Optimization under parameter uncertainties with application to product cost minimization

by

Ann-Sofi Kidwell

Masterarbete i matematik / tillämpad matematik
Master thesis in mathematics / applied mathematics

Date:
2018-01-25

Project name:
Optimization under parameter uncertainties with application to product cost minimization

Author:
Ann-Sofi Kidwell

Supervisors:
Dr. Anders Daneryd, Principal Scientist, ABB Corporate Research, Västerås
Dr. Linus Carlsson, Senior Lecturer, Mälardalens Högskola, Västerås

Reviewer:
Dr. Ying Ni, Senior Lecturer

Examiner:
Professor Sergei Silvestrov

Comprising:
30 ECTS credits
Abstract

This report will look at optimization under parameters of uncertainties. It will describe the subject in its wider form, then two model examples will be studied, followed by an application to an ABB product. The Monte Carlo method will be described and scrutinised, with the quasi-Monte Carlo method being favoured for large problems. An example will illustrate how the choice of Monte Carlo method will affect the efficiency of the simulation when evaluating functions of different dimensions. Then an overview of mathematical optimization is given, from its simplest form to nonlinear, nonconvex optimization problems containing uncertainties. A Monte Carlo simulation is applied to the design process and cost function for a custom made ABB transformer, where the production process is assumed to contain some uncertainties. The result from optimizing an ABB cost formula, where the in-parameters contains some uncertainties, shows how the price can vary and is not fixed as often assumed, and how this could influence an accept/reject decision.
Contents

1 Introduction
   1.1 Literature review .................................................. 2
   1.2 Aim of the thesis and problem formulation .......................... 3
   1.3 Disposition of the thesis ............................................ 3

2 Monte Carlo techniques .................................................... 4
   2.1 Quasi-random sequences ............................................. 6
   2.2 Lattice methods ..................................................... 7
   2.3 Van der Corput sequence ........................................... 8
   2.4 Halton sequences .................................................... 9
   2.5 Sobol' sequences ................................................... 11
   2.6 Generating a Sobol' sequence- a numerical example ................ 13
   2.7 Implementing quasi-random sequences in MATLAB .................... 15
   2.8 Summary ............................................................. 15

3 Numerical experiments on functions containing random variables 16
   3.1 Made up cost function .............................................. 16
   3.2 Evaluating function $g_k$ through Monte Carlo simulations in MATLAB 19
   3.3 Number of iterations needed for Monte Carlo versus quasi Monte Carlo 20
   3.4 Improvement techniques ............................................. 21
   3.5 Summary ............................................................. 23

4 Non-linear optimization problem containing uncertainties 24
   4.1 Nonlinear optimization in MATLAB .................................. 25
      4.1.1 Gradient based methods ...................................... 25
      4.1.2 Algorithms to choose from in fmincon ....................... 27
      4.1.3 GlobalSearch, MultiStart, and patternsearch .................. 28
   4.2 Optimizing problems containing uncertainty ........................ 28
      4.2.1 Stochastic optimization ....................................... 29
      4.2.2 Robust optimization ......................................... 29
      4.2.3 How to deal with equality constraints and uncertainties in optimization 30
   4.3 Penalty substituted for constraints .................................. 30
   4.4 Numerical experiment: Optimizing the design of a Cantilever beam 31
      4.4.1 Optimization using the the expected value .................... 32
List of Figures

2.1 Halton points in different dimensions .......................... 11
2.2 Sobol’ points in different dimensions. .......................... 11

3.1 Sobol’ versus uniform distribution convergence rate .......................... 21
3.2 Mean of functions $g_{90}$ and $\hat{g}_{90}$ .......................... 22

4.1 Active and non-active constraints. .......................... 25
4.2 Cantilever beam, subjected to some forces .......................... 31

5.1 Tailor made HVDC transformer. .......................... 36
5.2 Three-phase transformer, the Trafostar, manufactured by ABB .......................... 37
5.3 Histogram of objective function in Test 1. .......................... 45
5.4 Histogram of objective function in Test 2. .......................... 46

B.1 Probability distribution for $N(120,60)$. .......................... 58
B.2 Truncated normal distribution between -1.5 and 1.5 .......................... 59
B.3 Truncated normal distribution between -0.5 and 1.5 .......................... 59
B.4 Comparing the distribution of four different probability distributions. .......................... 59
B.5 Modified truncated normal distribution. .......................... 60
B.6 Kurtosis and normal distribution .......................... 61
## List of Tables

2.1 Van der Corput sequence for base 2. ............................................. 8
2.2 Halton sequences for bases 2, 3, and 5. ................................. 10
2.3 Halton sequences for bases 163, 167 and 173. ......................... 10

3.1 Expected value, variance, and standard deviation for \( g_k(p) \) ........ 19
3.2 Mean and standard deviation using different sample sets .................. 20
3.3 Sobol’ versus uniform distribution convergence rate for function \( g_{90} \) ... 21

4.1 Running time and average value for the three different algorithms in \textit{fmincon} ... 34

5.1 Uncertainty notation ................................................................. 41
5.2 Expected value and mean from Test 1 ...................................... 44
5.3 Standard deviation, quantile and kurtosis for Test 1 ...................... 45
5.4 Expected value and mean from Test 2 ...................................... 46
5.5 Standard deviation, quantile and kurtosis for Test 2 ...................... 46

A.1 Field axioms ................................................................. 56
A.2 Addition table for \( GF(2^3) \) .................................................. 57
A.3 Multiplication table for \( GF(2^3) \) ............................................. 57

D.1 Accuracy of simulation value vs. number of iterations .................. 64
D.2 Result from optimizing expected value using three different algorithms in \textit{fmincon} 65
To Arabella and Charles, to whom I would like to inspire, anything is possible

Firstly, I would like to express my gratitude to ABB Corporate Research, Västerås, for the opportunity to embark on this challenging and rewarding experience, and especially to my supervisor Dr. Anders Daneryd, for all the help and guidance. And I would like to express my gratitude to Jan Johansson at ABB Power Transformers, Ludvika, for his help and for sharing his expertise in transformers.

I also would like to thank my supervisor at Mälardalens Högskola, Dr. Linus Carlsson for providing me with valuable support and sharing his mathematical expertise, not only during the process of this thesis, but during my years at MDH. Also I would like to thank Dr. Ying Ni for reviewing my work and giving me valuable feedbacks.

Lastly, I would like to thank my family for all support and encouragement throughout this journey.
Chapter 1

Introduction

In engineering, designing complex systems designs, can be accomplished by using experience and good intuition of a design engineer, however, through optimization it is possible to improve an adequate good design into the best design out of all possible good designs. It could be that the process of the designing a system has been developed over a long time, and been both time consuming and costly, requiring large human and material resources, then it might seem easier to procedure a good design and use a system regardless if it is the best one.

With optimization the problem is solved in a systematic and efficient way, with the use of computers, to examine more possibilities than any human could even attempt. Any problem in which an input or design variable need to be determined while satisfying any number of constraints can be formulated as an optimization problem. The design system can be formulated as a problem of optimization and the performance is measured in a system which is optimal where all requirements are met.

In an industrial optimization setting, one can argue against treating parameters as certain, as they are most likely subjected to some uncertainties. The exact bounds of variables might not be known and functions might just be a rough approximation of a real working problem. As often in optimization, these uncertainties are left out and the mean of the variables are used instead. By doing so important information is being left out, crucial when making a decision based on an optimization result. For example, the decision of accepting or rejecting an order, which has a assumed margin of 10%, might not be the same, if it was known to have a standard deviation of 4%. This would make the accept/reject decision challenging. Counting on uncertainties, using sample based methods, when performing a design optimization, might give a business the competitive edge in terms of cost estimations. There are many ways to include uncertainties, one of the most intuitive way is the sampling based Monte Carlo method. Where a simulation is carried out by performing a number of experiments using samples from a known distribution, with the same settings, and by the law of large numbers the results will, with the probability of 1, converge, to the expected value, as the number of experiments increase.

The contribution of the present thesis is to present a framework to include uncertainties in a design process, involving optimization, for High Voltage Transformers manufactured at ABB. A model of the optimization stage of the transformer design is suggested, mimicking the the existing procedures, allowing desired levels of uncertainties to be added. The impact of the uncertainty has
on the cost will be evaluated. This has to the author’s knowledge not been attempted before on this scale of an ABB cost model.

1.1 Literature review

A large number of optimization problem related to production and engineering designs, require that decisions need to be made in the presence of uncertainty, and there are many different ways to deal with the complexity of optimizing under uncertainty.

Stochastic programming and Robust optimization are the most popular frameworks for incorporating uncertainty in optimization. Stochastic programming uses random variables with a known or estimated probability distribution to characterise the uncertainty, there are a number of different approaches in stochastic optimization, such as two-stage and multistage programming described in [8].

Robust optimization, also called worst-case scenario optimization, uses a set of outcomes to characterise the uncertainty and optimizes a worst possible case of the problem. The goal is to find a solution that is guaranteed to be feasible for all realizations of data in a given uncertainty set [3]. In [29] the authors discusses different ways to deal with equality constraints in robust optimization.

Apart from the classical recourse-based stochastic programming, and robust programming, there are also methods such as fuzzy programming, where random parameters are considered as fuzzy, and stochastic dynamic programming, where the system is allowed to evolve over time, see [30].

In this thesis we consider design optimization under parameter uncertainties for manufacturing of bespoke transformer at ABB, using Monte Carlo methods to incorporate the uncertainty into the optimization.

The importance of using Monte Carlo simulations in the field of applied mathematics cannot be underestimated, due to its ability to solve complex estimations and optimization problems in a simple and efficient way, and it is quoted as among ‘one of 10 algorithms with the greatest influence on the development and practice of science and engineering on the 20th century’ [9].

A large number of samples is needed to ensure that all outcomes will be considered. As Monte Carlo simulation uses a large number of samples to converge, it can make it an unrealistic option for time demanding experiments or models that are computationally expensive [18].

The use of quasi Monte Carlo method, which uses a sample-set more evenly distributed than normal Monte Carlo, can reduce the number of simulations needed significantly. The underlying idea of a quasi Monte Carlo method is to select numbers generated from a deterministic scheme, as discussed in [26], and [17]. In high dimensions the quasi Monte Carlo method can encounter some problems that can be resolved by various methods [27]. In [35], three dimension reduction techniques for the quasi Monte Carlo methods are discussed and compared.

There is a vast array of literature [17],[24],[26], and [35] to name a few, describing why quasi Monte Carlo is superior to Monte Carlo in certain key areas namely; effectiveness and fast convergence.

When using Monte Carlo, it can at times be preferable to use some variance reduction techniques as described in [12]. In [18] they compare various reductions method for Monte Carlo and quasi...
Monte Carlo. The comparison between Monte Carlo methods with pseudo-random numbers, Latin Hypercube Sampling, and quasi Monte Carlo with sampling based on Sobol’ sequences shows the result show superior performance of the quasi Monte Carlo approach based on Sobol’ sequences.

The standard Monte Carlo method effectiveness is independent of the number of dimensions, unlike the quasi Monte Carlo methods, which tend to become less effective in higher dimensions [25].

Quasi Monte Carlo uses quasi-random sequences, and depending on the dimension, there are many different to chose from. For example, the Halton sequence works well in lower dimensions, up to about dimension 10 (see [12]), whereas sequences such as the Sobol’ works well in higher dimensions.

1.2 Aim of the thesis and problem formulation

Project goal and expected contents of the report

1. A summary of available methods for typical industrial optimization situations, particular focus on the Monte Carlo methods and their advantages and shortcomings.

2. Promising method applied to a simple model problem, but with all features of the large scale application (nonlinear, nonconvex, inequality and equality constraints).

3. In depth analysis and numerical experiments with an ABB product cost application, the EDS (Electrical Design System) designer tool for ABB power transformers.

All simulation is done in MATLAB R2017b.

1.3 Disposition of the thesis

This thesis is structured as follow: In Chapter 2, we review Monte Carlo and quasi Monte Carlo methods, with the focus on quasi Monte Carlo methods. We compare pseudo-random and quasi-random sequences, and how they behave in higher dimensions. These findings will be the base for the choice of sample set for the numerical experiments in Chapter 3 to 5.

In Chapter 3, we apply Monte Carlo and quasi Monte Carlo simulations to evaluate a made up cost function in MATLAB, and compare the efficiency of the two different methods. In Chapter 4, we start with optimization under conditions of uncertainties, by looking at a simple example, which can be scaled. And finally in Chapter 5, we apply the findings from Chapter 3 and Chapter 4, into a numerical experiment with cost minimization under parameter uncertainties of an ABB product.

At the end of each chapter there is a summary highlighting the findings in each sections. Throughout this report, unless otherwise stated, vector notation is written in bold type, for example vector $\mathbf{x} = x_1, x_2, \ldots x_N$. The notation for binary numbers is a lower index 2, for example $6 = 110_2$. 
Chapter 2

Monte Carlo techniques

Since the Monte Carlo method plays such an important part of optimizing with parameter uncertainties in this report, it deserves a thorough explanation. To do that it is also important to understand how sample points are generated, this chapter will explain both.

Monte Carlo methods take its name after the famous casino in Monaco, and was named by Stan Ulam and John von Neumann, while they were working on a top secret research project into nuclear fission, after Ulam’s uncle who was a frequent visitor to the casino. In 1947 John von Neuman and Stanislaw Ulam, proposed a computer simulation to solve the problem of neutron diffusion in fissionable material.

Monte Carlo simulations is a powerful statistical analysis tool and is frequently used in engineering and non-engineering fields, to simulate real-life events, ranging from simulating atom collision, Dow Jones forecasting, to the simulation of traffic flow, and industrial design problems. It is suitable in complex engineering problems, since it can handle a large number of variables, with different distribution and both linear- and non linear models.

Monte Carlo is different from a physical experiment, because the random sampling and large number of experiments are done by computer. Then the experiments statistical characteristics are observed and conclusion is drawn. The number of samples needed is denoted as $N$, observe that each experiment will need its own set of samples, and the sample mean is denoted as $\mu = \mathbb{E}[X_i]$.

Three steps are required for the Monte Carlo simulation process:

1. Sampling of random input variables $X_i$, $i = 1, 2, ..., N$ from a chosen distribution. Generate samples that are of uniform distribution between 0 and 1. The purpose of sampling the input random variables $X_i$, $i = 1, 2, ..., N$ is to represent a known distribution. This could be known, or estimated from historical data or observations. To generate sample points, we start by generating points with an uniform distribution on the interval $[0, 1]$, denoted $\mathcal{U}[0, 1]$, after which the distribution of choice can be created using well known methods, such as the inverse transform method.

There are different ways to generate a sequence of numbers that appear to be random, the two main methods are:

1. Pseudo-random generator- attempt to simulate randomness. (For Monte Carlo method).
II Quasi-random sequences- attempts to fill a space as evenly as possible. (For quasi Monte Carlo method).

ii Numerical experiment- Samples of input variables $X_i, \ i = 1, 2, ..., N$, go into the model being analysed, after which the output is evaluated.

iii Statistical analysis on model output and probabilistic characteristics of output variables, such as mean, variance, probability of failure, etc.

The above steps are the same for quasi Monte Carlo simulations, the difference is that the samples are taken from a quasi-random sequence.

**Monte Carlo error and convergence rate**

Monte Carlo has a well known slow convergence rate, though the upside is that it does not depend on dimensions, and using pseudo Monte Carlo methods the error and convergence rate is

$$O\left(\frac{1}{\sqrt{N}}\right) \text{ and } O\left(\frac{1}{\sqrt{N}}\right).$$

It can be derived by starting by using the strong law of large numbers which gives

$$\Pr(\ lim_{N \to \infty} \bar{X}_N = \mu) = 1,$$

where $\bar{X}_N$ is the sample average, and Monte Carlo converges to the mean $\mu$, as $n$ increases to infinity, with the probability of 1. We can approximate $\mu$ by generating independent identically distributed, i.i.d., samples $X_1, X_2, ..., X_N$ and computing the sample mean

$$\hat{\mu}_N = \frac{1}{N} \sum_{i=1}^{N} X_i,$$

where $\hat{\mu}_n$ denotes the statistic estimate of $\mu$.

Assuming that $X_j$ have finite variance and since $X_j$ are i.i.d., they all have the same variance, hence

$$\text{Var}\left(\sum_{i=1}^{N} X_i\right) = N \sigma^2.$$

Giving us the variance of the sample mean

$$\text{Var}(\hat{\mu}_N) = \frac{1}{N^2} \text{Var}\left(\sum_{i=1}^{N} X_i\right) = \frac{\sigma^2}{N},$$

and the standard error of the mean is $\frac{\sigma}{\sqrt{N}}$. Thus, the difference of $|\hat{\mu}_N - \mu|$ is of the order $\frac{\sigma}{\sqrt{N}}$. 

5
The error decreases at a rate $O(1/\sqrt{N})$. This means, to decrease the error by a factor 2, one needs 4 times as many samples. However, the advantage with Monte Carlo method is that the rate of the error decreases is independent of dimension.

**Variance reduction techniques for the Monte Carlo method**

There are different variance reduction techniques that are are widely used for improving the efficiency of Monte Carlo methods, this report will briefly name a few of them.

- **Stratification**: Stratified sampling is a method of sampling from a population by dividing members of the population into homogeneous subgroups before sampling. Each element in the population can only be assigned to one stratum. Then simple random sampling or systematic sampling is applied within each stratum.

- **Latin hypercube sampling**: For higher dimensional problems a useful method is to apply latin hypercube sampling instead. The idea is to sample on the d-dimensional hypercube in such a way that only the marginal distributions are stratified.

- **Importance sampling**: One of the most important variance reduction techniques is importance sampling. This technique is especially useful for the estimation of rare-event probabilities, and then there is an alternative to the standard importance sampling estimator and the following weighted sample estimator

Note that variance reduction techniques does not affect the convergence rate. In [12] there is a full description of the different variance reduction techniques, including a comparison of efficiency versus ease of use.

### 2.1 Quasi-random sequences

Quasi-random sequences, also called *low discrepancy* sequences, consists of points that are better equidistributed in a given volume than pseudo-random points. Since the object for the quasi-random sequences is to fill a $d$-dimension unit hypercube as uniformly as possible, it can simply be seen as avoiding clustering by filling the large gaps between previous numbers in the sequence.

Hence, quasi-random sequences and pseudo-random sequences are distinctly different sequences used to create a uniform sample population. A pseudo-random sequences attempt to simulate randomness, and quasi-random sequences, decidedly *not* random, attempts to fill a space as uniformly as possible.

Here, *uniform* is not meant in the probabilistic sense of a distribution, but has the meaning of an even distribution of points that avoids extreme clusters or holes. The aim is to generate numbers for which the deviation from uniformity is as small as possible. This deviation is called *discrepancy*.

