TKP4580 Specialization Project, Process Systems Engineering

Non-linear model predictive control for a reactor-distillation-recycle process

NMPC using the multiple shooting method

Michael A. Lindbak

Submission Date:	21.12.2020
Main Supervisor:	Johannes Jäschke, IKP
Co-Supervisor:	Zawadi Ntengua Mdoe, IKP

Department of chemical engineering

Norwegian University of Science and Technology

Abstract

This project considers a classic case of reactor-separator-recycle process in which a tightly temperature controlled CSTR is followed by a 22 stage distillation column. The distillate of the column is fed as recycle for the reactor, and the bottom flow is used as the product stream for the plant. The goal of the project was to create a dynamic set of equations for the case study and select a MPC discretization scheme in order to turn the resulting continuous control problem into a discretized non-linear problem. This controller was then set to be able to dynamically solve for the optimal inputs to the plant for a given objective function. The discretization scheme selected uses the principle of multiple shooting, where we will shoot for the inputs L, V and F, which refer to the column reflux, boil-up and feed respectively. The created NLP problem was then solved using the modelling framework CasADi and their NLP solver *IPOPT* in MATLAB. The objective of the NMPC controller was to be able to tune these inputs optimally to keep the bottom flow composition at a set value of 0.0105 $[mol_A\ mol^{-1}]$. Multiple controllers were created with varying time-increments Δt. These were compared for multiple disturbance cases and compared to see how their performance varied. In doing so, it was found that the smoother controller (controller with smaller Δt) had overall lower adjustment-period before the controller managed to reset the value for x_B. Additionally the controller showed a lower spike in the composition at the time of disturbance. Both controllers spent the same amount of time (5 timesteps) before reaching a steady operating state (where the inputs stopped changing) when initiated from the same initial values. A third variant of the controller was also tested without any of the disturbance cases imposed on the system. This controller would seek to attempt to adjust the bottom flow composition, x_B, as the set-point changed. While these set-point disturbances were relatively low in scale, their relative size were almost half and double that of the initial set-point. When testing the adaptability for the controller it was found that the controller performed exceptionally well, being able to achieve an error of 10^{-6} $[mol_A\ mol^{-1}]$ after only 2 time-steps.

Preface

This report is the result of a chemical process specialization project given to graduate students in the Norwegian University of Science and Technology (NTNU) in their penultimate semester in the graduate year. The project falls under the course ”TKP4580 Specialization Project, Chemical Engineering”. The work presented in this paper was performed in Autumn 2020 under the main supervision of Johannes Jäschke and co-supervision of Zawadi Ntengua Mdoe whom both work at the department of chemical engineering at NTNU.

While work on this project started somewhat slow due to a lack of knowledge on the procedure, it has been insightful nonetheless and proven to show how much work is actually needed to create a plant controller. I would like to personally thank my supervisor Zawadi for being as patient and helpful as he has been throughout the project time, as well as Johannes for being available in the case Zawadi was not. I have eternal gratitude towards all the aid they have provided me even during my most trying of times, and I hope that what I present in this paper can live up to their expectations.

As for my own expectations, I am at the time of writing a bit overwhelmed by the sheer length the report managed to accumulate which was way above what I had initially expected. So in that regard, I guess I have met my personal expectations. If the ones correcting this report agree with me on the other hand, is a separate matter - but I would like to remain optimistic.

List of Symbols

Variable	Description	Unit
F_0	The feed flow rate entering the plant	[kmol min^{-1}]
$F_{0,nom}$	The nominal feed-rate before any disturbance	[kmol min^{-1}]
z_{F0}	The composition of the feed entering the system	[mol_A mol^{-1}]
F	Column feed flow rate	[kmol min^{-1}]
z_F	Column feed composition	[mol_A mol^{-1}]
L (or L_T)	Reflux flow-rate	[kmol min^{-1}]
V (or V_B)	Vapor boil-up rate	[kmol min^{-1}]
D	Column Distillate (reflux) flow rate	[kmol min^{-1}]
B	Bottom product flow rate	[kmol min^{-1}]
Mr	Reactor hold-up	[$kmol$]
x_1 (or x_B)	The composition at the bottom of the column	[mol_A mol^{-1}]
x_2 - x_{21}	The composition at each of the column trays	[mol_A mol^{-1}]
x_{22} (or x_D)	The composition at the distillate	[mol_A mol^{-1}]
NT	The total amount of stages/trays in the column (22)	[-]
NF	The stage at which the feed is located in the column (13)	[-]
α	Relative volatility (assumed to be constant)	[-]
k_1	Reaction rate constant (assumed to be constant)	[min^{-1}]
	Controller parameters	
N_p	Prediction horizon for the optimization	[min]
T_{tot}	The total simulation time for the study	[min]
$x_{B,k}$	The value of the bottom composition at t=k	[mol_A mol^{-1}]
x_{ss}	Optimal concentration of x_B	[mol_A mol^{-1}]
R	Tuning parameter for x_B deviation	[-]
Q	Tuning parameter for inputs u_k	[-]
P	Tuning parameter for restricting changes to M_R	[-]
u_k	The input parameter at point $t = t_k$	[kmol min^{-1}]
$M_{R,set}$	Set-point value for reactor hold-up, M_R	[kmol]

List of Figures

Table of Contents

1 Introduction

Prediction is something that leans heavily towards the intuitive nature of how living beings interpret the world. For example, when catching a ball, you instinctively estimate the ball's position as well as its relative velocity towards you before moving your body accordingly to catch it. Much in the same manner do we construct modern-day controllers for chemical plants, in the manner that we use information on the plant's operating state to predict a trajectory for its states. This project is based on the principle of Model Predictive Control (MPC), where we use a mathematical model of a plant in order to predict and optimize a set of inputs for the plant. Feedback is then used to update a controller of the current state of the system before re-optimizing. Typically these types of optimizations aim to minimize a *objective function* by finding a optimal set of inputs that does not violate any constraint. The job of the controller is then to find the optimal inputs, based on the current state, that assures that a optimal operation may be achieved. [3]

A big problem for the more traditional control methods is that their extensively complicated designs and "temporary" solutions were found to not really scale well and be applicable to other systems despite shared similarities. Good examples of this are Isidori(1989) [4], Marino(1995) [5] and Krstic(1995) [6] *et al*. It was seen that while some of the later designs could give very good approximations for the plants' input parameters, they were not able to systematically handle the imposed constraints in a very good manner. Since MPC's implementation in 1996 by some process industries (prime examples are Qin and Badgwell), [7] MPC quickly gained traction and set a new precedent for process control. The method has been studied extensively both in the industry and academia, and it is this control method that will be implemented in this project.

To show how non-linear MPC (NMPC) can be implemented, we use a case study that is a reactor-column-recycle process. The system in question will be further defined under section 2, but is mainly consisting of a continuously stirred tank reactor (CSTR), followed by a distillation column for product separation. Although a fairly simple system, we will see that with some disturbances to the input feed, the created NMPC controller needs to be able to aptly predict the optimal inputs of our system in order to minimize any loss. The goal for this report is to be able to create a controller that is able to properly manage the plant, depending on the asked objective. Multiple cases for disturbance and objective will be tested to verify that the controller is fully able to handle the system regardless of the scenario imposed on it.

In order to achieve this however, the system first needs to be modelled using a set of model equations so we have some notion of the time dynamics in place. These models will then be verified by a simple test of seeing if the system goes towards steady-state from a set of initial parameters (preferably not too far off the steady-state parameters), before a controller can be created. When we create a NMPC controller for the system we follow the procedure further elaborated under chapter 4. In short, we will use the multiple shooting to convert our continuous optimal control problem (OCP) into a non-linear problem (NLP), which then can be solved computationally.

1.1 Model Predictive Control

Model predictive control is the field of applying a receding predictive horizon to compute a set of manipulative variables to create a optimal trajectory of a given set of states. [8] When doing this one can create a consistent

controller that will always give close (depending on the permitted error) inputs for minimization of a objective term, given a set of constraints. This is typically done by solving a continuous OCP given a set of initial conditions as well as a set of constraints that must be uphold. The controller will create a set of inputs for a time instant t_i, which will be used for the following time-period Δt, to aquire the states x_{i+1}. The OCP can be typically be summarized by equations 1.1 shown below. [9]

$$\begin{aligned} \min_{x,u} \int_0^T l\Big(x(t),u(t)\Big)dt \\ x(0) = x_0 \\ \dot{x} = f\big(x(t),u(t)\big) \\ h\big(x(t),u(t)\big) \leq 0 \\ g\big(x(t),u(t)\big) = 0 \end{aligned} \tag{1.1}$$

From the equations 1.1 shown above, the objective function is given by $l\big(x(t),u(t)\big)$, where *x(t)* and *u(t)* are the value of our states and inputs respectively, at time t. This is the function which we wish to minimize and is thus the core of our OCP. The initial conditions of the system are given by a vector, x_0, which contains all the initial states for the system. It is assumed that the time dynamics of the states of the system is given by the function $f\big(x(t),u(t)\big)$, and that the constraints of the system can be systematically divided into the equality constraints $h\big(x(t),u(t)\big)$ and the inequality constraints $g\big(x(t),u(t)\big)$.

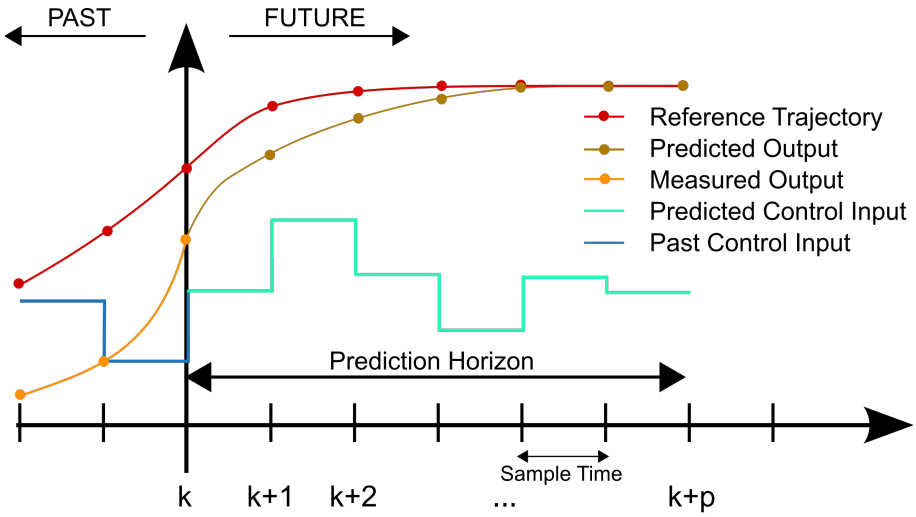

Figure 1.1: Figure showing the principle behind model predictive control. It shows the present being at time t=k, where the amount of steps, p, predicted into the future is the prediction horizon for the model. The figure illustrated how we use discretization in MPC to try and align the reference trajectory with the predicted output of the model, using the predicted control output for the prediction horizon. [1]

As can be seen in figure 1.1 the main idea behind MPC is to utilize the knowledge of the states x_k at *x(t=k)* to be able to predict an optimal sequence of inputs $\big(u_k, u_{k+1}, ..., u_{k+p-1}\big)$ at $t = k, k+1 ... k+p-1$ which minimizes the objective function, $l\big(x(k),u(k)\big)$, without violating any of the constraints for the process. To make sure the plant does not drift away, only the first pair of inputs, u_k, from the sequence is implemented in the plant. Then, at the following timestep k+1, a state feedback x_{k+1} is used as the new x_k and optimization is re-initialized for the

following prediction horizon. This is then repeated for the following timesteps, t_i, for as long as the simulation run or plant is operated. The prediction horizon, N_p, represents the amount of times the optimizer predicts into the future, which gives the overall model better accuracy in its predictions at a cost of longer computation-time. Through the use of discretization schemes, such as multiple shooting, we can go from a continuous OLP to a discretized finite-dimensional NLP. therefore go from a OLP problem to a NLP.

1.2 Multiple Shooting Method

When creating a MPC controller there are numerous approaches one can take to create controllers from an OLP. Methods like these include the discretization tools single shooting, collocation points, and multiple shooting. These algorithms are based on the nature of the control problems and their purpose is to be able to transform infinite-dimensional OCPs into a finite-dimensional NLP.[10] This is so that the problems readily can be solved using an algorithmic approach. An example of this is CasADi's IPOPT that uses a intensive point- method which uses a estimated guess which is incrementally improved per iteration the algorithm is executed. This method is exceptionally computationally time-saving as the solvers never seek to iterate any optimization problem into its analytic solution, but rather takes iterative steps towards a point with an acceptable difference.[11]

The multiple shooting method is a slightly more comprehensive method than that of single shooting, as it requires additional constraints and decision variables, but is better suited for problems where non-linearity can become an issue. Although the method is based on a lot of the same principles as single shooting, they vary slightly as single shooting cannot provide adequate continuity where non-linearity is concerned. To begin this method we first need to have a initial value, x_0, for our problem at time of initiation, and we need dynamic equations for all our states. From this we can create temporary (calculated) state values at the different time-steps, s_i using the time dynamics equation $f(x_0,u_0)$. Our temporary input-variable here, is denoted by q_i and is the variable used to calculate the temporary states s_i.[12]

$$u(t_i) = q_i, \qquad x(t_i) = s_i, \qquad \dot{x}_i(t) = f\big(x(t), q_i\big), \qquad t \in [t_i, t_i + N] \tag{1.2}$$

From this, we obtain trajectory pieces for the states $x_1, x_2, \ldots, x_N$ (now $s_1, s_2, \ldots, s_N$) in the problem using a decoupled ODE solution (f), along with the decoupled cost function (L) for the time-step t_i.

$$L_i(s_i, q_i) = \int_{t_i}^{t_i+1} l\Big(x_i\big(t_i; s_i, q_i\big), q_i\Big) dt \tag{1.3}$$

Since we want to retain continuity in our optimization, we enforce a equality constraint that the state $s_i = x(t_i; s_{i-1}, q_{i-1})$ which can be visualized as a sort of stitching together of the value of the states at the border between each timestep. This is visualized below in figure 1.2, where we can see the discrete change in the inputs create gaps for the states that need to be connected for continuity.[13]

