NTNU – Trondheim
Norwegian University of
Science and Technology

Plantwide- and Self-Optimizing Control of a Reactor with Recycle Process

Helene Paulsen

Chemical Engineering and Biotechnology
Submission date: June 2013
Supervisor: Sigurd Skogestad, IKP
Co-supervisor: Vladimiros L. Minasidis, IKP

Norwegian University of Science and Technology
Department of Chemical Engineering

Preface

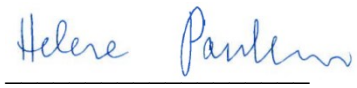
This thesis was performed as the last part of my master degree in Industrial Chemistry and Biology at the Norwegian University of Science and Technology (NTNU). My specialization was in chemical engineering, where I was a part of the process systems group. My master's thesis title is "Plantwide- and Self-Optimizing Control of a Reactor with Recycle Process.

I would like to thank my supervisor Sigurd Skogestad and my co-supervisor Vladimiros Minasidis who have helped me with any problems during the semester.

Declaration of Compliance

I declare that this is an independent work according to the exam regulations of the Norwegian University of Science and Technology (NTNU).

Trondheim, 13.6.2013



Helene Paulsen

Abstract

Plantwide control is a very important topic in today's process plants in a lot of different industries. With a large number of interacting process units, it is not enough to design a control structure for each single unit by itself without considering the whole system. Control decisions highly affect the economics of a plant, and that is why it is such an important topic. For example, a huge part of the energy costs can be saved by selecting a proper control system.

The objective of this work was to find a control structure for an entire process plant, including which variables to control, which variables to be manipulated and which variable to measure. Skogestad's procedure was applied to a reactor and distillation column process with recycle. This procedure included finding an economic cost function, finding the active constraints, defining the degrees of freedom, and identify self-optimizing variables based on economic loss.

Two different cases were studied in this thesis. Case I had a given reactor feed rate with the objective of minimizing the vapour boilup in the column, while case II had the objective of maximizing the feed rate with a given vapour boilup.

Matlab models of the reactor and distillation column were made, and the process was optimized for both of the studied cases. Following this, the nominal optimal values of compositions and flows were compared between the cases and also compared with existing literature. The optimization results were consistent with the literature written on the same process.

Three different methods were tested in order to find which self-optimizing variable to use for the last remaining degree of freedom in the system. These were the Brute force method, the null space method and the exact local method. In the Brute force method, the economic cost was computed for all the candidate controlled variables by keeping them constant and applying disturbances and implementation errors. The losses resulting from this were plotted to see which variable had the smallest loss. This ended up being the flow ratio L/F, and was therefore chosen as the self-optimizing variable for both case I and II.

Measurement combinations to be held constant were found by applying the null space and exact local method, followed by calculating the economic loss caused by this. Dynamic simulations were also performed in order to find the loss with the null space and exact local method. The dynamic model gave the same nominal optimization results as the steady-state model.

Control structures and pairings between controlled variables and manipulated variables were suggested for the two cases based on the results from Sigurd's procedure. For control of the self-optimizing variable L/F, the reflux flow L was used as manipulated variable for the single control loop. This was because the reflux L was the only remaining unconstrained degree of freedom after the control analysis.

Sammendrag

Regulering av hele anlegg er et meget viktig tema for de fleste av dagens prosessanlegg i mange forskjellige industrier. Med et stort antall samhandlende prosessenheter er det ikke tilstrekkelig å designe en reguleringsstruktur for hver enkelt enhet uten at hele systemet tas i betraktning. Avgjørelser med tanke på regulering har ofte stor innvirkning på de økonomiske sidene av anlegget, noe som er en viktig grunn til hvorfor dette er et så viktig tema. Mye energikostnader kan for eksempel spares ved å velge det riktige reguleringssystemet.

Formålet med dette arbeidet var å finne en reguleringsstruktur for et helt prosessanlegg, inkludert å identifisere hvilke variabler som skal reguleres, hvilke variables som skal manipuleres, og hvilke variabler som skal måles. Skogestads metode ble utført på en reaktor- og destillasjonskolonne-prosess med resirkulering. Denne metoden går blant annet ut på å definere en kostnadsfunksjon, aktive betingelser, antall frihetsgrader og selvoptimaliserende variabler basert på økonomisk tap.

To ulike tilfeller ble undersøkt. I det første tilfellet ble reaktorføden gitt, der formålet var å minimere damp-oppkoket i kolonnen, mens i det andre tilfellet var formålet å maksimere fødestrømmen med et gitt damp-oppkok.

Det ble lagd Matlab-modeller av reaktoren og destillasjonskolonnen, og prosessen ble så optimalisert i begge tilfellene. Etter dette ble de optimale verdiene av strømmer og sammensetninger sammenlignet mellom de to tilfellende og med eksisterende litteratur. Resultatene fra optimaliseringen stemte overens med litteraturen som ble funnet om den samme prosessen.

Tre ulike metoder ble prøvd for å finne den beste selvoptimaliserende variabelen som skulle bli brukt for den siste frihetsgraden. Disse var henholdsvis Brute-kraftmetoden, nullroms-metoden og den eksakte lokale metoden. I Brute-kraftmetoden ble kostnaden beregnet for alle kandidatene til regulert variabel ved å holde dem konstant på sin optimale verdi, for så å tilføre forstyrrelser og implementeringsfeil. De resulterende tapene ble plottet for å se hvilken variabel som hadde det minste tapet. Dette ble variabelen L/F, og den ble derfor brukt som selvoptimaliserende variabel for tilfelle I og II.

Nullroms- og den eksakte lokale metoden ble brukt til å finne målekombinasjoner som skulle holdes konstant, etterfulgt av beregning av tap på grunn av dette. I tillegg ble det utført dynamiske simuleringer for å finne tapet ved å bruke disse to metodene. Den dynamiske modellen ga de samme optimaliseringsresultatene som den stasjonære modellen.

Forslag til reguleringsstrukturer og paringer mellom regulerte variabler og manipulerte variabler ble utarbeidet for de to tilfellene. Disse forslagene var basert på resultatene fra Sigurds metode. Tilbakestrømningen L ble brukt som manipulert variabel for å regulere den selvoptimaliserende variabelen L/F i en enkel reguleringssløyfe. Dette ble gjort fordi tilbakestrømningen L var den eneste gjenstående frihetsgraden i systemet som ikke var brukt opp i løpet av reguleringsanalysen.

Contents

1. Introduction

Plantwide control deals with the overall control scheme of a chemical plant, where the structural decisions are important [1]. In this thesis the objective is to apply the plantwide control method to a process with a reactor and recycle. There is also a distillation column, where unreacted reactants is recycled back to the reactor as just mentioned. In addition, it is desired to find the self-optimizing variables of the process. This is necessary because by finding the self-optimizing variables, the process can be economically optimal. Also, the loss with different unconstrained degrees of freedom as self-optimizing variables will be evaluated. This can then prove why it is important to find which variables to control, and which not to control in a process plant.

Throughout this project it is assumed that a steady-state model is sufficient for most of the optimization and calculations. Also, the reaction happening in the reactor is assumed to be a first order reaction with two components, one reactant and one product.

Larsson et al. (2003) have already studied the control structure selection and what to control for the process considered in this project [2]. They found a complete control structure for the recycle process and also computed the losses for different self-optimizing variables for two cases. According to Larsson et al. (2003), the conventional way of controlling this process is by having a fixed reactor holdup and "two-point" distillation control [2]. More specifically, that means controlling M_r, x_B and x_D. Several other control configurations have been proposed in the literature.

Wu and Yu (1996) have looked at the reactor/separator process as a whole unit, and then at each of the separate units by themselves [3]. They found that the behaviour of the process differs from the behaviour of the different units, both at steady-state and dynamically. A control structure and tuning procedure was also proposed.

Regarding the search for self-optimizing variables, Alstad et al. (2009) studied a method to find optimal measurement combinations as controlled variables [4]. By doing that, self-optimizing control was achieved. This method is called the null space method, and was also used by Alstad and Skogestad (2007) earlier to find the self-optimizing variables [5]. They used a steady-state model of a process to achieve this. In addition, the optimal selection of controlled variables was studied by Halvorsen et al. (2003) with the exact local method, an extended version of the null space method [6].

Skogestad (2000) developed a strategic method to find the self-optimizing variables of a process and gives the criteria for this variable [7]. This also includes finding which variables to measure, which inputs to manipulate, and which links to make between those. Also, Skogestad (2004) follows a systematic procedure for finding the control structure for a complete chemical plant [8]. Other issues, such as inventory control, production rate control, decentralized versus multivariable control and loss in performance by bottom-up design is also considered in the paper just mentioned.

Luyben (1993) has explored and described the problems and challenges associated with the dynamics and control of recycle systems [9]. Several different process designs were analysed and the steady-state economics and controllability of the systems were compared.

Skogestad (2007) claims that the LV-configuration for level control combined with a fast temperature loop is the best choice for distillation column control [10]. The dos and don'ts of distillation column control are summarized in this paper. Three main issues are presented, namely the configuration problem, the temperature control problem and the composition control problem.

Wu et al. (2002) explored different control configurations to achieve the best composition control and to improve the disturbance rejection [11]. This was done to avoid complex control configurations for composition control.

The method of steady-state optimization and self-optimizing control has also been applied to a large-scale HDA plant (hydrodealkylation of toluene) by Araújo et al. (2007) [12]. Here, a linear analysis based on singular value decomposition (SVD) was used for pre-screening. This is because large plants can have combinatorial problems related to the choice of measurements and outputs.

Other processes have also been studied for plantwide control. Larsson et al. (2001) applied this to a large-scale plant where the issue was "what should we control" [13]. This was for the Tennessee Eastman process, where the concept of self-optimizing control was used to choose between many candidate variables. A systematic procedure was used for reducing the number of alternatives [13].

2. Theory

2.1. Plantwide Control

The plantwide control theory is about choosing which variables to control, which variables to measure, which inputs to manipulate, and which links to make between them [14]. Two different ways to approach this is available, that is a mathematically oriented approach (control structure design) and a process oriented approach.

2.2. Skogestad's Plantwide Control Procedure

Skogestad (2011) has suggested a method to decide the control structure of a plant [15]. This method includes starting with the economic objective of the plant and to design a control scheme that implements the optimal operation. Figure 1 shows the typical control hierarchy of any complete chemical plant.

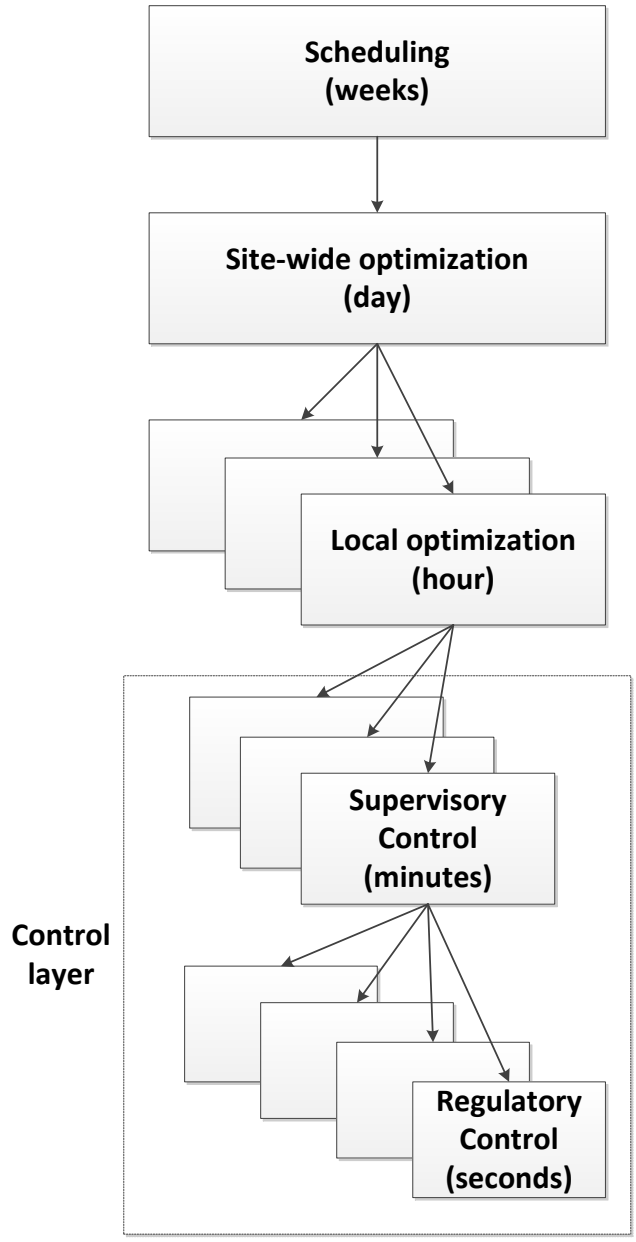


Figure 1: Typical control hierarchy in a chemical plant [12].

Skogestad's method consists of two parts, a top-down analysis and a bottom-up design, as described in the following sections [8]. As will be seen in chapter 2.2.2, the supervisory regulatory control layer in Figure 1 is included in the bottom-up design.

2.2.1. Top-Down Analysis

Four different steps make up the top-down analysis in Skogestad's procedure, and they are all explained in the following sub chapters. These steps deal mostly with plant economics. Steady-state models are usually sufficient for this part because plant economics are determined primarily by the steady-state behaviour [1].

Step S1: Definition of Operational Objectives

The first step is to find the operational constraints, and in addition define a scalar cost function J. In most cases, J represents the operational cost, but other possibilities exist [8]. This cost function is to be minimized. Equivalently, a scalar profit function P = -J, could be maximized [1]. A typical form of the scalar cost function can be given as follows:

$$J = cost\ feed + cost\ utilities\ (energy) - value\ products\ (\$/s) \qquad (1)$$

In the equation above, only factors that are influenced by plant operation on the given timescale are included. This is why neither fixed costs nor capital costs are included [1]. Further on, the objective of control and operation is to minimize the cost J, subject to satisfying the operational constraints. Operational constraints include both safety and environmental constraints. Some typical constraints include minimum and maximum values on temperatures, pressures, compositions, and flows. By intuition, it is clear that the values of all compositions, pressures and flows must be positive.

Taking the constraints into account, the minimization of the cost function can be defined as follows:

$$\min_u J_X(x, u, d), \qquad (2)$$

Subject to the constraints

$$g_1(x, u, d) = 0 \qquad (3)$$

$$g_2(x, u, d) \leq 0 \qquad (4)$$

In the equations above, x represents the state variables, u denote the degrees of freedom for optimization, and d represent the disturbances. Equation (3) represents the non-linear equality constraints, while equation (4) denotes the non-linear inequality constraints.

Step S2: Identify Degrees of Freedom and Optimize Operation for Important Disturbances

It is important to know what the optimal way of operating the process is. This should be known before designing the control structure. A reason for this is that variables that turn out to be active constraints need to be controlled [1]. A steady-state model is needed to obtain the steady-state optimal operation. Further on, the number of degrees of freedom and important disturbances must be determined. From this follows an optimization of the process for the expected disturbances. These steps are explained as follows.

(a) Degree of Freedom Analysis

The steady-state degrees of freedom (DOF) are given as

$$N_{ss} = N_m - \left(N_{0m} + N_{0y}\right) \tag{5}$$

Here, N_m is the number of dynamic or control degrees of freedom and m stands for manipulated [8]. Thus, N_m is equal to the number of manipulated variables. In most cases this number can be found by counting the number of adjustable valves plus other adjustable electrical and mechanical variables in a process.

The degrees of freedom that affect the operational cost J are referred to as the N_{opt} optimization degrees of freedom. Usually, N_{opt} is the number of steady-state degrees of freedom, N_{ss}, because the cost often only depends on the steady-state.

Further on, N_{0m} is the number of manipulated (input) variables with no steady-state effect or no influence on the operational cost. Sometimes, this can for example be a heat exchanger or an extra bypass [8]. The last parameter in the equation above, N_{0y}, is the number of (output) variables that must be controlled, but which have no effect on the cost, thus no steady-state effect. This often turns out to be liquid levels in holdup tanks.

(b) Identify Disturbances

The disturbances for a certain process need to be defined so that the optimal operation can be implemented in the next step. Often the most important disturbances are flows that cannot be controlled, for example a feed rate. Flow compositions can also be considered as disturbances.

Step S3: Implementation of Optimal Operation (Primary Controlled Variables)

This step deals with the choice of primary controlled variables (CVs) and self-optimizing control. First of all, the variables that are directly related to achieving optimal economic operation need to be controlled [8]. These are called the primary controlled variables. Thus, the first rule is to control active constraints. Secondly, control self-optimizing variables for the remaining unconstrained degrees of freedom [1]. Self-optimizing variables are defined as variables with constant set-points where almost optimal operation is achieved, even when disturbances or implementation errors are present. Self-optimizing control is described in more detail in chapter 2.3.

Step S4: Where to Set the Production Rate (Inventory Control)?

The production rate represents a degree of freedom which is called the throughput manipulator (TPM) [1]. So the amount of mass passing through the process is determined by specifying this variable and is often set as the feed rate or product rate. A common word for the TPM is the "gas pedal" of the process, and links the top-down and bottom-up parts of Sigurd's procedure. In most cases, the TPM is a flow, but it can be other variables as well.

2.2.2. Bottom-Up Design

The four last steps make up the bottom-up design.

Step S5: Regulatory Control Layer (Secondary Controlled Variables)

The main objective of the regulatory control layer is to stabilize the given plant. Selection of secondary controlled variables (CV_2) and selection of which inputs to be paired with these are the main decisions in this part [1]. When disturbances occur, the plant is stabilized if the process does not drift too far away from the optimal operation. Manipulated variables that may saturate are preferably not used as secondary controlled variables because this can cause loss of control and reconfiguration of loops may be required [8]. Skogestad (2004) has found that certain properties are wanted for a good secondary controlled variable (measurement), as listed here [8]:

- The variable is easy to measure
- The variable is easy to control using one of the available manipulated variables
- For stabilization: the unstable mode should be detected "quickly" by the measurement
- For local disturbance rejection: the variable is located "close" downstream of an important disturbance.

In order to stabilize overall inventory in different units in liquid phase systems, liquid level can be controlled. It is also possible to control a composition or a temperature in order to stabilize the inventory of different components. The regulatory layer can also be named the lower layer, secondary loops or inner loops [8].

Step S6: Supervisory Control Layer

The supervisory control layer is also referred to as the advanced control layer or the primary control system. The main objective of this layer is to keep the primary controlled outputs at their optimal set-points (c_s) by using the set-points in the regulatory layer as manipulated variables or degrees of freedom [8]. In addition, any unused manipulated inputs could also be used to control the primary outputs. Two other tasks for the supervisory layer include supervision of the performance of the regulatory layer, and switching of the controlled variables and control strategies due to change in disturbance or price. These changes can cause the process to enter a new region of active constraints.

Step S7: Optimization Layer

The real time optimization (RTO) layer is used to update the set-points for the controlled variables and to discover any changes in the active constraint regions which can be caused by changes in disturbances and implementation errors [1]. Changes in the active constraint region can lead to switching of the controlled variables. However, the benefit of the RTO layer can be to low compared to the cost of generating the steady-state model needed to do that.

Validation

When the plantwide control scheme is determined, it can sometimes be wise to validate the structure. This can for example be performed by dynamic simulation.

2.3. Self-Optimizing Control

As explained earlier, self-optimizing control is achieved when self-optimizing variables are controlled by remaining unused degrees of freedom after selecting the primary controlled variables. By using self-optimizing control, the system is optimized by itself without the need for on-line optimization [16]. The procedure is to keep a certain variable constant at a set-point, and this variable is the self-optimizing variable.

As before, a scalar cost function J has to be defined, and the goal is to minimize J. The real optimal solution would be to have a perfect model, and to re-optimize the process dynamically when disturbances and implementation errors occur [7]. However, this is not very realistic, and that is why self-optimizing control is used instead. If the loss by using self-optimizing control is acceptable, the process still operates satisfactorily. From this, the loss is defined as the difference between the value of the cost function when using self-optimizing control and the value of the cost function of the truly optimal solution:

$$Loss = J_u(u, d) - J_{opt}(u, d) \tag{6}$$

The definition of self-optimizing control is then as defined by Skogestad (2000) [7]:

> "Self-optimizing control is when we can achieve an acceptable loss with constant set-point values for the controlled variables (without the need to re-optimize when disturbances occur)."

Figure 2 illustrates the loss caused by keeping different controlled variables at constant set-points. The value of the cost function J is shown as a function of the disturbance value, where d* is the nominal value of the disturbance. In this case, the imposed loss is smaller for the first controlled variable ($c_{1,s}$) than the second one ($c_{2,s}$). At the nominal disturbance value, there is no loss for any of the controlled variables, but as the disturbance increases, so does the value of the loss. As a consequence, the variable c_1 would be chosen as the self-optimizing variable instead of c_2 if the loss was as shown in Figure 2.

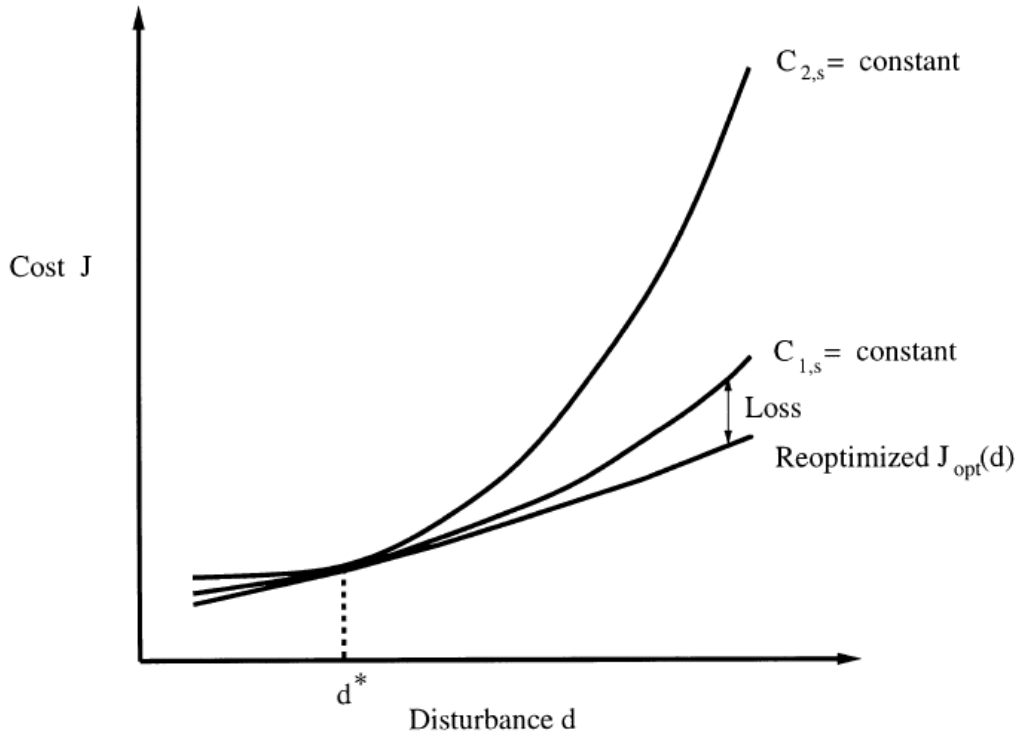


Figure 2: Loss imposed by keeping different controlled variables constant at their individual set-points [7].

Also, implementation errors will occur which is caused by for example measurement error or imperfect control. These are represented as follows:

$$d_c = c - c_s \tag{7}$$

There will always be a difference between the set-point c_s and the actual value of c because of the implementation errors. This causes an extra concern with the constant set-point-strategy, and the cost surface plotted as a function of c should therefore be as flat as possible [13]. This is to minimize the effect of the measurement or control errors. Figure 3 illustrates this for three different versions of the cost function surface.

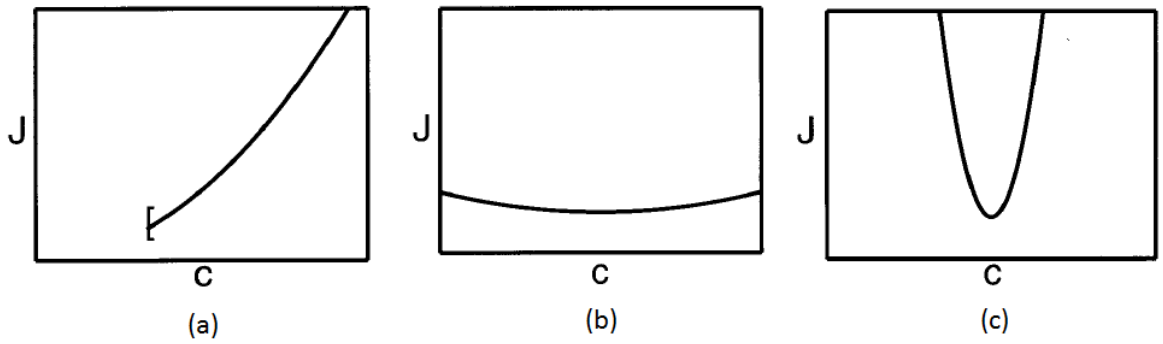


Figure 3: Implementation of the controlled variable [13].

In plot (a) the implementation is easy because of active constraint control. As seen in the figure, the optimum is constrained, so the optimal cost is obtained when one of the variables is at its minimum or maximum [13]. In this case Figure 3 shows a case where the optimum is when c is equal to c_{min}. By keeping the controlled variable constant at its active constraint, there is no loss occurring. In (b) there is an unconstrained flat optimum, where the cost is not highly affected by the value of the controlled variable [13]. In (c), the cost J is sensitive to the value of the controlled variable because of the constrained sharp optimum happening here. Therefore, self-optimizing control is not possible in the latter case.

For an active constraint case, the solution is to select the optimally constrained variables as controlled variables. However, in the unconstrained case in (b) is it not that clear which variables to control [13]. As a guideline, Skogestad has suggested four requirements for choosing a good candidate controlled variable [7]:

1. The candidate variable c should be easy to measure and control.
2. The optimal value of c should be insensitive to disturbances. Thus c_{opt} (d) should not be very affected by changes in d.
3. The value of c should be sensitive to changes in the manipulated variables u. Thus, the gain from u to y (G) should be large. This is equivalent to having a flat optimum with respect to c.
4. For cases with two or more controlled variables, the chosen variables should not be closely correlated.

The requirements above can be a good way to reduce the number of candidate controlled variables in a plant with a lot of possible candidate controlled variables. Also, the optimal self-optimizing variable would be the derivative of the cost with respect to the input u:

$$c = \frac{\partial J}{\partial u} = J_u \tag{8}$$

However, this is not realisable because the gradient in equation (8) cannot be measured. As a consequence, other self-optimizing variables with economic losses have to be found.

There are several methods to find the self-optimizing variables with the smallest loss. Three different ways of calculating the loss for different controlled variables are explained in the following sub-chapters.

2.3.1. Brute Force Method

In order to avoid issues with local behaviour and infeasibility when finding the loss, it is possible to use a direct evaluation method called the Brute force method [16]. This method directly calculates the loss for expected disturbances and implementation errors. A model of the system is needed to evaluate the loss directly. According to Skogestad (2004), the following steps are required [16]:

- Define the degrees of freedom for optimization
- State optimal operation in terms of a cost function J and operational constraints
- Determine the important disturbances
- Use the model to find the optimal operation, both nominally and with disturbances
- Find the active constraints and control them
- Evaluate the loss with constant set-points for alternative controlled variables; do this for the remaining unconstrained degrees of freedom
- Evaluate more carefully the alternatives with the smallest loss, for example by controllability analysis

However, even though this is a simple method, it requires a lot of computations due to the large number of alternative controlled variables that may be considered [16]. It is therefore desired to limit the number of alternative controlled variables to be studied. One way to do this is to use a method called the minimum singular value, and eliminate variables with a small minimum singular value. Also, the evaluation of the loss only requires a steady-state model of the process. Another positive property with the Brute force method is that it is easy to detect controlled variables that may cause infeasibility for certain disturbances or implementation errors [5].

2.3.2. Null Space Method

The minimization problem that is studied is given by the following equation:

$$\min_{u} J(u, d) \tag{9}$$

This means that all of the optimally constrained variables are assumed to be kept constant at their respective optimal values [5]. In other words, "active constraint control" is assumed.

In the null space method, the objective is to find a linear measurement combination to be kept constant at the set-point, c_s. The measurement combination, c, is given by equation (10):

$$\boldsymbol{c = Hy} \tag{10}$$

Here, **c** is a vector, **H** is a constant n_u x n_y matrix and y is a vector with individual available measurements. As before, only the steady-state problem is considered because the cost function value is mostly determined by the steady state. In addition, the disturbances considered all affect the steady-state operation. In addition, all implementation errors are neglected, which is the sum of the control error and the effect of the measurement error [5]. The assumption of no control error is correct if a controller with integral action is used. When

it comes to neglecting the measurement error, this can be satisfied if the measurements are selected wisely.

In the **H** matrix, n_u is the number of degrees of freedom or independent unconstrained free variables **u** and n_y is the number of individual measurements (**y**). Also, n_d is the number of independent disturbances **d**, and it is desired to achieve $n_c=n_u$ independent controlled variables **c** that are linear combinations of the measurements, as given in equation (10).

The next step is to define the optimal sensitivity matrix, evaluated with constant active constraints. This matrix is given by equation (11), which describes the sensitivity of the optimal value of the measurements to changes in disturbances [5].

$$\boldsymbol{F} = \frac{\delta \boldsymbol{y}^{opt}}{\delta \boldsymbol{d}^{T}} \tag{11}$$

If the following criterion is satisfied:

$$n_y \geq n_u + n_d, \tag{12}$$

The **H** matrix can be selected in the left null space of **F**, $\mathbf{H} \in \mathcal{N}(\mathbf{F}^{T})$, such that

$$\boldsymbol{HF} = 0 \tag{13}$$

When the matrix **H** is defined like this, first-order optimality for disturbances **d** is achieved by fixing **c** at its optimal value. So as long as the sensitivity matrix **F** in equation (11) does not change, this method gives zero loss.

In order to prove that this selection of **H** gives zero loss, let

$$\boldsymbol{y}^{opt}(\boldsymbol{d}) - \boldsymbol{y}^{opt}(\boldsymbol{d}^{*}) = \boldsymbol{F}(\boldsymbol{d} - \boldsymbol{d}^{*}) \tag{14}$$

Equation (14) shows the optimal change in the measurements due to a small change in the disturbances, where

$$\boldsymbol{F} = \begin{bmatrix} \frac{\delta y_1^{opt}}{\delta d_1} & \cdots & \frac{\delta y_1^{opt}}{\delta d_{n_d}} \\ \vdots & \ddots & \vdots \\ \frac{\delta y_{n_y}^{opt}}{\delta d_1} & \cdots & \frac{\delta y_{n_y}^{opt}}{\delta d_{n_d}} \end{bmatrix} \tag{15}$$

is the optimal sensitivity matrix evaluated at the nominal optimal point * [5]. Equation (14) is valid only for small disturbance changes because only the first-order term from the Taylor expansion is taken into account. Thus, the higher order terms from the Taylor expansion are neglected in this case.

The corresponding optimal change in the controlled variables is found from equation (10) as follows:

$$\boldsymbol{c}^{opt}(\boldsymbol{d}) - \boldsymbol{c}^{opt}(\boldsymbol{d}^*) = \boldsymbol{H}\left(y^{opt}(\boldsymbol{d}) - \boldsymbol{y}^{opt}(\boldsymbol{d}^*)\right) \tag{16}$$

Inserting equation (14) in equation (16) gives the following:

$$\boldsymbol{c}^{opt}(\boldsymbol{d}) - \boldsymbol{c}^{opt}(\boldsymbol{d}^*) = \boldsymbol{HF}(\boldsymbol{d} - \boldsymbol{d}^*) \tag{17}$$

In order to maintain the constant set-point policy as described earlier, the left hand side in the above equation needs to be equal to zero. Thus,

$$\boldsymbol{c}^{opt}(\boldsymbol{d}) - \boldsymbol{c}^{opt}(\boldsymbol{d}^*) = 0 \tag{18}$$

This leads to the condition

$$\boldsymbol{HF}(\boldsymbol{d} - \boldsymbol{d}^*) = 0 \tag{19}$$

In order for the requirement in equation (19) to hold, it is required that

$$\boldsymbol{HF} = 0 \tag{20}$$

That is because the condition in equation (19) needs to be valid for all values of (**d-d***).

The rank of the n_c x n_y matrix **H** and the n_y x n_d matrix **F** is n_u and n_d, respectively [5]. This is due to the fact that $n_y \geq n_c$ (n_c=n_u) and $n_y \geq n_d$. Both the controlled variables and disturbances are assumed to be independent. Alstad & Skogestad (2007) proved that in order to find an **H** matrix of rank n_u in the left null space of **F**, it is required that $n_y \geq n_u + n_d$ [5].

One downside with the null space method is that it is only locally optimal, except in cases where the sensitivity matrix **F** is not dependent on disturbances.

Theoretically, the sensitivity matrix can be calculated from the steady-state gain matrices ($\mathbf{G}^y$ and $\mathbf{G}^y_d$) and the Hessian matrices ($\mathbf{J}_{uu}$ and $\mathbf{J}_{ud}$), as shown here:

$$\boldsymbol{F} = -\left(\boldsymbol{G}^y \boldsymbol{J}_{uu}^{-1} \boldsymbol{J}_{ud} - \boldsymbol{G}_d^y\right) \tag{21}$$

Nevertheless, the **F** matrix is often computed directly instead, by optimizing the process with a steady-state model for the chosen disturbances. By perturbing the disturbances and re-optimize the process with constant active constraints, **F** can be obtained numerically. The steps for achieving this are as follows [5]:

(1) Compute the nominal optimum $\mathbf{y}^{opt}$ (d*) with the steady-state model and determine the active constraints.
(2) Make small perturbations in the chosen disturbances and resolve the optimization problem to get $\mathbf{y}^{opt}$ (d).
(3) Calculate $\Delta\mathbf{y}^{\mathbf{opt}} = \mathbf{y}^{opt}$ (d) - $\mathbf{y}^{opt}$ (d*) and obtain the matrix **F** by applying equation (15).

Since the implementation error is not taken into account in the null space method, there may be more measurements than degrees of freedom and disturbances, so that

$$n_y > n_u + n_d \tag{22}$$

This results in extra degrees of freedom for selecting the **H** matrix. Ideally, these could be used to reduce the sensitivity to measurement error.

2.3.3. The Exact Local Method

Unlike the null space method, the exact local method considers noise in addition to disturbances and any number of measurements can be used in this method.

Let

$$\Delta \boldsymbol{d} = \boldsymbol{W}_d \boldsymbol{d}' \tag{23}$$

$$\boldsymbol{n}^y = \boldsymbol{W}_{n^y} \boldsymbol{n}^{y\prime} \tag{24}$$

In the above equations, $\Delta \boldsymbol{d}$ and $\boldsymbol{n}^y$ are the magnitudes of the disturbances **d** and measurement errors, respectively [4]. These magnitudes are quantified by the diagonal scaling matrices $\boldsymbol{W}_d$ and $\boldsymbol{W}_{n^y}$, respectively. The vectors $\boldsymbol{d}'$ and $\boldsymbol{n}^{y\prime}$ can be any vectors satisfying

$$\left\| \begin{bmatrix} \boldsymbol{d}' \\ \boldsymbol{n}^{y\prime} \end{bmatrix} \right\|_2 \leq 1 \tag{25}$$

Let

$$\boldsymbol{Y} = \boldsymbol{F}\boldsymbol{W}_d \boldsymbol{W}_d, \tag{26}$$

where **F** is the optimal sensitivity matrix as in the null space method, and $\boldsymbol{W}_d$ and $\boldsymbol{W}_{n^y}$ are the diagonal scaling matrices for the disturbances **d** and measurement errors $\boldsymbol{n}^y$, respectively [17]. The measurement combination matrix **H** can then be found as follows [17]:

$$\boldsymbol{H} = \left(inv(\boldsymbol{Y} * \boldsymbol{Y}') * \boldsymbol{G}_y \right)' \tag{27}$$

The steady-state gain matrix $\boldsymbol{G}_y$ is defined by

$$\boldsymbol{G}_y = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{u}} \tag{28}$$

Thus, the matrix $\boldsymbol{G}_y$ can be computed by making a small perturbation in the input **u**, and calculating the new measurement **y** by re-solving the problem. As in the null space method, the goal is to find a linear measurement combination to be held constant at the set-point c_s. The measurement combination, c, is given by equation (10) in the previous chapter.

2.4. SIMC Tuning Rules

There exist a lot of different rules for tuning of controllers in a chemical plant, and the SIMC tuning rules have been used in the industry in Norway already [18]. In this method the first step includes finding an approximate first- or second-order time delay model on the form

$$g_1 = \frac{k}{\tau_1 s + 1} e^{-\theta s} = \frac{k'}{s + \frac{1}{\tau_1}} e^{-\theta s} \tag{29}$$

Where

$$k' = \frac{k}{\tau_1} \tag{30}$$

If the model is second-order, the transfer function g_2 becomes

$$g_2 = \frac{k}{(\tau_1 s + 1)(\tau_2 s + 1)} e^{-\theta s} \tag{31}$$

In the equations above, k is the plant gain, τ_1 is the dominant lag time constant, θ is the time delay and τ_2 is the second-order lag time constant [18]. In addition, s is the Laplace parameter which has replaced the time parameter (t). There are three ways to obtain these parameters, namely:

- From open-loop step response
- From closed-loop set-point response with P-controller
- From detailed model: approximation of effective delay using the half rule

In this project, the open-loop step response method will be used to find the parameters.