The discrepancy of a non probabilistic sequence as described in [23] is: For a hyperrectangle $Q$
contained in \( I^d \) with sides parallel to the coordinate axes. Then the discrepancy \( D_N \) of the sequence of \( N \) points is
\[
D_N = \sup_{Q \subseteq I^d} \left| \frac{\text{# points in } Q}{N} - \text{volume (} Q \text{)} \right|.
\]
and if a rectangle \( Q \) contained in \( I^d \) and includes the corner at 0, we get the star discrepancy \( D_N^* \)
\[
D_N^* = \sup_{Q \subseteq I^d} \left| \frac{\text{# points in } Q}{N} - \text{volume (} Q \text{)} \right|.
\]

Low discrepancy sequences are divided into two main classes; those based on van der Corput sequences such as Halton and Sobol’, and those based on lattice methods.

**Quasi Monte Carlo error and convergence rate**

Using quasi Monte Carlo method, which uses samples from quasi-random sequences, the convergence rate, is much faster than for the Monte Carlo method. The rate of which the error reduces varies in low discrepancy sequences, however, Halton [14] showed there exists low discrepancy infinite sequences of any dimension \( d \) which satisfy
\[
D_N = O\left( \frac{(\log N)^d}{N} \right),
\]
with a convergence rate of
\[
O\left( \frac{1}{N} \right),
\]
such as Van der Corput, Halton, and Sobol’ sequences, see next section for full description of those sequences.

Observe that the size of the dimension \( d \) is important, where \( d \) either needs to be small or \( N \) needs to be large, thus, for very large dimensions the Monte Carlo method is more effective than the quasi Monte Carlo method.

### 2.2 Lattice methods

One would expect that a lattice or a grid would work well, with its evenly spread points. However, lattices have a high discrepancy and to improve the accuracy, the mesh size needs to be smaller. This can only be done by doubling the number of points, i.e., the accuracy of a uniform grid cannot be increased one point at a time.

A better way would be to have an infinite sequence of points, where you have that for each \( N \) in the sequence, all points up to, and including, \( N \) are uniformly distributed, then we would achieve a better accuracy, even when points are added one at a time. Lattices will not be discussed any further in this report.
2.3 Van der Corput sequence

A simple one-dimensional sequence was published in 1935 by the Dutch mathematician Van der Corput. It is a low discrepancy sequence on the closed interval $[0, 1]$, also called the unit interval. In [19] we learn that no infinite sequence has yet been found that has a uniformly smaller discrepancy then the van der Corput sequence. It is constructed by reversing the base $b$ representation of the sequence of positive integers. It is equidistributed, meaning it is self-avoiding, for example, if $\frac{1}{2}$ exists in the sequence, then $\frac{3}{4}$ will not be added to the sequence. The van der Corput sequence $(x_n)$ is defined [19] as:

Let $b \geq 2$ be an integer. Any number $k \in \mathbb{N}$ has a so called $b$-ary expansion of the form

$$k = \sum_{i=1}^{r} a_i b^{i-1} = a_1 + a_2 b + ... + a_r b^{r-1},$$

where $r$ is finite and digits $a_1, ..., a_r \in \{0, ..., b - 1\}$. A number of low-discrepancy sequences are based on the following transformation of natural numbers.

For $k \in \mathbb{N}$ with a $b$-ary representation $a_r, ..., a_2, a_1$. The base $b$ radical\(^2\) inverse of $k$, maps each $k$ to a point in $[0, 1)$ by flipping the coefficients of $k$ about the $b$-base decimal point, to get the $b$-base fraction of $0.a_0 a_1 a_2...$ of the form

$$\sum_{i=1}^{r} a_i b^{i-1} = a_1 b^{-1} + a_2 b^{-2} + ... + a_r b^{-r}.$$

Table 2.1: Van der Corput sequence for base 2.

<table>
<thead>
<tr>
<th>$i$</th>
<th>binary form of $i$</th>
<th>radical inverse</th>
<th>$x_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1_2$</td>
<td>0.1_2</td>
<td>1/2 = 0.5</td>
</tr>
<tr>
<td>2</td>
<td>$10_2$</td>
<td>0.01_2</td>
<td>1/2^2 = 0.25</td>
</tr>
<tr>
<td>3</td>
<td>$101_2$</td>
<td>0.0101_2</td>
<td>1/2^3 = 0.125</td>
</tr>
<tr>
<td>4</td>
<td>$1011_2$</td>
<td>0.0011_2</td>
<td>1/2^4 = 0.625</td>
</tr>
<tr>
<td>5</td>
<td>$1001_2$</td>
<td>0.00011_2</td>
<td>1/2^5 = 0.375</td>
</tr>
<tr>
<td>6</td>
<td>$1010_2$</td>
<td>0.0011_2</td>
<td>1/2^6 = 0.875</td>
</tr>
<tr>
<td>7</td>
<td>$1110_2$</td>
<td>0.00011_2</td>
<td>1/2^7 = 0.0625</td>
</tr>
<tr>
<td>8</td>
<td>$10000_2$</td>
<td>0.00001_2</td>
<td>1/2^8 = 0.03125</td>
</tr>
</tbody>
</table>

Illustration of the first nine numbers in a Van der Corput sequence in base $b = 2$.

The van der Corput sequence’s limitation is that it can only be used in one-dimension, however, since this sequence consists only of dyadic fractions it is well adapted for the use in binary computers.

\(^2\) Radical, in this context, is the adjective for radix, which is another word for base.
2.4 Halton sequences

In 1960 John H. Halton invented a more generalized sequence version compared to the van der Corput sequence, one that also uses radical inversion based points, though, not confined to only one dimension. It is a low discrepancy sequence that uses coprime numbers\(^3\) as its bases. Usually the bases \(b_1, b_2, \ldots\) are selected as the first prime numbers greater than 1, as small prime bases produces a more accurate result [6]. The sequence is constructed by systematically evaluating fractions with the prime number bases as denominator. The next number in the sequences, the numerator is increased by one, unless the numerator creates a multiple of a previous number, in that case it is disregarded, as in the van der Corput sequence.

This algorithm based on [12], describes how to form a \(d\)-dimensional Halton sequence, where \(x_i = (x_{i1}, x_{i2}, \ldots, x_{id})\) and \(d\) has the bases \(b_1, b_2, \ldots, b_d\). The \(j^{th}\) base will be used to form the \(j^{th}\) component of each vector in the sequence.

**Halton sequence algorithm:**

Begin with some integer \(m\)

1. Choose \(t_{mj}\) suitable large, represent \(m\) in each base:

\[
m = \sum_{k=0}^{t_{mj}} a_{mk} b_j^k, \quad j = 1, \ldots, d.
\]

2. Form

\[
x_{ij} = \sum_{k=0}^{t_{mj}} a_{mk} b_j^{k-t_{mj}}, \quad j = 1, \ldots, d.
\]

3. Set \(m = m + 1\) and repeat.

The number of points needed is indicated by \(t_{mj}\), and \(m\) is normally chosen to be 0, or more often\(^4\).

1. Using the algorithm, we have calculated a Halton sequence for dimension three, using the three first primes 2, 3, and 5 as bases, and \(m = 15\). The result is displayed in Table 2.2, where \(x_i\) are the calculated points, and for each step a new \(m = m + 1\) is chosen, starting with \(m = 15\). The first three points are 0.9375, 0.2593, and 0.1200, and subsequent points are 0.0312, 0.5926, and 0.3200, and they both displays an uniformly spread of points on the interval \([0, 1]\).

---

\(^3\)Coprime- two integers \(a\) and \(b\) are said to be coprime if the only positive integer that divides both of them is 1. For example 7 and 8 are coprimes, while 7 and 14 are not.

\(^4\)The 0\(^{th}\) point in a low discrepancy sequence is typically zero itself, though at times it is not possible to use zero in a simulation algorithm, and is therefore avoided.
Table 2.2: Halton sequences for bases 2, 3, and 5.

<table>
<thead>
<tr>
<th>m</th>
<th>b-ary form of m</th>
<th>radical inverse</th>
<th>x_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1111_2</td>
<td>0.1111_2</td>
<td>15/2^4 = 0.9375</td>
</tr>
<tr>
<td>15</td>
<td>120_3</td>
<td>0.0213</td>
<td>7/3^3 = 0.2593</td>
</tr>
<tr>
<td>15</td>
<td>30_5</td>
<td>0.035</td>
<td>3/5^2 = 0.1200</td>
</tr>
</tbody>
</table>

m = m + 1

| 16  | 10000_2        | 0.00001_2      | 1/2^5 = 0.0312 |
| 16  | 121_3          | 0.121_3        | 16/3^3 = 0.5926 |
| 16  | 31_5           | 0.135          | 8/5^2 = 0.3200 |

Illustration of two numbers in a three-dimensional Halton sequence, using bases 2, 3, and 5.

As higher value prime numbers are needed for each dimension added, the Halton sequence struggles produce evenly spaced points on the interval [0, 1].

Some correlation problems have been noted between sequences generated from high prime numbers. This causes the sequences to be formed almost identical, thus, defeating the whole purpose of using a low discrepancy generator. In Table 2.3, the 38th, 39th and 40th prime numbers are used, with the corresponding prime numbers, 163, 167, and 173, as bases. Such high prime numbers needs to be used for dimension 40, though for lower dimension, such as dimension d = 3, lower primes are used, as discussed earlier. The first three points are 0.09202, 0.08982, and 0.08671, and subsequent points are 0.09816, 0.09581, and 0.09249, neither shows good uniformity. See Fig.2.1 for an illustration of points in dimension 39 and 40.

Table 2.3: Halton sequences for bases 163, 167 and 173.

<table>
<thead>
<tr>
<th>m</th>
<th>b-ary form of m</th>
<th>radical inverse</th>
<th>x_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>(15)_163</td>
<td>0.(15)_163</td>
<td>15/163 = 0.09202</td>
</tr>
<tr>
<td>15</td>
<td>(15)_167</td>
<td>0.(15)_167</td>
<td>15/167 = 0.08982</td>
</tr>
<tr>
<td>15</td>
<td>(15)_173</td>
<td>0.(15)_173</td>
<td>15/173 = 0.08671</td>
</tr>
</tbody>
</table>

m = m + 1

| 16  | (16)_163       | 0.(16)_163     | 16/163 = 0.09816 |
| 16  | (16)_167       | 0.(16)_167     | 16/167 = 0.09581 |
| 16  | (16)_173       | 0.(16)_173     | 16/173 = 0.09249 |

Illustration of two numbers in a three-dimensional Halton sequence, using bases 163, 167, and 173. See Fig.2.1(c) for a graphical illustration.

One way to combat the correlation problem is to apply simple modification of the Halton sequence, by a method called skipping, i.e., using only every nth number in the Halton sequence. The size of the skip n should be a prime that is different from the prime numbers used as bases [31].
Figure 2.1: Halton points in different dimensions
The first 1000 points of the Halton sequence in dimension 40. (a) Shows projection onto the first two coordinates (bases 2 and 3), and Halton generates perfectly uniform points. (b) Shows projection onto the 11th and 12th coordinates (bases 29 and 31), and a correlation between points is noticeable. (c) Shows projection onto the last two coordinates (bases 167 and 173), and there is an obvious pattern caused by the high number prime used as a base, see Table 2.3 for numerical values of (c).

The most common solution to combat the problem the Halton sequences has with linearity in higher dimension is to use another, more suited sequences, like Sobol’ sequences.

2.5 Sobol’ sequences
The Sobol’ sequence, introduced by the Russian mathematician Ilya M. Sobol in 1967, has base 2 for all dimensions and is based on a set of direction numbers, $v_j$. The advantage with Sobol’ sequences compared to Halton sequences is that it works well in higher dimensions.

Figure 2.2: Sobol’ points in different dimensions.
The first 1000 points of the Sobol’ sequence in dimension 800. (a) Shows projection onto the 11th and 12th coordinates. (b) Shows projection onto the 401st and 400th coordinates. (c) Shows projection onto the last two coordinates. To combat the problem of correlation, scrambled Sobol’ can be used, which reorder the points in the sequence.

The algorithm for generating Sobol’s sequences is explained in details in [7]. We present a condensed version, followed by a numerical example;
Sobol’ sequence algorithm

To generate the \( j \)th component of points in a Sobol’ sequence. Start by choosing a primitive polynomial of some degree \( d_j \), for brevity we write \( d \), in the Galios field \( GF(2^n) \) (see Appendix A for further explanation of Galios fields) of the form:

\[
x^d + a_1x^{d-1} + a_2x^{d-2} + \ldots + a_{d-1}x + 1.
\]  

(2.1)

Note that the coefficients \( a_1, a_2, \ldots, a_{d-1} \) takes on the value 0 or 1.

The recurrence for calculating \( m_i \), the sequence containing positive integers \( (m_1, m_2, \ldots) \) is defined as

\[
m_i = 2a_1m_{i-1} \oplus 2a_2v_{i-2} \oplus \ldots \oplus 2^{d-1}a_{d-1}m_{i-d+1} \oplus 2^dm_{i-d} \oplus m_{i-d},
\]  

(2.2)

where \( \oplus \) denotes a bit-by-bit exclusive-or operation (addition of 0's and 1's modulo 2). When using a primitive polynomial of degree \( d \), the values for \( m_1, m_2, \ldots, m_d \) can be chosen freely as long as each \( m_i \) is an odd integer and \( m_i < 2^d \), subsequent values \( m_{d+1}, m_{d+2}, \ldots \) are calculated using the recurrence.

The direction number \( (v_1, v_2, \ldots) \) which each is a binary fraction is defined as:

\[
v_i = \frac{m_i}{2^i}.
\]  

(2.3)

To construct the Sobol’ sequence, the values of \( m_i \) needs to be selected, for the construction of the set of direction numbers \( v_i \). Since there is some freedom in the selection of initial direction numbers it is possible to receive different realisations of the Sobol’ sequence for selected dimensions.

Finally, we can form the \( i \)th point in a Sobol’ sequence as

\[
x_i = i_1v_1 \oplus i_2v_2 \oplus \ldots,
\]  

(2.4)

where \( i_k, \ k = 1, 2, \ldots \) is the binary representation of the integer \( i \). In the next section there is an numerical example of how to generate the first five points in a Sobol’ sequence.

Gray code implementation

A more efficient way to produce Sobol’ points is to use Gray code implementation [7]. The Gray code shuffles each initial segment of length \( 2^k \) of Sobol’s original sequence. It is proven that this does not affect the discrepancy of the sequences, and using the Antonov-Saleev method is much faster than Sobol’s original scheme.\(^5\)

\(^5\)In MATLAB’s Statistics and Machine Learning Toolbox- one can decide in which order the Sobol’ sequence points are produced; ‘standard’- points are produced match to the original Sobol’ sequence implementation, and ‘graycode’- uses the Gray code of the index \( i \). MATLAB’s default is ‘standard’.

12
Sobol’ sequences using Gray code is defined as

\[ x = g_{i_1}v_1 \oplus g_{i_2}v_2 \oplus ..., \] (2.5)

where \( g_{i_k} \) is the \( k^{th} \) digit from the right of the Gray code of \( i \) in binary i.e., \( \text{Graycode}(i) = ...g_{i_1}g_{i_2}g_{i_3} \).

Its speed lies in the fact that the binary representation of \( \text{gray}(i) \) and \( \text{gray}(i - 1) \) differs only in one position.

The gray code of an integer \( i \) is defined in [11] as

\[ \text{gray}(i) := i \oplus \left\lfloor \frac{i}{2} \right\rfloor = (...i_3i_2i_1)_2 \oplus (...i_4i_3i_2)_2. \]

Next we show the difference in the normal binary code vs. Gray code, and how the Gray code reorders the integers within every block of \( 2^n \) numbers for \( n = 1, 2, ..., \)

\[
\begin{array}{c|c}
 i & \text{gray}(i) \\
 1 = 0001_2 & 0001_2 = 1 \\
 2 = 0010_2 & 0011_2 = 3 \\
 3 = 0011_2 & 0010_2 = 2 \\
 4 = 0100_2 & 0110_2 = 6 \\
 5 = 0101_2 & 0111_2 = 7 \\
 6 = 0110_2 & 0101_2 = 5 \\
 7 = 0111_2 & 0100_2 = 4 \\
 8 = 1000_2 & 1100_2 = 12 \\
\end{array}
\]

and so on.

2.6 Generating a Sobol’ sequence- a numerical example

As it is not intuitive to generate Sobol’ sequences, as with the two previous sequences, we present a numerical example to show how one is constructed. In this section we use the notation given in Appendix A.

Start by choosing a primitive polynomial\(^{6}\), note that there might be a a number of different polynomials to choose from. Here we will use:

\[ x^3 + x + 1, \]

and using the notation from Eq. (2.1), we get \( a_1 = 0, a_2 = 1 \) for a third degree polynomial, i.e., \( (x^3 + a_1x^2 + a_2x + 1) \), and \( a_1 = 1, a_2 = 0, a_3 = 1 \) for a forth degree polynomial \( (x^4 + a_1x^3 + a_2x^2 + a_3x + 1) \).

\(^{6}\)S. Joe and F.Y. Kuo has, using [11], compiled a list describing which primitive polynomial and values for \( m \), to be used for how to best construct Sobol’ sequences up to 21201 dimensions, it was last updated on 16 September 2010.
Choose three starting values \( m_1 = 1, m_2 = 3 \) and \( m_3 = 7 \). We can calculate the subsequent \( m \)'s using the recurrence in the following way:

\[
m_4 = 2a_1m_3 \oplus 2^2a_2m_2 \oplus 2^3m_1 \oplus m_1
\]
\[
= 2 \cdot 0 \cdot 7 \oplus 2^2 \cdot 1 \cdot 3 \oplus 2^3 \cdot 1 \oplus 1
\]
\[
= 0000_2 \oplus 1002_2 \oplus 1002_2 \oplus 0001_2
\]
\[
= 0101_2 = 5,
\]

\[
m_5 = 2a_1m_4 \oplus 2^2a_2m_3 \oplus 2^3a_3m_2 \oplus 2^4m_1 \oplus m_1
\]
\[
= 2 \cdot 1 \cdot 5 \oplus 2^2 \cdot 0 \cdot 7 \oplus 2^3 \cdot 1 \cdot 3 \oplus 2^4 \cdot 1 \oplus 1
\]
\[
= 01010_2 \oplus 00000_2 \oplus 11000_2 \oplus 10000_2 \oplus 00001_2
\]
\[
= 00011_2 = 3,
\]
and so on.