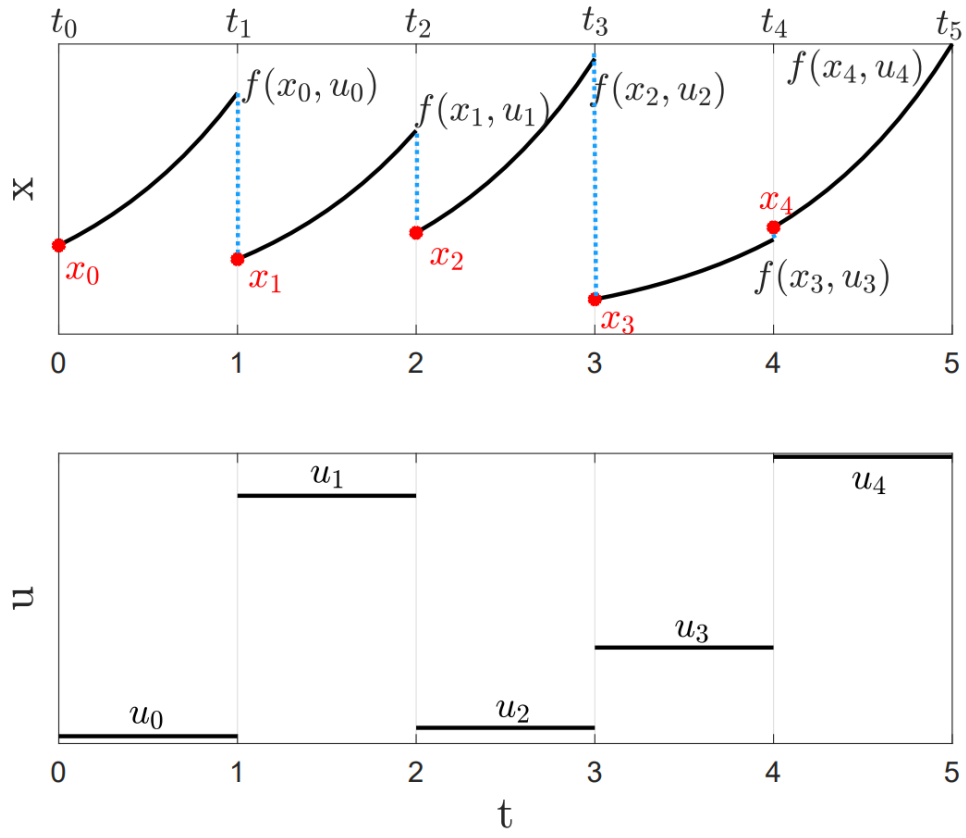

Figure 1.2: Figure illustrating the resulting gaps that can arise in the continuous states, $x_1,...,x_4$, when using discrete inputs $u_0,...,u_4$. The graph shows the equality constraint we need to enforce using multiple shooting to make sure our states are continuous. [2]

This is then repeated for each timestep, t_i, in the prediction horizon, N_p, to find states the states $(s_0,s_1,...,s_N)$ for the respective inputs $(q_1,q_2,...,q_{N-1})$. Combining these two we get a decision vector, which contains all the parameters that will be optimized in the NLP. In essence, what has been done is turning a continuous OCP into a discrete NLP. The CasADi NLP solver IPOPT can then be used to solve the problem using the decision variables, based on the following equations;

$$\min_{x,u} \sum_{i=0}^{n-1} L(s_i,q_i) \quad , \quad i=0,1,...,n-1$$

s.t.

$$g(s_i,q_i) = \begin{bmatrix} s_0 - x(0) \\ f(s_0,q_0) - s_1 \\ f(s_1,q_1) - s_2 \\ \vdots \\ f(s_{N-1},q_{N-1}) - s_N \end{bmatrix} = 0 \quad , \quad h(s_i,q_i) = \begin{bmatrix} h(s_0,q_0) \\ \vdots \\ h(s_{N-1},q_{N-1}) \\ h(s_N) \end{bmatrix} \leq 0 \tag{1.4}$$

1.3 Structure of Report

As the introductory material of the report is finished, the following chapter 2 will proceed with presenting the chemical plant which will represent our system for the case study. Following the case study a breakdown will

commence in chapter 3. This chapter will open with some key assumptions that are made for model simplification, before we start with the mathematical derivation of the state equations that are to be used for the dynamic optimization. The state equations will help us create a OCP, which we in chapter 4 will discretize into a NLP using the multiple shooting method. This chapter will prevent two disturbance cases which we will test our controller on as well as two different discretization intervals, that will be compared against each other. Additionally a separate non-disturbance case will be tested where we directly change the set-points in the controller to verify its adaptability.

The results from the simulations are presented in chapter 5, with time-graphs for all the disturbance scenarios showing the calculated optimal inputs as well as the resulting states. These are briefly discussed before a more elaborate discussion is performed in chapter 6. Finally, the entire report is summarized and wrapped with a conclusion in chapter 7.

2 Case Study

Chemical processes are usually some sort of reaction section, in which all of the reactants in the plant will turn to products and some bi-products, followed by a separation section. In this latter part, there are a variety of separation techniques that are utilized to isolate the desired product from all the (potentially) unwanted bi-products. One of the more common techniques used for product separation is distillation, in which the reactant stream from the ”reaction section” of the plant, is fed into a continuously boiling tower. This tower consists of multiple stages (or trays), which are continuously boiled for separation. The boiled steam has a different composition than the liquid it boiled from and is gathered in the tray above its original boiling-stage, before it is re-boiled again. The resulting outcome is a top and bottom flow with very pure streams for each of the respective chemicals involved. [14]

In this project, we investigate the optimal control of a simple CSTR connected to a distillation column. The system also has the distillate flow of the column feed back into the CSTR for recycle, while the bottom flow of the column is used as a product stream for the plant. An illustration of this is attached below in figure 2.1.

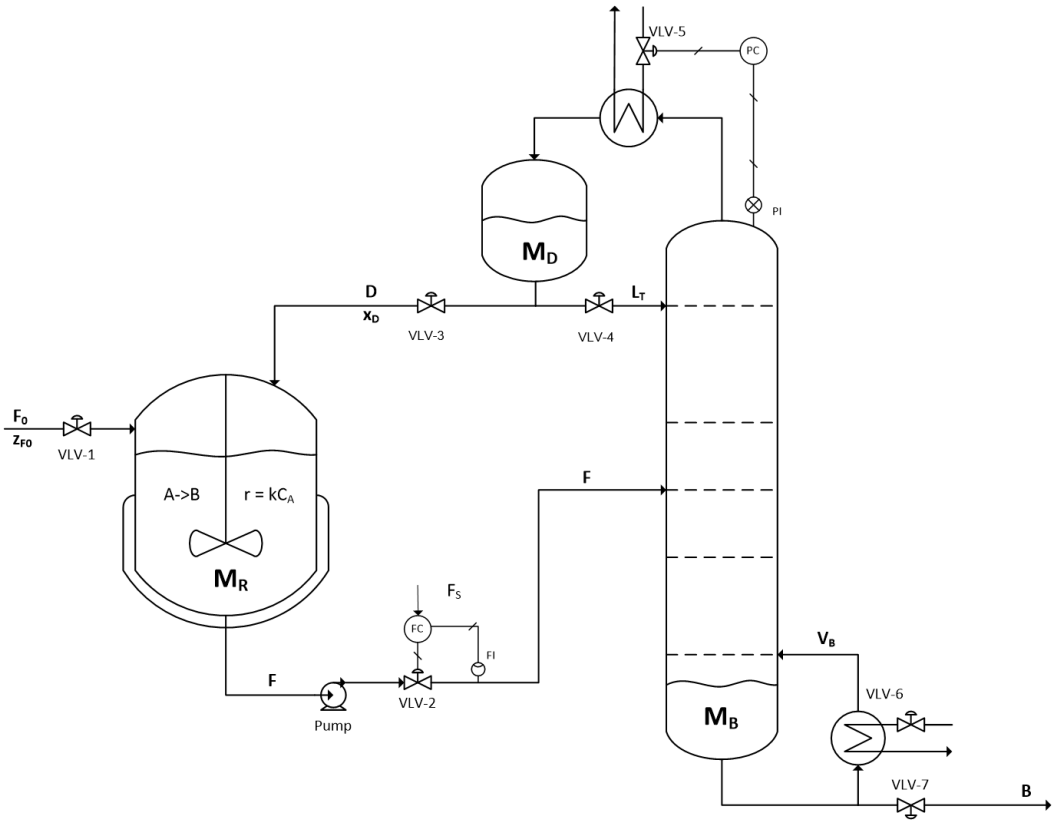

Figure 2.1: Process diagram of the case study used for this report. The diagram shows a CSTR connected to a distillation column, where the distillate is split into a reflux and a recycle to the CSTR. The bottom product of the column is extracted as the product stream.

2.1 Model assumptions and simplification

Typically chemical processes tend to be multi-stage complicated processes with several flow changes downstream, this system has been simplified to only accompany a two-component system with compound A and B. In this

system, the component A reacts in a simple first-order reaction to component B, shown below in equation 2.1.

$$A \rightarrow B \tag{2.1}$$

In order to create a simple model for the system it is simplified further by the following assumptions;

(i) Constant Pressure

(ii) Constant relative volatility, α

(iii) Total Condenser

(iv) Equilibrium on all stages (trays) in the column

(v) No vapor holdup

(vi) Constant molar flows

(vii) Constant reactor temperature

(viii) 1^{st} order reaction kinetics for the reaction

(ix) Column feed is assumed to have a liquid fraction = 1

While some of these assumptions might seem naive when performing chemical engineering to create a controller, these types of models prove helpful as they can be readily be expanded upon if more extensive models are required. For example, the assumption of strict temperature control on the CSTR makes it possible to assume that the reaction rate constant, k, is constant. If this was not the case however, one would simply have to create a further more detailed function for the rate in the dynamics part of the controller, and it would be able to adapt.

The constraints applied to the system was that the bottom composition, x_B could not increase above 0.0105 [mol_A mol^{-1}] and, that the Reactor holdup, M_R, could not go above 2800 [kmol]. In order to assure stability for the system however, these were implemented as soft constraints. A soft constraint is one where we allow the optimizer to cross the constraint, but at a great ”cost” in the objective term. When a constraint like this is left soft, the optimizer will therefore try to avoid crossing the constraint even though sometimes required for functional operation. The purpose of introducing them as soft constraints is that we can study the system using these constraints first, before going back to a real case with hard constraints with introduced back-off. Back-off is a term used for when a process is operated below optimal operation to not violate any hard constraint for a possible impending disturbance. A final constraint that was implemented as hard was that the input-parameter, V, should not go above 25 as this would cause flooding in the tower.

3 Modelling the system

The system consists of two main stages, namely the CSTR reactor and the distillation column. As these units are mostly independent, the models derived from them can also be sectioned. We will first proceed with a derivation for the equations used on the CSTR before considering the column equations. Finally at the end of this chapter (see chapter 3.3) a brief summary of the states, inputs and parameters is included.

3.1 Modelling the CSTR

As aforementioned in chapter 2, we assume that the hold-up in the reactor is not constant. Based on this assumption, we can derive the following equation 3.1 from a overall mass balance of the reactor. From this equation we get the accumulation in the tank, or the reactor hold-up. Here F_0 is a constant and D and F are the column reflux and tank product stream, respectively.

$$\frac{dM_R}{dt} = F_0 + D - F \tag{3.1}$$

Onto the component balance for the CSTR, we need to derive an expression which can express the resulting composition of the product stream, z_F, accurately given initial conditions. The derivation of this is shown below in equation 3.2 using a overall component balance over the reactor. Here the component accumulation term M_{RA} is defined as $z_F * M_R$.

$$\begin{aligned}
\frac{dM_{RA}}{dt} = \frac{dz_F M_R}{dt} &= z_{F0}F_0 + x_D D - F z_F - kM_{RA} \\
z_F \frac{dM_R}{dt} + M_R \frac{dz_F}{dt} &= z_{F0}F_0 + x_D D - F z_F - kM_{RA} \\
M_R \frac{dz_F}{dt} &= z_{F0}F_0 + x_D D - F z_F - kM_{RA} - z_F \frac{dM_R}{dt} \\
\frac{dz_F}{dt} = \frac{1}{M_R} * (z_{F0}F_0 + x_D D - F z_F - k * z_F M_R - z_F(F_0 + D - F)) & \\
\frac{dz_F}{dt} = \frac{F_0}{M_R}(z_{F0} - z_F) + \frac{D}{M_R}(x_D - z_F) - kz_F &
\end{aligned} \tag{3.2}$$

Another important mention for this step is the assumption made about the reaction kinetics inside the reactor. This was, as aforementioned, assumed to be a first-order reaction with a constant reaction constant k $[min^{-1}]$ based on the assumption that the CSTR is tightly temperature controlled. The reaction is used above as a consumption term for the composition inside the reactor which is represented by $-kM_{RA}$.

3.2 Modelling the column

For the distillation column we can derive an overall mass balance which nets us with equation 3.3.

$$F = D + B \tag{3.3}$$

From the assumptions made under chapter 2, we can set the following steady-state mass balances for the trays in the column shown in equation 3.4. Equation 3.5 shows the exception, which is the feed tray. These equations are for steady-state calculations as the change in tray-accumulation is assumed to be zero, $\frac{dM_x}{dt} = 0$.

$$Vy_{i-1} + Lx_{i+1} = Vy_i + L_i x_i \tag{3.4}$$

$$Fz_F + vy_{i-1} + Lx_{i+1} = Vy_i + Lx_i \tag{3.5}$$

Here the vapor-liquid equilibrium is described through equation 3.6, where the relative volatility α is (as mentioned earlier) assumed to be constant.

$$y_{i,A} = \frac{\alpha_{AB} x_{i,A}}{1 + (\alpha_{AB} - 1)x_{i,A}} \tag{3.6}$$

The condenser and reboiler here are assumed to be controlled and have a constant hold-up. This is done by sacrificing two degrees of freedom, which are the vents VLV-6 and VLV-4 (see fig 2.1). This simplifies their mass balances, and gives explicit expressions for the bottom flow, B, and distillate flow, D. Based on this we can do a overall mass balance for the condenser, as well as the lower half of the tower to get expressions for B and D, which is shown below in equations 3.7 - 3.8.

$$B = F + L - V \tag{3.7}$$

$$D = V - L \tag{3.8}$$

By modifying these steady-state equations, we can then derive component equations for the column trays for when the system is *not* in steady-state by doing component balances for each of the trays.