2.4.1. First-order Process

A first-order process usually has the step response behaviour shown in Figure 4.

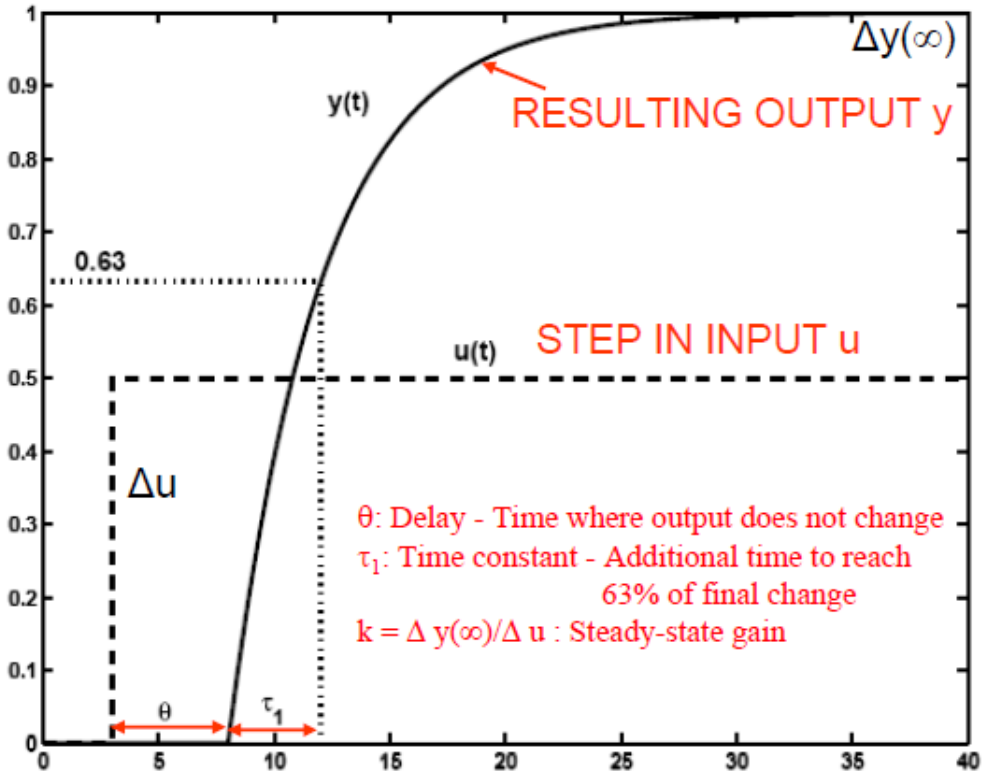


Figure 4: Open-loop step response test for a first-order process model [18].

It is possible to obtain the model parameters from the response above, even though it is not necessarily the most effective method to find them [18]. On the other hand, this is a quite simple way to do it. As explained on the figure, the steady-state gain is found as follows:

$$k = \frac{\Delta y(\infty)}{\Delta u} \tag{32}$$

Also explained on the figure, the time delay θ is the time where the output does not change (after the input step is applied). The time constant τ_1 is found as the time where 63% of the final output change is reached. To be clear, y is the output and u is the input. In addition, the unit of the x-axis is time, whereas the unit on the y-axis is any output y.

For a PI controller and a first-order model the SIMC method gives the tuning rules given in the following equations [18]:

$$K_c = \frac{1}{k}\frac{\tau_1}{\tau_c + \theta} = \frac{1}{k'}\frac{1}{\tau_c + \theta} \tag{33}$$

$$\tau_I = min\{\tau_1, 4(\tau_c + \theta)\} \tag{34}$$

Here, K_c is the controller gain, τ_I is the integral time and τ_c is the desired first-order closed loop time constant which is also the only tuning parameter [18].

2.4.2. Integrating Process

If the process is approximated as an integrating process, the parameter k' can be found as shown on Figure 5 below. Here, u is the input and Δu is the input change, Δy is the measurement change, and Δt is the time.

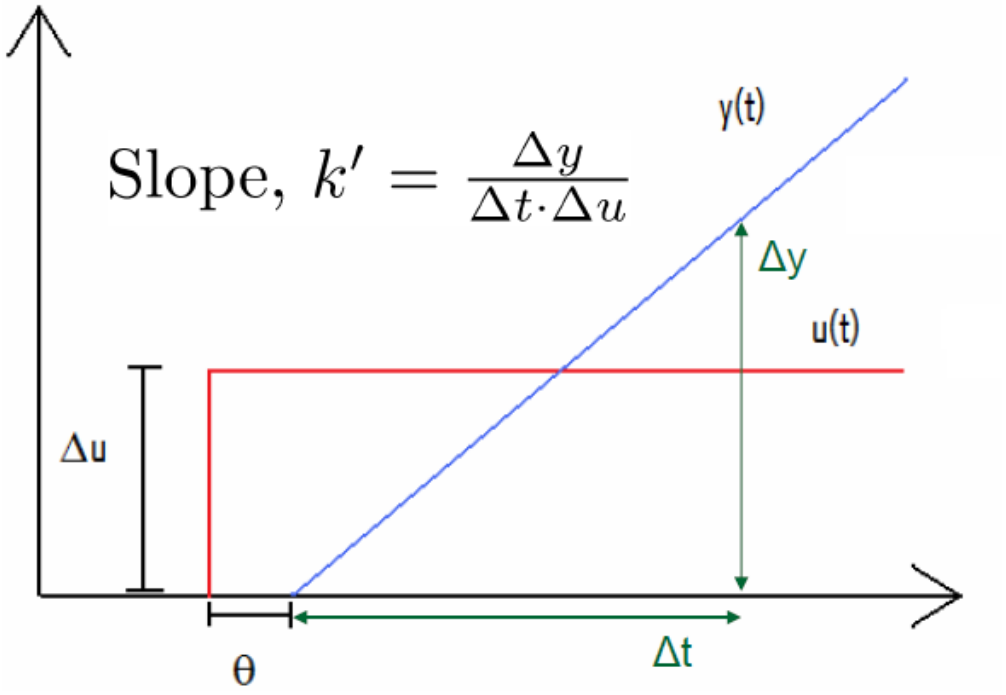


Figure 5: Open-loop step response for obtaining the parameters k' and θ for an integrating process [18].

With an integrating process the time constant τ_1 will go towards infinity. The controller gain K_c is still found by equation (33), but because τ_1 will always be larger than $4(\tau_c+\theta)$ in an integrating process, the integral time is now given by

$$\tau_I = 4(\tau_c + \theta) \tag{35}$$

3. Process Description

The process studied in this project is the same as given in Larsson et al. (2003), a reactor and separator with recycle process [2]. This recycle plant is sketched in Figure 6, and explained more detailed in the following.

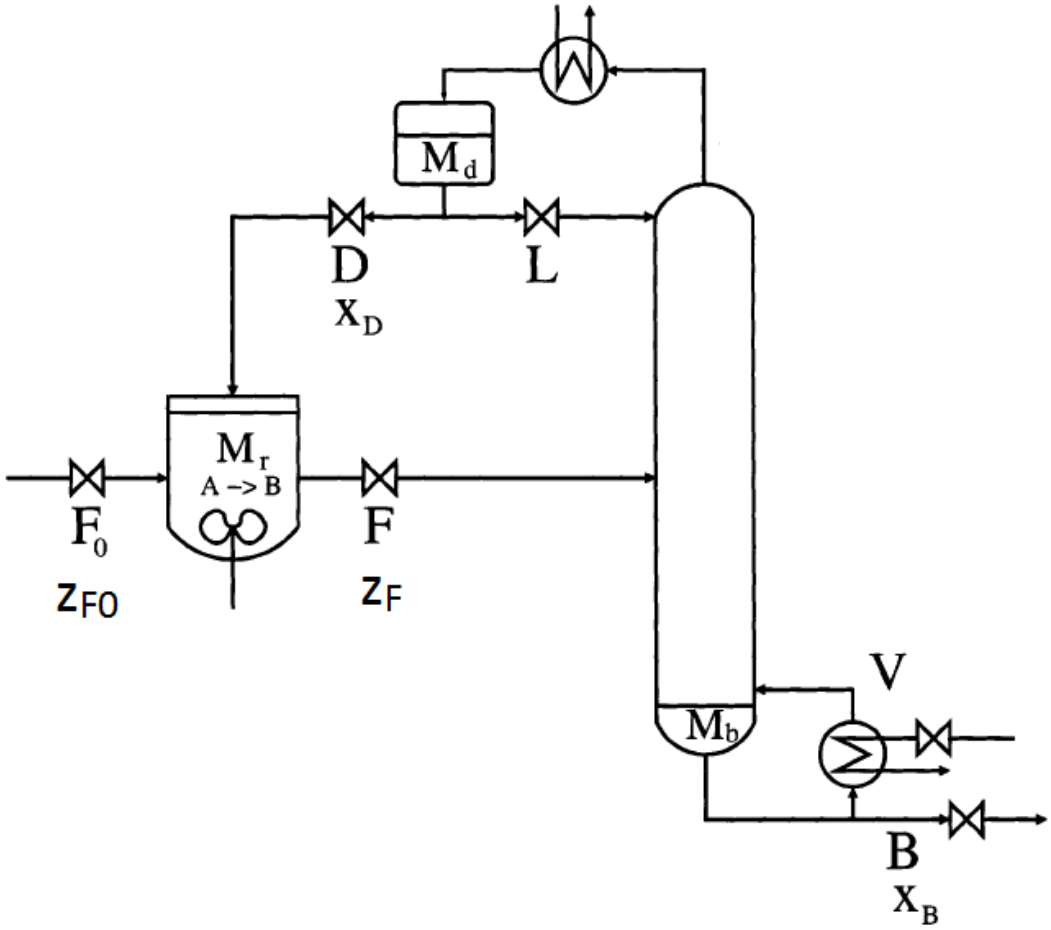


Figure 6: Schematic drawing of the reactor and recycle plant [2].

As seen in Figure 6, the plant consists of a reactor and a separator or distillation column, where some of the top product is recycled back to the reactor. The liquid feed F_0, is fed into the reactor, where the liquid reactant A is transformed to the liquid product B:

$$A(l) \rightarrow B(l) \tag{36}$$

The column feed, F, is separated in a distillation column with bottom product B and vapour boilup V. At the top of the column, some product is recycled back to the reactor as D, and the rest goes back into the column as the reflux L. Thus, this distillation column has a total condenser.

A description of the different symbols in Figure 6 is given in Table 1.

Table 1: Overview of the different flows and compositions in the recycle plant.

Symbol	Description
F_0 [kmol/h]	Liquid reactor feed
z_{F0} [mol A/mol]	Mole fraction of component A in the reactor feed
F [kmol/h]	Liquid feed to the distillation column
z_F [mol A/mol]	Mole fraction of component A in the column feed
M_r [kmol/h]	Liquid holdup in the reactor
A [-]	Component A
B [-]	Component B
M_b [kmol/h]	Liquid holdup in the bottom of the column
B [kmol/h]	Bottom product
x_B [mol A/mol]	Mole fraction of component A in the bottom product
L [kmol/h]	Reflux
M_d [kmol/h]	Liquid holdup at the top of the column
D [kmol/h]	Distillate or recycle
x_D [mol A/mol]	Mole fraction of component A in the recycle

A model of the process made in Simulink is shown in Figure 7 on the next page. This drawing shows all the inputs, outputs, parameters, measurements and controllers in the recycle plant. The model is the same as was used in one of the exercises in the subject Advanced Process Control in the autumn of 2012 [19]. As seen in the model, a CSTR reactor is used for the reactor and the distillation column is the one named separator. Though the steady-state model is utilized for the major parts of this project, the dynamic model in Simulink can also be helpful to see how the process reacts to changes in disturbances or implementation errors.

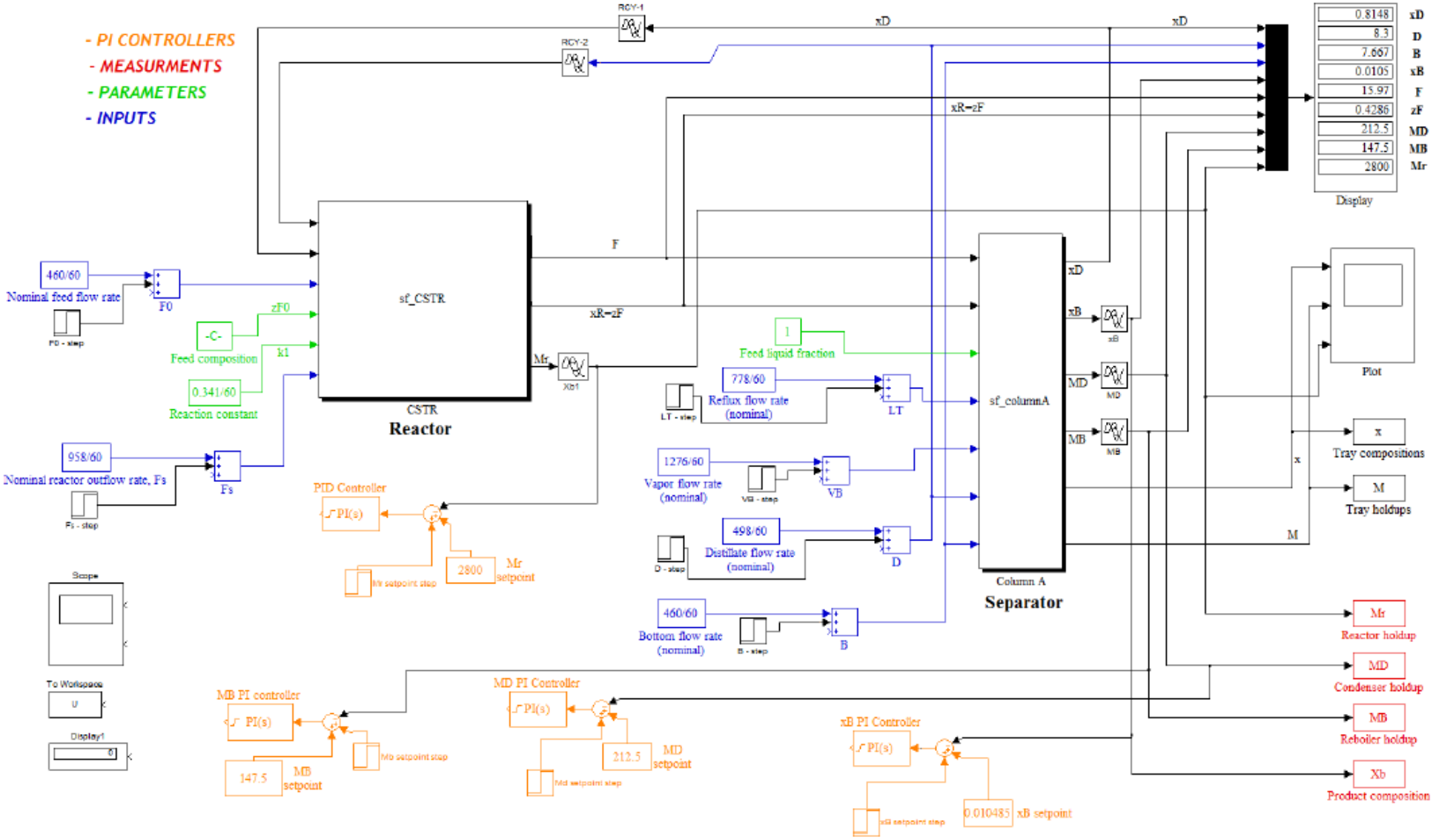


Figure 7: Flow sheet of the reactor/recycle process in Simulink [19].

3.1. Plant Data

For simplicity, the design and plant data used here is taken from the work by Larsson et al. (2003) [2]. The data and assumptions for the plant are given in Table 2.

Table 2: Plant data and design information [2].

Factor	Value/definition
Feed composition	0.9 mol A/mol
Feed F_0	460 kmol/h
Reactor type	Isothermal
Maximum reactor holdup	$M_{r,max}$ = 2800 kmol
Reaction in reactor	1. order reaction
Reaction rate constant	k = 0.341 h^{-1}
Number of stages in column	22 (including reboiler and condenser)
Feed stage in column	13
Relative volatility	α_{AB} = 2 (constant)
Flows	Constant molar flows
Purity requirement for product	$x_B \leq 0.0105$ mol A/mol

As seen above, both the feed composition and feed flow are given. That means the reactor composition z_F cannot be specified independently.

3.2. Model Equations for the Process

The reaction in the isothermal reactor is a first order reaction as assumed by Larsson et al. (2003), and thus has the following reaction rate [2]:

$$r = kc_A c_B \tag{37}$$

The concentrations of component A and B at the reactor outlet are represented by c_A and c_B, respectively. By studying the process in Figure 6, the steady-state overall total mass balance can be written as

$$F_0 = B \tag{38}$$

For component A, the overall mass balance becomes

$$F_0 z_{F0} = B x_B - r_A \tag{39}$$

The first order rate equation for component A is given by

$$r_A = kc_A \tag{40}$$

Here, c_A is the concentration of component A at the reactor outlet. This concentration is given by

$$c_A = M_r z_F \tag{41}$$

By inserting equation (40) and equation (41) into equation (39), the following overall mass balance for component A is obtained:

$$F_0 z_{F0} = B x_B - k M_r z_F \tag{42}$$

Considering only the distillation column, the total mass balance becomes

$$\mathrm{F} = \mathrm{B} + \mathrm{D} \tag{43}$$

For component A, the column mass balance is as follows:

$$F z_F = B x_B + D x_D \tag{44}$$

These mass balances were the basis for the model of the column and reactor in Matlab.

The temperature at each stage of the distillation column can be approximated in terms of the composition at the individual stages as follows :

$$T_J = \sum_{i=1}^{NC} T_{B_i} * x_{i,J} \tag{45}$$

Here, T_J is the temperature at stage number J, T_{Bi} is the boiling temperature of component i, and $x_{i,J}$ is the mole fraction of component i at stage J. Also, NC is the number of components, which in this case is two, that is component A and component B. In this project it will be assumed that equation (51) can be used for the distillation column that is studied here.

4. Skogestad's Procedure Applied to the Recycle Process

Two different cases are studied for the recycle process. In the first case, the feed rate (F_0) is given and in the second case the feed rate is a variable. It is assumed that the first constraint to become active in the last case is the vapour flow constraint, $V < V_{max}$ [2]. For simplicity and comparison, the data used is the same as in Larsson et al. (2003) [2].

4.1. Case I: Given Feed Rate

Step 1: Definition of Operational Objectives and Constraints

A scalar cost function J needs to be defined, which in case I is the difference between the value of the feed F_0 and the product B. The operational costs for recycling and distillation also need to be taken into account. Thus, the cost function is defined as

$$J = p_{F_0}F_0 - p_B B + p_D D + p_V V \tag{46}$$

From equation (38), and by defining $p_{F_0} = p_{F_0} - p_B$, equation (46) becomes

$$J = p_{F_0}F_0 + p_D D + p_V V \tag{47}$$

Since F_0 is given, and with negligible recycling cost, the objective is to minimize the boilup V. This is because the cost function is then only influenced by the distillation costs [2]. With constant energy price, the cost function is thus represented by

$$J = V \tag{48}$$

As explained in the theory chapter, the cost function in equation (48) is subject to the operational constraints for the process. With a given feed rate, there are constraints on the reactor holdup (M_r), the product composition (x_B) and on the column boilup (V). Larsson et al. (2003) found the most economical case to be when the reactor was operated with maximum liquid holdup M_r. This is because it may not be desirable to change the reactor holdup during operation due to achieving the maximum per pass conversion [2]. That is way the constraint on M_r is present. Further on, the desired composition of the bottom product is x_B=0.0105. Due to the extra distillation costs of over-purifying the bottom product, there is a constraint on x_B. The equality constraint on F_0 occurs because it is used as a degree of freedom where its value is set by the operator. It is expected that the constraints on M_r and x_B will be active [2].

These constraints are given11 in the following equations, respectively.

$$M_r \leq 2800 \tag{49}$$

$$x_B \leq 0.0105 \tag{50}$$

$$F_0 = F_{0,max} = 460 \tag{51}$$

Step 2: Identify Degrees of Freedom

By using the counting method described in chapter 2.2.1, the number of valves was found to be 6. This can be seen on Figure 6. Two of these valves need to be used for control of the liquid holdups M_d and M_b. Therefore, they have no steady-state effect on the operational cost. Thus, there are 4 degrees of freedom at steady-state, with F_0 included. The most important disturbance is the feed rate, F_0, and ±20% changes will be considered. Disturbance in the feed composition (z_{F0}) will also be considered, and the magnitude of the disturbance will be z_{F0} = 0.9 ± 0.1. In addition, implementation errors in the candidate variables, the bottom composition (x_B) and the reactor holdup (M_r) are analysed. The magnitude of these are ±20%, ±0.002 and +-400 kmol/h, respectively. These are all the same disturbances as found in Larsson et al. (2003), and this was chosen due to ability for comparison of results [2].

Step 3: Implementation of Optimal Operation

Since the active constraints were found to be the feed rate to the reactor (F_0), the bottom composition (x_B) and the liquid reactor holdup (M_r), these three variables will be the primary controlled variables (CV_1). According to Skogestad's procedure, the remaining unconstrained DOF should be used to control self-optimizing variables. In this case there are four steady-state degrees of freedom and three active constraints, which leaves one unconstrained DOF left to be utilized. In order to decide which variable should be the self-optimizing variable, simulations had to be performed. These simulations and results are described in chapter 5.

Step 4: Where to Set the Production Rate

In this process, the TPM will be the feed rate F_0 because this flow determines the amount of mass going through the plant. Thus, the production rate is determined by defining F_0.

Step 5: Regulatory Control Layer

After the previous steps, there are no unconstrained DOF left. However, this is because the liquid holdups in the condenser (M_d) and reboiler (M_b) must be controlled by the bottom flow (B) and recycle (D), respectively. These two control loops must be present in order to keep the system stable. Therefore, the liquid holdups M_d and M_b are considered as secondary controlled variables which helps stabilize the process. Also, both M_d and M_b satisfy the rules

for a good controlled variable given in chapter 2.2.2. That is, they are both assumed to be easily measured and easy to control by using the manipulated variables B and D.

Step 6: Supervisory Control Layer

In order to control the active constraints, the manipulated variables must be used. The reactor holdup (M_r) will be controlled by the reactor outflow (F), the product composition (x_B) will be controlled by the vapour boilup V, and F_0 is set. These pairings were chosen because the manipulated variable should be located close to the controlled variable which it is linked to. A

Step 7: Optimization Layer

In the case of a change in the active constraint regions, the controlled variables will have to be switched. Also, the set-points of the current controlled variables may have to be changed according to the different disturbances.

4.2. Case II: Maximize the Feed Rate

Step 1: Definition of Operational Objectives and Constraints

In the case of maximizing the feed rate, the feed rate will now be a variable and not chosen. Small losses in the production rate often have a large effect on the overall plant economics, so this case may have more practical importance than the previous.

As before, the original cost function is given in equation (46). Unlike the first case, the goal here is to maximize the production rate, and thus maximize the feed rate F_0. It is also assumed that the price of F_0 will be constant for this analysis. The cost function therefore becomes

$$J = -F_0 \tag{52}$$

As before, both the product composition (x_B) and the reactor holdup (M_r) are active constraints. In addition, the vapour boilup will be constrained since it is assumed to be the first constraint to become active. The actual value of the maximum boilup (V_{max}) is the most important disturbance in addition to the feed composition (z_{F0}). The considered magnitudes of these disturbances will be ±300 kmol/h and ±0.1, respectively. Also, implementation errors in the candidate controlled variables, the reactor boilup (M_r) and the product composition (x_B) are considered. The magnitudes of the implementation errors are ±20%, ±400 kmol/h and ±0.02, respectively.

Step 2: Identify Degrees of Freedom

Since the process is basically the same as in the first case, the number of degrees of freedom is the same. Thus, there are four degrees of freedom including the vapour boilup.

Step 3: Implementation of Optimal Operation

Using the same reasoning as for case I, the primary controlled variables are the vapour boilup (V), the bottom composition (x_B) and the liquid reactor holdup (M_r). One unconstrained degree of freedom remains after this, and this will be used to control a self-optimizing variable. The procedure for finding this variable and the result is given in chapter 5.

Step 4: Where to Set the Production Rate

The vapour boilup will be the "gas pedal" of the plant in case II. This is because when the vapour boilup is determined, it determines the amount of mass going through the plant.

Step 5: Regulatory Control Layer

Again, the liquid holdups in the condenser (M_d) and reboiler (M_b) must be controlled by the bottom flow (B) and recycle (D), respectively. These two control loops must be present in order to keep the system stable. Therefore, the liquid holdups M_d and M_b are considered as secondary controlled variables which helps stabilize the process. Also, both M_d and M_b satisfy the rules for a good controlled variable given in chapter 2.2.2. That is, they are both assumed to be easily measured and easy to control by using the manipulated variables B and D.

Step 6: Supervisory Control Layer

A possible way to control the active constraints is given here. As earlier, the condenser level and reboiler level could be controlled by the recycle flow D and bottom flow B, respectively. Since F_0 is now a degree of freedom, it can be used to control the reactor holdup (M_r), whereas the bottom product composition (x_B) can be controlled by the reactor outflow, F.

Step 7: Optimization Layer

In the case of a change in the active constraint regions, the controlled variables will have to be switched. Also, the set-points of the current controlled variables may have to be changed according to the different disturbances. The procedure here follows the same principles for both cases.

5. Simulation Procedure

5.1. Model of the Column and Reactor

A Matlab model of the reactor and distillation column had to be created. These models were based on column A in Skogestad and Postlethwaite (2007) [20]. The model of the column is a nonlinear steady-state model and assumes binary separation, constant relative volatility, no vapour holdup, one feed and two products, constant molar flows, and a total condenser at the top. All the inputs, outputs and mass balance equations are found in these models. The models are called *colamodSS.m* and *CSTR_SS_model.m*, respectively, and are given Appendix A.

5.2. Optimization of the Process

The optimization is a minimization process where the objective is to minimize a cost function. Thus, a function called fmincon in Matlab was used for the optimization. Fmincon finds the minimum of a constrained nonlinear multivariable function [21]. As explained before, the optimization was performed on both the cases studied for the recycle process. The scripts used for the optimization are named *fun.m*, *nlcon.m* and *testScriptOpt.m*. They are also given in Appendix A.

These last three scripts have the optimization of both cases included, so it is easy to change between the two by changing one parameter. *Fun.m* contains the function that is to be minimized in each case, while *nlcon.m* contains the nonlinear equality and inequality constraints for the optimization problem. Further on, *testScriptOpt.m* is where the process is optimized, subject to the specific constraints. Initial values for all the optimization variables also had to be included here in order for fmincon to solve the minimization problem.

For simplicity, global parameters were used in some of the Matlab scripts, and state variables x were defined where many of the variables were included. In total there were 30 x-parameters. The first 22 x-values are the compositions at the 22 distillation column stages, starting from the bottom. So the 22nd stage is the condenser at the top of the column. The 8 last x-values are flows and compositions. This is summarized in the following equations:

$$x[1 - 22(p.NT)] = (x_B, x_2, x_3, \dots, x_D) \tag{53}$$

Here, p.NT is the total number of stages, which is equal to 22.

$$x[23(p.NT + 1) - 30(p.NT + 8)] = (L, V, D, B, F, z_F, M_r, F_0) \tag{54}$$

This means that for example the column feed, F, can be written as x(27)=x(p.NT+5). The same holds for the other variables in equation (54).

5.3. Identification of Candidate Controlled Variables for Self-Optimizing Control

For simplicity and comparison, the candidate controlled variables for self-optimizing control used in Larsson et al. (2003) will also be used in this project [2]. These are F, L, D, L/D, L/F, L/V, x_D, F/F_0 and D/F_0. Some variables are not included in the candidate controlled variables considered, namely the reactor composition z_F and the boilup V. This is because the reactor composition is dependent on other variables so it is not a candidate for control. The boilup V is not included because specifying it below its minimum value (optimum) value results in infeasible operation in case I. In addition, V is an active constraint in case II. The same candidate controlled variables are studied for case I and case II.

5.4. Evaluation of the Loss From the Steady-State Model

In order to calculate the loss with self-optimizing control, the candidate controlled variables and disturbances have to be defined. These are summarized in Table 3 and Table 4 for case I and case II. The disturbances and implementation errors are deviations from the optimal values found by the optimization in chapter 5.2.

Table 3: Suggested candidate controlled variables for the two cases.

Candidate Controlled Variables	
Case I	Case II
F	F
L	L
D	D
L/D	L/D
L/F	L/F
L/V	L/V
x_D	x_D
F/F_0	F/F_0
F/F_0	F/F_0

Table 4: Disturbances and implementation errors for the two cases.

Disturbances/implementation errors			
Case I		Case II	
Variable	Deviation	Variable	Deviation
$F_{0,max}$	±20%	V_{max}	±20%
z_{F0}	±0.1	z_{F0}	±0.1
c	±20%	c	±20%
M_r	±400	M_r	±400
x_B	±0.002	x_B	±0.002

The choice of candidate controlled variables in Table 3 is justified by looking at the rules for a good candidate controlled variable in chapter 2.3. The first rules is satisfied because all of the suggested candidate variables are assumed to be easily measured and controlled if necessary. Further on, the second and third rule are not explored in detail in this work. However, since

Larsson et al. (2003) used the same candidate controlled variables, the two last rules are assumed to be fulfilled as well [2].

The loss was computed with three different methods as explained in the next paragraphs. It can be calculated by equation (6). However, it is desired to represent the loss in percentage, thus the loss was found by equation (55).

$$Loss = \frac{J_u(u,d) - J_{opt}(u,d)}{J_{opt}(u,d)} * 100\% \tag{55}$$

5.4.1. Brute Force Method

The loss was calculated by the Brute force method described in chapter 2.3.1. Thus, the loss for each candidate controlled variable was found by keeping this variable constant at its optimal value and solving the problem again for different values of the disturbance. This was computed for all the nine different candidate controlled variables listed in Table 3, and for all the disturbances and implementation errors mentioned in Table 4. After finding the loss, it was plotted against the disturbances in order to see which variables had the smallest and largest losses. The script for calculation of the cost with the Brute Force method is the one named *testScript_BF.m*, and is given in Appendix A

The plots of the losses were made in *LossBF1.m* and *LossBF2.m* for case I and case II, respectively. These scripts are given in Appendix A

5.4.2. Null Space Method

In order to use the null space method, a measurement vector was needed according to equation (10). This was defined as follows:

$$\boldsymbol{y} = [x_B\ x_6\ x_{10}\ x_{14}\ x_{18}\ x_D\ L\ V\ D\ B\ F\ F_0] \tag{56}$$

Here, x_B is the bottom composition, $x_{6\text{-}10}$ are the compositions at the different stages in the column and x_D is the recycle composition. The flows L, V, D, B, F, and F0 are the reflux, vapour boilup, recycle, bottom, column feed, and rector feed flow, respectively. Temperatures could also be used as measurements instead of the compositions, but in this case the compositions were used for simplicity. There are 12 measurements in total, and the goal was to find a measurement combination to be held constant, as in equation (10).

The first step was to find the optimal sensitivity matrix, F. It was found by making a small step of 1% in the disturbances, followed by calculating the new optimal measurement y. Two disturbances were present for both case I and case II, namely F_0 and z_{F0}, and V_{max} and z_{F0}, respectively. This procedure was performed in Matlab, and the script that was used is named *testScript_NP_EL.m*. This is also given in Appendix A

In addition, the criterion in equation (12) is satisfied for both cases because the number of measurements is 12, the number of unconstrained DOF is one, and the number of disturbances is two. This is summarized in the following equations.

$$n_y = 12 \tag{57}$$

$$n_u = 1 \tag{58}$$

$$n_d = 2 \tag{59}$$

Thus,

$$12 \geq 1 + 2 = 3 \tag{60}$$

After the measurement combination was found, this was held constant by adding it as an equality constraint in the Matlab script named nlcon.m. Then the problem was solved with the measurement combination kept constant for different disturbances. The loss with using the null space method was found in the same manner as for the Brute force method. The equality constraint that had to be added to the script *nlcon.m* is as follows:

$$\boldsymbol{Hy} - \boldsymbol{Hy_0} = 0 \tag{61}$$

As before, **H** is the measurement combination matrix, **y** is the measurement vector, and $\mathbf{y_0}$ is the nominal measurement vector without disturbances. Therefore, $\mathbf{Hy_0}$ is the set point of the measurement combination **c.** Thus, equation (65) can be written in the following manner:

$$\mathbf{c} - \mathbf{c_s} = 0 \tag{62}$$

5.4.3. Exact Local Method

When adding the implementation and measurement errors, the scaling matrices $\mathbf{W}_d$ and $\mathbf{W}_{n^y}$ had to be defined. These represent the magnitudes of the disturbances and measurement errors. For the measurement error, a change of 0.01 was used for the compositions, while a change of 2% was used for the measured flows. Thus, $\mathbf{W}_{n^y}$ is a n_y x n_y matrix with the diagonal being a vector with the different measurement errors:

$$\boldsymbol{W}_{n^y} = diag[0.01 * ones(1,6)\ 0.02 * y_0(7:12)\ ] \tag{63}$$

Here, y_0 is the optimal values of the measurements. Further on, the matrix in equation (63) will be the same for both cases due to the measurements being equal. The first six elements in equation (63) are the measurement errors that are present for the compositions, while the last six elements are the measurements errors that occurs for the flows in the system. Thus, the magnitude of the measurement error is approximated as 0.01 for compositions (and temperatures), and as 2% of the optimal values for all flows. This assumption is made because the exact local method is only valid for small disturbances and measurement errors.

When it comes to the second scaling matrix, $\mathbf{W}_d$, it will be different for the two cases due to the different disturbances. Since this matrix represents the magnitudes of the disturbances, the matrix for case 1 will consist of a diagonal with each element equal to 1% of the optimal value of F_0. In the second case, the diagonal will consist of elements equal to 1% of the optimal value of V_{max}. Thus, for case I the disturbance scaling matrix is as follows:

$$\boldsymbol{W}_{d,I} = diag[0.01 * F0s * ones(12,1)] \tag{64}$$

In the equation above, F0s is the optimal value of the reactor feed, which is equal to 460 kmol/h. For case II, the disturbance scaling matrix becomes

$$\boldsymbol{W}_{d,II} = diag[0.01 * Vmaxs * ones(12,1)] \tag{65}$$

Here, Vmaxs is the optimal value of the vapour boilup, which is equal to 1500 kmol/h.

The F matrix is the same as the one in the null space method, but the steady-state gain matrix $\mathbf{G_y}$ needs to be computed. As explained in the theory chapter, small steps are made in the inputs u, and the new optimal y-values are found. The inputs in the first case are L, V and F, and in the second case they are L, F and F_0. Both the bottom flow B and recycle flow D is utilized for control of the liquid holdups in the reboiler and condenser, and is therefore not available as inputs.

Further on, the **Y** matrix is found by applying equation (26), and then the **H** matrix can be calculated by using equation (27).

As for the null space method, an equality constraint had to be added to *nlcon.m*. The same equation as in the null space method holds for the exact local method as well, given in equation (61) and equation (62).

Again, the script where the exact local method was applied is *testScript_NP_EL.m*, and can be found in Appendix A.

5.5. Evaluation of the Loss from the Dynamic Simulink Model

In order to actually see how the process behaves dynamically, the model in Simulink was used to find the loss in addition to the loss found from the steady-state model. This was done with both the null space method and the exact local method. The same equations as before are valid for finding the **H** matrix in the two methods, as described in chapter 2.3.

5.5.1. Measurements

One change from the steady-state calculation is that temperature measurements will be used instead of composition measurements. This is because it is much cheaper, easier and more normal to use temperatures than compositions as measurements in a real plant. The temperatures will be measured at the same stages as the compositions were before. Further on, the same flows as in the steady-state calculations will be measured. Thus, the measurement matrix **y** becomes

$$\boldsymbol{y} = [T_1\ T_6\ T_{10}\ T_{14}\ T_{18}\ T_{22}\ L\ V\ D\ B\ F\ F_0] \tag{66}$$

As explained in the theory chapter, the temperatures at the distillation stages can be calculated by equation (45). However, the temperatures are already included in the Simulink model of the process, so it is not necessary to calculate them explicitly. In the beginning, the simulations were performed with only three measurements (F_0, z_{F0} and T_{16}). According to equation (12), only three measurements are necessary. That includes two measurements because of two disturbances, in addition to one measurement because of one remaining unconstrained degree of freedom u. However, the use of three measurement resulted in infeasible simulations with high disturbances. Thus, 12 measurements were used after all since the simulations worked for higher disturbance values.