By Eq. (2.3) we get the direction numbers \( v_i \)

\[
v_1 = \frac{m_1}{2^1} = \frac{1}{2}, \quad v_2 = \frac{m_2}{2^2} = \frac{3}{2^2}, \quad v_3 = \frac{m_3}{2^3} = \frac{7}{2^3}, \quad v_4 = \frac{m_4}{2^4} = \frac{5}{2^4}, \quad v_5 = \frac{m_5}{2^5} = \frac{3}{2^5}, \quad \text{and so on...}
\]

The \( j^{th} \) components for the first few points, are given by Eq. (2.4);

\[
\begin{array}{cccc}
i & x_i & \hline
0 & 0 & \\
1 & 1_2 & x_1 = i_1v_1 = 1 \cdot 0.1_2 = 0.5 \\
2 & 10_2 & x_2 = i_1v_1 \oplus i_2v_2 = 0.1 \cdot 1 \cdot 0.11 = 0.11_2 = 0.75 \\
3 & 11_2 & x_3 = i_1v_1 \oplus i_2v_2 = 1 \cdot 0.1 \cdot 1 \cdot 0.11 = 0.01_2 = 0.25 \\
4 & 100_2 & x_4 = i_1v_1 \oplus i_2v_2 \oplus i_3v_3 = 0 \oplus 0 \oplus 1 \cdot 0.11_2 = 0.11_2 = 0.875 \\
5 & 101_2 & x_5 = i_1v_1 \oplus i_2v_2 \oplus i_3v_3 = 1 \cdot 0.1 \oplus 0 \oplus 1 \cdot 0.11_2 = 0.01_2 = 0.375 \\
\end{array}
\]

One have to bear in mind that for each dimension, a different set of points will be needed, else all points would be correlated. This is easy achieved, either by using a different primitive polynomial or different values for \( m \), or both.
2.7 Implementing quasi-random sequences in MATLAB

When generating Halton sequences, then the problem with correlation between points in higher dimensions can be, to some extent, rectified. Methods available are; skipping the first entries in the sequence, and/or using leaped Halton, where points are excluded, or to use scrambled Halton-where MATLAB shuffles the points. All three methods are built in commands in MATLAB. However, these methods will have no or little effect on higher dimensions, where the method of Sobol’ sequences are a better choice.

The MATLAB command for a scrambled Halton sequence is ‘RR2’ and it is a permutation of the radical inverse coefficients derived by applying a reverse-radix operation to all of the possible coefficient values, in [15] the algorithm is described. For a scrambled Sobol’ sequence the command is called ‘MatousekAffineOwen’ and it is a random linear scramble combined with a random digital shift [22] for full description of the algorithm.

In [13] it is suggested that the Sobol’ sequence tends to perform better if an initial portion of the sequence is dropped and [1] the author is more specific and suggest that the number of points skipped should be the largest power of 2, smaller than the number of points to be used, i.e., for N=500, eight numbers should be skipped ($2^8 = 256$). For tests done using Sobol’ sequences in this report, unless otherwise stated, the skip function is used.

The Leap function in MATLAB’s sobolset is never used in this report, based on literature studies where in [28] it is suggested that, there is no reason to use the function Leap.

2.8 Summary

Quasi Monte Carlo method is more efficient than Monte Carlo method, except in very high dimensions, due to the use of quasi-random sequences that produces an more uniform spread of the random looking points.

Using a lattices or grids to form uniformly spaced points has the problem with high discrepancy and it has a scaling problem. To keep the symmetry of the points they cannot be added one by one, but has to be doubled.

Van der Corput produces a simple but very effective one dimensional sequence. For dimension two and above the Halton sequences gives very good results, though it has a correlation problem for higher dimensions. We have shown that from dimension ten there is a noticeable correlation problem causing the points to line up, due to the use of higher primes.

Sobol’ sequences is build on base two, therefore, the correlation problem experienced when using Halton sequences in higher dimensions are not a problem when using Sobol’ sequences. When using sobolset in MATLAB it is recommended to use the added option of scrambled and skip (skipping the number of points that are the largest power of 2 smaller than the number of points to be used).
Chapter 3

Numerical experiments on functions containing random variables

The aim for this section is to examine a simple made up function, where each component of the function contain some uncertainties, of the form of a random variable. For example, it could represent a cost function where the different inputs, like material, labour cost, and manufacturing cost, does not have an exact value but can vary within a certain range.

We evaluate the function using Monte Carlo methods, and we are not interested in the value of the function, per se, but more how the result varies depending on how each sample set is constructed. We are also interested in the number of simulations needed for the result to converge.

The chosen function is $g_k(p) = g_0(p_1, p_2, ..., p_k)$, which includes $k$ number of random variables. Through simulations we compare the value of the function rendered from Monte Carlo method using pseudo-random numbers and those from quasi Monte Carlo methods using the quasi-random Halton sequences, and Sobol’ sequences. In the first instance, $k = 6$, the function $g_6(p) = g(p_1, p_2, ..., p_6)$ has six random variables, and after that the function is scaled to $k = 30$, and $k = 90$ random variables, called $g_{30}(p)$ and $g_{90}(p)$ respectively. Notation: the random variables $p_1, p_2, ..., p_k$ will at times, be written in vector form $p = (p_1, p_2, ..., p_k)$.

3.1 Made up cost function

The cost function $g_6(p)$, where the random variables $p_i, i = 1, 2, ..., 6$ are from $\mathcal{U}(0, 1)$, is defined as

$$g_6(p) = \frac{1}{(1.01 - p_1)} + \frac{1}{(1.1 - p_2)^2} - \frac{1}{(1.1 - p_3)} + \frac{1}{(1.5 - p_4)} - \frac{1}{(1.05 - p_5)} + 2p_6.$$ 

The following two functions $g_{30}(p)$ and $g_{90}(p)$, are scaled versions of $g_6(p)$, where $g_6(p)$ is simply being repeated either 5 times for $g_{30}(p)$ or 15 times for $g_{90}(p)$.

We define $g_{30}(p)$ as

$$g_{30}(p) = \frac{1}{(1.01 - p_1)} + \frac{1}{(1.1 - p_2)^2} - \frac{1}{(1.1 - p_3)} + \frac{1}{(1.5 - p_4)} - \frac{1}{(1.05 - p_5)} + 2p_6 + ...$$

$$... + \frac{1}{(1.01 - p_{25})} + \frac{1}{(1.1 - p_{26})^2} - \frac{1}{(1.1 - p_{27})} + \frac{1}{(1.5 - p_{28})} - \frac{1}{(1.05 - p_{29})} + 2p_{30},$$
Since, all variables $p_i$ have the follows a normal distribution, the above equations can be rewritten in a more efficient way as

$$g_k(p) = \sum_{i=0}^{k/6-1} \frac{1}{(1.01 - p(6i+1))} + \frac{1}{(1.01 - p(6i+2))} + \frac{1}{(1.01 - p(6i+3))} + \frac{1}{(1.01 - p(6i+4))} + \frac{1}{(1.01 - p(6i+5))} + 2p_{6i+6};$$

where $k \in \{6, 30, 90\}$ denotes the number of random variables in the function and each $p_i$ is a i.i.d. random variable. In reality, a cost function would unlikely have the same probability distribution for all random variables.

To be able to evaluate the results from the simulations, we start by calculating the expected value and variance of $g_k(p)$.

First, to find the expected value $E[g_6(P)]$, we use Definition (5.9) of the expected value of a function of multiple random variables, taken from page 256, [34] of a continuous random variable $Y$.

**Definition 1.** Let $g(Y_1, Y_2, \ldots, Y_k)$ be a function of continuous random variables, $Y_1, Y_2, \ldots, Y_k$, with joint density function $f(y_1, y_2, \ldots, y_k)$. Then the expected value of $g(Y_1, Y_2, \ldots, Y_k)$ is

$$E[g(Y_1, Y_2, \ldots, Y_k)] = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} g(y_1, y_2, \ldots, y_k) f(y_1, y_2, \ldots, y_k) dy_1 dy_2 \cdots dy_k.$$

All the random variables in function $g_6(p)$, have uniform distribution from 0 to 1, thus, the probability density function can be described as

$$f(p) = \begin{cases} 1 & 0 \leq p \leq 1, \\ 0 & \text{elsewhere}. \end{cases}$$

The expected value of $E[g_6(P_1, P_2, \ldots, P_k)]$ is calculated using the following lemma:

**Lemma 1.** Let the variables $Y_1$ and $Y_2$ be independent random variables. For any two function $g$ and $h$ it follows

$$E[g(Y_1) + h(Y_2)] = E[g(Y_1)] + E[h(Y_2)].$$
Using Lemma 1, we calculate the expected value
\[
E[g_6(p_1,p_2,\ldots,p_6)] = \\
= \int_{[0,1]^6} \left( \frac{1}{(1.01 - p_1)} + \frac{1}{(1.1 - p_2)^2} - \frac{1}{(1.1 - p_3)} + \frac{1}{(1.5 - p_4)} - \frac{1}{(1.05 - p_5)} + 2p_6 \right) dp_1 \ldots dp_6 \\
= \int_0^1 \frac{1}{(1.01 - p_1)} \cdot dp_1 + \int_0^1 \frac{1}{(1.1 - p_2)^2} \cdot dp_2 - \int_0^1 \frac{1}{(1.1 - p_3)} \cdot dp_3 + \int_0^1 \frac{1}{(1.5 - p_4)} \cdot dp_4 \\
- \int_0^1 \frac{1}{(1.05 - p_5)} \cdot dp_5 + \int_0^2 p_6 \cdot dp_6 \\
= \left[ -\ln(1.01 - p_1) + \frac{1}{(1.1 - p_2)^2} + \ln(1.1 - p_3) - \ln(1.5 - p_4) + \ln(1.05 - p_5) + p_6^2 \right]_0^1 \\
= 10.3622... \\
\]

By the linearity of the expected value of function \(g_6(p)\), the expected value of \(E[g_{30}(p)]\) and \(E[g_{90}(p)]\) is calculated as \(E[g_{30}(p)] = 5 \cdot E[g_6(p)] = 51.8110\) and \(E[g_{90}(p)] = 15 \cdot E[g_6(p)] = 155.4334\).

To find the variance of \(g_k(p)\) we use Theorem (5.9), page 259 in [34].

**Theorem 1.** Let \(Y_1\) and \(Y_2\) be independent random variables and \(g(Y_1)\) and \(h(Y_2)\) be functions of only \(Y_1\) and \(Y_2\), respectively. Then
\[
E[g(Y_1)h(Y_2)] = E[g(Y_1)] E[h(Y_2)] \\
\]
In the following proof we use the variance of \(X\), \(\text{Var}[X]\).

**Lemma 2.** Take two functions \(f(Y_1)\) and \(g(Y_2)\) containing two independent random variables \(Y_1\) and \(Y_2\), then using Theorem 1 we can show that the variance of
\[
\text{Var}[g(Y_1) + h(Y_2)] = \text{Var}[g(Y_1)] + \text{Var}[h(Y_2)] \\
\]
**Proof:**
\[
\text{Var}[g(Y_1) + h(Y_2)] = \\
= E[(g(Y) + h(Y))^2] - (E[g(Y) + h(Y)])^2 \\
= E[(g(Y))^2 + (h(Y))^2 + 2g(Y)h(Y)] - (E[g(Y)]^2 + E[h(Y)]^2 + 2E[g(Y)]E[h(Y)]) \\
= E[(g(Y))^2] + E[(h(Y))^2] + 2E[g(Y)h(Y)] - E[g(Y)]^2 - E[h(Y)]^2 - 2E[g(Y)]E[h(Y)] \\
= E[(g(Y))^2] - E[g(Y)]^2 + E[(h(Y))^2] - E[h(Y)]^2 \\
= \text{Var}[g(Y)] + \text{Var}[h(Y)]. \qed
\]

\(^1\)Here \(E[g_{30}(p)] = 5 \cdot E[g_6(p)]\) is almost equal to 51.8110, though it is described with an equality sign, it is the same with \(E[g_{90}(p)]\).
To calculate the second moment, in our case calculate:

\[
E[(g(Y_1, Y_2, ..., Y_k))^2] = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} (g(y_1, y_2, ..., y_k))^2 f(y_1, y_2, ..., y_k) dy_1 \cdots dy_k,
\]

where the probability density function \( f \), uniform distribution in our case, is 1 in the given interval. The second moment for \( g_6(p) \) is calculated using the Lemma 1 and Theorem 1, and some standard integrals

\[
E[(g_6(P))^2] = \int_{[0,1]^6} \left( \frac{1}{(1.01 - p_1)} + \frac{1}{(1.1 - p_2)^2} - \frac{1}{(1.1 - p_3)} + \frac{1}{(1.5 - p_4)} - \frac{1}{(1.05 - p_5)} + 2p_6 \right)^2 dp
\]

\[
= 449.1037,...,
\]

thus, the variance is calculated by

\[
Var[g_6] = E[(g_6)^2] - (E[g_6])^2 = 449.1037 - 10.3622^2 = 341.7286.
\]

Since \( g_{30}(p) \) and \( g_{90}(p) \) are built with the repeating the components in function \( g_6(p) \), where all variables are i.i.d., we can use Lemma 2 to find the variance of \( g_{30}(p) \) and \( g_{90}(p) \) by

\[
Var[g_{30}] = 5Var[g_6] = 1708.64
\]

\[
Var[g_{90}] = 15Var[g_6] = 5125.93.
\]

Table 3.1: Expected value, variance, and standard deviation for \( g_k(p) \)

<table>
<thead>
<tr>
<th>Function</th>
<th>Mean</th>
<th>Variance</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_6 )</td>
<td>10.3622</td>
<td>341.73</td>
<td>18.4859</td>
</tr>
<tr>
<td>( g_{30} )</td>
<td>51.8111</td>
<td>1708.64</td>
<td>41.3357</td>
</tr>
<tr>
<td>( g_{90} )</td>
<td>155.4334</td>
<td>5125.93</td>
<td>71.5956</td>
</tr>
</tbody>
</table>

Compilation of the calculated values of the expected value and variance for the functions \( g_6(p) \), \( g_{30}(p) \), and \( g_{90}(p) \).

### 3.2 Evaluating function \( g_k \) through Monte Carlo simulations in MATLAB

In this section the mean and standard deviation for the functions \( g_6(p) \), \( g_{30}(p) \) and \( g_{90}(p) \) is estimated through Monte Carlo simulations in MATLAB. For brevity we write \( g_k \) instead of \( g_k(p) \) for the rest of this chapter. Three different sample sets are used: \texttt{rand}, \texttt{sobolset} and \texttt{scrambled}

\footnote{The default uniform random number generator in MATLAB uses an algorithm known as the Mersenne Twister, developed by M. Matsumoto and T. Nishimura, [10]. It produces floating point values in the closed interval \([2^{-53}, 1 - 2^{-53}]\), i.e., it cannot produce an exact 0 or an exact 1.}
sobolset, and four different sizes of iterations \(N \in \{10, 100, 1000, 10000\}\) are tested. The aim is to study how many iterations are needed to achieve a sufficient accuracy of the result, using the different sample sets.

Table 3.2: Mean and standard deviation using different sample sets

<table>
<thead>
<tr>
<th>k=6</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>k=30</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>k=90</th>
<th>Mean</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N)</td>
<td>(\mu)</td>
<td>(\sigma)</td>
<td>(N)</td>
<td>(\mu)</td>
<td>(\sigma)</td>
<td>(N)</td>
<td>(\mu)</td>
<td>(\sigma)</td>
</tr>
<tr>
<td>10</td>
<td>22.03</td>
<td>6.90</td>
<td>10</td>
<td>55.30</td>
<td>38.89</td>
<td>10</td>
<td>156.82</td>
<td>119.66</td>
</tr>
<tr>
<td>100</td>
<td>10.70</td>
<td>9.06</td>
<td>100</td>
<td>48.97</td>
<td>47.93</td>
<td>100</td>
<td>154.54</td>
<td>150.70</td>
</tr>
<tr>
<td>1000</td>
<td>10.73</td>
<td>10.25</td>
<td>1000</td>
<td>52.97</td>
<td>51.41</td>
<td>1000</td>
<td>152.80</td>
<td>154.50</td>
</tr>
<tr>
<td>10000</td>
<td>10.14</td>
<td>10.37</td>
<td>10000</td>
<td>52.30</td>
<td>51.78</td>
<td>10000</td>
<td>155.29</td>
<td>155.32</td>
</tr>
</tbody>
</table>

Comparing the simulated mean and standard deviation using different sample set when evaluating the function \(g_{90}\). The expected values are displayed in Table 3.1. A reminder that the standard deviation does not improve with more samples, since the standard deviation is a measurement of the "spread" of the data. If we take more measurements, we are getting a more accurate picture of the spread. The sample mean standard deviation is \(s = \frac{\sigma}{\sqrt{N}}\).

Notation: * indicates that the value changes slightly with each test run, due to the random nature of each simulation.

The tests comparing the mean and standard deviation when evaluation \(g_k, k \in \{6, 30, 90\}\) using MATLAB generated sample set rand and sobolset, showed that scrambled Sobol’ render values closer to the calculated values in comparison to unscrambled Sobol’, furthermore both performed better than rand.

### 3.3 Number of iterations needed for Monte Carlo versus quasi Monte Carlo

For complex problems, each iteration could take a long time, therefore, it could be of interest to compare how many iterations are needed of each method. The question we would like to know is: How many iterations are needed with quasi Monte Carlo methods, using scrambled sobolset versus Monte Carlo methods, using rand, to achieve the same accuracy? To try to answer this question, we will study how many times the calculated mean, using \(N\) iterations, are above or below the expected value of 155.4334 plus or minus 0.5%,1%, or 2.5%, using the two different
methods, the full result is displayed in Appendix D.1. The simulations are conducted a 1000 times for each different number of iterations, \( N \), and the percentage of times the result is within the set distant of the mean is recorded.

<table>
<thead>
<tr>
<th>Iterations ( N )</th>
<th>Sc. Sobol’ inside in %</th>
<th>U(0,1) inside in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>35.8</td>
<td>8.3</td>
</tr>
<tr>
<td>100</td>
<td>60.4</td>
<td>16.7</td>
</tr>
<tr>
<td>200</td>
<td>94.0</td>
<td>24.2</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>39.9</td>
</tr>
<tr>
<td>1000</td>
<td>100</td>
<td>53.0</td>
</tr>
<tr>
<td>10000</td>
<td>100</td>
<td>71.9</td>
</tr>
</tbody>
</table>

Table 3.3: Sobol’ versus uniform distribution convergence rate for function \( g_{90} \)

Table illustrates how close the simulated mean for the function \( g_{90} \) is compared to the calculated mean. It calculates, in percentage, how many times the simulations, out of 1000 tests, are inside the calculated mean of 155.4334 plus or minus 1%.

The results, see Table 3.3, shows how effective quasi Monte Carlo methods is, in this case, in comparison to Monte Carlo methods. Out of 1000 test runs using \( N = 100 \) iterations for each test, over 60% are within 1% of the expected mean when using scrambled Sobol’, in comparison with \texttt{rand} the same number was 16.7%.

### 3.4 Improvement techniques.

It is suggested that the efficiency of quasi Monte Carlo algorithm can be improved by reordering of the random variables [16]. We can shown here how the gap between the expected value and a quasi-random calculated value of function \( g_{90} \) can be in some instances be reduced, by simply rearranging the random variables in order of importance. This will ensure that the components with the largest influence on the result will be allocated the first part of the quasi-random sequence.