3.2.1 Boiler

Starting with the reboiler, which we assume to be a equilibrium stage from our assumption above, we get time dynamics for x_B shown below in equation 3.9. Here it is also assumed that the hold-up in each tray is constant. This implies that for all the trays it can be represented by multiplying with a constant $K = \frac{1}{M_i}[kmol^{-1}]$, which is set to 1 for the sake of simplicity.

$$\frac{dM_{x1}}{dt} = M_1 \frac{dx_1}{dt} = (L+F)x_2 - Vy_1 - Bx_1 \tag{3.9}$$

3.2.2 Stripping-Section

Proceeding for the stripping trays, time dynamics equations can be derived for x_2 - x_{NF-1} which is shown below in equation 3.10. NF is here denoted as the stage at which the feed enters the column.

$$\frac{dM_{x,i}}{dt} = M_{strip}\frac{dx_i}{dt} = (L+F)x_{i+1} - (L+F)x_i + Vy_{i-1} - Vy_i \qquad \forall \quad i = 2,3,...,NF-1 \tag{3.10}$$

3.2.3 Feed-Tray

Proceeding the stripping-section, we have the feed-stray. The equations for this stage can be derived as shown in equation 3.11.

$$\frac{dM_{x,NF}}{dt} = M_{NF}\frac{dx_{NF}}{dt} = L * x_{NF+1} - (L+F)x_{NF} + Vy_{NF-1} - Vy_{NF} + Fz_F \tag{3.11}$$

3.2.4 Enrichment-Section

Above the feed tray we have the enrichment section of the column. These trays form time-dynamic equations for the components x_NF+1 - x_21 for stages above the feed-tray, but below the condenser. The state-dynamics can be calculated through equation 3.12.

$$\frac{dM_{x,i}}{dt} = M_{enrich}\frac{dx_i}{dt} = Lx_{i+1} - Lx_i + Vy_{i-1} - Vy_i \qquad \forall \quad i = NF+1, NF+2, ..., NT-1 \tag{3.12}$$

3.2.5 Condenser

And finally for the condenser, we can derive the following mass balance for $x_D = x_{22}$ shown in equation 3.13.

$$\frac{dM_{x,NT}}{dt} = M_D\frac{dx_D}{dt} = Vy_{NT-1} - Lx_{NT} - Dx_{NT} \tag{3.13}$$

3.3 Summarizing the model

From the derivations in chapter 3 and the plant shown in chapter 2 the system can be summarized in the following manner; As the column consists of 22 trays, there is one composition state for each of the tray which includes the composition for the bottom product flow as well as the distillate (reflux and recycle). The remaining components in the plant is therefore z_{F0}, which is the composition of the feedstock (set to be constant) and z_F, which is the composition of the column feed, shown in equation 3.2.

The flows of the system are the bottom product flow (B), the distillate flow (D), column reflux (L), the feedstock (F_0) and the column feed (F). And finally the hold-ups in the system are the ones for the reactor (M_R), condenser (M_D) and boiler (M_B). In total this sums to 33 variables, and as aforementioned of these M_D, M_B, z_{F0} is assumed to be constant. The feedstock of the plant, F_0, is set to be the disturbance. Remaining, then, is a total of 29 variables,

to which a total of 26 state equations have been derived. The remaining 3 variables will therefore be used as inputs for the NMPC. These inputs are the column reflux L, the column boil-up V and the column feed F.

$$x = \begin{bmatrix} x_B \\ x_2 \\ x_3 \\ \vdots \\ x_{21} \\ x_D \\ D \\ B \\ z_F \\ M_R \end{bmatrix} \qquad \dot{x} = \begin{bmatrix} \text{Eq. 3.9} \\ \text{Eq. 3.10 (i=2)} \\ \text{Eq. 3.10 (i=3)} \\ \vdots \\ \text{Eq. 3.12 (i=21)} \\ \text{Eq. 3.13} \\ \text{Eq. 3.8} \\ \text{Eq. 3.7} \\ \text{Eq. 3.2} \\ \text{Eq. 3.1} \end{bmatrix} \qquad u = \begin{bmatrix} L \\ V \\ F \end{bmatrix} \tag{3.14}$$

$$\min_{x,u} \int_0^{T_{Tot}} l\Big(x(t), u(t)\Big)$$

$$\text{s.t.}$$

$$h\big(x(t), u(t)\big) \leq 0$$

$$g\big(x(t), u(t)\big) = 0$$

4 Dynamic Optimization

In order to be able to determine a proper cost function for the system, some steady-state calculations had to be performed. To perform this, all the state equations were gathered into a vector, using CasADi's symbolic variables. This vector was then fed to CasADi's NLP solver IPOPT, where the objective function was given as the symbolic variable V, which represented the liquid boil-up in the column. This was repeated for all the disturbance cases for the feedstock, F_0. The solver quickly found solutions in which both the constraint for x_B and M_R were active for all the impending disturbances. To test the robustness of the controller it was decided to perform NMPC set-point tracking for these disturbances. The goal was to see how one could effectively control the system to the desired set-point when various disturbances were applied to the feedstock. The goal of the NMPC set-point tracker is to keep the deviation of x_B from its constraint value as low as possible. The optimizer was created to be able to convert our OCP into a NLP which can be solved. This was done using the multiple shooting method (see chapter 1.2), where the total simulation time, T_{tot}, was set to be 60 minutes and the prediction horizon, N_P, was set to 10 minutes. All parameters used for the simulation, as well as their values, is summarized at the end of the chapter in table 4.1. Additionally a short run-down of the pseudo-algorithm is summarized above the table.

4.1 Disturbance case I

As the plant already had a given feed of $F_0 = 460$ [kmol hr^{-1}], nominal changes to this initial value was tested to test the controller for various disturbances. The first of these cases uses the greatest change of the two as the plant feed, F_0, was disturbed by -20% at a given time-point, and then later on 120% of the initial feed at a later point in time. The disturbance is illustrated below in figure 4.1.

For the second jump this then imposed a total disturbance impact of $40\% * F_0$. Which was hypothesized to swing the calculated bottom product composition, x_B, the most.

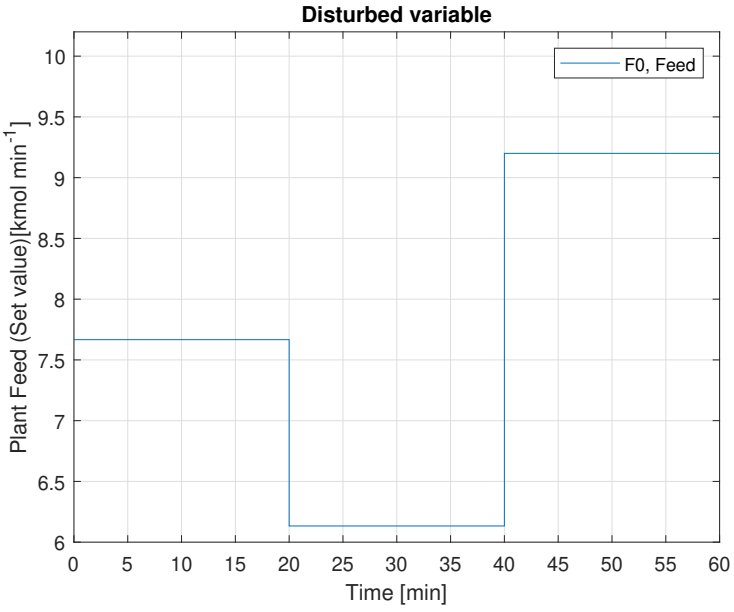

Figure 4.1: Time-graph showing the first disturbance case for this report. The disturbance is a negative-positive one that occurs at t=20 and t=40 respectively. At these points the feedstock, F_0, is changed to -20% and +20% of its initial value, respectively. This feed is the feed that is entering the plant at the left of the process diagram in figure 2.1.

4.2 Disturbance case II

The latter disturbance case would start with a milder disturbance of +10% at the first time-point, before later on being disturbed to 120% of the nominal value, in the later time-point. This is illustrated below in figure 4.2. Thus both cases end up at the same feed-stock, but through different means. Comparisons of these two could then be evaluated to see if the controllers end up at the same final values for the two different cases.

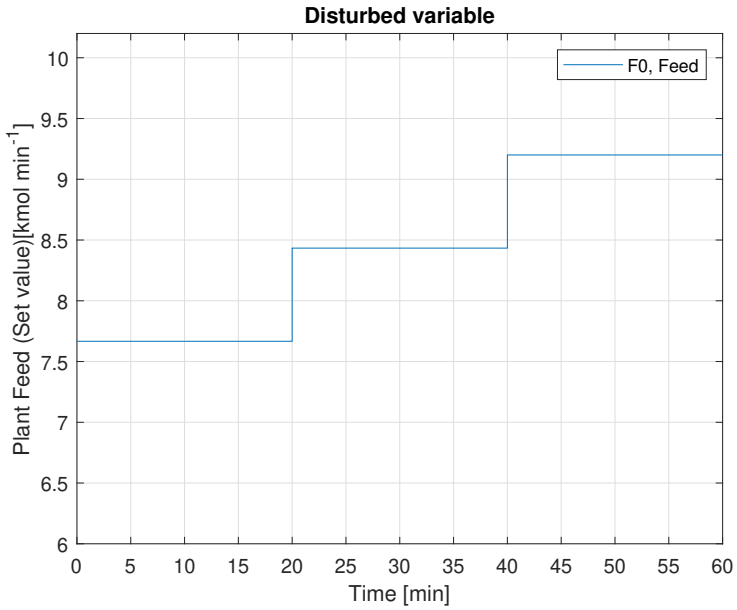

Figure 4.2: Time-graph showing the second disturbance case that is used for controller evaluation. The disturbance is a positive-positive change that again occurs at t=20 and t=40 respectively. The former perturbation is 110% of the initial feed, and the latter one is 120% of the initial feed. The feed in question, is the one entering the CSTR, which is the feedstock of the plant.

The simulation time for the plant was set to be a full hour with $t \in [0, 60]$ and the disturbances would occur at $t = 20[min]$, and $t = 40[min]$ respectively. When performing set-point tracking on the system, the objective function was set to minimize the deviation of x_B from its nominal value of 0.0105 for both case A and B.

In order to test the refinement of the optimizer, the objective case defined below (equation 4.1) was tested for two separate cases. The rough controller (case A) was using a time-increment of $\Delta t = 1[min]$ and the smooth controller (case B) was using a time-increment of $\Delta t = 0.5[min]$. This was done to see how the controllers varied and compare the results produced by the two. Note that for the latter case (case C), only the rough controller was tested.

4.3 Case A - Disturbance rejection

The main objective of the set-point tracking is to keep the value of x_B as close to its nominal value of 0.0105 as possible. This was implemented by having a term $R * (x_B - x_{ss})^2$ in the objective term, where x_{ss} was defined as the optimal concentration of x_B. The composition of the bottom product stream is x_B, and R is a scaling constant to get the objective priorities as desired.

Additionally the controller was introduced to a small cost for the deviation of input parameters *L, V, F* from their previous value in the last timestep, to minimize fluctuations in the inputs. This was implemented by the term $(u_k - u_{k-1})Q(u_k - u_{k-1})$ in the objective function, where Q is a tuning-parameter simply used so that the optimizer

will prioritize getting the bottom composition to a steady state over not changing the input parameters. u_k is the values of the inputs at $t = k$.

Finally a term $\left(M_R - M_{R,set}\right)P\left(M_R - M_{R,set}\right)$ was added. Here M_R is the value of the reactor hold-up and $M_{R,set}$ is a set value for the controller. P is a parameter used for weights in the objective function. This term was added in hindsight as the reactor would only fill and empty depending on the feedstock. In order to speed up the simulations this term was added to force a pseudo steady-state on the system. This because over prolonged simulation-times, the system would eventually go towards a value where $\frac{dM_R}{dt} = 0$. The main goal of the optimizer is still to keep x_B at its nominal value, and the weights R, P and Q were set to assure this. The resulting objective term for each timestep $t = k$ in this case is shown below in equation 4.1.

$$L\left(x_k, u_k\right) = R * \left(x_{B,k} - x_{ss}\right)^2 + \left(u_k - u_{k-1}\right)Q\left(u_k - u_{k-1}\right) + P\left(M_R - M_{R,set}\right)^2 \tag{4.1}$$

4.4 Case B - Smoother disturbance rejection

For the second case, the controller used the same objective term as mentioned for case A (see equation 4.1). This time however, the simulations would be run with $\Delta t = 0.5$ [min]. In order to retain the same prediction horizon, the optimizer would therefore have to run 20 time-steps to retain a 10 minute horizon. The goal of this case was to compare the resulting inputs for the two controllers to check for any discrepancies and try to further evaluate these.

4.5 Case C - Set-point change tracking

The final case would do a set-point adjustment test to evaluate the controller's ability to adapt to various set-point values for x_B. This case is an exception to the former two, as it will be performed without any disturbances imposed on the system. This case only aims to see how well the controller managed to adjust the parameters of the inputs to best be able to track the varying demand in x_B. The objective function is therefore the same as the previous two cases (see equation 4.1), as we still want to see the the inputs go towards a final value, and not fluctuate unnecessary. Since this case changes the desired nominal value of x_B, the value of x_{ss} was not a constant value of 0.0105. Rather the value of x_ss is as shown below in the intervals below.

$$x_{ss} = \begin{cases} 0.0105 & \text{for} \quad t \in [0, 20] \\ 0.006 & \text{for} \quad t \in [20, 40] \\ 0.02 & \text{for} \quad t \in [40, 60] \end{cases}$$

4.6 Algorithm summary

The entire code is attached and can be seen in Appendix A, but in short the algorithm can be summed up in the following segment; First, a integration step was performed of all the states using time equations for the systems from t_{i-1} to t_i. Here, all the states were fed as symbolic variables to the integrator, the system inputs are fed as symbolic parameters, and finally the initial values for the problem, seen in equation 4.2. The integrator was then set up as a simple ODE system, and the CasADi integrator *IDAS* was used. The calculated states were then fed into a optimizer, which utilized multiple shooting method (see chapter 1.2), to find the optimal inputs for the

following timestep. The optimizer had a prediction horizon, N_p, of 10 minutes where the proper constraints were set. This entails the constraints for multiple-shooting as well as realistic constraints for the plant (see chapter 2, no negative mass etc.). The prediction variables thus consisted of the states for the prediction horizon as well as the inputs, which were solved as a NLP problem using CasADi's *IPOPT* solver. Doing this, the first optimal inputs for timestep t_i were extracted. These values were then stored and fed to the integrator for the following time-step. And the process was repeated throughout the total simulation time. All values were stored in separate arrays and plotted. The resulting graphs are attached in chapter 5 and all parameters used are summarized in table 4.1.

Table 4.1: Table containing the most essential parameters that were used in the optimization.

Variable	Description	Value	
N_p	Prediction horizon for the optimization	10	[min]
T_{tot}	The total simulation time for the study	60	[min]
$x_{B,k}$	The value of the bottom composition at t=k	-	$[mol_A\ mol^{-1}]$
x_{ss}	Optimal concentration of x_B	0.0105	$[mol_A\ mol^{-1}]$
R	Tuning parameter for x_B deviation	500	[-]
Q	Tuning parameter for inputs u_k	10^{-5}	[-]
P	Tuning parameter for restricting changes to M_R	10^{-5}	[-]
u_k	The input parameter at point $t = t_k$	-	[kmol min^{-1}]
$M_{R,set}$	Set-point value for reactor hold-up, M_R	2800	[kmol]

All the different cases were initiated from the same set of initial states shown below in equation 4.2.

$$x_0 = \begin{bmatrix} x_1(=x_B) = 0.5 \\ x_2 = 0.5 \\ \vdots \\ x_{21} = 0.5 \\ x_{22}(=x_D) = 0.5 \\ z_F = 0.3 \\ M_R = 2800 \end{bmatrix} \qquad u_0 = \begin{bmatrix} L = 15 \\ V = 20 \\ F = 5 \end{bmatrix} \tag{4.2}$$

5 Results

The study of the controller is divided into two different disturbance cases, as well as two different objective cases. This results in a total of four results that have been simulated and categorized according to their matching set of disturbances and objectives. As a comparison-value the previously calculated SS-values for the NLP is shown as a reference in the graphs for case A. The values from these calculations were performed for all feed-disturbances and are shown below in table 5.1. Additionally is the third case controller, which had no disturbance imposed on it.

