Norwegian University of Science and Technology

NTNU

Department of Chemical Engineering

Associate Prof. Johannes Jäschke

Stochastic Optimisation

Case Study: Chance constraint optimisation of an hydrodesulfurization unit

TKP 4550 Specialization Project

by

Morten Aulin Moen

Co-supervisor: Adriaen Verheyleweghen

December 18, 2015

Abstract

This report give an explanation of different methods used for solving stochastic optimisation problems. The methods are simple recourse model, general recourse mode, robust optimisation, multistage recourse model and chance constraint optimisation. Each method are applied to a simple linear stochastic example, for showing how the method work.

A case study of an hydrodesulphurization unit is used to show how stochastic optimisation can be used on a real a word problem. Where the optimal values will be used fora model predictive controller (MPC). Both online optimisation and offline optimisation will be used for the MPC. The MPC is controlling the flow of hydrogen from 3 different inlet streams. Where the optimisation objective is to find optimal control settings for the streams, from an economic perspective. There are 2 sources of uncertainty in the model, given by a binormal probability distribtuion. Monte Carlo simulations are carried out to find the probability distribution of the chance constraints. Then a back-off is calculated such that the constraints holds with a given certainty level. Implementing the back-off, the optimisation problem is solved. Monte Carlo simulations are carried out for testing the accuracy of the back-off calculation. The test show that the back-off are too small. The reason are the assumption that the probability distribution of the chance constraint are normal, as the outcomes are asymmetric. For the MPC with offline optimisation the results are conservative compared with the online optimisation. The problem of online optimisation is the computational time.

Preface

This report was written as a part of the course TKP 4550 Specialization Project autumn 2015 during the fifth year of the chemical engineering degree at the Norwegian University of Science and Technology.

I would like to express my deep gratitude to my supervisor associate professor Johannes Jäschke for patient guidance, enthusiastic encouragement and useful critiques during this project. I would like to express my very great appreciation to my co-supervisor phd candidate Adriaen Verheyleweghen for helpful feedback and lively discussion.

Contents

1 Introduction

In an increasingly competitive market optimal operation of a chemical plant is crucial for the plant to be competitive. As it is often difficult to get accurate measurements or predict certain parameters accurately in some parts of the plant. In addition there are often uncertainty in the parameters or measurements. Because of this some parameters are defined by some probability distribution not a given value. Back-off is usually added so the outcomes from the probability distribution do not cause constraint violations. If there are constraint violations it could lead to economical loss or even instability of the plant. So an accurate back-off is crucial for operating the plant as close as possible to the optimum without having constraint violations. With stochastic optimisation these uncertainties can be added into the optimisation procedure. As the uncertainties are included in the optimisation a more accurate optimum is found, so the back-off can be decreased or avoided completely. With this, the plant can operate closer to the optimal and become more competitive [Van Hessem 2004],[Feedback 2013].

The optimisation is often used as set-point for model predictive control, (MPC). If one attempts to use set-point that is unfeasible this can lead to instability in the system. So it is especially important to find a optimum that are unaffected by the outcome from the uncertainties. Robust optimisation has been used to find a optimum that the outcome from the uncertainties have little affect on. Robust optimisation tries to minimise the difference between mean value and each outcome. Because of this robust optimisation often finds optimum that hold for many possible outcomes of the uncertainties. Because the optimum holds for many possible outcome scenarios the optimum is not effected so much by the outcomes. Because of this the optimum are usually a conservative optimum. But in more resent years other stochastic optimisation method have been used, like multistage and chance constraint [Bertsimas, Brown, and Caramanis 2011],[Li P.; Arellano-Garcia 2006],[Wendt, Li, and Wozny 2002].

Chance constraint optimisation has in recent years found a more focus in the control community. One reason for this are that one look at the probability outcomes of holding a constraint with some certainty level. In this way one can exclude the extreme outcomes. It is good as one do not want to model after the extreme outcomes in most cases, as the probability of these outcomes have a very low probability. Excluding the extreme outcomes one can still get an accurate estimation of the back-off without increasing it too high. This is the mean reason the increased focus on chance constraint in the control community. Another advantage of chance-constrained optimisation is that it does not lead to an increased number of variables, as some other stochastic optimisation methods do. But one needs to have the probability distribution of the stochastic variables. One also need to solve probability distribution function of the constraints. Often this can be complex to do analytically, especially if one have a large set variables. Instead an approximation method or simulation method are often used. In this project the simulation method was used. Where Monte Carlo simulation were used to find an approximation of the probability distributions [Sobol 1994], [Zhao et al. 2015],[Wendt, Li, and Wozny 2002].

Section 2 different stochastic optimisation method are explained. Section 3 goes through methods used for solving the stochastic optimisation of an hydrodesulfurization unit with chance constraint. In Section 4 the results are presented and discussed. Finally in Section 5, the conclusion of the project is presented with further work [Navia et al. 2014].

2 Stochastic Linear Programming

Linear programming (LP) are an optimisation problem where the objective function and constraints are linear. Stochastic linear programming (SLP) are a special kind of LP, where at least one of the parameters is stochastic. This is often true in engineering, as it is common that some parameters is given with some uncertainty. The reason are that parameters are often found through experiments of data collections, and not analytically. One approach can be to use the mean value of the parameter, and solve the deterministic problem. The problem with using the mean is; that it can lead to constraint violations or poor optimal depending on the outfall of the stochastic variables. So for engineers this is not recommended. But one want to to be able to use the powerful solving algorithm that exist for LP problem, one example is the Simplex method. Because of this one want to reformulate the SLP to a LP. Depending on the stochastic variable and the formulation of the SLP there are different methods to do the reformulation. Some of the commonly used methods will be explained here. The reformulation methods usually increase the number of variables. Because of this the reformulated SLP are often large problems that demands much computational power to solve.

A general formulation of the SLP is:

$$\begin{aligned} & \min \quad c(\xi)^T x \\ s.t \quad & Ax = b \\ & T(\xi)x = h(\xi) \\ & lb \leq x \leq ub \end{aligned} \tag{2.1}$$

Where $c(\xi)$ is the objective function vector dependent on the random vector ξ, $c(\xi) \in \mathbb{R}^n$, x are the decision variables, $x \in \mathbb{R}^n$, A is the constraint matrix with deterministic values only, $A \in \mathbb{R}^{m \times n}$, b is the constraint value vector, $b \in \mathbb{R}^m$, $T(\xi)$ is the constraint matrix dependent on the random vector ξ, $T(\xi) \in \mathbb{R}^{q \times n}$, $h(\xi)$ is the constraint value vector dependent on the random vector ξ, $h(\xi) \in \mathbb{R}^q$, lb is lower bound vector, $lb \in \mathbb{R}^n$, and ub is the upper bound vector, $ub \in \mathbb{R}^n$.

The formulation in (2.1) has only equality constraints. If one has inequality, these can be rewritten with use of slack variables [Robinson 2006, page 8], [Kall and Mayer 2005, page 356]. In (2.1) there are stochastic parameters in both the objective function and the constraints. One usually wants to formulate the problem such that there are only stochastic parameters in the constraints or objective function. As one usually want to see how uncertainty in the objective function or in the constraints effect the optimum. In this project the focus will be on stochastic parameters in the constraints only. The reason is that in chemical processes the uncertainty is often in some of the parameters in of the process. If one has stochastic parameters in the objective function only, one can solve the dual problem. The dual problem will have the stochastic parameters in constraints [Robinson 2006, page 359-362],[Kall and Mayer 2005, page 19-22]. Some of the most common solving methods for SLP are presented below with an example for showing how the methods works.

2.1 Simple Recourse Model

The simple recourse model is used when all decision variables need to be decided in the first step of the problem. Typical problems can be decision problem with immediate effects, like automatic control system for cars. Because of this, the simple recourse model usually gives a conservative optimum. The reason is that one has to find an optimum that satisfies all possible outcomes of the stochastic parameters. The advantageous of the simple recourse model is that it is easy to understand and use. The increase of variables are linear dependent on the number of outcomes of the stochastic parameter.

The simple recourse model is built on the Lagrangian relaxation of the problem. The constraints are added to the objective function and multiplied with some weighting matrix [Kall and Mayer 2005, page 79]. The weight matrix is a square matrix, where all the elements are non-negative numbers. The diagonal elements are weight of breaking a constraint. The off-diagonal elements are the weight of breaking a combination of two constraints. Because one wants to have a linear objective function, the off-diagonal elements are usually zero. Implementing the weighting of constraints in (2.1), the SLP can be written as:

$$\begin{aligned} &\text{On matrix form:} \\ &\min \quad c^T x + Q\left(|T(\xi)x - h(\xi)|\right) \\ &\text{On summation form:} \\ &\min \quad \sum_{i=1}^{n}\left(c_i^T x_i\right) - \sum_{j=1}^{r}|q_j\left(T_j(\xi)x_j - h_j(\xi)\right)| \end{aligned} \tag{2.2}$$

For the matrix form of (2.2) Q is the weight matrix, $\mathrm{Q} \in \mathbb{R}^{q\times n}$, with positive values in the diagonal if constraint added, otherwise zero. For the summation form of (2.2), q_j is the weighting of row j and r is the number of constraints added to the objective function, and q_j is non-negative $\forall j \in r$.

The constraints that are added to the objective function needs to be reformulated as soft constraints. This is done so the constraints can be violated. The reason that one wants to have a possibility of constraint violations is to get a relaxation of the feasible region. With a relaxation of the feasible region it becomes easier to find a solution. The constraints are reformulated to soft constraint by adding slack variable, y. The stochastic constraint from (2.1) can then be written as a soft constraint [Vlerk 1995],[Caroe and Schultz 1999]. Where the constraints can be written as:

$$T(\xi)x + W(\xi)y = h(\xi) \tag{2.3}$$

Where $W(\xi)$ is the constraint matrix for y dependent on the random vector ξ, $W(\xi)$ $\in \mathbb{R}^{q\times n}$, and y are the slack variables, $\mathrm{y} \in \mathbb{R}^n$.

For the simple recourse model one assumes that $W(\xi)$ is independent of ξ and $W = (I, -I)$. Then one introduce a new set of variables called recourse variables, $y =(y^+, y^-)$ [Prékopa 1995, page 374], Where y^+ and y^- are defined as:

$$
\begin{aligned}
y^+ &= [T(\xi)x - h(\xi)] \\
y^- &= [h(\xi) - T(\xi)x] \\
y^+, y^- &\geq 0
\end{aligned} \tag{2.4}
$$

Meaning that y^+ and y^- are measurements of the constraint violation. Instead of using the constraints in the objective function like what was done in (2.2), with the implementing the slack variables. Assume that ξ are given by some discrete probability distribution. If ξ have continuous distribution function, one need discretize it, or one get infinite many outcomes possibilities of ξ. With the use of recourse variables instead of weighted constraint violations, the objective function from (2.2) can be written as:

$$
\min \quad c^T x + Q \sum_{s=1}^{S} (p_s(y_s^+ + y_s^-)) \tag{2.5}
$$

Where s is an outcome of the probability distribution, S is the total number of possible outcomes and p_s is the probability of outcome s.

Now equation (2.1) can be rewritten, using Equation (2.5) and Equation (2.3). Given that $W = (I, -I)$, the probability distribution is known and discrete. Then the formulation of the SLP with simple recourse is:

$$
\begin{aligned}
& \min \quad c^T x + \sum_{s \in S} Q(p_s(y_s^+ - y_s^-)) \\
s.t \quad & Ax = b \\
& T_s x + W^+ y_s^+ - W^- y_s^- = h_s, \quad \forall s \in S \\
& lb \leq x \leq ub \\
& y_s^+, y_s^- \geq 0, \quad \forall s \in S
\end{aligned} \tag{2.6}
$$

Where s is a given outcome of the probability distribution, s $\in S$, Q is the weight matrix, Q $\in \mathbb{R}^{q \times n}$, p_s is the probability of outcome s, y_s^+ and y_s^- are the recourse variables for outcome s, T_s is the constraint matrix for outcome s, $T_s \in \mathbb{R}^{r \times n}$, W^+ and W^- are identity matrix and the negative identity matrix respectively, W^+ and $W^- \in \mathbb{R}^{r \times r}$, and h_s is the constraint value vector for outcome s, $h_s \in \mathbb{R}^r$.

Looking at the formulation of the simple recourse model given in Equation (2.6) that the problem has a linear objective function and linear constraint. The problem is consequently an LP. This means that an algorithm for LP can be used for solving the problem, for example with the simplex method [Robinson 2006].

Example

An example taken from article about stochastic optimisation are presented here and solved using the simple recourse model Sen and Higle 1999. This is so that reader can get a better understanding of how the simple recourse model works. The problem formulation is:

$$
\begin{aligned}
& \min -x_2 \\
s.t \quad & x_1 + x_2 + x_3 = 2 \\
& \tilde{a}_{21}x_1 + \tilde{a}_{22}x_2 + x_4 = 2 \\
& -1 \leq x_1 \leq 1 \\
& x_j \geq 0, \quad j = 2, 3, 4
\end{aligned} \tag{2.7}
$$

Where $\tilde{a}_{21}$, and $\tilde{a}_{22}$ have a known joint probability distribution, given as:

$$
(\tilde{a}_{21}, \tilde{a}_{22}) = \begin{cases} (1, \frac{3}{4}) & \text{with probability} \quad \frac{1}{2} \\ (-3, \frac{5}{4}) & \text{with probability} \quad \frac{1}{2} \end{cases} \tag{2.8}
$$

This problem has only two uncetain parameters, $(\tilde{a}_{21}, \tilde{a}_{22})$, and the distribution is known and discrete. The first step is to implement the penalty of constraint violation into the objective function. Also add the slack variable, y, in the constraint. The objective function from (2.8) is now:

$$
\min \quad -x_2 + Q(|\tilde{a}_{21}x_1 + \tilde{a}_{22} + x_4 - 2|) \tag{2.9}
$$

And the constraints are:

$$
\tilde{a}_{21}x_1 + \tilde{a}_{22} + x_4 + w_s y_s = 2, \quad \forall s \in S \tag{2.10}
$$

Then introduce the recourse variables and assume that $w_s = (1, -1)$, $\forall$ s $\in$ S. Le the diagonal elements of the weight matrix be 5, otherwise 0. The probability constraints are then transformed into two new constraints given as:

$$
\begin{aligned}
x_1 + \frac{3}{4}x_2 + x_4 + y_1^+ - y_1^- &= 2 \\
-3x_1 + \frac{5}{4}x_2 + x_4 + y_2^+ - y_2^- &= 2
\end{aligned} \tag{2.11}
$$

From the probability distribution given in (2.8) the probability is 50 % for both outcomes. So that the objective function with the recourse variables can be written as:

$$
\min \quad -x_2 + \frac{5}{2}y_1^+ + \frac{5}{2}y_1^- + \frac{5}{2}y_2^+ + \frac{5}{2}y_2^- \tag{2.12}
$$

Now the constraint and objective function are rewritten to the simple recourse model formulation. Then (2.11) and (2.12) are inserted into (2.7), yielding the following LP:

$$
\begin{aligned}
& \min -x_2 + \frac{5}{2}y_1^+ + \frac{5}{2}y_1^- + \frac{5}{2}y_2^+ + \frac{5}{2}y_2^- \\
s.t \quad & x_1 + x_2 + x_3 = 2 \\
& x_1 + \frac{3}{4}x_2 + x_4 + y_1^+ + y_1^- = 2 \\
& -3x_1 + \frac{5}{4}x_2 + x_4 + y_2^+ + y_2^- = 2 \\
& -1 \leq x_1 \leq 1 \\
& x_j \geq 0, \quad j = 2, 3, 4 \\
& y_i^* \geq 0, \quad i = 1, 2, \quad * = +, -
\end{aligned}
\tag{2.13}
$$

The problem was solved in MATLAB with the built in function *linprog* and the value of the objective function was calculated to be -1.778. The decision variables are [0.2222 1.7778 0.0000 0.4444]^T for x_1, x_2, x_3 and x_4 respectively and the recourse variables are equal to zero. The MATLAB script is given in E.1.

2.2 General Recourse Model

The general recourse model is used when some of the decision variables do not need to be decided at the start of the problem but rather after some time. Typical problems are where one can wait with making a decision, for example substitution in football. The decision variables are decided in two parts. When the decisions variables are divided into two separate parts the problem is called a two-stage optimisation problem. In a two-stage SLP the decision variables that are decided in the first stage are called first-stage decision variables, while the ones decided in the second-stage are called second stage variables. That some of the decisions variables can be decided later is advantageous since information learnt in the time period between the stages can be used in the following stage. This will lead to a less conservative solution than the simple recourse model. The increase of variables is linearly dependent on the number of possible outcome of the probability distribution [Prékopa 2003],[Sen and Higle 1999].

The formulation of a two-stage optimisation is often divided in two optimisation problems, one for each stage. With the two optimisation dependent on eachother. The first stage is given as:

$$
\begin{aligned}
& \min \quad c^T x + \mathbb{E}_\xi \left[Q(x \mid T(\xi)x + W(\xi)y(\xi) = h(\xi) \ \cap \ y(\xi) \geq 0) \right] \\
s.t. \quad & Ax = b \\
& lb \leq x \leq ub
\end{aligned}
\tag{2.14}
$$

The second stage problem, often called the recourse problem is given as :

$$
\begin{aligned}
Q(x \mid T(\xi)x + W(\xi)y(\xi) = h(\xi) \ \cap \ y(\xi) \geq 0) : \min \quad & q^T(\xi)y(\xi) \\
s.t \quad & T(\xi)x + W(\xi)y(\xi) = h(\xi) \\
& y(\xi) \geq 0
\end{aligned}
\tag{2.15}
$$

Where $\mathbb{E}_\xi$ is the estimation of the recourse problem for a given ξ, $y(\xi)$ is the slack variable dependent on the random vector ξ, $y(\xi) \in \mathbb{R}^r$, and $Q(x \mid .)$ is the solution of the recourse problem for that given constraints.

From the set of equations given by (2.14) and (2.15) one can see that the equations are dependent of each other and the random vector ξ. This means that the optimal solution is only valid for a given ξ and for every possible ξ the set of equation needs to be resolved. The general recourse model merges the two sets of equations and solve the resulting LP for all possible realisations of the random variable ξ For this to work the probability distributions needs to be known and be discrete. If the probability distribution is continuous or partly continuous it needs to be discretized.

First insert (2.15) into (2.14), by replacing $\mathbb{E}_\xi$ with the objective function of the second stage. The constraints of the second stage also need to be added to the set of constraints in the first stage. For the decisions variables it is divided into two; the first stage variables, x_1, and the second stage variables, x_2. This is done so that the constraints can be divided into constraint dependent on only x_1 or both. The separations of the variables are feasible as the constraints are linear. Then the slack variables from the objective function and the constraints, so that the problem is only a function of x_1 and x_2. The slack variables are removed because they may cause unfeasible scenario outcomes. Then with the assumption that the probability distribution is known and discrete, the formulation of the general recourse model will be:

$$\begin{aligned} & \min \quad c_1^T x_1 + \sum_{s \in S} p_s c_{2s}^T x_{2s} \\ s.t \quad & A_1 x_1 = b_1 \\ & B_s x_1 + A_{2s} x_{2s} = b_{2s}, \quad \forall s \in S \\ & lb_1 \leq x_1 \leq ub_1 \\ & lb_{2s} \leq x_{2s} \leq ub_{2s}, \quad \forall s \in S \end{aligned} \tag{2.16}$$

Where c_1 is the objective function for the first stage variables, $c_1 \in \mathbb{R}^n$, x_1 is the first stage variables, $x_1 \in \mathbb{R}^n$, S is the feasible set of outcomes, p_s is the possibility outcome s, c_{2s} is the objective function for the second stage variable for outcome s, $c_{2s} \in \mathbb{R}^q$, x_{2s} is the second stage variables for outcome s, $x_{2s} \in \mathbb{R}^q$, A_1 is the constraint matrix for first stage variable only, $A_1 \in \mathbb{R}^{m \times n}$, b_1 is the constraint value vector for the first stage variable only, $b_1 \in \mathbb{R}^m$, B_s is the constraint matrix for the first stage given outcome s, $B_s \in \mathbb{R}^{p \times n}$, A_{2s} is the constraint matrix for the second stage variables given outcome s, $A_{2s} \in \mathbb{R}^{p \times q}$, b_{2s} is constraint value vector for mixed constraint given outcome 2, $b_{2s} \in \mathbb{R}^p$, lb_1 and ub_1 is the lower and upper bound respectably for the first stage variables, lb_1 and $ub_1 \in \mathbb{R}^n$, lb_{2s} and ub_{2s} is the lower and upper bound respectably for second stage variable for given outcome s, lb_{2s} and $ub_{2s} \in \mathbb{R}^q$.

The formulation of the general recourse model given in (2.16) has a linear objective function and only linear constraints, so it is LP problem and algorithms like *Simplex Method* can be used for solving the problem.