Looking at the effect each random variable has on \( g_{90} \), we compare the range, meaning in this case the lower and upper value that each component on \( g_{90} \) can take, see below:
\[ g_{90} = \frac{15}{1} \left( 1 - p_i \right) + \frac{30}{1} \left( 1 - p_i \right)^2 - \frac{45}{1} \left( 1 - p_i \right) + \frac{60}{1} \left( 1.5 - p_i \right) - \frac{75}{1} \left( 1.05 - p_i \right) + \frac{90}{2} p_i \] 
\[ \text{Range: (0.99, 1.00)} \quad \text{Range: (0.83, 1.00)} \quad \text{Range: (0.91, 1.00)} \quad \text{Range: (0.67, 2.00)} \]

The order of importance has been determined by the size of the range of each component, a larger range can contribute more to the value of \( g_{90} \), than a small range. Let the new function \( \hat{g}_{90} \) be similar to \( g_{90} \), but with the components ordered according to importance.

\[ \hat{g}_{90} = \frac{15}{1} \left( 1.1 - p_i \right)^2 + \frac{30}{1} \left( 1.01 - p_i \right) - \frac{45}{1} \left( 1.05 - p_i \right) - \frac{60}{1} \left( 1.1 - p_i \right) + \frac{75}{2} p_i + \frac{90}{1.5} \left( 1.5 - p_i \right) \]

Figure 3.2 displays the results from using the \( \hat{g}_{90} \) versus \( g_{90} \). As expected, using the reordered function of \( \hat{g}_{90} \) gave much better result due to the correlation problem Halton sequences has in higher dimensions, as discussed earlier. Furthermore, when using Sobol’ sequences, there was no improvement when using the re-ordered \( \hat{g}_{90} \) versus \( g_{90} \), though, it shows how Sobol’ performs better than Halton, re-ordered or not in higher dimensions.

(a) The improvement using importance order when calculating the value of a function, for Halton sequences is clearly visible.
(b) There is no obvious improvement using importance order when calculating the value of a function, for Sobol’ sequences.

**Figure 3.2: Mean of functions \( g_{90} \) and \( \hat{g}_{90} \)**

Mean of the functions \( g_{90} \) (red) and \( \hat{g}_{90} \) (reordered-blue) are compared with the calculated expected mean. In the reordered \( \hat{g}_{90} \), the more important components of function \( g_{90} \) has been identified and put in an ascending order, to ensure those are sampled using the smaller prime bases. Both Halton- and Sobol’ sequences are used, and the result shows a significant improvement for the Halton sequences, whereas the result for the Sobol’ is inconclusive.
3.5 Summary

In this section we used Monte Carlo and quasi Monte Carlo simulations to evaluate the mean of a function containing multiple variables, and compare different Monte Carlo and quasi Monte Carlo methods. For the Monte Carlo method the MATLAB command `rand` was used to create random variables from the sample set $\mathcal{U}(0,1)$, and for the quasi Monte Carlo methods the MATLAB commands `haltonset` and `sobolset` was used with the optional functions of skipping, leaping or scramble. Tests showed that when using `sobolset` it is preferable to use the skip and scramble functions, as recommended in literature.

Comparing the two quasi Monte Carlo methods, it has been shown that for functions containing a large number of variables, i.e., high dimension, the Sobol’ sequences outperforms Halton sequences, which is consistent with empirical evidence. This is caused by the Halton sequence having correlation problems in dimensions $d = 6$, or higher.

By rearranging the function where the components witch influence the value of the function are placed first, thus, using the first part of Halton sequences shows an improvement, however, the problem can be avoided by using Sobol’ sequences instead. For that reason, the tests in the rest of the report will be done using Sobol’ sequences.

In the first test we compared the mean and standard deviation when evaluation a function $g_k(p)$ using MATLAB generated sample set `rand` and `sobolset`, and it showed (with small differences) that scrambled Sobol’ rendered values closer to the expected mean than unscrambled Sobol’, furthermore both performed better then `rand`.

In the second test we compared the number of iterations needed to achieve a result where the simulated mean is within a set distance from expected value. Here the quasi Monte Carlo using scrambled `sobolset` outperformed `rand`.

Simulation showed, if the aim is to be within 1% of the expected mean of the test function $g_{90}(p)$ in at least 90% of the simulations, we showed that quasi Monte Carlo simulations needed less than 200 iterations vs. over 10000 iterations for Monte Carlo simulations.
Chapter 4

Non-linear optimization problem containing uncertainties

In mathematical optimization, the optimum solution is sought to a objective function given some choices, and the optimal solution needs to satisfy all specified constraints, within a defined domain.

An optimization problem consists of an objective function $f(x)$, which is to be minimized, or maximized, in this report we will always seek to minimize the objective function. The constraints could be either inequality constraints or equality constraints, where the latter is naturally more restrictive.

The standard mathematical optimization problem is defined as

$$\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad g_i(x) \leq 0 \quad i = 1, \ldots, m \\
& \quad h_j(x) = 0 \quad j = 1, \ldots, n,
\end{align*}$$

(4.1)

where $g_i, i = 1, \ldots, m$, contains the inequality constrains functions, and $h_j, j = 1, \ldots, n$, contains the equality constraint functions. A point $x$ is feasible if it fulfills $g_i$ and $h_j$, and the feasible set $D$ is made up by all feasible points. A feasible point $x^*$ is called a local optimizer if $f(x^*) \leq f(x)$ is true for all feasible points $x$ in the neighbourhood of $x^*$, and a global optimizer if it holds true for the whole feasible set.

If the objective function and the constraint functions contains only linear functions, the optimization problem can be solved using well established methods like the Simplex method, though, more advanced methods are needed for nonlinear problems.

To verify optimality for nonlinear optimization problems the Karush-Kuhn-Tucker (KKT) conditions [5], also known as the Kuhn-Tucker conditions, are used. Since, when using Lagrangian multiplier the constraints are limited to be only equalities, and if our constraints also have inequalities, we need to extend the method to the KKT conditions.

The KKT conditions are first order necessary conditions for a point $x^*$ to be an optimum, and describes the relation between the gradient of the objective function and the gradient of the active constraints.
They are defined as
\[ \nabla f(p) + \sum_{j=1}^{n} \lambda_j \nabla h_j(p) + \sum_{i=1}^{m} \mu_i \nabla g_i(p) = 0 \]
\[
j_j = 0, \quad j = 1, \ldots, n
\]
\[
g_i \leq 0, \quad i = 1, \ldots, m
\]
\[
\mu_i \geq 0, \quad i = 1, \ldots, m
\]
\[
\mu_i g_i(x) = 0, \quad i = 1, \ldots, m.
\]

The first equation implicitly states that the gradients of \( f(p) \), and \( g_i(p) \) and \( h_j(p) \) are parallel, however, it does not state that they are of the same magnitude, and therefore, they need to be scaled by \( \lambda_j \) and \( \mu_i \). The Lagrange multipliers \( \lambda_j \) and \( \mu_i \) also states whether the constraints \( g_i \) and \( h_j \) are active or non-active. If a multiplier is zero, this indicates that the corresponding constraints are non-active, see Fig.4.1. If an optimum lies in the interior then the gradient of the objective function is equal to zero. The last condition is the complementary slackness condition. See Appendix C for proof of the Kuhn-Tucker conditions.

**Figure 4.1: Active and non-active constraints.**

Graph showing how an active and non-active constraints will affect the optimal point. The round circles are the contour curves of the objective function. Both \( g_1 \) and \( g_2 \) are active constraints and the optimal point will lie on the joint edge of those constraints, closest to the center of the contour curves of the objective function. Here \( g_3 \) is a non active constraint, therefore \( \mu_3 \) is zero.

### 4.1 Nonlinear optimization in MATLAB

There are a number of optimization solvers to choose from in MATLAB. The solver \texttt{fmincon} optimizes nonlinearly constrained problems (as in the case of the problems in this report), using gradient based methods. It starts from a provided initial point, then uses the gradient to move to a better point, and finishes when a stopping criteria is met.

#### 4.1.1 Gradient based methods

Gradient based methods use the gradient, the first derivatives and the Hessian, the second derivatives, of the objective function, to find a local minima.
The gradient $\nabla f$ is a vector containing all partial derivatives of function $f(x)$, where $x$ is the n-vector $x = (x_1, x_2, \ldots, x_n)$ and can be expressed as

$$\nabla f(x_1, \ldots, x_n) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}.$$ 

In the multivariate case, the gradient vector is perpendicular to the hyperplane tangent of the contour surfaces of constant $f(x)$ (visualize a flat paper touching the surface of a balloon).

The Hessian $H$ is a square matrix containing all partial second derivatives of function $f(x)$, and can be expressed as

$$H(x_1, \ldots, x_n) = \nabla^2 f(x_1, \ldots, x_n) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}.$$ 

In Sequential Quadratic Programming (SQP), a sequence of Quadratic Program subproblems is solved in order to find the descent direction [21]. Then a line search is performed to decide the step length and the KKT conditions ensure that a local minima is reached.

Quadratic Program subproblem is formulated as

$$\text{minimize } d \quad \frac{1}{2} d^T H_k d + \nabla f(x_k)^T d$$
$$\text{subject to } \nabla g_i(x_k)^T d + g_i(x_k) \leq 0 \quad i = 1, \ldots, m$$
$$\nabla h_j(x_k)^T d + h_j(x_k) = 0 \quad j = 1, \ldots, n,$$

where the Hessian is positive definite, to ensure a minimum is found. Every iteration $k$ computes a new Hessian $H_k$ and solve the corresponding subproblem. The search direction $d_k$, the optimal value of $d$, is used to chose the next point, which is closer to the local minima, as follows

$$x_{k+1} = x_k + \alpha_k d_k.$$ 

The step length parameter $\alpha_k$ is determined by a line search, and the procedure continuous until a stopping criteria is met.

In fmincon, two variants of SQP methods called active-set and sqp are used, see next section.

26
4.1.2 Algorithms to choose from in fmincon

In fmincon there are four different algorithms to choose from [21]. If no algorithm is chosen using option, then the default algorithm interior-point is used.

1. interior-point algorithm (default)

The interior-point method, also called barrier method, successively solves a sequence of approximate minimization problems. A log-barrier penalty term is used for the inequality constraints by introducing one slack variable for each of the inequality constraints and the problem is reduced to having only equality constraints. See [21] for full description.

The interior-point can handle large, sparse problems.

For poorly scaled problems, where some components have much different scales than other components, it is recommended to use the option ScaleProblem set to 'obj- and-constr', this will normalize the problem, which makes the solving faster and more accurate. The option ScaleProblem is available for the interior-point and sqp algorithms only.

2. active-set algorithm

A SQP algorithm, which make use of a quasi-Newton method to iterate toward a solution that satisfies the Karush-Kuhn-Tucker conditions. It is called a quasi-Newton method since the Hessian is not computed exactly due to high computational cost, but rather approximated using an update scheme, see [2], for further explanation.

SQP implementation consists of three main stages

(a) Compute gradient and update the Hessian
(b) Set up and solve QP to find descent direction
(c) Perform a line search to find a step length

The active-set can take intermediate infeasible steps outside bounds.

3. sqp algorithm

This algorithm is very similar to the active-set algorithm, though there are some differences. During its iterations, the sqp algorithm can attempt to take a step that fails. This means an objective function or nonlinear constraint function you supply returns an invalid value. In this case, the algorithm makes the step smaller and attempts again. Also, sqp uses more efficient linear algebra algebra routines to solve the quadratic programming subproblem, that both require less memory and are faster than the routines used in the active-set method.

The option ScaleProblem can be used to normalize poorly scaled problems.

4. trust-region-reflective algorithm

Starting from a chosen point \( x \), the algorithm approximates \( f \) with a simpler function \( q \), which represents the behaviour of function \( f \) in a neighborhood \( N \), around the point \( x \). Restrictions
are that it requires that a gradient is provided, and allows only bounds or linear equality con-
straints, not both. Within these limitations, the algorithm handles both large sparse problems
and small dense problems efficiently. This algorithm will not be used in this report.

### 4.1.3 GlobalSearch, MultiStart, and patternsearch

The non gradient based optimization method patternsearch is a direct search method, where
a search is conducted systematically around the current point, by testing for feasibility and a lower
objective function value. This will continue until the stopping criteria is met [21]. This method can
be used for non differentiable functions, though, with no gradient to provide directional information,
makes it harder to find the global minimum. This method will not be used in the experiments
in this report.

The algorithms GlobalSearch and MultiStart have similar approaches to finding global or
multiple minima, both starts fmincon from multiple starting points and use these starting points
to sample multiple basins of attraction. The basin of attraction for steepest descent is the set of
initial values leading to the same local minimum, i.e., initial values $x(0)$ that are close to each other
give steepest descent paths that tend to the same minimum point.

Differences between GlobalSearch and MultiStart are:

GlobalSearch runs fmincon from a given start point. If this run converges, then GlobalSearch
records the start point and end point for an initial estimate on the radius of a basin of attraction, as
well as the value of the objective function. GlobalSearch repeatedly examines trial points and
throws away points that will not generate a better solution.

MultiStart uses several initial guesses and solves them using fmincon. The result of each
run is saved in a vector and after which the best result is chosen as the global minima. MultiStart
can run in parallel, distributing start points to multiple processors for local solution.

### 4.2 Optimizing problems containing uncertainty

Most real-world optimization problems contains some uncertainty, either in the objective function
or in the constraint functions, and these can come from a multitude of sources, such as, imperfect
material properties and manufacturing processes, or uncertain operating environment, etc. All of
these, will most likely, have an impact on the design, therefore, these uncertainties should be con-
sidered in design process. Often models simply ignore the uncertainty, and evaluate the expected
value instead, this is done to avoid having to optimize with uncertainty, which is not always a trivial
task.

Stochastic Programming and Robust Optimization are the most popular frameworks for incor-
porating uncertainty into optimization. Stochastic programming uses random variables with known
or estimated probability distributions to characterise the uncertainty. Robust optimization, also
called worst-case scenario optimization, uses a set of outcomes to characterise the uncertainty and
optimizes a worst possible case of the problem.
When optimizing there could exist uncertainty in both the objective function and in the constraint functions. Let us call a vector containing all uncertainties \( \xi = (\xi_1, \xi_2, ..., \xi_k) \), then the minimization of function \( f(x, \xi) \) can be formulated as

\[
\begin{align*}
\text{minimize} & \quad f(x, \xi) \\
\text{subject to} & \quad g_i(x, \xi) \leq 0 \quad i = 1, \ldots, m \\
& \quad h_j(x, \xi) = 0 \quad j = 1, \ldots, n,
\end{align*}
\]

where \( g_i, i = 1, \ldots, m \) contains the inequality constraints functions and \( h_j, j = 1, \ldots, n \) contains the equality constraint functions.

### 4.2.1 Stochastic optimization

Stochastic optimization takes into account that the random variables probability distribution is known, or can be estimated, and the object is to find the best on average. There are different approaches to stochastic optimization, one common method is the two stage programming model [4][32], where the decision variables are divided into one first-stage variables, and second-stage variables. The first-stage variables are "here-and-now" decisions made before a random event happens and is based on the data available at the time, and the second-stage variables are "wait-and-see" decisions, which are acted upon after realisation of the uncertain data. An extension of this method is multi-stage programming model, where future outcomes of scenarios are unknown to the decision maker, and a new optimization is performed in each time step after new information is known.

One of the problem is how to define the objective in a problem which includes stochastic parameters. Again, a common solution, and the easiest way to deal with uncertainty, is to using an estimate, thus, solving the problem using an estimate. This might not always be the best practice, since in some situations a robust solution is sought, where all possible outcomes is feasible. For example, if an optimal solution, based solely on the minimum expected cost, might have a large standard deviation which in the worst case will lead to a very high cost.

### 4.2.2 Robust optimization

Robust optimization is used to find a robust optimal solution, i.e., an optimal solution which is insensitive to uncertainties and is robust for all possible realizations of a uncertainty set. The size of the uncertainty set depends on the required robustness, and goal is to make a decision that is feasible no matter what the realization of \( \xi \) is. At times, a robust solution can be too rigid, then a more flexible solution can be obtained by softening the constraints or by substituting the constraints into penalty costs in the objective function.

There are different definitions of robustness, generally we can distinguish between constraint robustness where the data uncertainty affects the feasibility of the potential solution, and objective robustness where the uncertainty lies in the objective function and the feasibility constraints are fixed. In both constraint and objective robustness, the solution optimizes the worst-case scenario.

29
In some cases it is impossible to find a robust solution as some realisations of the uncertainty may generate an infeasible solution, however, in such case, one may consider softening the the level of robustness. This can be done by relaxing the constraints by introducing chance constraints or soft constraints, see [33] for definitions.

4.2.3 How to deal with equality constraints and uncertainties in optimization

To obtain a feasible solution when solving an optimization problem with equality constraints, all equality constraint must be strictly satisfied. This creates a dilemma, since the uncertainties makes it a probabilistic problem, strict satisfaction of equality constraint might not always be possible. Three approaches to deal with this problem is suggested in [29]

- **Constraint relaxation**: only suitable when the equality constraints does not need to be satisfied exactly, or needs to hold "most of the time".
- **Mean value satisfaction**: satisfies the equality constraints at their mean values, though this approach does not take into account the variations on the constraints.
- **Elimination**: can be used for those equality constraints that must be satisfied under uncertainty. For example, such constraints might be physical law of nature, laws of motion, or the law of conservation of energy. This example describes elimination in an optimization problem without uncertainties.

\[
\begin{align*}
\text{minimize} \quad & x_1 - x_2 x_3 + 4 \\
\text{subject to} \quad & x_1 - 4 x_2 = 0 \\
& x_1 - 3 x_2 + x_3 = 0.
\end{align*}
\]

By substituting \( x_1 = 4x_2 \) and \( 4x_2 - 3x_2 + x_3 = 0 \Rightarrow x_3 = -x_2 \) into the objective function, the constraints has been eliminated. The problem has changed from a constrained to an unconstrained optimization problem, i.e.,

\[
\text{minimize} \quad x_2^2 + 4x_2 + 4.
\]

Caution has to be used when eliminating variables containing uncertainty, since after the elimination, and the resulting problem has been solved, it is unclear which values to use for the eliminated variables, since they also depend on the uncertain parameters [3].

4.3 Penalty substituted for constraints

Robust optimization can at times deemed be too restrictive, where a violation of a constraint can be regarded as an infinite cost. Since violations of constraints leads to an unfeasible solution, the constraints can be substituted for penalty costs in the objective function.

Instead of the constraints \( g_i(x, \xi) \) and \( h_j(x, \xi) \), we add a cost whenever a constraint is violated. The cost is often exponentially proportional to the size of the violation, at a rate \( \lambda_i > 0 \). This makes
The Cantilever beam, attached at one end is subjected to some, vertical force $Y$ and horizontal force $X$, causing some displacement at the tip, and stress at the fixed end.

Large violations very expensive, while small violations will still render a feasible solution, i.e., the penalty is very small for small violation. The constraint functions $g_i(x, \xi)$ and $h_j(x, \xi)$ are removed from the problem formulation and the penalty functions $\theta_i(g_i(x, \xi))$ and $\psi_j(h_j(x, \xi))$ are added as a costs in the objective function.

The new problem formulation is

$$\min_x f(x, \xi) + \sum_{i=1}^{m} \theta_i(g_i(x, \xi)) + \sum_{j=1}^{n} \psi_j(h_j(x, \xi)),$$

where

$$\theta_i(\alpha_i) = \begin{cases} 0 & \text{when } \alpha_i \leq 0 \\ \lambda_i \alpha_i & \text{when } \alpha_i > 0 \end{cases}, \quad \text{and} \quad \psi_j(\beta_j) = \begin{cases} 0 & \text{when } \beta_j \leq 0 \\ \mu_j \beta_j & \text{when } \beta_j > 0 \end{cases},$$

and $\alpha_i$ and $\mu_j$ are the size of the violations. The penalty functions $\theta_i$ and $\psi_j$ can be linear, as in this case, or more likely take a nonlinear form, such as exponential. The latter makes more sense, with the penalty increasing exponentially, as you move away further from the feasible zone.