A new problem that appears now is the controller used in Simulink for calculating the loss. It needs to be tuned, and the SIMC rules will be used for this. Thus, step tests will be made in the input L, and the response in the measurement combination c will be plotted against time. This was done for the input flow L, which is the only unused manipulated variable in the dynamic model of the process. Thus, the last remaining degree of freedom is used up by L. This procedure was executed for case I only, with the null space method and the exact local method. First, the H matrix had to be found for the two methods, which was done in the script called *script_np_el_T.m*, given in Appendix A. Because these methods was considered for the steady-state problem as well, it will not be described in detail here, but the same equations were used for finding the measurement combination matrices.

5.5.2. Controller

The new controller had to be added to the Simulink model, and it is shown in Figure 8. By manipulating the last unconstrained degree of freedom L, the measurement combination c_s-c will be controlled. Here, y_n is the measurement vector, H is the measurement matrix, c is the measurement combination (c=H*y), and csns is the set-point of the measurement combination.

On the right side of Figure 8, a block called 'mux' combines several input signals into a vector. As seen in the figure, the input signals T_1, T_6, T_{10}, T_{14}, T_{18}, T_{22}, L, V, D, B, F and F_0 are combined into the measurement vector y_n. Both the set-point of c (csns) and the actual value of c are sent to the PI controller, where the signal from the controller is sent to the LT block (not shown here). Also, the counting of the distillation steps starts at the bottom, which makes T_1 the reboiler temperature and T_{22} the temperature at the top of the column.

Both the setpoint (csns) and the measurement combination matrix (H) was defined in Matlab and sent to the Simulink file. The 'goto' and 'from' blocks were used instead of arrows to pass signals between several blocks. More specific, the 'goto' block passes the block input to the 'from' block. Thus, the 'suL' block passes the controller output to the LT block. The LT block can be seen on the previous Simulink model in Figure 7.

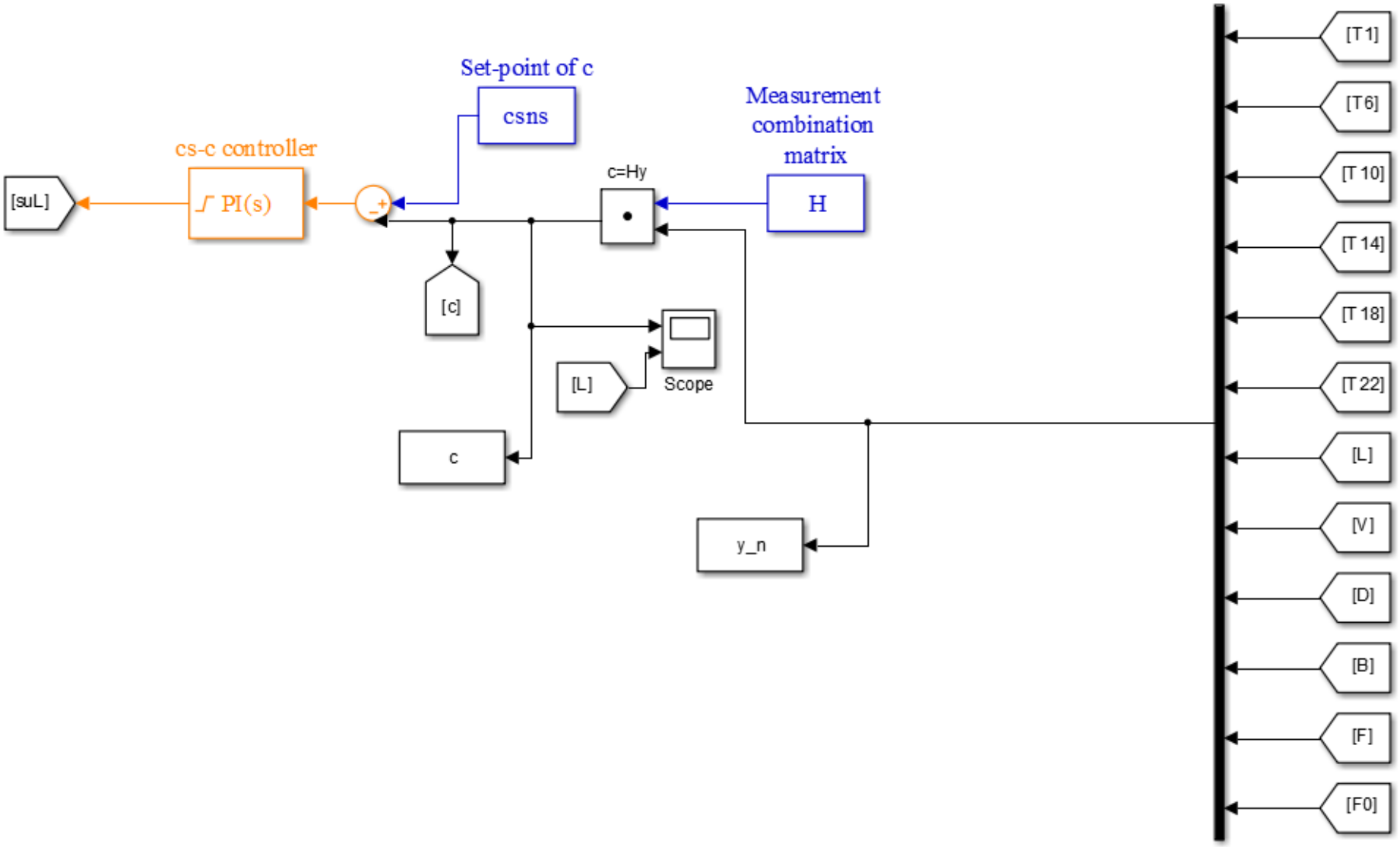


Figure 8: Simulink flow sheet showing the cs-c controller were the input L is used to control the measurement combination csns-c.

5.5.3. Step Tests and SIMC Tuning

Further on, the step test discussed above was applied to the process. This is shown in the Matlab scripts named *StepTest_NS.m* and *StepTest_EL.m* for the two methods. These scripts are also shown in Appendix A. Then the response in the measurement combination (c) was plotted against time. Since the L-step was applied at t = 500 minutes, c was plotted in the time interval t ∈ [500, 510]. This time interval was chosen because the process could be assumed to follow an integrating process response in this region.

Moving on, the tuning parameter and process time constant τ_c was set to τ_c=5 minutes in both methods. This value was chosen based on the fact that only the steady-state values are interesting, Thus, it does not matter if the controller is a bit slow. An examination of the step responses revealed a time delay of just above zero. Thus, the time delay was set to zero for simplicity. It was assumed to have little effect on the result, which will be explained further in chapter 6.5.

In order to find the tuning parameters P and I for the PI controller, the SIMC tuning rules were utilized. These are the ones described in chapter 2.4.

After finding the controller tunings for the L controller, they were used to keep the measurement combination c constant. Then the loss was found by running the simulation with the L controller operating and applying disturbances in the feed rate F_0. This was performed in the same scripts as for the step tests, which are *StepTest_NS.m* and *StepTest_EL.m*.

Further on, the loss by utilizing the null space method and exact local method in Simulink was plotted against the disturbance in F_0. These plots were made in the Matlab scripts named *LossNSsimulink.m* and *LossELsimulink.m*, respectively.

6. Results and Discussion

6.1. Optimization Results

The nominal results from the optimization of the reactor with recycle process are presented here. Table 5 shows the results for case I and case II, where the most important variables are represented.

Table 5: Nominal optimization results for case I and case II.

Variable		Case I: min V	Case II: max F_0
Feed rate	F_0 [kmol/h]	460	498
Bottom flow	B [kmol/h]	460	498
Reactor outflow	F [kmol/h]	958	1113
Vapour boilup	V [kmol/h]	1276	1500
Reflux	L [kmol/h]	778	885
Recycle/distillate	D [kmol/h]	498	615
Recycle composition	x_D [molA/mol]	0.82	0.83
Bottom composition	x_B [molA/mol]	0.0105	0.0105
Reactor composition	z_F [molA/mol]	0.43	0.46
Reactor holdup	M_r [kmol/h]	2800	2800

Comparing the results in Table 5 with the results from Larsson et al. (2003) confirms that they are the same, except from a difference of 1 kmol/h in the recycle flow in case I, and a change of 0.2 kmol/h in the feed rate in case II [2]. As expected, the active constraints on the bottom composition and reactor holdup are the same in both cases. In both of the cases, there is one unconstrained degree of freedom left because F_0 is given in case I, while V is given in case II. It is also noted that the vapour boilup V is smaller in case I than case II, which makes sense because the objective in case I was to minimize the vapour boilup. Correspondingly, the feed rate F_0 is larger in case II than case I because the goal in case II was to maximize the feed rate. Due to the different cost functions, the other variables except the constraints are slightly different for the two cases. That is, all the flows have higher values in case II compared to case I because the feed rate is maximized in case II. This, and the fact that the vapour boilup is maximized causes the other flows to increase compared to case I.

Another observation is that the feed rate F_0 and the bottom flow B are exactly the same in each of the cases. This was also expected due to the total mass balance in equation (38), which says that the reactor feed rate is always equal to the bottom flow at steady-state. Since the vapour boilup is a measure of the economic cost in case I, the optimal cost for case I is equal to V = 1276 kmol/h. Correspondingly, the optimal cost for case II is equal to $-F_0$ = -498 kmol/h because the goal is to maximize the feed rate, and thus minimize the negative feed rate.

6.2. Loss with Brute Force Method

The loss computed with the Brute force method was plotted against the disturbances and implementation errors described in chapter 5. In the following figures, these plots are given for the two different cases. The disturbances and implementation errors are in the feed rate, the feed composition, the candidate controlled variables, the reactor holdup, and in the bottom composition for case I. Case II has the same disturbances and implementation errors except of disturbance in the maximum vapour boilup instead of the reactor feed rate.

6.2.1. Loss for Case I: Given Feed Rate

Figure 9 and Figure 10 show the plots of the candidate controlled variables versus disturbances in the feed rate F_0. They separately show the graphs for the variables with relatively large losses and the variables with relatively small losses, respectively. As seen from the first plot, the variables with the highest losses are F, L, D, L/V, F/F_0 and D/F_0.

Also seen in the figures, the same values of the axes as in Larsson et al. (2003) were used in order to compare the results [2]. In addition, the same candidate variables were plotted in each of the plots. It is not surprising that disturbances in F_0 produce relatively large economic losses, due to the fact that it is the most important disturbances of the ones considered (F_0 and z_{F0}). For low values of F_0, the highest loss occurs for the reflux L, followed by the column feed (F), the recycle (D), the ratios F/F0 and D/F0, whereas the smallest loss is for L/V. On the other hand, large values of F_0 give the highest loss for F as a self-optimizing variable, followed by D, F/F_0, D/F_0, L, and L/V. Thus, the ratio L/V used for self-optimizing control yields the smallest loss of the variables with the relatively large losses.

Looking at the plot in Figure 10, it is observed that L/F, L/D and x_D all have much smaller losses than the previous variables. This is because the magnitude on the y-axis goes up to 10% on the first plot, and up to only 0.5% in the second plot. This indicates that one of the variables L/F, L/D or x_D should be controlled and used as a self-optimizing variable for disturbances in the feed rate.

Unlike the first figure, the same candidate variable has the smallest and largest loss for all values of the disturbance F_0. Self-optimizing control of L/F results in the smallest loss for the entire disturbance region, followed by the recycle composition x_D and the ratio L/D.

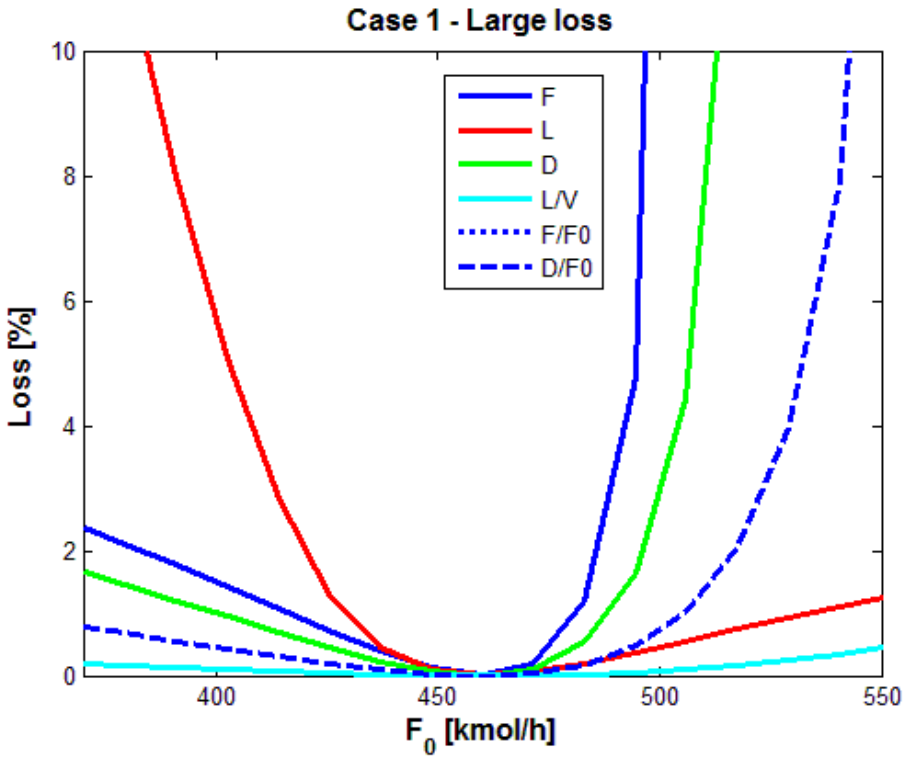


Figure 9: Loss due to disturbance in F_0 for case I, where the variables with large losses are plotted against the disturbance.

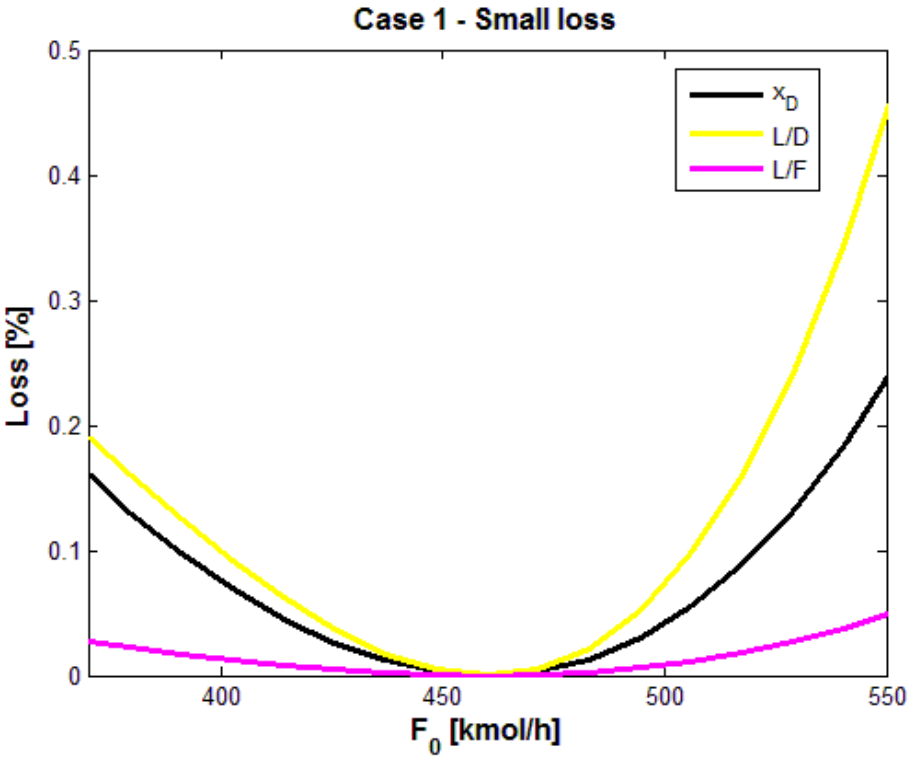


Figure 10: Loss due to disturbance in F_0 for case I, where the variables with small losses are plotted against the disturbance.

For disturbance in the feed composition, the resulting plots are shown in Figure 11 and Figure 12. As for a disturbance in the feed rate, the variables with the largest losses are F, L, D, L/V, F/F_0 and D/F_0 here as well. Thus, L/F, L/D and x_D have the smallest losses when they are kept constant at their optimal point. Again this indicates that one of the latter candidate CVs should be used as a self-optimizing variable in order to use the last unconstrained degree of freedom. Also noted is that L/F has the smallest loss of the three variables, followed by x_D and L/D. This is the same as for a disturbance in the feed rate, discussed above.

Another observation from the discussed plots is that the losses are generally larger from a disturbance in the feed rate than for a disturbance in the feed composition. This makes sense because the feed rate F_0 was expected to be the most important disturbance of the two. The reason why flow disturbances gives higher loss than composition disturbances in the feed may be because flow disturbances has a higher effect on the economics of the plant. That makes sense because the cost function to be minimized is the vapour boilup rate. So changes in the feed or production rate changes the cost value as well. Since the feed rate is considered as the throughput manipulator, it is logical to say that changes in the feed rate causes changes in the vapour boilup as well.

Further on, comparing the four first plots with the ones in Larsson et al. (2003) shows good correspondence between the plots [2]. This is probably because they were all based on a steady-state model of the recycle process.

Unlike the first plot, the losses in Figure 11 are very similar, at least for a feed composition of $z_{F0} < 0.95$. For compositions of $0.95 < z_{F0} < 1$, the loss for the flow ratios F/F_0 and D/F_0 increases compared to the other variables. The same trend as in Figure 10 is observed in Figure 12, with the main difference being the magnitude of the losses.

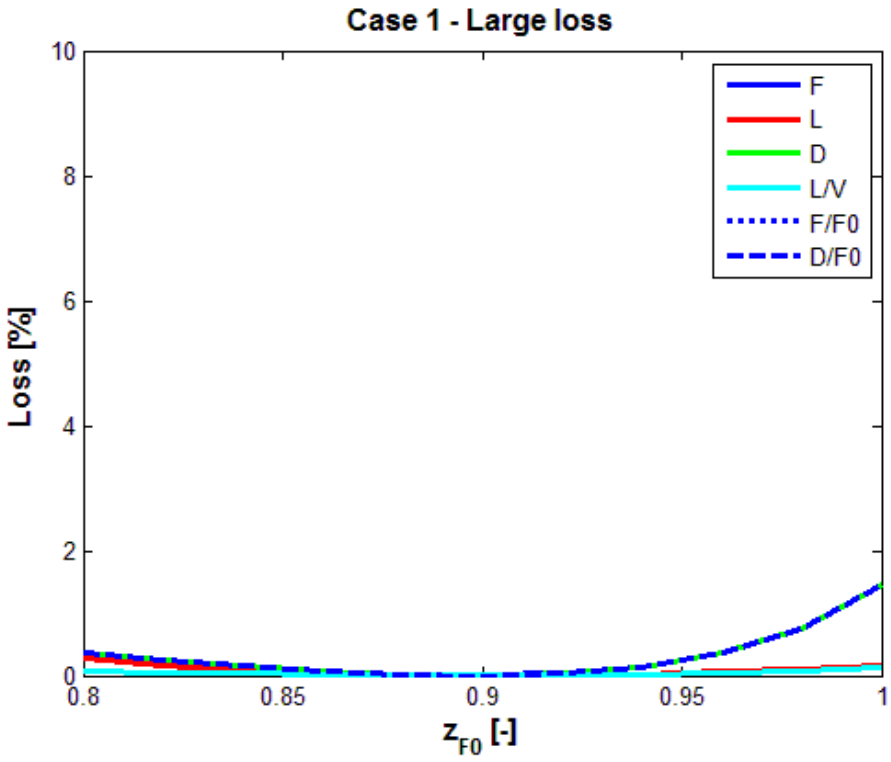


Figure 11: Loss due to disturbance in z_{F0} for case I, where the variables with large losses are plotted against the disturbance.

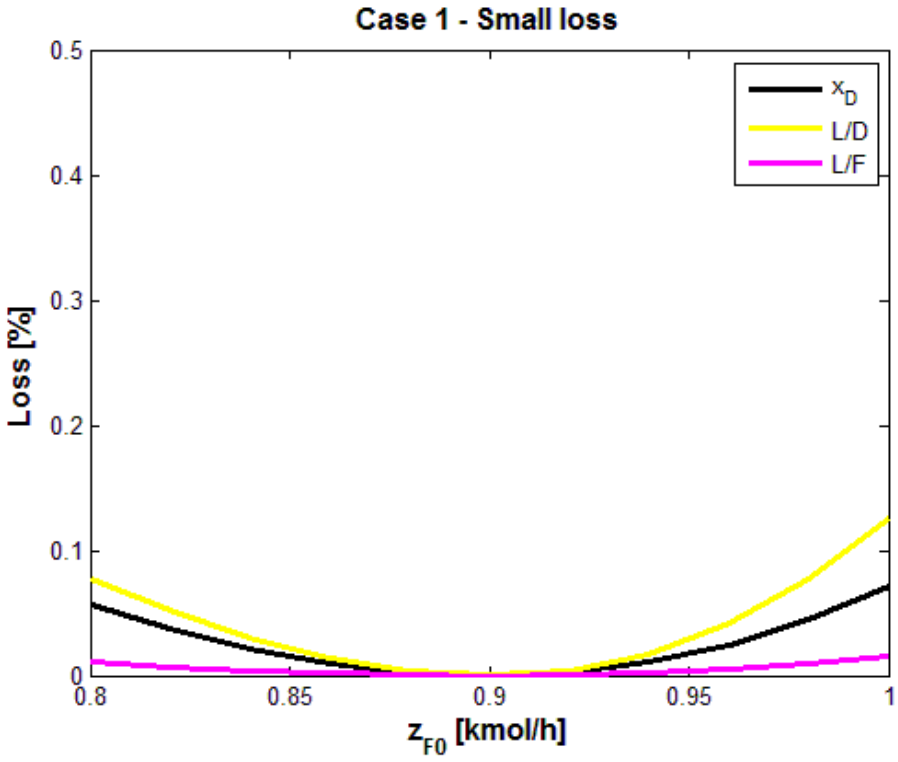


Figure 12: Loss due to disturbance in z_{F0} for case I, where the variables with small losses are plotted against the disturbance.

Three types of implementation errors were considered, namely implementation error in the candidate controlled variables, in the reactor holdup M_r, and the bottom composition x_B. The resulting losses due to these errors are plotted in the following figures.

Losses due to implementation error in the controlled variables are shown in Figure 13 and Figure 14 for large and small losses, respectively. As before, F, L, D, L/V, F/F_0 and D/F_0 as controlled variables results in the biggest losses. The largest losses of these are the ones coming from F, F/F_0 and L/V. Also, Figure 13 matches the results from Larsson et al. (2003). However, when the variables with small losses are considered, there is something strange with the loss from x_D. Comparing Figure 14 with the results in Larsson et al. (2003) gives some differences. The losses from using L/F and L/D as a self-optimizing variable are the same, but not the one from the recycle composition x_D. According to Larsson et al. (2003), the smallest loss should be produced by the recycle composition when comparing it with L/D and L/F, but in this case x_D has the largest loss of the three mentioned variables.

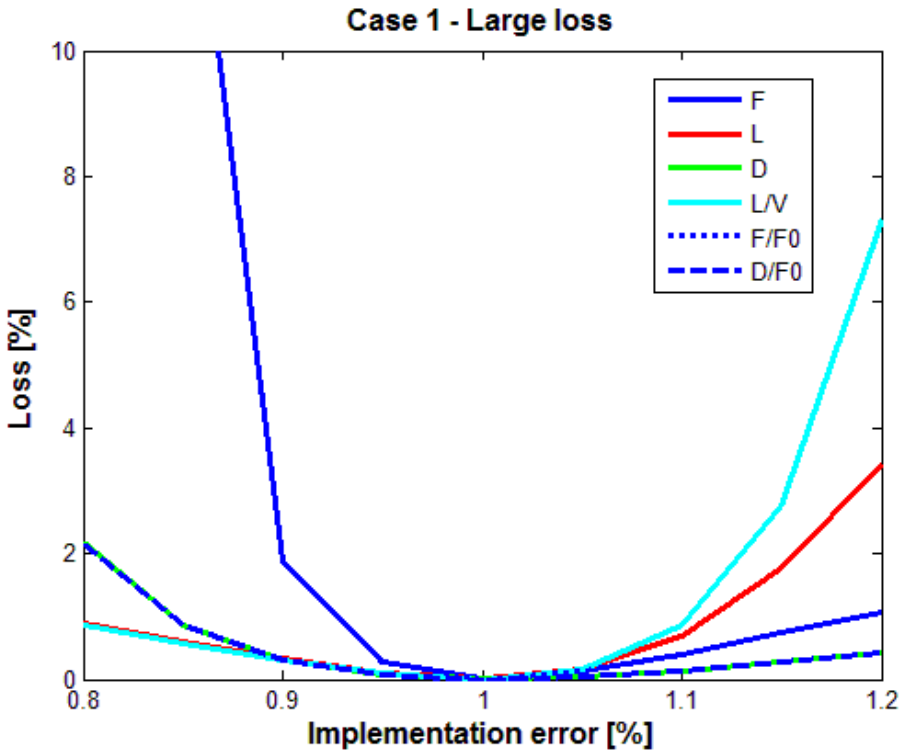


Figure 13: Loss due to implementation error in the candidate controlled variables for case I, where the variables with large losses are plotted against the implementation error. The implementation error goes from 80%-120% of the nominal value.

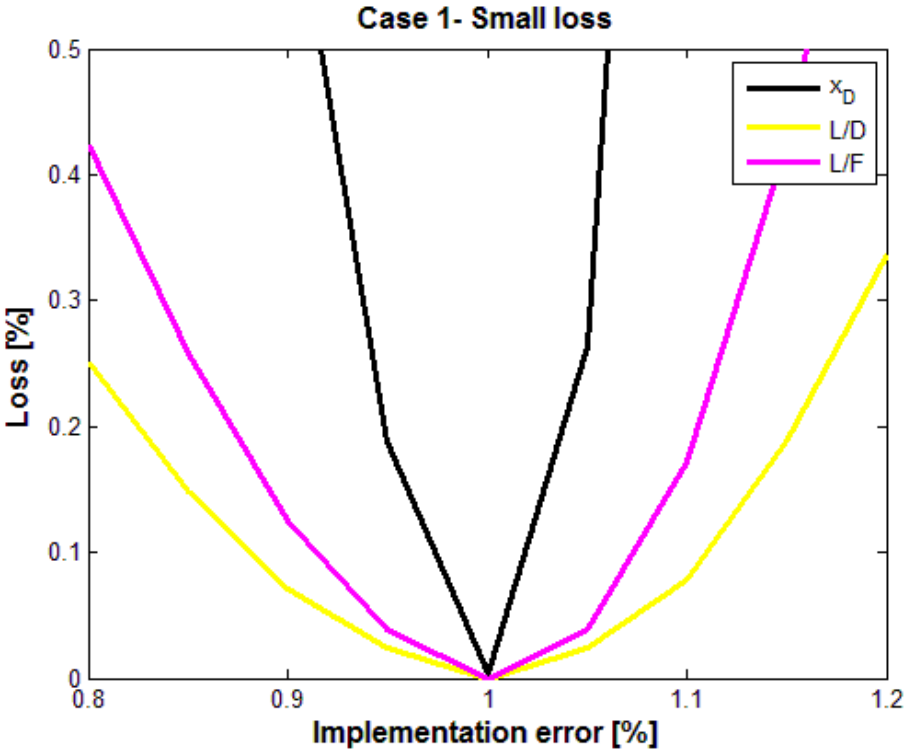


Figure 14: Loss due to implementation error in the candidate controlled variables for case I, where the variables with small losses are plotted against the implementation error. The implementation error goes from 80%-120% of the nominal value.

Moving on to implementation error in the reactor holdup in Figure 15, it is seen that D, F, L/V, F/F_0 and D/F_0 have relatively higher losses than the remaining variables. Also, it is infeasible to have a liquid holdup in the reactor above 2800 kmol/h. This is because the maximum possible liquid holdup in the reactor is 2800 kmol/h, as found in Larsson et al. (2003) [2]. There was therefore no point in doing simulations or optimization with $M_r > 2800$ kmol/h.

As a consequence of this result, implementation errors in M_r are not very dangerous for the variables with small losses. It should of course be avoided if possible.

Comparing Figure 15 with the corresponding figure in Larsson et al. (2003) reveals some differences. The most prominent is the differences in magnitude of the losses. In the figure made here, the magnitude of the loss is from 0-10%, while it goes from 0-100% in Larsson et al. (2003) [2]. Since the previous comparisons of loss plots matched each other, the method for calculating the losses are most likely the same. Thus, there must be something else causing the difference. One suggestion regards the reactor models. Maybe there exist some differences in the Matlab model of the reactor which influences the loss here.

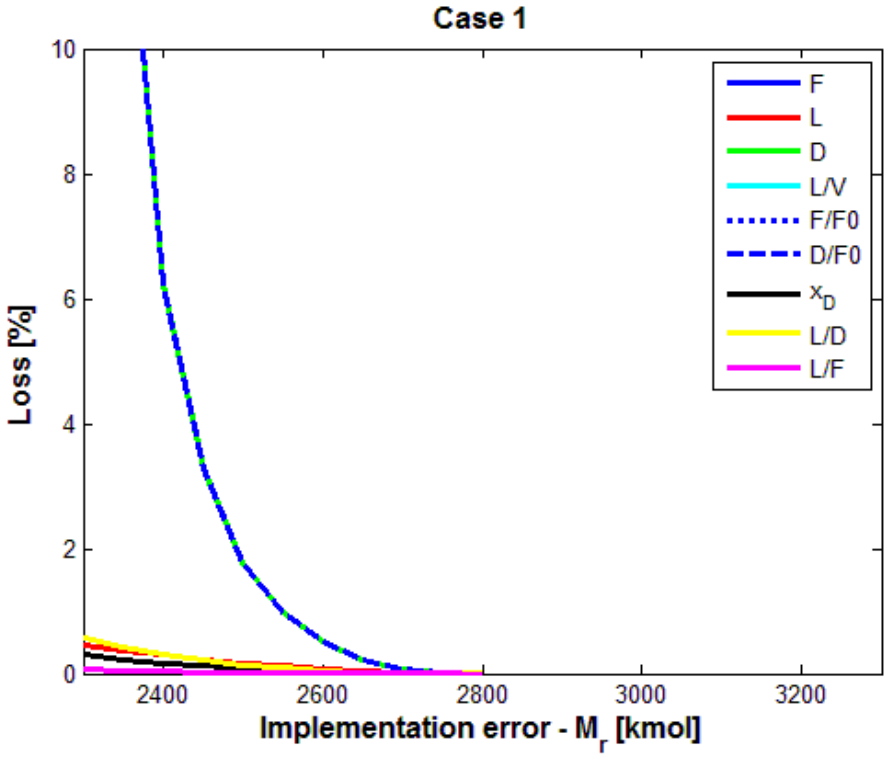


Figure 15: Loss due to implementation error in the reactor holdup (M_r) for case I, where all the candidate controlled variables are plotted against M_r. The process becomes infeasible when M_r>2800 kmol/h, so the loss is only computed for values below this.

The last implementation error considered was in the bottom composition x_B, and the loss from this is plotted in Figure 16. Here all the candidate variables have similar losses for different values of the implementation error. As for the previous figure, the process becomes infeasible for x_B-values above 0.0105 because this is the desired value of the bottom composition, and there is no point in over-purifying the product.

The same observation is made here as for implementation error in the reactor holdup. That is, the magnitude of the loss is much smaller than found by Larsson et al. (2003). The magnitude in Figure 16 goes up to 0.5%, while in Larsson et al. (2003) it goes up to 10%.

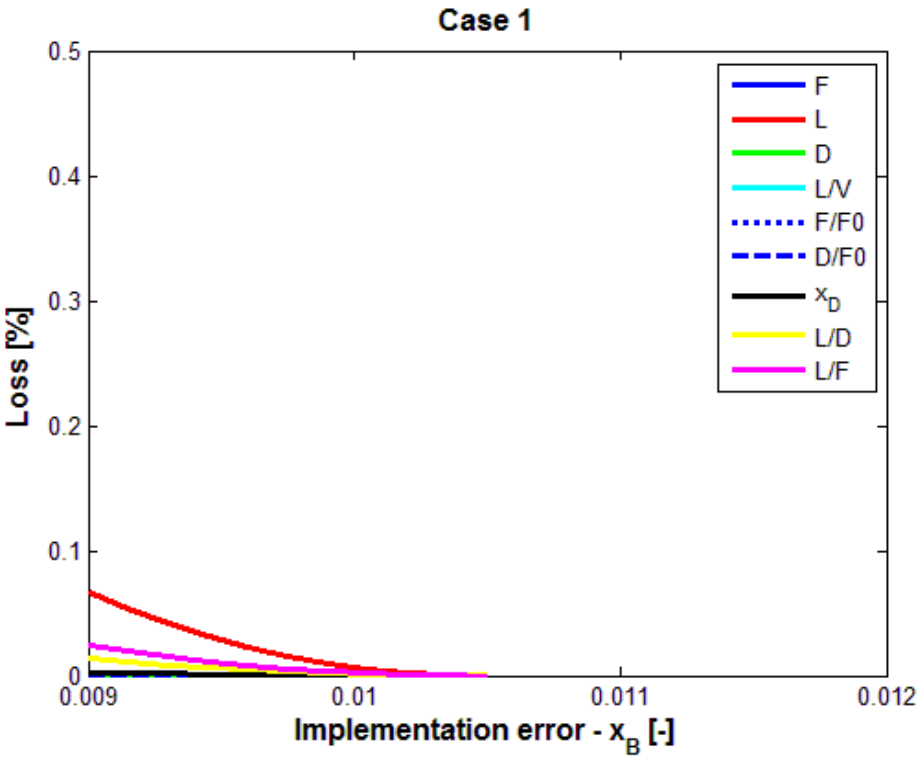


Figure 16: Loss due to implementation error in the reactor holdup (x_B) for case I, where all the candidate controlled variables are plotted against x_B. The process becomes infeasible when x_B>0.0105, so the loss is only computed for values below this.

6.2.2. Loss for Case II: Maximize the Feed Rate

Figure 17 and Figure 18 show the plots of the candidate controlled variables versus disturbances in the vapour boilup V_{max}. They separately show the graphs for the variables with relatively large losses and the variables with relatively small losses, respectively. As seen from the first of the plots, the candidate variables with the highest losses are F, L, D, F/F_0 and D/F_0.

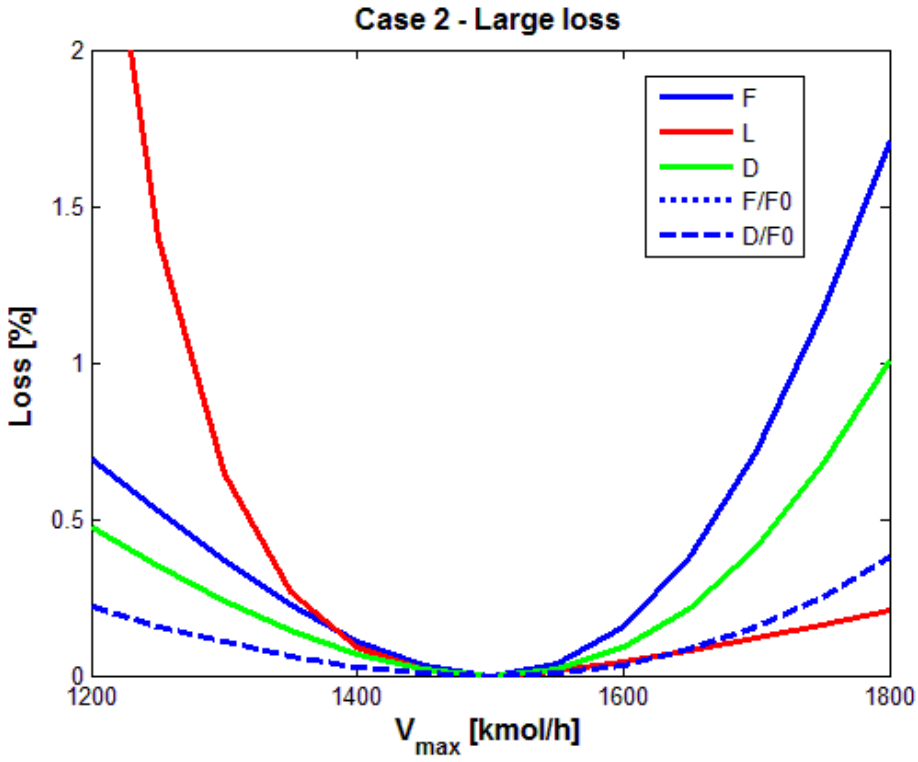


Figure 17: Loss due to disturbance in V_{max} for case II, where the variables with large losses are plotted against the disturbance.

Looking at the plot in Figure 18, it is observed that L/F, L/D, L/V and x_D all have much smaller losses than the previous variables. This is seen because the magnitude on the y-axis is up to 2% on the first plot, and goes up to only 0.2% in the second plot. This indicates that one of the variables L/F, L/D, L/V or x_D should be controlled and used as a self-optimizing variable for disturbances in the vapour boilup. It is also observed that the flow ratio L/F gives the lowest loss overall in both of the plots. Also, the yellow line for L/D cannot be seen in the graph because it has the same losses for all values of V_{max}. That means it is hidden behind the turquoise line that is L/V.