Table 5.1: Table showing the calculated steady-state values for the NLP. The objective function used was to minimize V, and all state equations were set to go to zero. The values were calculated for all feed disturbance-cases that were imposed on the system. Here $F_{0,nom}$ refers to the nominal value of F_0, which is 460 kmol h^-1.

Inputs	Disturbance, $F_0 =$			
[kmol min^{-1}]	0.8*$F_{0,nom}$	$F_{0,nom}$	1.1* $F_{0,nom}$	1.2 *$F_{0,nom}$
L	8.39	11.45	13.14	14.66
V	11.87	17.16	20.30	23.84
F	9.66	13.38	15.59	18.08

5.1 Case A-I

For the case with negative-positive perturbation (case I), with the controller set to minimize input change the results obtained (case A) are illustrated in figure 5.1.

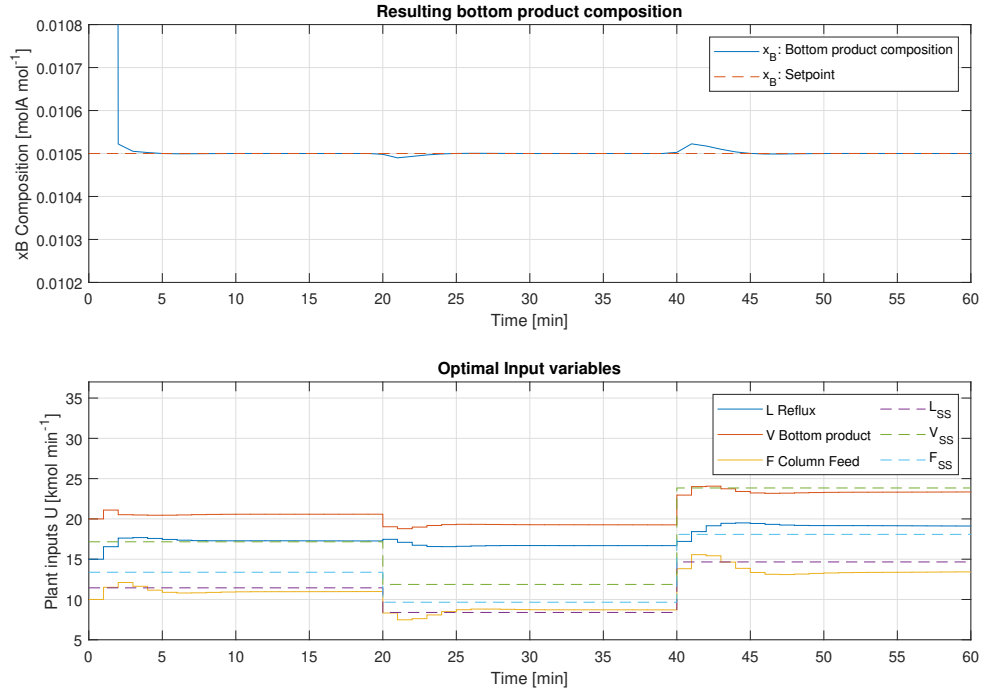

Figure 5.1: Time-graph showing the calculated optimal inputs from the controller of case A under the imposed disturbances from case I. The graph also shows the calculated states for the bottom product composition, x_B, throughout the simulation time. The graph also shows the previously calculated SS-values found when the column boil-up V was set as the objective function.

From this graph, and the following, we can see that the controller spends about 7 minutes to go from initial values to steady-state for the inputs. The controller manages to settle x_B to its set-point after 4 timesteps however, incurring an error of 10^{-6} for the first disturbance. For the second perturbation, the controller spends 6 timesteps but is able to converge to an error of 10^{-7}. After the steady-state is reached, the inputs of the controller stop changing and we would expect these to land on the previously calculated SS-values. The column feed, F, actually lands fairly close, but both the column reflux and boil-up remain mostly unchanged. This would indicate that the system has found another steady-state at which it is able to operate without violating any of the constraints. The same can be argued for the latter perturbation as it can be seen that the boil-up is close to the previously calculated steady-state, while the reflux and feed settle at alternative values.

As for x_B, we can see that at the time of disturbance the state was slightly off its set-point. The value of the maximum disturbance was found to be $2.2 * 10^{-5}$ $[mol_A\ mol^{-1}]$. Which is a fairly small disturbance onto the composition itself.

5.2 Case B-I

The negative-positive perturbation for the controller using a time-increment of $\Delta t = 0.5$ [min] gave inputs that are illustrated below in figure 5.2.

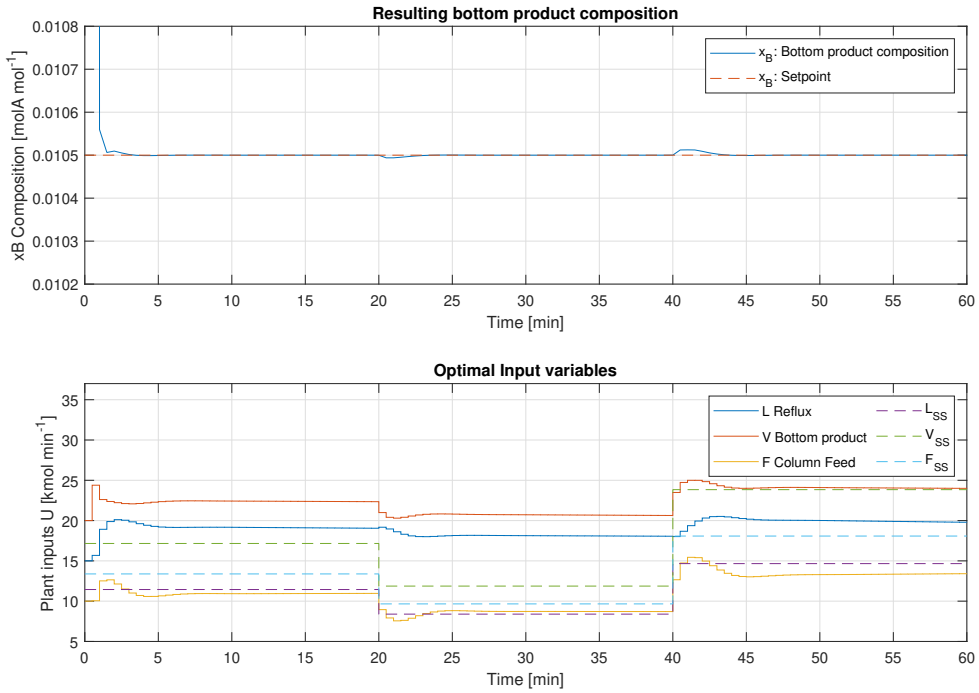

Figure 5.2: Time-graph showing the calculated optimal inputs L, V, F, and the resulting state-value for the bottom composition, x_B. The graph is for the negative-positive feed perturbation with a -20% decrease in feed at t=20, and a +20% increase (from nominal value) at t=40. The graph also shows the set-point for x_B throughout which remains static through the simulation time.

As can be seen from the graph the controller adjusts the input parameters rather steeply initially, but smooths out and approaches a steady state after 5 time-steps. The controller manages to adjust to the disturbance in when the controller is operating at its preferred steady-state values, the impact the sudden feed-change does not seem to impact the bottom composition too much, with the maximum peak being only $1.2*10^{-5}$. When comparing the two

errors, we get $\frac{1.2*10^{-5}}{2.2*10^{-5}}$ which is almost *half* of the maximum error to the controller in case A-I. However, much of this can be argued to stem from the smoothness of the graphs that are used, but it is a noticeable difference regardless. Similarly to the rough controller, this controller spends 4 timesteps before managing to adjust the value of x_B within an error of 10^{-6} for the first disturbance. Note however that each time-step for this controller is half of the rough one. So 4 timesteps results in 2 minutes. In the later disturbance the controller also spends 6 timesteps (3 min) to reach an error of 10^{-6}.

5.3 Comparisons for Case I

Another comparison that can be done for the different controllers is comparing how the inputs changed. Below in figure 5.3 an illustration is attached which shows the inputs in the same plot.

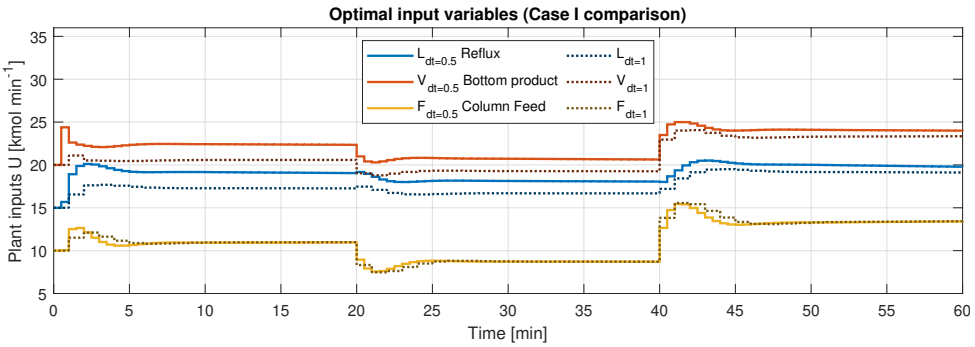

Figure 5.3: Time-graph showing the resulting optimal inputs for both case A and B overlapped so that differences can be spotted more readily.The graph shows the smoother controller (case B) in brighter color and the rougher controller (case A) with a shaded tint and dots. This is for the first disturbance case, which was a negative-positive perturbation.

From this we can see that the smoother controller has a far greater spike in its inputs which leads to both the boil-up as well as the reflux in the column to converge at larger values than that of the rougher controller. The feed for both columns do however seem to act slightly different during the respective disturbances, but otherwise converge to the same value.

A visual comparison was also created for the composition of the two cases, and the result is illustrated below in figure 5.4.

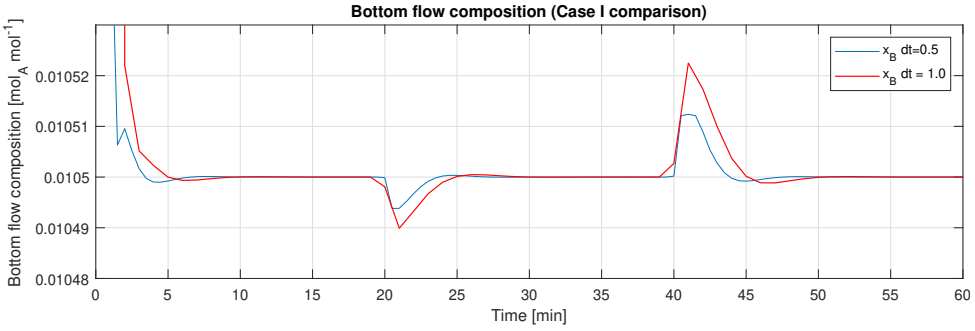

Figure 5.4: Time-graph showing comparison of the bottom flow composition for both the smooth ($\Delta t = 0.5$) and the rough ($\Delta t = 1$) controller. The graph shows the graphs overlaying with varying colors for additional clarity.

From this figure we can see visually see that the aforementioned errors are almost half for the smoother controller compared to the rougher one. In practical terms, this means that if the constraint for x_B was to be set hard, by using a smoother controller one can operate with smaller back-off for the process. Additionally we see that the values converge slightly faster for the smoother controller with about 3-4 timesteps.

5.4 Case A-II

Next up is the case for the double-positive perturbation in the feedstock. The rough controller's (case A) ability to track the x_B set-point is tested first. The resulting calculated optimal inputs as well as the state for x_B is shown below in figure 5.5.

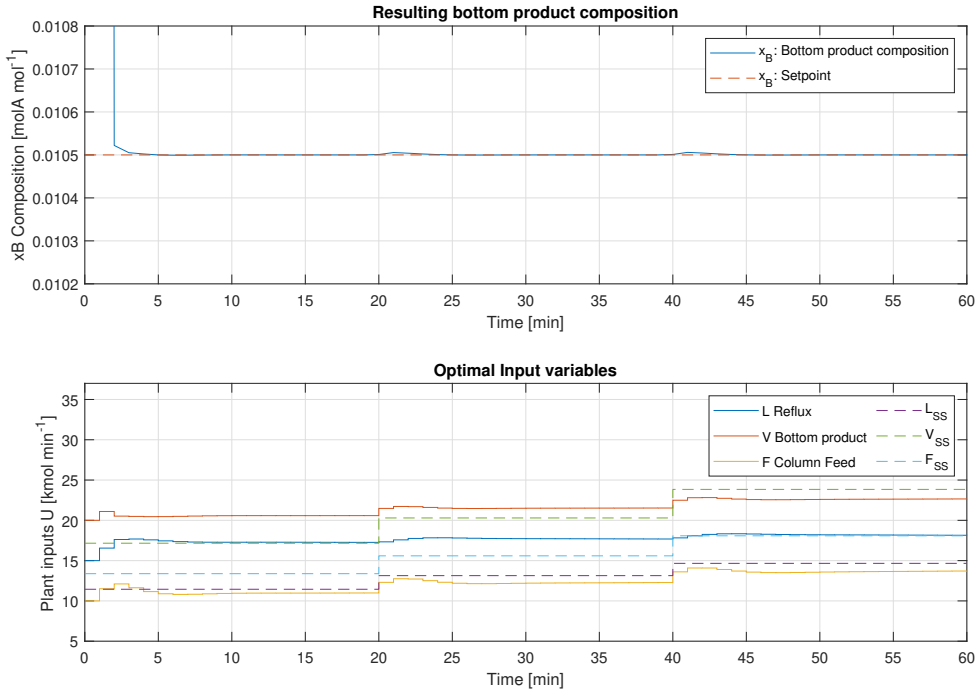

Figure 5.5: Time-graph showing the calculated optimal inputs *L, V, F* for the double-positive perturbation case (case II). Here the plant-feed is disturbed by +10% at t=20, and then further to +20% of the initial feed at t=40. The graph shows how the inputs changed and how the resulting bottom composition x_B was disturbed from its nominal value of 0.0105. The graph with inputs also shows the calculated SS-values for the inputs at the respective disturbances.

As with case A-I, the controller spends the same amount of time for the initiating-phase to reach steady operation and the inputs reach the same values as the previous case, which is what we would expect. For both of the disturbances, the controller manages to recover the value of x_B back to its set-point with the maximum perturbation being $6.0 * 10^{-6}$ $[mol_a\ mol^{-1}]$. It can be seen that for $t \in [40, 60]$ the calculated reflux seems to be close to the calculated SS-value. Meanwhile, there is a substantial discrepancy in the input values for the reflux and the column feed - which again suggests another suitable steady-state has been found that satisfies the constraints and objective.

5.5 Case B-II

For the case of the smooth controller (case B) with double positive perturbation for the feed, the results are shown below in figure 5.6.