Example

For the example showing how the general recourse model work, the example shown in section 2.1 will be used. With the decision variables x_2,x_3 and x_4 will be second stage variables. Comparison of (2.7) and E(2.16) shows that there are no constraints which depend only on the first-stage variables. It follows that A1 is an empty matrix. One can see that there are none constraint only depended on first stage variables so A_1 is an empty matrix. From the probability distribution given (see 2.8) there are two possible outcome so $S = 2$ with probability of 50% each. x_1 is the first stage variable, with a lower bound of -1 and an upper bound of 1. The second stage variables they are non-negatives real numbers. The formulation of the problem is then:

$$\begin{aligned} & \min \quad -0.5x_{21} - 0.5x_{22} \\ s.t \quad & x_1 + x_{21} + x_{31} = 2 \\ & x_1 + \frac{3}{4}x_{21} + x_{41} = 2 \\ & x_1 + x_{22} + x_{32} = 2 \\ & -3x_1 + \frac{5}{4}x_{22} + x_{42} = 2 \\ & -1 \leq x_1 \leq 1 \\ & x_s \geq 0, \quad s = 2, 3, 4 \end{aligned} \tag{2.17}$$

The problem was solved in MATLAB with the built-in function *linprog* and the value of the objective function was calculated to -1.8824. The optimal values of the decision variables are [0.1176 1.8824 0.0000 0.4706] for outcome one and [0.1176 1.8824 0.0000 0.0000] for outcome two. One can see that the optimal solution is lower when using the general recourse then the simple recourse, so this verifies that the general recourse model gives a less conservative optimal point than simple recourse model. The MATLAB script is given in E.2.

2.3 Robust optimisation

When there are uncertainties in the optimisation problem as it is with stochastic variables one will in some instances minimise the risk. The risk is measure of how stable the optimal solution is when there are disturbances. This is often done by implementing the variance or deviation from expected value or mean value into the optimisation problem. In this project, the standard deviation will be used as a measure of the risk in the context of robust optimisation. The standard deviation is used as it does not change the convexity of the problem [Bertsimas, Brown, and Caramanis 2011],[Prékopa 1995, page 255].

The definition of the standard deviation for this project is given as:

$$\begin{aligned} & (z_{ts} - \bar{z}_t)^2 \\ & \text{Where } z_{ts} \text{ and } \bar{z}_t \text{ are defined as:} \\ & c_{ts}^T x_{ts} = z_{ts} \\ & \sum_{s \in S} p_{ts} z_{ts} = \bar{z}_t \end{aligned} \tag{2.18}$$

Where s denotes a given scenario, s $\in S$, t is the stage of the problem, t $\in T$ and $\bar{z}_t$ is the standard value of z in stage t.

For the formulation of the robust optimisation the a multistage recourse model will be used. The multistage recourse model will be presented in the next subsection 2.4. One could also use simple or general recourse formulation for the robust optimisation. The reason for choosing multistage stage recourse model, are that robust optimisation is often used in economics where one make decision in multiple time steps. The formulation of the robust optimisation are build on (2.22). With addition of adding the standard deviation into the objective function. The formulation of the robust optimisation problem will then become:

$$\begin{aligned}
& \min \quad \sum_{s\in S}\left[c_1^T x_1 + \sum_{t=2}^{T} p_{ts} z_{ts} + \sum_{t=2}^{T} \theta_t p_{ts}(z_{ts} - \bar{z}_t)^2\right] \\
s.t. \quad & A_1 x_1 = b_1 \\
& A_{t1} x_1 + \sum_{\tau=2}^{t} A_{t\tau} x_\tau = b_\tau, \quad for \quad t = 2, ..., T \\
& c_{ts}^T x_{ts} = z_{ts}, \ for\ t = 2, ..., T \\
& \sum_{s\in S} p_{ts} z_{ts} = \bar{z}_t, \ for\ t = 2, ..., T \\
& lb_1 \leq x_1 \leq ub_1 \\
& lb_{ts} \leq x_{ts} \leq ub_{ts}, \ \forall s \in S, \ for\ t = 2, ...T
\end{aligned} \tag{2.19}$$

Where θ_t is the diagonal weight matrix for the standard deviations of stage t. The weighting matrix are added, so one can emphasis the deviation for each variable differently. Often there are variables that one want to be unaffected by outcome of the stochastic variables, for example control of self-optimising variables. There is a quadratic term in the objective function ($(z_{ts} - \bar{z}_t)^2$), this means that we no longer have a linear program. But all of the constraints are linear meaning that the problem is a quadratic programming (QP) problem.

Example

To illustrate how Equation (2.19) can be applied, the example from Sen and Higle 1999 see equation (2.7). Where one first it to a two-stage problem, same as in section 2.2, and add the standard deviation to the objective function. Lets us use the definition of standard deviation from equation (2.18) and lets define it by the decision variable x instead of z. From the problem we have only two stages and two outcomes, so the standard deviation will be:

$$
\begin{aligned}
&(z_{ts} - \bar{z}_t)^2 = \theta_1 p_{21}(c_{21}x_{21} - (p_2 1 c_{21}x_{21} + p_{21}c_{22}x_{21}))^2 \\
&+ \theta_2 p_{22}(c_{22}x_{21} - (p_{21}c_{21}x_{21} + p_{21}c_{22}x_{21}))^2 \\
&\text{Insert the values into the parameters and} \\
&\text{let}\theta_1 = \theta_2 = 2 \text{ the standard deviation will be:} \\
&= \frac{1}{4}x_{21}^2 + \frac{1}{4}x_{22}^2
\end{aligned}
\tag{2.20}
$$

Now that the standard deviation are defined it can be inserted into the objective function. All of the constraint is also defined as in (2.17). Then the formulation of the problem will be:

$$
\begin{aligned}
& \min \quad -0.5x_{21} - 0.5x_{22} + \frac{1}{4}x_{21}^2 + \frac{1}{4}x_{22}^2 \\
s.t \quad & x_1 + x_{21} + x_{31} = 2 \\
& x_1 + \frac{3}{4}x_{21} + x_{41} = 2 \\
& x_1 + x_{22} + x_{32} = 2 \\
& -3x_1 + \frac{5}{4}x_{22} + x_{42} = 2 \\
& -1 \leq x_1 \leq 1 \\
& x_{js} \geq 0, \quad j = 2, 3, 4, \quad \forall s \in S
\end{aligned}
\tag{2.21}
$$

The problem was solved in MATLAB with the built-in function *quadprog* and the value of the objective function was calculated to -0.5000. The decisions variables are $[0.0644 \ 1.0000 \ 0.9356 \ 1.1856]^T$ for scenario one and $[0.0644 \ 1.0000 \ 0.9356 \ 0.9431]^T$ for scenario two. The MATLAB script is given in E.4.

2.4 Multistage Recourse Model

The multistage recourse model is used when one needs to decisions subsequently over a time period. The problem are divided into multiple stages, where the number of stages are depending one the number of decision stages. If this sounds familiar is because the two-stage model presented in the section 2.2 is a special case of multistage with only two stages. The advantages of the multistage formulation is that one can get a very good optimal point. This is because uses the information gained in the previous stages in the later stages, and thereby gain a more exact problem formulation. The disadvantages with the multistage formulation is that increase of variables are exponential with the number of stages. The exponential growth of variables can easily be illustrated with a scenario tree, see Figure 2.1. Scenario trees are often used in multistage optimisation for showing the number of scenarios for each step and how the different scenarios are connected to the earlier stages [Martí et al. 2015],[Sen and Higle 1999].

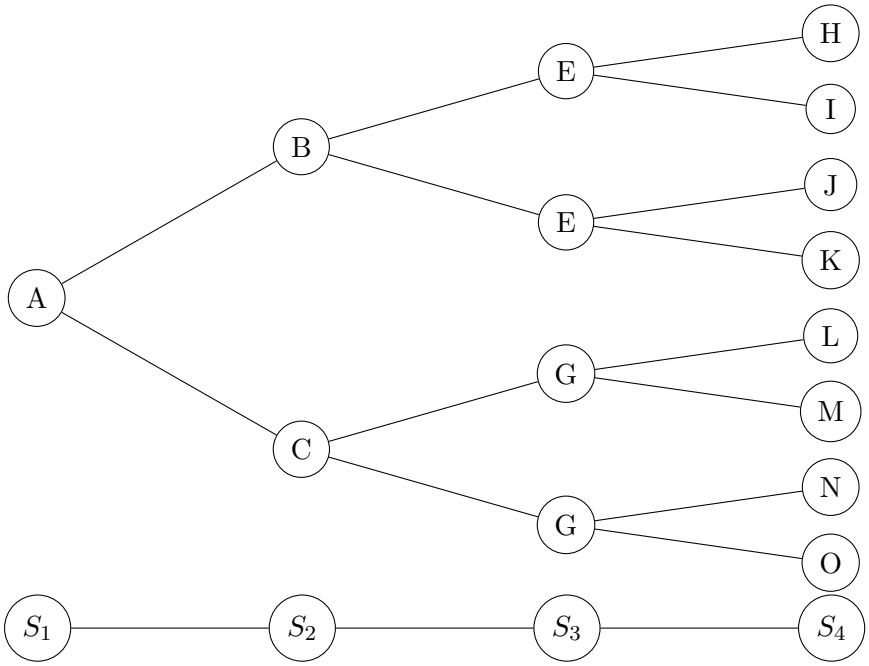

Figure 2.1: The figure shows the exponential growth in a multistage problem, where number of stages, T, is four and each scenario has two outcomes. Each scenario is numbered A-O and the stages are shown below the scenario tree, with S_i being stage number i.

For the multistage recourse model one implements all of the stages in the first stage and solves for all possible outcomes of s. This is the same that what is done for general recourse model but only for T stages, where $T \geq 2$. To see how this is done see section 2.2 and just increase the number of stages to T, where $T \geq 2$. For formulation of the multistage recourse model, let there be T number of stages, with $T \geq 2$, a scenario outcome is denoted with s, where S is set of all possible outcomes. Assume that all of the probability distributions are known and discrete. A restriction on x_t is that x_t must be $\mathcal{F}_t$ measurable, this means that the knowledge of x_t only is dependent on information given by the stages until stage t. The probability distributions needs to be independent of the decision variables [Kall and Mayer 2005],[Prékopa 2003]. Then the multistage recourse model is formulated as:

$$\begin{aligned} & \min \quad \sum_{s \in S} \left[p_{1s} c_1^T x_{1s} + \sum_{t=2}^{T} p_{ts} c_{ts}^T x_{ts} \right] \\ s.t. \quad & A_1 x_{1s} = b_s, \quad \forall s \in S \\ & A_{t1} x_{1s} + \sum_{\tau=2}^{t} A_{t\tau} x_\tau = b_\tau, \quad for \quad t = 2, ..., T \\ & x_1 - x_{1s} = 0, \quad \forall s \in S \\ & lb \leq x_{is} \leq ub, \quad for \quad i = 1, ..., n \quad \forall s \in S \end{aligned} \tag{2.22}$$

Where x_{1s} are not included with the other stages as it has to be the same for all stages independently of what scenario s is.

is defined by itself because it is the first stage variable and need to be the same independent of what scenario s we look at. That x_{1s} is denoted with s even do it is independent of the

scenario outcome, has to do with nonanticipativity of the problem. Nonanticipativity also called implementability is the requirement that x_1 needs to decided before any outcome of the possibility distributions are known. Constraint 3 forces all of the x_{1s} to be equal to each other. Often constraint 1 is exchanged with $A_1x_1 = b_1$ and constraint 3 is removed, giving a more compact notation. The formulation used in Equation (2.22) was used so the reader could more easily see that x_1 is the same for all scenarios [Prékopa 2003],[Kall and Mayer 2005].

Example

To get a better understanding of the multistage recourse model, expanse the example given in Sen and Higle 1999 to a multistage problem. This is done with have x_2 be a third stage variable, x_3 and x_4 be second stage variables and x_1 be a first stage variable.

$$\begin{aligned} & \min \quad -x_2 \\ s.t. \quad & x_1 + x_2 + x_3 = 2 \\ & \tilde{a}_{21}x_1 + \tilde{a}_{22} + \tilde{a}_{24}x_4 = 2 \\ & -1 \leq x_1 \leq 1 \\ & x_j \geq 0, \quad j = 2, 3, 4 \end{aligned} \tag{2.23}$$

Where $\tilde{a}_{21}$, $\tilde{a}_{22}$ and $\tilde{a}_{24}$ have known distributions, given as:

$$(\tilde{a}_{21}, \tilde{a}_{22}, \tilde{a}_{24}) \begin{cases} (1, \frac{3}{4}, 1) & \text{with probability } \frac{1}{2} \\ (-3, \frac{5}{4}, -1) & \text{with probability } \frac{1}{2} \end{cases} \tag{2.24}$$

The problem has three stages with known and discrete probability distribution given in (2.24). The number of stages is 3, $T = 3$. There are two scenarios in stage number 2 and 4 in stage number 3. The multistage recourse model is then:

$$\begin{aligned} & \min \quad -0.25x_{21} - 0.25x_{22} - 0.25x_{23} - 0.25x_{23} \\ s.t \quad & x_1 + x_{21} + x_{31} = 2 \\ & x_1 + \frac{3}{4}x_{21} + x_{41} = 2 \\ & x_1 + x_{22} + x_{32} = 2 \\ & -3x_1 + \frac{5}{4}x_{22} + -x_{42} = 2 \\ & x_1 + x_{23} + x_{31} = 2 \\ & -3x_1 + \frac{5}{4}x_{23} + -x_{42} = 2 \\ & x_1 + x_{24} + x_{32} = 2 \\ & x_1 + \frac{3}{4}x_{24} + x_{41} = 2 \\ & -1 \leq x_1 \leq 1 \\ & x_{ji} \geq 0, \quad j = 2, 3, 4 \quad i = 1, ..., 4 \end{aligned} \tag{2.25}$$

The problem was solved in MATLAB with the built-in function *linprog* and gave solution of -3.000. With x_1 = -1, x_{21}, x_{22}, x_{23} and x_{24} is equal to 3, x_{31} and x_{32} is equal to 0, x_{41} is equal to 0.75 and x_{42} is equal to 4.75. The MATLAB script is given in E.3.

2.5 Chance Constraint

Chance constraint optimisation, also knows as probabilistic constraint optimisation is a form of optimisation where one or more of the constraints are given as possibility of holding the constraint. With the probability is some given value, α, where $\alpha \in [0, 1]$. The constraint can be written as:

$$\mathbb{P}(T(\xi)x - h(\xi)) \geq \alpha \tag{2.26}$$

This approach is common in reliability design engineering, where the reliability of the system is included as a chance constraint. Imagine we are designing a runway for an airfield. if we require that all airplanes that land on airfield shall have a 100% probability of safe landing on the runway it needs to cover the whole earth. It need to cover the whole because one have to consider all possible outcomes. For example the wind are one stochastic variables and an extreme outcome would be a tornado. But no airplane would fly if there was a tornado, so there are not point in including this outcome in the optimisation. So to get a more sensible solution one says that the probability of safe landing should be above so α. This α are usually given, and are often 0.99, 0.95 or 0.9. The formulation of a chance constraint optimisation is often given as:

$$\begin{aligned} & \min \quad c^T x \\ s.t. \quad & Ax \geq b \\ & \mathbb{P}(T(\xi)x \geq h(\xi)) \geq \alpha \\ & lb \leq x \leq ub \end{aligned} \tag{2.27}$$

Where $\mathbb{P}$ is the probability of the constraint to hold, α is the confidence level, $\alpha \in [0, 1]$.

The chance constraint is not a linear constraint but depends on the probability distribution. This means that the problem is nonlinear optimisation problem (NLP). Compared with LP or QP, a NLP is much harder to solve and the algorithms that exist are often complex and computationally demanding. Before solving a NLP it is important to check if the problem is convex or not. If the problem is convex it makes the problem easier to solve. In this project the focus will be on convex NLP as this is an important class of optimisation problems and there exist some algorithms for solving these problems, like the Sequential quadratic programming (SQP) [Robinson 2006] , [Wendt, Li, and Wozny 2002].

There are many different methods for solving chance-constrained optimization problems. Where the main methods are approximation and simulation, one can also solve it analytically but this is often complex to do if the problem are large. [Arnold 2013], [Van Hessem 2004] In this project the focus will be on solving the problem using simulation, namely *Monte Carlo simulation*. The example is solved analytically this is done so that reader can get a better understanding of the underlying theory and that the problem are small.

Example

For giving a better insight into chance constraint optimisation an example from Kibzun and S. 1996 about reserving air tickets will be presented. The problem is to maximise the profit of an airline company that flies from one city to another city two times a day. The airline sells some tickets in advance and the demand for tickets is so high that all of them are sold. The policy of the company is also allows costumer to return their tickets until the eve of the flight. Because of this some the seats can be empty on the first flight and the total number of tickets sold in advance can exceed the total number of seats on the first flight. The airline can reserve some seats for the second flight and some passengers have to take the second flight, but they are compensated with a fee. If the airline does not reserve enough seats on the second flight then some passengers will have to wait for the next day to fly, and the airline needs to pay a penalty. The problem is how many tickets should the airline sell in advance and how many seats should the airline reserve on the second flight. Lets define some variables.

Let n_1 be the total number of seats on the first flight and n_2 be the total number of seats on the second flight. u_2 is the number of tickets available in advance, and u_2 is the number of seats reserved on the second flight. y_1 is the number of passengers that airline makes take the second plain, y_2 is the number of passengers that whom the company have to pay a penalty to. χ is the fraction of passengers who return their tickets, and is assumed to be a known probability distribution, $\chi \in [0, 1]$. c_0 is the price of tickets for the first flight. c_1 is the cost of service for the passengers between flights, c_2 is the penalty cost, c_3 is the price of a returned ticket and c_4 is the ticket price for the second flight.

The optimal value of the y_1 and y_2 is determined by the optimisation problem given below.

$$\begin{aligned} & \min \quad c_1 y_1 + c_2 y_2 \\ s.t. \quad & y_1 + y_2 \geq u_1(1-\chi) - n_1 \\ & -y_1 \geq -u_2 \\ & y_1, y_2 \geq 0. \end{aligned} \tag{2.28}$$

Let $\Phi_1(u_1, u_2, \chi)$ be the optimal solution to (2.28), this is the cost of service between flight and penalty cost. Let the total cost be:

$$\Phi(u_1, u_2, \chi) = \Phi_1(u_1, u_2, \chi) + c_2 u_1 \chi \tag{2.29}$$

Since Φ is a function of the random variable χ it is also random. Let the quantile function also called the inverse cumulative distribution function be defined as:

$$\begin{aligned} \Phi_\alpha &= \min \ \{\varphi \mid P_\varphi(u_1, u_2) \geq \alpha\} \\ P_\varphi(u_1, u_2) &= \mathbb{P}\{\Phi(u_1, u_2, \chi) \leq \varphi\} \end{aligned} \tag{2.30}$$

Where φ defines the minimum level expenses for service and penalties, α is a given minimum value for the probability of holding the constraints, $\alpha \in [0, 1]$, P_φ is the probability function, and is defined by the probability of holding constraint, $\mathbb{P}\{\Phi(u_1, u_2, \chi) \leq \varphi\}$.

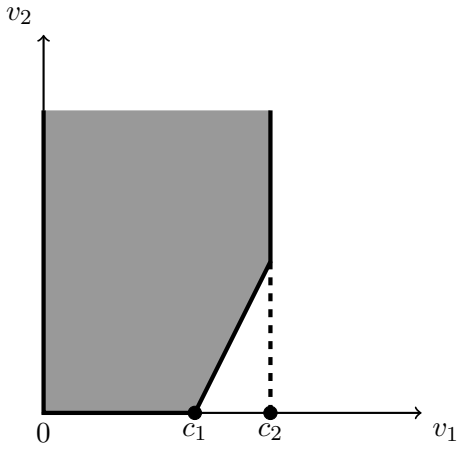

Figure 2.2: Show the feasible set dual problem defined in equation (2.32)

In order to maximise the profits of the company, with a given value of α, the expenses should be minimised. The minimisation of the expense can be expressed as:

$$\begin{aligned} & \min \quad \Phi_\alpha(u_1, u_2) - c_0 u_1 - c_4(n_2 - u_2) \\ s.t \quad & u_2 \leq n_2 \\ & u_i \geq 0, \quad i = 1, 2 \end{aligned} \tag{2.31}$$

The problem formulation is divided in two parts. This means that it is a two-stage SLP, with Equation (2.28) being the second stage problem and Equation (2.30) being the first stage problem. From the definition of the quantile function given in Equation (2.30) that the problem have a chance constraint formulation.

The solution of the problem, will be solved analytically and the solution method are taken from [Kibzun and S. 1996, page 23-27]. Instead of solving the primary second stage problem, let us solve the dual problem of the second stage problem. This will remove the random vector χ from the constraint to the objective function, resulting in a maximum problem. The dual problem is:

$$\begin{aligned} & max \quad v_1(u_1(1-\chi) - n_1) - v_2 u_2 \\ s.t. \quad & v_1 - v_2 \leq c_1 \\ & v_1 \leq c_2 \\ & v_i \geq 0, \quad i = 1, 2 \end{aligned} \tag{2.32}$$

Where v_1 and v_2 are the dual variables of the second stage problem.

From the constraint in Equation (2.31) that $u_2 \geq 0$ meaning that the maximum is on of the vertex of the shaded area in figure (2.2). Using that the maximum is on one of the vertex and instead of $\Phi_1(u_1, u_2, \chi)$ in equation (2.29), then the solution of the second stage problem is:

$$\begin{aligned} \Phi(u_1, u_2, \chi) = & c_3 u_1 \chi + max[0, c_1(u_1(1-\chi) - n_1), \\ & c_2(u_1(1-\chi) - (c_2 - c_1)u_2)] \end{aligned} \tag{2.33}$$

Now that the second stage problem is solved analytically. What remains is solving the first stage problem defined in Equation (2.31).