4.4 Numerical experiment: Optimizing the design of a Cantilever beam

The purpose of this example is to compare and evaluate the different algorithms used in MATLAB’s nonlinear optimization solver, fmincon.

The objective of the problem is to minimize the weight of a cantilever beam, by designing the smallest possible cross section area $f(b, h) = bh$, while not exceeding allowable displacement and stresses. The cantilever beam has the length $L$, and is attached at one end, and at the opposite end subjected to some $Y$-vertical force and $X$-horizontal force, see Fig. 4.2. The beam is made out of steel with the modulus of elasticity $E$. 
The width $b$ and the height $h$ of the beam are deterministic design variables, and are to be determined. Two inequality constraint functions are considered.

The first constraint is that the tip displacement must not exceed an allowable value $D_0$,

\[
\frac{4L^3}{E} \sqrt{\left( \frac{X}{b^3h} \right)^2 + \left( \frac{Y}{bh^3} \right)^2} \leq D_0.
\]

The second constraint is the maximum stress at the fixed end of the beam is less than the yield strength $S$,

\[
\frac{6L}{bh} \left( \frac{X}{b} + \frac{Y}{h} \right) \leq S.
\]

In a real, physical problem, it is very unlikely for all the inputs to be exact, and as discussed in an earlier chapter, often the problem is optimized using average values. In this example we will first optimize using the expected value of the inputs with three different optimization algorithms in MATLAB’s `fmincon`; they are active set, interior point, and sqp. Thereafter, we will look at the same problem with some uncertainties added, and using quasi Monte Carlo simulations, with Sobol’ sequences as our quasi-random point generator.

### 4.4.1 Optimization using the the expected value

The cross section of the beam are design in the optimal way using the expected value of the inputs $X, Y, L, E,$ and $S$. The maximum displacement of the tip, $D_0$ is considered to be constant.

**Given data for the expected value of inputs:**

- $b$ : width of beam -a design variable (m)
- $h$ : height of beam -a design variable (m)
- $E[X] = 400$ : horizontal force (N)
- $E[Y] = 800$ : vertical force (N)
- $E[E] = 200$ : modulus of elasticity (GPa)
- $E[L] = 1.00$ : length of beam (m)
- $E[S] = 200$ : yield strength (MPa)
- $D_0 = 0.025$ : maximum allowed displacement of tip (m).

The cantilever beam dimensions are optimized using the the expected value of the variables, for brevity, in this section, we will drop the expected value notation in the constraint, i.e., $E[X]$ is denoted as $X$. 

32
The optimization model is given by

\[
\begin{align*}
\text{minimize} & \quad \mathbb{E}_\xi \left[ f(b, h, \xi) \right] = bh \\
\text{subject to} & \quad 4L^3 \frac{X^2}{E} \sqrt{\left( \frac{X}{b^3} \right)^2 + \left( \frac{Y}{bh^3} \right)^2} - D_0 \leq 0 \\
& \quad 6L \frac{X}{bh} \left( \frac{X}{b} + \frac{Y}{h} \right) - S \leq 0
\end{align*}
\]  

(Displacement constraint)  

(Maximum stress constraint)  

(4.3)

The starting point is denoted \( d_0 \), and can also be described as \( d_0 = (b_0, h_0) \). The test shows that \textit{active set} works well in all starting points, even if the starting point is outside the feasible zone, i.e., both \( d_0 = (0.01, 0.01) \) and \( d_0 = (0.02, 0.02) \) are outside the feasible zone. The \textit{interior point} showed some problems near the infeasible zone, where it exceeded maximum allowed iterations, and \textit{sqp} converges to an infeasible point if starting outside the feasible zone. This problem is caused by the large differences in size of the variables, ranging from \( 25 \cdot 10^{-3} \) to \( 200 \cdot 10^9 \) in the same formula. However, it can be solved by normalizing the variables during optimization, by setting the option \textit{’ScaleProblem’} to \textit{’obj-and-constraint’}. The full results from optimizing using the expected value and starting from different points, is illustrated in Appendix D.2 Table D.2.

### 4.4.2 Optimization with uncertainties using quasi Monte Carlo method

We will look at the same problem as in Eq. (4.3), however in this case the objective function, the area of the cantilever, is optimized using in-data containing uncertainties. The objective function, is optimized for different realizations of the uncertain data, using quasi Monte Carlo simulation, giving us an average value for the area and dimensions of \( b \) and \( h \).

The optimization is performed with \( N = 10000 \) numbers of different samples points, generated using scrambled \textit{sobolset}, where the first five points are skipped. We are interested in how accurate the three algorithms \textit{active set}, \textit{interior point}, and \textit{sqp} are, and how long each algorithm takes to perform, and the result is illustrated in Table 4.1.

**Given data, for uncertainty:**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X \sim \mathcal{U}(350, 450) )</td>
<td>Vertical force</td>
<td>(N)</td>
</tr>
<tr>
<td>( Y \sim \mathcal{U}(700, 900) )</td>
<td>Horizontal force</td>
<td>(N)</td>
</tr>
<tr>
<td>( E \sim \mathcal{U}(180, 220) )</td>
<td>Modulus of elasticity</td>
<td>(GPa)</td>
</tr>
<tr>
<td>( L \sim \mathcal{U}(0.99, 1.01) )</td>
<td>Length of beam</td>
<td>(m)</td>
</tr>
<tr>
<td>( S \sim \mathcal{U}(180, 220) )</td>
<td>Yield strength</td>
<td>(MPa)</td>
</tr>
<tr>
<td>( D_0 = 0.025 )</td>
<td>Maximum allowed displacement of tip</td>
<td>(m).</td>
</tr>
</tbody>
</table>
The starting point at \( d_0 = (b_0, h_0) = (0.04, 0.04) \), is inside the feasible zone, though it is close to the infeasible zone, appeared to be a problem for two of the algorithms: \texttt{sqp} and \texttt{interior point}, where they both fail to converge. This problem is rectified by the added option of normalizing the constraints and the objective function by their initial values.

Table 4.1: Running time and average value for the three different algorithms in \texttt{fmincon}.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>b</th>
<th>h</th>
<th>A</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>active set</td>
<td>0.0229031</td>
<td>0.0458712</td>
<td>0.0010506</td>
<td>97</td>
</tr>
<tr>
<td>interior point</td>
<td>0.0229362</td>
<td>0.0458178</td>
<td>0.0010509</td>
<td>612</td>
</tr>
<tr>
<td>sqp</td>
<td>0.0229033</td>
<td>0.0458709</td>
<td>0.0010506</td>
<td>63</td>
</tr>
</tbody>
</table>

Comparing running time and average value for the three different algorithms, starting at point \( d_0 = (b_0, h_0) = (0.04, 0.04) \), and \( N=10000 \).

Test runs shows that the three algorithms render slightly different results, \texttt{active set} and \texttt{sqp} have similar results, while \texttt{interior point} finds a minima different to the two algorithms. In [21], they explain why the solution can be slightly less accurate using \texttt{interior-point}, and that the reason for this potential inaccuracy is that the barrier function keep iterating away from inequality constraint boundaries. In tests results, not displayed here, the \texttt{interior-point} algorithm renders the same result as the two other algorithms, when starting at \( d_0 = (0.09, 0.19) \) which is not near the infeasible boundary. Noteworthy, is the difference in time it takes for the different algorithms. The fastest algorithm is \texttt{sqp}, there after it is \texttt{active set}, taking a considerable longer time is the \texttt{interior point} (which is MATLAB’s chosen default algorithm).

### 4.4.3 Substitute constraint for penalties

In this example we demonstrate how the cantilever problem could be formulated by exchanging constraints with penalties. The constraints are moved to the objective function by introducing penalty functions.

The optimization problem is reformulated as

\[
\text{minimize } f(b, h, \xi) = bh + \sum_{i=1}^{2} \theta_i(g_i(b, h, \xi))
\]

where

\[
1 \leq b \leq 10 \quad \text{(Boundary for beam width)}
\]
\[
1 \leq h \leq 20 \quad \text{(Boundary for beam height)}
\]
\[ \theta_1(\alpha_1)= \begin{cases} 0 & \text{when } g_1(b,h,\xi) = \frac{4L^3}{E}\sqrt{\left(\frac{x}{b^3h}\right)^2 + \left(\frac{y}{bh}\right)^2} - D_0 \leq 0 \\ \lambda_1 \alpha_1 & \text{when } g_1(b,h,\xi) = \frac{4L^3}{E}\sqrt{\left(\frac{x}{b^3h}\right)^2 + \left(\frac{y}{bh}\right)^2} - D_0 > 0, \end{cases} \]

and

\[ \theta(\alpha_2)= \begin{cases} 0 & \text{when } g_2(b,h,\xi) = \frac{6L}{bh}\left(\frac{x}{b^3} + \frac{y}{bh}\right) - S \leq 0 \\ \lambda_2 \alpha_2 & \text{when } g_2(b,h,\xi) = \frac{6L}{bh}\left(\frac{x}{b^3} + \frac{y}{bh}\right) - S > 0. \end{cases} \]

Note that both the constraint functions are positive penalties in the objective function, and the size of the penalty cost depends on \( \lambda_i, i \in \{1,2\} \).

### 4.5 Summary

We optimized the Cantilever beam design variables width \( b \) and height \( h \), using three different algorithms, active set, interior point and sqp, in the nonlinear bounded constraint problems optimizer fmincon. The test showed that active set worked well in all starting points, even if the starting point was outside the feasible zone. The interior point showed some problems near the infeasible zone, where it exceeded maximum iterations. The sqp converged to an infeasible point if starting outside the feasible zone. This problem is caused by the large differences in size of the different variables, and could be solved using the added option of normalization of the variables, by adding the option ‘ScaleProblem’, ‘obj-and-constraint’.

In the second simulations the variables had uncertainties added, we then used quasi Monte Carlo simulations (\( N = 10000 \)), to find the best design on average, using again three different algorithms in fmincon. The active set and sqp gave similar results, and interior point differed quite a bit in the value for both \( b \) and \( h \) from the other two algorithms, there were also a significant difference in time it took for each algorithm to run the optimization 10000 times. It took sqp 50 sec., active set 86 sec., and interior point took 427 sec.

Most interesting, is the finding that MATLAB’s default solver interior-point, render an answer less accurate then active set and sqp, and this has been confirmed in MATLAB user manual [21] that at times this is the case. For large optimization problems, a solution which is, a only accurate up till the third significant digit, might be deemed as not very impressive.
Chapter 5

Numerical experiments with cost minimization of an ABB product

In this section we will apply what we learnt from the two previous chapters and optimize an ABB cost function. This cost function calculates the price of a transformer, where the design variables is optimized in the process. The aim is to calculate the cost of a transformer should a large number of the parameters and formulas contain some uncertainties, and then evaluate the results. We can thereby see how a cost, normally accepted as certain, can actually vary in price if the uncertainties are taken into consideration.

Figure 5.1: Tailor made HVDC transformer.
ABB has won orders worth over $300 million to supply its breakthrough technologies for the world’s first 1100 kV ultra-high-voltage direct current transmission link.
5.1 Background

ABB is one of the worlds leading producers on high voltage transformers and offers bespoke designs, where each transformer is design and built, by highly qualified engineers to meet each customers exact requirements. When a customer orders a high voltage (HV) transformer they specify certain requirements such as; frequency, maximum size and weight, as well as specify their running cost of the transformer with load vs. with no load. A design engineer will then optimize the design variables using a program called the Optimizer in the EDS (Electrical Design System). It will decide the dimension of the transformer, type of winding, cooling ducts, etc.

The Optimizer will come up with a list of suggested designs, each with different advantages, such as low productions cost, low running cost, etc., as some customer the might just look at the purchase price, while others takes into consideration the running cost during the full life of the transformer. Once the design is finalized, and the designer chosen a design, it is priced using the Uniform Cost Model (UCM) program in the EDS. The UCM is the official cost model for BU Transformers (small, medium, and large power transformer).

5.1.1 Technical background

A HV transformer is a large, custom-built piece of equipment that is a vital component of the bulk transmission grid. Since HV transformer are very expensive and tailored to to customer’s specification, they are each individually designed.

![Figure 5.2: Three-phase transformer, the Trafostar, manufactured by ABB](image)

Illustration of a standard core-type Large Power Transformer and its major internal components. This transformer is a liquid-filled three-phase transformer, the Trafostar, manufactured by ABB.
The transformers come in a wide variety of sizes and configurations, they consist of two main active parts: the core and the windings. The core is made of high quality silicon electrical steel, cut and layered in pieces, and the the windings are made out of copper conductors wound around the core, providing electrical input and output. Out of the raw materials, copper and electrical steel, are a significant factor in a power transformers price.

Two basic configurations of core and windings exist: the core form and the shell form. In a shell-type power transformer, both primary and secondary are on one leg and are surrounded by the core, and in a core-type power transformer, cylindrical windings cover the core legs.

5.2 Task

The task is to incorporate uncertainty in the optimization stage of the transformer design. Since pricing all parts included in a transformer is a very big task, a part of the transformer is chosen. We will price the manufacturing cost of the core and windings, and this could serve as a guidance to how much the total price of a transformer can vary.

Under normal circumstances all inputs; cost parameters, and internal formulas, are treated as given, and the aim is to show that if there are some uncertainties in the parameters, this will influence the price, and therefore, the choice of design.

The cost function being optimized consists of; cost of material, labour cost for manufacturing of the core and the windings, and cost of losses. This differs from the standard procedure of pricing the transformer, which is done in two stages, of first optimizing the general design in Optimizer, then pricing the chosen design using UCM. In our model the pricing of material and labour are all included in the optimization stage. Where a separate optimization is done for each realization of the uncertain data. The number of optimizations are decided by the size of N, which is the number of quasi Monte Carlo simulations performed.

This task will minimize the cost function depending on the design variables; Limb height (mm) of the transformer legs, the Core diameter (mm), the Winding Area normalized, and the Flux density (T). Starting with a general design of a Trafo transformer, see Fig. 5.2 with; three main limbs, no side limbs, 100% magnetic material in core clamps and core clamp detail materials, core type-‘T’, core steel grade- ‘27’, and ‘WindingConductorType’=S. In a real setting the customer will give certain specifications before the optimization, and in our design they are:

The ’customer’ specifications given for this transformer are:

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Loss load</td>
<td>110 kW</td>
</tr>
<tr>
<td>Load loss evaluation</td>
<td>4000 USD/kW</td>
</tr>
<tr>
<td>No Load loss evaluation</td>
<td>2000 USD/kW</td>
</tr>
<tr>
<td>Frequency</td>
<td>50 Hz</td>
</tr>
<tr>
<td>Reactance</td>
<td>10.5 %</td>
</tr>
<tr>
<td>Maximum tank width</td>
<td>4000 mm</td>
</tr>
<tr>
<td>Maximum tank height</td>
<td>4000 mm</td>
</tr>
<tr>
<td>Maximum tank length</td>
<td>10000 mm</td>
</tr>
<tr>
<td>Maximum transport weight</td>
<td>500·10³ kg</td>
</tr>
</tbody>
</table>
5.3 Optimization under parameter uncertainties

The optimization in this section is done in MATLAB using simplified version of the formulas used by the *Optimizer*. This simplified version consists of 63 formulas, which collectively trying to describe dimensions, masses and losses in the transformer core. The solver `fmincon` is used with the `sqp` algorithm, and the added option of normalizing the variables during optimization. Simulations with uncertainties uses sample points from the Sobol’ sequence in dimension 97, created with `scramble sobolset`.

The choice to build a separate calculation model, one that mimics the calculations done by the *Optimizer*, is to be able to add different uncertainties to each parameter used in the calculations. Here we will not treat the given parameters as certain, but allow a little flexibility in the calculations. A given value of a parameter is given a percentage of uncertainty (with a uniform probability distribution), for example; for `Total Yoke distance` $Yoke_{\text{dist}} = 260$ (mm) $\pm 2\%$. However, more often than not, the values are given together with their standard deviation, not a percentage uncertainty. This standard deviation can be matched to coincide with the standard deviation for the chosen distribution. For example, for uniform distribution defined over the interval from $a$ to $b$, the standard deviation equals

$$\text{Standard deviation} = \frac{(b - a)}{\sqrt{12}},$$

where the , the values for $a$ and $b$ can easily be found, if the standard deviation is known. Uniform distribution is chosen for the uncertainties in the calculations through out this chapter. There are a range of distributions to chose from, uniform, triangular, normal distribution, etc. In the Appendix B there is a section which discuss how to chose a probability distribution.

The objective function, which is to be minimized, consists of $COST_{\text{Losses}}$ – the cost of various losses, $COST_{\text{Material}}$ and $COST_{\text{Labour}}$ which are material and labour costs respectively.

There are two ways to calculate the $COST_{\text{Material}}$ and $COST_{\text{Labour}}$ used in the core and the windings,

- The first is the *OLD* way, where the cost of the material and labour for copper and steel, is a simple price per kilo. This price combine both the material and labour, disregarding for example, the fact that some types of windings are more labour intense compared to others. Another example is the cutting of the core, where the thickness of the steel plate will play a part in the the time it takes to assemble the core. The *OLD* way is also used by the *Optimizer* and will be used in TEST 1, and .

- The second is pricing labour and material, using the formulas in the UCM’s Technical Standard manual, where only the relevant formulas has been extracted. These calculations are much more refined, and where for example each winding type, and core configurations are calculated specifically, not just a ‘one-off’ kilo price. The formulas from UCM will be used in TEST 2, and these are the formulas used to recalculate the price of the transformer design in the EDS.
The optimization in TEST 1 and TEST 2 is formulated as

\[
\begin{align*}
\text{minimize} & \quad \text{COST}_{\text{Losses}}(x) + \text{COST}_{\text{Material}}(x) + \text{COST}_{\text{Labour}}(x) \\
\text{subject to} & \quad T_w(x) - 4000 \leq 0 & \text{Tank width (mm)} \\
& \quad T_h(x) - 4000 \leq 0 & \text{Tank height (mm)} \\
& \quad T_L(x) - 10000 \leq 0 & \text{Tank length (mm)} \\
& \quad P_0(x) - 20 \leq 0 & \text{No load loss (kW)} \\
& \quad P_k(x) - 110 \leq 0 & \text{Total load loss (kW)} \\
& \quad G_{\text{tot}}(x) - 500 \cdot 10^3 \leq 0 & \text{Total mass (kg)} \\
& \quad X(x) = 10.5 & \text{Reactance (kg)} \\
& \quad 300 \leq D_{\text{nom}} \leq 1030 & \text{Core diameter (mm)} \\
& \quad 500 \leq H_{\text{limb}} \leq 4000 & \text{Limb height (mm)} \\
& \quad 1.2 \leq B \leq 2 & \text{Flux density (T)} \\
& \quad 1 \leq A_n \leq 10 & \text{Winding area normalized (-)} \\
\end{align*}
\]

where \( x \) is a vector consisting of \( x = (H_{\text{limb}}, D_{\text{nom}}, A_n, B) \). The upper and lower bound for the design variables is set by the customer. It is assumed that all formulas in the rest of this chapter are dependent of \( x \), unless otherwise stated, however for brevity, notation for vector \( x \) is excluded.