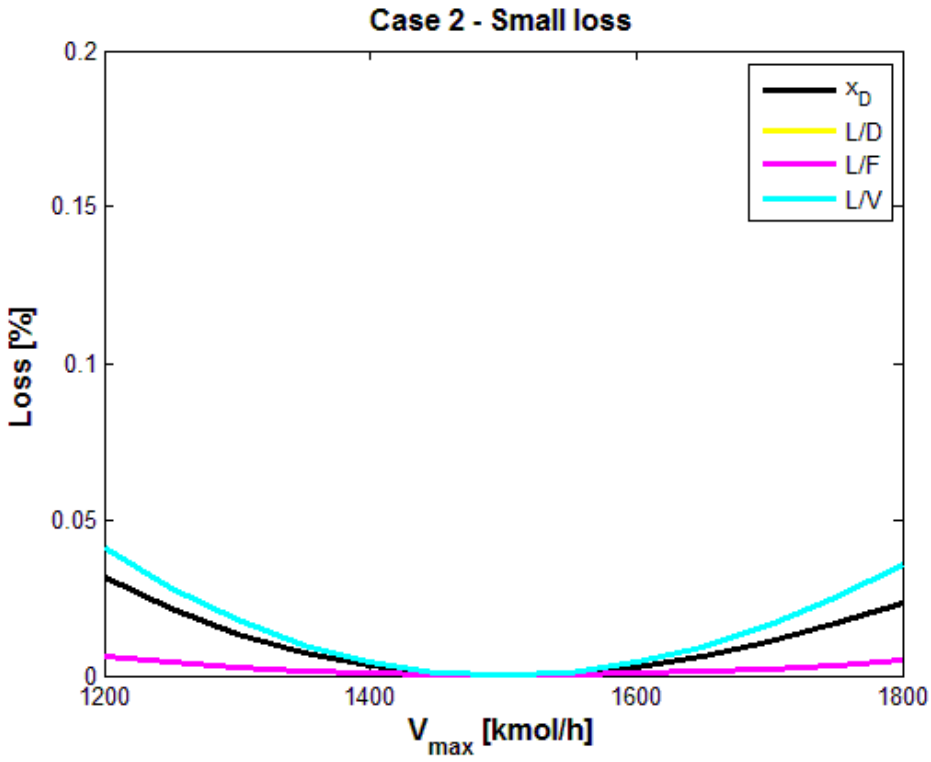


Figure 18: Loss due to disturbance in V_{max} for case II, where the variables with small losses are plotted against the disturbance. L/D has the same loss as L/V, that is way it cannot be seen.

For disturbances in the feed composition, the resulting plots are shown in Figure 19 and Figure 20. As for a disturbance in the vapour boilup, the variables with the largest losses are F, L, D, F/F_0 and D/F_0. Thus, L/F, L/D, L/V and x_D have the smallest losses when they are kept constant at their optimal point. Again this indicates that one of the latter candidate CVs should be used as a self-optimizing variable in order to use the last unconstrained degree of freedom. Also noted is that L/F represents the smallest loss of the three variables, followed by x_D, L/D and L/V. This is true also for a disturbance in the vapour boilup.

Another observation from the discussed plots is that the losses are generally larger from a disturbance in the vapour boilup than for a disturbance in the feed composition. This makes sense because the vapour boilup V was expected to be the most important disturbance for the case of maximum production.

Further on, comparing the four first plots with the ones in Larsson et al. (2003) shows good correspondence between the plots [2]. This is probably because they were all based on a steady-state model of the recycle process. The reason why flow disturbances gives higher loss than composition disturbances in the feed may be because flow disturbances has a higher effect on the economics of the plant. That makes sense because the cost function to be minimized is the negative feed rate. So changes in the vapour boilup rate changes the cost value as well. Due to the fact that V is assumed to be the gas pedal of the process, it is logical that changes in V will cause changes in the feed rate and production rate.

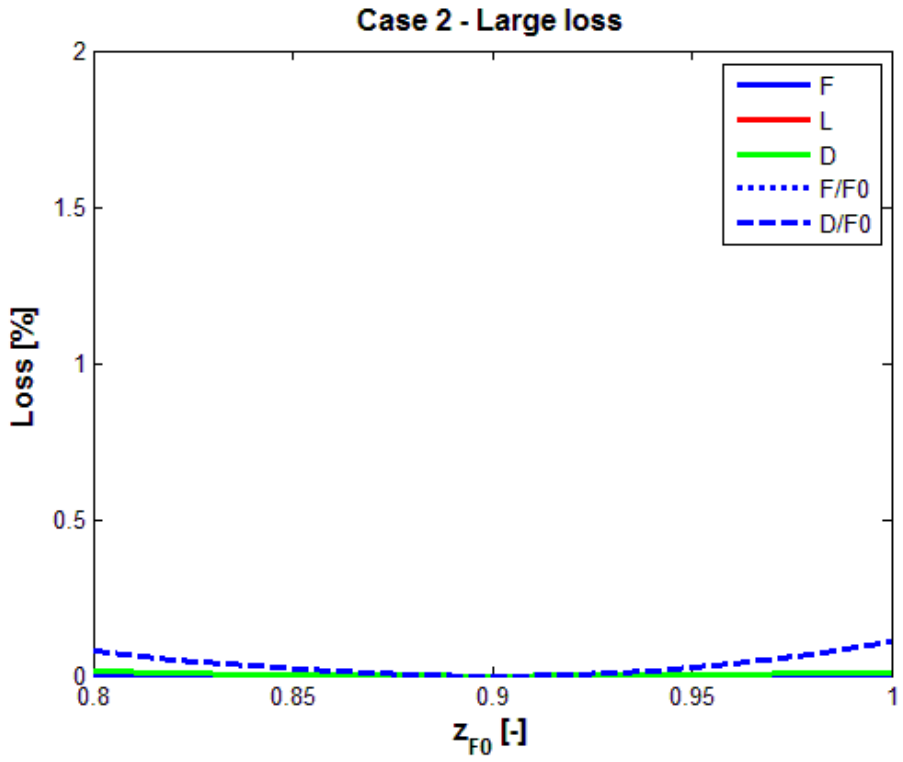


Figure 19: Loss due to disturbance in z_{F0} for case II, where the variables with large losses are plotted against the disturbance.

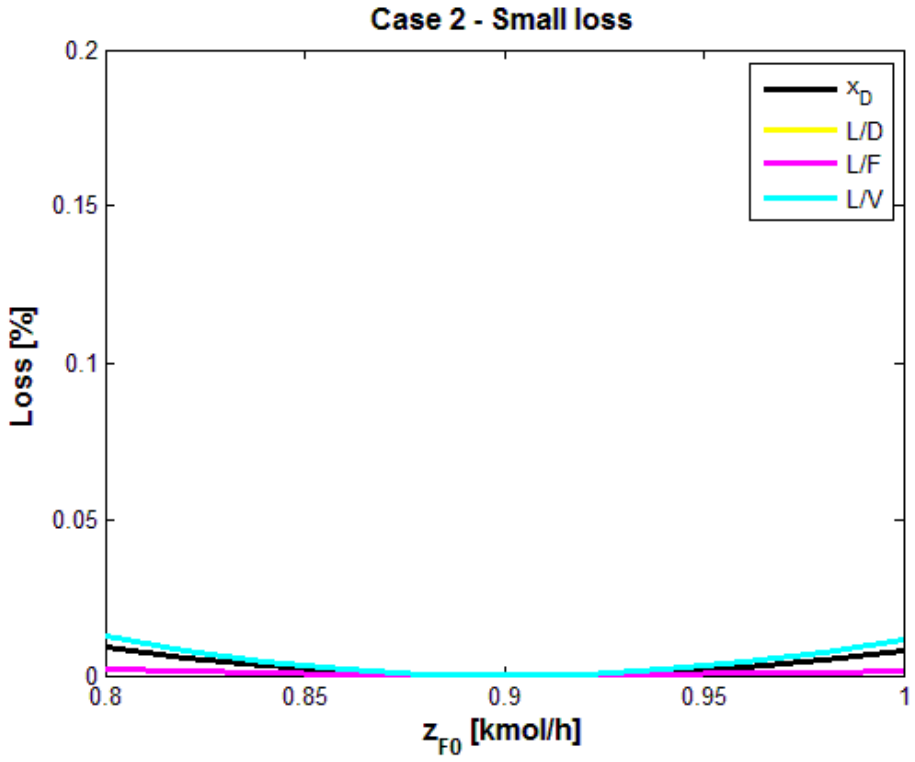


Figure 20: Loss due to disturbance in z_{F0} for case II, where the variables with small losses are plotted against the disturbance. L/V and L/D has the same loss, therefore only L/V is visible.

Three types of implementation errors were considered, namely implementation error in the candidate controlled variables, in the reactor holdup M_r and the bottom composition x_B. The resulting losses due to these errors are plotted in the following figures. Losses due to implementation error in the controlled variables are shown in Figure 21 and Figure 22 for large and small losses, respectively. As before, F, L, D, L/V, F/F_0 and D/F_0 as controlled variables results in the biggest losses. The largest of these losses are the ones resulting from F, F/F_0 and L/V. Also, Figure 21 matches the results from Larsson et al. (2003).

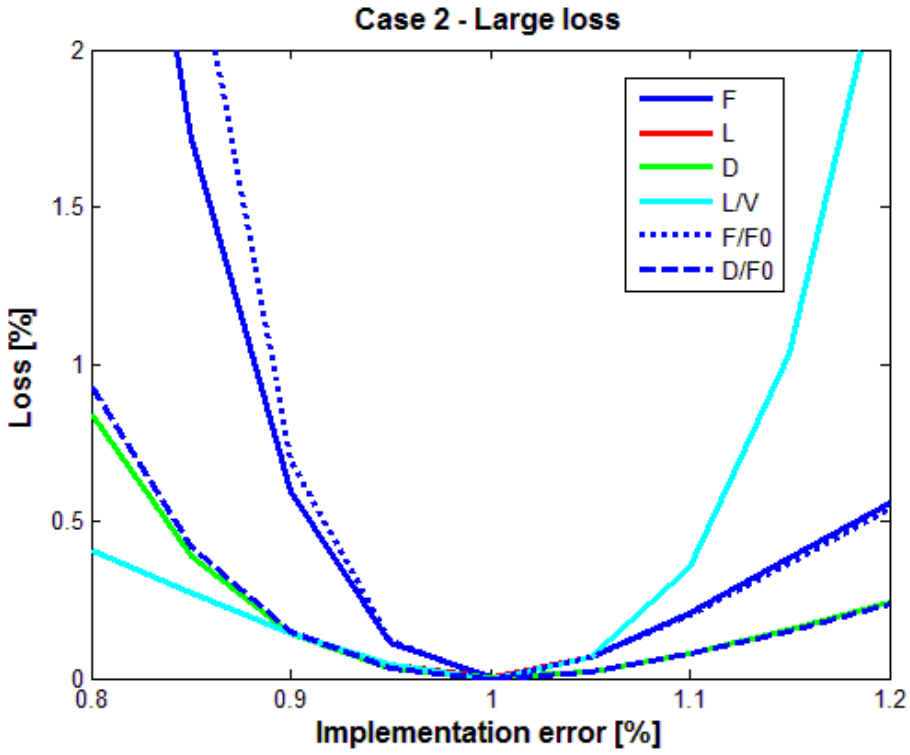


Figure 21: Loss due to implementation error in the candidate controlled variables for case II, where the variables with large losses are plotted as a function of the implementation error. The implementation error goes from 80%-120% of the nominal value.

However, when the variables with small losses are considered, there is something strange with the loss coming from keeping x_D constant at its optimal value. Comparing Figure 22 with the results in Larsson et al. (2003) gives some differences. The losses from using L/F and L/D as a self-optimizing variable are the same, but not the one from the recycle composition x_D. According to Larsson et al. (2003), the smallest loss should be produced by the recycle composition when comparing it with L/D and L/F, but in this case x_D has the largest loss of the three mentioned variables. This is the same phenomenon that occurred in case I as discussed above. One reason for this can be errors in the calculation of the loss only for x_D, but not for the other variables.

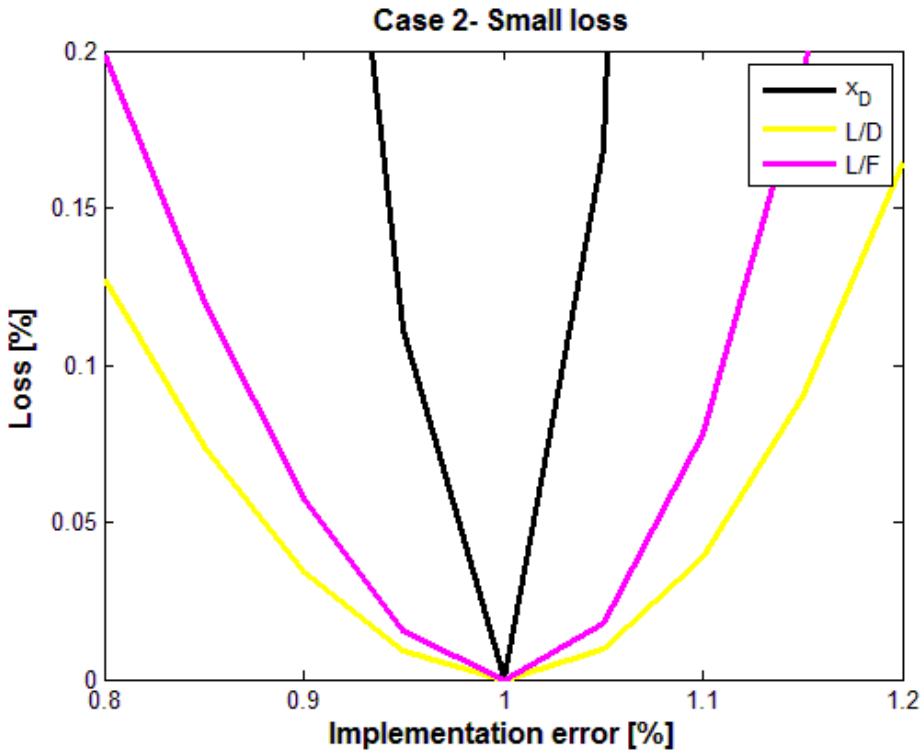


Figure 22: Loss due to implementation error in the candidate controlled variables for case II, where the variables with small losses are plotted as a function of the implementation error. The implementation error goes from 80%-120% of the nominal value.

Moving on to implementation error in the reactor holdup in Figure 23, it is seen that the candidate controlled variables differ in the magnitudes of the loss. Also, it is infeasible to have a liquid holdup in the reactor above 2800 kmol/h. This is due to the maximum possible reactor holdup, which is equal to 2800 kmol/h.

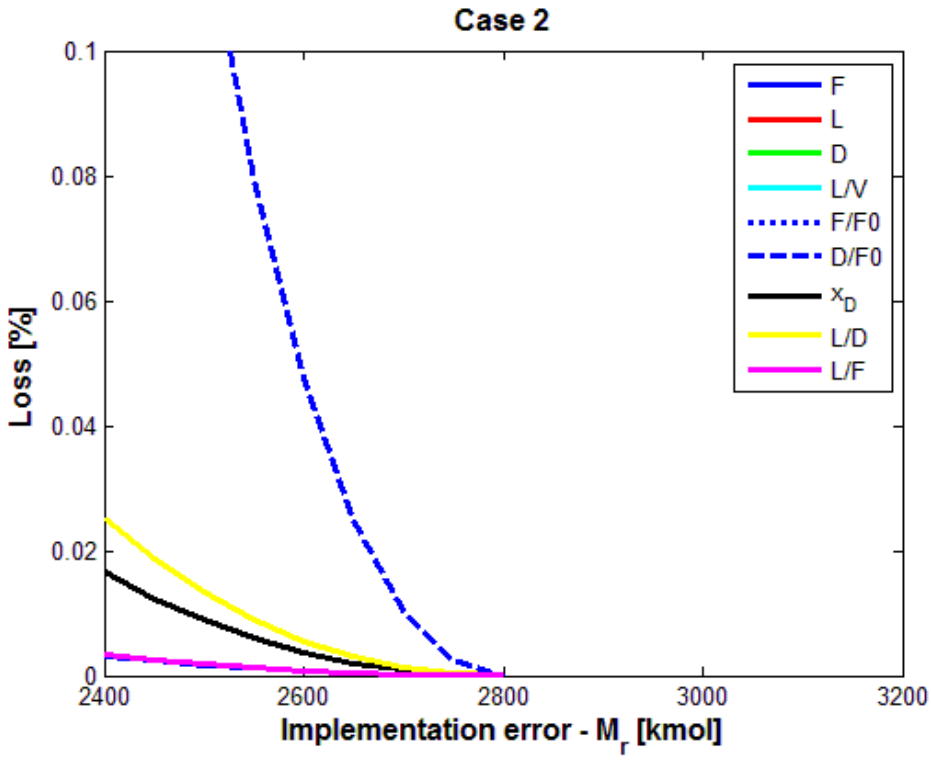


Figure 23: Loss due to implementation error in the reactor holdup (M_r) for case II, where all the candidate controlled variables are plotted against M_r. The process becomes infeasible when M_r>2800 kmol/h, so the loss is only computed for values below this.

The last considered implementation error was in the bottom composition x_B. Here all the candidate variables have similar losses for different values of the implementation error, as shown in Figure 24. As for the previous figure, the process becomes infeasible for x_B-values above 0.0105 because it is not desired to purify the product above this value.

For both the implementation error in reactor holdup and bottom composition, the losses are relatively smaller than the ones found in Larsson et al. (2003).

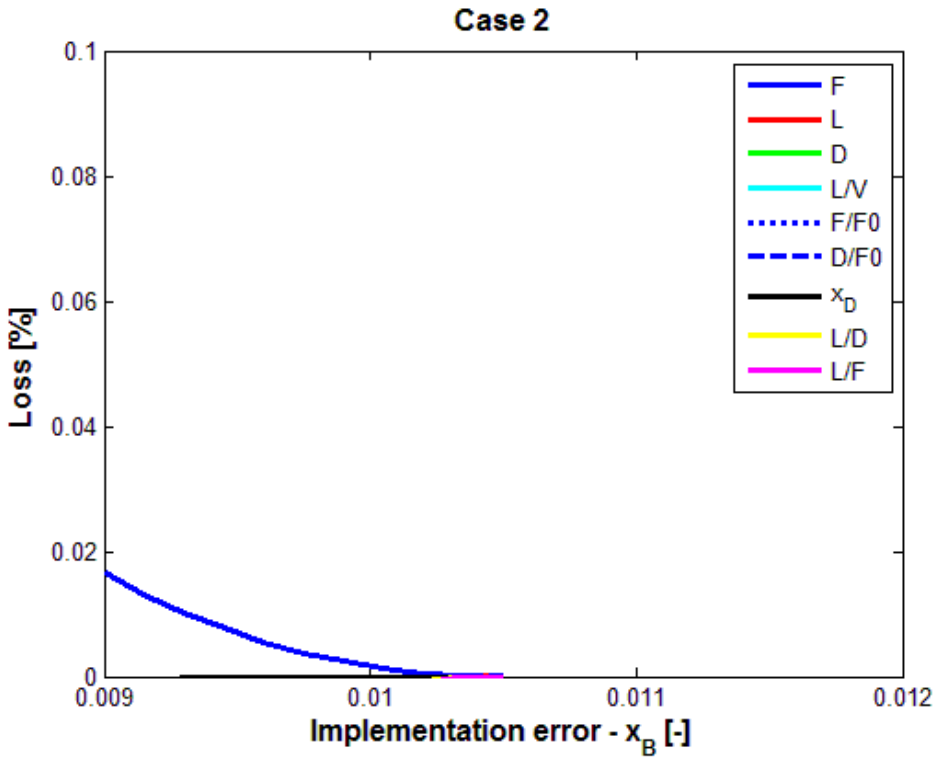


Figure 24: Loss due to implementation error in the reactor holdup (x_B) for case II, where all the candidate controlled variables are plotted against x_B. The process becomes infeasible when x_B>0.0105, so the loss is only computed for values below this.

6.3. *Null Space Method Results*

The loss resulting from the null space method was plotted against the main disturbance for case I and case II in Figure 25 and Figure 26, respectively.

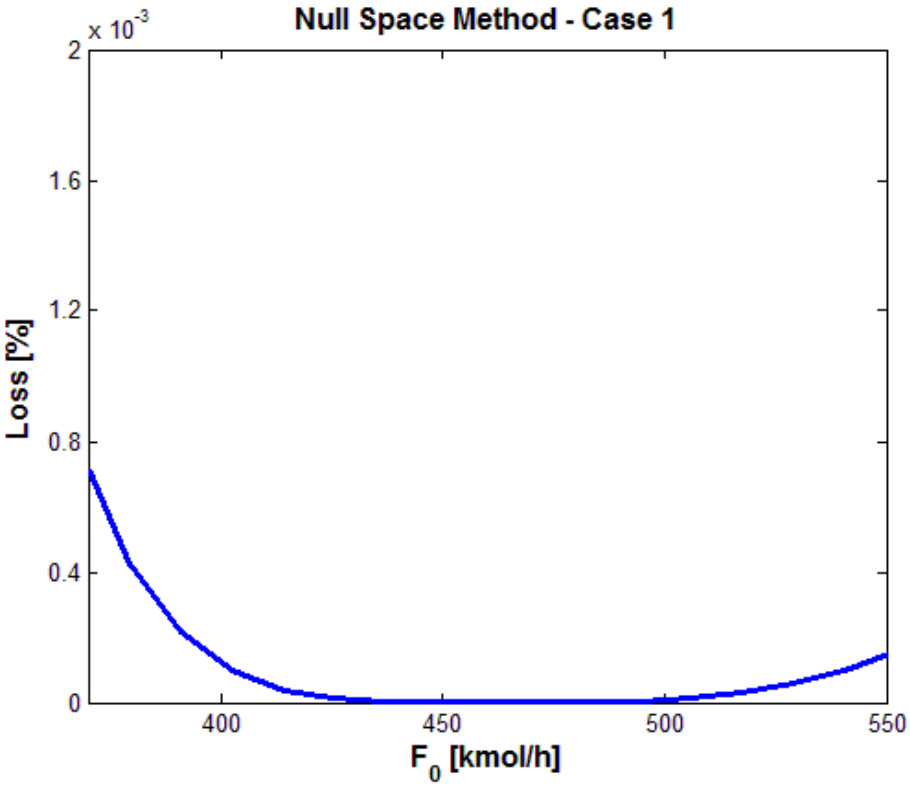


Figure 25: Loss with the null space method for case I and disturbance in F_0.

For the first case, the loss with the null space method is significantly smaller than the losses computed with the Brute force method in the previous chapter. As seen in Figure 25, the magnitude of the loss is on the order of 10^{-3}, compared to 10^{-2} for the smallest loss with disturbance in F_0 found with the Brute force method. This is as expected because the null space method is a more accurate method than the Brute force method. This is probably mostly due to the measurement combination being kept constant instead of a single measurement. The resulting measurement combination c that was kept constant in this case is as follows:

$$\begin{aligned} c = {} & -0.3377y_1 + 0.3343y_2 + 1.1328 * 10^{-4}y_3 + 1.5339 \\ & * 10^{-4}y_4 + 1.5067 * 10^{-4}y_5 + 7.0341 * 10^{-4}y_6 \\ & + 0.7742y_7 - 0.1838y_8 + 0.0419y_9 \\ & - 0.2285y_{10} - 0.1866y_{11} - 0.2285y_{12} \end{aligned} \tag{67}$$

The measurements y_1-y_{12} make up the measurement vector **y** which is given in chapter 5.4.2.

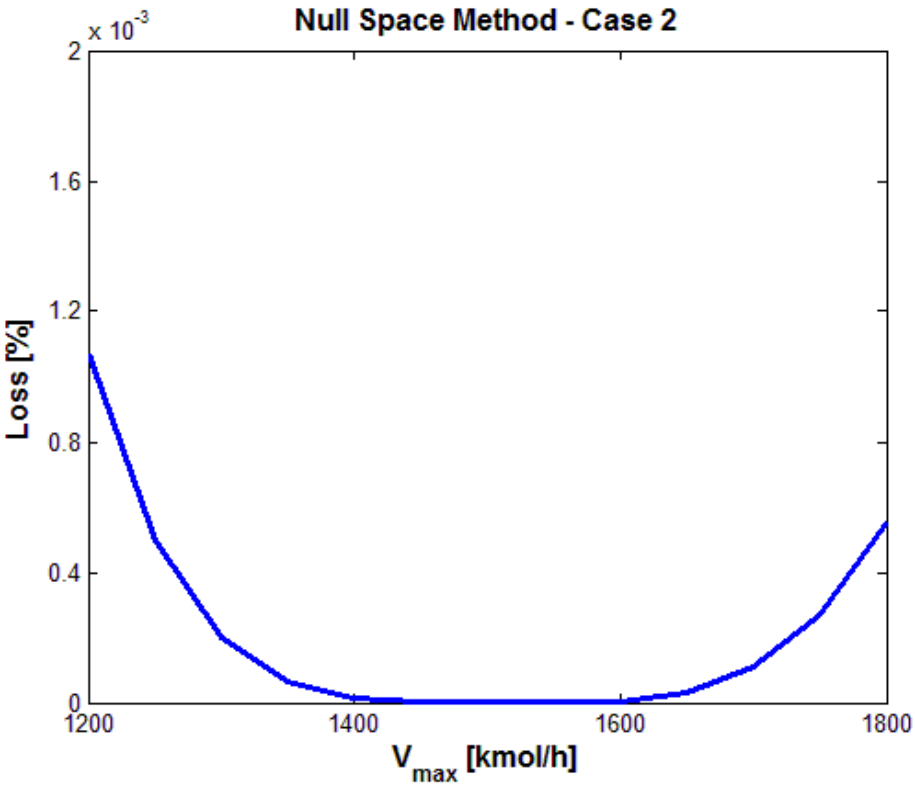


Figure 26: Loss with the null space method for case II and disturbance in V_{max}.

As for case I, the null space method gives smaller loss than the Brute force method in case II as well. In the Brute force method, the smallest loss (for L/F) was also on the order of 10^{-3}, but it was still a bit higher than in Figure 26. This again confirms that the null space method is better and more accurate than the Brute force method. Also, the measurement combination in this case is given here:

$$\begin{aligned} c = {} & -4.3161 * 10^{-5} y_1 - 2.1256 * 10^{-4} y_2 - 8.6716 * 10^{-8} y_3 \\ & - 1.1281 * 10^{-7} y_4 \pm 1.0824 * 10^{-7} y_5 + y_6 \\ & + 6.0919 * 10^{-5} y_7 - 5.7972 * 10^{-5} y_8 - 1.1889 \\ & * 10^{-4} y_9 + 1.1358 * 10^{-4} y_{10} \\ & - 5.3098 * 10^{-4} y_{11} + 1.1358 * 10^{-4} y_{12} \end{aligned} \qquad (68)$$

The graph in Figure 26 is a bit angular, and it could have been smoother with more data points, but the shape of the plot is still visible enough.

6.4. *Exact Local Method Results*

The loss resulting from the exact local method was plotted against the main disturbance in case I and case II in Figure 27 and Figure 28, respectively.

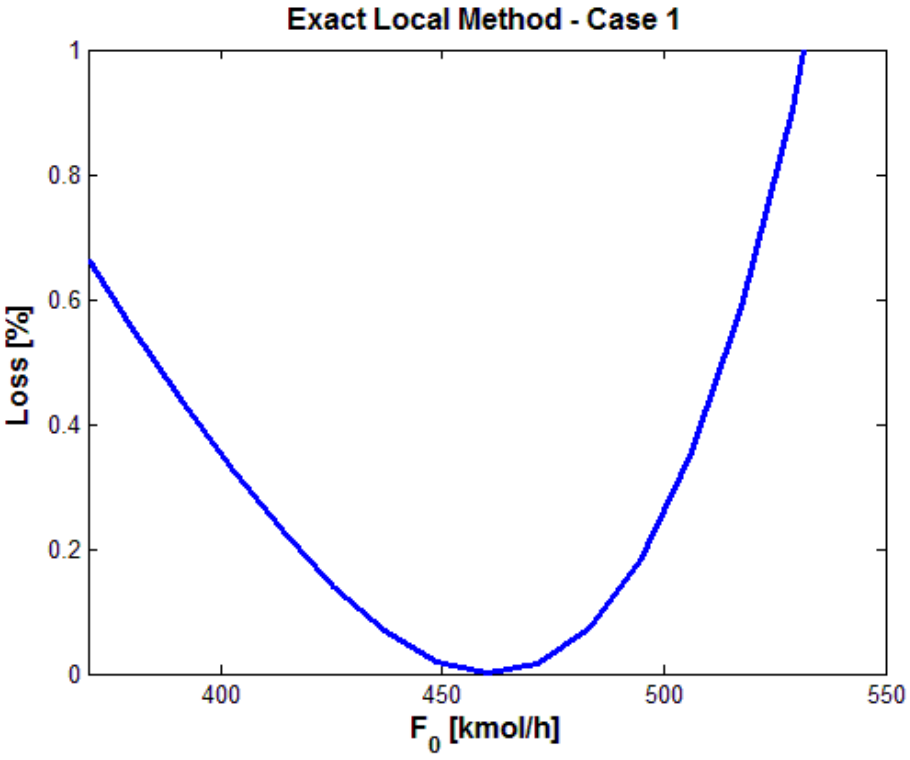


Figure 27: Loss with the exact local method for case I and disturbance in F_0.

Straight away, it is observed that the loss in case I with the exact local method is actually larger than the loss with null space method and the lowest loss found with the Brute force method. It was expected that the exact local method would give the lowest losses of all the methods since this is supposed to be the most accurate of the three. One reason why this have happened may be the choice of the magnitude of the disturbances and measurement errors in the calculation of the measurement combination matrix. Perhaps the scaling matrixes should have been changed.

Also, the measurement combination in this case is given here:

$$\begin{aligned} c = {} & 7.8452 * 10^{-13} y_2 + 3.3470 * 10^{-12} y_3 + 9.7345 * 10^{-11} y_4 \\ & + 1.1286 * 10^{-9} y_5 + 1.3057 * 10^{-9} y_6 + 1.6679 \\ & * 10^{-6} y_7 + 1.3430 * 10^{-8} y_8 - 1.6545 * 10^{-6} y_9 \\ & + 7.4312 * 10^{-19} y_{10} - 1.6545 * 10^{-6} y_{11} \end{aligned} \tag{69}$$

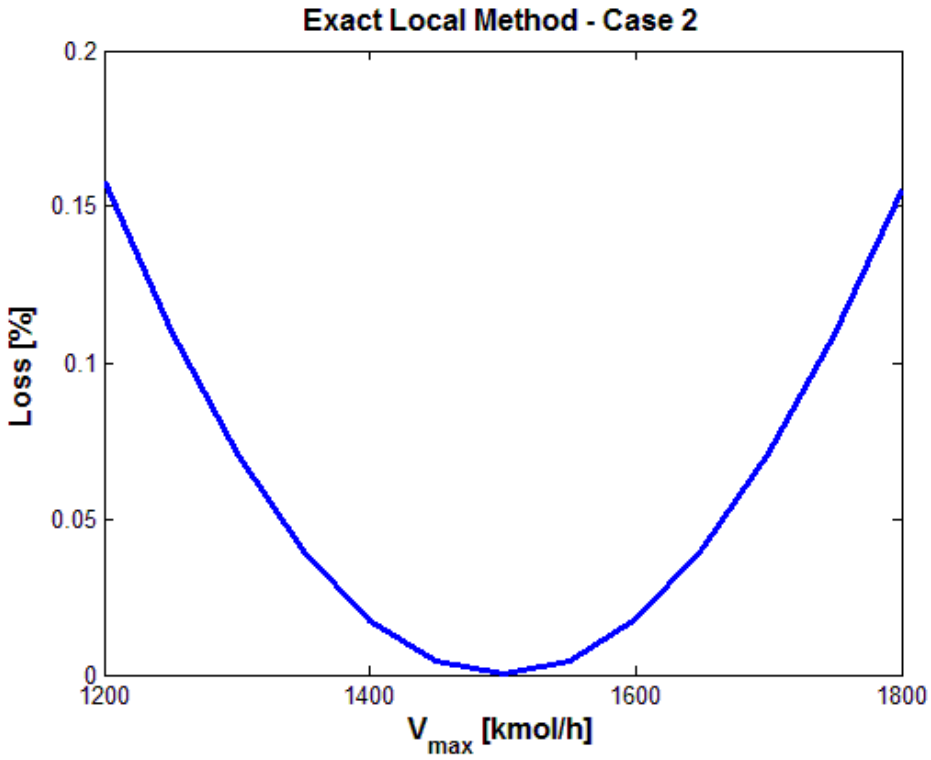


Figure 28: Loss with the exact local method for case II and disturbance in V_{max}.

As for case I, the loss with the exact local method in case II is larger than for the two previous methods of computing the loss. It is assumed that the same reason could be present in this case. Also, the measurement combination in this case is given here:

$$\begin{aligned} c = {} & 6.8564 * 10^{-13} y_2 + 2.9301 * 10^{-12} y_3 + 1.7713 * 10^{-10} y_4 \\ & + 1.9822 * 10^{-9} y_5 + 2.1331 * 10^{-9} y_6 + 3.5457 \\ & * 10^{-6} y_7 - 3.5457 * 10^{-6} y_9 \\ & - 4.4985 * 10^{-9} y_{10} - 3.5502 * 10^{-6} y_{11} \\ & - 4.4985 * 10^{-9} y_{12} \end{aligned} \tag{70}$$

6.5. Results from Dynamic Simulations

As explained above, dynamic simulations were performed for case I, followed by calculation of the economic losses due to the null space and exact local method. The results from this are presented in the two next chapters. Also, for comparison with the steady-state results, the optimal values of the process variables were found from the Simulink model as well. They are presented in Table 6 below.

Table 6: Optimal results with the dynamic Simulink model.

Variable		Case I: min. V
Feed rate	F_0 [kmol/h]	460
Bottom flow	B [kmol/h]	460
Reactor outflow	F [kmol/h]	958
Vapour boilup	V [kmol/h]	1276
Reflux	L [kmol/h]	778
Recycle/distillate	D [kmol/h]	498
Recycle composition	x_D [molA/mol]	0.81
Bottom composition	x_B [molA/mol]	0.0105
Reactor composition	z_F [molA/mol]	0.43
Reactor holdup	M_r [kmol/h]	2800

Comparison with the steady-state optimization results confirm that the results are the same for case I with the two models.

6.5.1. Null Space Method

Figure 29 shows the response in c over time when a step is made in the reflux flow L. As before, c is the measurement combination c=H*y. In addition, the step in L was occurring at t=500 minutes, and the magnitude of the step was 10% of the nominal value of L.

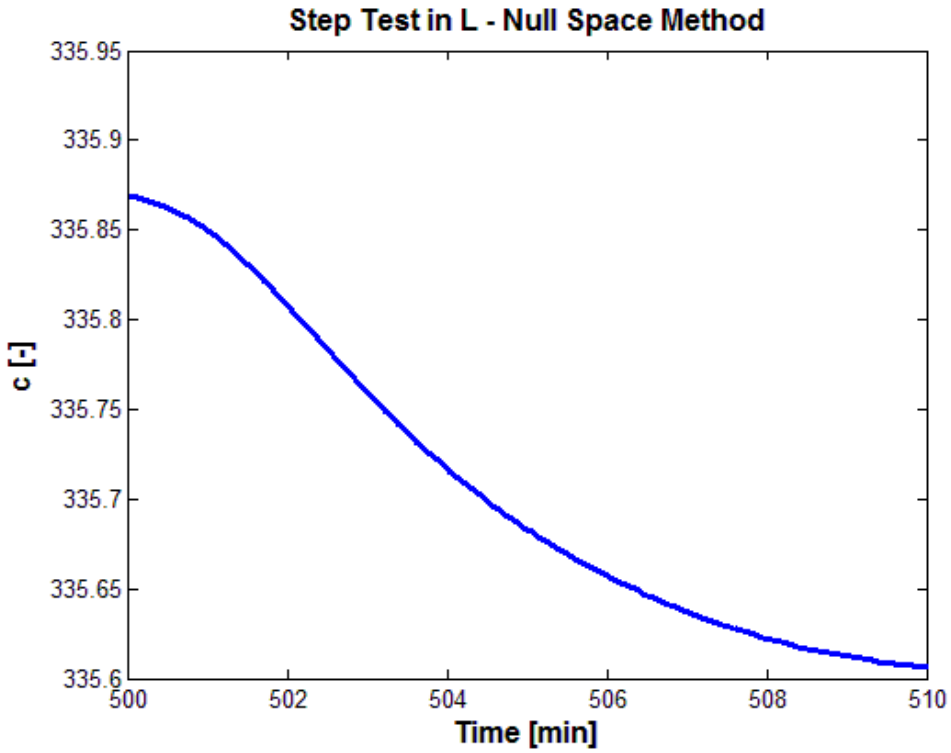


Figure 29: Response in measurement combination c to a step in the reflux L.