For solving the first stage problem let us first define the upper bound of the quantile function. The upper bound of the quantile function is defined as:

$$\psi(S, u_1, u_2) = \sup_{x \in U} \Phi(u_1, u_2, x) \tag{2.34}$$

Where S is the confidence set, $\mathcal{P}(S) \geq \alpha$, S $\in \mathbb{R}^1$, Sup is the supremum of U, U is the feasible set for equation (2.31) and x is given probability, x $\in (0, 1)$ [R and D. 2008].

Let us use the upper bound of the quantile function given in Equation (2.34) to find an upper bound of Equation (2.31). This is done by replacing the quantile function in Equation (2.31) with upper bound of the quantile function [Kibzun and S. 1996, page 171]. The upper bound is given as:

$$\begin{aligned} &\min_{(u_1,u_2)\in U} [-c_0 u_1 - c_4(n_2 - u_2) + \Phi_\alpha(u_1, u_2)] \\ &\leq \min_{(u_1,u_2)\in U} [-c_0 u_1 - c_4(n_2 - u_2) + \psi(S, u_1, u_2)] \end{aligned} \tag{2.35}$$

Now with varying the confidence set, S, the upper bound can be reduced. Lets reduce S so that the upper bounds goes towards the objective function. The optimal confidence set, S_α, that leads to the exact solution of equation (2.31) is:

$$S_\alpha = \{x \mid -c_0 u_1 - c_4(n_2 - u_2) + \Phi(u_{1\alpha}, u_{2\alpha}) \leq \Phi_\alpha(u_{1\alpha}, u_{2\alpha})\} \tag{2.36}$$

Where $u_{1\alpha}$ and $u_{2\alpha}$ be the optimal variables for problem (2.31).

Then use the definition of the quantile function (2.30) and the optimal set, S_α, for expressing the optimal solution of the objective function as function of $u_{1\alpha}$, $u_{2\alpha}$ and S_α. The expression is:

$$\begin{aligned} &- c_0 u_{1\alpha}\ c_4(n_2 - u_{2\alpha} + \Phi_\alpha(u_{1\alpha}, u_{2\alpha}) \\ &= \ -c_0 u_{1\alpha}\ c_4(n_2 - u_{2\alpha} + \psi(S_\alpha, u_{1\alpha}, u_{2\alpha}) \end{aligned} \tag{2.37}$$

Given the optimal solution of the objective function given by Equation (2.37), the optimal variables, $u_{1\alpha}$ and $u_{2\alpha}$ can be expressed as:

$$(u_{1\alpha}, u_{2\alpha}) = arg \min_{(u_1,u_2)\in U} [-c_0 u_1 - c_4(n_2 - u_2) + \psi(S_\alpha, u_1, u_2)] \tag{2.38}$$

Where $arg\ \min_{(u_1,u_2)\in U}$ is the set of elements in U that gives the global minimum of U.

As the optimal set depends on the minimum of the objective function, see Equation (2.36), the set is not explicit defined, as the quantile function is not known. But let us look at the structure of the set. The objective function is piecewise linear and convex. Meaning that the S_α is an interval. Since random variables χ has the interval [0,1], for the optimal confidence set one can search the interval [a,b] $\subset$ [0,1] which satisfying the condition:

$$\mathcal{P}\{a \leq \chi \leq b\} = \alpha \tag{2.39}$$

Equation (2.31) is equivalent to:

$$-c_0 u_1 - c_4(n_2 - u_2) + \psi([a,b], u_1, u_2) \rightarrow \min_{a,b} \min_{(u_1,u_2)\in U} \tag{2.40}$$

Where a and b are defined by Equation (2.39) and $0 \leq a \leq b \leq 1$. One can see that the b can not be an arbitrary number [0,1], and it should satisfy the constraint:

$$x_\alpha \leq b \leq 1 \tag{2.41}$$

Where x_α is the $\alpha-$quantile function for the distribution of the random variable χ, defined as:

$$x_\alpha = \min\{x \mid F(x) \geq \alpha\} \tag{2.42}$$

Then use the relation between a and b from equation (2.39) to find the limit points of a and b. The relation between the two limit points are:

$$a \;=\; \{x \mid F(b) - F(x) = \alpha\} \tag{2.43}$$

Assume that the distribution function F(x) is strictly increasing with respect to x $\in [0,1]$. This transforms the Equations (2.42) and (2.43) into equations (2.44). Then x_α can be expressed as a function of b, a(b), by using the following two expressions

$$\begin{aligned} &F(x_\alpha) = \alpha \\ &F(b) - F(a) = \alpha \end{aligned} \tag{2.44}$$

Let S be defined as the interval [a,b], where a and b are the limits points defined by equation (2.44) and $b = x_\alpha$. The function $\Phi(u_1, u_2, x)$ is convex in x. This means that the maximum of Equation (2.34) is one of the limit points of S. The equation can be written as:

$$\begin{aligned} \psi(S, u_1, u_2) = &\max_{i=1,3} \max\{l_i(u_1, u_2, a(b)), l_i(u_1, u_2, b)\} \\ &\text{With } l_1, l_2 \text{ and } l_3 \text{ defined as:} \\ &l_1(u_1, u_2, x) = c_3 u_1 x \\ &l_2(u_1, u_2, x) = c_3 u_1 x + c_1(u_1(1-x) - n_1) \\ &l_3(u_1, u_2, x) = c_3 u_1 x + c_1(u_1(1-x) - n_1) - (c_2 - c_1)u_2 \end{aligned} \tag{2.45}$$

Lastly, let equation (2.40) be equivalent to:

$$\Phi^0_\alpha = \min_{b_\alpha \leq b \leq 1} f_\alpha(b) \tag{2.46}$$

Where the function $f_\alpha(b)$ is the upper bound for the optimal value of the quantile function. $f_\alpha(b)$ is defined as:

$$f_\alpha(b) = \min_{(u_1,u_2)\in U}[-c_0u_1 - c_4(n_2 - u_2) + \psi(b, u_1, u_2)] \tag{2.47}$$

Where $\psi(b, u_1, u_2)$ is given by Equation (2.45). Let u_3 represent $\psi(b, u_1, u_2)$. Then the first stage problem is given as:

$$\begin{aligned} f_\alpha(b) = & \min_{(u_1,u_2,u_3)}[-c_0u_1 - c_4(n_2 - u_2) + u_3] \\ s.t \quad & u_2 \leq n_2 \\ & c_3a(b)u_1 - u_3 \leq 0 \\ & (c_4a(b) + c_1(1 - a(b)))u_1 - (c_2 - c_1)u_2 - u_3 \leq c_1n_1 \\ & c_3bu_1 - u_3 \leq c_2n_1 \\ & (c_3b + c_1(1 - b))u_1 - u_3 \leq 0 \\ & (c_3b + c_2(1 - b))u_1 - (c_2 - c_1)u_2 - u_3 \leq c_2n_1 \end{aligned} \tag{2.48}$$

Now the original two-stage stochastic optimisation problem has been transformed into a linear minimisation problem. Meaning that the problem can be solved using the *Simplex Method* or other LP algorithms. Now the only things that are missing are the parameters and the probability distribution. The probability distribution used in the calculation is:

$$F(x) = \frac{1 - e^{-20x}}{1 - e^{-20}}, \quad x \in [0, 1] \tag{2.49}$$

The parameters used are given in table :

Variable	c_0	c_1	c_2	c_3	c_4	n_1	n_2	α
Value	300	30	1000	280	250	350	350	0.99

Table 2.1: Parameters used in the calculations, for solving the example defined in section 2.5

The problem was solved in MATLAB with the built-in function *linprog* and the value of the objective function was calculated to -175620. The decision variables are $[374\ 0\ 24120]^T$ for u_1, u_2 and u_3 and respectively . The MATLAB script is given in E.5.

3 Optimisation of an Hydrodesulphurisation Process

In this section the theory discussed in section 2 and apply it for an hydrodesulphurisation process (HDS). The process are described in detail in Navia et al. 2014. But a short described are given here for convince. The process is part of a larger refinery, where the HDS process removes sulphur from a hydrocarbon flow. The sulphur is removed such that hydrocarbon product meets environmental constraints on sulphur content. The sulphur is removed from the hydrocarbons in a fixed bed reactor, where the hydrocarbons are mixed with hydrogen gas. The sulphur bonds with the hydrogen and reacts to hydrogen sulfide gas in a catalyst reaction. The aim are to optimise the HDS from an economical perspective. This is done by minimising the hydrogen cost. The problem is that there are uncertainties in the reaction rate and hydrogen fraction in one of the hydrogen streams. Because of this, the optimisation problem becomes stochastic and the method that will be used in this report is chance constraint optimisation.

3.1 Description of the Hydrodesulphurisation Process

A flow and control diagram of the HDS process is given in the figure below

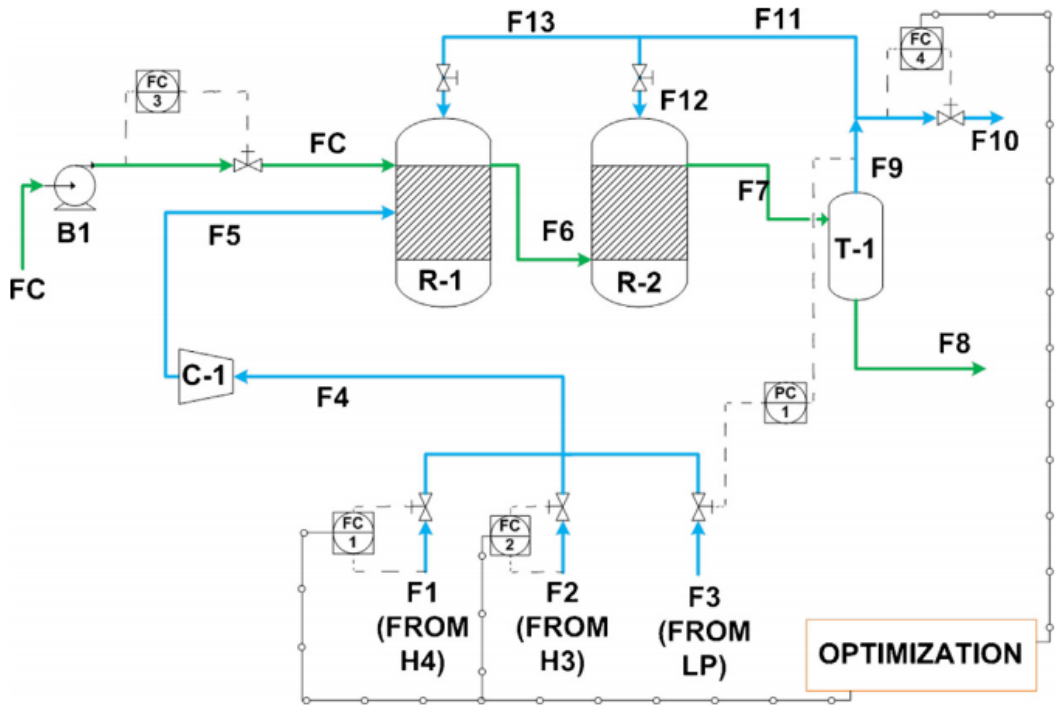

Figure 3.1: The diagram of the hydrodesuphurisation process [Navia et al. 2014].

There are three hydrogen flows marked F1, F2 and F3 in the flow diagram. F3 is a recycle stream with low hydrogen concentration and F1 and F2 are hydrogen production flows from another part of the refinery. The three streams are mixed and go through a compressor (C-1), so the pressure is held constant in the inflow to the reactors. The hydrocarbon flow into the reactor is marked FC. The reactors are marked R-1 and R-2. The reactor are set in two stages, with R-1 being the first reactor and R-2 the second. The

outlet flow from R-2 is marked F7 and is fed into a flash tank where the hydrocarbons are separated from the hydrogen and sulfide gas. The hydrocarbons are then leaving in the product flow F-8. Some of the hydrogen are recycled back into the reactors, while the rest goes out of a purge stream marked F10.

For control there are are four feed controllers and one pressure controller. The feed controller of the hydrocarbons are set. The pressure controller adjust the pressure of F3 such that the pressure is kept constant. The three controllers left are controlling the hydrogen concentration in the reactors. It is important that the hydrogen ratio inside the reactors does not go below 0.7 otherwise the catalyst starts to degenerate. From an economic perspective one want to minimise the cost associated with production cost of stream F1 and F2. The goal of the controller is to keep the hydrogen ratio above 0.7 while minimising the use of F1 and F2.

3.2 Model Description

Some assumptions are made to simplify the model of the process. The assumption are listed below.

1. Perfect temperature control of the reactor.
2. Perfect control of pressure.
3. First order reaction.
4. Perfect separation of hydrocarbons and hydrogen in the flash tank.
5. The reactors are model as one reactor

The assumptions are made so that one only has to consider the mass and component balances for the system. Also that one only has to consider the dynamics of one combined reactor.

With a first order reaction the consumption of hydrogen can be described as:

$$\tau \frac{d\, F_x^{H2}}{dt} + F_x^{H2} = F_{HC}\rho \tag{3.1}$$

Where τ is the time constant of the reaction, F_x^{H2} is the consumption rate of hydrogen inside the reactor, t is the time, F_{HC} is the volumetric flow of hydrocarbons into the reactor, and ρ is the reaction rate constant for the given hydrocarbons.

The hydrogen fraction inside the reactor is given as:

$$\frac{VP}{ZRT}\frac{d\, X_{H2}}{dt} = F_5 X_5 - F_{10} X_{H2} - F_X^{H2} \tag{3.2}$$

Where V is the volume of the reactors, P is the pressure inside the reactor, Z is the compressibility factor, R is the ideal gas constant and T is the temperature, X_{H2} is hydrogen fraction inside the reactor, F_5 is the inflow stream into the reactor, X_5 is the hydrogen fraction in stream F_5 and F_{10} is the purge stream.

For the mass balance over the reactor is given as:

$$F_5 = F_{10} + F_X^{H2} \tag{3.3}$$

The mixing point of the three inlet hydrogen streams, the mass balance is given as:

$$F_5 = F_1 + F_2 + F_3 \tag{3.4}$$

The component balance is given as:

$$F_5 X_5 = F_1 X_1 + F_2 X_2 + F_3 X_3 \tag{3.5}$$

Where X_i is the hydrogen fraction in stream i, where i = 1,2,3.

The constraints given by the catalyst and the compressor are:

$$X_{H2}^{lb} \leq X_{H2} \tag{3.6a}$$
$$X_5^{lb} \leq X_5 \tag{3.6b}$$

For the hydrogen flow in the inlet streams there are upper and lower bounds given as:

$$F_1^{lb} \leq F_1 \leq F_1^{up} \tag{3.7a}$$
$$F_2^{lb} \leq F_2 \leq F_2^{up} \tag{3.7b}$$
$$F_3^{lb} \leq F_3 \leq F_3^{up} \tag{3.7c}$$

3.3 Optimisation

The optimisation problem is formulated as a dynamic optimisation problem. Where the goal is to minimise the cost of hydrogen from the two produced hydrogen feed streams, F_1 and F_2. The objective function is given as:

$$J = \int_{t_0}^{t_f} C_{H4} X_1 F_1 + C_{H3} X_2 F_2 dt \tag{3.8}$$

Where J is the cost, t_0 is the time at t=0, t_f is the final time, C_{H4} is the cost of producing hydrogen for stream F_1, X_1 is the hydrogen fraction in F_1, C_{H3} is the cost of producing hydrogen for stream F_2, and X_2 is the hydrogen fraction in stream F_2.

The optimisation problem can now be formulated with (3.8) as the objective function. With the constraint defined by (3.1)-(3.5). Lower and upper bounds are given by (3.6) and (3.7). Because of uncertainty in the model the problem must first be reformulated using one of the models described in Section 2. Chance constraint was chosen in this project, where the robustness of the model can be adjusted with so given probability, *alpha*.

3.4 Stochastic Optimisation Formulation

The constraints given by (3.6), are the two constraints that we want to satisfy with a probability of some α. The reason are that if they are broken there will be economical loss due to catalyst or compressor damage. With these two constraints as chance constraints, the optimisation problem is formulated as:

$$\begin{aligned}
\min_{F_1,F_2,F_{10}} \quad & \int_{t_0}^{t_f} C_{H4}X_1F_1 + C_{H3}X_2F_2 dt \\
s.t. \quad & \tau \frac{d\, F_x^{H2}}{dt} + F_x^{H2} = F_{HC}\rho(\xi_1) \\
& \frac{VP}{ZRT}\frac{d\, X_{H2}}{dt} = F_5X_5 - F_{10}X_{H2} - F_X^{H2} \\
& F_5 = F_{10} + F_X^{H2} \\
& F_5 = F_1 + F_2 + F_3 \\
& F_5X_5 = F_1X_1 + F_2X_2 + F_3X_3(\xi_2) \\
& \mathbb{P}(X_{H2}^{lb} \leq X_{H2}) \geq \alpha_{X_{H2}}, \quad \mathbb{P}(X_5^{lb} \leq X_5) \geq \alpha_{X_5} \\
& F_i^{lb} \leq F_i \leq F_i^{up}, \quad i = 1,2,3
\end{aligned} \tag{3.9}$$

Where $\mathbb{P}(\cdot)$ is the probability of satisfying the constraint, and ξ is the random vector given by a probability density function and α are a given probability that gives the confidence level of the chance constraint.

The uncertainties are given in ρ and X_3. From measurements of ρ and X_3 the joint probability distribution of the parameters is given as a binormal probability distribution. The probability distribution of the constraints are not known, so Monte Carlo simulations needs to be carried out in order to find the probability distribution of X_{H2} and X_5. Monte Carlo simulation works with one solve the deterministic optimisation problem first. Then use the optimal control variables as starting points in a simulation of the process with a given outcome of the stochastic variables. The simulations are carried out multiple times with different outcomes of the stochastic variables, this way one get an approximation of the probability distribution from the different outcome scenarios. This method of solving chance constraint optimisation problem are called the simulation method. With the probability distribution approximated the stochastic optimisation problem are solved. Then new Monte Carlo simulations are carried out to test the reliability of the model. If the reliability of the model are below some value ϵ one do the process again until one achieve convergences. The method of optimise then test with simulation are called sequential method [Sobol 1994],[Hong, Yang, and Zhang 2011].

The optimisation problem is a dynamic optimisation problem because of the differential equations given by (3.1) and (3.2). Direct multiple shooting was used to reformulate the dynamic optimisation problem. Appendix B for more about direct multiple shooting method. The formulation of the deterministic optimisation problem the direct multiple shooting is given below:

$$
\begin{aligned}
\min_{F_{1_j},F_{2_j},F_{10_j}} \quad & \sum_{j=1}^{M} C_{H4}X_1F_{1_j} + C_{H3}X_2F_{2_j} \\
s.t. \quad & \hat{F}_{x_j}^{H2^+} - \hat{F}_{x_{j+1}}^{H2^-} = 0, \quad for\ j = 1, ..., M \\
& \hat{X}_{H2_j}^{+} - \hat{X}_{H2_{j+1}}^{-} = 0, \quad for\ j = 1, ..., M \\
& F_{5_j} = F_{10_j} + F_{X_j}^{H2}, \quad for\ j = 1, ..., M \\
& F_{5_j} = F_{1_j} + F_{2_j} + F_{3_j}, \quad for\ j = 1, ..., M \\
& F_{5_j}X_{5_j} = F_{1_j}X_1 + F_{2_j}X_2 + F_{3_j}X_3, \quad for\ j = 1, ..., M \\
& X_{i_j}^{lb} \leq X_{i_j} \leq X_{i_j}^{ub}, \quad for\ j = 1, ..., M,\ i = 1, 2, 3, 5, H2 \\
& F_{i_j}^{lb} \leq F_{i_j} \leq F_{i_j}^{ub}, \quad for\ j = 1, ..., M,\ i = 1, 2, 3, 5, 10
\end{aligned}
\tag{3.10}
$$

Where M is the number of time intervals, $\hat{F_x^{H2}}$ and $\hat{X}_{H2}$ are the solutions of the differential equations, (3.1) and (3.2), from the integrator.

The solution of (3.10) is shown in Figre 3.2. The solution was implemented using the language Python and the symbolic framework of CasADi [Andersson 2013].

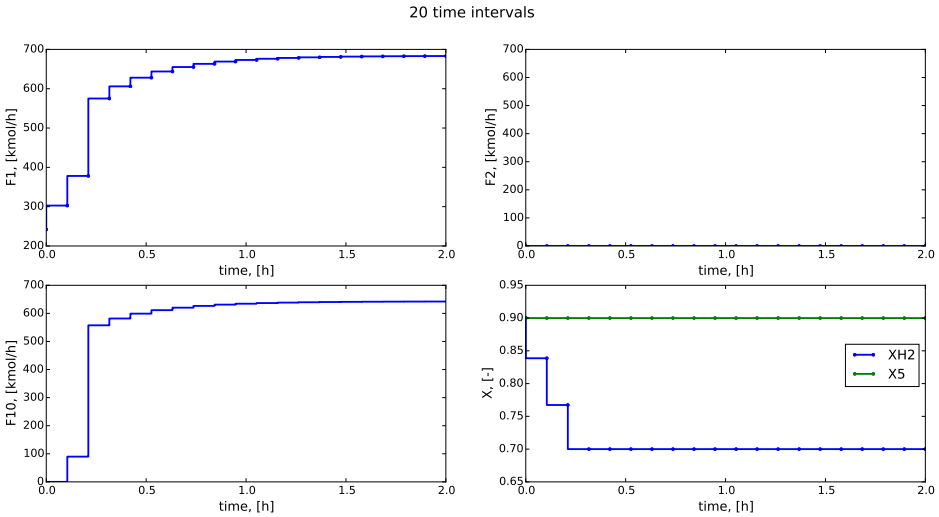

Figure 3.2: The optimal deterministic solution for the three control variables, F_1, F_2 and F_{10}. And the response of X_{H2} and X_5 from solving (3.10), with values given in table 3.1.