**Formulas used in the constraint functions**

\[
\begin{align*}
\text{No Load Loss (kW)} & \quad P_0^* = \\
\text{Tank length (mm)} & \quad T_L = \\
\text{Tank width (mm)} & \quad T_w = \\
\text{Tank height (mm)} & \quad T_h = \\
\text{Total mass (kg)} & \quad G_{\text{tot}} = \\
\text{Reactance (kg)} & \quad X^* = \\
\end{align*}
\]

( Formulas intentionally left blank, due to confidential information )

where formulas marked with an asterisk (*), are given an uncertainty, see next section.

**Uncertainty added parameters and formulas**

The task of adding uncertainty to the parameters and formulas is not trivial. Great care has been taken so the added uncertainty are realistic. The size of the uncertainty for the different parameters has been set by ABB, see Table 5.1.

All formulas marked with an asterisk (*), are given an uncertainty of the size \( p_4 \), applied to the formula as a whole. The notation and the size of the uncertainties are displayed in Table 5.1.
Table 5.1: Uncertainty notation

<table>
<thead>
<tr>
<th>Uncertainty Type</th>
<th>Range</th>
<th>Denoted</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input-dimension parameters</td>
<td>1% – 2%</td>
<td>$p_1$</td>
<td>(Used in Test 1 and 2)</td>
</tr>
<tr>
<td>Price1 and Price2</td>
<td>5% – 15%</td>
<td>$p_2$</td>
<td>(Used in Test 1)</td>
</tr>
<tr>
<td>Constants in UCM</td>
<td>1% – 2%</td>
<td>$p_3$</td>
<td>(Used in Test 2)</td>
</tr>
<tr>
<td>Formula uncertainty</td>
<td>1% – 2%</td>
<td>$p_4$</td>
<td>(Denoted with *)</td>
</tr>
<tr>
<td>Labour cost in UCM</td>
<td>10% – 15%</td>
<td>$p_5$</td>
<td>(Used in Test 2)</td>
</tr>
<tr>
<td>Material cost in UCM</td>
<td>1% – 2%</td>
<td>$p_6$</td>
<td>(Used in Test 2)</td>
</tr>
<tr>
<td>Copper and Steel price- added to COST1-3 in UCM</td>
<td>5%</td>
<td>$p_8$</td>
<td>(Used in Test 2)</td>
</tr>
</tbody>
</table>

Notation for the different uncertainties used in the objective and constraints formulas.

It is important to understand the importance and difference in the result between adding a uncertainty after an optimization, which contain no uncertainty, and allowing the uncertainty to be added before optimization, as the uncertainty will affect the optimization result.

Formulas used in the objective function

The objective function consists of the running costs $\text{COST}_{\text{Losses}}$, and the manufacturing costs $\text{COST}_{\text{Material}}$ and $\text{COST}_{\text{Labour}}$, which are material costs and labour costs (a more detailed description of the cost formulas taken from the UCM can be found in the Appendix C).

- **$\text{COST}_{\text{Losses}}$**
  
  \[
  C_{\text{lll}} = P_k \cdot P_{\text{keval}} \quad \text{(Cost of Loss with Load)}
  \]
  
  \[
  C_{\text{nil}} = P_0 \cdot P_{\text{0eval}} \quad \text{(Cost of Loss without Load)}
  \]

- **$\text{COST}_{\text{Material}}$ and $\text{COST}_{\text{Labour}}$ is** (Used in TEST 1)
  
  \[
  \text{Price}_1 \cdot G_w \cdot ML + \text{Price}_2 \cdot (G_{\text{limbs}} + G_{\text{yokes}})
  \]
  
  where the combined price for steel is $\text{Price}_1 = 6.34\ \text{USD/kg}$, and copper $\text{Price}_2 = 3.72\ \text{USD/kg}$

- **$\text{COST}_{\text{Material}}$ and $\text{COST}_{\text{Labour}}$ is** (Used in TEST 2)
  
  $\text{COST}_1+\text{COST}_2+\text{COST}_3+\text{COST}_{18}+\text{COST}_{19}+\text{COST}_{20}+\text{COST}_{23}+\text{COST}_{24}$
  
  Consisting of:
  
  - Material: Core Steel (Section 3.1 in UCM)
    
    $\text{COST}_1=
  
  - Material: Inactive core parts (Section 3.2 in UCM)
    
    $\text{COST}_2=$
- Material: Winding Conductor (Section 3.3 in UCM)  
  COST3=

- Labour: Core cutting (Section 3.18 in UCM)  
  COST18=

- Labour: Inactive core parts manufacturing (Section 3.19 in UCM)  
  COST19=

- Labour: Core manufacturing (Section 3.20 in UCM)  
  COST20=

- Labour: Winding (Section 3.23 in UCM)  
  COST23=

- Labour: Winding assembly and drying (Section 3.24 in UCM)  
  COST24=

( Formulas intentionally left blank, due to confidential information )

**Formulas used for calculating COST_{Losses}**

- Active core area (cm$^2$)  
  \[ A_{core}^* = \]

- Volt per turn (V)  
  \[ V_0^* = \]

- Turns (-)  
  \[ N^* = \]

- Winding height (mm)  
  \[ H_w^* = \]

- Winding area (mm$^2$)  
  \[ A_w^* = \]

- Winding width (mm)  
  \[ W_w^* = \]

- Winding 1 inner diameter (mm)  
  \[ D_{wi1}^* = \]

- Winding 1 outer diameter (mm)  
  \[ D_{wo1}^* = \]

- Winding 2 inner diameter (mm)  
  \[ D_{wi2}^* = \]

- Winding 2 outer diameter (mm)  
  \[ D_{wo2}^* = \]

- Winding 1 mean diameter (mm)  
  \[ D_{wm1}^* = \]

- Winding 2 mean diameter (mm)  
  \[ D_{wm2}^* = \]

- Ampere turn (A)  
  \[ M^* = \]

- Conductor length winding 1 (m)  
  \[ L_{w1}^* = \]

- Conductor length winding 2 (m)  
  \[ L_{w2}^* = \]

- Volume of conductor 1 (m$^3$)  
  \[ V_{c1}^* = \]

- Volume of conductor 2 (m$^3$)  
  \[ V_{c2}^* = \]

- Ohmic loss winding 1 (kW)  
  \[ R_{II1}^* = \]

- Ohmic loss winding 2 (kW)  
  \[ R_{II2}^* = \]

- Mass of conductor 1 (kg)  
  \[ G_{w1}^* = \]

- Mass of conductor 2 (kg)  
  \[ G_{w2}^* = \]

- Total mass of conductors (kg)  
  \[ G_{w}^* = \]

- Eddy Loss AR wind 1 (kW)  
  \[ P_{W_{edda1}}^* = \]

- Eddy Loss AR wind 2 (kW)  
  \[ P_{W_{edda2}}^* = \]
<table>
<thead>
<tr>
<th>Term</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axial eddy factor</td>
<td>$r_1$</td>
<td></td>
</tr>
<tr>
<td>Axial eddy factor</td>
<td>$r_2$</td>
<td></td>
</tr>
<tr>
<td>Axial eddy factor</td>
<td>$c_1^*$</td>
<td></td>
</tr>
<tr>
<td>Axial eddy factor</td>
<td>$c_2^*$</td>
<td></td>
</tr>
<tr>
<td>Eddy loss RR wind 1 (kW)</td>
<td>$P_{w_{eddr1}}$</td>
<td></td>
</tr>
<tr>
<td>Eddy loss RR wind 2 (kW)</td>
<td>$P_{w_{eddr2}}$</td>
<td></td>
</tr>
<tr>
<td>Flux in wind 1 (Vs)</td>
<td>$F_{i_{w1}}$</td>
<td></td>
</tr>
<tr>
<td>Flux in wind 2 (Vs)</td>
<td>$F_{i_{w2}}$</td>
<td></td>
</tr>
<tr>
<td>Flux in duct (Vs)</td>
<td>$F_{i_{d}}$</td>
<td></td>
</tr>
<tr>
<td>Max flux (Vs)</td>
<td>$F_{i_{max}}$</td>
<td></td>
</tr>
<tr>
<td>Other eddy loss (kW)</td>
<td>$P_{oed}$</td>
<td></td>
</tr>
<tr>
<td>Total Load Loss (kW)</td>
<td>$P_k$</td>
<td></td>
</tr>
<tr>
<td>Limb pitch (mm)</td>
<td>$P_{limb}$</td>
<td></td>
</tr>
<tr>
<td>Core length (mm)</td>
<td>$L_{core}$</td>
<td></td>
</tr>
<tr>
<td>Volume of limbs (m$^3$)</td>
<td>$V_{ml}$</td>
<td></td>
</tr>
<tr>
<td>Volume of yokes (m$^3$)</td>
<td>$V_{yokes}$</td>
<td></td>
</tr>
<tr>
<td>Mass of Core limbs (kg)</td>
<td>$G_{limbs}^*$</td>
<td></td>
</tr>
<tr>
<td>Mass of Yokes (kg)</td>
<td>$G_{yokes}^*$</td>
<td></td>
</tr>
<tr>
<td>Base loss in core (W/kg)</td>
<td>$P_{b}^*$</td>
<td></td>
</tr>
<tr>
<td>Mass of Corners loss region (kg)</td>
<td>$G_c$</td>
<td></td>
</tr>
<tr>
<td>Mass of yokes loss region (kg)</td>
<td>$G_y$</td>
<td></td>
</tr>
<tr>
<td>Mass of t-joint loss region (kg)</td>
<td>$G_{tj}$</td>
<td></td>
</tr>
<tr>
<td>Mass of outer limb loss region (kg)</td>
<td>$G_{ol}$</td>
<td></td>
</tr>
<tr>
<td>Mass of center limb loss region (kg)</td>
<td>$G_{cl}$</td>
<td></td>
</tr>
<tr>
<td>Yoke clamp length (mm)</td>
<td>$yc l^*$</td>
<td></td>
</tr>
<tr>
<td>Yoke clamp width (mm)</td>
<td>$yc w^*$</td>
<td></td>
</tr>
<tr>
<td>Thickness of Yoke clamp (mm)</td>
<td>$yc t^*$</td>
<td></td>
</tr>
<tr>
<td>Yoke clamp mass (kg)</td>
<td>$G_{ycmwe}^*$</td>
<td></td>
</tr>
<tr>
<td>Asecond layers</td>
<td>n1</td>
<td></td>
</tr>
<tr>
<td>Asecond layers</td>
<td>n2</td>
<td></td>
</tr>
<tr>
<td>Length of asecond (m)</td>
<td>$L_{asec}^*$</td>
<td></td>
</tr>
<tr>
<td>Mass of Asecond (kg)</td>
<td>$G_{asec}^*$</td>
<td></td>
</tr>
<tr>
<td>Total mass</td>
<td>$G_{tot}$</td>
<td></td>
</tr>
</tbody>
</table>

(Formulas intentionally left blank, due to confidential information)
5.3.1 Test 1- Optimizing using the 'Simple cost' function

In the first test we are running the MATLAB optimization script ABBcost Combined, see Appendix F for MATLAB code, using a simplified version of the cost function. Where the Material and Labour cost are combined in a price per kg., for both steel and copper, where $Price_1 = 6.34$ USD/kg for steel, and $Price_2 = 3.72$ USD/kg for copper. Starting with no uncertainty, then adding the uncertainty $p_1$ on all the parameters except $Price_1$ and $Price_2$, which are given the uncertainty $p_5$. Each simulation is performed $N = 100 000$ times, the result will show the average of the design parameters and object function, as well as standard deviation, quantiles, kurtosis and skewness.

In Test 1, the cost function to be optimized in Eq. (5.3) is formulated as

$$C_{lll} + C_{nll} + Price_1 \cdot G_w \cdot ML + Price_2 \cdot (G_{limbs} + G_{yokes}).$$

Three different levels of uncertainty tested are denoted as: 'Low' with an uncertainty of $p_1 = p_4=0.01$ and $p_2 = 0.05$, 'Medium' with an uncertainty of $p_1 = p_4=0.02$ and $p_2 = 0.1$, and 'High' with an uncertainty of $p_1 = p_4=0.02$ and $p_2 = 0.15$. The result is illustrated in Table 5.2 and Table 5.3.

Table 5.2: Expected value and mean from Test 1

<table>
<thead>
<tr>
<th>No uncertainty</th>
<th>Value</th>
<th>With Uncertainty:</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E[D_{nom}]$</td>
<td>538.6</td>
<td>$D_{nom}$</td>
<td>538.62</td>
<td>538.64</td>
<td>538.66</td>
</tr>
<tr>
<td>$E[H_{limb}]$</td>
<td>1760</td>
<td>$H_{limb}$</td>
<td>1760</td>
<td>1761</td>
<td>1762</td>
</tr>
<tr>
<td>$E[B]$</td>
<td>1.7503</td>
<td>$B$</td>
<td>1.75</td>
<td>1.75</td>
<td>1.75</td>
</tr>
<tr>
<td>$E[A_n]$</td>
<td>1.6175</td>
<td>$A_n$</td>
<td>1.6163</td>
<td>1.6173</td>
<td>1.6170</td>
</tr>
<tr>
<td>$E[Objective]$</td>
<td>382043</td>
<td>Objective</td>
<td>382004</td>
<td>381941</td>
<td>381835</td>
</tr>
</tbody>
</table>

Illustrating the results from Test 1 using the simple cost formula, together with the collection of formulas calculation losses. The first two column gives the expected value, and the remaining columns gives the values with uncertainties. 'Low' has uncertainty $p_1 = p_4=0.01$ and $p_2 = 0.05$, 'Medium' has $p_1 = p_4=0.02$ and $p_2 = 0.1$, and 'High' has $p_1 = p_4=0.02$ and $p_2 = 0.15$.

The simulations show that a small deviation in the in-parameters and the $Price_1$ and $Price_2$ has a big influence on the optimized cost formula. Taking a closer look at the case labeled 'High', where the in-parameters and constants in the formulas calculating losses has a 2% uncertainty, and $Price_1$ and $Price_2$ has a 10% uncertainty. The result shows a normalized standard deviation of 0.023. From the 0.90-quantile we learn that 10% of the simulations shows the price to be over 3.7% of the expected value of the price.

The tests shows the value of the kurtosis decreases as the uncertainties increases, and that the skewness in this case is negligible. Figure 5.3 graphically illustrates the spread of values the cost function take.
Table 5.3: Standard deviation, quantile and kurtosis for Test 1

<table>
<thead>
<tr>
<th>Uncertainty</th>
<th>SD (USD)</th>
<th>SD norm.</th>
<th>0.90-quantile</th>
<th>0.95-quantile</th>
<th>Kurtosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>4084</td>
<td>0.011</td>
<td>1.018</td>
<td>1.021</td>
<td>2.77</td>
</tr>
<tr>
<td>medium</td>
<td>6373</td>
<td>0.017</td>
<td>1.027</td>
<td>1.032</td>
<td>2.75</td>
</tr>
<tr>
<td>high</td>
<td>8746</td>
<td>0.023</td>
<td>1.037</td>
<td>1.043</td>
<td>2.58</td>
</tr>
</tbody>
</table>

Illustrating the standard deviation, both in USD and normalized, quantiles, and kurtosis from Test 1 using the simple cost formula. Kurtosis for a standard normal distribution is 3.

![Histogram of objective function in Test 1](image)

Figure 5.3: Histogram of objective function in Test 1.

The histogram to the right is of the result of simulated value of the objective function for Test 1, to the right is the same value normalized. This test used the cost function with the simplified version of material and labour costs. In the histogram a low kurtosis is clearly visible (kurtosis=2.58 versus 3 for standard normal distribution).

### 5.3.2 Test 2- Optimizing using the UCM cost formula

In the second test we are running the MATLAB optimization script using a cost function taken from the UCM, where separate Material and Labour cost formulas are used. In Test 2, the cost function to be optimized in Eq. (5.3) is formulated as

\[ C_{III} + C_{nll} + COST1 + COST2 + COST3 + COST18 + COST19 + COST20 + COST23 + COST24. \]

First the optimization is performed with no uncertainty, using expected values. Then uncertainties \( p_1, p_2, p_4, p_6, \) and \( p_8 \) are added, see Table 5.1 for explanation.

Three different levels of uncertainty tested are:

- 'Low': \( p_1 = p_3 = p_4 = p_6 = 0.01, p_2 = 0.05 \) and \( p_8 = 0.05 \),
- 'Medium': \( p_1 = p_3 = p_4 = p_6 = 0.02, p_2 = 0.10 \) and \( p_8 = 0.05 \),
- 'High': \( p_1 = p_3 = p_4 = p_6 = 0.02, p_2 = 0.15 \) and \( p_8 = 0.05 \).

Each simulation is performed \( N = 100000 \) times, the result will show the average of the design parameters and object function, as well as standard deviation, quantile, kurtosis and skewness.
Table 5.4: Expected value and mean from Test 2

<table>
<thead>
<tr>
<th>No uncertainty</th>
<th>Value</th>
<th>With Uncertainty: Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E[D_{nom}]$</td>
<td>551.3</td>
<td>531.37</td>
<td>498.80</td>
<td>498.80</td>
</tr>
<tr>
<td>$E[H_{limb}]$</td>
<td>1581</td>
<td>1676</td>
<td>1864</td>
<td>1864</td>
</tr>
<tr>
<td>$E[B]$</td>
<td>1.7525</td>
<td>1.7821</td>
<td>1.8286</td>
<td>1.8286</td>
</tr>
<tr>
<td>$E[A_n]$</td>
<td>1.8315</td>
<td>1.8509</td>
<td>1.8831</td>
<td>1.8831</td>
</tr>
<tr>
<td>$E[Objective]$</td>
<td>436471</td>
<td>437372</td>
<td>440993</td>
<td>440994</td>
</tr>
</tbody>
</table>

Illustrating the results from Test 2 using the simple cost formula. The first two column gives the expected value, and the remaining columns gives the values with uncertainties. Low has uncertainty $p_1 = p_4 = 0.01$ and $p_2 = 0.05$, Medium has $p_1 = p_4 = 0.02$ and $p_2 = 0.1$, and High has $p_1 = p_4 = 0.02$ and $p_2 = 0.15$.

In the case, labeled 'High' in Table 5.4, the test show a normalized standard deviation of 0.013, and from the 0.90-quantile we learn that 10% of the simulations showed the price to be over 3.0% of the average price. As in Test 1, the value of the kurtosis decreases as the uncertainties increases, and that the skewness in this case is negligible.