Further on, the response in Figure 29 is not perfectly linear. However, it was approximated to be almost linear as an integrating process. Thus, the SIMC rules for an integrating process were applied. The resulting parameters and controller tunings are given in Table 7.

Table 7: SIMC tuning results for the null space method.

Parameter	Value
k' [min^{-1}]	-0.0183
θ [min]	0
τ_c [min]	5
τ_I [min]	20
K_c [-]	-10.9
P [-]	-10.9
I [min^{-1}]	-0.55

In the table above, k' is the response gain found by examining the plot in Figure 29. This method was explained in chapter 2.4. Further on, θ is the time delay, τ_c is the closed-loop time constant, τ_I is the integral time, K_c is the controller gain, and P and I are the controller tunings used in Simulink. The two latter parameters can also be called the proportional action and integral action.

Since the time delay was measured to be very small, it was set to be equal to zero for simplicity. This assumption makes sense because the time delay would not have affected the results too much. A look at equation (33) confirms this. Here, the controller gain (K_c) is a function of k', τ_c and θ. However, the closed-loop time constant is equal to 5, and is thus much larger than the time delay would have been. The time delay had a value of around 0.05-0.01. Further on, the controller gain is found by dividing some value by the sum of τ_c and θ. That indicates that the time delay in this case will have a very small effect on the controller gain.

A high value of τ_c gives a slow controller, because the controller gain decreases by increasing τ_c. Due to the fact that only the steady-state values of the cost are interesting in this project, it is not important to have a very fast controller. It can be very slow and still move to the steady-state point in the end. That is way the closed-loop time constant was chosen to be relatively high ($\tau_c = 5$).

The two last parameters are the controller tuning parameters P and I, also called the proportional and integral gain. These are applied in the Simulink model of the process when L is used to control c_s - c. They are negative in this case because the step response in Figure 29 has a negative slope. This results in a negative value of k', and thus a negative value of K_c, P and I.

Due to the assumption of an integrating process, the integral time was given by $4(\tau_c+\theta)$. Examination of equation (34) confirms this. In an integrating process, τ_I goes to infinity and is therefore the largest of the two possibilities.

Figure 30 shows the loss imposed by the null space method, with disturbance in the feed rate, F_0. However, the loss values were originally negative, but the absolute values were calculated in order to better see the graph.

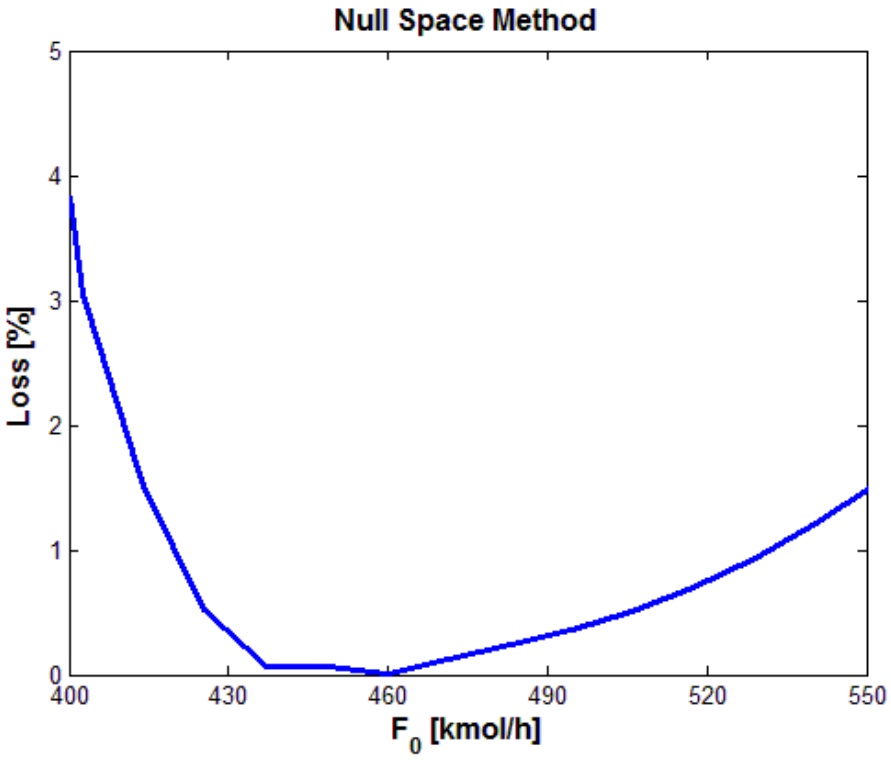


Figure 30: Loss with the null space method for several values of the feed rate, F_0.

Negative losses indicate that the operational cost is lower with the null space method than with the optimal values. Of course, this does not make sense. The optimal values of the vapour boilup should have been smaller than the ones computed with the null space method. Both the optimal cost and the cost with the null space method were calculated from the dynamic Simulink model. In addition, it was ensured that the process had reached steady-state in both cases. Apparently, either the calculation of the optimal loss or the calculation of the loss from the null space method contains some errors.

6.5.2. Exact Local Method

Figure 31 shows the response in c over time when a step is made in the reflux flow L for the exact local method. As before, c is the measurement combination c=H*y. In addition, the step in L was occurring at t=500 minutes, and the magnitude was 10% of the nominal value of L.

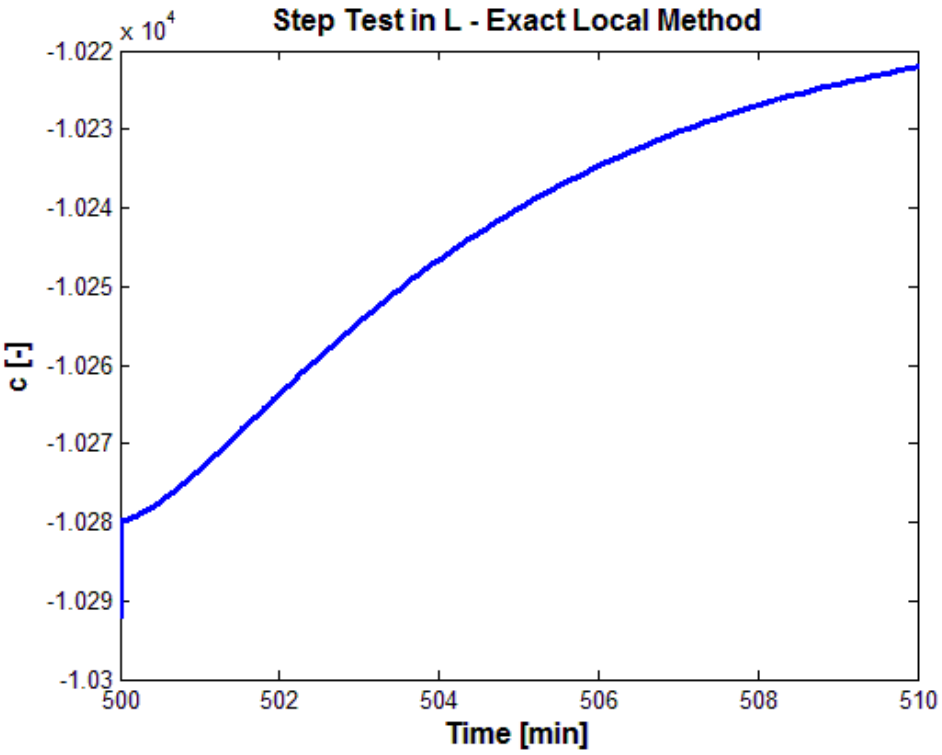


Figure 31: Response in measurement combination c to a step in the reflux L.

Further on, the response in Figure 31 is not perfectly linear. However, it was assumed to be almost linear as an integrating process. Thus, the SIMC rules for an integrating process were applied. The resulting parameters and controller tunings are given in Table 8.

Table 8: SIMC tuning results for the exact local method.

Parameter	Value
k' [min^{-1}]	5,99
θ [min]	0
τ_c [min]	5
τ_I [min]	20
K_c [-]	0.0334
P [-]	0.0334
I [min^{-1}]	0.0017

As in the previous chapter, k' is the response gain, θ is the time delay, τ_c is the closed-loop time constant, τ_I is the integral time, K_c is the controller gain, P is the proportional gain and I is the integral part of the gain. The reasoning for choosing the closed-loop time constant and the method for finding k' is still the same as for the null space method discussed above.

Figure 32 shows the loss by using the exact local method to control the measurement combination. Unfortunately, the graph does not look as expected. It should have been a curve similar to the ones found with the Brute force method. In this plot the loss increases all the time as F_0 increases. Ideally, the loss would be zero at the optimal feed rate, which is $F_0 = 460$ kmol/h. Below and above this value the loss was expected to increase as the feed rate moves away from the optimal value.

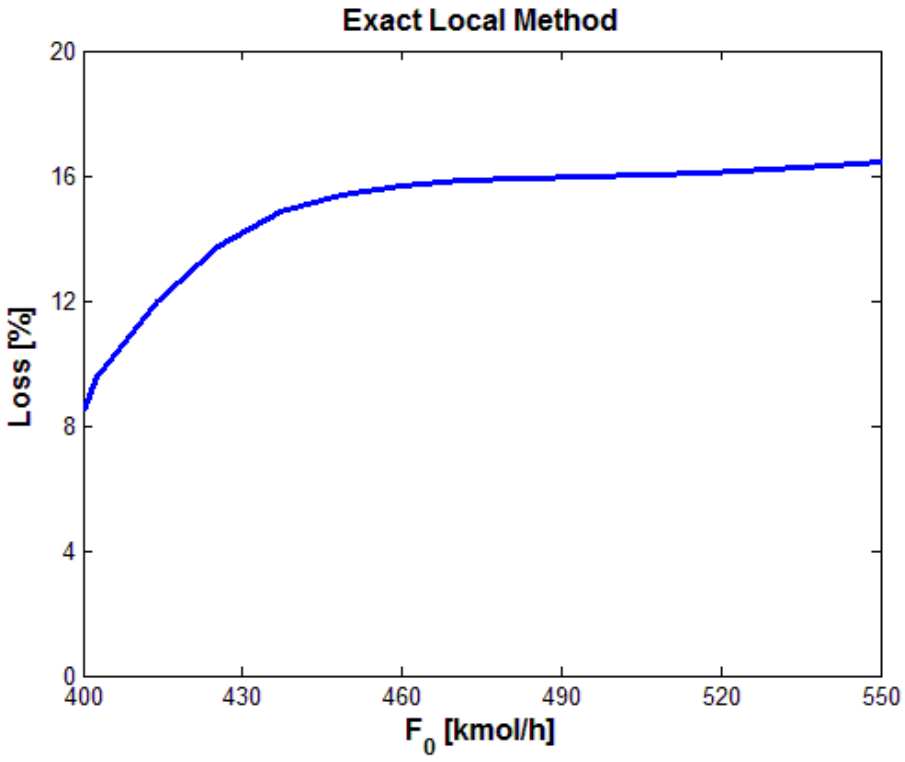


Figure 32: Loss with the exact local method for several values of the feed rate, F_0.

In summary, there are probably some errors in the calculation of the losses with the dynamic Simulink model. This should be investigated further, as will be discussed in the discussion part in chapter 7. Another reason why the results are not as expected, may be the tuning of the controller. The assumption of an integrating process for the tuning part may not be valid. Perhaps the process should have been tuned as a non-integrating first-order process instead of an integrating process.

6.6. *Proposed Control Structures*

Control structures for case I and case II will be suggested in the following sections, based on the simulation results. This is proposed subject to the assumption that the computed losses with self-optimizing control are acceptable.

6.6.1. Case I: Given Feed Rate

As mentioned earlier, the conventional way of designing the control structure for this case is to have maximum reactor holdup (M_r), constant bottom product composition (x_B) and constant set-point for the recycle composition (x_D). This control structure has very small losses with self-optimizing control of x_D, but it can be too expensive to have online measurements of x_D [2]. Also, two-point distillation composition control is already known to have difficult control problems due to interactions between x_B and x_D. The loss plots from the simulations show that L/F had the smallest losses in case I, followed by x_D and L/D. Thus, L/F has the best self-optimizing properties in terms of the loss imposed by keeping the self-optimizing variable constant at its optimal point. Control of L/F is a relatively simple control problem, and L/F should therefore be chosen as the self-optimizing variable in case I. Figure 33 shows the proposed control structure.

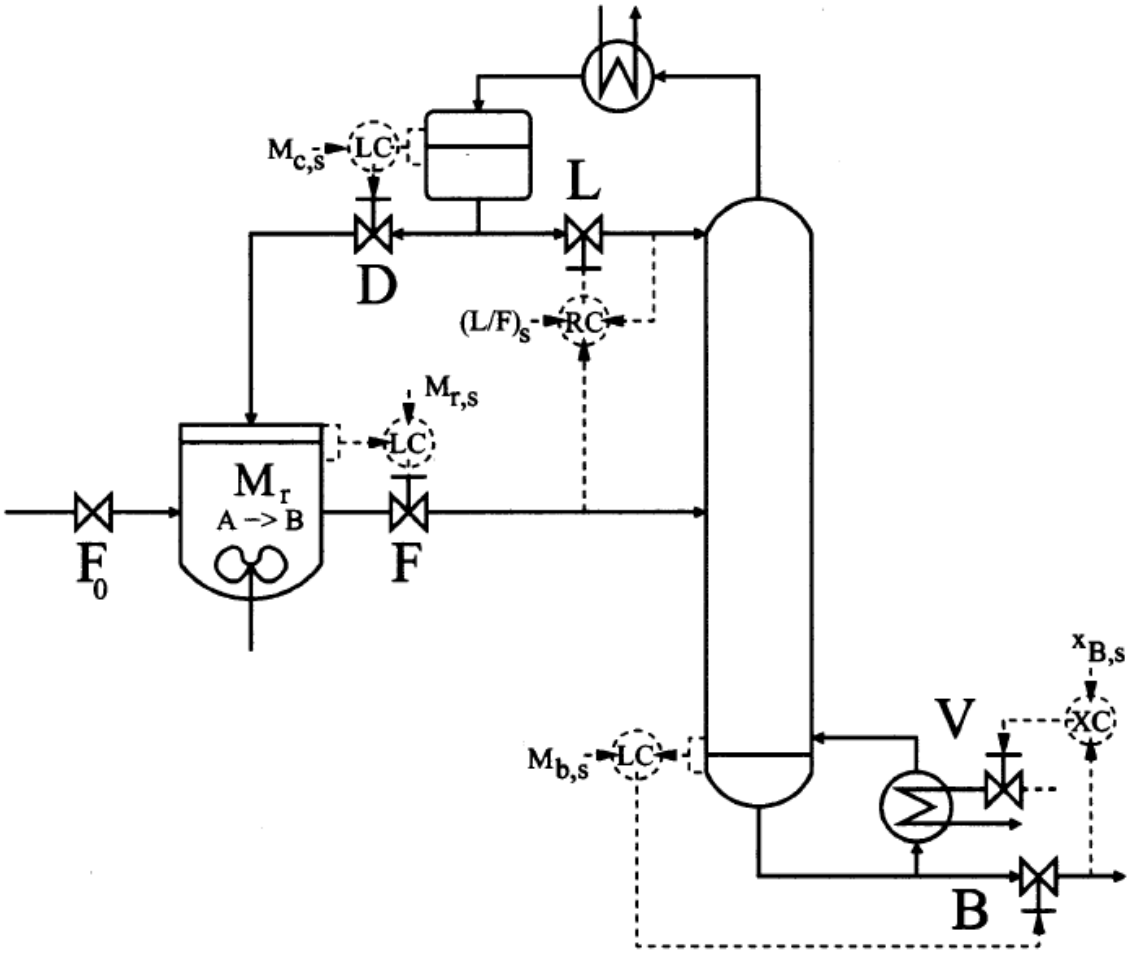


Figure 33: Control structure design for case I with given feed rate F_0 [2].

As seen in the figure above, the following single control loops are included:

$$M_r \leftrightarrow F \tag{71}$$

$$M_d \leftrightarrow D \tag{72}$$

$$M_b \leftrightarrow B \tag{73}$$

$$x_b \leftrightarrow V \tag{74}$$

$$L/F \leftrightarrow L \tag{75}$$

All the active constraints (M_r, x_B, and F_0) should be controlled according to the principle of always controlling active constraints. Since all the pairings should be close, M_r is controlled by the column feed F, x_B is controlled by the vapour boilup V, and F_0 is set by the operator. Also the liquid levels in the reboiler and condenser need to be controlled by the bottom flow B and recycle D, respectively. This is chosen because they are close to each other. That is, the reboiler holdup is close to the bottom flow and the condenser level is close to the recycle flow. After this, the only remaining manipulated variable is the reflux flow L. Therefore, the reflux L will be used to control the variable L/F by measuring both F and L. This is a version of ratio control. All the degrees of freedom are now used and the control structure can be applied. It is already seen in chapter 6.2 that this control structure produces small losses compared to the optimal operation.

6.6.2. Case II: Maximize Production

Figure 34 shows the suggested control structure for case II. As for the previous case, L/F will be used as a self-optimizing variable and controlled instead of x_D due to the simpler control problem this causes. Further on, the variables with the smallest loss in case II were L/F, x_D, L/V and L/D, in increasing order. Therefore, it makes sense to control L/F instead of x_D in terms of economic loss.

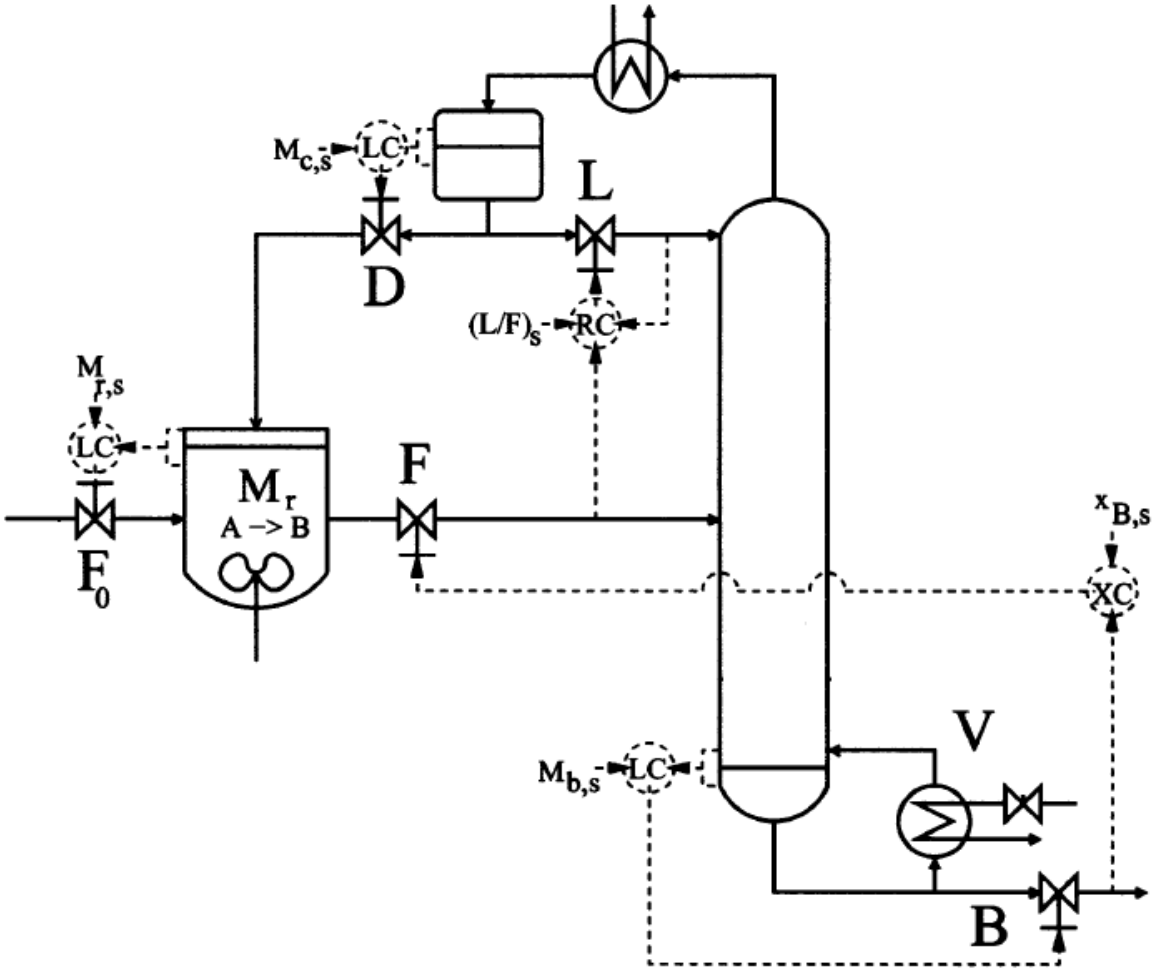


Figure 34: Control structure design for case II with maximum vapour boilup V_{max} [2].

As explained for the first case, the condenser holdup M_d will be controlled by the recycle D and the reboiler level M_b will be controlled by the bottom flow B. Since the vapour boilup now is constrained, the product composition x_B can be controlled by the column feed F instead. Further on, the self-optimizing variable L/F will still be controlled by the reflux L, because this is simple compared to other configurations. This leaves the reactor feed F_0 as the last remaining degree of freedom. It should therefore be used to control the reactor holdup M_r.

All the single control loops are summarized in the following equations:

$$M_r \leftrightarrow F_0 \tag{76}$$

$$M_d \leftrightarrow D \tag{77}$$

$$M_b \leftrightarrow B \tag{78}$$

$$x_b \leftrightarrow F \tag{79}$$

$$L/F \leftrightarrow L \tag{80}$$

In the two control structure figures discussed above, LC stands for level control, RC stands for ratio control and XC means composition control. Further on, $M_{c,s}$, $(L/F)_s$, $M_{b,s}$ and $x_{B,s}$ are the set-points for the condenser holdup, the ratio between L and F, the reboiler holdup and the bottom composition, respectively.

7. Discussion

7.1. Modelling

Most of the work in this project was done in the programming language called Matlab. Quite a lot of time was spent trying and failing with getting the codes to work as wanted. For example, it took a while to get the correct optimization values with the reactor and column model. Since the results were to be compared with existing literature, the model was changed until the same optimization results were obtained.

Regarding the different plots made in Matlab, a function called spline was used for generating smoother plots. This function includes cubic spline data interpolation for any data points. Some of the plots were very angular without utilization of the cubic spline interpolation applied. On the other hand, increasing the number of data points would have given smoother plots in the first case. But this would also have increased the computation time in Matlab, and most of the trends were obvious enough without increasing the number of points.

7.2. Simulation

Some dynamic simulations were also performed, as described in the simulation procedure chapter. For this part, the Simulink model was already available from one of the exercises in the subject "Advanced Process Control" [19]. The intention from the beginning was to only consider steady-state operation, but by having a large enough simulation time, the steady-state was reached. Also, some other problems occurred here. First of all, the dynamic simulations did not give the same optimal values of the operational cost as the steady-state model did when applying disturbances. Secondly, the dynamic model gave a lower cost with the null space method than the optimal cost value.

7.3. Further Work

The work in this thesis focused on a simple process with two components and a simple plant with only one reactor and a distillation column. Since the reality is usually more complex than this, this procedure should be applied to a more complicated process, for example one with three components and a purge included in the plant. This could maybe show how Sigurd's procedure could be utilized for large-scale plants in addition to small, simple processes.

As explained earlier, compositions were used in the measurement vector for the null space and exact local method in the steady-state calculations. However, they were replaced by temperature measurements in the dynamic simulations because that was more realistic and easier in a real plant. Temperatures could have been utilized as measurements also in the steady-state part by applying equation (45). However, the steady-state simulations were already finished when this was discovered. Thus, the composition measurements were still used. The results may not have been that different anyway. Thus, future work could include using temperature measurements also in the steady-state calculations for the null space method and exact local method. In addition, the losses with the null space and exact local method should be found again with the dynamic model to get more realistic results.

Further on, the number of candidate controlled variables could be reduced by plotting the cost J as a function of each candidate CV and looking at the shape of the graph. By using the rules

in chapter 2.3, bad candidate CVs could be eliminated and this could have reduced the computation time. There is also another way to select self-optimizing variables, called the minimum singular value rule [7]. This method was not considered in this work, but could be used for eliminating infeasible controlled variables.

The problem regarding switching of active constraints regions can arise when processes are disturbed above a certain point. Other active constraints regions probably exist for the process studied here as well. This could be one of the things to study further by disturbing the process enough. In addition, in order for a plant to handle case I and case II at the same time, model predictive control (MPC) could be implemented. By using MPC, it would be relatively easy to switch between the two cases because one of the advantages of MPC is fast switching of set-points. That would also cause the need for a dynamic model of the process.

In the step test results, the process was assumed as an integrating process. However, the responses were not entirely linear, so SIMC tuning with a non-integrating process could improve the tuning parameters and maybe result in more reliable loss plots. Since case II was found to be of more industrial relevance, dynamic simulations should be applied there as well.

One other thing to consider is the prices of the different streams in the process. If for example the distillation costs were not neglected, this would have given a different cost function in the two cases. That could give a case where the prices could be considered as disturbances as well.

8. Conclusions

The objective of this work was to apply Skogestad's procedure for plantwide control on a given process plant. Another goal was to find the self-optimizing variables for the process in two different cases.

A simple reactor and distillation column process with a recycle stream has been studied in this master's thesis. A two component system was assumed for the plant due to simplicity and relevance to existing literature. Two cases were studied, one with a given feed rate F_0, and the other one where the vapour boilup V was given. The cost in case I was found to be equal to the amount of vapour boilup in the plant, as seen in equation (46) - (48). In the second case, the cost function to be minimized was found to be equal to the negative value of the feed rate, F_0. This is derived in equation (52).

Optimization of the process was one of the first tasks, in order to find the nominal optimal values of all the variables, in addition to active constraints. The optimization results are given in Table 5 for case I and case II. They show the active constraints and the values for some other flows and compositions in the plant. It is seen that the two active constraints hold for both case I and II. These are the bottom composition ($x_B \leq 0.0105$ mol A/mol) and the maximum reactor holdup ($M_r \leq 2800$ kmol/h). In case I, the optimal vapour boilup was equal to V=1276 kmol/h. For case II, the optimal reactor feed rate turned out to be F_0=498 kmol/h.

There was one unconstrained degree of freedom left for self-optimizing control in both cases. Nine different candidate controlled variables were suggested, namely F, L, D, L/D, L/F, L/V, x_D, F/F0 and D/F0. The Brute force method was used to calculate the cost by keeping each and one of these candidate CVs constant at a time while applying different disturbances. Then the economic loss was calculated by using equation (55). L/F, x_D and L/D were found to have the smallest losses in both cases for most of the disturbances and implementation errors. The variable with the smallest loss of the three was L/F, and was thus chosen as the self-optimizing variable for the plant in both cases. This was also because a two-point composition control with x_D as self-optimizing variable can be difficult and expensive.

Two other methods were used to find the economic loss, the null space method and the exact local method. A measurement combination (c) was found and kept constant instead of a single variable. The null space method resulted in significantly smaller losses with disturbance in F_0 for both case I and II. This is shown in Figure 25 and Figure 26. The exact local method did not give smaller losses than the null space method, so this method should be investigated further.

Also, the dynamic simulations gave the same nominal optimal results as for the steady-state model. This is seen in Table 6. Step tests were made in the reflux L, followed by plotting the response in the measurement combination c. Tuning parameters were found from the step test responses by using the SIMC tuning rules. These responses are shown in Figure 29 and Figure 31, and the tuning results are given in Table 7 and Table 8.

The proposed control structures for the two cases are sketched in Figure 33 and Figure 34. Also, the pairings between controlled variables and manipulated variables are listed in the equations (71) - (80).

It is recommended to investigate the null space method and exact local method further, especially in the dynamic model since this did not give the expected loss plots. Another possibility is to implement cascade control for the bottom composition in order to get faster and better control.

9. Nomenclature

9.1. List of Symbols

Symbol	Unit	Description
A	-	Component A
B	-	Component B
B	kmol/h	Bottom rate
c	-	Candidate controlled variable
c_{opt}	-	Optimal value of controlled variable
c_s	-	Set-point of controlled variable
c_A	mol/L	Concentration of component A
c_B	mol/L	Concentration of component B
CV1	-	Primary controlled variables
CV2	-	Secondary controlled variables
d	-	Disturbance
D	kmol/h	Recycle/distillate rate
F	-	Optimal sensitivity matrix
F	kmol/h	Feed rate to the distillation column
F_0	kmol/h	Feed rate to the reactor
g	-	Process transfer function
G_y	-	Steady-state gain matrix
H	-	Measurement combination matrix
I	$[min^{-1}]$	Integral part of controller gain
J	\$/h	Cost function
J_{opt}		Optimal cost
J_u		Cost function with constant input
k	h^{-1}	Reaction rate constant
k	?	Plant gain
k'	?	Plant gain divided by the dominant lag time constant
K_c	?	Controller gain
L	kmol/h	Reflux rate
M_b	kmol/h	Reboiler holdup
M_d	kmol/h	Condenser holdup
M_r	kmol/h	Reactor holdup
N_{0m}	-	Number of manipulated variables with no steady-state effect
N_{0y}	-	Number of output variables that must be controlled and have no steady-state effect
N_m	-	Number of dynamic degrees of freedom or manipulated variables
N_{ss}	-	Number of steady-state degrees of freedom
n_d	-	Number of disturbances
n_u	-	Number of inputs/manipulated variables
n_y	-	Number of measurements
n^y	-	Measurement error
p_B	\$/mol	Price of the bottom product

9.1. List of Symbols continued

Symbol	Unit	Description
p_D	\$/mol	Price of the recycle
P	-	Proportional part of controller gain
p_{F0}	\$/mol	Price of the reactor feed
p_v	\$/mol	Price of the vapour boilup
r	mol/L•h	Reaction rate
T_{Bi}	K	Boiling temperature of component i
T_J	K	Temperature at stage J in the column
u	-	Input/manipulated variable
V	kmol/h	Vapour boilup
W_d	-	Diagonal scaling matrix for quantifying the disturbances
W_n	-	Diagonal scaling matrix for quantifying the measurement errors
x	-	State variables
x_B	mol A/mol	Mole fraction in bottom flow
x_i	-	Mole fraction of component i
$x_{i,J}$	-	Mole fraction of component I at stage J in the column
y	-	Vector of single measurements
y_{opt}	-	Optimal values of measurements
z_{F0}	mol A/mol	Mole fraction in reactor feed
z_F	mol A/mol	Mole fraction in column feed
α_{AB}	-	Relative volatility between component A and B
θ	min	Time delay
τ_l	Min	Dominant lag time constant
τ_c	Min	Desired closed-loop time constant
τ_I	Min	Integral time

9.2. List of Abbreviations

Abbreviation	Stands for
CV	Controlled variable
DOF	Degrees of freedom
HDA	Hydrodealkylation of toluene
MPC	Model predictive control
NC	Number of components
RTO	Real time optimization
TPM	Throughput manipulator

10. References

1. Skogestad, S., *Plantwide Control - Recent Development and Applications*, ed. G. Rangaiah, P., Kariwala, V.,2012: Wiley & Sons.
2. Larsson, T., Govatsmark, M. S., Skogestad, S., Yu, C. C.,, *Control Structure Selection for Reactor, Separator, and Recycle Processes.* 2003. **42**: p. 1225-1234.
3. Wu, K., Yu, C.,, *Reactor/separator processes with recycle - 1. Candidate control structure for operability.* 1996. **20**(11): p. 1291-1316.
4. Alstad, V., Skogestad, S., Hori, E. S.,, *Optimal measurement combinations as controlled variables.* 2009.
5. Alstad, V., Skogestad, S.,, *Null Space Method for Selecting Optimal Measurement Combinations as Controlled Variables.* 2007. **46**: p. 846-853.
6. Halvorsen, I.J., Skogestad, S., Morud, J. C., Alstad, V.,, *Optimal Selection of Controlled Variables.* 2003. **42**: p. 3273-3284.
7. Skogestad, S., *Plantwide control: the search for the self-optimizing control structure.* 2000. **10**: p. 487-507.
8. Skogestad, S., *Control structure design for complete chemical plants.* 2004. **28**: p. 219-234.
9. Luyben, W.L., *Dynamics and Control of Recycle Systems. 2. Comparison of Alternative Process Designs.* 1993. **32**: p. 476-486.
10. Skogestad, S., *The Dos and Don'ts of Distillation Column Control*

2007. **85**(A1): p. 13-23.

11. Wu, K., Yu, C., Luyben, W. L., Skogestad, S.,, *Reactor/separator processes with recycles-2. Design for composition control.* 2002. **27**: p. 401-421.
12. de Araujo, A.C.B., Govatsmark, M., Skogestad, S.,, *Application of plantwide control to the HDA process. I - steady-state optimization and self-optimizing control.* 2007. **15**: p. 1222-1237.
13. Larsson, T., Hestetun, K., Hovland, E., Skogestad, S.,, *Self-Optimizing Control of a Large-Scale Plant: The Tennessee Eastman Process.* 2001. **40**: p. 4889-4901.
14. Larsson, T., Skogestad, S.,, *Plantwide control - A review and a new design procedure.* 2000. **21**(4): p. 209-240.
15. Skogestad, S., *Economic Plantwide Control.* 2011.
16. Skogestad, S., *Near-Optimal operation by self-optimizing control: from process control to marathon running and business systems.* 2004. **29**: p. 127-137.
17. Skogestad, S., *Advanced Process Control: Solution for the Exercise 2*, 2012.

18. Skogestad, S., Grimholt, C.,, *The SIMC method for smooth PID controller tuning.* 2011.
19. Skogestad, S. *Exercise 5: Dynamic simulation of a simple plant process with reactor, separator and recycle using Simulink*. 2012.
20. Skogestad, S., Postlethwaite, I.,, *Multivariable Feedback Control - Analysis and Design.* 2007.
21. MathWorks. *fmincon - Find minimum of constrained nonlinear multivariable function*. 2013; Available from: http://www.mathworks.se/help/optim/ug/fmincon.html.