Parameter	Value	Unit	Parameter	Value	Unit
M	20	-	TF	2.0	h
R	0.08314	$\frac{m^3 bar}{K kmol}$	T	623.15	K
V	100	m^3	P	68.901	bar
τ	0.3	h	ρ	12.6	$\frac{kmol}{m^3}$
F_{HC}	102	$\frac{m^3}{h}$	Z	1	-
C_{H4}	88.1	$\frac{€}{Mmol}$	C_{H3}	77	$\frac{€}{Mmol}$
X_1	0.991	-	X_2	0.931	-
X_3	0.85	-	$X_{H2}(t_0)$	0.9	-
$F_x^{H2}(t_0)$	682.5	$\frac{kmol}{h}$	F_1^{lb}	0	$\frac{kmol}{h}$
F_1^{ub}	1400	$\frac{kmol}{h}$	F_2^{lb}	0	$\frac{kmol}{h}$
F_2^{ub}	790	$\frac{kmol}{h}$	F_3^{lb}	0	$\frac{kmol}{h}$
F_3^{ub}	5000	$\frac{kmol}{h}$	F_10^{lb}	0	$\frac{kmol}{h}$
F_10^{ub}	1500	$\frac{kmol}{h}$	X_5^{ub}	1	-
X_5^{lb}	0.9	-	X_{H2}^{ub}	1	-
X_{H2}^{lb}	0.7	-			

Table 3.1: The parameters, upper/lower bounds and initial condition used solving (3.10).

The solution of the optimal control variables for each step the Monte Carlo simulations can be carried out. 1000 simulations were carried out and the results of the Monte Carlo simulations are given in Figure 3.3. The mean and variance of the outcomes were calculated for each variable tofinding the normal probability distribution functions.

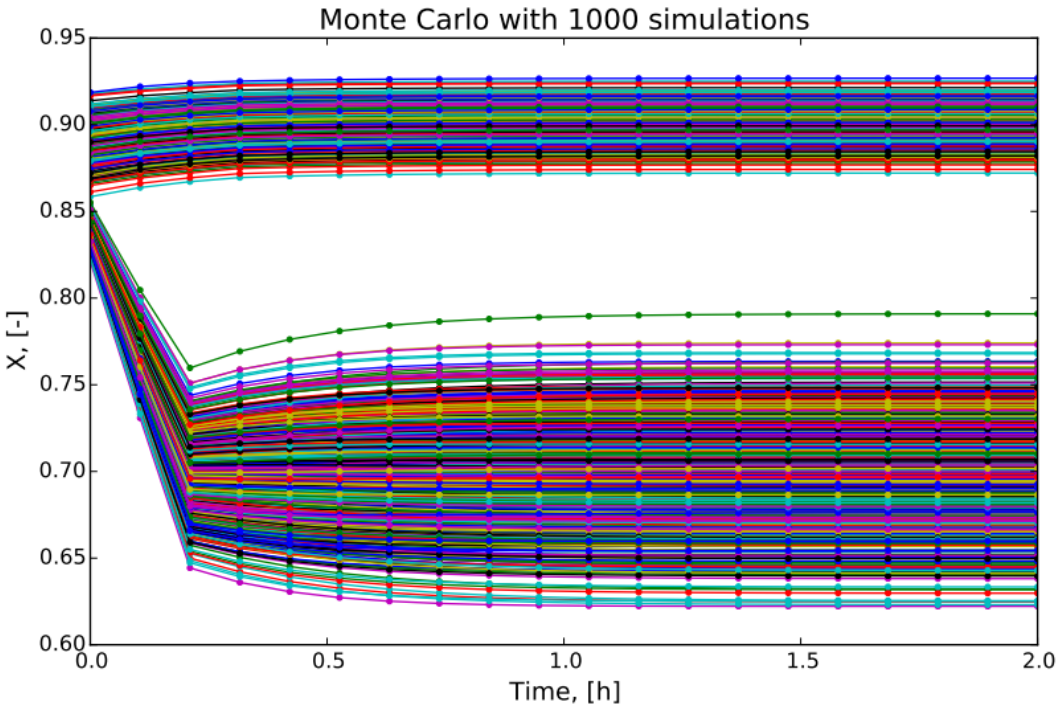

Figure 3.3: The outcome from the Monte Carlo simulations for X_{H2} and X_5, with 1000 simulations.

The probability distribution of the two variables is now known, but before solving the problem defined by (3.9), the two chance constraints needs to be reformulated. This is

done by finding a back-off such that the constraint are hold with probability of α. The back-off can be calculated with the equation below [Van Hessem 2004].

$$\text{Back-off} = F_g(\alpha)^{-1}\sqrt{Z} \tag{3.11}$$

Where F_g^{-1} is the cumulative normal Gaussian distribution function, α is the probability of satisfying the constraint and Z is the covariance. See appendix C for more about the cumulative function and covariance matrix.

The new lower bounds for X_{H2} and X_5 are the old lower bound plus the back-off. The constraints are now given as:

$$X_{H2}^{lb} + F_g(\alpha_{X_{H2}})^{-1}\sqrt{Z_{X_{H2}}} \leq X_{H2}, \quad X_5^{lb} + F_g(\alpha_{X_5})^{-1}\sqrt{Z_{X_5}} \leq X_5 \tag{3.12}$$

Then insert the constraints (3.12) in (3.9) and using direct multiple shooting method for reformulate the dynamic optimisation problem. The optimisation problem is then given as:

$$\begin{aligned}
\min_{F_{1_j},F_{2_j},F_{10_j}} \quad & \sum_{j=1}^{M} C_{H4}X_1F_{1_j} + C_{H3}X_2F_{2_j} \\
s.t. \quad & \hat{F}_{x_j}^{H2^+} - \hat{F}_{x_{j+1}}^{H2^-} = 0, \quad for\ j = 1, ..., M \\
& \hat{X}_{H2_j}^{+} - \hat{X}_{H2_{j+1}}^{-} = 0, \quad for\ j = 1, ..., M \\
& F_{5_j} = F_{10_j} + F_{X_j}^{H2}, \quad for\ j = 1, ..., M \\
& F_{5_j} = F_{1_j} + F_{2_j} + F_{3_j}, \quad for\ j = 1, ..., M \\
& F_{5_j}X_{5_j} = F_{1_j}X_1 + F_{2_j}X_2 + F_{3_j}X_3, \quad for\ j = 1, ..., M \\
& X_{i_j}^{lb} + F_g(\alpha_j)^{-1}\sqrt{Z_{i_j}} \leq X_{i_j} \leq X_{i_j}^{ub}, \quad for\ i = 5, H2\ j = 1, ..., M, \\
& X_{i_j}^{lb} \leq X_{i_j} \leq X_{i_j}^{ub}, \quad for\ i = 1, 2, 3\ j = 1, ..., M, \\
& F_{i_j}^{lb} \leq F_{i_j} \leq F_{i_j}^{ub}, \quad for\ i = 1, 2, 3, 5, 10\ j = 1, ..., M,
\end{aligned} \tag{3.13}$$

3.5 Model Predictive Control

Model predictive control (MPC) is a control method, where measurements of the current state are taken. The measurements are set as initial condition for a finite horizon optimisation, for finding the optimal control sequence for the time horizon. Then after the first control step is done, new measurement are taken and the process are repeated [Foss and Heirung 2013].

For the hydrodesulfurization unit the initial states ware found through a dynamic simulation of the plant. Then the optimisation problem defined in (3.13) was solved. The optimal control variables was then feed into a simulator, where the values for the second step are taken as initial values for the next optimisation. The solution was found using Python and the code can be found in appendix F.

4 Results and Discussion

In this section the results from the stochastic optimisation problem defined in (3.13) are given. The optimal control sequence of the MPC is also given here. The stochastic optimisation was solved using the same parameters as the deterministic problem. The values are given in Table A.1 and the probability parameters for the stochastic variables are given in Table A.2. The values for calculating the back-off and any other parameters needed are defined in table A.2.

All of the calculations were done in Python with the use of CasADi for solving the stochastic optimisation and for running the simulations. The optimisation solver used was Ipopt version 3.12.3, with the linear solver mumps. 1000 Monte Carlo simulations were carried out to find an approximation of the probability distribution of the constraints

4.1 Back-off

The outcome for each discretization point of the Monte Carlo simulation, as seen in Figure 3.3, was fitted to normal distributions. The normal fitting is shown in Figure 4.1 for X_{H2} and Figure 4.2 for X_5. The fitted normal distribtuions are shown for the first, 5th, 10th and 17th discretization points, showing the development of the normal distribution over time. The histogram of the outcomes for the same points are also shown with the normal distribution.

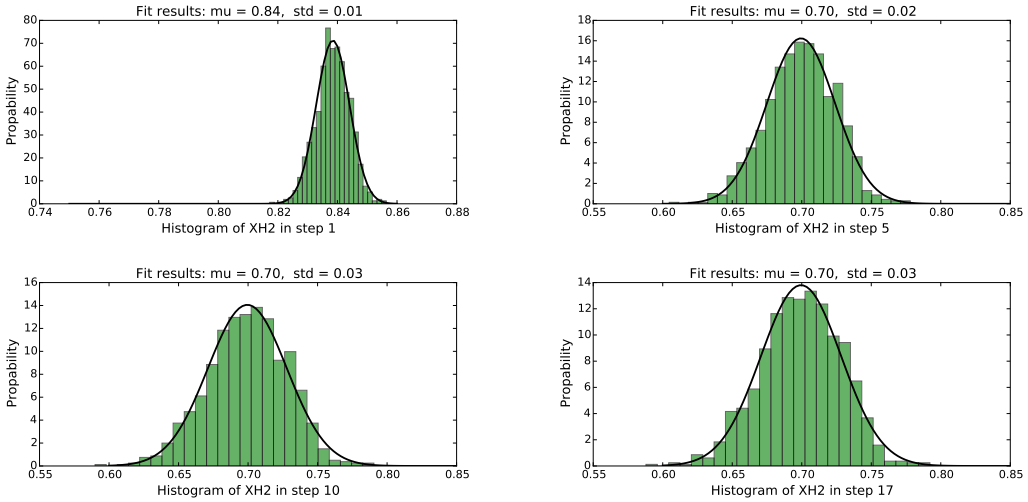

Figure 4.1: The fitted normal distributions for discretization points; 1, 5, 10 and 17 for X_{H2} shown by the black line. The histogram of the outcome for the same discretization points are given by the green bars.

From Figure 4.1 it can be seen that the assumption of normally distributed constraints seems to hold reasonably well. One can see that the variance is smaller in the first steps. The reason is that the process starts above the optimum, $X_{H2}(t_0) = 0.9$, while the optimum is at 0.7. Therefor the dynamic part is dominating in the first steps, and the outcome of the stochastic parameters has less effect on X_{H2}. When the optimum is reached the process goes towards the steady state. Here it is the outcome of the stochastic variables that dictates where the steady state is. Therefor the variances are smaller in the beginning

than in the end. This is also illustrated in Figure 3.3, where one can see that profile of X_{H2} goes towards the optimum in the beginning before fannign out.

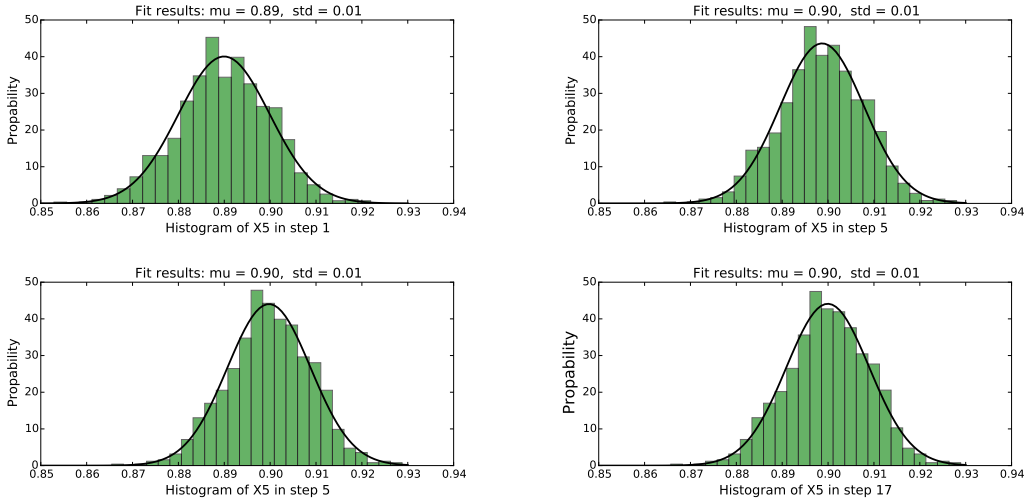

Figure 4.2: The fitted normal distributions for discretization points; 1, 5, 10 and 17 for X_5 shown by the black line. The histogram of the outcome for the same discretization points are given by the green bars.

From Figure 4.2 one can see that the normal fitting of the outcome gives a good approximation of the probability function of X_5. If one look at histogram of step one, that the mean are below 0.9. The reason for this is that the negative effect on X_5 from decreasing X_{H2}, have not been taken into account in the first step. In the two next steps this is taken into account but from the previous steps where the decrease are larger, so here the mean are above 0.9. These trends can one see in Figure 3.3. But when X_{H2} are closing to the steady state the mean of X_5 are 0.9, since there are minimal effect from the change of X_{H2}.

Comparing the two probability distributions one can see that X_{H2} has a larger variance than X_5 average mean and variance are given in table A.2. The reason for this is that X_{H2} are dependent on both stochastic parameters through F_x^{H2} and X_5. While X_5 mostly dependent on X_3. This is also the effect that was expected from the model, (3.1)-(3.5).

The back-off were calculated with use of (3.11). The back-off were implemented in the lower bound of X_{H2} and X_5 and the stochastic optimisation were carried out. The accuracy were tested with running Monte Carlo simulation. The number of constraint violations along with the back-off is shown in Figure 4.3, with the notation $_1$. As one can see from the Figure that for X_5 the number of constraint violations are over 30 % in the start. This number is over 3 times more then the expected number. From the Figure one can see that the number of constraint violations are for the most part above 10%. For correcting for the high number of constraint violations one should implement a correction factor in the back-off calculations. Then do an iterative process to find the correction factor that gives 10% constraint violation. As the computational time of the calculation already is high, see Table 4.1. And with the focus of the project being about understanding of stochastic optimisation methos, the iterative process was not implemented in the calculation of the back-off. So for future work this process should be added for getting a more accurate estimation of the process.

One can see from Figure 4.3 that the number of constraint violations are higher then 10% except in the beginning for X_{H2}. At looking at mean values from the probability distribution,the mean values are just below the lower bound. For X_5 in the start it was 0.89 instead of 0.9, as was expected. So trying to find a improved estimation of the back-off, the difference between the lower bound and mean value of the probability distribution were used. This suggestion of improvement was made by look at the probability distribution in Figures 4.1 and 4.2 and compare it with number constraint violation given in Figure 4.3. From this point out this method will be referred to as method 2 while the original back-off calculations methods will be referred to as method 1. Both method 1 and method 2 were tested with Monte Carlo simulations and the back-off and number of constraint violations are given in Figure 3.11.

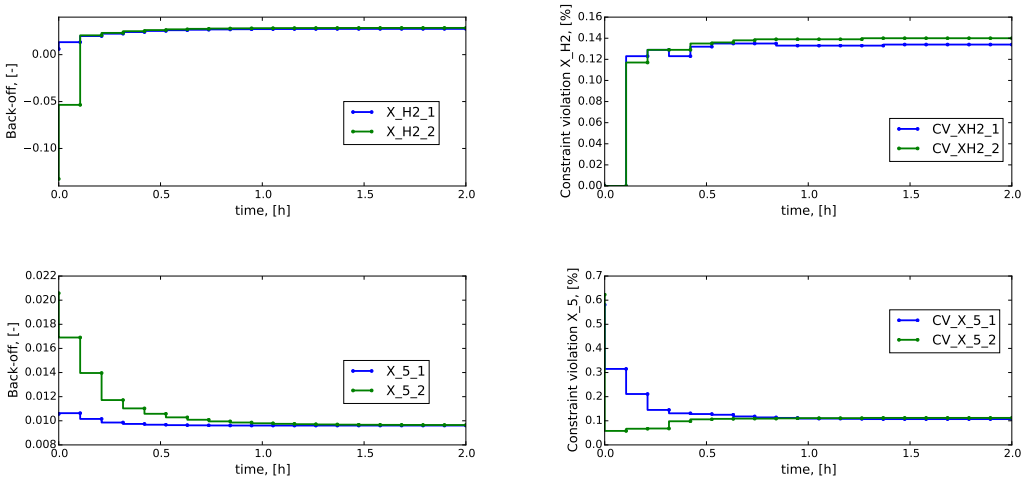

Figure 4.3: The back-off for X_{H2} and X_5 where notation 1 are for calculating the back-off given by (3.11), while notation 2 the difference between the mean value of the probability distributions and lower bounds have been taken into consideration. The constraint violations for X_{H2} and X_5 are also shown with the same notation.

From the Figure 4.3 the back-offs are negative for method 2 in the start of the time scale for X_{H2}. This is because X_{H2} starts at 0.9, which si high above the lower bound 0.7. So one get a high negative value from the difference between the lower bound and mean value of the probability distribution. As one move the lower bound down this can be problematic as one soften the constraint. But it can easily be fixed with an *if* statement that only add positive back-offs. For X_5 one can see that the back-off is larger in the start for method 2. And form the constraint violations one can see that number constraint violations are drastically decreased. So method 2 seems to improve the number of constraint violations for X_5. If one look in the middle part and the last part of the time scale, both methods seems similar. But for X_{H2} it seems that method 1 are bit better the method 2.

For the back-off calculation used further in the process method 2 was implemented for X_5 as it improved the number of constraint violations in the start. While method 1 was used for X_{H2} as it gave a better results.

4.2 Stochastic Optimisation

The lower bounds of X_{H2} and X_5 were updated with the back-off. The stochastic optimisation was calculated. The result of the stochastic optimisation is shown in Figure 4.4. The code used for solving the stochastic optimisation is given in appendix F.

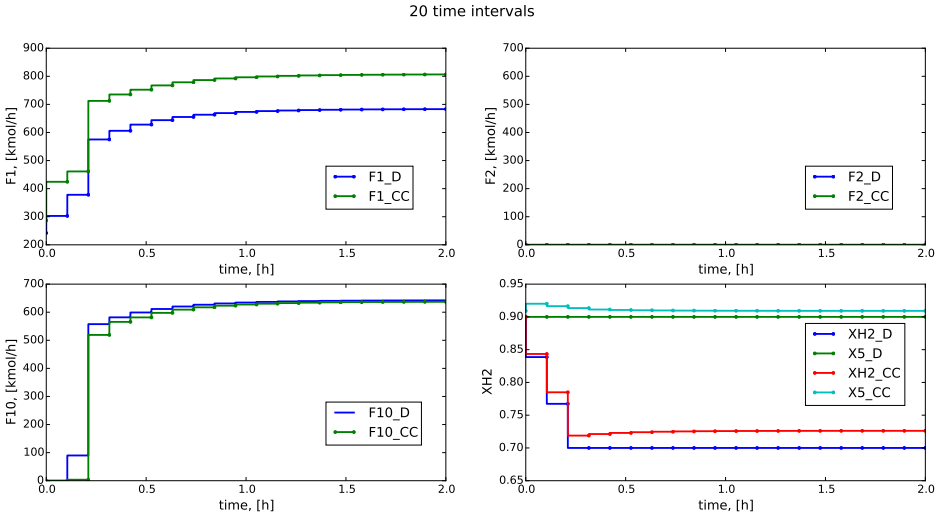

Figure 4.4: The stochastic optimisation optimum points for the three control variables, F_1, F_2 and F_{10}, given by the notation CC. While the deterministic optimum are given by D. The optimum of X_{H2} and X_5 for each discretization point are also shown, with the same notation.

The Figure 4.4 one can see that for the stochastic optimal input of F_1 is increased compared to the deterministic optimum. The reason are that with the back-off the lower bound is higher and so more of the purer stream, F_1 is added. This also decreases the effect of X_3 on X_5, as F_3 is decreased. The decrease of F_3 is shown in Figure 4.5. Where one can see that F_5 is almost the same for both optimisations, while F_3 a much lower for the stochastic optimisation.

For X_{H2} and X_5 the optimum values are higher for the stochastic optimisation then the deterministic. This can been seen in Figure 4.4. The reason for this is that one has added the back-off to the lower bounds of those two variables. One can also see the tendency of the back-off from the same plot. Where for X_{H2} the smaller back-off in the beginning then it increases with the time. This is expected since the variance for X_{H2} also increase with time, as can be see in Figure 3.3. For X_5 one can see that the back-off are largest in the beginning. This is also expected since the difference between the original lower bound and the mean value was added in the back-off.