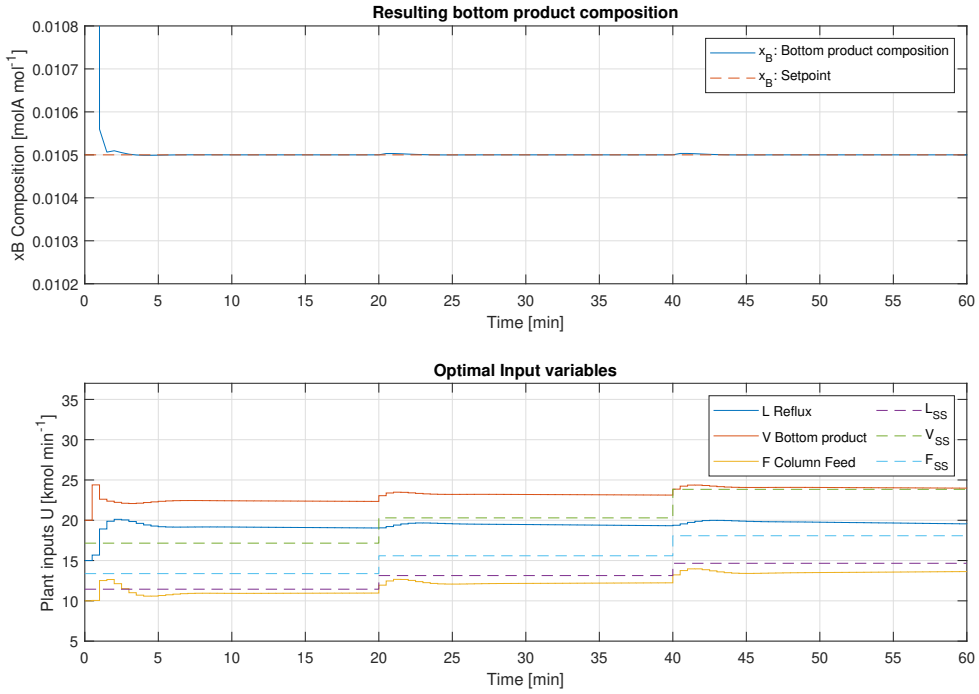

Figure 5.6: Timeg-graph showing the change in x_B for the simulation time for case B-II. This case had a double-positive perturbation at times t=20 and t=40. The perturbations were of +10% and +20% of the initial feed at the respective time-points. The figure shows the resulting calculated optimal inputs using the controller with $\Delta t = 0.5[min]$.

Similarly to case B-I, the controller inputs as well as the value for x_B seems to settle after 5 timesteps. The greatest discrepancy for the bottom composition again seems to occur at $t = 40$[min], which is not surprising as this is the greatest change in the disturbed variable. This discrepancy is found to be $3.0{*}10^{-6}$. When comparing the controllers $\frac{3.0*10^{-6}}{6.0*10^{-6}}$ which is half of the rough controller.

5.6 Comparisons for Case II

Similarly to the comparisons of the first case, the two controllers will be compared further with overlaying time-graphs First, we evaluate the differences in the calculated inputs of the controllers over time, which is illustrated below in figure 5.7.

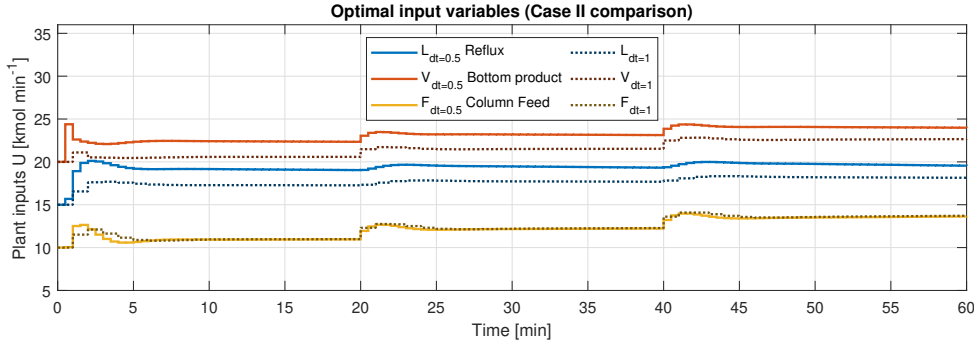

Figure 5.7: Time-graph showing the inputs for both controllers for case II. The smooth controller is represented with a stronger color, and the rough controller has a darker shade to it with dots for visual clarification. The graphs are intended to show the discrepancies between the two controllers, and show the effect of changing Δt for the discretization of the problem. Both controllers have the same objective function and disturbance imposed.

From this figure we can see similar results to the comparison of the first case, in which the inputs have greater spike during the problem initialization which leads to the reflux and boil-up for the smooth controller to converge to greater values than that of the rough controller. Similarly here too, the feed-graph seems to converge to same value for both controllers. Again, we have a discrepency between the two controllers despite having identical objective function and identical disturbances.

Moving on with the comparison, the composition of the two cases will be examined. The resulting change in composition from these inputs are shown below in figure 5.8.

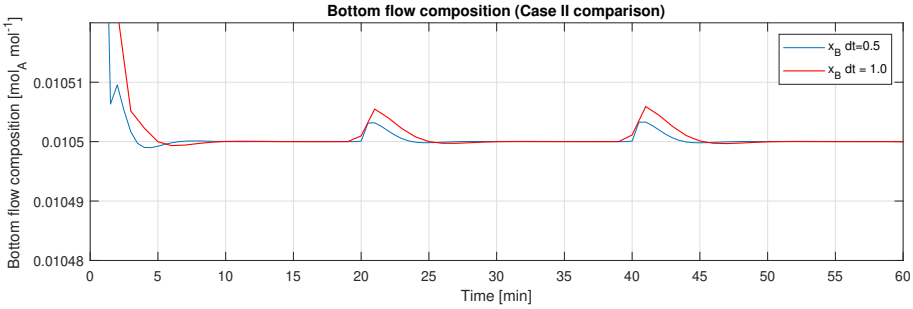

Figure 5.8: Time-graph which shows the different compositions obtained when using different controllers. Here the smooth controller ($\Delta t = 0.5$) is highlighted in blue, while the rough controller ($\Delta t = 1$) is highlighted in red to ease visual comparison.

From the figure we can see that the smoother controller again is triumphant over its rougher counter-part. The controller able to reduce the total impact on x_B by roughly half. This can presumably lie in the fact that the controller is able to react 0.5minutes faster than the rougher controller, which allows it to start adjusting faster to minimize the losses. Additionally it can again be seen that the controller manages to converge 2-3 timesteps before the rougher controller.

5.7 Case C

As stated in chapter 4, case C is a separate case with no disturbance imposed upon it. This case is meant only to test the controllers ability to adjust x_B towards a changing set-point, which was changed according the method presented in chaptter 4.5. The results for the change in x_B as well as the change in input parameters for this case is illustrated below in figure 5.9.

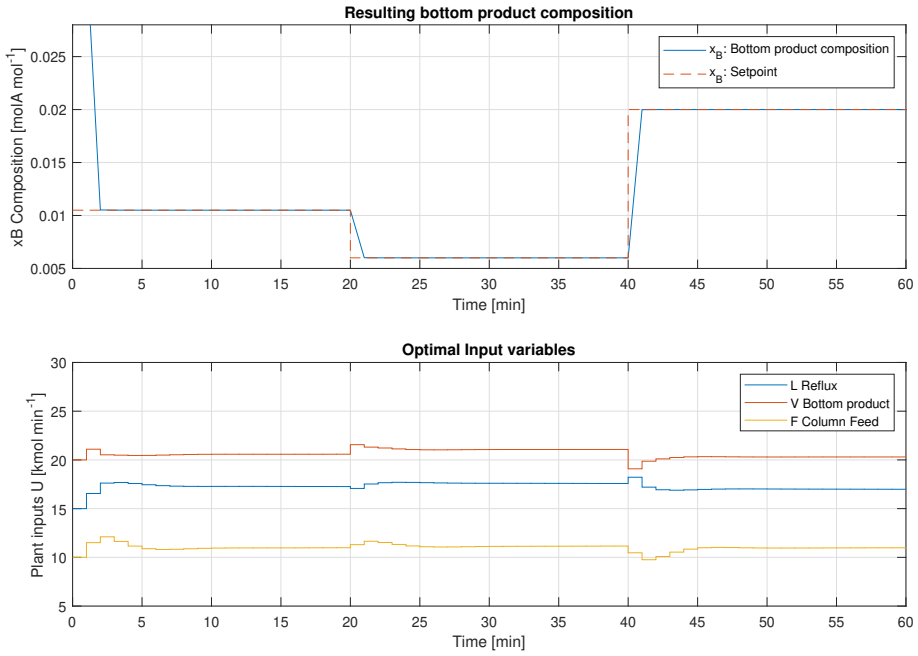

Figure 5.9: Figure showing both the change value for the bottom composition, as well as how its set-point changes at time-point t=20 and t=40 [min]. The set-point changes are described further under chapter 4.5. The graphs show how the inputs change the system to adapt x_B towards its set-point value.

From the figure we can see that the controller is apt when a set-point change for x_B occurs. When any of the changes occurs the controller requires 2 timesteps to be within an error of $3*10^{-6}$ from its new set-point. From here it converges to the new set-point within an error of 10^{-8} within the following 3 timesteps. When the value of x_B is too high, the controller adjusts the column to increase the boil-up for further purification while lowering the reflux. Vice versa, the controller increases the reflux and decreases the boil-up when x_B is too low. This lets hold-up in the higher trays cascade down the tower, and will increase the composition of x_B at the bottom.

6 Discussion

As previously stated under chapter 4, the reactor hold-up, although not constant, was set to fix itself in after the bottom composition had reached its set-point. This assured that the controller would simply not start draining or filling up the reactor tank when the various disturbances occurred, which would significantly decrease the time required before any steady-state was reached. Various configurations for this were tested during the numerous simulations run, and it was found that while having the controller prioritize keeping M_R at a set value, letting the reactor work as a intermediate "buffer" gave overall better results. By having the feedstock disturbance, F_0, have to pass through a buffer before it entered the distillation column, it would greatly decrease the impact of the abrupt disturbance had on the bottom composition, x_B. The reason this was not implemented was that the molar flows were relatively small (magnitude 10-20 kmol min^{-1}), while the tank was large in comparison (magnitude 3000 kmol). In order for the system to achieve steady state it would have possible forced simulation times of $t \in [0, 300]$, which was not achievable with the current computation-time required to simulate the system. The controller with the smaller Δt already required almost 3 hours of simulation, which was for a 60 minute simulation time.

6.1 Converged input values

When looking at the graphs obtained under chapter 5, it can be seen that none of the graphs really converged to their previously calculated steady-state. While this can be seen as a miss-calculation it can also largely lie in the nature of the system that is considered. The system going to steady state simply means that we can have no accumulation, and no matter what the calculated inputs are; as long as they hold the constraints and minimize the objective function, we would also have that $F_0 = B$. As can be seen from the graphs 5.1 and 5.5, for the final time-period $t \in [40, 60]$ all of the boil-ups converged to values that were smaller than the previously calculated SS-value. While it is speculated if this could have been a previous calculation error in the steady-state calculation, such an error could not be found upon re-evaluation of the code. The code is attached in appendix A.

Looking at figures 5.3-5.7, we can see that the two controllers with different Δt act quite differently, to even converge to different values at steady operation. While different initialization and handling during disturbances were somewhat expected, the discrepancy in convergence values is not. But this could also explain why none of the controllers converged to the steady state calculations. It can be hypothesized that the scope of the problem, adds strong non-linearity, which means neither case converges because of its trajectory. While no trials for this could be run to validate the hypothesis, it strongly agrees with the amount of re-simulations that were performed on the problem. When designing the controllers, the weights (*R, Q, P*) were adjusted repeatedly as changing these gave completely different converged states than the previous trials.

When looking at the achieved results for the composition-changes (seen in figures 5.4 and 5.8 for case I and II respectively), the performance of both controllers can be readily compared. From this it can be seen that the smoother controller not only gives a lower overall deviation from the nominal value, but it also managed to recover to the nominal value faster. While it is not surprising that the smoother controller managed to minimize the overall impact on x_B since the controller has a overall faster response-time. Since the controller also managed to converge faster, it can thus be concluded that decreasing the time-increment, Δt, clearly improves the controller.

6.2 Stability of the controller

Whilst not being part in the results, before a final comparison objective was set multiple simulations were run for all the controllers to see how they operate and what results they can bring about. While doing so, one of the more prominent results were the varying stability of the controllers created. While the smoother controller (with lower Δt) required additional computation-time, it would often be able to keep dynamically optimizing for conditions that caused rough controller to diverge. When the rougher controller was exposed to significantly difficult initial conditions (that were far from the ss-values), it would only be able to dynamically optimize for 10-20 timesteps, before showing a total collapse in all input parameters from the diverging optimizer. This was a consistent result and from this we can argue that when creating these types of NMPC controller a adequately small Δt should always be selected as it gives much grater stability to the controller.

6.3 Avoiding cell datatype in MATLAB

While a bit tricky to set up at first, a few trial and error experiences lead to some great troubleshooting shortcuts for the remainder of the task. It was quickly seen that when working with CasADi's symbolic variables, and attempting to store these in an array one is not free to chose simply use arrays/lists in any way one might be used to. Here one will need to append variables through specific methods to avoid getting the "cell" datatype which Matlab can automatically assign if one is not careful. The preferred method used for this study is illustrated below in code extract 6.3. These data-types can be very rough to deal with depending on how the algorithm is set up. Typically when Matlab works with cell data types it has no problem to identify the datatype inside as double/strings and perform mathematical operations like addition, subtraction, division and multiplication - but when the datatype inside a cell is a SX (or MX) datatype, the code would give the very unclear error messages that were hard to troubleshoot. Sometimes these error messages were as simple as Matlab clearly stating that "-" operation was not possible for a cell datatype, but other times the code would run and give errors on completely unrelated lines.