Table 5.5: Standard deviation, quantile and kurtosis for Test 2

<table>
<thead>
<tr>
<th>Uncertainty</th>
<th>SD (USD)</th>
<th>SD norm.</th>
<th>0.90-quantile</th>
<th>0.95-quantile</th>
<th>Kurtosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>4415</td>
<td>0.010</td>
<td>1.019</td>
<td>1.022</td>
<td>2.79</td>
</tr>
<tr>
<td>medium</td>
<td>4987</td>
<td>0.011</td>
<td>1.029</td>
<td>1.033</td>
<td>2.87</td>
</tr>
<tr>
<td>high</td>
<td>5288</td>
<td>0.0130</td>
<td>1.0303</td>
<td>1.0341</td>
<td>2.87</td>
</tr>
</tbody>
</table>

Illustrating the standard deviation, both in USD and normalized, quantiles, and kurtosis from Test 2, using the UCM formulas in the cost formula.

Figure 5.4: Histogram of objective function in Test 2.
The histogram to the right is of the result of simulated value of the objective function for Test 2, to the right is the same value normalized. The cost function used the UCM version of material and labour costs.
5.4 Summary

The simulations show that a collection of small deviations in the parameters has a big effect on the monetary evaluation of the cost formula in the process of designing transformers. Simple test has shown that it is not possible to single out a single uncertainty in a parameter and claim it has a bigger impact on the objective function than the others, but they are important and all affect the result. A more thorough sensitivity analysis is needed to get accurate result, and this did not fit the time scale of this report, but it is suggested in the further research section.

In Test 1 we showed that 10% of the simulations showed the price to be over 3.7% of the average price, and for Test 2 it was 3.0%.

For example, if the accepted profit margin for a product, in this case transformers, is 10%, then the tests in this report shown that one in ten transformers will have a profit margin of 7% or less should uncertainties be taken into account. This would lead to a reevaluation of the decision to accept or reject an order.
Chapter 6

Conclusion

This thesis has shown the importance of including uncertainties for cost optimizations in an industrial setting. It has built a framework to incorporate uncertainty into very large optimization problems for companies such as ABB. For this framework, we first started with how to best evaluate functions containing uncertainties, where Monte Carlo simulations were the chosen methods.

In Chapter 2 we illustrated why quasi Monte Carlo method more efficient than Monte Carlo method, except in very high dimensions, since it uses of the evenly distributed quasi-random sequences. Quasi-random sequences, also called low-discrepancy sequences, tested in this report are Halton- and Sobol’ sequences. For dimensions two and above, Halton sequences gives very good results, though it exists a correlation problem for higher dimensions. We have shown that from dimension ten there is a noticeable correlation problem causing the points to line up, due to the use of higher prime numbers as bases. The Sobol’ sequence is shown to work well in higher dimensions, and when used in MATLAB, we have shown that scrambled sobolset with the added option skip should be used.

In Chapter 3, we used Monte Carlo and quasi Monte Carlo methods to evaluate functions containing multiple variables, and compared different Monte Carlo and quasi Monte Carlo methods. For the Monte Carlo method the MATLAB command rand was used, and for the quasi Monte Carlo methods haltonset and sobolset was used with the optional functions of skipping, leaping or scramble.

In the first test we compared the mean and standard deviation when evaluation a made-up function \( g_k(p) \) using MATLAB generated sample set; rand, and sobolset, and it showed (with small differences) that scrambled Sobol’ rendered values closer to the mean than unscrambled Sobol’, furthermore both performed better then rand. The second test we compared the number of iterations needed to achieve a result where the simulated average value is within a set distance from the mean. Again, the quasi Monte Carlo using scrambled sobolset outperformed rand. Simulation showed, if the aim is to be within 1% of the mean of the test function \( g_{90}(p) \) in at least 90% of the simulations, we showed that quasi Monte Carlo simulations needed less than 200 iterations versus over 10000 iterations for Monte Carlo simulations.

In Chapter 4, we incorporated uncertainties into optimization and showed through a numerical example the affect on choosing the right algorithm in fmincon when optimizing with uncertainties. We optimized a Cantilever beam design, using three different algorithms, active set,
interior point and sqp, in the nonlinear bounded constraint problems optimizer fmincon. Problems with big variations in variables when using sqp and interior point, were solved using the added option of normalization of the variables during the optimization, this is not an option for the active set algorithm. Simulations showed sqp and active set to be both faster and more accurate than interior-point.

Most interesting, is the finding that MATLAB’s default solver interior-point, render an answer less accurate then active set and sqp, and this has been confirmed in MATLAB user manual [21] that at times this is the case. For large optimization problems, a solution which is, only accurate up till the third significant digit, might be deemed as not very impressive.

In Chapter 5 where experiments with cost minimization on for the design of bespoke High Voltage transformers, produced by ABB worldwide, was conducted. Two different cost functions were minimized, taken into account; the first function used a simplified version of a combined material and labour cost, the second used a more specific formula for material and labour taken from an internal ABB formula collection. The simulations show that a small deviation in the parameters has a big effect on the monetary evaluation of the cost formula. In Test 1 we showed that 10% of the simulations showed the price to be over 3.7% of the expected price, and for Test 2 it was 3.0%. If the accepted profit margin for a product, in this case transformers, is 10%, then the tests in this report shown that one in ten transformer will have a profit margin of 7% or less if uncertainties are taken into account. This would lead to an evaluation of the decision to accept or reject an order.

This thesis has just scratched the surface of the problem and importance of taking uncertainties into account when optimizing a transformer design, and there are many areas and angles to continue to work on this very complex problem. It has though, built a framework, of how uncertainties can be taken into account in complex optimization problems, such as transformer designs.

### 6.1 Further research

- Simulate using different probability distributions, such as triangle. More research into how the values of the parameters has been determined; are they accurately measured? Then choose an accurate probability distribution to use in further simulations.

- Optimize using different starting points, using GlobalSearch and MultiStart, to be certain a global minima has been found.

- Simulate using all parts of the UCM, i.e., more components other than the core and windings.

- A sensitivity analysis could show how a small changes in each of the parameters would affect the cost minimization.

Another areas that this report showed could benefit from further investigations, like the specific dimensions for using different low-discrepancy sequences in an industrial setting.
Chapter 7

Fulfilment of Thesis

In this section the author, Ann-Sofi Kidwell, provides a brief declaration of the fulfilment of the objectives set as requirements of the Swedish National Agency for Higher Education to Master theses.

**Objective 1:** For Master degree, student should demonstrate knowledge and understanding in the major field of study, including both broad knowledge in the field and substantially deeper knowledge of certain parts of the area as well as insight into current research and development.

Fulfilment: The author has used theory from many areas of mathematics, such as probability, statistics, calculus, stochastic processes, and has shown a deeper knowledge in the field of optimization, field theory and, Monte Carlo techniques. The author has reviewed and discussed current research papers in the mentioned fields and in areas such as variance reduction in Monte Carlo simulations, pro and cons for various low-discrepancy sequences. Where a survey and comparison of available Monte Carlo methods are included.

**Objective 2:** For Master Degree, student should demonstrate deeper methodological knowledge in the major field of study.

Fulfilment: In the report the author show how an industrial optimization problem is formulated as a mathematical problem. Using tools from statistics and optimization theory, it is present a logical framework within which the problem is solved. Deeper presentation of specific methods used in the project are included in Chapter 3, 4 and 5. Mathematical proofs are included and explained in the chapters and in appendices.

**Objective 3:** For Master degree, student should demonstrate the ability to critically and systematically integrate knowledge and to analyse, assess and deal with complex phenomena, issues and situations even with limited information.

Fulfilment: The author has shown the ability to conduct research into an area new to many industry settings, where the results could not be predicted, and where over sixty formulas, containing uncertainty, had to interact in a correct way for the mathematical optimization to work. The results came about after a thorough groundwork into the subject and research into current methods, to fi-
nally in Chapter 5, bring all of the different aspect together, to conduct a complex ABB cost formula optimization. To finalize this thesis, the author has demonstrated the ability to bring together results from different chapters.

**Objective 4:** For Master degree, student should demonstrate the ability to critically, independently and creatively identify and formulate issues and to plan and carry out advanced tasks within specified time frames, thereby contributing to the development of knowledge and to evaluate this work.

Fulfilment: The author has met the deadlines throughout the research and the process of writing this thesis, and shown a significant ability to critically, independently and creatively identify and formulate question and analyse and solve problems, in the time frame given.

**Objective 5:** For Master degree, student should demonstrate ability in both national and international contexts, orally and in writing to present and discuss their conclusions and the knowledge and arguments behind them, in dialogue with different groups.

Fulfilment: This will be fulfilled during the presentation on 25th of January 2018 at MDH, and with separate presentations at ABB, to different groups that might benefit from the research area and the findings. The author has demonstrated a ability to discuss the findings and conclusion during the course of this thesis, both in written correspondence and verbally with supervisors. The language and concepts used in the thesis are universal to any reader with a background in statistics, optimization and general field of mathematics, and since the intended readers at ABB are engineers and scientists, some parts has been given a more in-dept explanations.

**Objective 6:** For Master degree, student should demonstrate ability in the major field of study make judgments taking into account relevant scientific, social and ethical aspects, and demonstrate an awareness of ethical issues in research and development.

Fulfilment: The author carefully make reference to the ideas, research and results from different authors that have been used as input in this thesis. Also, the author has been careful and aware of current practice, when writing Chapter 5.
Bibliography


Appendices
Appendix A

Galois field

Galois field, also called finite fields, as the title suggest, is a field containing a finite number of elements, and we require two operations; addition and multiplication, defined on the elements of this finite set to satisfy the field axioms. The field axioms are; associativity, commutativity, distributivity, identity, and the existence of an inverse, see Table A.1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Addition</th>
<th>Multiplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>associativity</td>
<td>(a+b)+c=a+(b+c)</td>
<td>(ab)c=a(bc)</td>
</tr>
<tr>
<td>commutativity</td>
<td>a+b=b+a</td>
<td>ab=ba</td>
</tr>
<tr>
<td>distributivity</td>
<td>a(b+c)=ab+ac</td>
<td>(a+b)c=ac+bc</td>
</tr>
<tr>
<td>identity</td>
<td>a + 0 = a</td>
<td>a ⋅ 1 = a</td>
</tr>
<tr>
<td>inverse exist</td>
<td>a + (−a)=0</td>
<td>a ⋅ a−1 = 1</td>
</tr>
</tbody>
</table>

Field axioms; associativity, commutativity, distributivity, identity, and the existence of an inverse, and the two require two operations; addition and multiplication.

In Table A.2, showing the Addition table, we can easily find the identity by either adding zero and the inverse by adding the negative corresponding value, and in Table A.3, the multiplication table, the identity is found in the table and the inverse are the boxes containing ones.

The order of a finite field is always a prime or a power of a prime. Example of a Galios field $GF(8) = GF(2^3)$ is a finite fields of order 2³, also called binary field, whose elements are binary polynomials, i.e., polynomials whose coefficients are either 0 or 1, and the degree of the polynomials is no more than $n − 1$. The Galios field, $GF(2^3)$, contains 8 elements \{0, 1, x, x^2, x+1, x^2+1, x^2+x, x^2+x+1\}.

Polynomial addition and multiplication are done using operations modulo 2, while satisfying the field axioms. If after polynomial multiplication, the terms are of order $x^n$ or higher, the result need to be reduced modulo a suitable primitive polynomial of order $n$. Divide the result from the multiplication with the primitive polynomial using long division and keep the reminder, which will be of the order $x^{n−1}$ or less. Primitive polynomials are the minimal polynomials for primitive elements in a Galois field.
A primitive polynomial must have a non-zero constant term, for otherwise it will be divisible by \( x \). In \( GF(2^n) \), when the degree of the result is more than \( n - 1 \), it needs to be reduced modulo a irreducible polynomial. This is done by dividing the polynomial by the irreducible polynomial and the keeping the rest. When generating Sobol’ sequences, different primitive polynomial will render different sequences. Examples of primitive polynomials in \( GF(2^n) \) are:

<table>
<thead>
<tr>
<th>Degree</th>
<th>Primitive polynomials</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>( x^2 + x + 1 )</td>
</tr>
<tr>
<td>3</td>
<td>( x^3 + x + 1 )</td>
</tr>
<tr>
<td></td>
<td>( x^3 + x^2 + 1 )</td>
</tr>
<tr>
<td>4</td>
<td>( x^4 + x + 1 )</td>
</tr>
<tr>
<td></td>
<td>( x^4 + x^3 + 1 )</td>
</tr>
</tbody>
</table>

Example of the addition and multiplication tables for \( GF(2^3) \) using the primitive polynomial \( x^3 + x + 1 \).

**Table A.2: Addition table for \( GF(2^3) \)**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>x+1</th>
<th>x^2</th>
<th>x^2+1</th>
<th>x^2+x</th>
<th>x^2+x+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x+1</td>
<td>x^2</td>
<td>x^2+1</td>
<td>x^2+x</td>
<td>x^2+x+1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x+1</td>
<td>x</td>
<td>x^2+1</td>
<td>x^2</td>
<td>x^2+x</td>
<td>x+1</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x+1</td>
<td>0</td>
<td>1</td>
<td>x^2+x</td>
<td>x^2+1</td>
<td>x^2</td>
<td>x^2+1</td>
</tr>
<tr>
<td>x+1</td>
<td>x+1</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>x^2+x+1</td>
<td>x^2+x</td>
<td>x^2+1</td>
<td>x^2</td>
</tr>
<tr>
<td>x^2+1</td>
<td>x^2+1</td>
<td>x^2+1</td>
<td>x^2+1</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x+1</td>
<td>x</td>
</tr>
<tr>
<td>x^2</td>
<td>x^2</td>
<td>x^2+x+1</td>
<td>x^2+x</td>
<td>1</td>
<td>0</td>
<td>x+1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Multiplication table for \( GF(2^3) \), which contains the elements \( \{0, 1, x, x^2, x+1, x^2+1, x^2+x, x^2+x+1\} \).

**Table A.3: Multiplication table for \( GF(2^3) \)**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>x+1</th>
<th>x^2</th>
<th>x^2+1</th>
<th>x^2+x</th>
<th>x^2+x+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x+1</td>
<td>x^2</td>
<td>x^2+1</td>
<td>x^2+x</td>
<td>x^2+x+1</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>x</td>
<td>x^2</td>
<td>x^2+x</td>
<td>x+1</td>
<td>1</td>
<td>x^2+x+1</td>
<td>x^2+x</td>
</tr>
<tr>
<td>x+1</td>
<td>0</td>
<td>x+1</td>
<td>x^2+x</td>
<td>x^2+1</td>
<td>x^2+x+1</td>
<td>x^2</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>x^2</td>
<td>0</td>
<td>x^2</td>
<td>x+1</td>
<td>x^2+x+1</td>
<td>x^2+x</td>
<td>x</td>
<td>x^2+1</td>
<td>1</td>
</tr>
<tr>
<td>x^2+1</td>
<td>0</td>
<td>x^2+1</td>
<td>1</td>
<td>x^2</td>
<td>x</td>
<td>x^2+x+1</td>
<td>x+1</td>
<td>x^2+x</td>
</tr>
<tr>
<td>x^2+x</td>
<td>0</td>
<td>x^2+x</td>
<td>x^2+x+1</td>
<td>1</td>
<td>x^2+1</td>
<td>x+1</td>
<td>x</td>
<td>x^2</td>
</tr>
<tr>
<td>x^2+x+1</td>
<td>0</td>
<td>x^2+x+1</td>
<td>x^2+1</td>
<td>x</td>
<td>1</td>
<td>x^2+x</td>
<td>x+1</td>
<td>x^2+x+1</td>
</tr>
</tbody>
</table>

Multiplication table for \( GF(2^3) \), which contains the elements \( \{0, 1, x, x^2, x+1, x^2+1, x^2+x, x^2+x+1\} \).
Appendix B

Working with Normal Probability Distribution

Some problems can arise when sampling from normal probability distribution. It could be that when samples are taken from the tail of a normal distribution it might render preposterous results. Imagine doing a calculation involving time, where the cost of producing a product depends on how long it takes to perform a certain task; if the average (mean) time is 120 seconds and the standard deviation is 30 seconds, there will be samples drawn from the associated probability populations which will end up negative, see Fig. B.1.

Figure B.1: Probability distribution for $\mathcal{N}(120, 60)$.
This is the probability distribution for $\mathcal{N}(120, 60)$. The x-axis displays the probability distribution of how many second it takes to perform a task, which on average takes 120 seconds, with a standard deviation of 60 seconds. Note that there is a chance of the task taking negative time values—which is clearly not possible.

To solve the problem of unrealistic values being chosen, one can simply put a lower and upper bound on the random values. This can be done in a number of ways, one obviously method is to choose uniform distribution in between some interval. Other ways could be to use triangular distribution, where the lower and upper bound could be set and also the size and location of the tip of the triangle, or to use truncated normal distribution, see Fig. B.4. In the case of truncated
normal distribution, the area from the cut off tail bits are moved to the inside of the lower and upper bound. This might not always be the preferred way, one might want to let the samples chosen from below the lower limit keep the value of the lower limit, in that way it is still counted as a end point sample, instead of being moved towards the middle as in the truncated case, see Fig. B.2 and Fig. B.3, for an example of this being applied in MATLAB.

Figure B.2: Truncated normal distribution between -1.5 and 1.5
The truncated normal distribution is the probability distribution of a normally distributed random variable whose value is bounded below by -1.5 and above by 1.5.

Figure B.3: Truncated normal distribution between -0.5 and 1.5
Here the truncated probability distribution is bounded by -0.5 and 1.5, it shows a more distorted curve compared to the normal distribution curve then the previous Fig.B.2.

Figure B.4: Comparing the distribution of four different probability distributions.
The purpose of this figure is to compare various probability distribution: Normal distribution, Truncated normal distribution, Triangular distribution, and Uniform distribution between -1 and 1.
B.1 Modified truncated normal probability distribution

When using a truncated distribution, the points lying outside the 'cut off' areas, i.e., the tails, are redistributed inside see Fig. B.2 and Fig. B.3, if this is not desired, then a mixed distribution method could be used.

A mixed distribution is a distributions with a discrete as well as a continuous components. In [34], for a mixed distribution function $F(y)$, it is defined as

$$F(y) = c_1 F_1(y) + c_2 F_2(y),$$

where $F_1(y)$ is a step distribution function, $F_2(y)$ a continuous function, $c_1$ is the accumulated probability off all discrete points, and $c_2$ is the accumulated probability of the continuous part.

Figure B.5 illustrates a histogram, where the values in the truncated area (discrete distribution) has been given the value of the lower and upper boundary respectively.

![Modified truncated normal distribution](image)

Figure B.5: Modified truncated normal distribution.

Truncated normal distribution, where the values in the truncated area has been given the value of the lower and upper boundary respectively.

B.2 Kurtosis

To be able to study and interpret the distribution curve, different measures are in place. For a normally distributed curve, the curve has a bell shape form, though it does not mean it is a standard normal distribution. Kurtosis measures if the data are heavy- or light tailed compared to a standard normal distribution, and it is calculated using the fourth standardized moment, i.e., the expected value of the standardized data raised to the power of four.