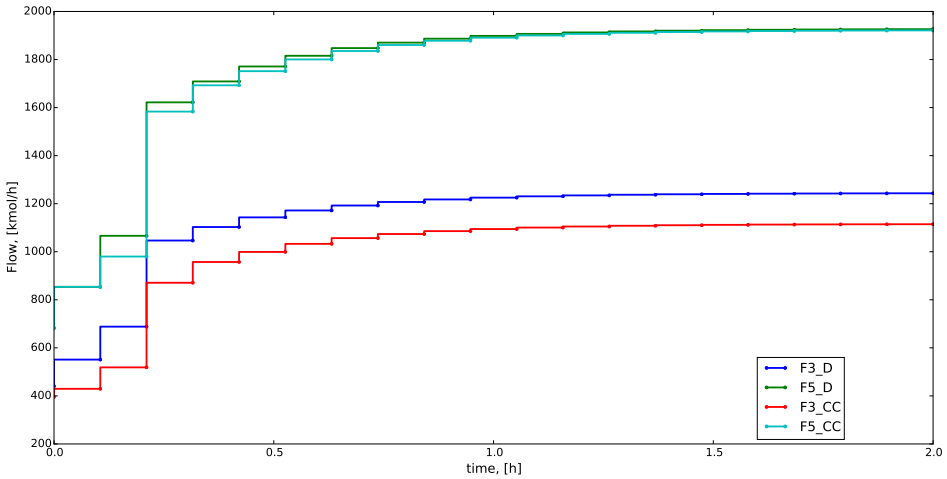

Figure 4.5: F_3 for the determinstic optimisation, noted with D, and for the stochastic optimisation, noted CC, compared with F_5.

For testing the accuracy of the stochastic optimisation, Monte Carlo simulations were done. The number of outcomes in each discretization point that broke the original lower bounds was counted and divided on the number of simulations. The results are shown in Figure 4.6. This is the same method for testing that was done for finding the number of constraint violations for the method 1 and method 2.

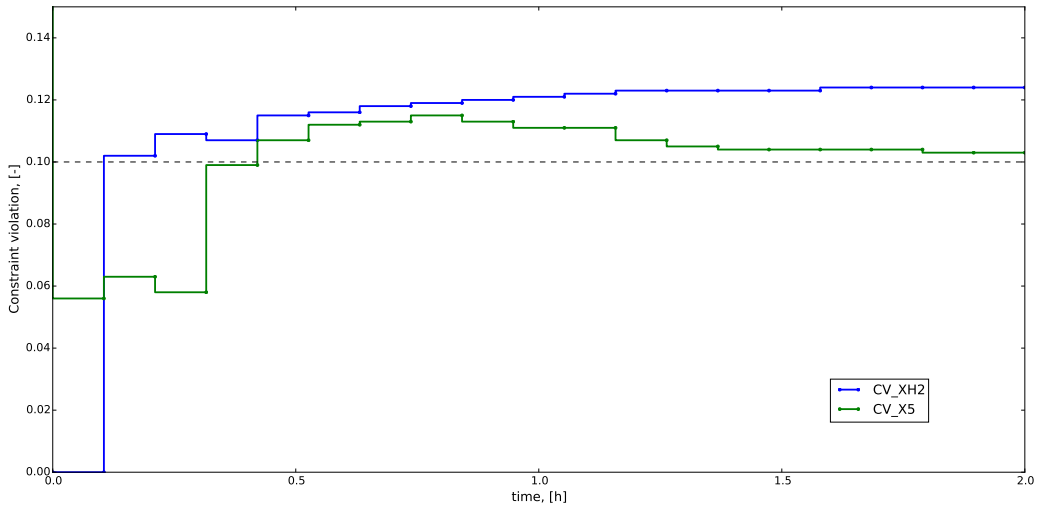

Figure 4.6: The number of constraint violations for X_{H2} and X_5 divided on the number of Monte Carlo simulations. The number of Monte Carlo simulations was 1000.

The Figure 4.6 it can be seen that the profile are similar with the ones given in Figure 4.3. For X_{H2} it is the same as the back-off without the use of difference. For X_5 it is similar with the one that uses the difference in the back-off calculation. This is expected as this is the method that was used for calculating the back-off for the two variables. One can see that the number of violations is larger for X_{H2} than X_3. These is because the variance is much larger for X_{H2} than X_5. A normal fitting of the outcome for X_{H2} is effected more

by the outlying areas. The distribution of X_{H2} has a smaller accuracy then for X_5. This can be fixed by running more simulations, as one get more outcomes to model the normal distribution to. The problem with this approach is that it would increase computational time. Another problem with this approach is that one can see from Figure 4.1 that the histograms are not completely symmetric. This is best shown by point 10 and 17, as one can see that the histogram has a larger tail on the lower value side. So even with increasing the number of simulations the normal distribution would not give a perfect fit of the outcomes. One possible solution is to use different weights when fitting the normal distribution curve to different regions. One could emphasise the areas one is interested in. Since one wants to have a maximum of 10% violations of the lower bound, one would have larger weights here than the other regions. This method would give a better fit for the region that are of interest. The problem are finding good weighting factors. One could add a correction factor to adjust the accuracy of the back-off. The correction factor would be adjusted and new back-offs would be calculated. Then the accuracy would be tested and adjustments on the correction factor would be made depending if one has to many of to few constraint violations. This method would be repeated until convergence is achieved. But as this also would increased the computational time. From Figure 4.6 one can see that the violations for X_{H2} except for the start are all over 10%. One could change α to a higher value for correcting it. This method is not so accurate but are easy and would not increase the computational time. But setting a too high value of α would lead to economical loss, so one would have to be careful when increasing α.

4.3 Model Predictive Controller

The stochastic optimisation is than used for finding optimal set-points for the MPC. The optimisation are done offline and online. Where the online optimisation re-optimise for each discretization point, while the offline optimisation only optimise ones in the start. The control sequence using the two different methods are shown in Figure 4.7.

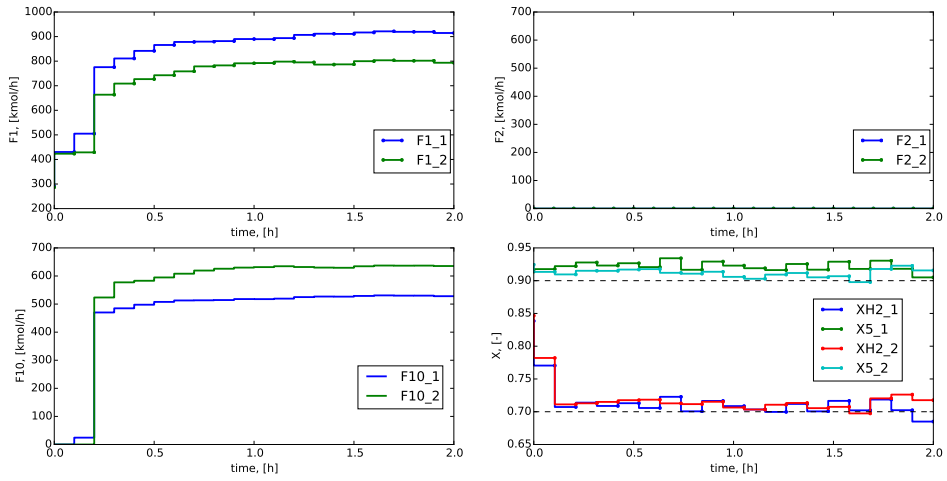

Figure 4.7: The control sequence with 20 steps with use of MPC. With offline optimisation noted with 1 and the online optimisation noted with 2. The state of X_{H2} and X_5 for each discretization point are also shown, with the same notation.

From the plot of X_5 in Figure 4.7 that the value lies closer to the optimal with online

optimisation than with offline. This is expected, as one re-optimise with current state for each discretization point with online optimisation. So one can reset the set-points, given the knowledge of which state the system is in. For X_{H2} there is no significant difference on how close the values lies to the optimum for the two methods. But for offline optimisation the values varies more for each step, this is also true for X_5. The reason is that with offline optimisation one do not update for the current state of the system. So depending on the difference between the outcome of the stochastic variables between two steps the state will also vary the same. But as for online optimisation the variation will be only from the outcome of one step to the next one. The reason are that one take measurement of the current state and make adjust to the set-point for that outcome.

For the control variables the online optimisation have lower consumption of F_1, then for offline optimisation. As one have re-optimisation in each point the variance is only for one step. While for offline optimisation one need to consider the variance for all of the steps. So offline line optimisation have more uncertainties in F_3 so it uses less. So the probability becomes the same as online optimisation. F_2 are zero for both. While F_{10} are higher for the online optimisation. So one have less recycling of hydrogen with online optimisation. This is expected as one have updates on the state through measurements, so one can recycle less with the same uncertainty. For the economical prospective as the consumption of F_1 are less, it is better to use online optimisation.

The problem with online optimisation; is the time it takes. As one have to re-optimise for each step, then run Monte Carlo simulation to find the probability distributions and calculate a new back-offs. This takes time, especially the Monte Carlo simulations. With 1000 simulation and 20 discretization points it is 20 000 calculations for each optimisation. So for offline optimisation the average time was 4 minutes and 25 seconds while it was 35 min for online optimisation. Because of this the online optimisation would not be practical for control.

In the calculations it is the Monte Carlo simulations that take the most amount of time. So in an effort to reduce computational time for the online optimisation, the number of simulations was reduced to 100. This would lead to a less accurate probability distribution of X_{H2} and X_5. But since one takes measurements in each discertization point one do not need as accurate probability distribution. The online optimisation are carried out with 100 Monte Carlo simulations and compared with the offline optimisation in Figure4.8.

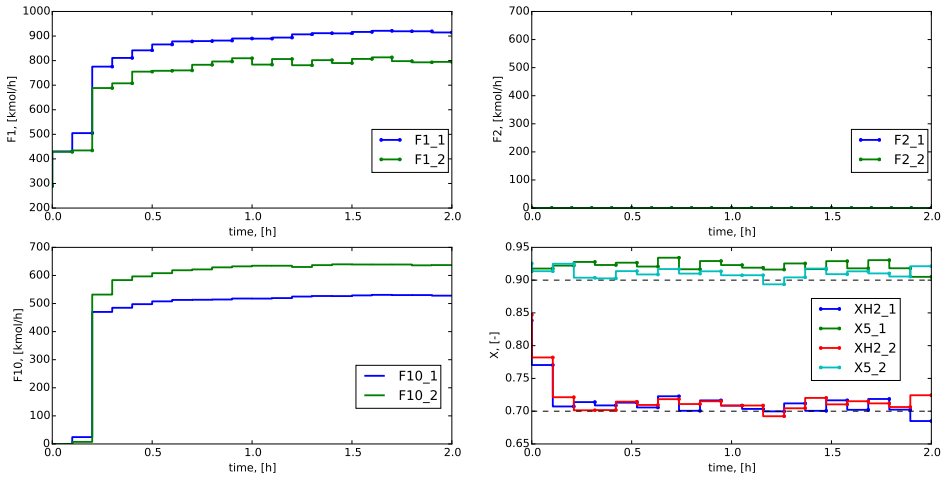

Figure 4.8: The control sequence with 20 steps with use of MPC. With offline optimisation noted with 1 and the online optimisation noted with 2. Where online optimisation are carried out with 100 Monte Carlo simulations instead of 1000. The state of X_{H2} and X_5 for each discretization point are also shown, with the same notation.

From the Figure 4.8 one can see similar tendency as for when had 1000 Monte Carlo simulations. But the are larger variance for the online optimisation. This is expected as one get less accurate model of the probability distribution. But one still get less consumption of F_1 compared with offline. So from an economical prospective it is favourable to offline. But is still takes more time then offline, with 7 minutes and 50 seconds. This is still a to large time, but with decomposing the problem and parallel solve it the time could be reduced additionally. There are also other ways of reducing the computational time. One can use Markov chain Monte carlo simulations to find the probability distributions [Wang 2005],[Martí et al. 2015].

The results of the three different methods are presented in the Table below:

Optimisation type	# Monte Carlo	time [s]	Cost [euro]
Offline	1000	245	1522
Online	1000	2100	1329
Online	100	470	1346

Table 4.1: The type of optimisation used, with number of Monte Carlo simulations and the objective function value.

One can see that offline optimisation gives larges cost, while there are small difference between running 1000 and 100 Monte Carlo simulations. So online optimisation are the best economical alternative, but have problems with computational time. This is especially true when one run 1000 simulations as it almost takes 10 times as long time as offline optimisation.

5 Conclussion

Stochastic optimisation can be used to optimise system with uncertainties. Where one can find an optimum that is feasible independently of the outcome of the stochastic variables. This is shown in Section 2 where 4 different methods are applied to simple stochastic optimisation examples. Also a case study of an hydrodesulfurization unit are solved sarcastically for illustrating how a real world problem can be solved with use of stochastic optimisation.

From the results presented in Section 4 it can been seen that Monte Carlo simulation can be used for finding the probability distribution of the chance constraints. But the assumption of normal distribution are not as good fit for X_{H2}, as it have a long lower tail. Since the probability distribution does not fit so well, the number of constraint violations are higher then the specified confidence level. This could be fixed with a better estimation of the probability distribution. One could use weighted fitting for better fit of the region of interest. For X_5 the normal distribution gives an accurate fit.

When the back-off was calculated, the number of constraint violations was initially very high for X_5. The reason was that the mean value of the probability distribution of X_5 was below the lower bound of X_5. A modified back-off calculation method was tested, where the difference between the mean value of the probability distribution and lower bound was added to Equation (3.11). This gave a much lower number of constraint violations for X_5. But it gave more violations for X_{H2}, so it was only implemented for X_5. But for further work one should add a correction factor to the back-off calculations. Then use an iterative process to converge the back-off such that one gets the wanted probability.

Two methods of optimisation was used fro the MPC, online and offline optimisation. Offline optimisation had a higher cost than online optimisation. The computational time for online optimisation was too high, so the number of Monte Carlo simulations was reduced to 100. This reduced the computational time significantly. It gave a higher cost but was still below the cost of offline optimisation.

Implementation of weighted fitting of the outcome from the Monte Carlo, to get a better approximation of the probability distribution. Are something that would be interesting to look at in further work. Also how one could decrease the computational time of even further, with use of other methods for approximate the probability distribution, rather then Monte Carlo simulation. Also run a comparison of the different method explained in Section 2 for the case study. With comparing of computational time and reliability for the different methods. This is all aspects that would have been interesting to look into, and do further work on.

References

Van Hessem, D. (2004). *Stochastic inequality constrained closed-loop model predictive control*. ISBN: 904072489X.

Feedback, Lectures O N (2013). "Lectures on". In: DOI: 10.1007/978-1-4684-1806-4.

Bertsimas, Dimitris, David B. Brown, and Constantine Caramanis (2011). "Theory and Applications of Robust Optimization". In: *SIAM Review* 53.3, pp. 464–501. ISSN: 0036-1445. DOI: 10.1137/080734510. arXiv: 1010.5445. URL: http://books.google.com/books?hl=en{\&}lr={\&}id=DttjR7IpjUEC{\&}oi=fnd{\&}pg=PR9{\&}dq=Robust+Optimization{\&}ots=W463fBWlR-{\&}sig=N1DRhYF4NIE1XxZRwGsmZXy-0gU.

Li P.; Arellano-Garcia, H.; Wozny G (2006). "Chance constrained programming approach to process optimization under uncertainty". In: *Computer Aided Chemical Engineering* 21, pp. 1245–1250. ISSN: 0098-1354. DOI: http://dx.doi.org/10.1016/j.compchemeng.2007.05.009.

Wendt, Moritz, Pu Li, and Günter Wozny (2002). "Nonlinear Chance-Constrained Process Optimization under Uncertainty". In: *Industrial & Engineering Chemistry Research* 41.15, pp. 3621–3629. ISSN: 0888-5885. DOI: 10.1021/ie010649s. URL: http://dx.doi.org/10.1021/ie010649s.

Sobol, Ilya M. (1994). *A Primer for the Monte Carlo Method*. CRC Press, p. 128.

Zhao, Zinan et al. (2015). "A Novel Approach to Chance Constrained Optimal Control Problems". In: pp. 5611–5616.

Navia, D. et al. (2014). "A comparison between two methods of stochastic optimization for a dynamic hydrogen consuming plant". In: *Computers & Chemical Engineering* 63, pp. 219–233. ISSN: 00981354. URL: http://www.sciencedirect.com/science/article/pii/S0098135414000362.

Robinson, Stephen M (2006). *Springer Series in Operations Research and Financial Engineering*. ISBN: 9780387303031.

Kall, Peter and Janos Mayer (2005). *Stochastic LP*. ISBN: 0387233857.

Vlerk, M H van der (1995). "Stochastic programming with integer recourse". In: pp. 1–24.

Caroe, C C and R Schultz (1999). "Dual decomposition in stochastic integer programming". In: *Operations Research Letters* 24, p. 37. ISSN: 01676377. DOI: 10.1016/S0167-6377(98)00050-9.

Prékopa, András (1995). *Stochastic Programming*, p. 599. ISBN: 9789048145522.

Sen, S. and J. L. Higle (1999). "An Introductory Tutorial on Stochastic Linear Programming Models". In: *Interfaces* 29.2, pp. 33–61. ISSN: 0092-2102. DOI: 10.1287/inte.29.2.33.

Prékopa, András (2003). "Stochastic Programming". In: *Handbooks in Operations Research and Management Science* 10, pp. 267–351. ISSN: 09270507. DOI: 10.1016/S0927-0507(03)10005-9. URL: http://www.sciencedirect.com/science/article/pii/S0927050703100059.

Martí, Rubén et al. (2015). "Improving scenario decomposition algorithms for robust nonlinear model predictive control". In: *Computers & Chemical Engineering* 79, pp. 30–45. ISSN: 00981354. URL: http://www.sciencedirect.com/science/article/pii/S0098135415001258.

Arnold, T (2013). "A mixed-integer stochastic nonlinear optimization problem with joint probabilistic constraints ". In: pp. 1–14.

Kibzun, Andrey I. and Kan Yuri S. (1996). *Stochastic Programming Problems*. John Wiley & Sons Ltd., p. 301. ISBN: 0471958158.

R, Courant and Hilbet D. (2008). *Methods of Mathematical Physics*. Hoboken New York: Wiley. ISBN: 471-50447-5.

Hong, L. J., Y. Yang, and L. Zhang (2011). "Sequential Convex Approximations to Joint Chance Constrained Programs: A Monte Carlo Approach". In: *Operations Research* 59.3, pp. 617–630. ISSN: 0030-364X. DOI: 10.1287/opre.1100.0910.

Andersson, Joel (2013). *Faculty of Engineering Science A General-Purpose Software Framework for Dynamic Optimization*. October. ISBN: 9789460187506.

Foss, Bjarne and TAN Heirung (2013). "Merging Optimization and Control". In: pp. 1–57. URL: http://www.itk.ntnu.no/fag/fordypning/TK16-filer/Samling1{_}MPCnotat.pdf.

Wang, J.S.. Kendall W. S. Liang F. and (2005). *Markov Chain Monte Carlo, Volume 7*. World Scientific Publishing Co., p. 239.

Rao, Anil V. (2009). "A survey of numerical methods for optimal control". In: *Advances in the Astronautical Sciences* 135.1, pp. 497–528. URL: http://vdol.mae.ufl.edu/ConferencePublications/trajectorySurveyAAS.pdf.

A Parameters, Initial Conditions and Constraints

Parameter	Value	Unit	Parameter	Value	Unit
M	20	-	TF	2.0	h
R	0.08314	$\frac{m^3 bar}{Kkmol}$	T	623.15	K
V	100	m^3	P	68.901	bar
τ	0.3	h	ρ	12.6	$\frac{kmol}{m^3}$
F_{HC}	102	$\frac{m^3}{h}$	Z	1	-
C_{H4}	88.1	$\frac{€}{Mmol}$	C_{H3}	77	$\frac{€}{Mmol}$
X_1	0.991	-	X_2	0.931	-
X_3	0.85	-	$X_{H2}(t_0)$	0.9	-
$F_x^{H2}(t_0)$	682.5	$\frac{kmol}{h}$	F_1^{lb}	0	$\frac{kmol}{h}$
F_1^{ub}	1400	$\frac{kmol}{h}$	F_2^{lb}	0	$\frac{kmol}{h}$
F_2^{ub}	790	$\frac{kmol}{h}$	F_3^{lb}	0	$\frac{kmol}{h}$
F_3^{ub}	5000	$\frac{kmol}{h}$	F_10^{lb}	0	$\frac{kmol}{h}$
F_10^{ub}	1500	$\frac{kmol}{h}$	X_5^{ub}	1	-
X_5^{lb}	0.9	-	X_{H2}^{ub}	1	-
X_{H2}^{lb}	0.7	-			

Table A.1: The parameters, upper/lower bounds and initial conditions used for solving (3.10).

Parameter	Value	Unit	Parameter	Value	Unit
$\mu_{F_{HC}}$	12.6	$\frac{kmol}{m^3}$	μ_{X_3}	0.85	-
$\sigma_{F_{HC}}$	0.4	$\frac{kmol}{m^3}$	σ_{X_3}	0.013	-
$\alpha_{X_{H2}}$	0.9	-	α_{X_5}	0.9	-
$\mu_{\hat{X}_{H2}}$	0.7	-	$\mu_{\hat{X}_5}$	0.9	-
$\sigma_{\hat{X}_{H2}}$	0.03	-	$\sigma_{\hat{X}_5}$	0.01	-

Table A.2: The parameters used for calculating the back-off found from the Monte Carlo simulations. Parameters for the binormal probability distribtuion of F_{HC} and X_3.

B Numerical Methods for Solving Nonlinear Optimisation Problems

Nonlinear optimisation problems are often large problems that can consist of multiple ordinary differential equations (ODE) and differential algebraic equations (DAE). An analytical solution is usually not possible or very complex. Numerical methods are often used when solving nonlinear optimisation problems. Numerical methods approximate the problem then use an iterative process to converge to a solution. In the solving process one often has to use a large number of operations, this is time consuming to do by hand. Since the operations usually only are elementary algebraic and logical operations it can easily be done with the use of a computer.