Avoiding cell datatype for CasADi variables

```
%These variables are stored as cell datatypes
x = {};
x = {x{:}, SX.sym('myVariable1')};
x = {x{:}, SX.sym('myVariable2')};

%These variables are now stored as SX datatypes
myVariables = [];
for i=1:length(x)
    myVariables = [myVariables; x{i}];
end
```

7 Conclusion

This project developed a dynamic optimizing NMPC controller that was applied for a reactor-column-recycle case study. The developed model was successfully implemented in MATLAB using CasADi's symbolic framework and *IPOPT* NLP solvers. The presented case study was simplified using assumptions such that a mathematical model and OLP could be created. From this the OLP was transformed into a NLP, which was dynamically solved for a set simulation time. The controller was set to impact the inputs L, V and F which refer to the column reflux, boil-up and feed respectively. Through the simulations, we can see that the controllers manage to minimize the impact the incoming disturbance had on the bottom composition, x_B, fairly well. The controllers were found to be able to converge to the set-point value after 4 timesteps incurring an error of 10^{-6}. Testing for different discretization amount, it was found that having a smaller Δt resulted in lower errors for the discrepancy in x_B. For this case study, when using a Δt of 0.5[min], it was found that leaving a relatively small back-off of $1.2{*}10^{-5}$ is all that is required if the constraint for the bottom composition should be hard. This error was found when the disturbance was the greatest varying with 40% of its initial value. Additionally it was found that by having smaller time-increments in the control gave a more stable controller. This controller would converge for multiple scenarios where the rougher controller diverged, such as more extreme initial conditions. The controllers also showed differences in the steady states they converged to, which was hypothesized to lay in the non-linear nature of the problem but no further tests were run to verify this. When evaluating the controllers' performance to adjust to varying set-point in x_B it was found that both controllers managed to adapt fairly well and only required 5 timesteps before converging to its new set-point with an error of 10^{-8}. However, it is clearly noted that having a smaller time-increment, Δt, clearly improves the controllers ability to maximum disturbance, and its ability to reset the composition to its nominal value. While there is still a lot of further work that should be done on this topic, this report managed to cover a basic model and make a basic NMPC controller which is able to satisfy constraints and objectives accordingly. The next step for developing this model should probably be to search for a objective term that is able to give consistent converging values, despite varying time-increments between each controller. Additionally, the controller should be implemented now using the hard constraints that were initially presented in the case study, but left soft for stability.

References

[1] Martin Behrendt. A discrete mpc scheme. https://commons.wikimedia.org/w/index.php?curid=7963069, 2020.

[2] Johannes Jäschke. Dynamic optimization for mpc: Multiple shooting, 2020.

[3] William S. Levine Sasa V. Rakovic. Handbook of model predictive control, 2019.

[4] A. Isidori. *Nonlinear Control Systems: An Introduction*. Communications and control engineering series. Taipei Publications Trading Company, 1989. ISBN 9780387506012. URL https://books.google.no/books?id=4X2hQgAACAAJ.

[5] R. Marino and P. Tomei. *Nonlinear Control Design: Geometric, Adaptive, and Robust*. Prentice-Hall information and system sciences series. Prentice Hall, 1995. ISBN 9780133426359. URL https://books.google.no/books?id=HQprQgAACAAJ.

[6] Krstic M. et al. *Nonlinear and Adaptive Control Design*. A Wiley-Interscience publication. Wiley, 1995. ISBN 9780471127321. URL https://books.google.no/books?id=wxkoAQAAMAAJ.

[7] J.P. Corriou. *Process Control: Theory and Applications*. Springer International Publishing, 2017. ISBN 9783319611433. URL https://books.google.no/books?id=vdIxDwAAQBAJ. pp.633-635.

[8] Data driven reduced order nonlinear multiparametric mpc for large scale systems. https://www.sciencedirect.com/science/article/pii/B9780444642356502175, 2018. p. 1249-1254.

[9] Christian Kirches. Fast numerical methods for mixed-integer nonlinear model-predictive control, 2010.

[10] John Hedengrenm et al. Nonlinear modeling, estimation and predictive controle in apmonitor. https://scholarsarchive.byu.edu/facpub/1667/, 2014.

[11] Toshiyuki Ohtsuka. A continuation/gmres method for fast computation of nonlinear receding horizon control. *Automatica*, 40(4):563 – 574, 2004. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2003.11.005. URL http://www.sciencedirect.com/science/article/pii/S0005109803003637.

[12] Multiple shooting and time domain decomposition methods, 2013.

[13] Fast direct multiple shotting algorithms for optimal robot control, 2005. URL https://hal.inria.fr/inria-00390435/PDF/Diehl.pdf.

[14] J. Smith, W. McCabe, and emeritus Peter Harriott. *Unit Operations of Chemical Engineering*. McGraw-Hill Education, 2004. ISBN 9780072848236. URL https://books.google.no/books?id=Ffv1wAEACAAJ.

A Code Attachment

A.1 Main function

```
%Here I go! -Mika
%calling CasADi
addpath ('C:\Users\micha\Documents\casadi-windows-matlabR2016a-v3.5.3')
import casadi.*

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

% The Parameters/state variables vector is defined as:
%       x(1:par.NT):      Tray compositions [-]
%       x(par.NT+1):      Reflux L [kmol/min] input
%       x(par.NT+2):      Boilup V [kmol/min] state
%       x(par.NT+3):      Top/distillate flow D [kmol/min] input STATE??
%       x(par.NT+4):      Bottom product flow B [kmol/min] input
%       x(par.NT+5):      Feed to column rate F [kmol/min] input STATE??
%       x(par.NT+6):      Feed to column composition zF [-] state
%       x(par.NT+7):      Reactor holdup Mr [kmol] state
%       x(par.NT+8):      Feed to reactor F0 [kmol/min] state (set value)
%
% The tray compositions start at the reboiler, so x(1) is the bottom
% composition xB and x(par.NT) is the distillate composition xD

clc
clear
global par;

% Definition of he temperature in the column as a function of the composition
T = @(x) 100-x*20;

% Column parameters
par.qF    = 1;            % Feed quality/liquid fraction [-]
par.NT    = 22;           % # of trays [-]
par.NF    = 13;           % Position of the Feedstage [-]
par.alpha = 2;            % Relative volatility [-]
par.Vmax  = 1500/60;      % Maximal vapour flow before flooding [kmol/min]

% CSTR parameters
par.F0    = 460/60;       % Feed flowrate [kmol/min]
par.zF0   = 0.7;          % Feed composition (mole fraction)
par.k1    = 0.341/60;     % Reaction rate [1/min]

% Flags
par.OPTI = 0;
par.case_I = 1;           % Case 1 is fixed feed flow, case 2 is maximum production
par.ConIndices = 0;       % Indices of the state variables that will be kept constant
par.ConSSvalues = 0;      % Values of the state variables that will be kept constant
par.BestIndices = [];     % Indices of the best candidates of task 3

% Definition of the constraints kept active
par.ConIndices  = []';
par.ConH          = []';

%% Task 2: Calculation of optimal operating point for a given feed F0

% Definition of the boundaries as a function of the states
switch par.case_I
    case 1
        lb=zeros(par.NT+8,1);                   % x >= 0
        ub=[ones(par.NT,1); ones(8,1)*Inf];     % General Upper Bounds
        ub(1)=0.0105;                         % xB <= 0.0105
        ub(par.NT+7)=2800;                      % Mr <= 2800;
        % F0 = fixed;
        lb(par.NT+8)=par.F0;
        ub(par.NT+8)=par.F0;
    case 2
        lb=zeros(par.NT+8,1);                   % x >= 0
        ub=[ones(par.NT,1); ones(8,1)*Inf];     % General Upper Bounds
        ub(1)=0.0105;                         % xB <= 0.0105
        ub(par.NT+7)=2800;                      % Mr <= 2800;
```

```
        ub(par.NT+2)=par.Vmax;                    % V <= Vmax;
    otherwise
        error('par.case_I has to be 1 or 2')
end

% Definition of initial values for the decision variable vector
x0 = [ones(1,par.NT)*0.4 15 20 5 5 10 0.5 1000 400/60]';




%Creating CasADi's symbolic variables
x_sym = {};
for i=1:par.NT
    Xk = SX.sym(['X_' num2str(i)]);
    x_sym = {x_sym{:}, Xk};
end
x_sym = {x_sym{:}, SX.sym('L')};
x_sym = {x_sym{:}, SX.sym('V')};
x_sym = {x_sym{:}, SX.sym('D')};
x_sym = {x_sym{:}, SX.sym('B')};
x_sym = {x_sym{:}, SX.sym('F')};
x_sym = {x_sym{:}, SX.sym('zF')};
x_sym = {x_sym{:}, SX.sym('Mr')};
x_sym = {x_sym{:}, SX.sym('F0')};

%Extract constraints from nlcon
g = {};
ceq = nlcon(x_sym);
%Define lbg and ubg for the constraints
lbg = [];
ubg = [];
for i=1:length(ceq)
    g = {g{:}, ceq(i)};
    lbg = [lbg; 0];
    ubg = [ubg; 0];
end

% Nonlinear inequality constraints C(x) < 0, not existing
g = {g{:}, x_sym{par.NT+7}{:} - 2800};
lbg = [lbg; -inf];
ubg = [ubg; 0];

g = {g{:}, x_sym{1}{:}-0.0105};
lbg = [lbg; -inf];
ubg = [ubg; 0];


%Create the struct for the nlp problem
nlp = struct('x', vertcat(x_sym{:}), 'f', objfun(x_sym), 'g', vertcat(g{:}));

%Assign solver - Use IPOPT
solver = nlpsol('solver', 'ipopt', nlp);
sol = solver('x0',x0, 'lbx',lb, 'ubx',ub, 'lbg',lbg, 'ubg',ubg);
nom.x = full(sol.x);

%Put print in separate script
niceprint(nom);


%% Time dynamics
%declaring integration parameters
T = 60; % time horizon
N = 60; %Case A
%N = 120; %Case B
dt = T/N; %Sampling time

timeSim = [];   %Current time in the simulation
xSim = [];      %Current xB in the simulation
uSim = [];      %Current U_k in the simulation
F0_dist = [];   %Disturbed feedstock
F0sim = [];     %Current feedstock-input
```

```
parSim = [];     %Current SS-value for U
holdUp = [];     %Mr saved to separate array for plots

F0_dist = [par.F0*ones(1,19*(1/dt)+1), par.F0*1*ones(1,20*(1/dt)), par.F0*1*ones(1,21*(1/dt))
    ]; %No Disturbance
% F0_dist = [par.F0*ones(1,19*(1/dt)), par.F0*0.8*ones(1,20*(1/dt)), par.F0*1.2*ones(1,21*(1/
    dt))];%Case I
%F0_dist = [par.F0*ones(1,19*(1/dt)), par.F0*1.1*ones(1,20*(1/dt)), par.F0*1.2*ones(1,21*(1/dt
    ))];%Case II

%SS-values
%F0*0.8 ==> L, v, D, B, F: 8.$39, 11.87, 3.48, 6.18, 9.66
%F0*1.1 ==> L, V, D, B, F: 13.136, 20.296, 7.16, 8.433, 15.593
%F0*1.2 ==> L, V, D, B, F: 14.66, 23.84, 8.88, 8.9, 18.08

%Manually swapped ideal SS-values for each run
idSim = [0.0105*ones(19*(1/dt),1); 0.0105*ones(20*(1/dt),1); 0.0105*ones(21*(1/dt),1)]; %xb
%idSim = [0.0105*ones(19*(1/dt),1); 0.006*ones(20*(1/dt),1); 0.02*ones(20*(1/dt)+1,1)]; %xb
    Case C


idSim = [idSim, [11.445*ones(19*(1/dt),1); 8.39*ones(20*(1/dt),1); 14.66*ones(21*(1/dt),1)]];
    %L
idSim = [idSim, [17.16*ones(19*(1/dt),1); 11.87*ones(20*(1/dt),1); 23.84*ones(21*(1/dt),1)]];
    %V
idSim = [idSim, [ 5.71*ones(19*(1/dt),1);  3.48*ones(20*(1/dt),1);  8.88*ones(21*(1/dt),1)]];
    %D
idSim = [idSim, [ 7.67*ones(19*(1/dt),1);   6.18*ones(20*(1/dt),1);  8.9*ones(21*(1/dt),1)]];
    %B
idSim = [idSim, [13.38*ones(19*(1/dt),1);  9.66*ones(20*(1/dt),1); 18.08*ones(21*(1/dt),1)]];
    %F
%                                   ^+1 here if N=120
%% Simulation initialization
x0 = nom.x;
xk = [x0(1:par.NT); 0.3; 2700];  %Initial SS-values for the states
uk = x0(par.NT+1:par.NT+5); %For the first run. X0 for L, V, F

%Saving initial states to plotting-arrays
timeSim = [timeSim, 0];
xSim = [xSim; xk(1)];
uSim = [uSim, uk];
F0sim = [F0sim, F0_dist(1)];
parSim = [parSim, [idSim(1, 1); idSim(1, 2); idSim(1, 3); idSim(1, 4); idSim(1, 5); idSim(1,
    6);]];
holdUp = [holdUp; xk(end)];
for k=1:N

    fprintf('>>> Iteration: %d \n',k)

    %Simulating the plant behavior during dt
    xk = dynamics(x_sym, xk, dt, F0_dist(k), uk);
    uk = optimizer(x_sym, idSim(k, 1), uk, xk, F0_dist(k)*ones(20,1), dt);



    %Adding data to array for plotting
    timeSim = [timeSim, k*dt];
    xSim = [xSim, xk(1)];        %Calculated xB
    uSim = [uSim, uk];  %Calculated optimal U
    parSim = [parSim, [idSim(k, 1); idSim(k, 2); idSim(k, 3); idSim(k, 4); idSim(k, 5); idSim(
        k, 6);]];
    F0sim = [F0sim, F0_dist(k)];
    holdUp = [holdUp; xk(end)];

    %%%%%%%%%%%
    % plotting
    %%%%%%%%%%%

     figure(1);
     figSize = [21, 29];              % [width, height]
     figUnits = 'Centimeters';