Any values within one standard deviation of the mean, has a standardized value less than 1. And such small value raised to the power four will contribute very little to the kurtosis. By contrast, values outside one standard deviation, the outliers, will have a large contribution to the kurtosis.
For univariate data, data consisting on one variate, $X_i$ the kurtosis is estimated as

$$Kurtosis = \frac{1}{N} \sum_{i=1}^{N} \frac{(X_i - \bar{X})^4}{s^4}$$

where $\bar{X}$ is the mean, $s$ is the standard deviation and $N$ is the number of data points. The kurtosis for standard normal distribution is three, a lower number means that there are many outliers and the peak is lower, where a kurtosis above three has a higher peak, since it contains less outliers, see Fig.B.6.

Figure B.6: Kurtosis and normal distribution
Kurtosis for standard normal distribution is 3, a lower number means a high number of outliers, and when the kurtosis is above three it has few outliers
Appendix C

Kuhn-Tucker Conditions- Proof

This proof of the Kuhn-Tucker Theorem has been adapted from [20].

For the basic optimization problem, see Eq. (C.1), where the task is to chose the value of a number of variables, such that the function \( f(x) \) is minimized subjected \( m \) constraints.

The following equation is a generalization of the classical optimization problem, since the equality constraints are a special case of inequality constraints.

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0 \quad i = 1, \ldots, m.
\end{align*}
\]

By introducing \( m \) new variables, called slack variables \( y_i, \quad (i = 1, 2, \ldots, m) \) the generalized optimization problem Eq. (C.1) can be rewritten as

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) + y_i^2 = 0 \quad i = 1, \ldots, m.
\end{align*}
\]

**Theorem 2.** For the differentiable function \( f_k(x), \quad k = 0, 1, \ldots, m \). If the function \( f_0(x) \) attains a local minimum at a point \( x^0 \), subject to a set \( k = \{ x \mid f_i(x) \leq 0, \quad i = 1, \ldots, m \} \), then there exists a vector of Lagrange multipliers \( \lambda^0 \) such that the following conditions are satisfied

\[
\begin{align*}
\frac{\partial f_0(x^0)}{\partial x_j} + \sum_{i=1}^{m} \lambda_i^0 \frac{\partial f_i(x^0)}{\partial x_j} & = 0, \quad j = 1, \ldots, n, \\
f_i(x^0) & \leq 0, \quad i = 1, \ldots, m, \\
\lambda_i^0 f_i(x^0) & = 0, \quad i = 1, \ldots, m, \\
\lambda_i^0 & \geq 0, \quad i = 1, \ldots, m.
\end{align*}
\]

This means that Eq. (C.3)–(C.6) are necessary conditions for a local minimum to the problem Eq. (C.1). These conditions are called the Kuhn-Tucker conditions.

**Proof:** The Lagrange function can be defined as a function of the original variables, \( x \) and \( y \), and the Lagrange multipliers \( \lambda \)

\[
L(x, y, \lambda) = f_0(x) + \sum_{i=1}^{m} \lambda_i (f_i(x) + y_i^2).
\]
For a local minimum, the necessary conditions are

\[
\frac{\partial L}{\partial x_j} = \frac{\partial f_0(x^0)}{\partial x_j} + \sum_{i=1}^{m} \lambda_i^0 \frac{\partial (f_i(x^0) + (y_i^0)^2)}{\partial x_j} \quad j = 1, \ldots, n, \tag{C.8}
\]

\[
\frac{\partial L}{\partial y_i} = 2 \lambda_i^0 y_i^0 = 0 \quad i = 1, \ldots, m, \tag{C.9}
\]

\[
\frac{\partial L}{\partial \lambda_i} = f_i(x^0) + (y_i^0)^2 = 0 \quad i = 1, \ldots, m. \tag{C.10}
\]

We can show that the condition in Eq. (C.9) corresponds to the Kuhn-Tucker conditions in Eq. (C.5).

If \( \lambda_i^0 = 0 \), then \( \lambda_i^0 y_i^0 = \lambda_i^0 f_i(x^0) = 0 \), satisfying the conditions in both Eq. (C.9) and Eq. (C.5).

For \( \lambda_i^0 \neq 0 \), it follows from Eq. (C.9) and Eq. (C.10) that \( y_i^0 = 0 \) and therefore \( (y_i^0)^2 = -f_i(x^0) = 0 \), thereby, satisfying the conditions in Eq. (C.5). It follows from Eq. (C.5) that if \( f_i(x^0) = 0 \), and thus, \( y_i^0 = 0 \), the conditions in Eq. (C.9) and Eq. (C.10) are satisfied.

Since the added variables \( y_i \) are auxiliary variables, they can be eliminated from the conditions in Eq. (C.8) and Eq. (C.10), and we obtain conditions Eq. (C.3) and Eq. (C.4).

Finally, it is shown that the Lagrange multipliers must be non-negative. Start with the classical optimization problem

\[
\text{minimize} \quad f_0(x) \\
\text{subject to} \quad f_i(x) \leq b_i \quad i = 1, \ldots, m. \tag{C.11}
\]

For the Lagrange multipliers \( \lambda_i^0, i = 1, \ldots, m \), from Eq. (C.11), the following holds

\[
\frac{\partial f_0(x_0(b))}{\partial b_i} = -\lambda_i^0 \quad i = 1, \ldots, m, \tag{C.12}
\]

where \( x_0 \) denotes the optimal solution for Eq. (C.11). Therefore, the Lagrange multipliers \( \lambda_i^0, i = 1, \ldots, m \), is the value of the objective function due to a small change of the constraint \( b_i \).

A value higher than zero will indicate an increase, thus the new optimal value of the objective function \( f_0(x) \) cannot, for a minimization problem, be worse than

\[
\frac{\partial f_0(x^0)}{\partial b_i} \leq 0. \tag{C.13}
\]

It follows from Eq. (C.12) and Eq. (C.13) that the condition for the Lagrange multipliers, Eq. (C.6), are non-negative.
Appendix D

Test results

This sections contains some of the larger tables from Chapter 3 (D.1) and Chapter 4 (D.2).

D.1 Numerical experiments on functions containing random variables.

Table D.1: Accuracy of simulation value vs. number of iterations

<table>
<thead>
<tr>
<th>Mean ±p%</th>
<th>Iterations</th>
<th>Sc. Sobol' inside in %</th>
<th>U(0,1) inside in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>50</td>
<td>17.4</td>
<td>5.6</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>30.0</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>74.8</td>
<td>12.1</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>98.1</td>
<td>19.7</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>100</td>
<td>28.0</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>100</td>
<td>72.1</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>35.8</td>
<td>8.3</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>60.4</td>
<td>16.7</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>94.0</td>
<td>24.2</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>100</td>
<td>39.9</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>100</td>
<td>53.0</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>100</td>
<td>71.9</td>
</tr>
<tr>
<td>2.5</td>
<td>50</td>
<td>72.8</td>
<td>31.1</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>96.1</td>
<td>42.9</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>100</td>
<td>57.1</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>100</td>
<td>78.5</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>100</td>
<td>90.2</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Testing how close the simulated mean for the function \( g_{90} \) is compared to the calculated mean. The table show the percentage of times the simulations are inside 1% of the calculated mean, i.e., within 155.4334 plus or minus 0.5%, 1%, or 2.5%.
D.2 Cantilever example- Optimization with uncertainties

Test runs shows that the three algorithms render slightly different results, active set and sqp have similar results, and interior point finds a minima different to the two algorithms. In [21], they explain why the solution can be slightly less accurate using interior-point, compared to those from active set and sqp algorithms, and that the reason for this potential inaccuracy is that the barrier function keeps iterates away from inequality constraint boundaries.

Table D.2: Result from optimizing expected value using three different algorithms in fmincon

<table>
<thead>
<tr>
<th>$d_0 = (b_0, h_0)$</th>
<th>b</th>
<th>h</th>
<th>A</th>
<th>Nr. of iter.</th>
<th>time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>active set</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.01,0.01)</td>
<td>0.022903</td>
<td>0.045789</td>
<td>0.0010483</td>
<td>13</td>
<td>0.026</td>
</tr>
<tr>
<td>(0.02,0.02)</td>
<td>0.022903</td>
<td>0.045789</td>
<td>0.0010483</td>
<td>9</td>
<td>0.023</td>
</tr>
<tr>
<td>(0.03,0.03)</td>
<td>0.022903</td>
<td>0.045789</td>
<td>0.0010483</td>
<td>9</td>
<td>0.025</td>
</tr>
<tr>
<td>(0.04,0.04)</td>
<td>0.022903</td>
<td>0.045789</td>
<td>0.0010483</td>
<td>10</td>
<td>0.034</td>
</tr>
<tr>
<td>(0.09,0.19)</td>
<td>0.022903</td>
<td>0.045789</td>
<td>0.0010483</td>
<td>13</td>
<td>0.033</td>
</tr>
<tr>
<td><strong>interior point</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.01,0.01)</td>
<td>0.022930</td>
<td>0.045731</td>
<td>0.0010486</td>
<td>23</td>
<td>0.070</td>
</tr>
<tr>
<td>(0.02,0.02)</td>
<td>0.022930</td>
<td>0.045739</td>
<td>0.0010486</td>
<td>30</td>
<td>0.042</td>
</tr>
<tr>
<td>(0.03,0.03)</td>
<td>0.022930</td>
<td>0.045731</td>
<td>0.0010485</td>
<td>36</td>
<td>0.054</td>
</tr>
<tr>
<td>(0.04,0.04)</td>
<td>0.022930</td>
<td>0.045739</td>
<td>0.0010487</td>
<td>40</td>
<td>0.066</td>
</tr>
<tr>
<td>(0.09,0.19)</td>
<td>0.022930</td>
<td>0.045731</td>
<td>0.0010486</td>
<td>18</td>
<td>0.075</td>
</tr>
<tr>
<td><strong>sqp</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.01,0.01)</td>
<td>0.022903</td>
<td>0.045789</td>
<td>0.0010483</td>
<td>11</td>
<td>0.021</td>
</tr>
<tr>
<td>(0.02,0.02)</td>
<td>0.022903</td>
<td>0.045789</td>
<td>0.0010483</td>
<td>7</td>
<td>0.014</td>
</tr>
<tr>
<td>(0.03,0.03)</td>
<td>0.022903</td>
<td>0.045789</td>
<td>0.0010483</td>
<td>7</td>
<td>0.017</td>
</tr>
<tr>
<td>(0.04,0.04)</td>
<td>0.022903</td>
<td>0.045789</td>
<td>0.0010483</td>
<td>7</td>
<td>0.021</td>
</tr>
<tr>
<td>(0.09,0.19)</td>
<td>0.022903</td>
<td>0.045789</td>
<td>0.0010483</td>
<td>10</td>
<td>0.022</td>
</tr>
</tbody>
</table>

Test runs showing that the three algorithms render slightly different results, active set and sqp have similar results, while interior point finds a minima different to the two algorithms.
Appendix E

Uniform Cost Model pricing formulas explained

Here the formulas are explained for pricing transformer designs in ABB’s Electrical Design System (EDS), using the formulas from the Uniform Cost Model (UCM), Technical Standard manual.

The relevant formulas from each section in the UCM had been sieved out, by excluding all IF-ELSE loops not relevant to our design, i.e., three main limbs, no side limbs, 100% magnetic material in core clamps and core clamp detail material, core type-'T', core steel grade-'27', and 'WindingConductorType'=S. In the MATLAB code, the formulations are reduced further by excluding any values set to zero or 1.

E.1 Notation used for the variables and constants in the UCM formulas

( Intentionally left blank, due to confidential information )

E.1.1 Pricing the design using the formulas taken from the UCM.

The relevant formulas from each section in the UCM had been sieved out, by excluding all IF-ELSE loops not relevant to our design, i.e., three main limbs, no side limbs, 100% magnetic material in core clamps and core clamp detail material, core type-'T', core steel grade-'27', and 'WindingConductorType'=S. In the MATLAB formulations the formulations are reduced further by excluding any values set to zero or 1.

( Intentionally left blank, due to confidential information )

E.2 MATLAB code used in optimization of ABB cost formula

This MATLAB code is used in the cost minimization problem in Chapter 5. It used the optimization solver for nonlinear constrained problems, fmincon, and it consists of four parts,
three functions and one script. Each of the three functions contains: the constraints functions, the objective functions, initiate constraints, and finally the a script giving a starting point and number of simulations, and calling for the three functions.

E.2.1 Script

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Creator: Ann-Sofi Kidwell %
%% Date: 19 December 2017 %
%% Created as a part of a Master Thesis for the program %
%% Engineering Mathematics at Malardalens Hogskola %
%% in cooperation with ABB Corporate Research, Vasteras. %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Script optimizing a ABB cost function of transformer core design, with the choice of design parameters UCM cost for %material and labour. There are two options of objective function; %Simple version using kilo price for material incl. labour, %or the material and labour cost calculated with UCM formulas.

n=10;
varnumber=98; % number of variables (cost parameters)
Dnom=zeros(n,1);
Hlimb=zeros(n,1);
B=zeros(n,1);
An=zeros(n,1);
d0=[527,1528,1.5,1.5];
%An=1.6871,B=1.4372,Dnom=679.0237,Hlimb=1319.3,obj=45025
lb=[300,500,1.2,1];
%lower bounds for design variables; % [Dnom,Hlimb,B,An]
ub=[1030,4000,2,10];
%upper bounds for design variables;
obj=zeros(n,1);
option=optimset('display','off','Algorithm','sqp','...'
'ScaleProblem','obj-and-constr');
x1=zeros(n,varnumber);
for i =1:n
[Inom,Unom,f,P0eval,Pkeval,Spfw,Yokedist,Ducts,Disttnk,RR,AR,ML,...
Distph,Dcorcov,Price1,Price2,p4,p8,y1,y2,y3,y4,y5,y6,...
y7,y8,y9,y10,y11,y12,y13,y14,y15,y16,...
y17,y18,y19,y20,y21,y22,y23,y24,...
y25,y26,y27,y28,y29,y30,y31,y32,y33,y34,y35,y36,y37,...
y38,y39,y40,y41,y42,y43,y44,y45,y46,y47,...
aa,an,as,at,ca,da1,dc1,ea2,eb2,ec2,ea4,eb4,ea5,eb5,fa,...
fb,fc,fd,fe,ff,lb,lf,lq,lx,mb,mf,ml,mn,mq,mu,mv,...

67
raknare,x1) = ABBcostCombined_init_const(i,x1,n,varnumber);

d = fmincon(@(d) ABBcostCombined_obj_fun(d, Inom, Unom, f, P0eval,...
Pkeval, Spfw, Yokedist, Ducts, Disttnk, RR, AR, ML, Distph, Dcorcov,...
Price1, Price2, p4, p8, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10,...
y11, y12, y13, y14, y15, y16, y17, y18, y19, y20, y21, y22, y23, y24,...
y25, y26, y27, y28, y29, y30, y31, y32, y33, y34, y35, y36, y37,...
38, 39, 40, 41, 42, 43, 44, 45, 46, 47,...
aa, an, as, at, ca, da1, db1, dc1, ea2, eb2, ec2, ea4, eb4, ea5, eb5,...
fa, fb, fc, fd, fe, ff, lb, lf, lk, lx, mb, mf, ml, mn, mq, mu, mv),...)
d0, [], [], [], [], lb, ub, @(d) ABBcostCombined_constr_fun(d, Inom,...
Unom, f, P0eval,...
Pkeval, Spfw, Yokedist, Ducts, Disttnk, RR, AR, ML, Distph, Dcorcov,...
Price1, Price2,...
p4, p8, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14,...
y15, y16, y17, y18, y19, y20, y21, y22, y23, y24,...
y25, y26, y27, y28, y29, y30, y31, y32, y33, y34, y35, y36, y37,...
y38, y39, y40, y41, y42, y43, y44, y45, y46, y47,...
aa, an, as, at, ca, da1, db1, dc1, ea2, eb2, ec2, ea4, eb4, ea5, eb5,...
fa, fb, fc, fd, fe, ff, lb, lf, lk, lx, mb, mf, ml, mn, mq, mu, mv), option);

Dnom(i) = d(1);

Hlimb(i) = d(2);

B(i) = d(3);

An(i) = d(4);

objj(i) = ABBcostCombined_obj_fun(d, Inom, Unom, f, P0eval, Pkeval, Spfw,...
Yokedist, Ducts, Disttnk, RR, AR, ML, Distph, Dcorcov, Price1, Price2,...
p4, p8, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14,...
y15, y16, y17, y18, y19, y20, y21, y22, y23, y24,...
y25, y26, y27, y28, y29, y30, y31, y32, y33, y34, y35, y36, y37,...
y38, y39, y40, y41, y42, y43, y44, y45, y46, y47,...
aa, an, as, at, ca, da1, db1, dc1, ea2, eb2, ec2, ea4, eb4, ea5, eb5,...
fa, fb, fc, fd, fe, ff, lb, lf, lk, lx, mb, mf, ml, mn, mq, mu, mv);

objQ(i) = (ABBcostCombined_obj_fun(d, Inom, Unom, f, P0eval, Pkeval, Spfw,...
Yokedist, Ducts, Disttnk, RR, AR, ML, Distph, Dcorcov, Price1, Price2,...
p4, p8, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14,...
y15, y16, y17, y18, y19, y20, y21, y22, y23, y24,...
y25, y26, y27, y28, y29, y30, y31, y32, y33, y34, y35, y36, y37,...
y38, y39, y40, y41, y42, y43, y44, y45, y46, y47,...
aa, an, as, at, ca, da1, db1, dc1, ea2, eb2, ec2, ea4, eb4, ea5, eb5,...
fa, fb, fc, fd, fe, ff, lb, lf, lk, lx, mb, mf, ml, mn, mq, mu, mv))/436470;

% 436470 (UCM); % 382040 (simple cost function)

end

Kurtosis = kurtosis(objj)
Skewness = skewness(objj)
Mean_Dnom=mean(Dnom)
Mean_Hlimb=mean(Hlimb)
Mean_B=mean(B)
Mean_An=mean(An)
Mean_Objective=mean(objj)
Standard_deviation_Norm_obj=std(objj)/mean(objj)
Standard_deviation_obj=std(objj)

Normalized_Quantile=quantile(objQ,[0.025 0.05 0.1 0.9 0.95 0.975 0.99])
c=ABBcostCombined_constr_fun(d,Inom,Unom,f,P0eval,Pkeval,Spfw,...
  Yokedist,Ducts,Disttnk,RR,AR,ML,Distph,Dcorcov,Price1,Price2,...
  p4,p8,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,y13,y14,...
  y15,y16,y17,y18,y19,y20,y21,y22,y23,y24,...
  y25,y26,y27,y28,y29,y30,y31,y32,y33,y34,y35,y36,y37,...
  y38,y39,y40,y41,y42,y43,y44,y45,y46,y47,...
  aa,an,as,at,ca,da1,db1,dc1,ea2,eb2,ec2,ea4,eb4,ea5,eb5,...
  fa,fb,fc,fd,fe,ff,lb,lf,lk,lq,lx,mb,mf,ml,mn,mq,mu,mv);

ObjN=objj/mean(objj);

E.2.2  Objective function

( Intentionally left blank, due to confidential information )

E.2.3  Constraint function

( Intentionally left blank, due to confidential information )

E.2.4  Initiate constraints

( Intentionally left blank, due to confidential information )