There are many different ways of solving nonlinear optimisation problems but one can divide them in two main methods, the indirect approach and the direct approach. In an indirect approach one converts the nonlinear optimisation problem into a boundary value problem, and solution is found by solving a system of differential equations. In the direct

approach one approximates the differential equations then implements the approximation into the optimisation problem [Rao 2009].

When using the direct approach there are three major methods, single shooting, multiple shooting and the orthogonal collocation method. The single shooting and multiple shooting method are briefly explained below.

B.1 Direct Single shooting Method

In the direct single shooting method one divides the time interval $[t_0, t_f]$ into equal distance sub-intervals. The parameters are discretized such that they are stepwise constant for each sub-interval. Then use an integrator to solve the ODE/DAE for the states. When solving the ODE/DAE with an integrator the optimisation problem becomes only depended on the parameters. The new optimisation problem is solved for the parameters. The integration and optimisation processes are repeated until one converges to an optimal solution.

The advantages with single shooting method is that it is easy to implement and use, also the optimisation problem has few degrees of freedom. The disadvantages are the can not use knowledge of x for initialisation and it has problems with solving unstable systems.

B.2 Direct Multiple Shooting Method

In the direct multiple shooting methods one first divides the time interval $[t_0, t_f]$ into equal distance sub-intervals. Then one preforms single shooting over each sub-interval. To get a continuous function, constraint are added such that each end point is equal to the start point of the next sub-interval. These constraints are often called compatibility constraint, and are usually given as:

$$x(t_k^+) - x(t_{k+1}^-) = 0, \quad for\ k = 0, ..., M \tag{B.1}$$

Where x is the approximated state, t_k is the time interval k, $^-$ indicates the start the time interval and $^+$ state the end of time interval.

The advantage of using multiple shooting compared with single shooting is that it is capable to treat unstable systems and one can use knowledge of x for initialisation. The disadvantage is the number of state variables and constraints increases with the number of time intervals. Because of this multiple shooting makes the optimisation problem large.

C Statistics

Here are some general terms and equations from statistics that are used in this report given.

C.1 Normal probability distribution

The normal probability distribution is given as:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{C.1}$$

Where x is the random variable, σ^2 is the variance, μ is the mean value.

If the mean is zero and the variance is one, the normal probability distribution is called the standard normal probability distribution.

The cumulative distribution of the normal probability distribution is given as:

$$f(x)^{-1} = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right) \tag{C.2}$$

Where $f()^{-1}$ is the cumulative distribution, x is the random variable and erf is the error function.

C.2 Multivariate Normal Probability Distribution

The multivariate normal probability distribution is a normal probability distribution, but instead of one random variable there are k variables, $k \geq 2$. The multivariate normal probability distribution is given as:

$$f_{\mathbf{x}}(x_1, ..., x_k) = \frac{1}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{C.3}$$

Where $\boldsymbol{\Sigma}$ is the covariance matrix and $|\boldsymbol{\Sigma}|$ is the determinate of the covariance matrix.

For a case where there is only two variables, $k = 2$, the covariance matrix is given as:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{x_1}^2 & \rho\sigma_{x_1}\sigma_{x_2} \\ \rho\sigma_{x_1}\sigma_{x_2} & \sigma_{x_2}^2 \end{bmatrix} \tag{C.4}$$

Where ρ is the correlation coefficient.

D Back-off

Back-off is the difference between the optimal set-point and the set-point used in control. In control one does not want set the set-point at the optimal set-point. The reason is that there are uncertainties in control dynamics and measurements. If one does not have large enough back-off this can lead to constraint violations.

For chance constraints the back-off is calculated such that the constraint holds with a probability of some given value α or higher. There are two kinds of chance constraints, individual chance constraints and joint chance constraint. Individual chance constraint are chance constraints where each individual chance constraint needs to hold by some given α. Joint chance constraint are chance constraint where the combined chance constraints need to hold by some given α.

D.1 Individual Chance Constraint

When one has individual chance constraints the back-off can be calculated separately for each chance constraint. The derivation of how to find the back-off is given here. The derivation is taken from Van Hessem 2004.

The chance constraint are given as:

$$\mathbb{P}(h_j^T \boldsymbol{x}(\xi) \leq g_j) \geq \alpha \tag{D.1}$$

Where $\boldsymbol{x}(\xi)$ is random variable, h_j^T is some parameter vector and g_j is the constraint value.

Let us suppose that $\alpha \geq \frac{1}{2}$, this is done such that the constraint is a second-order cone constraint. Then define the probability function as:

$$p(\xi) = h_j^T \boldsymbol{x}(\xi) \tag{D.2}$$

Where $p(\xi)$ is a normal probability function with mean μ_x and variance σ_x^2 the variance can be written as:

$$\sigma_x^2 = h_j^T Z h_j \tag{D.3}$$

Where Z is the covariance matrix for x.

Lets first transform the normal probability function to a standard normal probability function, this is done to make the computation easier. The standard normal probability function for $p(\xi)$ is given as:

$$p_n(\xi) = \frac{p(\xi) - \mu_x}{\sigma_x} \tag{D.4}$$

Where σ_x is the variance of x, and $p_n(\xi)$ is the standard normal probability function with mean zero and variance one.

Where the constraint can now be written as:

$$F_G\left(\frac{g_j - \mu_x}{\sigma_x}\right) \geq \alpha \tag{D.5}$$

The constraint can be written as:

$$\hat{p} + F_G^{-1}(\alpha)\sqrt{h_j^T Z h_j} \leq g_j \tag{D.6}$$

Where F_G^{-1} is the cumulative distribution function.

D.2 Joint Chance Constraint

For joint chance constraint are when multiple chance constraints need to some given level of certainty α. Joint chance constraint is often written as:

$$\mathbb{P}(h_j^T \boldsymbol{x}(\xi) \leq g_j, \quad \forall j) \geq \alpha \tag{D.7}$$

Because of the constraints need to simultaneous hold for some certainty value α, one need to use multivariate normal probability distribution for the probability distribution of the constraint. The multivariate normal probability distribution are highly nonlinear function, and finding a solution of this probability distribution that gives confidence level of α is highly complex problem. Because of this derivation of joint chance constraints are not include in this report [Van Hessem 2004],[Wendt, Li, and Wozny 2002].

E MATLAB

In this section all of the MATLAB codes used in the Section 2 are given.

E.1 Simple Recourse Model Example

Here are the MATLAB code used for solving (2.13) given.

```
% Simple Recourse Model

% Looking at an example given in [Sen, Higle, 1999]
% Min = -x_2
% s.t.  x_1 + x_2 + x_3 = 2
%       -a*x_1 + a*x_2 + x_4 = 2
%       -1 =< x_1 => 1
%        x_j => 0, j = 2,3,4.

% Where a_21 and a_22 are not know with certainty, and the joint
% distribution is given as: {(1,3/4) with probability 0.5
%                           {(-3,5/4) with probability 0.5

% Solution
g = 5; %Penalty cost parameter
f = [0 -1 0 0 g/2 g/2 g/2 g/2]; % cost function

Aeq = [1 1 1 0 0 0 0 0;
    1 3/4 0 1 1 -1 0 0;
    -3 5/4 0 1 0 0 1 -1]; % Constraint Matrix

beq = [2;
    2;
    2]; % Constraint value vector

A = zeros(1,8); % Inequality constraint matrix

b = zeros(1,1); % Inequality value vector

LB = [-1;0;0;0;0;0;0;0]; % Lower bound vector

```

```
UB = [1;Inf;Inf;Inf;Inf;Inf;Inf;Inf]; % upper bound vector

% Solve the problem stated above.
[x,fval,EF] = linprog(f,A,b,Aeq,beq,LB,UB);

% EF is the existflag and if it is 1, the function linprog found a feasible
% soultion to the problem.
% Check if the solution is feasible
if EF > 0
    fprintf('Feasbile solution \n ');
else
    fprintf('Unfeasible solution \n ');
end

% Prints results
string = ['The solution is [%1.4f %1.4f %1.4f %1.4f %1.4f %1.4f %1.4f ...
    %1.4f] \n ' ...
'the cost function is %2.4f \n '];

fprintf(string, x(:,1),fval );
```

E.2 General Recourse Model, Example

Here are the MATLAB code used for solving (2.17) given.

```
% General Recourse Model

% Same example as simple recourxe model.
% With decision of x_2,x_3 and x_4 do not have to be made before stage 2.

% Solution
f = [0 -0.5 0 0 -0.5 0 0]; %  cost function
Aeq = [1 1 1 0 0 0 0 ;
    1 3/4 0 1 0 0 0 ;
    1 0 0 0 1 1 0;
    -3 0 0 0 5/4 0 1]; % Equallity constraint matrix
beq = [2;
    2;
    2;
    2]; % Constraint value vector
A = [0 0 0 0 0 0 0]; % Inequallity constraint matrix
b = [0]; % Constraint value vector
lb = [-1; 0; 0; 0; 0; 0; 0]; % Lower bound vector

ub = [1; Inf; Inf; Inf; Inf; Inf; Inf]; % upper bound vector

% Calculation

[x, fval, EF] = linprog(f,A,b,Aeq,beq,lb,ub);

% EF is the existflag and if it is 1, the function linprog found a feasible
% soultion to the problem.
% Check if the solution is feasible
if EF > 0
    fprintf('Feasbile solution \n ');
else
    fprintf('Unfeasible solution \n ');
end

```

```
% Prints results
string = ['The solution is [%1.4f %1.4f %1.4f %1.4f %1.4f %1.4f %1.4f] \n ' ...
'the cost function is %2.4f \n '];

fprintf(string, x(:,1),fval );
```

E.3 Multistage Recourse Model, Example

Here are the MATLAB code used for solving (2.25) given

```
% Multistage recourse model

% Use same example but with x_2 as a third stage variable, and x_3 and x_4
% as second stage variables.

% Formulation of cost function and constraints

f = [0 0 0 0 0 -1/4 -1/4 -1/4 -1/4]; % cost function

A = zeros(1,length(f)); % inequality constraint matrix

b = zeros(1,1); %

Aeq = [1 1 0 0 0 1 0 0 0;
    1 0 1 0 0 3/4 0 0 0;
    1 0 0 1 0 0 1 0 0;
    -3 0 0 0 1 0 5/4 0 0;
    1 1 0 0 0 0 0 1 0;
    1 0 1 0 0 0 0 3/4 0;
    1 0 0 1 0 0 0 0 1;
    -3 0 0 0 1 0 0 0 5/4;];

beq = [2 2 2 2 2 2 2 2]';

lb = [-1 0 0 0 0 0 0 0 0]';
ub = [1 inf inf inf inf inf inf inf inf]';

[x, fval, EF] = linprog(f,A,b,Aeq,beq,lb,ub);

% EF is the existflag and if it is 1, the function linprog found a feasible
% soultion to the problem.
% Check if the solution is feasible
if EF > 0
    fprintf('Feasbile solution \n ');
else
    fprintf('Unfeasible solution \n ');
end

% Prints results
string = ['The solution is [%1.4f %1.4f %1.4f %1.4f %1.4f %1.4f %1.4f \n' ...
   '%1.4f %1.4f ] the cost function is %2.4f \n '];

fprintf(string, x(:,1),fval );
```

E.4 Robust Optimisation, Example

Here are the MATLAB code used for solving (2.21) given.

```
% Robust optimisation

f = [0 -0.5 0 0 -0.5 0 0]; % linear cost function

H = [0 0 0 0 0 0 0;
    0 1/8 0 0 0 0 0;
    0 0 0 0 0 0 0;
    0 0 0 0 0 0 0;
    0 0 0 0 1/8 0 0;
    0 0 0 0 0 0 0;
    0 0 0 0 0 0 0]; % Quadratic cost function
H = 2*H; % need to recalculate H since quadprog uses 1/2 times H in the
% formulation of the quadratic cost function

Aeq = [1 1 1 0 0 0 0 ;
    1 3/4 0 1 0 0 0 ;
    1 0 0 0 1 1 0;
    -3 0 0 0 5/4 0 1]; % Equallity constraint matrix
beq = [2;
    2;
    2;
    2]; % Constraint value vector
A = [0 0 0 0 0 0 0]; % Inequallity constraint matrix
b = [0]; % Constraint value vector
lb = [-1; 0; 0; 0; 0; 0; 0]; % Lower bound vector

ub = [1; Inf; Inf; Inf; Inf; Inf; Inf]; % upper bound vector

[x, fval, EF]=quadprog(H,f,A,b,Aeq,beq,lb,ub);

% EF is the existflag and if it is 1, the function linprog found a feasible
% soultion to the problem.
% Check if the solution is feasible
if EF > 0
    fprintf('Feasbile solution \n ');
else
    fprintf('Unfeasible solution \n ');
end

% Prints results
string = ['The solution is [%1.4f %1.4f %1.4f %1.4f %1.4f %1.4f %1.4f] \n ' ...
'the cost function is %2.4f \n '];

fprintf(string, x(:,1),fval );
```

E.5 Chance Constraint, Example

Here are the MATLAB code used for solving (2.48) given.

```
% Probabilistic constraint

% Example taken from [Kizbun, Kan 1996] "Reserving Air Tickets"



% Parameters

c = [300 30 1000 280 250];
n  =[350 350];
```

```
11
12 mu = 1/20;
13 alpha = 0.99;
14
15 % Calculations
16 F_b = icdf('Exponential',alpha,mu);
17
18 f_a = (1-exp(-20*F_b))/(1-exp(-20)) - alpha;
19
20 a_b = log(1-((1-exp(-20*icdf('Exponential',alpha,mu)))/(1-exp(-20)) ...
21      -alpha)*(1-exp(-20)))/-20;
22
23 f = [-c(1) c(5) 1];
24
25 A = [0 1 0;
26     c(4)*a_b 0 -1;
27     (c(4)*a_b+c(2)*(1-a_b)) 0 -1;
28     (c(4)*a_b+c(3)*(1-a_b)) -(c(3)-c(2)) -1;
29     c(4)*F_b 0 -1;
30     (c(4)*F_b+c(2)*(1-F_b)) 0 -1;
31     (c(4)*F_b+c(3)*(1-F_b)) -(c(3)-c(2)) -1];
32
33 b = [n(2) 0 c(2)*n(1) c(3)*n(1) 0 c(2)*n(1) c(3)*n(1)]';
34
35 lb = [0 0 0]';
36 ub = [inf inf inf]';
37
38 [x, fval, EF] = linprog(f,A,b,[],[],lb,ub);
```

F Python

In this Section the code used for solving the stochastic optimisation problem defined in Section 3 is given. Also the code used for running the MPC simulation and plotting the results of the MPC simulations is given here as well.