Appendix A

Matlab model of the column, colamodSS.m:

```
function residue=colamodSS(x,U)
%
%           This is a nonlinear steady state model of a distillation column
with
%           NT-1 theoretical stages including a reboiler (stage 1) plus a
%           total condenser ("stage" NT).
%
%           Model assumptions:
%
%           Two components (binary separation);
%           Constant relative volatility;
%           No vapor holdup;
%           One feed and two products;
%           Constant molar flows (same vapor flow on all stages);
%           Total condenser
%
%           The model is based on column A in Skogestad and Postlethwaite
%           (1996). The model has NT states.
%
% Inputs:    x    - states, the NT compositions of light component A with
%                   reboiler/bottom stage as x(1) and condenser as x(NT).
%            U(1) - reflux L,
%            U(2) - boilup V,
%            U(3) - top or distillate product flow D,
%            U(4) - bottom product flow B,
%            U(5) - feed rate F,
%            U(6) - feed composition, zF.
%            U(7) - feed liquid fraction, qF.
%            U(8) - number of stages, NT.
%            U(9) - location of feed stage, NF.
%            U(10)- relative volatility, alpha.
%
% Outputs:   [residue] = f(x), residue=0 if x is a solution of the
%            system of nonlinear equations

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%-------------------------------------

% Inputs and disturbances
LT = U(1);                              % Reflux
VB = U(2);                              % Boilup
D  = U(3);                              % Distillate
B  = U(4);                              % Bottoms

F  = U(5);                              % Feedrate
zF = U(6);                              % Feed composition
qF = U(7);                              % Feed liquid fraction

NT = U(8);                              % Number of stages (including
reboiler and total condenser:
NF = U(9);                              % Location of feed stage (stages are
counted from the bottom):
alpha = U(10);                          % Relative volatility
```

```
%Preallocation
y=ones(NT-1,1);
dMxdt = ones(NT,1);
dMdt = ones(2,1);

% THE MODEL

% Vapor-liquid equilibria
i=1:NT-1;    y(i)=alpha*x(i)./(1+(alpha-1)*x(i));
% --------------------Column-------------------------
%
% Component balances
% ============================================
% Reboiler (assumed to be an equilibrium stage)
dMxdt(1)= (LT+qF*F)*x(2) - VB*y(1) - B*x(1);
% Stripping section trays
i=2:NF-1;
dMxdt(i)= (LT+qF*F)*x(i+1) - (LT+qF*F)*x(i) + VB*y(i-1) - VB*y(i);
% Feed tray
dMxdt(NF)= LT*x(NF+1) - (LT+qF*F)*x(NF) + VB*y(NF-1) - (VB+(1-qF)*F)*y(NF)
+ F*zF;
% Enrichment section trays
i=NF+1:NT-1;
dMxdt(i)= (LT)*x(i+1) - LT*x(i) + (VB+(1-qF)*F)*y(i-1) - (VB+(1-
qF)*F)*y(i);
% Total condenser (no equilibrium stage)
dMxdt(NT)= (VB+(1-qF)*F)*y(NT-1) - LT*x(NT) - D*x(NT);
%=============================================
%
% Mass balances
%=============================================
% Reboiler
dMdt(1) = LT+qF*F      - VB      - B;
% Condenser
dMdt(2) = VB+(1-qF)*F  - LT      - D;
% ============================================
%-------------------------------------------------------
%
% Output
residue=[dMxdt; dMdt];
```

Model of the reactor, CSTR_SS_model.m:

```
function xpr = CSTR_SS_model(t,X,U)
%
% CSTRmod - This is a nonlinear model of a CSTR with two feeds and one
%           product.
%           Model assumptions: Two components.
%           The model has two states (the composition of A and holdup)
%
% Inputs:   t     - time in [min]
%           X     - state, the first state is the composition of light
component A (X(1))
%                   and the second state is the holdup in the reactor
(X(2)).
%           U(1)  - product rate F
%           U(2)  - recycle/distillate D
%           U(3)  - feed rate F0
%           U(4)  - feed composition, zF0
%           U(5)  - recycle composition, xD.
%
% Output:   xprime - vector with time derivative of the two states

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Splitting the states
zFA=X(1);                   % Liquid hold up of A in reactor
Mr=X(2);                    % Liquid hold up in reactor

MrA=zFA*Mr;

% Inputs and disturbances
F   = U(1);                  % Product rate
D   = U(2);                  % Recycle/distillate

F0  = U(3);                  % Feed rate
zF0 = U(4);                  % Feed composition
xD  = U(5);                  % Recycle composition
k1  = U(6);                  % Reaction rate constant

% The model
% Dynamic component balances
dMrAdt = F0*zF0+D*xD-F*zFA-k1*MrA;
dMrdt  = F0+D-F;

xpr = [dMrAdt; dMrdt];

end
```

Script for the function which is to be minimized, fun.m:

```
function [j] = fun( x)
% Objective function
%
%Parameters
global p;
if p.case_I==1
    j=x(p.NT+2);
else
    j=-x(p.NT+8);
end
```

Script for the nonlinear equality and inequality constraints, nlcon.m:

```
function [c ,ceq ] = nlcon(x)
%NLCON Nonlinear equality and inequality constraints

% Parameter
global p;

% Column parameters
U1=[x(p.NT+1:p.NT+6); p.qF; p.NT; p.NF; p.alpha];
% Column state variables
X1=x(1:p.NT);
% CSTR parameters
U2=[x(p.NT+5) x(p.NT+3) x(p.NT+8) p.zF0 x(p.NT) p.k1]';

% CSTR state variables
X2=x(p.NT+6:p.NT+7);

% Nonlinear inequality constraints C(x)<0
c=[];
% Nonlinear equality constraints C(x)=0
ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2)];      % 1 unconstrained DOF

%%
%Keeping CV constant at its optimal value for case 1
load yopt_nom_case1.mat     % Optimal values of the measurements for case I

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+5)-(A1(1)/60)*1];
% keeping F constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+1)-(A1(2)/60)*1.0];
% keeping L constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+3)-(A1(3)/60)*1.0];
% keeping D constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+1)-
(A1(4))*1.0*x(p.NT+3)];  % keeping L/D constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+1)-
A1(5)*1.0*x(p.NT+5)];    % keeping L/F constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+1)-
A1(6)*1.0*x(p.NT+2)];    % keeping L/V constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(22)-A1(7)*0.8];
% keeping xD constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+5)-
A1(8)*1.0*x(p.NT+8)];    % keeping F/F0 constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+3)-
A1(9)*1.0*x(p.NT+8)];    % keeping D/F0 constant

%%
%Keeping CV constant at its optimal value for case 2
load yopt_nom_case2.mat    % Optimal values of the measurements for case II
```

```
% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+5)-(A4(1)/60)*1.0];
% keeping F constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+1)-(A4(2)/60)*1.0];
% keeping L constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+3)-(A4(3)/60)*1.0];
% keeping D constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+1)-
(A4(4))*1.0*x(p.NT+3)];  % keeping L/D constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+1)-
(A4(5))*1.0*x(p.NT+5)];  % keeping L/F constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+1)-
(A4(6))*1.0*x(p.NT+2)];  % keeping L/V constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(22)-(A4(7))*1.0];
% keeping xD constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+5)-
(A4(8))*1.0*x(p.NT+8)];  % keeping F/F0 constant

% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); x(p.NT+3)-
(A4(9))*1.0*x(p.NT+8)];  % keeping D/F0 constant

%% Null space method case 1
load H11.mat            % The H-matrix with the measurement combinations
% y=[x(1);
%     x(6);
%     x(10);
%     x(14);
%     x(18);
%     x(22);
%     x(23)*60;
%     x(24)*60;
%     x(25)*60;
%     x(26)*60;
%     x(27)*60;
%     x(30)*60];        % Measurement vector y
%
% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); H11*y-0.0272];


%% Null space method case 2

load H2.mat             % The H-matrix with the measurement combinations
% y=[x(1);
%     x(6);
%     x(10);
%     x(14);
%     x(18);
%     x(22);
%     x(23)*60;
%     x(24)*60;
%     x(25)*60;
%     x(26)*60;
%     x(27)*60;
```

```
%     x(30)*60];        % Measurement vector y
%
% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); H2*y-0.8315];

%% Exact local method
% Case 1
load Hel.mat            % The H-matrix with the measurement combinations

% y=[x(1);
%     x(6);
%     x(10);
%     x(14);
%     x(18);
%     x(22);
%     x(23)*60;
%     x(24)*60;
%     x(25)*60;
%     x(26)*60;
%     x(27)*60;
%     x(30)*60];        % Measurement vector y
%
%
% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); Hel*y+0.0011];

% With p=0.01 instead of p=0.02
% load Heln.mat
% y=[x(1);
%     x(6);
%     x(10);
%     x(14);
%     x(18);
%     x(22);
%     x(23)*60;
%     x(24)*60;
%     x(25)*60;
%     x(26)*60;
%     x(27)*60;
%     x(30)*60];        % Measurement vector y
%
%
% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); Heln*y-0.0046];

% Case 2
%
load Hel2.mat           % The H-matrix with the measurement combinations
%
% y=[x(1);
%     x(6);
%     x(10);
%     x(14);
%     x(18);
%     x(22);
%     x(23)*60;
%     x(24)*60;
%     x(25)*60;
%     x(26)*60;
%     x(27)*60;
%     x(30)*60];        % Measurement vector y
%
%
% ceq=[colamodSS(X1,U1); CSTR_SS_model(0,X2,U2); Hel2*y+0.0030];
```

```
%%

%For OPTI toolbox
if (nargout ==1&&p.OPTI==1)
    c=[c;ceq];
end
```

Script for the optimization of the process, testScriptOpt.m:

```
% Script for optimization of reactor, separator and recycle process.
% The numerical description of the process is taken from Larsson et al
(2003)

clc
clear all
global p;

% Column parameters
p.qF    = 1;            % Liquid fraction in column feed [-]
p.NT    = 22;           % Number of stages in distillation column [-]
p.NF    = 13;           % Feed stage in distillation column [-]
p.alpha = 2;            % Relative volatility between component A and B [-]

Vmaxs=(1500/60);        % The optimal value of vapour boilup in case II
[kmol/h]
p.Vmax  = Vmaxs;        % Making the vapour boilup a global parameter
[kmol/h]
p.xB=0.0105;            % Bottom composition (original value) [mol A/mol]
p.F=958;                % Column feed (original value) [kmol/h]
p.L=778;                % Reflux (original value) [kmol/h]
p.D=498;                % Recycle (original value) [kmol/h]
p.LD=1.6;               % Ratio L/D (original value) [-]
p.LF=0.8;               % Ratio L/F (original value) [-]
p.LV=0.61;              % Ratio L/V (original value) [-]
p.xD=0.82;              % Distillate composition (original value) [mol
A/mol]
p.FF0=2;                % Ratio F/F0 (original value) [-]
p.DF0=2;                % Ratio D/F0 (original value) [-]

% CSTR parameters
F0s = (460/60);         % The optimal value of reactor feed (case I)
[kmol/h]
zF0 = 0.9;              % The optimal value of reactor feed composition
[mol A/mol]

p.F0 = F0s;             % Making the reactor feed a global parameter
[kmol/h]
p.zF0 = zF0;            % Making the reactor feed a global parameter [-]
p.k1 = 0.341/60;        % Reaction rate constant [min^-1]
p.Mr=2800;              % Liquid holdup in reactor [kmol/h]
% Flags
p.OPTI=0;
p.case_I=0;             % Switching between case I and case II

if p.case_I==1          % Case I
    % Constraints
    lb=zeros(p.NT+8,1);                 % Lower bounds for the 30 variables
    ub=[ones(p.NT,1); ones(8,1)*Inf];   % Upper bounds for the 30 variables

    % Active constraints for case I
    % xB <= 0.0105
    ub(1)=0.0105;
    % Mr <= 2800;
    ub(p.NT+7)=2800;
    % F0 = fixed;
    lb(p.NT+8)=p.F0;
    ub(p.NT+8)=p.F0;
```

```
else                    % Case II
    % Constraints
    lb=zeros(p.NT+8,1);                 % Lower bounds for the 30 variables
    ub=[ones(p.NT,1); ones(8,1)*Inf];   % Upper bounds for the 30 variables

    % Active constraints for case II
    % xB <= 0.0105
    ub(1)=0.0105;
    % Mr <= 2800;
    ub(p.NT+7)=2800;
    % V <= Vmax;
    ub(p.NT+2)=p.Vmax;
end

% Initial values of the 30 variables
x0= [ones(1,p.NT)*0.5 10 15 5 5 1.1 0.5 1000 400/60]';

% fmincon options
options = optimset('TolFun',10e-6,'TolCon',10e-6,'MaxFunEvals',1e4,...
'Display','none','Algorithm','sqp','Diagnostics','off'...
);
% fmincon
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
x0=x;

% Results
if p.case_I==1
    casename='case I: min operation cost(energy)\n';
else
    casename='case II: max production rate\n';
end
results_fmincon=sprintf(strcat(...
    casename,...
    'feed rate,             F0[kmol/h]      = %1$0.4d\n',...
    'reactor effluent,      F[kmol/h]       = %2$0.4d\n',...
    'vapor boilup,          V[kmol/h]       = %3$0.4d\n',...
    'reflux,                L[kmol/h]       = %4$0.4d\n',...
    'recycle (distilate),   D[kmol/h]       = %5$0.4d\n',...
    'recycle composition,   xD[molA/mol]    = %6$0.4d\n',...
    'bottom composition,    xB[molA/mol]    = %7$0.4d\n',...
    'reactor composition,   zF[molA/mol]    = %8$0.4d\n',...
    'reactor holdup,        Mr[kmol/h]      = %9$0.4d\n'...
    ),x(p.NT+8)*60,x(p.NT+5)*60,x(p.NT+2)*60,x(p.NT+1)*60, x(p.NT+3)*60,...
    x(p.NT), x(1), x(p.NT+6),x(p.NT+7))
% Nominal results
Vnom=x(p.NT+2)*60           % Nominal vapour boilup
F0n=-x(p.NT+8)*60           % Nominal negative feed
```

Script for calculation of the cost with the Brute force method, testScript_BF.m:

```
% Script for optimization of reactor, separator and recycle process.
% The numerical description of the process is taken from Larsson et al
(2003)

clc
clear all
global p;

% Column parameters
p.qF    = 1;            % Liquid fraction in column feed [-]
p.NT    = 22;           % Number of stages in distillation column [-]
p.NF    = 13;           % Feed stage in distillation column [-]
p.alpha = 2;            % Relative volatility between component A and B [-]

Vmaxs=(1500/60);        % The optimal value of vapour boilup in case II
[kmol/h]
p.Vmax  = Vmaxs;        % Making the vapour boilup a global parameter
[kmol/h]
p.xB=0.0105;            % Bottom composition (original value) [mol A/mol]
p.F=958;                % Column feed (original value) [kmol/h]
p.L=778;                % Reflux (original value) [kmol/h]
p.D=498;                % Recycle (original value) [kmol/h]
p.LD=1.6;               % Ratio L/D (original value) [-]
p.LF=0.8;               % Ratio L/F (original value) [-]
p.LV=0.61;              % Ratio L/V (original value) [-]
p.xD=0.82;              % Distillate composition (original value) [mol
A/mol]
p.FF0=2;                % Ratio F/F0 (original value) [-]
p.DF0=2;                % Ratio D/F0 (original value) [-]

% CSTR parameters
F0s = (460/60);         % The optimal value of reactor feed (case I)
[kmol/h]
zF0 = 0.9;              % The optimal value of reactor feed composition
[mol A/mol]

p.F0 = F0s;             % Making the reactor feed a global parameter
[kmol/h]
p.zF0 = zF0;            % Making the reactor feed a global parameter [-]
p.k1 = 0.341/60;        % Reaction rate constant [min^-1]
p.Mr=2800;              % Liquid holdup in reactor [kmol/h]
% Flags
p.OPTI=0;
p.case_I=1;             % Switching between case I and case II

if p.case_I==1          % Case I
    % Constraints
    lb=zeros(p.NT+8,1);                 % Lower bounds for the 30 variables
    ub=[ones(p.NT,1); ones(8,1)*Inf];   % Upper bounds for the 30 variables

    % Active constraints for case I
    % xB <= 0.0105
    ub(1)=0.0105;
    % Mr <= 2800;
    ub(p.NT+7)=2800;
    % F0 = fixed;
    lb(p.NT+8)=p.F0;
    ub(p.NT+8)=p.F0;
```

```
else                    % Case II
    % Constraints
    lb=zeros(p.NT+8,1);                 % Lower bounds for the 30 variables
    ub=[ones(p.NT,1); ones(8,1)*Inf];   % Upper bounds for the 30 variables

    % Active constraints for case II
    % xB <= 0.0105
    ub(1)=0.0105;
    % Mr <= 2800;
    ub(p.NT+7)=2800;
    % V <= Vmax;
    ub(p.NT+2)=p.Vmax;
end

% Initial values of the 30 variables
x0= [ones(1,p.NT)*0.5 10 15 5 5 1.1 0.5 1000 400/60]';

% fmincon options
options = optimset('TolFun',10e-6,'TolCon',10e-6,'MaxFunEvals',1e4,...
'Display','none','Algorithm','sqp','Diagnostics','off'...
);
% fmincon
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
x0=x;

% Results
if p.case_I==1
    casename='case I: min operation cost(energy)\n';
else
    casename='case II: max production rate\n';
end
results_fmincon=sprintf(strcat(...
    casename,...
    'feed rate,             F0[kmol/h]      = %1$0.4d\n',...
    'reactor effluent,      F[kmol/h]       = %2$0.4d\n',...
    'vapor boilup,          V[kmol/h]       = %3$0.4d\n',...
    'reflux,                L[kmol/h]       = %4$0.4d\n',...
    'recycle (distilate),   D[kmol/h]       = %5$0.4d\n',...
    'recycle composition,   xD[molA/mol]    = %6$0.4d\n',...
    'bottom composition,    xB[molA/mol]    = %7$0.4d\n',...
    'reactor composition,   zF[molA/mol]    = %8$0.4d\n',...
    'reactor holdup,        Mr[kmol/h]      = %9$0.4d\n'...
    ),x(p.NT+8)*60,x(p.NT+5)*60,x(p.NT+2)*60,x(p.NT+1)*60, x(p.NT+3)*60,...
    x(p.NT), x(1), x(p.NT+6),x(p.NT+7))
% Nominal results
Vnom=x(p.NT+2)*60           % Nominal vapour boilup
F0n=-x(p.NT+8)*60           % Nominal negative feed
% break

%% Calculation of cost with the Brute force method
if p.case_I==1
%% Calculation of the optimal cost for different F0 values, case I.
VoptF0=zeros(1,17);                         % Preallocation of the optimal
values of V
for i=1:17
    p.F0=(F0s*0.8):(11.5/60):(F0s*1.2);     % Vector of F0-values
    lb(p.NT+8)=p.F0(i);                     % Keeping F0 at a given value
    ub(p.NT+8)=p.F0(i);                     % Keeping F0 at a given value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
```

```
    VoptF0(i)=x(p.NT+2)*60;                    % Vector of optimal values of V
end

% Calculation of the cost when F, L, D, L/D, L/F, L/V, xD, F/F0 and D/F0
% is kept constant at their optimal values with different F0 values

load yopt_nom_case1.mat                        % Matrix with optimal values of
the CVs
% lb(p.NT+5)=A1(1)/60;                         % Keeping F constant
% ub(p.NT+5)=A1(1)/60;                         % Keeping F constant
% lb(p.NT+1)=A1(2)/60;                         % Keeping L constant
% ub(p.NT+1)=A1(2)/60;                         % Keeping L constant
% lb(p.NT+3)=A1(3)/60;                         % Keeping D constant
% ub(p.NT+3)=A1(3)/60;                         % Keeping D constant
% lb(p.NT)=A1(7);                              % Keeping xD constant
% ub(p.NT)=A1(7);                              % Keeping xD constant

% The remaining variables L/D, L/F, L/V, F/F0 and D/F0 was kept constant by
% adding a non-linear equality constraint in the script named nlcon.m
% because the lower and upper bounds can not be used for keeping those
variables constant.

VF0_F=zeros(1,17);                             % Vector of V-values with the
different CVs kept constant
for i=1:17
    p.F0=(F0s*0.8):(11.5/60):(F0s*1.2);        % Vector of F0-values
    lb(p.NT+8)=p.F0(i);                        % Keeping F0 at a given value
    ub(p.NT+8)=p.F0(i);                        % Keeping F0 at a given value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    VF0_F(i)=x(p.NT+2)*60;                     % Vector of V-values with CV
constant
end
% break

%% Calculation of the optimal cost for different zF0 values.
p.F0 = F0s;                                    % Setting feed rate back to
F0=460
lb(p.NT+8)=p.F0;                               % Keeping F0 at the given value
ub(p.NT+8)=p.F0;                               % Keeping F0 at the given value

VoptzF0=zeros(1,11);                           % Preallocation of the optimal
values of V
for i=1:11
    p.zF0=0.80:0.02:1.0;                       % Vector of zF0-values
    p.zF0=p.zF0(i);
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    VoptzF0(i)=x(p.NT+2)*60;                   % Vector of optimal values of V
end


% Calculation of the cost when F, L, D, L/D, L/F, L/V, xD, F/F0 and D/F0
% is kept constant at their optimal values with different zF0 values

load yopt_nom_case1.mat                        % Matrix with optimal values of
the CVs
% lb(p.NT+5)=A1(1)/60;                         % Keeping F constant
% ub(p.NT+5)=A1(1)/60;                         % Keeping F constant
% lb(p.NT+1)=A1(2)/60;                         % Keeping L constant
% ub(p.NT+1)=A1(2)/60;                         % Keeping L constant
% lb(p.NT+3)=A1(3)/60;                         % Keeping D constant
```

```
% ub(p.NT+3)=A1(3)/60;                          % Keeping D constant
% lb(p.NT)=A1(7);                               % Keeping xD constant
% ub(p.NT)=A1(7);                               % Keeping xD constant

% The remaining variables L/D, L/F, L/V, F/F0 and D/F0 was kept constant by
% adding a non-linear equality constraint in the script named nlcon.m
% because the lower and upper bounds can not be used for keeping those
variables constant.

VzF0_DF0=zeros(1,11);                          % Vector of V-values with the
different CVs kept constant
for i=1:11
    p.zF0=0.80:0.02:1.0;                       % Vector of zF0-values
    p.zF0=p.zF0(i);
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    VzF0_DF0(i)=x(p.NT+2)*60;                  % Vector Vs with CVs kept
constant
end
% break

%% Calculation of the cost for implementation errors in c (CVs).
p.F0 = F0s;                                    % F0=460
lb(p.NT+8)=p.F0;
ub(p.NT+8)=p.F0;
p.zF0 = zF0;                                   % zF0=0.9
load yopt_nom_case1.mat                        % Matrix with optimal values of
the CVs

% Implementation error in F
VieF=zeros(1,9);                               % Preallocation of Vs for
implementation error in F
for i=1:9
    p.F=A1(1)*0.8/60: A1(1)*0.05/60: A1(1)*1.2/60;  % Vector of F-values
with +-20% impl. error
    lb(p.NT+5)=p.F(i);                                % Keeping F at given
value
    ub(p.NT+5)=p.F(i);                                % Keeping F at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    VieF(i)=x(p.NT+2)*60;                             % Vector of V-values
for +-20% impl.error in F
end
lb(p.NT+5)=0;
ub(p.NT+5)=Inf;

% Implementation error in L
VieL=zeros(1,9);                                  % Preallocation of Vs
for implementation error in L
for i=1:9
    p.L=A1(2)*0.8/60: A1(2)*0.05/60: A1(2)*1.2/60;  % Vector of L-values
with +-20% impl. error
    lb(p.NT+1)=p.L(i);                                % Keeping L at given
value
    ub(p.NT+1)=p.L(i);                                % Keeping L at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    VieL(i)=x(p.NT+2)*60;                             % Vector of V-values
for +-20% impl.error in L
end
lb(p.NT+1)=0;
```

```
ub(p.NT+1)=Inf;

% Impl.error in D
VieD=zeros(1,9);                                    % Preallocation of Vs
for implementation error in D
for i=1:9
    p.D=A1(3)*0.8/60: A1(3)*0.05/60: A1(3)*1.2/60;  % Vector of D-values
with +-20% impl. error
    lb(p.NT+3)=p.D(i);                              % Keeping D at given
value
    ub(p.NT+3)=p.D(i);                              % Keeping D at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    VieD(i)=x(p.NT+2)*60;                           % Vector of V-values
for +-20% impl.error in D
end
lb(p.NT+3)=0;
ub(p.NT+3)=Inf;

% Impl.error in xD
ViexD=zeros(1,9);                                   % Preallocation of Vs
for implementation error in xD
for i=1:9
    p.xD=A1(7)*0.8: A1(7)*0.05: A1(7)*1.2;          % Vector of xD-values
with +-20% impl. error
    lb(p.NT)=p.xD(i);                               % Keeping xD at given
value
    ub(p.NT)=p.xD(i);                               % Keeping xD at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    ViexD(i)=x(p.NT+2)*60;                          % Vector of V-values
for +-20% impl.error in xD
end
lb(p.NT)=0;
ub(p.NT)=1;

% The operational cost for +-20% implementation errors in the remaining
% variables (L/D, L/F, L/V, F/F0 and D/F0) was found by adding equality
% constraints in the script nlcon.m, and were then typed in below for
% simplicity. Again, this is because the lower and upper bounds could not
% be used to keep ratios constant.

% Impl.error in L/D
Vie_LD=[1278.9 1277.6 1276.6 1276 1275.7 1276 1276.7 1278.1 1280];

% Impl.error in L/F
Vie_LF=[1281.1 1279.0 1277.3 1276.2 1275.7 1276.2 1277.9 1281.1 1286.2];

% Impl.error in L/V
Vie_LV=[1286.7 1283 1279.6 1276.9 1275.7 1277.7 1286.6 1311 1369.1];

% Impl.error in F/F0
Vie_FF0=[7117.3 1460.3 1299.8 1279.1 1275.7 1277.4 1280.9 1285.1 1289.4];

% Impl.error in D/F0
Vie_DF0=[1303.3 1286.7 1279.5 1276.5 1275.7 1276.3 1277.5 1279.2 1281.2];
% break

%% Optimal cost for implementation errors in Mr
p.F0=F0s;
```

```
p.zF0=zF0;

VoptMr=zeros(1,11);                                 % Preallocation of the
optimal values of V
for i=1:11
    p.Mr=2300:50:2800;                              % Vector of Mr-values
(infeasible with Mr>2800 kmol/h)
    lb(p.NT+7)=p.Mr(i);                             % Keeping Mr at given
value
    ub(p.NT+7)=p.Mr(i);                             % Keeping Mr at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    VoptMr(i)=x(p.NT+2)*60;                         % Vector of optimal V-
values with impl.error in Mr
end

% Calculation of the cost when F, L, D, L/D, L/F, L/V, xD, F/F0 and D/F0
% is kept constant at their optimal values with different Mr values

load yopt_nom_case1.mat                             % Matrix with optimal
values of the CVs for case I
% lb(p.NT+5)=A1(1)/60;                              % Keeping F constant
% ub(p.NT+5)=A1(1)/60;                              % Keeping F constant
% lb(p.NT+1)=A1(2)/60;                              % Keeping L constant
% ub(p.NT+1)=A1(2)/60;                              % Keeping L constant
% lb(p.NT+3)=A1(3)/60;                              % Keeping D constant
% ub(p.NT+3)=A1(3)/60;                              % Keeping D constant
% lb(p.NT)=A1(7);                                   % Keeping xD constant
% ub(p.NT)=A1(7);                                   % Keeping xD constant

% The remaining variables L/D, L/F, L/V, F/F0 and D/F0 was kept constant by
% adding a non-linear equality constraint in the script named nlcon.m
% because the lower and upper bounds can not be used for keeping those
variables constant.

VMr_F=zeros(1,11);                                  % Preallocation of V-
values with CVs kept constant
for i=1:11
    p.Mr=2300:50:2800;                              % Vector of Mr-values
    lb(p.NT+7)=p.Mr(i);                             % Keeping Mr at given
value
    ub(p.NT+7)=p.Mr(i);                             % Keeping Mr at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    VMr_F(i)=x(p.NT+2)*60;                          % Vector of V-values
with impl. error in CVs
end
lb(p.NT+7)=0;
ub(p.NT+7)=2800;

%%
% Optimal cost for implementation errors in xB
p.F0=F0s;
p.zF0=zF0;

VoptxB=zeros(1,11);                                 % Preallocation of
optimal values of V with impl. error in xB
for i=1:11
    p.xB=0.009:0.00015:0.0105;                      % Vector of xB-values
(infeasible with xB>0.0105 mol A/mol)
```

```
    lb(1)=p.xB(i);                                  % Keeping xB at given
value
    ub(1)=p.xB(i);                                  % Keeping xB at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    VoptxB(i)=x(p.NT+2)*60;                         % Vector of optimal V-
values with impl. error in xB
end


% Calculation of the cost when F, L, D, L/D, L/F, L/V, xD, F/F0 and D/F0
% is kept constant at their optimal values with different Mr values

load yopt_nom_case1.mat                             % Matrix with optimal
values of the CVs
% lb(p.NT+5)=A1(1)/60;                              % Keeping F constant
% ub(p.NT+5)=A1(1)/60;                              % Keeping F constant
% lb(p.NT+1)=A1(2)/60;                              % Keeping L constant
% ub(p.NT+1)=A1(2)/60;                              % Keeping L constant
% lb(p.NT+3)=A1(3)/60;                              % Keeping D constant
% ub(p.NT+3)=A1(3)/60;                              % Keeping D constant
% lb(p.NT)=A1(7);                                   % Keeping xD constant
% ub(p.NT)=A1(7);                                   % Keeping xD constant

% The remaining variables L/D, L/F, L/V, F/F0 and D/F0 was kept constant by
% adding a non-linear equality constraint in the script named nlcon.m
% because the lower and upper bounds can not be used for keeping those
variables constant.

VxB_F=zeros(1,11);                                  % Preallocation of V-
values with CVs kept constant
for i=1:11
    p.xB=0.009:0.00015:0.0105;                      % Vector of xB-values
    lb(1)=p.xB(i);                                  % Keeping xB at given
value
    ub(1)=p.xB(i);                                  % Keeping xB at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    VxB_F(i)=x(p.NT+2)*60;                          % Vector of V-values
with CVs kept constant
end
lb(1)=0;
ub(1)=1;

else

%% Case II
% Calculation of the optimal cost for different Vmax values.
F0oVm=zeros(1,13);                                  % Preallocation of the
optimal values of F0
for i=1:13
    p.Vmax=1200/60:50/60:1800/60;                   % Vector of Vmax-values
    lb(p.NT+2)=p.Vmax(i);                           % Keeping Vmax at given
value
    ub(p.NT+2)=p.Vmax(i);                           % Keeping Vmax at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0oVm(i)=-x(p.NT+8)*60;                         % Vector of optimval F0-
values
end
```

```
% Calculation of the cost when F, L, D, L/D, L/F, L/V, xD, F/F0 and D/F0
% is kept constant at their optimal values with different Vmax values

load yopt_nom_case2.mat                          % Matrix with optimal
values of the CVs
% lb(p.NT+5)=A4(1)/60;                           % Keeping F constant
% ub(p.NT+5)=A4(1)/60;                           % Keeping F constant
% lb(p.NT+1)=A4(2)/60;                           % Keeping L constant
% ub(p.NT+1)=A4(2)/60;                           % Keeping L constant
% lb(p.NT+3)=A4(3)/60;                           % Keeping D constant
% ub(p.NT+3)=A4(3)/60;                           % Keeping D constant
% lb(p.NT)=A4(7);                                % Keeping xD constant
% ub(p.NT)=A4(7);                                % Keeping xD constant

% The remaining variables L/D, L/F, L/V, F/F0 and D/F0 was kept constant by
% adding a non-linear equality constraint in the script named nlcon.m
% because the lower and upper bounds can not be used for keeping those
variables constant.

F0Vm_CV=zeros(1,13);                             % Preallocation of F0-
values
for i=1:13
    p.Vmax=1200/60:50/60:1800/60;                % Vector of Vmax-values
    lb(p.NT+2)=p.Vmax(i);                        % Keeping Vmax at given
value
    ub(p.NT+2)=p.Vmax(i);                        % Keeping Vmax at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0Vm_CV(i)=-x(p.NT+8)*60;                    % Vector of F0-values with
CVs constant
end
% break

%% Calculation of the optimal cost for different zF0 values.
p.Vmax = Vmaxs;                                  % Vmax=1500
lb(p.NT+2)=p.Vmax;                               % Keeping Vmax at given
value
ub(p.NT+2)=p.Vmax;                               % Keeping Vmax at given
value


F0ozF0=zeros(1,11);                              % Preallocation of optimal
values of F0
for i=1:11
    p.zF0=0.80:0.02:1.0;                         % Vector of zF0-values
    p.zF0=p.zF0(i);
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0ozF0(i)=-x(p.NT+8)*60;                     % Vector of optimal F0-
values
end

% Calculation of the cost when F, L, D, L/D, L/F, L/V, xD, F/F0 and D/F0
% is kept constant at their optimal values with different zF0 values

load yopt_nom_case2.mat                          % Matrix with optimal
values of the CVs
% lb(p.NT+5)=A4(1)/60;                           % Keeping F constant
% ub(p.NT+5)=A4(1)/60;                           % Keeping F constant
% lb(p.NT+1)=A4(2)/60;                           % Keeping L constant
```

```
% ub(p.NT+1)=A4(2)/60;                                % Keeping L constant
% lb(p.NT+3)=A4(3)/60;                                % Keeping D constant
% ub(p.NT+3)=A4(3)/60;                                % Keeping D constant
% lb(p.NT)=A4(7);                                     % Keeping xD constant
% ub(p.NT)=A4(7);                                     % Keeping xD constant

F0zF0_CV=zeros(1,11);                                 % Preallocation of F0-
values
for i=1:11
    p.zF0=0.80:0.02:1.0;                              % Vector with zF0-
values
    p.zF0=p.zF0(i);
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0zF0_CV(i)=-x(p.NT+8)*60;                        % Vector with F0-values
with CVs constant
end
% break
%% Calculation of the cost for implementation errors in c (CVs).
p.zF0 = zF0;                                          % zF0=0.9
p.Vmax=Vmaxs;                                         % Vmax=1500
ub(p.NT+2)=p.Vmax;
load yopt_nom_case2.mat                               % Matrix with optimal
values of the CVs for case 2

% Impl.error in F
F0ieF=zeros(1,9);                                     % Preallocation of F0-
vector
for i=1:9
    p.F=A4(1)*0.8/60: A4(1)*0.05/60: A4(1)*1.2/60;    % Vector of F-values
with +-20% implerror
    lb(p.NT+5)=p.F(i);                                % Keeping F at given
value
    ub(p.NT+5)=p.F(i);                                % Keeping F at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0ieF(i)=-x(p.NT+8)*60;                           % Vector of F0-values
for impl.error in F
end
lb(p.NT+5)=0;
ub(p.NT+5)=Inf;

% Impl.error in L
F0ieL=zeros(1,9);                                     % Preallocation of F0-
vector
for i=1:9
    p.L=A4(2)*0.8/60: A4(2)*0.05/60: A4(2)*1.2/60;    % Vector of L-values
with +-20% implerror
    lb(p.NT+1)=p.L(i);                                % Keeping L at given
value
    ub(p.NT+1)=p.L(i);                                % Keeping L at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0ieL(i)=-x(p.NT+8)*60;                           % Vector of F0-values
for impl.error in L
end
lb(p.NT+1)=0;
ub(p.NT+1)=Inf;

% Impl.error in D
F0ieD=zeros(1,9);                                     % Preallocation of F0-
vector
```

```
for i=1:9
    p.D=A4(3)*0.8/60: A4(3)*0.05/60: A4(3)*1.2/60;  % Vector of D-values
with +-20% implerror
    lb(p.NT+3)=p.D(i);                              % Keeping D at given
value
    ub(p.NT+3)=p.D(i);                              % Keeping D at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0ieD(i)=-x(p.NT+8)*60;                         % Vector of F0-values
for impl.error in D
end
lb(p.NT+3)=0;
ub(p.NT+3)=Inf;

% Impl.error in xD
F0iexD=zeros(1,9);                                  % Preallocation of F0-
vector
for i=1:9
    p.xD=A4(7)*0.8: A4(7)*0.05: A4(7)*1.2;          % Vector of xD-values
with +-20% implerror
    lb(p.NT)=p.xD(i);                               % Keeping xD at given
value
    ub(p.NT)=p.xD(i);                               % Keeping xD at given
value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0iexD(i)=-x(p.NT+8)*60;                        % Vector of F0-values
for impl.error in xD
end
lb(p.NT)=0;
ub(p.NT)=1;

% The operational cost for +-20% implementation errors in the remaining
% variables (L/D, L/F, L/V, F/F0 and D/F0) was found by adding equality
% constraints in the script nlcon.m, and were then typed in below for
% simplicity. Again, this is because the lower and upper bounds could not
% be used to keep ratios constant.

% Impl.error in L/D
F0ie_LD=[-497.2457 -497.5115 -497.7107 -497.8362 -497.8799 -497.8332 -
497.6872 -497.4323 -497.0590];

% Impl.error in L/F
F0ie_LF=[-496.89 -497.2817 -497.5937 -497.8027 -497.8799 -497.7899 -
497.4908 -496.9346 -496.0690];

% Impl.error in L/V
F0ie_LV=[-495.8496 -496.5349 -497.1659 -497.6631 -497.8799 -497.5409 -
496.1306 -492.7389 -486.0753];

% Impl.error in F/F0
F0ie_FF0=[-466.3790 -485.9074 -494.4255 -497.2917 -497.8799 -497.5584 -
496.8714 -496.0507 -495.2004];

% Impl.error in D/F0
F0ie_DF0=[-493.2664 -495.8115 -497.1334 -497.7255 -497.8799 -497.7692 -
497.4975 -497.1291 -496.7046];
% break

%% Optimal cost for implementation errors in Mr
p.zF0=zF0;
```

```
p.Vmax=Vmaxs;
ub(p.NT+2)=p.Vmax;

F0oMr=zeros(1,11);                              % Preallocation of the
optimal values of F0
for i=1:11
    p.Mr=2300:50:2800;                          % Vector of Mr-values
(infeasible with MR>2800 kmol/h)
    lb(p.NT+7)=p.Mr(i);                         % Keeping Mr at given value
    ub(p.NT+7)=p.Mr(i);                         % Keeping Mr at given value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0oMr(i)=-x(p.NT+8)*60;                     % Vector of optimal F0-
values with impl. error in Mr
end

% Calculation of the cost when F, L, D, L/D, L/F, L/V, xD, F/F0 and D/F0
% is kept constant at their optimal values with different Mr values

load yopt_nom_case2.mat                         % Matrix with optimal
values of the CVs
% lb(p.NT+5)=A4(1)/60;                          % Keeping F constant
% ub(p.NT+5)=A4(1)/60;                          % Keeping F constant
% lb(p.NT+1)=A4(2)/60;                          % Keeping L constant
% ub(p.NT+1)=A4(2)/60;                          % Keeping L constant
% lb(p.NT+3)=A4(3)/60;                          % Keeping D constant
% ub(p.NT+3)=A4(3)/60;                          % Keeping D constant
% lb(p.NT)=A4(7);                               % Keeping xD constant
% ub(p.NT)=A4(7);                               % Keeping xD constant

% The remaining variables L/D, L/F, L/V, F/F0 and D/F0 was kept constant by
% adding a non-linear equality constraint in the script named nlcon.m
% because the lower and upper bounds can not be used for keeping those
variables constant.

F0Mr_F=zeros(1,11);                             % Preallocation of F0-
values
for i=1:11
    p.Mr=2300:50:2800;                          % Vector of Mr-values
    lb(p.NT+7)=p.Mr(i);                         % Keeping Mr at given value
    ub(p.NT+7)=p.Mr(i);                         % Keeping Mr at given value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0Mr_F(i)=-x(p.NT+8)*60;                    % Vector of F0-values with
CVs kept constant
end
lb(p.NT+7)=0;
ub(p.NT+7)=2800;
% break
%% Optimal cost for implementation errors in xB
ub(p.NT+7)=2800;        % Mr=2800
p.zF0=zF0;              % zF0=0.9

F0oxB=zeros(1,11);                              % Preallocation of optimal
F0-values
for i=1:11
    p.xB=0.009:0.00015:0.0105;                  % Vector of xB-values
(infeasible with xB>0.0105 mol A/mol)
    lb(1)=p.xB(i);                              % Keeping xB at given value
    ub(1)=p.xB(i);                              % Keeping xB at given value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
```

```
    F0oxB(i)=-x(p.NT+8)*60;                         % Vector of optimal F0-
values with impl. error in xB
end

% Calculation of the cost when F, L, D, L/D, L/F, L/V, xD, F/F0 and D/F0
% is kept constant at their optimal values with different Mr values

load yopt_nom_case2.mat                             % Matrix with optimal
values of the CVs
% lb(p.NT+5)=A4(1)/60;                              % Keeping F constant
% ub(p.NT+5)=A4(1)/60;                              % Keeping F constant
% lb(p.NT+1)=A4(2)/60;                              % Keeping L constant
% ub(p.NT+1)=A4(2)/60;                              % Keeping L constant
% lb(p.NT+3)=A4(3)/60;                              % Keeping D constant
% ub(p.NT+3)=A4(3)/60;                              % Keeping D constant
% lb(p.NT)=A4(7);                                   % Keeping xD constant
% ub(p.NT)=A4(7);                                   % Keeping xD constant

F0xB_DF0=zeros(1,11);                               % Preallocation of F0-
values
for i=1:11
    p.xB=0.009:0.00015:0.0105;                      % Vector of xB-values
    lb(1)=p.xB(i);                                  % Keeping xB at given value
    ub(1)=p.xB(i);                                  % Keeping xB at given value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0xB_DF0(i)=-x(p.NT+8)*60;                      % Vector of F0-values with
CVs kept constant
end
lb(1)=0;
ub(1)=1;
% break
end
```