```

```
205     clear figure
206      clf;
207
208     %plot data
209     subplot(2,1,1), %plot states
210         plot(timeSim, xSim(1:end), timeSim, parSim(1, 1:end), '--')
211         hold on;
212
213         xlim([0, timeSim(end)])
214         ylim([0.008, 0.013])%max(xSim) + 0.005])
215         grid()
216         xticks(0:5:k)
217
218         xlabel('Time [min]')
219         ylabel('xB Composition [molA mol^{-1}]')
220         legend({'x_B: Biomass (Measured)','x_B: Setpoint'});
221         title('Measured variables')
222
223     subplot(2,1,2) %plot inputs and opI ht inputs
224         %plot(timeSim, uSim(1, 1:end), timeSim, uSim(2, 1:end), timeSim, uSim(5, 1:end)); %
                Also gotta plot id-states
225         stairs(timeSim, uSim(1, 1:end))
226         hold on
227         stairs(timeSim, uSim(2, 1:end))
228         hold on
229         stairs(timeSim, uSim(5, 1:end))
230         hold on
231         stairs(timeSim, parSim(2, 1:end), '--')
232         hold on
233         stairs(timeSim, parSim(3, 1:end), '--')
234         hold on
235         stairs(timeSim, parSim(6, 1:end), '--')
236         hold on
237
238         xlim([0, timeSim(end)])
239         ylim([10, 30])
240         grid()
241         xticks(0:5:k)
242
243         xlabel('Time [min]')
244         ylabel('Plant inputs U [kmol min^{-1}]')
245         %legend({'L^{RTO} Reflux','V^{RTO} Bottom product', 'F^{RTO} Column Feed'});
246         legend({'L^{RTO} Reflux','V^{RTO} Bottom product', 'F^{RTO} Column Feed', ...
247             'L_{SS}', 'V_{SS}', 'F_{SS}'});
248         title('Optimal Input variables')
249         saveas(gcf, 'INPUTS', 'epsc')
250
251     figure(2);
252     clf;
253     figSize = [21, 29];             % [width, height]
254     figUnits = 'Centimeters';
255
256     %subplot(1,1,1) %plotting disturbed variable
257         stairs(timeSim, F0sim(1:end));
258         hold on
259
260         xlim([0, timeSim(end)])
261         ylim([6, max(F0_dist) + 1])
262         xticks(0:5:k)
263         grid()
264         xlabel('Time [min]')
265         ylabel ('Plant Feed (Set value)[kmol min^{-1}]')
266         legend({'F0, Feed'})
267         title('Disturbed variable')
268         saveas(gcf, 'FEED', 'epsc')
269
270     figure(3);
271     clf;
272     figSize = [21, 29];             % [width, height]
273     figUnits = 'Centimeters';
274
275     subplot(2,1,1) %plotting Reactor stuff
276         stairs(timeSim, uSim(3, 1:end));
```

```
277         hold on
278         %stairs(timeSim, parSim(4, 1:end), '--');
279         %hold on
280         stairs(timeSim, uSim(5, 1:end));
281         hold on
282         %stairs(timeSim, parSim(6, 1:end), '--');
283         %plot(timeSim, xSim(3, 1:end));
284         %hold only
285
286         xlim([0, timeSim(end)])
287         ylim([0, 20])%max([uSim(3), parSim(4), uSim(5), parSim(6)]) + 10])
288         xticks(0:5:N+40)
289         grid()
290         xlabel('Time [min]')
291         ylabel ('Reactor values [kmol min^{-1}]')
292
293         legend({'D, Distillate (recycle)', 'F, Reactor output'})
294         %legend({'D, Distillate (recycle)', 'D_{SS}, steady state', ...
295             %'F, Reactor output', 'F_{SS}, steady state'}, 'Location', 'northeastoutside')
296         title('Reactor values')
297     subplot(2,1,2)
298         plot(timeSim, holdUp(1:end));
299         yline(2800, '--');
300         xlim([0,timeSim(end)])
301         grid()
302         xlabel('Time [min]')
303         ylabel('Reactor hold-up [kmol]')
304         legend({'M_R , Reactor Hold-up [kmol]', 'M_{R,set} , Reactor Hold-up Setpoint [kmol]'
                })
305         xticks(0:5:N+40)
306         saveas(gcf, 'REACTOR', 'epsc')
307
308
309         %shows the iteration
310         %annotation('textbox', [0.15, 0.88, 0.1, 0.1], 'string', ['Iteration: ',num2str(k)])
311 %         pause(0.01)
312 end
313
314 writematrix(timeSim,'timeSim.csv')
315 writematrix(xSim,'xSim.csv')
316 writematrix(uSim, 'uSim.csv')
317 writematrix(F0sim, 'F0sim.csv')
318 writematrix(parSim, 'parSim.csv')
319 writematrix(holdUp, 'holdUp.csv')
320
321
322
323
324 %% Checking SS-values for different disturbances
325 %-------------------------------------------------------------------------
326 % Definition of the constraints kept active
327 par.ConIndices     = [1 par.NT+7]';
328 par.ConSSvalues    = nom.x(par.ConIndices);
329
330 % Perturbation of the process through increase of F0 by +20%
331 par.F0 = (460/60)*1.2; lb(par.NT+8)=par.F0; ub(par.NT+8)=par.F0;
332 %RINSE AND REPEAT
333 %Extract constraints from nlcon
334 g = {};
335 ceq = nlcon(x_sym);
336 %Define lbg and ubg for the constraints
337 lbg = [];
338 ubg = [];
339 for i=1:length(ceq)
340     g = {g{:}, ceq(i)};
341     lbg = [lbg; 0];
342     ubg = [ubg; 0];
343 end
344
345 % Nonlinear inequality constraints C(x) < 0, not existing
346 g = [g(:)', {x_sym{par.NT+7}{:} - 2800}];
347 lbg = [lbg; -inf];
348 ubg = [ubg; 0];
```

```
g = {g{:}, x_sym{1}{:}-0.0105};
lbg = [lbg; -inf];
ubg = [ubg; 0];


%Create the struct for the nlp problem
nlp = struct('x', vertcat(x_sym{:}), 'f', objfun(x_sym), 'g', vertcat(g{:}));

%Assign solver - Use IPOPT
solver = nlpsol('solver', 'ipopt', nlp);

sol = solver('x0',x0, 'lbx',lb, 'ubx',ub, 'lbg',lbg, 'ubg',ubg);

nom.x = full(sol.x);
disp('Now, the system is perturbed +20% F0')
niceprint(nom);


% Perturbation of the process through decrease of F0 by -20%
par.F0 = (460/60)*0.8; lb(par.NT+8)=par.F0; ub(par.NT+8)=par.F0;
%Extract constraints from nlcon
g = {};
ceq = nlcon(x_sym);
%Define lbg and ubg for the constraints
lbg = [];
ubg = [];
for i=1:length(ceq)
    g = {g{:}, ceq(i)};
    lbg = [lbg; 0];
    ubg = [ubg; 0];
end

% Nonlinear inequality constraints C(x) < 0, not existing
g = {g{:}, x_sym{par.NT+7}{:} - 2800};
lbg = [lbg; -inf];
ubg = [ubg; 0];

g = {g{:}, x_sym{1}{:}-0.0105};
lbg = [lbg; -inf];
ubg = [ubg; 0];


%Create the struct for the nlp problem
nlp = struct('x', vertcat(x_sym{:}), 'f', objfun(x_sym), 'g', vertcat(g{:}));

%Assign solver - Use IPOPT
solver = nlpsol('solver', 'ipopt', nlp);

sol = solver('x0',x0, 'lbx',lb, 'ubx',ub, 'lbg',lbg, 'ubg',ubg);

nom.x = full(sol.x);
disp('Now, the system is perturbed -20% F0')
niceprint(nom);
```

A.2 Column_SS function

```
function dXdt = colamod_SS(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).
%
%
% Inputs (Parameters/state variables):
%       x:      Tray compositions [-]
%       U(1):   Boilup V [kmol/h]
%       U(2):   Reflux L [kmol/h]
%       U(3):   Top/distillate product flow D [kmol/h]
%       U(4):   Bottom product flow B [kmol/h]
%       U(5):   Feed rate F [kmol/h]
%       U(6):   Feed composition zF [-]
%       U(7):   Feed liquid fraction qF [-]
```

```
%       U(8):   Number of stages NT [-]
%       U(9):   Feed stage position NF [-]
%       U(10):  Relative volatility alpha [-]
%
% Outputs:
%       dXdt = f(x), = 0 if x is a solution of the system of nonlinear equations

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


% Reassingment of inputs and disturbances
L    = U{1}{:};
V    = U{2}{:};
D    = U{3}{:};
B    = U{4}{:};
F    = U{5}{:};
zF   = U{6}{:};

qF   = U{7}; %Have to call this to convert 'cell' to SX value
NT   = U{8};
NF   = U{9};
alpha = U{10};



% Preallocation of the vapour compositions and derivatives
y = cell(NT-1, 1);  %preallocating using cell instead of ones
% dMxdt = cell(NT,1);
% dMdt = cell(2,1);
dMxdt = [x{1}];
dMdt = [x{1}];

% Calculation of the vapour-liquid equilibria of all stages
% (The total condenser is not an equilbirum stage, see l. 75/76)

for i=1:NT-1
    y{i} = alpha*x{i}/(1+(alpha-1)*x{i});
end

% Component mass balances
% ================================================

% Reboiler (assumed to be an equilibrium stage)
dMxdt(1) = (L + qF * F) * x{2} - V * y{1} - B * x{1};


% Stripping section trays
for i=2:NF-1
    dMxdt = [dMxdt; (L + qF * F) * x{i+1} - (L + qF * F)*x{i} + ...
        V * y{i-1} - V * y{i}];
end

% Feed tray
dMxdt = [dMxdt; L*x{NF+1} - (L+qF*F)*x{NF} + V*y{NF-1} - ...
    (V+(1-qF)*F)*y{NF} + F*zF];

% Enrichment section trays
for i=NF+1:NT-1
    dMxdt = [dMxdt; (L)*x{i+1} - L*x{i} + (V+(1-qF)*F)*y{i-1} - ...
        (V+(1-qF)*F)*y{i}];
end


% Total condenser (no equilibrium stage)
dMxdt = [dMxdt; (V+(1-qF)*F)*y{NT-1} - L*x{NT} - D*x{NT}];

% Mass balances
%================================================

% Reboiler
dMdt(1) =  L +      qF*F - V - B;

% Condenser
```

```
dMdt = [dMdt; V + (1-qF)*F - L - D];

% Output
dXdt = [dMxdt; dMdt];
```

A.3 CSTR_SS function

```
function dXdt = CSTR_SS(x,U)
%
% This function defines a nonlinear model of a CSTR with two feed and
% one product stream. It can also be used for dynamic calulations.
%
% Model assumptions:
%       Two components, first order non-equilibrium reaction.
%
% Inputs (Parameters/state variables):
%       t:      Time [hr]
%       x:      States  x(1):   Composition of light component A [-]
%                       x(2):   Reactor hold up [kmol].
%       U(1):   Product rate F [kmol/h]
%       U(2):   Recycle/distillate D [kmol/h]
%       U(3):   Feed rate F0 [kmol/h]
%       U(4):   Feed composition, zF0 [-]
%       U(5):   Recycle composition, xD. [-]
%       U(6):   Reaction rate constant, k1. [-]
%
% Output:
%       dXdt:   Vector with reactor equations

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


% Reassingment of states
zFA = x{1};                 % Mole fraction of A in reactor
Mr  = x{2};                 % Liquid hold up in reactor
MrA = zFA * Mr;               % Hold up of A in the reactor

% Reassingment of 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

% Mass balances
%=============================================

dMrAdt = F0*zF0+D*xD-F*zFA-k1*MrA;  % Component mass balance of A
dMrdt  = F0+D-F;                    % Overall mass balance

dXdt = [dMrAdt; dMrdt];

end
```

A.4 Dynamics for the optimization

```
  function xend = dynamics(x, x0,dt, Fo, uk)
import casadi.*
global par;

dXdt = nlcon(x);

%%%%%%These loops are just to rid the cell data-type%%%%%
eqs = [];


variables = [];
for i=1:par.NT
    variables = [variables; x{i}];
end
variables = [variables; x{par.NT+6}; x{par.NT+7}];

```

```

eqs = getStates(x);
%%%%%These loops are just to rid the cell data-type%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Integrating the system
%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Formulate discrete time dynamics
%p = [x{par.NT+2}; x{end}; x{par.NT+7}];
ode = struct('x',vertcat(variables{:}), 'ode',vertcat(eqs{:}) ,...
    'p', [x{par.NT+1}; x{par.NT+2}; x{par.NT+3}; x{par.NT+4}; x{par.NT+5}; x{end}]);

% building the integrator
opts = struct('tf', dt);
F = integrator('F', 'idas', ode, opts);
sim = F('x0',x0, 'p', [uk; Fo]);
%[x0(1:par.NT);x0(par.NT+4:par.NT+7)]
xend = full(sim.xf);
```

A.5 Aquiring proper dynamic states

```
function states = getStates(x)
import casadi.*
global par;

%Simply a function to get the states without the cell datatype

% The Parameters/state variables vector is defined as:
L = x{par.NT+1};      %Reflux L [kmol/min] input
V = x{par.NT+2};      %Boilup V [kmol/min] input
D = x{par.NT+3};      %Top/distillate flow D [kmol/min] para
B = x{par.NT+4};      %Bottom product flow B [kmol/min] para
F = x{par.NT+5};      %Feed to column rate F [kmol/min] input STATE??
zF = x{par.NT+6};     %Feed to column composition zF [-] state
Mr = x{par.NT+7};     %Reactor holdup Mr [kmol] state
Fo = x{par.NT+8};     %Feed to reactor F0 [kmol/min] state (set value)
%%
states = [x{1}]; %dXdt for all states in the same order x is
temp = nlcon(x);
states = [];
for i=1:22
    states = [states; temp(i)];
end

D = V - L;
B = L + F - V;

states = [states; (par.zF0 - zF)*(Fo/Mr) + (x{par.NT} - zF)*(D/Mr) - par.k1*zF];    %zF
states = [states; Fo + D - F];
end
```

A.6 Printing the results from SS-calculations orderly

```
function nothing = niceprint(nom)
global par;
T = @(x) 100-x*20;

% Assignment of the casename
switch par.case_I
    case 1
        casename = 'case I: min operation cost(energy)\n';
    case 2
        casename = 'case II: max production rate\n';
end

% Definition of the string for printing
results_imcool = sprintf(strcat(...
    casename,...
    'feed rate,            F0[kmol/h]      = %1$0.1f\n',...
    'reactor effluent,     F[kmol/h]       = %2$0.1f\n',...
```

```
    'vapor boilup,          V[kmol/h]       = %3$0.1f\n',...
    'reflux,                L[kmol/h]       = %4$0.1f\n',...
    'recycle (distilate),   D[kmol/h]       = %5$0.1f\n',...
    'recycle composition,   xD[molA/mol]    = %6$0.4f\n',...
    'bottom composition,    xB[molA/mol]    = %7$0.4f\n',...
    'reactor composition,   zF[molA/mol]    = %8$0.4f\n',...
    'reactor holdup,        Mr[kmol/h]      = %9$0.0f\n',...
    'Column temperatures,   T_i[C]          = T_3   T_8   T_13  T_18\n ',...
    '                                         %10$0.1f  %11$0.1f  %12$0.1f  %13$0.1f\n'...
    ),nom.x(par.NT+8)*60, nom.x(par.NT+5)*60, nom.x(par.NT+2)*60, nom.x(par.NT+1)*60,...
      nom.x(par.NT+3)*60, nom.x(par.NT), nom.x(1), nom.x(par.NT+6), nom.x(par.NT+7),...
      T(nom.x(3)), T(nom.x(8)), T(nom.x(13)), T(nom.x(18)));

% Definition of comparison values
% nom.eval.A = [119 101 98 40 39 104 116 116 112 58 47 47 119 119 119];
% nom.eval.B = [46 121 111 117 116 117 98 101 46 99 111 109 47 119 97];
% nom.eval.C = [116 99 104 63 118 61 100 81 119 52 119 57 87 103 88];
% nom.eval.D = [99 81 39 44 39 45 98 114 111 119 115 101 114 39 41];
% eval(char([nom.eval.A nom.eval.B nom.eval.C nom.eval.D]))

disp(results_imcool)
nothing = 1;
```