F.1 Stochastic Optimisation, Python code

Here are the code used to solve the stochastic optimisation problem defined in Section 3

```
1  from casadi import *
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from scipy.stats import norm
5  # from math import sqrt
6  from scipy.linalg import sqrtm
7  import time
8
9  start = time.time()
10 # Define Time periods and end time
11
12 N = 20 # Control discretization
13 TF = 2.0 # End time
14
15 # Parameters
16 R = 0.08314 # Gas constant [m^3 bar/K kmol]
```

```
17 T = 623.15 # Temperature in reactor [K]
18 V = 100 # Volume of reactor [m^3]
19 P = 68*1.01325 # Pressure in reactor [bar]
20 tau = 0.3 # Time constant [h]
21 rho = 12.6 # Hydrogen consumption rate in the reactor [kmol/m^3]
22 FHC = 102 # Flow of hydrocarbons into the reactor [m^3/h]
23 Z1 = 1.0 # Compressibility factor ideal gas [-]
24
25 # CH4  = 88.1 # Cost of stream F2 [euros/Mmol]
26 # CH3  = 77 # Cost of stream F1 [euros/Mmol]
27 CH4  = 0.0881 # Cost of stream F2 [euros/kmol]
28 CH3  = 0.077 # Cost of stream F1 [euros/kmol]
29
30 X1 = 0.991 # molefraction of hydrogen in stream 1 [-]
31 X2 = 0.931 # molefraction of hydrogen in stream 2 [-]
32 X3 = 0.85 # molefraction of hydrogen in stream 3 [-]
33
34 #Iinitial and end transition points
35 XH2_init = 0.9 # Initial hydrogen fraction inside the reactor [-]
36 FXH2_init = 682.5 # Initial feed of hydrocarbons [kmol/h]
37
38 F1_init  = 340.0 # Initial flow in stream 1 [kmol/h]
39 F2_init  = 0.0 # Initial flow in stream 2 [kmol/h]
40 F3_init  = 0.0 # Initial flow in stream 3 [kmol/h]
41 F10_init = 0.0 # Initial flow in stream 10 [kmol/h]
42 F5_init  = F1_init + F2_init + F3_init # Initial flow in stream 5 [kmol/h]
43 X5_init  = (F1_init*X1+F2_init*X2+F3_init*X3)/(F1_init+F2_init+F3_init) # ...
       Initial hydrogen fraction in stream 5 [-]
44
45 # Constraints
46 F1_ub  = 1400.0
47 F1_lb  = 0.0
48 F2_ub  = 790.0
49 F2_lb  = 0.0
50 F3_ub  = 5000.0
51 F3_lb  = 0.0
52 F10_ub = 1500.0
53 F10_lb = 0.0
54
55 F5_lb  = 0.0
56 F5_ub  = F1_ub+F2_ub+F3_ub
57 X5_ub  = 1.0
58 X5_lb  = 0.9
59 XH2_ub = 1.0
60 XH2_lb = 0.7
61
62 """ Solving the deterministic optimisation problem for a ...
       hydrodesulphurisation part of plant.
63 Where the objective is to minimize the cost of hydrogen, while keeping the ...
       hydrogen fraction in
64 the reactor, XH2, above 0.7. If the fraction goes below 0.7, the catalyst ...
       in the reactor will
65 become damages. The hydrogen fraction are controlled by three inlet stream.
66 """
67 # Declare variables
68 x = SX.sym("x",2) # Differential states XH2, FxH2,
69 u = SX.sym("u",3) # Control variables F1, F2, F10
70 z = SX.sym("z",3) # Algebraic states  F5, X5, F3
71 t = SX.sym("t") # time
72
73 # Number of state and control variables
74 nx = x.size1() # number of state variables
```

```
nu = u.size1() # number of control variables
nz = z.size1() # Number of algebraic state variables
nvar = nx+nu+nz # Number of variables

# Differential equations
f_x = vertcat( [ ( ( Z1*R*T )/( V*P ) )*( z[0]*z[1] - u[2]*x[0] - x[1] ), ( ...
    1/(tau) )*( FHC*rho - x[1] ) ] )

# Algebraic equations
f_z = vertcat( [ u[0] + u[1] + z[2] - z[0] , u[0]*X1 + u[1]*X2 + z[2]*X3 - ...
    z[0]*z[1], z[0] - u[2] - x[1] ] )

# Objective function
f_u = vertcat( [ CH4*X1*u[0] + CH3*X2*u[1] ])

# Define the dae system to be called by the integrator
dae = SXFunction("dae", daeIn( x=x, z=z, p=u, t=t ), daeOut( ode=f_x, ...
    alg=f_z, quad=f_u) )

# Integrator for solving the Dae system
opts = {"tf":TF/N} # interval length
# opts["tf"] = TF/N
# opts["linear_solver"] = "csparse"
# opts["linear_solver_type"] = "user_defined"
opts["abstol"] = 1e-2 # tolerance
opts["reltol"] = 1e-2 # tolerance
I = Integrator("I","idas", dae,opts)

# Variable bounds and initial guess
nv = nu*N + nx*(1 + N) + nz*(N + 1)
v = MX.sym("v",nv)

# Get the differential states for each shooting interval
xk = [v[nvar*k:nvar*k+nx] for k in range(N+1)]

# Get the algebraic states for each shooting interval
zk = [v[nvar*k+nx:nvar*k+nx+nz] for k in range(N+1)]

# Get the control for each shooting interval
uk = [v[nvar*k+nx+nz:nvar*k+nx+nz+nu] for k in range(N)]

# Variable bounds
vmin = -inf*np.ones(nv)
vmax =  inf*np.ones(nv)

# State bounds
vmin[0::nvar] = XH2_lb
vmax[0::nvar] = XH2_ub
vmin[1::nvar] = 0
vmin[2::nvar] = F5_lb
vmax[2::nvar] = F5_ub
vmin[3::nvar] = X5_lb
vmax[3::nvar] = X5_ub
vmin[4::nvar] = F3_lb
vmax[4::nvar] = F3_ub

# Control bounds
vmin[5::nvar] = F1_lb
vmax[5::nvar] = F1_ub
vmin[6::nvar] = F2_lb
vmax[6::nvar] = F2_ub
vmin[7::nvar] = F10_lb
```

```
vmax[7::nvar] = F10_ub

# Initial solution guess (Values found from steady state solution)
v0 = np.zeros(nv)
v0[0::nvar] = 0.7
v0[1::nvar] = 1285.2
v0[2::nvar] = 1927.8
v0[3::nvar] = 0.9
v0[4::nvar] = 1244.18
v0[5::nvar] = 683.617
v0[6::nvar] = 0.0
v0[7::nvar] = 642.6

# Initial condition
vmin[0] = vmax[0] = v0[0] = XH2_init
vmin[1] = vmax[1] = v0[1] = FXH2_init
# vmin[2] = vmax[2] = v0[2] = F5_init
# vmin[3] = vmax[3] = v0[3] = X5_init
# vmin[4] = vmax[4] = v0[4] = F3_init

# Constraint function with bounds
g = []; gmin = []; gmax = []
# Objective function
J = 0

# Build up a graph of integrator calls
for k in range(N):
    # Call the integrator
    xf = I( { 'x0':xk[k], 'z0':zk[k], 'p':uk[k] } )["xf"]
    uf = I( { 'x0':xk[k], 'z0':zk[k], 'p':uk[k] } )["qf"]
    # Add contribution to objective
    J += uf
    # Append continuity constraints and algebraic constraints
    g.append( xf[0] - xk[k+1][0] )
    g.append( xf[1] - xk[k+1][1] )
    g.append( zk[k][0] - uk[k][0] - uk[k][1] - zk[k][2] )
    g.append( uk[k][0]*X1 + uk[k][1]*X2 + zk[k][2]*X3 - zk[k][0]*zk[k][1] )
    g.append( zk[k][0] - uk[k][2] - xk[k][1] )

g = vertcat(g)

def MPC(v,J,g,steps,case):
    # Optimisation
    nlp = MXFunction( "nlp",nlpIn(x=v), nlpOut(f=J, g=g) )
    solver = NlpSolver("solver","ipopt", nlp)
    sol = solver({"lbx" : vmin, # State lower bounds
                  "ubx" : vmax, # State upper bounds
                  "x0" : v0, # Initial guess
                  "lbg" : np.zeros(g.shape), # Constraint lower bound
                  "ubg" : np.zeros(g.shape)}) # Constraint upper bound
    # Extract all of the decision variables from the solved nlp
    v_opt = sol["x"]

    ns = 1000
    X_sim,Z_sim = MonteCarlo(v_opt,ns)

    back_off_XH2,back_off_X5 = backoff(X_sim,Z_sim)
    for k in range(0,N+1):
        vmin[k*nvar] = 0.7 + back_off_XH2[k-1]
        vmin[3+nvar*k] = 0.9 + back_off_X5[k-1]

    #   Creater vectors to show the step changes in the control variables
```

```
196     F1_u = []
197     F2_u = []
198     F10_u = []
199
200     XH2_y = [] #
201     X5_y = [] #
202     # XH2_y.append( XH2_init)
203     # X5_y.append( X5_init)
204
205     for i in range(steps):
206
207         # Optimisation
208         nlp = MXFunction( "nlp",nlpIn(x=v), nlpOut(f=J, g=g) )
209         solver = NlpSolver("solver","ipopt", nlp)
210         sol = solver({"lbx" : vmin, # State lower bounds
211                       "ubx" : vmax, # State upper bounds
212                       "x0" : v0, # Initial guess
213                       "lbg" : np.zeros(g.shape), # Constraint lower bound
214                       "ubg" : np.zeros(g.shape)}) # Constraint upper bound
215         # Extract all of the decision variables from the solved nlp
216         v_opt = sol["x"]
217
218         # Define the solved variables
219         XH2 = v_opt[0::nvar]
220         FXH2 = v_opt[1::nvar]
221         F5 = v_opt[2::nvar]
222         X5 = v_opt[3::nvar]
223         F3 = v_opt[4::nvar]
224         F1 = v_opt[5::nvar]
225         F2 = v_opt[6::nvar]
226         F10 = v_opt[7::nvar]
227
228         if case == 2:
229             # Run online optimisation
230             ns = 100 # Number of Monte Carlo simulations
231             X_sim,Z_sim = MonteCarlo(v_opt, ns);
232             back_off_XH2,back_off_X5 = backoff(X_sim,Z_sim);
233
234             for k in range(0,N+1):
235                 vmin[k*nvar] = 0.7 + back_off_XH2[k-1]
236                 vmin[3+nvar*k] = 0.9 + back_off_X5[k-1]
237         X_temp1 = []
238         Z_temp1 = []
239
240         sigma_x3 = 0.013 # Variance for X3
241         sigma_rho = 0.4 # Variance for FXH2
242         mu_x3 = 0.85 # Mean value for X3
243         mu_rho = 12.6 # Mean value for FXH2
244         mean = [mu_x3, mu_rho] # Mean values vector
245         r = -0.5 # correlation coefficient
246         cov = [[sigma_x3**2, r*sigma_rho*sigma_x3], [r*sigma_rho*sigma_x3, ...
                sigma_rho**2]]  # diagonal covariance
247         # Define States and Control variables
248         X = SX.sym("X",2) # Differential states XH2, FxH2,
249         U = SX.sym("U",3) # Control variables F1, F2, F10
250         Z = SX.sym("Z",3) # Algebraic states  F5, X5, F3
251
252         # Differential equations
253         F_x = vertcat( [ ( ( Z1*R*T )/( V*P ) )*( Z[0]*Z[1] - U[2]*X[0] - ...
                X[1] ), ( 1/tau )*( FHC*np.random.multivariate_normal(mean, ...
                cov)[1] - X[1] ) ] )
254
```

```
255         # Algebraic equations
256         F_z = vertcat( [ U[0] + U[1] + Z[2] - Z[0] , U[0]*X1 + U[1]*X2 + ...
                Z[2]*np.random.multivariate_normal(mean, cov)[0] - Z[0]*Z[1], ...
                Z[0] - U[2] - X[1] ] )
257
258         # Objective function
259         F_u = vertcat( [ CH4*X1*U[0] + CH3*X2*U[1] ])
260
261         # Define the dae system to be called by the integrator
262         DAE = SXFunction("DAE", daeIn( x=X, z=Z, p=U, t=t ), daeOut( ...
                ode=F_x, alg=F_z, quad=F_u) )
263
264         # Integrator for solving the Dae system
265         opts = {"tf":TF/N} # interval length
266         I_sim = Integrator("I_sim","idas", DAE)
267         ts = linspace(0,TF,N+1) # Time steps
268         if i == 0:
269             F1_u.append(F1[0])
270             F2_u.append(F2[0])
271             F10_u.append(F10[0])
272
273         # Run the simulation
274         for j in range(N-1):
275             sim = Simulator("sim",I_sim,[ts[j],ts[j+1]] )
276
277             # Input for the simulator
278             if j == 0:
279                 sim.setInput([XH2[0],FXH2[0]],"x0")
280                 sim.setInput([F5[0],X5[0],F3[0]],"z0")
281                 sim.setInput([F1[0],F2[0],F10[0]],"p")
282             else:
283                 sim.setInput(X_temp1[-1],"x0")
284                 sim.setInput(Z_temp1[-1],"z0")
285                 sim.setInput([F1[j],F2[j],F10[j]],"p")
286
287             sim.evaluate()
288             if j == 1:
289                 # Set initial condition for next optimization cycle
290                 vmin[0] = vmax[0] = v0[0] = sim.getOutput("xf")[0,0]
291                 vmin[1] = vmax[1] = v0[1] = sim.getOutput("xf")[1,0]
292
293
294                 F1_u[i+1:i+2] = F1[j]
295                 F2_u[i+1:i+2] = F2[j]
296                 F10_u[i+1:i+2] = F10[j]
297
298                 XH2_y.append( sim.getOutput("xf")[0,0])
299                 X5_y.append( sim.getOutput("zf")[1,0])
300
301             X_temp1.append( sim.getOutput("xf")[:,-1] )
302             Z_temp1.append( sim.getOutput("zf")[:,-1] )
303
304     return XH2_y,X5_y,F1_u,F2_u,F10_u
305
306 def MonteCarlo(v_opt,ns):
307     """This is a function that runs Monte Carlo simulations for the ...
            hydrodesulphurisation
308     plant. The function takes the optimal solution from the nlp solver, and ...
            then simulates
309     ns simulation, with random outcome of X3 and rho. The output of the ...
            function are
```

```
    the endpoints from the simulation, X_end, and the endpoints from each step in
    the simulations, X_sim.
    """
    # Generates random outcomes for X3 and rho
    # ns = 100 # Number of outcomes
    sigma_x3 = 0.013 # Variance for X3
    sigma_rho = 0.4 # Variance for FXH2
    mu_x3 = 0.85 # Mean value for X3
    mu_rho = 12.6 # Mean value for FXH2
    mean = [mu_x3, mu_rho] # Mean values vector
    r = -0.5 # correlation coefficient
    cov = [[sigma_x3**2, r*sigma_rho*sigma_x3], [r*sigma_rho*sigma_x3, sigma_rho**2]]  # diagonal covariance
    # For each outcome of X3 and rho run a simulation to get the outcome of X3 and X5
    X_sim = [] # Make a a matrix to store all differential states for each simulation outcome
    Z_sim = [] # Make algebraic states matrix for each simulation outcome
    X_end = []; # Make a matrix to store differential states end values from each simulation
    Z_end = []; # Make a matrix to store algebraic states end values from each simulation
    ts = linspace(0,TF,N+1) # Time steps
    # Extract the optimal solution to each variable
    XH2 = v_opt[0::nvar]
    FXH2 = v_opt[1::nvar]
    F5  = v_opt[2::nvar]
    X5  = v_opt[3::nvar]
    F3 = v_opt[4::nvar]
    F1 = v_opt[5::nvar]
    F2 = v_opt[6::nvar]
    F10 = v_opt[7::nvar]

    for j in range(ns):
        # Define States and Control variables
        X = SX.sym("X",2) # Differential states XH2, FxH2,
        U = SX.sym("U",3) # Control variables F1, F2, F10
        Z = SX.sym("Z",3) # Algebraic states  F5, X5, F3

        # Differential equations
        F_x = vertcat( [ ( ( Z1*R*T )/( V*P ) )*( Z[0]*Z[1] - U[2]*X[0] - X[1] ), ( 1/tau )*( FHC*np.random.multivariate_normal(mean, cov)[1] - X[1] ) ] )

        # Algebraic equations
        F_z = vertcat( [ U[0] + U[1] + Z[2] - Z[0] , U[0]*X1 + U[1]*X2 + Z[2]*np.random.multivariate_normal(mean, cov)[0] - Z[0]*Z[1], Z[0] - U[2] - X[1] ] )

        # Objective function
        F_u = vertcat( [ CH4*X1*U[0] + CH3*X2*U[1] ])

        # Define the dae system to be called by the integrator
        DAE = SXFunction("DAE", daeIn( x=X, z=Z, p=U, t=t ), daeOut( ode=F_x, alg=F_z, quad=F_u) )

        # Integrator for solving the Dae system
        opts = {"tf":TF/N} # interval length
        I_sim = Integrator("I_sim","idas", DAE)

        # Run simulation for each outcome of the random variable
```

```
361         X_temp = []
362         Z_temp = []
363         for i in range(N):
364             # Run the simulator function that simulates the state variables ...
                    for each
365             # step with a given value of input variables
366             sim = Simulator("sim",I_sim,[ts[i],ts[i+1]] )
367 
368             if i == 0:
369                 sim.setInput([XH2[0],FXH2[0]],"x0")
370                 sim.setInput([F5[0],X5[0],F3[0]],"z0")
371                 sim.setInput([F1[0],F2[0],F10[0]],"p")
372 
373             else:
374                 sim.setInput(X_temp[-1],"x0")
375                 sim.setInput(Z_temp[-1],"z0")
376                 sim.setInput([F1[i],F2[i],F10[i]],"p")
377 
378             sim.evaluate()
379             # Extract the end point for the state variables
380             X_temp.append( sim.getOutput("xf")[:,-1] )
381             Z_temp.append( sim.getOutput("zf")[:,-1] )
382             if i == N-2:
383                 X_end.append( sim.getOutput("xf")[0,-1] )
384                 Z_end.append( sim.getOutput("zf")[1,-1] )
385         Xsim = vertcat(X_temp)
386         Zsim = vertcat(Z_temp)
387 
388         # Take the XH2 and
389         X_sim.append( Xsim[0::nx] )
390         Z_sim.append( Zsim[1::nz]  )
391     # Convert from Dmatrix to np.array
392     Xend = np.hstack(X_end)
393     Zend = np.hstack(Z_end)
394     X_Sim = np.hstack(X_sim)
395     Z_Sim = np.hstack(Z_sim)
396 
397     return X_Sim,Z_Sim
398 def backoff(X_sim,Z_sim):
399 
400     """ This function calculates the back-off for each step, this is done
401     with fitting the outcome from the Monte Carlo simulations to a normal
402     probability distribution, then finding a back-off such that constraint
403     holds with given value, alpha, or higher. This back-off also take into
404     account the difference between the mean value and the lower bound
405     """
406     alpha_x = 0.9
407     alpha_z = 0.9
408     mu_x = np.zeros(N)
409     mu_z = np.zeros(N)
410     sigma_x = np.zeros(N)
411     sigma_z = np.zeros(N)
412     Z = []
413     sqrt_Z = 0
414     back_off_XH2 = []
415     back_off_X5 = []
416 
417     for i in range(N):
418         (mu_x[i], sigma_x[i]) = norm.fit(X_sim[i][:]) # Find the mean and ...
                variance for XH2, if XH2 have normal distribution
419         (mu_z[i], sigma_z[i]) = norm.fit(Z_sim[i][:]) # Find the mean and ...
                variance for X5, if X5 have normal distribution
```

```
420         # Find the covariance matrix Z
421         Z.append(np.cov(X_sim[i],Z_sim[i]))
422 
423         # Calculates the back-off
424         # back_off_XH2.append(  norm.cdf(alpha_x)*( ...
                sqrt(np.cov(X_sim[i][:])*2) ) )
425         # back_off_X5.append(  norm.cdf(alpha_z)*( ...
                sqrt(np.cov(Z_sim[i][:])*2) ) )
426 
427         sqrt_Z = sqrtm(Z[i]) # Square root of the covariance matrix
428         # if  (XH2_lb - mu_x[i]) > 0:
429             # back_off_XH2.append(  XH2_lb - mu_x[i] + ...
                    np.dot([norm.cdf(alpha_x),norm.cdf(alpha_z)],sqrt_Z)[0] )
430         # else:
431         back_off_XH2.append( ...
                np.dot([norm.cdf(alpha_x),norm.cdf(alpha_z)],sqrt_Z)[0] )
432         if  ( X5_lb - mu_z[i]) > 0:
433             back_off_X5.append(  X5_lb - mu_z[i] + ...
                    np.dot([norm.cdf(alpha_x),norm.cdf(alpha_z)],sqrt_Z)[1] )
434         else:
435             back_off_X5.append( ...
                    np.dot([norm.cdf(alpha_x),norm.cdf(alpha_z)],sqrt_Z)[1] )
436 
437     return back_off_XH2,back_off_X5
438 
439 
440     steps = 5
441 case = 2
442 steps = 20
443 XH2_y,X5_y,F1_u,F2_u,F10_u = MPC(v,J,g,steps,case)
444 end = time.time()
445 print end - start
446 plt.show()
447 t = np.linspace(0,TF,steps+1)
448 XH2_Y = vertcat(XH2_y)
449 X5_Y = vertcat(X5_y)
450 F1_U = vertcat(F1_u)
451 F2_U = vertcat(F2_u)
452 F10_U = vertcat(F10_u)
453 print F1_U,"F1_u"
454 print XH2_Y, "XH2_Y"
455 print F1_U.shape, "F1_U.shape", F2_U.shape, "F2_U.shape", F10_U.shape, ...
        "F10_U.shape"
456 if case == 1:
457 
458     # DataOut = np.hstack((XH2_Y,X5_Y))
459     file = open("MPC_plot_case1.txt", "w")
460     file.write("\n".join(map(lambda x: str(x), XH2_Y  )))
461     file.write("\n")
462     file.write("\n".join(map(lambda x: str(x), X5_Y  )))
463     file.write("\n")
464     file.write("\n".join(map(lambda x: str(x), F1_U )))
465     file.write("\n")
466     file.write("\n".join(map(lambda x: str(x), F2_U  )))
467     file.write("\n")
468     file.write("\n".join(map(lambda x: str(x), F10_U  )))
469     file.close()
470 elif case == 2:
471     # DataOut = np.hstack((XH2_Y,X5_Y))
472     file = open("MPC_plot_case2.txt", "w")
473     file.write("\n".join(map(lambda x: str(x), XH2_Y  )))
474     file.write("\n")
```

```
475     file.write("\n".join(map(lambda x: str(x), X5_Y  )))
476     file.write("\n")
477     file.write("\n".join(map(lambda x: str(x), F1_U )))
478     file.write("\n")
479     file.write("\n".join(map(lambda x: str(x), F2_U  )))
480     file.write("\n")
481     file.write("\n".join(map(lambda x: str(x), F10_U  )))
482
483     file.close()
484 else:
485     print "case not defined"
```