Script for plotting the losses in case I with the Brute force method, LossBF1.m:

```
clc
clear all
close all

%% Script for plotting the losses with the Brute force method in case I

% Loss for disturbances in F0, with F, L, D, L/D, L/F, L/V, xD, F/F0
% and D/F0 kept constant, respectively

F0=368:11.5:552;

% Loading the saved values of the cost (V) for all the candidate controlled
% variables with disturbances in F0.
load VoptF0.mat
load VF0_F.mat
load VF0_L.mat
load VF0_D.mat
load VF0_LD.mat
load VF0_LF.mat
load VF0_LV.mat
load VF0_xD.mat
load VF0_FF0.mat
load VF0_DF0.mat

% Calculating the loss for each candidate controlled variable
LossF_F0 = [(VF0_F-VoptF0)./VoptF0]*100;

LossL_F0 = [(VF0_L-VoptF0)./VoptF0]*100;

LossD_F0 = [(VF0_D-VoptF0)./VoptF0]*100;

LossLD_F0 = [(VF0_LD-VoptF0)./VoptF0]*100;

LossLF_F0 = [(VF0_LF-VoptF0)./VoptF0]*100;

LossLV_F0 = [(VF0_LV-VoptF0)./VoptF0]*100;

LossxD_F0 = [(VF0_xD-VoptF0)./VoptF0]*100;

LossFF0_F0 = [(VF0_FF0-VoptF0)./VoptF0]*100;

LossDF0_F0 = [(VF0_DF0-VoptF0)./VoptF0]*100;

%%
% Plotting the losses due to disturbance in F0, using the spline function
% to get smoother graphs.
figure (1)
xx=F0;
yy=spline(F0,LossF_F0,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([370 550])
ylim([0 10])
set(gca,'XTick',400:50:550);
set(gca,'YTick',0:2:10);
xlabel('F_0 [kmol/h]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
```

```
title('Case 1 - Large loss','FontSize',12,'FontWeight','bold')
hold on

yy=spline(F0,LossL_F0,xx);
plot(xx,yy,'r','LineWidth',2.5)

yy=spline(F0,LossD_F0,xx);
plot(xx,yy,'g','LineWidth',2.5)

yy=spline(F0,LossLV_F0,xx);
plot(xx,yy,'c','LineWidth',2.5)

yy=spline(F0,LossFF0_F0,xx);
plot(xx,yy,':','LineWidth',2.5)

yy=spline(F0,LossDF0_F0,xx);
plot(xx,yy,'--','LineWidth',2.5)

legend('F','L','D','L/V','F/F0','D/F0')

hold off

figure (2)

yy=spline(F0,LossxD_F0,xx);
plot(xx,yy,'k','LineWidth',2.5)
xlim([370 550])
ylim([0 0.5])
set(gca,'XTick',400:50:550);
set(gca,'YTick',0:0.1:0.5);
xlabel('F_0 [kmol/h]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 1 - Small loss','FontSize',12,'FontWeight','bold')

hold on

yy=spline(F0,LossLD_F0,xx);
plot(xx,yy,'y','LineWidth',2.5)

yy=spline(F0,LossLF_F0,xx);
plot(xx,yy,'m','LineWidth',2.5)

legend('x_D','L/D','L/F')

hold off

%%
% Loss for disturbances in zF0, with F, L, D, L/D, L/F, L/V, xD, F/F0
% and D/F0 kept constant, respectively

zF0=0.80:0.02:1.0;

% Loading the saved values of the cost (V) for all the candidate controlled
% variables with disturbances in zF0.
load VoptzF0.mat
load VzF0_F.mat
load VzF0_L.mat
load VzF0_D.mat
```

```
load VzF0_LD.mat
load VzF0_LF.mat
load VzF0_LV.mat
load VzF0_xD.mat
load VzF0_FF0.mat
load VzF0_DF0.mat

% Calculating the loss for each candidate controlled variable
LossF_zF0 = [(VzF0_F-VoptzF0)./VoptzF0]*100;

LossL_zF0 = [(VzF0_L-VoptzF0)./VoptzF0]*100;

LossD_zF0 = [(VzF0_D-VoptzF0)./VoptzF0]*100;

LossLD_zF0 = [(VzF0_LD-VoptzF0)./VoptzF0]*100;

LossLF_zF0 = [(VzF0_LF-VoptzF0)./VoptzF0]*100;

LossLV_zF0 = [(VzF0_LV-VoptzF0)./VoptzF0]*100;

LossxD_zF0 = [(VzF0_xD-VoptzF0)./VoptzF0]*100;

LossFF0_zF0 = [(VzF0_FF0-VoptzF0)./VoptzF0]*100;

LossDF0_zF0 = [(VzF0_DF0-VoptzF0)./VoptzF0]*100;

%%
% Plotting loss due to disturbance in zF0, using the spline function
% to get smoother graphs.
xx=zF0;

figure (3)

yy=spline(zF0,LossF_zF0,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([0.8 1])
ylim([0 10])
set(gca,'XTick',0.8:0.05:1);
set(gca,'YTick',0:2:10);
xlabel('z_F_0 [-]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 1 - Large loss','FontSize',12,'FontWeight','bold')

hold on

yy=spline(zF0,LossL_zF0,xx);
plot(xx,yy,'r','LineWidth',2.5)

yy=spline(zF0,LossD_zF0,xx);
plot(xx,yy,'g','LineWidth',2.5)

yy=spline(zF0,LossLV_zF0,xx);
plot(xx,yy,'c','LineWidth',2.5)

yy=spline(zF0,LossFF0_zF0,xx);
plot(xx,yy,':','LineWidth',2.5)

yy=spline(zF0,LossDF0_zF0,xx);
```

```
plot(xx,yy,'--','LineWidth',2.5)

legend('F','L','D','L/V','F/F0','D/F0')

hold off

figure (4)

yy=spline(zF0,LossxD_zF0,xx);
plot(xx,yy,'k','LineWidth',2.5)
xlim([0.80 1])
ylim([0 0.5])
set(gca,'XTick',0.8:0.05:1);
set(gca,'YTick',0:0.1:0.5);
xlabel('z_F_0 [kmol/h]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 1 - Small loss','FontSize',12,'FontWeight','bold')

hold on

yy=spline(zF0,LossLD_zF0,xx);
plot(xx,yy,'y','LineWidth',2.5)

yy=spline(zF0,LossLF_zF0,xx);
plot(xx,yy,'m','LineWidth',2.5)

legend('x_D','L/D','L/F')

hold off

% break

%%
% Loss for implementation error in F, L, D, L/D, L/F, L/V, xD, F/F0
% and D/F0, respectively

ie=0.8:0.05:1.2;
% Loading the saved values of the cost (V) for all the candidate controlled
% variables with implementation error in the candidate controlled
variables.
Vopt_ie=1275.7;             % Optimal cost for case I
load VieF.mat
load VieL.mat
load VieD.mat
load Vie_LD.mat
load Vie_LF.mat
load Vie_LV.mat
load ViexD.mat
load Vie_FF0.mat
load Vie_DF0.mat

% Calculating the loss for each candidate controlled variable
LossF_ie = [(VieF-Vopt_ie)./Vopt_ie]*100;

LossL_ie = [(VieL-Vopt_ie)./Vopt_ie]*100;

LossD_ie = [(VieD-Vopt_ie)./Vopt_ie]*100;
```

```
LossLD_ie = [(Vie_LD-Vopt_ie)./Vopt_ie]*100;

LossLF_ie = [(Vie_LF-Vopt_ie)./Vopt_ie]*100;

LossLV_ie = [(Vie_LV-Vopt_ie)./Vopt_ie]*100;

LossxD_ie = [(ViexD-Vopt_ie)./Vopt_ie]*100;

LossFF0_ie = [(Vie_FF0-Vopt_ie)./Vopt_ie]*100;

LossDF0_ie = [(Vie_DF0-Vopt_ie)./Vopt_ie]*100;

%%
% Plotting loss due to implementation error in c, using the spline function
% to get smoother graphs.

figure (5)

xx=ie;
yy=spline(ie,LossF_ie,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([0.8 1.2])
ylim([0 10])
set(gca,'XTick',0.8:0.1:1.2);
set(gca,'YTick',0:2:10);
xlabel('Implementation error [%]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 1 - Large loss','FontSize',12,'FontWeight','bold')
hold on

yy=spline(ie,LossL_ie,xx);
plot(xx,yy,'r','LineWidth',2.5)

yy=spline(ie,LossD_ie,xx);
plot(xx,yy,'g','LineWidth',2.5)

yy=spline(ie,LossLV_ie,xx);
plot(xx,yy,'c','LineWidth',2.5)

yy=spline(ie,LossFF0_ie,xx);
plot(xx,yy,':','LineWidth',2.5)

yy=spline(ie,LossDF0_ie,xx);
plot(xx,yy,'--','LineWidth',2.5)

legend('F','L','D','L/V','F/F0','D/F0')

hold off

figure (6)

yy=spline(ie,LossxD_ie,xx);
plot(xx,yy,'k','LineWidth',2.5)
xlim([0.8 1.2])
ylim([0 0.5])
set(gca,'XTick',0.8:0.1:1.2);
set(gca,'YTick',0:0.1:0.5);
xlabel('Implementation error [%]','FontSize',12,'FontWeight','bold')
```

```
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 1- Small loss','FontSize',12,'FontWeight','bold')

hold on

yy=spline(ie,LossLD_ie,xx);
plot(xx,yy,'y','LineWidth',2.5)

yy=spline(ie,LossLF_ie,xx);
plot(xx,yy,'m','LineWidth',2.5)

legend('x_D','L/D','L/F')

hold off

% break

%%
% Loss for implementation error in Mr with F, L, D, L/D, L/F, L/V, xD, F/F0
% and D/F0 kept constant, respectively

Mr=2300:50:2800;
% Loading the saved values of the cost (V) for all the candidate controlled
% variables with implementation error in Mr.
load VoptMr.mat
load VMr_F.mat
load VMr_L.mat
load VMr_D.mat
load VMr_LD.mat
load VMr_LF.mat
load VMr_LV.mat
load VMr_xD.mat
load VMr_FF0.mat
load VMr_DF0.mat

% Calculating the loss for each candidate controlled variable
LossF_Mr = [(VMr_F-VoptMr)./VoptMr]*100;

LossL_Mr = [(VMr_L-VoptMr)./VoptMr]*100;

LossD_Mr = [(VMr_D-VoptMr)./VoptMr]*100;

LossLD_Mr = [(VMr_LD-VoptMr)./VoptMr]*100;

LossLF_Mr = [(VMr_LF-VoptMr)./VoptMr]*100;

LossLV_Mr = [(VMr_LV-VoptMr)./VoptMr]*100;

LossxD_Mr = [(VMr_xD-VoptMr)./VoptMr]*100;

LossFF0_Mr = [(VMr_FF0-VoptMr)./VoptMr]*100;

LossDF0_Mr = [(VMr_DF0-VoptMr)./VoptMr]*100;

%%
% Plotting loss due to implementation error (back-off) in Mr, using the
spline function
% to get smoother graphs.
```

```
figure (7)

xx=Mr;
yy=spline(Mr,LossF_Mr,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([2300 3300])
ylim([0 10])
set(gca,'XTick',2400:200:3200);
set(gca,'YTick',0:2:10);
xlabel('Implementation error - M_r
[kmol]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 1','FontSize',12,'FontWeight','bold')
hold on

yy=spline(Mr,LossL_Mr,xx);
plot(xx,yy,'r','LineWidth',2.5)

yy=spline(Mr,LossD_Mr,xx);
plot(xx,yy,'g','LineWidth',2.5)

yy=spline(Mr,LossLV_Mr,xx);
plot(xx,yy,'c','LineWidth',2.5)

yy=spline(Mr,LossFF0_Mr,xx);
plot(xx,yy,':','LineWidth',2.5)

yy=spline(Mr,LossDF0_Mr,xx);
plot(xx,yy,'--','LineWidth',2.5)

yy=spline(Mr,LossxD_Mr,xx);
plot(xx,yy,'k','LineWidth',2.5)

yy=spline(Mr,LossLD_Mr,xx);
plot(xx,yy,'y','LineWidth',2.5)

yy=spline(Mr,LossLF_Mr,xx);
plot(xx,yy,'m','LineWidth',2.5)

legend('F','L','D','L/V','F/F0','D/F0','x_D','L/D','L/F')

hold off

%%
% Loss for implementation error in xB with F, L, D, L/D, L/F, L/V, xD, F/F0
% and D/F0 kept constant, respectively

xB=0.009:0.00015:0.0105;
% Loading the saved values of the cost (V) for all the candidate controlled
% variables with implementation error in xB.
load VoptxB.mat
load VxB_F.mat
load VxB_L.mat
load VxB_D.mat
load VxB_LD.mat
load VxB_LF.mat
load VxB_LV.mat
load VxB_xD.mat
load VxB_FF0.mat
```

```
load VxB_DF0.mat

% Calculating the loss for each candidate controlled variable
LossF_xB = [(VxB_F-VoptxB)./VoptxB]*100;

LossL_xB = [(VxB_L-VoptxB)./VoptxB]*100;

LossD_xB = [(VxB_D-VoptxB)./VoptxB]*100;

LossLD_xB = [(VxB_LD-VoptxB)./VoptxB]*100;

LossLF_xB = [(VxB_LF-VoptxB)./VoptxB]*100;

LossLV_xB = [(VxB_LV-VoptxB)./VoptxB]*100;

LossxD_xB = [(VxB_xD-VoptxB)./VoptxB]*100;

LossFF0_xB = [(VxB_FF0-VoptxB)./VoptxB]*100;

LossDF0_xB = [(VxB_DF0-VoptxB)./VoptxB]*100;

%%
% Plotting loss due to implementation error (back-off) in xB, using the
spline function
% to get smoother graphs.
figure (8)

xx=xB;
yy=spline(xB,LossF_xB,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([0.009 0.012])
ylim([0 0.5])
set(gca,'XTick',0.009:0.001:0.012);
set(gca,'YTick',0:0.1:0.5);
xlabel('Implementation error - x_B [-]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 1','FontSize',12,'FontWeight','bold')
hold on

yy=spline(xB,LossL_xB,xx);
plot(xx,yy,'r','LineWidth',2.5)

yy=spline(xB,LossD_xB,xx);
plot(xx,yy,'g','LineWidth',2.5)

yy=spline(xB,LossLV_xB,xx);
plot(xx,yy,'c','LineWidth',2.5)

yy=spline(xB,LossFF0_xB,xx);
plot(xx,yy,':','LineWidth',2.5)

yy=spline(xB,LossDF0_xB,xx);
plot(xx,yy,'--','LineWidth',2.5)

yy=spline(xB,LossxD_xB,xx);
plot(xx,yy,'k','LineWidth',2.5)

yy=spline(xB,LossLD_xB,xx);
```

```
plot(xx,yy,'y','LineWidth',2.5)

yy=spline(xB,LossLF_xB,xx);
plot(xx,yy,'m','LineWidth',2.5)

legend('F','L','D','L/V','F/F0','D/F0','x_D','L/D','L/F')

hold off
```

Script for plotting the loss in case II with the Brute force method, LossBF2.m:

```
clc
clear all
close all

%% Script for plotting the losses with the Brute force method in case II

% Loss for disturbances in Vmax, with F, L, D, L/D, L/F, L/V, xD, F/F0
% and D/F0 kept constant, respectively

Vm=1200:50:1800;

% Loading the saved values of the cost (F0) for all the candidate
controlled
% variables with disturbances in Vmax.
load F0oVm.mat
load F0Vm_F.mat
load F0Vm_L.mat
load F0Vm_D.mat
load F0Vm_LD.mat
load F0Vm_LF.mat
load F0Vm_LV.mat
load F0Vm_xD.mat
load F0Vm_FF0.mat
load F0Vm_DF0.mat

% Calculating the loss for each candidate controlled variable
LossF_Vm = [(F0Vm_F-F0oVm)./abs(F0oVm)]*100;

LossL_Vm = [(F0Vm_L-F0oVm)./abs(F0oVm)]*100;

LossD_Vm = [(F0Vm_D-F0oVm)./abs(F0oVm)]*100;

LossLD_Vm = [(F0Vm_LD-F0oVm)./abs(F0oVm)]*100;

LossLF_Vm = [(F0Vm_LF-F0oVm)./abs(F0oVm)]*100;

LossLV_Vm = [(F0Vm_LV-F0oVm)./abs(F0oVm)]*100;

LossxD_Vm = [(F0Vm_xD-F0oVm)./abs(F0oVm)]*100;

LossFF0_Vm = [(F0Vm_FF0-F0oVm)./abs(F0oVm)]*100;

LossDF0_Vm = [(F0Vm_DF0-F0oVm)./abs(F0oVm)]*100;


%%
% Plotting loss due to disturbance in Vmax, using the spline function
% to get smoother graphs.
figure (1)
xx=Vm;
yy=spline(Vm,LossF_Vm,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([1200 1800])
ylim([0 2])
set(gca,'XTick',1200:200:1800);
set(gca,'YTick',0:0.5:2);
```

```
xlabel('V_m_a_x [kmol/h]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 2 - Large loss','FontSize',12,'FontWeight','bold')
hold on

yy=spline(Vm,LossL_Vm,xx);
plot(xx,yy,'r','LineWidth',2.5)

yy=spline(Vm,LossD_Vm,xx);
plot(xx,yy,'g','LineWidth',2.5)

yy=spline(Vm,LossFF0_Vm,xx);
plot(xx,yy,':','LineWidth',2.5)

yy=spline(Vm,LossDF0_Vm,xx);
plot(xx,yy,'--','LineWidth',2.5)

legend('F','L','D','F/F0','D/F0')

hold off

figure (2)

yy=spline(Vm,LossxD_Vm,xx);
plot(xx,yy,'k','LineWidth',2.5)
xlim([1200 1800])
ylim([0 0.2])
set(gca,'XTick',1200:200:1800);
set(gca,'YTick',0:0.05:0.2);
xlabel('V_m_a_x [kmol/h]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 2 - Small loss','FontSize',12,'FontWeight','bold')

hold on

yy=spline(Vm,LossLD_Vm,xx);
plot(xx,yy,'y','LineWidth',2.5)

yy=spline(Vm,LossLF_Vm,xx);
plot(xx,yy,'m','LineWidth',2.5)

yy=spline(Vm,LossLV_Vm,xx);
plot(xx,yy,'c','LineWidth',2.5)

legend('x_D','L/D','L/F','L/V')

hold off


%%
% Loss for disturbances in zF0, with F, L, D, L/D, L/F, L/V, xD, F/F0
% and D/F0 kept constant, respectively

zF0=0.8:0.02:1.0;

% Loading the saved values of the cost (F0) for all the candidate
controlled
% variables with disturbances in zF0.
```

```
load F0ozF0.mat
load F0zF0_F.mat
load F0zF0_L.mat
load F0zF0_D.mat
load F0zF0_LD.mat
load F0zF0_LF.mat
load F0zF0_LV.mat
load F0zF0_xD.mat
load F0zF0_FF0.mat
load F0zF0_DF0.mat

% Calculating the loss for each candidate controlled variable
LossF_zF0 = [(F0zF0_F-F0ozF0)./abs(F0ozF0)]*100;

LossL_zF0 = [(F0zF0_L-F0ozF0)./abs(F0ozF0)]*100;

LossD_zF0 = [(F0zF0_D-F0ozF0)./abs(F0ozF0)]*100;

LossLD_zF0 = [(F0zF0_LD-F0ozF0)./abs(F0ozF0)]*100;

LossLF_zF0 = [(F0zF0_LF-F0ozF0)./abs(F0ozF0)]*100;

LossLV_zF0 = [(F0zF0_LV-F0ozF0)./abs(F0ozF0)]*100;

LossxD_zF0 = [(F0zF0_xD-F0ozF0)./abs(F0ozF0)]*100;

LossFF0_zF0 = [(F0zF0_FF0-F0ozF0)./abs(F0ozF0)]*100;

LossDF0_zF0 = [(F0zF0_DF0-F0ozF0)./abs(F0ozF0)]*100;

%%
% Plotting loss due to disturbance in zF0, using the spline function
% to get smoother graphs.
xx=zF0;

figure (3)

yy=spline(zF0,LossF_zF0,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([0.8 1])
ylim([0 2])
set(gca,'XTick',0.8:0.05:1);
set(gca,'YTick',0:0.5:2);
xlabel('z_F_0 [-]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 2 - Large loss','FontSize',12,'FontWeight','bold')

hold on

yy=spline(zF0,LossL_zF0,xx);
plot(xx,yy,'r','LineWidth',2.5)

yy=spline(zF0,LossD_zF0,xx);
plot(xx,yy,'g','LineWidth',2.5)

yy=spline(zF0,LossFF0_zF0,xx);
plot(xx,yy,':','LineWidth',2.5)
```

```
yy=spline(zF0,LossDF0_zF0,xx);
plot(xx,yy,'--','LineWidth',2.5)

legend('F','L','D','F/F0','D/F0')

hold off

figure (4)

yy=spline(zF0,LossxD_zF0,xx);
plot(xx,yy,'k','LineWidth',2.5)
xlim([0.80 1])
ylim([0 0.2])
set(gca,'XTick',0.8:0.05:1);
set(gca,'YTick',0:0.05:0.2);
xlabel('z_F_0 [kmol/h]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 2 - Small loss','FontSize',12,'FontWeight','bold')

hold on

yy=spline(zF0,LossLD_zF0,xx);
plot(xx,yy,'y','LineWidth',2.5)

yy=spline(zF0,LossLF_zF0,xx);
plot(xx,yy,'m','LineWidth',2.5)

yy=spline(zF0,LossLV_zF0,xx);
plot(xx,yy,'c','LineWidth',2.5)

legend('x_D','L/D','L/F','L/V')

hold off

%%
% Loss for implementation error in F, L, D, L/D, L/F, L/V, xD, F/F0
% and D/F0, respectively

ie=0.8:0.05:1.2;
F0o_ie=-497.8799;       % Optimal value of cost in case II

% Loading the saved values of the cost (F0) for all the candidate
controlled
% variables with implementatione error in CV.
load F0ieF.mat
load F0ieL.mat
load F0ieD.mat
load F0ie_LD.mat
load F0ie_LF.mat
load F0ie_LV.mat
load F0iexD.mat
load F0ie_FF0.mat
load F0ie_DF0.mat

% Calculating the loss for each candidate controlled variable
LossF_ie = [(F0ieF-F0o_ie)./abs(F0o_ie)]*100;

LossL_ie = [(F0ieL-F0o_ie)./abs(F0o_ie)]*100;
```

```
LossD_ie = [(F0ieD-F0o_ie)./abs(F0o_ie)]*100;

LossLD_ie = [(F0ie_LD-F0o_ie)./abs(F0o_ie)]*100;

LossLF_ie = [(F0ie_LF-F0o_ie)./abs(F0o_ie)]*100;

LossLV_ie = [(F0ie_LV-F0o_ie)./abs(F0o_ie)]*100;

LossxD_ie = [(F0iexD-F0o_ie)./abs(F0o_ie)]*100;

LossFF0_ie = [(F0ie_FF0-F0o_ie)./abs(F0o_ie)]*100;

LossDF0_ie = [(F0ie_DF0-F0o_ie)./abs(F0o_ie)]*100;

%%
% Plotting loss due to implementation error in c, using the spline function
% to get smoother graphs.

figure (5)

xx=ie;
yy=spline(ie,LossF_ie,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([0.8 1.2])
ylim([0 2])
set(gca,'XTick',0.8:0.1:1.2);
set(gca,'YTick',0:0.5:2);
xlabel('Implementation error [%]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 2 - Large loss','FontSize',12,'FontWeight','bold')
hold on

yy=spline(ie,LossL_ie,xx);
plot(xx,yy,'r','LineWidth',2.5)

yy=spline(ie,LossD_ie,xx);
plot(xx,yy,'g','LineWidth',2.5)

yy=spline(ie,LossLV_ie,xx);
plot(xx,yy,'c','LineWidth',2.5)

yy=spline(ie,LossFF0_ie,xx);
plot(xx,yy,':','LineWidth',2.5)

yy=spline(ie,LossDF0_ie,xx);
plot(xx,yy,'--','LineWidth',2.5)

legend('F','L','D','L/V','F/F0','D/F0')

hold off

figure (6)

yy=spline(ie,LossxD_ie,xx);
plot(xx,yy,'k','LineWidth',2.5)
xlim([0.8 1.2])
ylim([0 0.2])
```

```
set(gca,'XTick',0.8:0.1:1.2);
set(gca,'YTick',0:0.05:0.2);
xlabel('Implementation error [%]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 2- Small loss','FontSize',12,'FontWeight','bold')

hold on

yy=spline(ie,LossLD_ie,xx);
plot(xx,yy,'y','LineWidth',2.5)

yy=spline(ie,LossLF_ie,xx);
plot(xx,yy,'m','LineWidth',2.5)

legend('x_D','L/D','L/F')

hold off

%%
% Loss for implementation error in Mr with F, L, D, L/D, L/F, L/V, xD, F/F0
% and D/F0 kept constant, respectively

Mr=2300:50:2800;

% Loading the saved values of the cost (F0) for all the candidate
controlled
% variables with implementation error in Mr.
load F0oMr.mat
load F0Mr_F.mat
load F0Mr_L.mat
load F0Mr_D.mat
load F0Mr_LD.mat
load F0Mr_LF.mat
load F0Mr_LV.mat
load F0Mr_xD.mat
load F0Mr_FF0.mat
load F0Mr_DF0.mat

% Calculating the loss for each candidate controlled variable
LossF_Mr = [(F0Mr_F-F0oMr)./abs(F0oMr)]*100;

LossL_Mr = [(F0Mr_L-F0oMr)./abs(F0oMr)]*100;

LossD_Mr = [(F0Mr_D-F0oMr)./abs(F0oMr)]*100;

LossLD_Mr = [(F0Mr_LD-F0oMr)./abs(F0oMr)]*100;

LossLF_Mr = [(F0Mr_LF-F0oMr)./abs(F0oMr)]*100;

LossLV_Mr = [(F0Mr_LV-F0oMr)./abs(F0oMr)]*100;

LossxD_Mr = [(F0Mr_xD-F0oMr)./abs(F0oMr)]*100;

LossFF0_Mr = [(F0Mr_FF0-F0oMr)./abs(F0oMr)]*100;

LossDF0_Mr = [(F0Mr_DF0-F0oMr)./abs(F0oMr)]*100;

%%
```

```
% Plotting loss due to implementation error (back-off) in Mr, using the
spline function
% to get smoother graphs.
figure (7)

xx=Mr;
yy=spline(Mr,LossF_Mr,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([2400 3200])
ylim([0 0.1])
set(gca,'XTick',2400:200:3200);
set(gca,'YTick',0:0.02:0.1);
xlabel('Implementation error - M_r
[kmol]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 2','FontSize',12,'FontWeight','bold')
hold on

yy=spline(Mr,LossL_Mr,xx);
plot(xx,yy,'r','LineWidth',2.5)

yy=spline(Mr,LossD_Mr,xx);
plot(xx,yy,'g','LineWidth',2.5)

yy=spline(Mr,LossLV_Mr,xx);
plot(xx,yy,'c','LineWidth',2.5)

yy=spline(Mr,LossFF0_Mr,xx);
plot(xx,yy,':','LineWidth',2.5)

yy=spline(Mr,LossDF0_Mr,xx);
plot(xx,yy,'--','LineWidth',2.5)

yy=spline(Mr,LossxD_Mr,xx);
plot(xx,yy,'k','LineWidth',2.5)

yy=spline(Mr,LossLD_Mr,xx);
plot(xx,yy,'y','LineWidth',2.5)

yy=spline(Mr,LossLF_Mr,xx);
plot(xx,yy,'m','LineWidth',2.5)

legend('F','L','D','L/V','F/F0','D/F0','x_D','L/D','L/F')

hold off

%%
% Loss for implementation error in xB with F, L, D, L/D, L/F, L/V, xD, F/F0
% and D/F0 kept constant, respectively

xB=0.009:0.00015:0.0105;

% Loading the saved values of the cost (F0) for all the candidate
controlled
% variables with implementation error in xB.
load F0oxB.mat
load F0xB_F.mat
load F0xB_L.mat
load F0xB_D.mat
```

```
load F0xB_LD.mat
load F0xB_LF.mat
load F0xB_LV.mat
load F0xB_xD.mat
load F0xB_FF0.mat
load F0xB_DF0.mat

% Calculating the loss for each candidate controlled variable
LossF_xB = [(F0xB_F-F0oxB)./abs(F0oxB)]*100;

LossL_xB = [(F0xB_L-F0oxB)./(F0oxB)]*100;

LossD_xB = [(F0xB_D-F0oxB)./(F0oxB)]*100;

LossLD_xB = [(F0xB_LD-F0oxB)./(F0oxB)]*100;

LossLF_xB = [(F0xB_LF-F0oxB)./(F0oxB)]*100;

LossLV_xB = [(F0xB_LV-F0oxB)./(F0oxB)]*100;

LossxD_xB = [(F0xB_xD-F0oxB)./(F0oxB)]*100;

LossFF0_xB = [(F0xB_FF0-F0oxB)./(F0oxB)]*100;

LossDF0_xB = [(F0xB_DF0-F0oxB)./(F0oxB)]*100;

%%
% Plotting loss due to implementation error (back-off) in xB, using the
spline function
% to get smoother graphs.
figure (8)

xx=xB;
yy=spline(xB,LossF_xB,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([0.009 0.012])
ylim([0 0.1])
set(gca,'XTick',0.009:0.001:0.012);
set(gca,'YTick',0:0.02:0.1);
xlabel('Implementation error - x_B [-]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Case 2','FontSize',12,'FontWeight','bold')
hold on

yy=spline(xB,LossL_xB,xx);
plot(xx,yy,'r','LineWidth',2.5)

yy=spline(xB,LossD_xB,xx);
plot(xx,yy,'g','LineWidth',2.5)

yy=spline(xB,LossLV_xB,xx);
plot(xx,yy,'c','LineWidth',2.5)

yy=spline(xB,LossFF0_xB,xx);
plot(xx,yy,':','LineWidth',2.5)

yy=spline(xB,LossDF0_xB,xx);
plot(xx,yy,'--','LineWidth',2.5)
```

```
yy=spline(xB,LossxD_xB,xx);
plot(xx,yy,'k','LineWidth',2.5)

yy=spline(xB,LossLD_xB,xx);
plot(xx,yy,'y','LineWidth',2.5)

yy=spline(xB,LossLF_xB,xx);
plot(xx,yy,'m','LineWidth',2.5)

legend('F','L','D','L/V','F/F0','D/F0','x_D','L/D','L/F')

hold off
```

