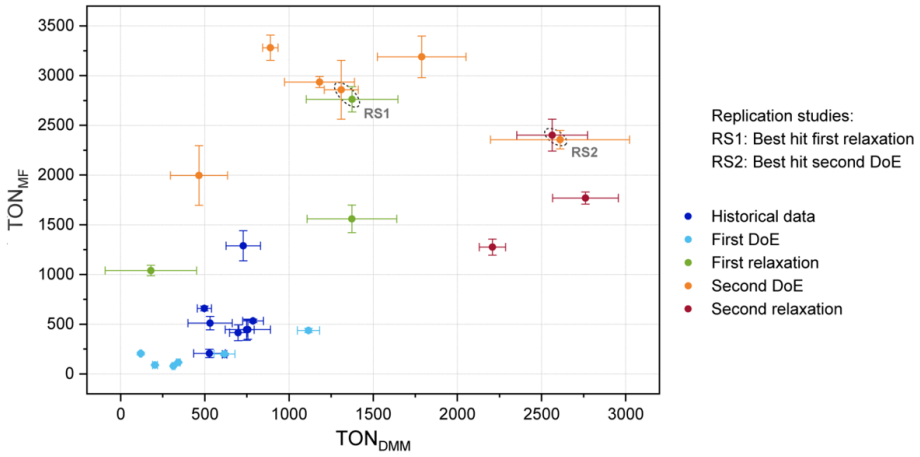

Figure 7.9: Summary of improvements over all optimization steps. The error bars denote one standard deviation based on three replicates

The following table shows the experimental results of the above relaxation trials achieved in the lab, reported as mean values of triple replicates.

Table 7.7: Experimental results achieved by running the relaxation trials from table 7.6 in the laboratory

Relaxation	T	p.h2	p.co2	t.h	ton.dmm	ton.mf
0	100	100	15.0	20	2563.43	2401.47
25	105	105	12.5	21	2760.86	1768.81
50	110	110	10.0	22	2208.56	1275.56

Again, the second relaxation trial (25%) shows a small improvement which turns out reproducible upon replication, while the third relaxation candidate (50%) performs comparatively poor in both responses.

7.9 Conclusion

Figure 7.9 is a graphical summary of the achieved improvement of **ton.dmm** and **ton.mf**, both in terms of location (mean values) and dispersion (standard deviation).

Comparison of the raw figures of the optimization project is impressive: With 46 unique trials in the historical record a **ton.dmm** lift of $3.9 \rightarrow 786$ was achieved which boils down to $lift/trial \approx 17$.

In the present algorithmically driven workflow, a $lift/trial \approx 165$ was achieved in 16 DoE1,2 and relaxation trials that revealed a lift $119 \rightarrow 2761$. Based on these figures multivariate methods are found to be approximately ten-times more effective than one-factor-at-the-time optimization.

Reference

A. Sen, M. Srivastava. 1990. *Regression Analysis, Theory, Methods and Applications*. 1st ed. Springer-Verlag, New York.

D.C. Montgomery. 2013. *Design and Analysis of Experiments*. 8th ed. John Wiley & Sons Inc.

G.E.P. Box, W.G. Hunter, J.S. Hunter. 2005. *Statistics for Experimenters: Design, Innovation, and Discovery*. 2nd ed. John Wiley & Sons, Hoboken.

George E.P. Box, Norman R. Draper. 1987. *Empirical Model-Building and Response Surfaces*. 1st ed. John Wiley & Sons.

J.G. Kalbfleisch. 1985. *Probability and Statistical Inference, Vol 1&2*. 2nd ed. Springer.

John A. Cornell. 1990. *Experiments with Mixtures*. 2nd ed. Wiley, New York.

John Lawson. 2015. *Design and Analysis of Experiments with R*. 1st ed. Chapman & Hall.

John Lawson, Cameron Willden. 2016. "Mixture Experiments in R Using Mixexp." *Journal of Statistical Software* 72. https://www.jstatsoft.org/article/view/v072c02.

K.M. Mullen. 2014. "Continuous Global Optimization in R." *Journal of Statistical Software* 60, 6. https://www.jstatsoft.org/article/view/v060i06.

R.H. Myers, D.C. Montgomery. 2002. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. 2nd ed. John Wiley & Sons.

Siebert M., Krennrich G., Seibicke M., Siegle A.F., Trapp O. 2019. "Identifying High-Performance Catalytic Conditions for Carbon Dioxide Reduction to Dimethoxymethane by Multivariate Modelling." *Chemical Science* 10:45. https://pubs.rsc.org/en/content/articlelanding/2019/sc/c9sc04591k#!divAbstract.

T. Hastie, R. Tibshirani, J. Friedman. 2009. *The Elements of Statistical Learning*. 2nd ed. Springer-Verlag.