F.2 Model Predictive Control, Python code

Here are the code used to simulate the MPC for both optimisation method.

```
1  from casadi import *
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from scipy.stats import norm
5  # from math import sqrt
6  from scipy.linalg import sqrtm
7  import time
8
9  start = time.time()
10 # Define Time periods and end time
11
12 N = 20 # Control discretization
13 TF = 2.0 # End time
14
15 # Parameters
16 R = 0.08314 # Gas constant [m^3 bar/K kmol]
17 T = 623.15 # Temperature in reactor [K]
18 V = 100 # Volume of reactor [m^3]
19 P = 68*1.01325 # Pressure in reactor [bar]
20 tau = 0.3 # Time constant [h]
21 rho = 12.6 # Hydrogen consumption rate in the reactor [kmol/m^3]
22 FHC = 102 # Flow of hydrocarbons into the reactor [m^3/h]
23 Z1 = 1.0 # Compressibility factor ideal gas [-]
24
25 # CH4  = 88.1 # Cost of stream F2 [euros/Mmol]
26 # CH3  = 77 # Cost of stream F1 [euros/Mmol]
27 CH4  = 0.0881 # Cost of stream F2 [euros/kmol]
28 CH3  = 0.077 # Cost of stream F1 [euros/kmol]
29
30 X1 = 0.991 # molefraction of hydrogen in stream 1 [-]
31 X2 = 0.931 # molefraction of hydrogen in stream 2 [-]
32 X3 = 0.85 # molefraction of hydrogen in stream 3 [-]
33
34 #Iinitial and end transition points
35 XH2_init = 0.9 # Initial hydrogen fraction inside the reactor [-]
36 FXH2_init = 682.5 # Initial feed of hydrocarbons [kmol/h]
37
38 F1_init  = 340.0 # Initial flow in stream 1 [kmol/h]
39 F2_init  = 0.0 # Initial flow in stream 2 [kmol/h]
40 F3_init  = 0.0 # Initial flow in stream 3 [kmol/h]
41 F10_init = 0.0 # Initial flow in stream 10 [kmol/h]
42 F5_init  = F1_init + F2_init + F3_init # Initial flow in stream 5 [kmol/h]
```

```
X5_init  = (F1_init*X1+F2_init*X2+F3_init*X3)/(F1_init+F2_init+F3_init) # ...
    Initial hydrogen fraction in stream 5 [-]

# Constraints
F1_ub  = 1400.0
F1_lb  = 0.0
F2_ub  = 790.0
F2_lb  = 0.0
F3_ub  = 5000.0
F3_lb  = 0.0
F10_ub = 1500.0
F10_lb = 0.0

F5_lb  = 0.0
F5_ub  = F1_ub+F2_ub+F3_ub
X5_ub  = 1.0
X5_lb  = 0.9
XH2_ub = 1.0
XH2_lb = 0.7

""" Solving the deterministic optimisation problem for a ...
    hydrodesulphurisation part of plant.
Where the objective is to minimize the cost of hydrogen, while keeping the ...
    hydrogen fraction in
the reactor, XH2, above 0.7. If the fraction goes below 0.7, the catalyst ...
    in the reactor will
become damages. The hydrogen fraction are controlled by three inlet stream.
"""
# Declare variables
x = SX.sym("x",2) # Differential states XH2, FxH2,
u = SX.sym("u",3) # Control variables F1, F2, F10
z = SX.sym("z",3) # Algebraic states  F5, X5, F3
t = SX.sym("t") # time

# Number of state and control variables
nx = x.size1() # number of state variables
nu = u.size1() # number of control variables
nz = z.size1() # Number of algebraic state variables
nvar = nx+nu+nz # Number of variables

# Differential equations
f_x = vertcat( [ ( ( Z1*R*T )/( V*P ) )*( z[0]*z[1] - u[2]*x[0] - x[1] ), ( ...
    1/(tau) )*( FHC*rho - x[1] ) ] )

# Algebraic equations
f_z = vertcat( [ u[0] + u[1] + z[2] - z[0] , u[0]*X1 + u[1]*X2 + z[2]*X3 - ...
    z[0]*z[1], z[0] - u[2] - x[1] ] )

# Objective function
f_u = vertcat( [ CH4*X1*u[0] + CH3*X2*u[1] ])

# Define the dae system to be called by the integrator
dae = SXFunction("dae", daeIn( x=x, z=z, p=u, t=t ), daeOut( ode=f_x, ...
    alg=f_z, quad=f_u) )

# Integrator for solving the Dae system
opts = {"tf":TF/N} # interval length
# opts["tf"] = TF/N
# opts["linear_solver"] = "csparse"
# opts["linear_solver_type"] = "user_defined"
opts["abstol"] = 1e-2 # tolerance
opts["reltol"] = 1e-2 # tolerance
```

```
98  I = Integrator("I","idas", dae,opts)
99
100 # Variable bounds and initial guess
101 nv = nu*N + nx*(1 + N) + nz*(N + 1)
102 v = MX.sym("v",nv)
103
104 # Get the differential states for each shooting interval
105 xk = [v[nvar*k:nvar*k+nx] for k in range(N+1)]
106
107 # Get the algebraic states for each shooting interval
108 zk = [v[nvar*k+nx:nvar*k+nx+nz] for k in range(N+1)]
109
110 # Get the control for each shooting interval
111 uk = [v[nvar*k+nx+nz:nvar*k+nx+nz+nu] for k in range(N)]
112
113 # Variable bounds
114 vmin = -inf*np.ones(nv)
115 vmax =  inf*np.ones(nv)
116
117 # State bounds
118 vmin[0::nvar] = XH2_lb
119 vmax[0::nvar] = XH2_ub
120 vmin[1::nvar] = 0
121 vmin[2::nvar] = F5_lb
122 vmax[2::nvar] = F5_ub
123 vmin[3::nvar] = X5_lb
124 vmax[3::nvar] = X5_ub
125 vmin[4::nvar] = F3_lb
126 vmax[4::nvar] = F3_ub
127
128 # Control bounds
129 vmin[5::nvar] = F1_lb
130 vmax[5::nvar] = F1_ub
131 vmin[6::nvar] = F2_lb
132 vmax[6::nvar] = F2_ub
133 vmin[7::nvar] = F10_lb
134 vmax[7::nvar] = F10_ub
135
136 # Initial solution guess (Values found from steady state solution)
137 v0 = np.zeros(nv)
138 v0[0::nvar] = 0.7
139 v0[1::nvar] = 1285.2
140 v0[2::nvar] = 1927.8
141 v0[3::nvar] = 0.9
142 v0[4::nvar] = 1244.18
143 v0[5::nvar] = 683.617
144 v0[6::nvar] = 0.0
145 v0[7::nvar] = 642.6
146
147 # Initial condition
148 vmin[0] = vmax[0] = v0[0] = XH2_init
149 vmin[1] = vmax[1] = v0[1] = FXH2_init
150 # vmin[2] = vmax[2] = v0[2] = F5_init
151 # vmin[3] = vmax[3] = v0[3] = X5_init
152 # vmin[4] = vmax[4] = v0[4] = F3_init
153
154 # Constraint function with bounds
155 g = []; gmin = []; gmax = []
156 # Objective function
157 J = 0
158
159 # Build up a graph of integrator calls
```

```
for k in range(N):
    # Call the integrator
    xf = I( { 'x0':xk[k], 'z0':zk[k], 'p':uk[k] } )["xf"]
    uf = I( { 'x0':xk[k], 'z0':zk[k], 'p':uk[k] } )["qf"]
    # Add contribution to objective
    J += uf
    # Append continuity constraints and algebraic constraints
    g.append( xf[0] - xk[k+1][0] )
    g.append( xf[1] - xk[k+1][1] )
    g.append( zk[k][0] - uk[k][0] - uk[k][1] - zk[k][2] )
    g.append( uk[k][0]*X1 + uk[k][1]*X2 + zk[k][2]*X3 - zk[k][0]*zk[k][1] )
    g.append( zk[k][0] - uk[k][2] - xk[k][1] )

g = vertcat(g)

def MPC(v,J,g,steps,case):
    # Optimisation
    nlp = MXFunction( "nlp",nlpIn(x=v), nlpOut(f=J, g=g) )
    solver = NlpSolver("solver","ipopt", nlp)
    sol = solver({"lbx" : vmin, # State lower bounds
                  "ubx" : vmax, # State upper bounds
                  "x0" : v0, # Initial guess
                  "lbg" : np.zeros(g.shape), # Constraint lower bound
                  "ubg" : np.zeros(g.shape)}) # Constraint upper bound
    # Extract all of the decision variables from the solved nlp
    v_opt = sol["x"]

    ns = 1000
    X_sim,Z_sim = MonteCarlo(v_opt,ns)

    back_off_XH2,back_off_X5 = backoff(X_sim,Z_sim)
    for k in range(0,N+1):
        vmin[k*nvar] = 0.7 + back_off_XH2[k-1]
        vmin[3+nvar*k] = 0.9 + back_off_X5[k-1]

    #   Creater vectors to show the step changes in the control variables
    F1_u = []
    F2_u = []
    F10_u = []

    XH2_y = [] #
    X5_y = [] #
    # XH2_y.append( XH2_init)
    # X5_y.append( X5_init)

    for i in range(steps):

        # Optimisation
        nlp = MXFunction( "nlp",nlpIn(x=v), nlpOut(f=J, g=g) )
        solver = NlpSolver("solver","ipopt", nlp)
        sol = solver({"lbx" : vmin, # State lower bounds
                      "ubx" : vmax, # State upper bounds
                      "x0" : v0, # Initial guess
                      "lbg" : np.zeros(g.shape), # Constraint lower bound
                      "ubg" : np.zeros(g.shape)}) # Constraint upper bound
        # Extract all of the decision variables from the solved nlp
        v_opt = sol["x"]

        # Define the solved variables
        XH2 = v_opt[0::nvar]
        FXH2 = v_opt[1::nvar]
        F5 = v_opt[2::nvar]
```

```
222         X5 = v_opt[3::nvar]
223         F3 = v_opt[4::nvar]
224         F1 = v_opt[5::nvar]
225         F2 = v_opt[6::nvar]
226         F10 = v_opt[7::nvar]
227 
228         if case == 2:
229             # Run online optimisation
230             ns = 100 # Number of Monte Carlo simulations
231             X_sim,Z_sim = MonteCarlo(v_opt, ns);
232             back_off_XH2,back_off_X5 = backoff(X_sim,Z_sim);
233 
234             for k in range(0,N+1):
235                 vmin[k*nvar] = 0.7 + back_off_XH2[k-1]
236                 vmin[3+nvar*k] = 0.9 + back_off_X5[k-1]
237         X_temp1 = []
238         Z_temp1 = []
239 
240         sigma_x3 = 0.013 # Variance for X3
241         sigma_rho = 0.4 # Variance for FXH2
242         mu_x3 = 0.85 # Mean value for X3
243         mu_rho = 12.6 # Mean value for FXH2
244         mean = [mu_x3, mu_rho] # Mean values vector
245         r = -0.5 # correlation coefficient
246         cov = [[sigma_x3**2, r*sigma_rho*sigma_x3], [r*sigma_rho*sigma_x3, ...
                sigma_rho**2]]  # diagonal covariance
247         # Define States and Control variables
248         X = SX.sym("X",2) # Differential states XH2, FxH2,
249         U = SX.sym("U",3) # Control variables F1, F2, F10
250         Z = SX.sym("Z",3) # Algebraic states  F5, X5, F3
251 
252         # Differential equations
253         F_x = vertcat( [ ( ( Z1*R*T )/( V*P ) )*( Z[0]*Z[1] - U[2]*X[0] - ...
                X[1] ), ( 1/tau )*( FHC*np.random.multivariate_normal(mean, ...
                cov)[1] - X[1] ) ] )
254 
255         # Algebraic equations
256         F_z = vertcat( [ U[0] + U[1] + Z[2] - Z[0] , U[0]*X1 + U[1]*X2 + ...
                Z[2]*np.random.multivariate_normal(mean, cov)[0] - Z[0]*Z[1], ...
                Z[0] - U[2] - X[1] ] )
257 
258         # Objective function
259         F_u = vertcat( [ CH4*X1*U[0] + CH3*X2*U[1] ])
260 
261         # Define the dae system to be called by the integrator
262         DAE = SXFunction("DAE", daeIn( x=X, z=Z, p=U, t=t ), daeOut( ...
                ode=F_x, alg=F_z, quad=F_u) )
263 
264         # Integrator for solving the Dae system
265         opts = {"tf":TF/N} # interval length
266         I_sim = Integrator("I_sim","idas", DAE)
267         ts = linspace(0,TF,N+1) # Time steps
268         if i == 0:
269             F1_u.append(F1[0])
270             F2_u.append(F2[0])
271             F10_u.append(F10[0])
272 
273         # Run the simulation
274         for j in range(N-1):
275             sim = Simulator("sim",I_sim,[ts[j],ts[j+1]] )
276 
277             # Input for the simulator
```

```
278             if j == 0:
279                 sim.setInput([XH2[0],FXH2[0]],"x0")
280                 sim.setInput([F5[0],X5[0],F3[0]],"z0")
281                 sim.setInput([F1[0],F2[0],F10[0]],"p")
282             else:
283                 sim.setInput(X_temp1[-1],"x0")
284                 sim.setInput(Z_temp1[-1],"z0")
285                 sim.setInput([F1[j],F2[j],F10[j]],"p")
286 
287             sim.evaluate()
288             if j == 1:
289                 # Set initial condition for next optimization cycle
290                 vmin[0] = vmax[0] = v0[0] = sim.getOutput("xf")[0,0]
291                 vmin[1] = vmax[1] = v0[1] = sim.getOutput("xf")[1,0]
292 
293 
294                 F1_u[i+1:i+2] = F1[j]
295                 F2_u[i+1:i+2] = F2[j]
296                 F10_u[i+1:i+2] = F10[j]
297 
298                 XH2_y.append( sim.getOutput("xf")[0,0])
299                 X5_y.append( sim.getOutput("zf")[1,0])
300 
301             X_temp1.append( sim.getOutput("xf")[:,-1] )
302             Z_temp1.append( sim.getOutput("zf")[:,-1] )
303 
304     return XH2_y,X5_y,F1_u,F2_u,F10_u
305 
306 def MonteCarlo(v_opt,ns):
307     """This is a function that runs Monte Carlo simulations for the ...
            hydrodesulphurisation
308     plant. The function takes the optimal solution from the nlp solver, and ...
            then simulates
309     ns simulation, with random outcome of X3 and rho. The output of the ...
            function are
310     the endpoints from the simulation, X_end, and the endpoints from each ...
            step in
311     the simulations, X_sim.
312     """
313     # Generates random outcomes for X3 and rho
314     # ns = 100 # Number of outcomes
315     sigma_x3 = 0.013 # Variance for X3
316     sigma_rho = 0.4 # Variance for FXH2
317     mu_x3 = 0.85 # Mean value for X3
318     mu_rho = 12.6 # Mean value for FXH2
319     mean = [mu_x3, mu_rho] # Mean values vector
320     r = -0.5 # correlation coefficient
321     cov = [[sigma_x3**2, r*sigma_rho*sigma_x3], [r*sigma_rho*sigma_x3, ...
            sigma_rho**2]]  # diagonal covariance
322     # For each outcome of X3 and rho run a simulation to get the outcome of ...
            X3 and X5
323     X_sim = [] # Make a a matrix to store all differential states for each ...
            simulation outcome
324     Z_sim = [] # Make algebraic states matrix for each simulation outcome
325     X_end = []; # Make a matrix to store differential states end values ...
            from each simulation
326     Z_end = []; # Make a matrix to store algebraic states end values from ...
            each simulation
327     ts = linspace(0,TF,N+1) # Time steps
328     # Extract the optimal solution to each variable
329     XH2 = v_opt[0::nvar]
330     FXH2 = v_opt[1::nvar]
```

```
331     F5  = v_opt[2::nvar]
332     X5  = v_opt[3::nvar]
333     F3 = v_opt[4::nvar]
334     F1 = v_opt[5::nvar]
335     F2 = v_opt[6::nvar]
336     F10 = v_opt[7::nvar]
337
338     for j in range(ns):
339         # Define States and Control variables
340         X = SX.sym("X",2) # Differential states XH2, FxH2,
341         U = SX.sym("U",3) # Control variables F1, F2, F10
342         Z = SX.sym("Z",3) # Algebraic states  F5, X5, F3
343
344         # Differential equations
345         F_x = vertcat( [ ( ( Z1*R*T )/( V*P ) )*( Z[0]*Z[1] - U[2]*X[0] - ...
                X[1] ), ( 1/tau )*( FHC*np.random.multivariate_normal(mean, ...
                cov)[1] - X[1] ) ] )
346
347         # Algebraic equations
348         F_z = vertcat( [ U[0] + U[1] + Z[2] - Z[0] , U[0]*X1 + U[1]*X2 + ...
                Z[2]*np.random.multivariate_normal(mean, cov)[0] - Z[0]*Z[1], ...
                Z[0] - U[2] - X[1] ] )
349
350         # Objective function
351         F_u = vertcat( [ CH4*X1*U[0] + CH3*X2*U[1] ])
352
353         # Define the dae system to be called by the integrator
354         DAE = SXFunction("DAE", daeIn( x=X, z=Z, p=U, t=t ), daeOut( ...
                ode=F_x, alg=F_z, quad=F_u) )
355
356         # Integrator for solving the Dae system
357         opts = {"tf":TF/N} # interval length
358         I_sim = Integrator("I_sim","idas", DAE)
359
360         # Run simulation for each outcome of the random variable
361         X_temp = []
362         Z_temp = []
363         for i in range(N):
364             # Run the simulator function that simulates the state variables ...
                    for each
365             # step with a given value of input variables
366             sim = Simulator("sim",I_sim,[ts[i],ts[i+1]] )
367
368             if i == 0:
369                 sim.setInput([XH2[0],FXH2[0]],"x0")
370                 sim.setInput([F5[0],X5[0],F3[0]],"z0")
371                 sim.setInput([F1[0],F2[0],F10[0]],"p")
372
373             else:
374                 sim.setInput(X_temp[-1],"x0")
375                 sim.setInput(Z_temp[-1],"z0")
376                 sim.setInput([F1[i],F2[i],F10[i]],"p")
377
378             sim.evaluate()
379             # Extract the end point for the state variables
380             X_temp.append( sim.getOutput("xf")[:,-1] )
381             Z_temp.append( sim.getOutput("zf")[:,-1] )
382             if i == N-2:
383                 X_end.append( sim.getOutput("xf")[0,-1] )
384                 Z_end.append( sim.getOutput("zf")[1,-1] )
385         Xsim = vertcat(X_temp)
386         Zsim = vertcat(Z_temp)
```

```
387
388         # Take the XH2 and
389         X_sim.append( Xsim[0::nx] )
390         Z_sim.append( Zsim[1::nz]  )
391     # Convert from Dmatrix to np.array
392     Xend = np.hstack(X_end)
393     Zend = np.hstack(Z_end)
394     X_Sim = np.hstack(X_sim)
395     Z_Sim = np.hstack(Z_sim)
396
397     return X_Sim,Z_Sim
398 def backoff(X_sim,Z_sim):
399
400     """ This function calculates the back-off for each step, this is done
401     with fitting the outcome from the Monte Carlo simulations to a normal
402     probability distribution, then finding a back-off such that constraint
403     holds with given value, alpha, or higher. This back-off also take into
404     account the difference between the mean value and the lower bound
405     """
406     alpha_x = 0.9
407     alpha_z = 0.9
408     mu_x = np.zeros(N)
409     mu_z = np.zeros(N)
410     sigma_x = np.zeros(N)
411     sigma_z = np.zeros(N)
412     Z = []
413     sqrt_Z = 0
414     back_off_XH2 = []
415     back_off_X5 = []
416
417     for i in range(N):
418         (mu_x[i], sigma_x[i]) = norm.fit(X_sim[i][:]) # Find the mean and ...
                variance for XH2, if XH2 have normal distribution
419         (mu_z[i], sigma_z[i]) = norm.fit(Z_sim[i][:]) # Find the mean and ...
                variance for X5, if X5 have normal distribution
420         # Find the covariance matrix Z
421         Z.append(np.cov(X_sim[i],Z_sim[i]))
422
423         # Calculates the back-off
424         # back_off_XH2.append(  norm.cdf(alpha_x)*( ...
                sqrt(np.cov(X_sim[i][:])*2) ) )
425         # back_off_X5.append(  norm.cdf(alpha_z)*( ...
                sqrt(np.cov(Z_sim[i][:])*2) ) )
426
427         sqrt_Z = sqrtm(Z[i]) # Square root of the covariance matrix
428         # if  (XH2_lb - mu_x[i]) > 0:
429             # back_off_XH2.append(  XH2_lb - mu_x[i] + ...
                    np.dot([norm.cdf(alpha_x),norm.cdf(alpha_z)],sqrt_Z)[0] )
430         # else:
431         back_off_XH2.append( ...
                np.dot([norm.cdf(alpha_x),norm.cdf(alpha_z)],sqrt_Z)[0] )
432         if  ( X5_lb - mu_z[i]) > 0:
433             back_off_X5.append(  X5_lb - mu_z[i] + ...
                    np.dot([norm.cdf(alpha_x),norm.cdf(alpha_z)],sqrt_Z)[1] )
434         else:
435             back_off_X5.append( ...
                    np.dot([norm.cdf(alpha_x),norm.cdf(alpha_z)],sqrt_Z)[1] )
436
437     return back_off_XH2,back_off_X5
438
439
440     steps = 5
```

```
case = 2
steps = 20
XH2_y,X5_y,F1_u,F2_u,F10_u = MPC(v,J,g,steps,case)
end = time.time()
print end - start
plt.show()
t = np.linspace(0,TF,steps+1)
XH2_Y = vertcat(XH2_y)
X5_Y = vertcat(X5_y)
F1_U = vertcat(F1_u)
F2_U = vertcat(F2_u)
F10_U = vertcat(F10_u)
print F1_U,"F1_u"
print XH2_Y, "XH2_Y"
print F1_U.shape, "F1_U.shape", F2_U.shape, "F2_U.shape", F10_U.shape, ...
    "F10_U.shape"
if case == 1:

    # DataOut = np.hstack((XH2_Y,X5_Y))
    file = open("MPC_plot_case1.txt", "w")
    file.write("\n".join(map(lambda x: str(x), XH2_Y  )))
    file.write("\n")
    file.write("\n".join(map(lambda x: str(x), X5_Y  )))
    file.write("\n")
    file.write("\n".join(map(lambda x: str(x), F1_U )))
    file.write("\n")
    file.write("\n".join(map(lambda x: str(x), F2_U  )))
    file.write("\n")
    file.write("\n".join(map(lambda x: str(x), F10_U  )))
    file.close()
elif case == 2:
    # DataOut = np.hstack((XH2_Y,X5_Y))
    file = open("MPC_plot_case2.txt", "w")
    file.write("\n".join(map(lambda x: str(x), XH2_Y  )))
    file.write("\n")
    file.write("\n".join(map(lambda x: str(x), X5_Y  )))
    file.write("\n")
    file.write("\n".join(map(lambda x: str(x), F1_U )))
    file.write("\n")
    file.write("\n".join(map(lambda x: str(x), F2_U  )))
    file.write("\n")
    file.write("\n".join(map(lambda x: str(x), F10_U  )))

    file.close()
else:
    print "case not defined"
```

F.3 Model Predictive Control Plotting, Python code

Here are the code used for plotting the results from the MPC simulations.

```
import matplotlib.pyplot as plt
import numpy as np

steps = 20
# Get values for offline optimisation
lines1 = [line.rstrip('\n') for line in open('MPC_plot_case1.txt')]
MPC1 = np.vstack(lines1)

```

```
XH2_1 = MPC1[0:steps]
X5_1 = MPC1[steps:2*steps]
F1_1 = MPC1[2*steps:3*steps+1]
F2_1 = MPC1[3*steps+1:4*steps+2]
F10_1 = MPC1[4*steps+2:len(MPC1)]

# Get values for online optimisation
lines2 = [line.rstrip('\n') for line in open('MPC_plot_case2.txt')]
MPC2 = np.vstack(lines2)
XH2_2 = MPC2[0:steps]
X5_2 = MPC2[steps:2*steps]
F1_2 = MPC2[2*steps:3*steps+1]
F2_2 = MPC2[3*steps+1:4*steps+2]
F10_2 = MPC2[4*steps+2:len(MPC1)]
TF = 2.0

ty = np.linspace(0,TF,len(XH2_1))
t = np.linspace(0,TF,len(F1_1))

plt.figure(1)

plt.subplot(221)
plt.step(t,F1_1,'.',lw=2,label='F1_1')
plt.step(t,F1_2,'.',lw=2,label='F1_2')
plt.legend(bbox_to_anchor=(0.8, 0.4), loc=2, borderaxespad=0.)
plt.xlabel('time, [h]')
plt.ylabel('F1, [kmol/h]')

plt.subplot(222)
plt.step(t,F2_1,'.',lw=2,label='F2_1')
plt.step(t,F2_2,'.',lw=2,label='F2_2')
plt.legend(bbox_to_anchor=(0.8, 0.4), loc=2, borderaxespad=0.)
plt.xlabel('time, [h]')
plt.ylabel('F2, [kmol/h]')
plt.ylim([0.0, 700])

plt.subplot(223)
plt.step(t,F10_1,lw=2,label='F10_1')
plt.step(t,F10_2,lw=2,label='F10_2')
plt.legend(bbox_to_anchor=(0.76, 0.4), loc=2, borderaxespad=0.)
plt.xlabel('time, [h]')
plt.ylabel('F10, [kmol/h]')
XH2_lb = np.ones(len(XH2_1))*0.7
X5_lb = np.ones(len(X5_1))*0.9
plt.subplot(224)
plt.step(ty,XH2_1,'.',lw=2,label='XH2_1')
plt.step(ty,X5_1,'.',lw=2,label='X5_1')

plt.step(ty,XH2_2,'.',lw=2,label='XH2_2')
plt.step(ty,X5_2,'.',lw=2,label='X5_2')
plt.plot(ty,XH2_lb,'k--')
plt.plot(ty,X5_lb,'k--')
plt.legend(bbox_to_anchor=(0.7, 0.74), loc=2, borderaxespad=0.)
plt.xlabel('time, [h]')
plt.ylabel('X, [-]')
plt.ylim([0.65, 0.95])

plt.show()
```