Script for calculating the cost with the null space and exact local method, testScript_NP_EL.m:

```
% Script for optimization of reactor, separator and recycle process.
% The numerical description of the process is taken from Larsson et al
(2003)

clc
clear all
global p;

% Column parameters
p.qF    = 1;            % Liquid fraction in column feed [-]
p.NT    = 22;           % Number of stages in distillation column [-]
p.NF    = 13;           % Feed stage in distillation column [-]
p.alpha = 2;            % Relative volatility between component A and B [-]

Vmaxs=(1500/60);        % The optimal value of vapour boilup in case II
[kmol/h]
p.Vmax  = Vmaxs;        % Making the vapour boilup a global parameter
[kmol/h]
p.xB=0.0105;            % Bottom composition (original value) [mol A/mol]
p.F=958;                % Column feed (original value) [kmol/h]
p.L=778;                % Reflux (original value) [kmol/h]
p.D=498;                % Recycle (original value) [kmol/h]
p.LD=1.6;               % Ratio L/D (original value) [-]
p.LF=0.8;               % Ratio L/F (original value) [-]
p.LV=0.61;              % Ratio L/V (original value) [-]
p.xD=0.82;              % Distillate composition (original value) [mol
A/mol]
p.FF0=2;                % Ratio F/F0 (original value) [-]
p.DF0=2;                % Ratio D/F0 (original value) [-]

% CSTR parameters
F0s = (460/60);         % The optimal value of reactor feed (case I)
[kmol/h]
zF0 = 0.9;              % The optimal value of reactor feed composition
[mol A/mol]

p.F0 = F0s;             % Making the reactor feed a global parameter
[kmol/h]
p.zF0 = zF0;            % Making the reactor feed a global parameter [-]
p.k1 = 0.341/60;        % Reaction rate constant [min^-1]
p.Mr=2800;              % Liquid holdup in reactor [kmol/h]
% Flags
p.OPTI=0;
p.case_I=0;             % Switching between case I and case II

if p.case_I==1          % Case I
    % Constraints
    lb=zeros(p.NT+8,1);                 % Lower bounds for the 30 variables
    ub=[ones(p.NT,1); ones(8,1)*Inf];   % Upper bounds for the 30 variables

    % Active constraints for case I
    % xB <= 0.0105
    ub(1)=0.0105;
    % Mr <= 2800;
    ub(p.NT+7)=2800;
    % F0 = fixed;
```

```
    lb(p.NT+8)=p.F0;
    ub(p.NT+8)=p.F0;

else                    % Case II
    % Constraints
    lb=zeros(p.NT+8,1);                 % Lower bounds for the 30 variables
    ub=[ones(p.NT,1); ones(8,1)*Inf];   % Upper bounds for the 30 variables

    % Active constraints for case II
    % xB <= 0.0105
    ub(1)=0.0105;
    % Mr <= 2800;
    ub(p.NT+7)=2800;
    % V <= Vmax;
    ub(p.NT+2)=p.Vmax;
end

% Initial values of the 30 variables
x0= [ones(1,p.NT)*0.5 10 15 5 5 1.1 0.5 1000 400/60]';

% fmincon options
options = optimset('TolFun',10e-6,'TolCon',10e-6,'MaxFunEvals',1e4,...
'Display','none','Algorithm','sqp','Diagnostics','off'...
);
% fmincon
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
x0=x;

% Results
if p.case_I==1
    casename='case I: min operation cost(energy)\n';
else
    casename='case II: max production rate\n';
end
results_fmincon=sprintf(strcat(...
    casename,...
    'feed rate,             F0[kmol/h]      = %1$0.4d\n',...
    'reactor effluent,      F[kmol/h]       = %2$0.4d\n',...
    'vapor boilup,          V[kmol/h]       = %3$0.4d\n',...
    'reflux,                L[kmol/h]       = %4$0.4d\n',...
    'recycle (distilate),   D[kmol/h]       = %5$0.4d\n',...
    'recycle composition,   xD[molA/mol]    = %6$0.4d\n',...
    'bottom composition,    xB[molA/mol]    = %7$0.4d\n',...
    'reactor composition,   zF[molA/mol]    = %8$0.4d\n',...
    'reactor holdup,        Mr[kmol/h]      = %9$0.4d\n'...
    ),x(p.NT+8)*60,x(p.NT+5)*60,x(p.NT+2)*60,x(p.NT+1)*60, x(p.NT+3)*60,...
    x(p.NT), x(1), x(p.NT+6),x(p.NT+7))
% Nominal results
Vnom=x(p.NT+2)*60           % Nominal vapour boilup
F0n=-x(p.NT+8)*60           % Nominal negative feed
% break

%% Null space method

if p.case_I==1

%% Case I

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
```

```
% Measurements for the nominal case I, the measurements are xB,
% x6, x10, x14, x18, xD, L, V, D, B, F, F0, in that order.
y0=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];

% dd1
pert=0.01;                                  % Magnitude of disturbance, 10%
dd1=p.F0*pert;                              % Disturbance in F0
p.F0 = F0s+dd1;                             % New value of F0 with
disturbance

lb(p.NT+8)=p.F0;                            % Keeping F0 at given value
ub(p.NT+8)=p.F0;                            % Keeping F0 at given value

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
% Calculating new values of the measurements.
ydd1=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];

%% dd2
p.F0 = F0s;                                 % Setting F0 back to optimal
value
lb(p.NT+8)=p.F0;                            % Keeping F0 at given value
ub(p.NT+8)=p.F0;                            % Keeping F0 at given value

dd2=zF0*pert;                               % Disturbance in zF0
p.zF0 = zF0 +dd2;                           % New value of zF0 with
disturbance

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
% Calculating new values of the measurements.
ydd2=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
```

```
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];

else
%% Case II

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
% Measurements for the nominal case II, the measurements are xB,
% x6, x10, x14, x18, xD, L, V, D, B, F, F0, in that order.
y0=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];

% dd3
pert=0.01;                                  % Magnitude of disturbance, 10%
dd3=p.Vmax*pert;                            % Disturbance in Vmax
p.Vmax = Vmaxs+dd3;                         % New value of Vmax with
disturbance
ub(p.NT+2)=p.Vmax;                          % Keeping Vmax at given value

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
% Calculating new values of the measurements.
ydd3=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];
% dd4
p.Vmax=Vmaxs;                               % Setting Vmax back to optimal
value
dd4=zF0*pert;                               % Disturbance in zF0
p.zF0 = zF0 +dd4;                           % New value of zF0 with
disturbance
ub(p.NT+2)=p.Vmax;                          % Keeping Vmax at given value

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
% Calculating new values of the measurements.
ydd4=[x(1);
```

```
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];
end

if p.case_I==1
% Case I
F1=[(ydd1-y0)/dd1 (ydd2-y0)/dd2];           % Optimal sensitivity matrix
case I
H1=null(F1');                               % Finding the H matrix
H1=H1';                                     % Finding the H matrix
H11=H1(5,:);                                % Selecting on of the rows
cs11=H11*y0;                                % Setpoint for the measurement
combination
% cs11=0.0272;                              % Calculated set-point of cs11
else
% Case II
F2=[(ydd3-y0)/dd3 (ydd4-y0)/dd4];           % Optimal sensitivity matrix
case II
H2=null(F2');                               % Finding the H matrix
H2b=H2';                                    % Finding the H matrix
H2=H2b(4,:);                                % Selecting on of the rows
cs2=H2*y0;                                  % Setpoint for the measurement
combination
% cs2=0.8315                                % Calculated set-point of cs2
end
% break

%% Exact local method
if p.case_I==1
% Case1
p.F0 = F0s;
lb(p.NT+8)=p.F0;
ub(p.NT+8)=p.F0;
p.zF0 = zF0;

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);

% Measurements for the nominal case I, the measurements are xB,
% x6, x10, x14, x18, xD, L, V, D, B, F, F0, in that order.
y0=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];
```

```
F0=460/60;                                  % Optimal value of F0
% dd=[0.01*0.9*ones(6,1); 0.01*F0*ones(6,1)];
dd=0.01*F0*ones(12,1);                      % Magnitude of disturbance in
F0
Wd=diag(dd);                                % Diagonal scaling matrix for
disturbances

pert=0.02;                                  % Magnitude of measurement
error, 2%
Wn=diag([0.01; 0.01; 0.01; 0.01; 0.01; 0.01; pert*y0(7:12)]);   % Diagonal
scaling matrix for measurement noise

% Finding Gy, the steady-state gain matrix
du=y0*0.01;                                 % Input change, 1%

%Finding dy/dL
lb(p.NT+1)=(y0(7)+du(7))/60;                % Increasing L with 1% from the
optimal value
ub(p.NT+1)=(y0(7)+du(7))/60;                % Increasing L with 1% from the
optimal value

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
% Calculating the new measurements with step in L
yL=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];

dydL=(yL-y0)/du(7);                         % Calculating dy/dL

%Finding dy/dV
lb(p.NT+1)=0;                               % Setting L back to its optimal
value
ub(p.NT+1)=Inf;                             % Setting L back to its optimal
value
lb(p.NT+2)=(y0(8)+du(8))/60;                % Increasing V with 1% from the
optimal value
ub(p.NT+2)=(y0(8)+du(8))/60;                % Increasing V with 1% from the
optimal value

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
% Calculating the new measurements with step in V
yV=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
```

```
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];

dydV=(yV-y0)/du(8);                         % Calculating dy/dV

%Finding dy/dF
lb(p.NT+2)=0;                               % Setting V back to its optimal
value
ub(p.NT+2)=Inf;                             % Setting V back to its optimal
value
lb(p.NT+5)=(y0(11)+du(11))/60;              % Increasing F with 1% from the
optimal value
ub(p.NT+5)=(y0(11)+du(11))/60;              % Increasing F with 1% from the
optimal value

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
% Calculating the new measurements with step in F.
yF=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];

dydF=(yF-y0)/du(11);                        % Calculating dy/dV

lb(p.NT+5)=0;                               % Setting F back to its optimal
value
ub(p.NT+5)=Inf;                             % Setting F back to its optimal
value

Gy=[dydL dydV dydF];                        % Steady-state gain matrix
dy/du

Y=(F1(:,1))'*Wd*Wn;                         % Finding the matrix Y=F'*Wd*Wn
H4=(inv(Y*Y')*Gy)';                         % Finding the matrix H
Hel=H4(1,:);                                % Selecting one of the rows
cs4=Hel*y0;                                 % Set-point of the measurement
combination
%cs4=-0.0011;                               % The calculated set-point

else

%% Case2
p.zF0 = zF0;
p.Vmax = Vmaxs;

% Running optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
```

```
% Measurements for the nominal case II, the measurements are xB,
% x6, x10, x14, x18, xD, L, V, D, B, F, F0, in that order.
y0=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];

Vm=1500/60;                                 % Optimal value of Vmax
dd=0.01*Vm*ones(12,1);                      % Magnitude of disturbance in
Vmax
Wd=diag(dd);                                % Diagonal scaling matrix for
disturbances

pert=0.02;
Wn=diag([0.01; 0.01; 0.01; 0.01; 0.01; 0.01; pert*y0(7:12)]);   % Diagonal
scaling matrix for measurement noise

%Finding Gy, the steady-state gain matrix
% du=u0*0.01;
du=y0*0.01;                                 % Input change, 1%

%Finding dy/dL
lb(p.NT+1)=(y0(7)+du(7))/60;                % Increasing L with 1% from the
optimal value
ub(p.NT+1)=(y0(7)+du(7))/60;                % Increasing L with 1% from the
optimal value

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);

% Calculating the new measurements with step in L.
yL=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];

dydL=(yL-y0)/du(7);                         % Calculating dy/dL

%Finding dy/dF
lb(p.NT+1)=0;                               % Setting L back to its optimal
value
ub(p.NT+1)=Inf;                             % Setting L back to its optimal
value
```

```
lb(p.NT+5)=(y0(11)+du(11))/60;              % Increasing F with 1% from the
optimal value
ub(p.NT+5)=(y0(11)+du(11))/60;              % Increasing F with 1% from the
optimal value

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);

% Calculating the new measurements with step in F
yF=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];

dydF=(yF-y0)/du(11);                        % Calculating dy/dF

%Finding dy/dF0
lb(p.NT+5)=0;                               % Setting F back to its optimal
value
ub(p.NT+5)=Inf;                             % Setting F back to its optimal
value
lb(p.NT+8)=(y0(12)+du(12))/60;              % Increasing F0 with 1% from
the optimal value
ub(p.NT+8)=(y0(12)+du(12))/60;              % Increasing F0 with 1% from
the optimal value

% Running the optimization
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);

% Calculating the new measurements
yF0=[x(1);
    x(6);
    x(10);
    x(14);
    x(18);
    x(22);
    x(23)*60;
    x(24)*60;
    x(25)*60;
    x(26)*60;
    x(27)*60;
    x(30)*60];

dydF0=(yF0-y0)/du(12);                      % Calculating dy/dF0

lb(p.NT+8)=0;                               % Setting F0 back to its
optimal value
ub(p.NT+8)=Inf;                             % Setting F0 back to its
optimal value

Gy=[dydL dydF dydF0];                       % Steady-state gain matrix
dy/du
```

```
Y=F2(1:12)*Wd*Wn;                       % Finding the matrix Y=F'*Wd*Wn
H5=(inv(Y*Y')*Gy)';                     % Finding the matrix H
Hel2=H5(1,:);                           % Selecting one of the rows
cs5=Hel2*y0                             % Set-point of the measurement
combination
%cs5=-0.0030                            % The calculated set-point
end
```

Script for plotting the loss with the null space method, LossPlotNP.m:

```
% Script for optimization of reactor, separator and recycle process.
% The numerical description of the process is taken from Larsson et al
(2003)

clc
clear all
global p;

% Column parameters
p.qF    = 1;            % Liquid fraction in column feed [-]
p.NT    = 22;           % Number of stages in distillation column [-]
p.NF    = 13;           % Feed stage in distillation column [-]
p.alpha = 2;            % Relative volatility between component A and B [-]

Vmaxs=(1500/60);        % The optimal value of vapour boilup in case II
[kmol/h]
p.Vmax  = Vmaxs;        % Making the vapour boilup a global parameter
[kmol/h]
p.xB=0.0105;            % Bottom composition (original value) [mol A/mol]
p.F=958;                % Column feed (original value) [kmol/h]
p.L=778;                % Reflux (original value) [kmol/h]
p.D=498;                % Recycle (original value) [kmol/h]
p.LD=1.6;               % Ratio L/D (original value) [-]
p.LF=0.8;               % Ratio L/F (original value) [-]
p.LV=0.61;              % Ratio L/V (original value) [-]
p.xD=0.82;              % Distillate composition (original value) [mol
A/mol]
p.FF0=2;                % Ratio F/F0 (original value) [-]
p.DF0=2;                % Ratio D/F0 (original value) [-]

% CSTR parameters
F0s = (460/60);         % The optimal value of reactor feed (case I)
[kmol/h]
zF0 = 0.9;              % The optimal value of reactor feed composition
[mol A/mol]

p.F0 = F0s;             % Making the reactor feed a global parameter
[kmol/h]
p.zF0 = zF0;            % Making the reactor feed a global parameter [-]
p.k1 = 0.341/60;        % Reaction rate constant [min^-1]
p.Mr=2800;              % Liquid holdup in reactor [kmol/h]
% Flags
p.OPTI=0;
p.case_I=0;             % Switching between case I and case II

if p.case_I==1          % Case I
    % Constraints
    lb=zeros(p.NT+8,1);                 % Lower bounds for the 30 variables
    ub=[ones(p.NT,1); ones(8,1)*Inf];   % Upper bounds for the 30 variables

    % Active constraints for case I
    % xB <= 0.0105
    ub(1)=0.0105;
    % Mr <= 2800;
    ub(p.NT+7)=2800;
    % F0 = fixed;
    lb(p.NT+8)=p.F0;
    ub(p.NT+8)=p.F0;
```

```
else                    % Case II
    % Constraints
    lb=zeros(p.NT+8,1);                 % Lower bounds for the 30 variables
    ub=[ones(p.NT,1); ones(8,1)*Inf];   % Upper bounds for the 30 variables

    % Active constraints for case II
    % xB <= 0.0105
    ub(1)=0.0105;
    % Mr <= 2800;
    ub(p.NT+7)=2800;
    % V <= Vmax;
    ub(p.NT+2)=p.Vmax;
end

% Initial values of the 30 variables
x0= [ones(1,p.NT)*0.5 10 15 5 5 1.1 0.5 1000 400/60]';

% fmincon options
options = optimset('TolFun',10e-6,'TolCon',10e-6,'MaxFunEvals',1e4,...
'Display','none','Algorithm','sqp','Diagnostics','off'...
);
% fmincon
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
x0=x;

% Results
if p.case_I==1
    casename='case I: min operation cost(energy)\n';
else
    casename='case II: max production rate\n';
end
results_fmincon=sprintf(strcat(...
    casename,...
    'feed rate,             F0[kmol/h]      = %1$0.4d\n',...
    'reactor effluent,      F[kmol/h]       = %2$0.4d\n',...
    'vapor boilup,          V[kmol/h]       = %3$0.4d\n',...
    'reflux,                L[kmol/h]       = %4$0.4d\n',...
    'recycle (distilate),   D[kmol/h]       = %5$0.4d\n',...
    'recycle composition,   xD[molA/mol]    = %6$0.4d\n',...
    'bottom composition,    xB[molA/mol]    = %7$0.4d\n',...
    'reactor composition,   zF[molA/mol]    = %8$0.4d\n',...
    'reactor holdup,        Mr[kmol/h]      = %9$0.4d\n'...
    ),x(p.NT+8)*60,x(p.NT+5)*60,x(p.NT+2)*60,x(p.NT+1)*60, x(p.NT+3)*60,...
    x(p.NT), x(1), x(p.NT+6),x(p.NT+7))
% Nominal results
Vnom=x(p.NT+2)*60           % Nominal vapour boilup
F0n=-x(p.NT+8)*60           % Nominal negative feed
% break

%% Plotting loss with the Null Space Method and Exact Local Method
% In order for the plots to be made correctly, the equality constraints
% with the septoints found above need to be added to nlcon.m for each of
% the plots.
if p.case_I==1

%% Null Space Method, Case I
    Vnp=zeros(1,17);                        % Preallocation of V-values
for i=1:17
    p.F0=(F0s*0.8):(11.5/60):(F0s*1.2);     % Vector of F0-values
```

```
    lb(p.NT+8)=p.F0(i);                         % Keeping F0 at given value
    ub(p.NT+8)=p.F0(i);                         % Keeping F0 at given value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    Vnp(i)=x(p.NT+2)*60;                        % Cost with the null space
method
end
load VoptF0                                     % Optimal cost for disturbance
in F0, saved earlier.
loss_np=((Vnp-VoptF0)./VoptF0)*100;             % Calculating the loss
F0=368:11.5:552;                                % F0-values for the x-axis
% Making the plot of the loss versus F0
figure (1)
xx=F0;
yy=spline(F0,loss_np,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([370 550])
ylim([0 0.002])
set(gca,'XTick',400:50:550);
set(gca,'YTick',0:0.0004:0.002);
xlabel('F_0 [kmol/h]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Null Space Method - Case 1','FontSize',12,'FontWeight','bold')

else
%% Null Space case II
    F0np=zeros(1,13);                           % Preallocation of F0-values
for i=1:13
     p.Vmax=1200/60:50/60:1800/60;              % Vector of Vmax-values
    lb(p.NT+2)=p.Vmax(i);                       % Keeping Vmax at given value
    ub(p.NT+2)=p.Vmax(i);                       % Keeping Vmax at given value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0np(i)=-x(p.NT+8)*60;                      % Cost with the null space
method
end
load F0oVm                                      % Optimal cost for disturbance
in Vmax, saved earlier.
loss_np=((F0np-F0oVm)./abs(F0oVm))*100;         % Calculating the loss
Vm=1200:50:1800;                                % Vmax-values for the x-axis
% Making the plot of the loss versus Vmax
figure (3)
xx=Vm;
yy=spline(Vm,loss_np,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([1200 1800])
ylim([0 0.002])
set(gca,'XTick',1200:200:1800);
set(gca,'YTick',0:0.0004:0.002);
xlabel('V_m_a_x [kmol/h]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Null Space Method - Case 2','FontSize',12,'FontWeight','bold')
end
```

Script for plotting the loss with the exact local method, LossPlotEL.m:

```
% Script for optimization of reactor, separator and recycle process.
% The numerical description of the process is taken from Larsson et al
(2003)

clc
clear all
global p;

% Column parameters
p.qF    = 1;            % Liquid fraction in column feed [-]
p.NT    = 22;           % Number of stages in distillation column [-]
p.NF    = 13;           % Feed stage in distillation column [-]
p.alpha = 2;            % Relative volatility between component A and B [-]

Vmaxs=(1500/60);        % The optimal value of vapour boilup in case II
[kmol/h]
p.Vmax  = Vmaxs;        % Making the vapour boilup a global parameter
[kmol/h]
p.xB=0.0105;            % Bottom composition (original value) [mol A/mol]
p.F=958;                % Column feed (original value) [kmol/h]
p.L=778;                % Reflux (original value) [kmol/h]
p.D=498;                % Recycle (original value) [kmol/h]
p.LD=1.6;               % Ratio L/D (original value) [-]
p.LF=0.8;               % Ratio L/F (original value) [-]
p.LV=0.61;              % Ratio L/V (original value) [-]
p.xD=0.82;              % Distillate composition (original value) [mol
A/mol]
p.FF0=2;                % Ratio F/F0 (original value) [-]
p.DF0=2;                % Ratio D/F0 (original value) [-]

% CSTR parameters
F0s = (460/60);         % The optimal value of reactor feed (case I)
[kmol/h]
zF0 = 0.9;              % The optimal value of reactor feed composition
[mol A/mol]

p.F0 = F0s;             % Making the reactor feed a global parameter
[kmol/h]
p.zF0 = zF0;            % Making the reactor feed a global parameter [-]
p.k1 = 0.341/60;        % Reaction rate constant [min^-1]
p.Mr=2800;              % Liquid holdup in reactor [kmol/h]
% Flags
p.OPTI=0;
p.case_I=0;             % Switching between case I and case II

if p.case_I==1          % Case I
    % Constraints
    lb=zeros(p.NT+8,1);                 % Lower bounds for the 30 variables
    ub=[ones(p.NT,1); ones(8,1)*Inf];   % Upper bounds for the 30 variables

    % Active constraints for case I
    % xB <= 0.0105
    ub(1)=0.0105;
    % Mr <= 2800;
    ub(p.NT+7)=2800;
    % F0 = fixed;
    lb(p.NT+8)=p.F0;
    ub(p.NT+8)=p.F0;
```

```
else                    % Case II
    % Constraints
    lb=zeros(p.NT+8,1);                 % Lower bounds for the 30 variables
    ub=[ones(p.NT,1); ones(8,1)*Inf];   % Upper bounds for the 30 variables

    % Active constraints for case II
    % xB <= 0.0105
    ub(1)=0.0105;
    % Mr <= 2800;
    ub(p.NT+7)=2800;
    % V <= Vmax;
    ub(p.NT+2)=p.Vmax;
end

% Initial values of the 30 variables
x0= [ones(1,p.NT)*0.5 10 15 5 5 1.1 0.5 1000 400/60]';

% fmincon options
options = optimset('TolFun',10e-6,'TolCon',10e-6,'MaxFunEvals',1e4,...
'Display','none','Algorithm','sqp','Diagnostics','off'...
);
% fmincon
[x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
x0=x;

% Results
if p.case_I==1
    casename='case I: min operation cost(energy)\n';
else
    casename='case II: max production rate\n';
end
results_fmincon=sprintf(strcat(...
    casename,...
    'feed rate,             F0[kmol/h]      = %1$0.4d\n',...
    'reactor effluent,      F[kmol/h]       = %2$0.4d\n',...
    'vapor boilup,          V[kmol/h]       = %3$0.4d\n',...
    'reflux,                L[kmol/h]       = %4$0.4d\n',...
    'recycle (distilate),   D[kmol/h]       = %5$0.4d\n',...
    'recycle composition,   xD[molA/mol]    = %6$0.4d\n',...
    'bottom composition,    xB[molA/mol]    = %7$0.4d\n',...
    'reactor composition,   zF[molA/mol]    = %8$0.4d\n',...
    'reactor holdup,        Mr[kmol/h]      = %9$0.4d\n'...
    ),x(p.NT+8)*60,x(p.NT+5)*60,x(p.NT+2)*60,x(p.NT+1)*60, x(p.NT+3)*60,...
    x(p.NT), x(1), x(p.NT+6),x(p.NT+7))
% Nominal results
Vnom=x(p.NT+2)*60           % Nominal vapour boilup
F0n=-x(p.NT+8)*60           % Nominal negative feed
% break

%% Plotting loss with the Null Space Method and Exact Local Method
% In order for the plots to be made correctly, the equality constraints
% with the septoints found above need to be added to nlcon.m for each of
% the plots.
if p.case_I==1
    %% Exact Local Method, Case I
    Vel=zeros(1,17);                        % Preallocation of V-values
for i=1:17
    p.F0=(F0s*0.8):(11.5/60):(F0s*1.2);     % Vector of F0-values
    lb(p.NT+8)=p.F0(i);                     % Keeping F0 at given value
```

```
    ub(p.NT+8)=p.F0(i);                         % Keeping F0 at given value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    Vel(i)=x(p.NT+2)*60;                        % Cost with the exact local
method
end
load VoptF0                                     % Optimal cost for disturbance
in F0, saved earlier.
loss_el=((Vel-VoptF0)./VoptF0)*100;             % Calculating the loss
F0=368:11.5:552;                                % F0 values for the x-axis
% Making the plot of the loss versus F0
figure (2)
xx=F0;
yy=spline(F0,loss_el,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([370 550])
ylim([0 1])
set(gca,'XTick',400:50:550);
set(gca,'YTick',0:0.2:1);
xlabel('F_0 [kmol/h]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Exact Local Method - Case 1','FontSize',12,'FontWeight','bold')

else
    %% Exact Local Case II
    F0el=zeros(1,13);                           % Preallocation of F0-values
for i=1:13
     p.Vmax=1200/60:50/60:1800/60;              % Vector of Vmax-values
    lb(p.NT+2)=p.Vmax(i);                       % Keeping Vmax at given value
    ub(p.NT+2)=p.Vmax(i);                       % Keeping Vmax at given value
    [x,fval,exitflag]=fmincon(@fun,x0,[],[],[],[],lb,ub,@nlcon,options);
    F0el(i)=-x(p.NT+8)*60;                      % Cost with the exact local
method
end
load F0oVm                                      % Optimal cost for disturbance
in Vmax, saved earlier.
loss_el=((F0el-F0oVm)./abs(F0oVm))*100;         % Calculating the loss
Vm=1200:50:1800;                                % Vmax-values for the x-axis
% Making the plot of the loss versus Vmax
figure (4)
xx=Vm;
yy=spline(Vm,loss_el,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([1200 1800])
ylim([0 0.2])
set(gca,'XTick',1200:200:1800);
set(gca,'YTick',0:0.05:0.2);
xlabel('V_m_a_x [kmol/h]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Exact Local Method - Case 2','FontSize',12,'FontWeight','bold')
end
```

Script for finding the measurement combination matrices in the null space and exact local method from the Simulink model, script_np_el_T.m:

```
% Script for finding the measurement combination matrices (H) for the
% and exact local methods with the dynamic process model.
% The setpoints of c=Hy are also calcualted

close all
clc
warning off

%%
open('CSTRandColumnAConnected')              % Open the Simulink model
sim('CSTRandColumnAConnected')               % Run the Simulink model

% Storing the optimal values of the measurements
y0=[T.signals.values(end,1);T.signals.values(end,6);T.signals.values(end,10
);

T.signals.values(end,14);T.signals.values(end,18);T.signals.values(end,22);

ws.signals.values(end,10);ws.signals.values(end,11);ws.signals.values(end,2
);

ws.signals.values(end,3);ws.signals.values(end,5);ws.signals.values(end,12)
];

%% Null space method in Simulink
% Disturbing the process by a 1% change in F0
set_param('CSTRandColumnAConnected/F0_step','before','0','after','460/60*0.
01')
dF0=460/60*0.01;                             % Magnitude of input change

sim('CSTRandColumnAConnected')               % Run the Simulink model

% Storing the new values of the measurements
y_stepF0=y_n.signals.values(end,:)';

% Removing the disturbance in F0
set_param('CSTRandColumnAConnected/F0_step','before','0','after','460/60*0.
01*0')

% Calculating the optimal sensitivity matrix
F=(y_stepF0-y0)./dF0;                        % Optimal sensitivity matrix
Hnsb=null(F');
Hnsb=Hnsb';                                  % Measurement combination
matrix
Hns=Hnsb(3,:);                               % Choosing one of the rows
cs_ns=Hns*y0;                                % Set-point of c=H*y
% csns=335.8686                              % Calculated value of setpoint
cs=H*y0

%% Exact local method in Simulink
% Disturbing the process by a 1% change in L
set_param('CSTRandColumnAConnected/LT -
step','before','0','after','778/60*0.01')
sim('CSTRandColumnAConnected')               % Run the Simulink model

stepL=778/60*0.01;                           % Magnitude of input change
```

```
y_stepL=y_n.signals.values(end,:)';          % New measurement values

GyL=(y_stepL-y0)/stepL;                      % Finding dy/dL

% Removing the change in L
set_param('CSTRandColumnAConnected/LT - step','before','0','after','0')

% Disturbing the process by a 1% change in F
set_param('CSTRandColumnAConnected/Fs_step','before','0','after','958/60*0.
01')
sim('CSTRandColumnAConnected')               % Run the Simulink model

stepF=958/60*0.01;                           % Magnitude of input change
y_stepF=y_n.signals.values(end,:)';          % New measurement values
GyF=(y_stepF-y0)/stepF;                      % Finding dy/dF

% Removing the change in F
set_param('CSTRandColumnAConnected/Fs_step','before','0','after','0')

% Disturbing the process by a 1% change in V
set_param('CSTRandColumnAConnected/VB_step','before','0','after','1276.1/60
*0.01')

sim('CSTRandColumnAConnected')               % Run the Simulink model

stepV=1276.1/60*0.01;                        % Magnitude of input change
y_stepV=y_n.signals.values(end,:)';          % New measurement values
GyV=(y_stepV-y0)/stepV;                      % Finding dy/dF

% Removing the change in V
set_param('CSTRandColumnAConnected/VB_step','before','0','after','0')

% Computing the steady-state gain matrix Gy
Gy=[GyL GyF GyV];
% 2% measurement noise in flows
p=0.02;
% Magnitude of noise in temperatures was chosen as 0.01
% Cimputing the scaling matrixes for noise and disturbance (Wn and Wd)
Wn=diag([0.01; 0.01; 0.01; 0.01; 0.01; 0.01; p*y0(7:12)]);  %
F0=460/60;
% Magnitude of disturbance in F0 was chosen as 1%
dd=0.01*F0*ones(12,1);
Wd=diag(dd);

Y=F'*Wd*Wn;                                  % Y matrix
Helb=(inv(Y*Y')*Gy)';                        % Measurement combination matrix
% Helb=((Y.\(Y'))*Gy)';
HelS=Helb(1,:);                              % Choosing one of the rows
cs_el=HelS*y0;                               % Set-point of c=H*y
%cs_el=-1.0e4;                               % Calculated value of setpoint
cs=H*y0
```

Script for doing a step test in L and calculating the cost resulting from the null space method in the Simulink model, StepTest_NS.m:

```
clc
clear all
warning off

%% Null space method
open('CSTRandColumnAConnected')
load Hns
H=Hns';                     % Sending Hns to Simulink
csns=335.8686;              % Sending csns (setpoint of c) to Simulink

%% Step test in L for finding the tuning parameters for the L controller

% % Making a step in L
% set_param('CSTRandColumnAConnected/LT -
step','time','500','before','0','after','778/60*0.1')
% % Running the simulation
% sim('CSTRandColumnAConnected')
% % Removing the step in L
% set_param('CSTRandColumnAConnected/LT -
step','time','500','before','0','after','0')
%
% % Plotting the step response in c versus time
% figure (1)
% plot(c.time,c.signals.values,'b','LineWidth',2.5)
% % The step was made at time=500 min, and the time interval 500-510 min
% % was considered and assumed as an integating process.
% xlim([500 510])
% set(gca,'XTick',500:2:510);
% xlabel('Time [min]','FontSize',12,'FontWeight','bold')
% ylabel('c [-]','FontSize',12,'FontWeight','bold')
% title('Step Test in L - Null Space
Method','FontSize',12,'FontWeight','bold')
%
% % Calculating the SIMC tuning parameters for an integrating process
% dy=c.signals.values(723)-c.signals.values(526);
% dt=c.time(723)-c.time(526);
% du=778/60*0.1;
%
% kprime=dy/(dt*du)                          % kprime
% theta_L=0                                  % time delay in L
% tauc_L=5                                   % time constant for L
% tauI_L=4*(tauc_L+theta_L)                  % integral time
% Kc_L=(1/kprime)*(1/(tauc_L+theta_L))       % Controller gain for L
% P_L=Kc_L                                   % Tuning parameter P for
controlling L
% I_L=Kc_L/tauI_L                            % Tuning parameter I for
controlling L
% % P=-10.8519;                              % Calculated value of P
% % I=-0.5426;                               % Calculated value of I
%
% break

%% Controlling L and finding V for different disturbances in F0

% F0 is set to different values in the range F0opt +- 20% and the simulaton
% is runned, followed by storing the value of V when using the L controller
% to keep (csns-c) constant.
```

```
set_param('CSTRandColumnAConnected/LT -
step','time','500','before','0','after','0')
set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.2')
sim('CSTRandColumnAConnected')
V1=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.175')
sim('CSTRandColumnAConnected')
V2=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.15')
sim('CSTRandColumnAConnected')
V3=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.125')
sim('CSTRandColumnAConnected')
V4=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.10')
sim('CSTRandColumnAConnected')
V5=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.075')
sim('CSTRandColumnAConnected')
V6=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.05')
sim('CSTRandColumnAConnected')
V7=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.025')
sim('CSTRandColumnAConnected')
V8=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.0')
sim('CSTRandColumnAConnected')
V9=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.025')
sim('CSTRandColumnAConnected')
V10=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.05')
sim('CSTRandColumnAConnected')
V11=ws.signals.values(end,11)*60;
```

```
set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.075')
sim('CSTRandColumnAConnected')
V12=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.10')
sim('CSTRandColumnAConnected')
V13=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.125')
sim('CSTRandColumnAConnected')
V14=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.15')
sim('CSTRandColumnAConnected')
V15=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.175')
sim('CSTRandColumnAConnected')
V16=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.20')
sim('CSTRandColumnAConnected')
V17=ws.signals.values(end,11)*60;

VnsSim=[V1;V2;V3;V4;V5;V6;V7;V8;V9;V10;V11;V12;V13;V14;V15;V16;V17];

break

% Calculating optimal values of V for differnt F0 values in Simulink

% F0 is set to different values in the range F0opt +- 20% and the simulaton
% is runned, followed by storing the value of V.
set_param('CSTRandColumnAConnected/LT -
step','time','500','before','0','after','0')
set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.2')
sim('CSTRandColumnAConnected')
V1=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.175')
sim('CSTRandColumnAConnected')
V2=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.15')
sim('CSTRandColumnAConnected')
V3=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.125')
sim('CSTRandColumnAConnected')
V4=ws.signals.values(end,11)*60;
```

```
set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.10')
sim('CSTRandColumnAConnected')
V5=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.075')
sim('CSTRandColumnAConnected')
V6=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.05')
sim('CSTRandColumnAConnected')
V7=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.025')
sim('CSTRandColumnAConnected')
V8=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.0')
sim('CSTRandColumnAConnected')
V9=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.025')
sim('CSTRandColumnAConnected')
V10=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.05')
sim('CSTRandColumnAConnected')
V11=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.075')
sim('CSTRandColumnAConnected')
V12=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.10')
sim('CSTRandColumnAConnected')
V13=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.125')
sim('CSTRandColumnAConnected')
V14=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.15')
sim('CSTRandColumnAConnected')
V15=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.175')
sim('CSTRandColumnAConnected')
```

```
V16=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.20')
sim('CSTRandColumnAConnected')
V17=ws.signals.values(end,11)*60;

VnsSimOpt=[V1;V2;V3;V4;V5;V6;V7;V8;V9;V10;V11;V12;V13;V14;V15;V16;V17];
```

Script for doing a step test in L and calculating the cost resulting from the exact local method in the Simulink model, StepTest_EL.m:

```
clc
clear all
warning off

%% Null space method
open('CSTRandColumnAConnected')
load HelS
H=HelS';                    % Sending Hns to Simulink
csns=-1.0e4;                % Sending csns (setpoint of c) to Simulink

%% Step test in L for finding the tuning parameters for the L controller
% % Making a step in L
% set_param('CSTRandColumnAConnected/LT -
step','time','500','before','0','after','778/60*0.1')
% % Running the simulation
% sim('CSTRandColumnAConnected')
% % Removing the step in L
% set_param('CSTRandColumnAConnected/LT -
step','time','500','before','0','after','0')
%
% % Plotting the step response in c versus time
% figure (1)
% plot(c.time,c.signals.values,'b','LineWidth',2.5)
% % The step was made at time=500 min, and the time interval 500-510 min
% % was considered and assumed as an integating process.
% xlim([500 510])
% set(gca,'XTick',500:2:510);
% xlabel('Time [min]','FontSize',12,'FontWeight','bold')
% ylabel('c [-]','FontSize',12,'FontWeight','bold')
% title('Step Test in L - Exact Local
Method','FontSize',12,'FontWeight','bold')
%
% % Calculating the SIMC tuning parameters for an integrating process
% dy=c.signals.values(697)-c.signals.values(526);
% dt=c.time(697)-c.time(526);
% du=778/60*0.1;
%
% kprime=dy/(dt*du)                           % kprime
% theta_L=0                                   % time delay in L
% tauc_L=5                                    % time constant for L
% tauI_L=4*(tauc_L+theta_L)                   % integral time
% Kc_L=(1/kprime)*(1/(tauc_L+theta_L))        % Controller gain for L
% P_L=Kc_L                                    % Tuning parameter P for
controlling L
% I_L=Kc_L/tauI_L                             % Tuning parameter I for
controlling L
% % P=0.0336                                  % Calculated value of P
% % I=0.0017                                  % Calculated value of I
% break

%% Controlling L and finding V for different disturbances in F0

% F0 is set to different values in the range F0opt +- 20% and the simulaton
% is runned, followed by storing the value of V when using the L controller
% to keep (csns-c) constant.
```

```
set_param('CSTRandColumnAConnected/LT -
step','time','500','before','0','after','0')
set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.2')
sim('CSTRandColumnAConnected')
V1=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.175')
sim('CSTRandColumnAConnected')
V2=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.15')
sim('CSTRandColumnAConnected')
V3=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.125')
sim('CSTRandColumnAConnected')
V4=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.10')
sim('CSTRandColumnAConnected')
V5=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.075')
sim('CSTRandColumnAConnected')
V6=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.05')
sim('CSTRandColumnAConnected')
V7=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','-460/60*0.025')
sim('CSTRandColumnAConnected')
V8=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.0')
sim('CSTRandColumnAConnected')
V9=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.025')
sim('CSTRandColumnAConnected')
V10=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.05')
sim('CSTRandColumnAConnected')
V11=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.075')
```

```
sim('CSTRandColumnAConnected')
V12=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.10')
sim('CSTRandColumnAConnected')
V13=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.125')
sim('CSTRandColumnAConnected')
V14=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.15')
sim('CSTRandColumnAConnected')
V15=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.175')
sim('CSTRandColumnAConnected')
V16=ws.signals.values(end,11)*60;

set_param('CSTRandColumnAConnected/F0_step','time','200','before','0','afte
r','460/60*0.20')
sim('CSTRandColumnAConnected')
V17=ws.signals.values(end,11)*60;

VelSim=[V1;V2;V3;V4;V5;V6;V7;V8;V9;V10;V11;V12;V13;V14;V15;V16;V17];
```

Script for plotting the loss resulting from the null space method applied on the Simulink model, LossNSsimulink.m:

```
% Script for plotting the loss with the null space method
% resulting from simulations in Simulink

load VnsSim                                          % Loading the cost
values
load VnsSimOpt                                       % Loading the optimal
cost values
loss_ns=((VnsSim-VnsSimOpt)./VnsSimOpt)*100;         % Calculating the loss
loss_ns=abs(loss_ns);                                % Making the values
positive
F0=368:11.5:552;                                     % F0-values for the x-
axis

% Making the plot of the loss versus F0
figure (1)
xx=F0;
yy=spline(F0,loss_ns,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([400 550])
ylim([0 5])
set(gca,'XTick',400:30:550);
set(gca,'YTick',0:1:5);
xlabel('F_0 [kmol/h]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Null Space Method','FontSize',12,'FontWeight','bold')
```

Script for plotting the loss resulting from the exact local method applied on the Simulink model, LossELsimulink.m:

```
% Script for plotting the loss with the exact local method
% resulting from simulations in Simulink

load VelSim                                          % Loading the cost values
load VnsSimOpt                                       % Loading the optimal cost
values
loss_el=((VelSim-VnsSimOpt)./VnsSimOpt)*100;         % Calculating the loss
F0=368:11.5:552;                                     % F0-values for the x-axis

% Making the plot of the loss versus F0
figure (1)
xx=F0;
yy=spline(F0,loss_el,xx);
plot(xx,yy,'b','LineWidth',2.5)
xlim([400 550])
ylim([0 20])
set(gca,'XTick',400:30:550);
set(gca,'YTick',0:4:20);
xlabel('F_0 [kmol/h]','FontSize',12,'FontWeight','bold')
ylabel('Loss [%]','FontSize',12,'FontWeight','bold')
title('Exact Local Method','FontSize',12,'FontWeight','bold')
```