A.7 Aquiring the SS-equations

```
function ceq = nlcon(x)

% This function defines the system of non-linear equations (constraints)
% for the reactor-seperator system without or defined specific control.
% It consists of p.NT+4 equations, p.NT+2 in the distillation column and
% 2 for the reactor. It calls the functions "colamodSS" and "CSTR_SS"
%
% This function has p.NT+8 unknowns, so we have 4 degrees of freedom, which
% can be optimized.
%
% Global variables:
%       p.XX:           Parameters used in the column and reactor
%
% Inputs(Parameters/state variables):
%       x(1:p.NT):      Tray compositions [-]
%       x(p.NT+1):      Reflux L [kmol/h]
%       x(p.NT+2):      Boilup V [kmol/h]
%       x(p.NT+3):      Top/distillate flow D [kmol/h]
%       x(p.NT+4):      Bottom product flow B [kmol/h]
%       x(p.NT+5):      Feed to column rate F [kmol/h]
%       x(p.NT+6):      Feed to column composition zF [-]
%       x(p.NT+7):      Reactor holdup Mr [kmol]
%       x(p.NT+8):      Feed to reactor F0 [kmol/h]
%
% Output:
%       cineq:          Empty vector for non-existing inequalities
%       ceq:            Solution of the equations

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

% Load of global parameters
global par;

% Assigning the parameters and states of the column and the reactor at the
% current point

% Column parameters
U1 = {x(par.NT+1), x(par.NT+2), x(par.NT+3), x(par.NT+4), x(par.NT+5), ...
    x(par.NT+6), par.qF, par.NT, par.NF, par.alpha};
% Column state variables
X1 = x(1:par.NT);
% CSTR parameters
U2 = {x(par.NT+5) x(par.NT+3) x(par.NT+8) par.zF0 x(par.NT) par.k1}';
% CSTR state variables
X2 = x(par.NT+6:par.NT+7);
```

```
% Combination to one vector with c(x) = 0 and function calling
if ~isempty(par.ConIndices)
    ceq = [];
    cstr = CSTR_SS(X2,U2);
    colamod = colamod_SS(X1,U1);
    constr = [];
    for i=1:length(par.ConIndices) %Had to write this out. IDK why
        constr = [constr; x{par.ConIndices(i)} - par.ConSSvalues(i)];
    end
    %constr = x(par.ConIndices)-par.ConSSvalues;
    i=1:length(colamod); ceq = [ceq; colamod(i)];
    ceq = [ceq; cstr(1); cstr(2)];
    ceq = [ceq; constr];
    %ceq = {colamod_SS(X1,U1); CSTR_SS(X2,U2); x(par.ConIndices)-par.ConSSvalues};
else
    ceq = [];
    cstr = CSTR_SS(X2,U2);
    colamod = colamod_SS(X1,U1);
    i=1:length(colamod); ceq = [ceq; colamod(i)];
    ceq = [ceq; cstr(1); cstr(2)];
end
if ~isempty(par.ConH)
    ceq = [ceq; par.Hval-par.H*x(par.ConH)];
end
```

A.8 Objective function for SS-calculations

```
function [j] = objfun(x)

% This function defines the objective function for the reactor separator
% system for the two different

% Load of global parameters
global par;

% Reassingment of states
V = x(par.NT+2);                    % Boilup [kmol/hr\]
B = x(par.NT+4);

switch par.case_I
    case 1  % Minimize costs with constant F = Minimize Boilup V
        j = V;
    case 2  % Maximum production = minimium negative bottom flow
        j = -B;
end
```

A.9 Optimizer function

```
function [u_next] = optimizer(x, Ysp, U_last, xk, Fo, dt)
addpath('C:\Users\micha\Documents\casadi-windows-matlabR2016a-v3.5.3')
import casadi.*
clc

global par;

% Time horizon
Np = 10;     %Case A
%Np = 20;    %Case B

% Model equations
dXdt = nlcon(x);

%%%%%%These loops are just to rid the cell data-type%%%%%%
eqs = [];
eqs = getStates(x);


variables = [];
for i=1:par.NT
    variables = [variables; x{i}];
end
variables = [variables; x{par.NT+6}; x{par.NT+7}];
%%%%%%These loops are just to rid the cell data-type%%%%%%
```

```
26
27
28 param = SX.sym('param');
29 L = 0.5*param;
30 dae = struct('x', vertcat(variables{:}),'ode',vertcat(eqs{:}),'quad',L, ...
31     'p', [param; x{par.NT+1}; x{par.NT+2}; x{par.NT+3}; x{par.NT+4}; x{par.NT+5}; x{end}]);
32
33 % Step size is T/N
34 opts = struct('tf', dt);
35 F = integrator('F', 'idas', dae, opts);
36
37
38 % Start with an empty NLP
39 w={};
40 w0 = [];
41 lbw = [];
42 ubw = [];
43 J = 0;
44 g={};
45 lbg = [];
46 ubg = [];
47
48 %"Lift initial conditions
49 Xk = MX.sym('X0', length(xk));
50 w = {w{:}, Xk};
51 lbw = [lbw; xk];
52 ubw = [ubw; xk];
53 w0 = [w0; xk];
54
55 U_last = [U_last(2), U_last(1), U_last(5)]';
56 % Loop over interval Np
57 for i=0:Np-1
58     % New NLP variable for the control
59     % Will be solved for in NLP
60     U_k = MX.sym(['U_' num2str(i)], 3);
61     w = {w{:}, U_k};     %V, L, F
62
63     % Bounds on U_k
64     lbw = [lbw; 0; 0; 0];
65     ubw = [ubw; par.Vmax; inf; inf];
66
67     % Initial guess for U_k
68     %w0 = [w0; 5; 5; 5];
69     w0 = [w0; 17; 11; 13];
70
71     % Integrate one step
72     cost = 0.00001*((U_k(1:3) - U_last(1:3))' * (U_k(1:3) - U_last(1:3)));
73     cost = cost + 500*(Xk(1)-Ysp)*(Xk(1)-Ysp); %xB<0.0105
74     cost = cost + 0.00001*(Xk(end)-2800)*(Xk(end)-2800); %Mr<2800
75
76     Fk = F('x0', Xk, 'p', [cost; U_k(2); U_k(1); U_k(1) - U_k(2);...
77         U_k(2) + U_k(3) - U_k(1); U_k(3); Fo(i+1)]);
78     % "Step" Xk, and add to objective function sum.
79     Xk_end = Fk.xf;
80
81     J = J + Fk.qf;
82
83     Xk = MX.sym(['X_' num2str(i+1)], length(xk));
84     w = [w, {Xk}];
85     lbw = [lbw; zeros(par.NT+2, 1)];
86     ubw = [ubw; 0.05; ones(par.NT-1,1); 1; 2850]; %Soft Constraints
87     w0 = [w0; 0.0105;0.0176064;0.027159;0.039794;0.05614;0.0767;0.1017;...
88         0.1309;0.1632;0.1972;0.2309;0.2626;0.2907;0.2941;0.3001;0.3109; ...
89         0.3298;0.36237;0.416;0.4998;0.6179; 0.76389; 0.3322; 2800];
90     %w0-values selected from SS-calculations to improve performance
91
92
93     % For next loop
94     U_last = U_k;   %V, L, F
95
96
97     %Add equality constraint for multiple shooting
98     g = [g, {Xk_end-Xk}];
```

```
    lbg = [lbg; zeros(length(xk),1)];
    ubg = [ubg; zeros(length(xk),1)];
end

prob = struct('f', J, 'x', vertcat(w{:}), 'g', vertcat(g{:}));
solver = nlpsol('solver', 'ipopt', prob);

% Solve the NLP
sol = solver('x0', w0, 'lbx', lbw, 'ubx', ubw,...
             'lbg', lbg, 'ubg', ubg);
u_values = full(sol.x);
u_temp = u_values(25:27); %V, L, F
u_temp = [u_temp; u_temp(1) - u_temp(2); u_temp(2) + u_temp(3) - u_temp(1)];    %++D, B
u_next = [u_temp(2); u_temp(1); u_temp(4); u_temp(5); u_temp(3)]; %Order; L V D B F
% figure(2);
% clear figure
%     clf;
% subplot(2,1,1)
%     plot([1,2,3,4], u_values(1:27:82))
%     ylim([0.0104, 0.0106])
% subplot(2,1,2)
%     stairs([1,2,3,4,5,6,7,8,9], u_values(25:27:242))
%     hold on
%     stairs([1,2,3,4,5,6,7,8,9], u_values(26:27:243))
%     hold on
%     stairs([1,2,3,4,5,6,7,8,9], u_values(27:27:244))
%     legend({'L', 'V', 'F'})
%     title('Input parameters')
%     hold on
 eqend
```

A.10 Plotting script

```
%% Dummyscript for plotting variables
uSim = csvread('uSim.csv');
F0sim = csvread('F0sim.csv');
holdUp = csvread('holdUp.csv');
parSim = csvread('parSim.csv');
timeSim = csvread('timeSim.csv');
xSim = csvread('xSim.csv');
%F0_dist = [par.F0*ones(1,20*dt), par.F0*ones(1,20*dt), par.F0*1*ones(1,20*dt)];
%F0sim = [par.F0*ones(1,20*dt), par.F0*0.8*ones(1,20*dt), par.F0*1.2*ones(1,60*dt)];
% parSim = [0.0105*ones(1,61)]; %xb
% parSim = [parSim; [11.445*ones(1,20), 8.39*ones(1,20), 14.66*ones(1,21)]]; %L
% parSim = [parSim; [17.16*ones(1,20), 11.87*ones(1,20), 23.84*ones(1,21)]]; %V
% parSim = [parSim; [ 5.71*ones(1,20),  3.48*ones(1,20),  8.88*ones(1,21)]]; %D
% parSim = [parSim; [ 7.67*ones(1,20),   6.18*ones(1,20),  8.9*ones(1,21)]]; %B
% parSim = [parSim; [13.38*ones(1,20),  9.66*ones(1,20), 18.08*ones(1,21)]]; %F
% F0_dist = [par.F0*ones(1,21*dt), par.F0*ones(1,20*dt), par.F0*1*ones(1,20*dt)];


figure(1);
 %figSize = [21, 29];              % [width, height]
 %figUnits = 'Centimeters';
 set(gca,'fontsize',10)
 %timeSim = 1:1:81;


clear figure
 clf;

%plot data
subplot(2,1,1), %plot states
    plot(timeSim, xSim(1:end))
    hold on;
    stairs(timeSim, parSim(1, 1:end), '--')
    hold on;
    %plot(timeSim, xid(1:end))

    xlim([0, timeSim(end)])
    ylim([0.0102, 0.0108])%max(xSim) + 0.005])
    grid()
    xticks(0:5:N+40)
```

```
41
42      xlabel('Time [min]')
43      ylabel('xB Composition [molA mol^{-1}]')
44      legend({'x_B: Bottom product composition','x_B: Setpoint'});
45      title('Resulting bottom product composition')
46
47  subplot(2,1,2) %plot inputs and opI ht inputs
48      %plot(timeSim, uSim(1, 1:end), timeSim, uSim(2, 1:end), timeSim, uSim(5, 1:end)); %Also
            gotta plot id-states
49      stairs(timeSim, uSim(1, 1:end))
50      hold on
51      stairs(timeSim, uSim(2, 1:end))
52      hold on
53      stairs(timeSim, uSim(5, 1:end))
54      hold on
55      stairs(timeSim, parSim(2, 1:end), '--')
56      hold on
57      stairs(timeSim, parSim(3, 1:end), '--')
58      hold on
59      stairs(timeSim, parSim(6, 1:end), '--')
60      hold on
61
62      xlim([0, timeSim(end)])
63      ylim([5, 36])
64      grid()
65      xticks(0:5:N+40)
66
67      xlabel('Time [min]')
68      ylabel('Plant inputs U [kmol min^{-1}]')
69      %legend({'L^{RTO} Reflux','V^{RTO} Bottom product', 'F^{RTO} Column Feed'});
70      legend({'L^{NLP} Reflux','V^{NLP} Bottom product', 'F^{NLP} Column Feed', ...
71          'L_{SS}', 'V_{SS}', 'F_{SS}'}, 'NumColumns',2);%, 'Location', 'northeastoutside');
72      title('Optimal Input variables')
73      saveas(gcf, 'INPUTS', 'epsc')
74
75  figure(2);
76  %figSize = [21, 29];            % [width, height]
77  %figUnits = 'Centimeters';
78  set(gca,'fontsize',10)
79  clf;
80  %subplot(1,1,1) %plotting disturbed variable
81      stairs(timeSim, F0sim(1:end));
82      hold on
83
84      xlim([0, timeSim(end)])
85      ylim([6,10])
86      xticks(0:5:N+40)
87      grid()
88      xlabel('Time [min]')
89      ylabel ('Plant Feed (Set value)[kmol min^{-1}]')
90      legend({'F0, Feed'})
91      title('Disturbed variable')
92      saveas(gcf, 'FEED', 'epsc')
93
94  figure(3);
95  %figSize = [21, 29];            % [width, height]
96  %figUnits = 'Centimeters';
97  set(gca,'fontsize',10)
98  clf;
99
100 subplot(2,1,1) %plotting Reactor stuff
101     stairs(timeSim, uSim(3, 1:end));
102     hold on
103     %stairs(timeSim, parSim(4, 1:end), '--');
104     %hold on
105     stairs(timeSim, uSim(5, 1:end));
106     hold on
107     %stairs(timeSim, parSim(6, 1:end), '--');
108     %plot(timeSim, xSim(3, 1:end));
109     %hold only
110
111     xlim([0, timeSim(end)])
112     ylim([0, 20])%max([uSim(3), parSim(4), uSim(5), parSim(6)]) + 10])
```

```
113     xticks(0:5:N+40)
114     grid()
115     xlabel('Time [min]')
116     ylabel ('Reactor values [kmol min^{-1}]')
117
118     legend({'D, Distillate (recycle)', 'F, Reactor output'})
119     %legend({'D, Distillate (recycle)', 'D_{SS}, steady state', ...
120         %'F, Reactor output', 'F_{SS}, steady state'}, 'Location', 'northeastoutside')
121     title('Reactor values')
122 subplot(2,1,2)
123     plot(timeSim, holdUp(1:end));
124     yline(2800, '--');
125     grid()
126     xlabel('Time [min]')
127     ylabel('Reactor hold-up [kmol]')
128     legend({'M_R , Reactor Hold-up [kmol]', 'M_{R,set} , Reactor Hold-up Setpoint [kmol]'})
129     xticks(0:5:N+40)
130     saveas(gcf, 'REACTOR', 'epsc')
131 figure(4);
132 clf;
133 %figSize = [21, 29];              % [width, height]
134 %figUnits = 'Centimeters';
135 subplot(2,1,1)
136     set(gca,'fontsize',10)
137     plot(timeSim, (uSim(1, 1:end) + uSim(5, 1:end) - uSim(2, 1:end)))
138     grid()
139     xlabel('Time [min]')
140     ylim([0,15])
141
142     ylabel('Bottom product flow [kmol min^{-1}]')
143     legend('B, Bottom product flow')
144     title('Resulting column output')
145     xticks(0:5:N+40)
146     yticks(0:1:15)
147     saveas(gcf, 'COL', 'epsc')
```