Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

Kristian Ødegård

Self Optimizing Control of Recirculated Gas-Lift Problem

Master's thesis in Industrial Chemistry and Biotechnology
Supervisor: Sigurd Skogestad
Co-supervisor: Risvan Driza
June 2023

NTNU
Norwegian University of
Science and Technology

Kristian Ødegård

Self Optimizing Control of Recirculated Gas-Lift Problem

Master's thesis in Industrial Chemistry and Biotechnology
Supervisor: Sigurd Skogestad
Co-supervisor: Risvan Driza
June 2023

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

NTNU
Norwegian University of
Science and Technology

Abstract

The oil and gas industry is characterized by complex and dynamic production systems involving multiple wells, riser systems, separators, compressors, and other interconnected components. These systems are subject to various uncertainties, including fluctuating reservoir conditions and changing well dynamics. Developing control strategies tailored to such systems are essential to optimize the production processes, reduce operational costs, and ensure safe and stable operations.

The objective of this study is to use plantwide control design to further develop the oil and gas production system initiated in the author's specialization project [1]. This production system consists of six wells, a riser system, a separator, and a recycled gas lift compressor system. The study focuses on developing control structures that enhance safety and stabilization in the design of the regulatory control layer and self-optimizing control (SOC) structures in the design of the supervisory layer. In 1980, Morari et al. [2] suggested a new approach to optimizing systems by moving the problem to the control layer. Following their idea, we seek to identify controlled variables that, when set to a constant value, will ensure near-optimal performance when the system is disturbed. By doing so, we can remove the need for advanced optimization tools, which are computationally expensive and prone to model error. Various methods were employed to obtain self-optimizing controlled variables using local strategies. These methods were assessed through a Brute force approach and the utilization of a Branch and Bound algorithm across changing active constraint regions. Furthermore, decentralized PI control was utilized to assess the losses from implementing the potential controlled variables. The model developed in this study has integrated a recycled-gas lift system, making it more complex than previous studies of SOC implementation in oil production systems [3] [4] [5]. The complexity increases due to supply pressure being dependent on the performance of the compressor and the pressure of the separator rather than being constant.

The results of this study reveal the challenges of applying local methods to highly non-linear systems. The complexity of the system makes it difficult to accurately determine the optimal measurement combinations. Additionally, trying to control the multiple gas lift chokes proved difficult using decentralized control, which limited the number of unconstrained degrees of freedom evaluated in this study. However, despite these challenges, the study demonstrates that simple approaches often generate adequate results in regions with relatively stable process conditions.

Sammendrag

Olje- og gassindustrien kjennetegnes av komplekse og dynamiske produksjonssystemer som involverer flere brønner, stigerørssystemer, separatorer, kompressorer og andre sammenkoblede komponenter. Disse systemene er underlagt ulike usikkerheter, inkludert variabel reservoarforhold og endringer i brønnenes dynamikk. Utvikling av styringsstrategier skreddersydd for slike systemer er avgjørende for å optimalisere produksjonsprosessen, redusere driftskostnader og sikre trygg og stabil drift.

Målet med denne studien er å bruke anleggsbasert kontroll utforming for å videreutvikle olje- og gassproduksjonssystemet som ble initiert i forfatterens spesialiseringprosjekt [1]. Dette produksjonssystemet består av seks brønner, et stigerørssystem, en separator og et resirkulert gassløft kompressorsystem. Studien fokuserer på å utvikle kontrolsystemer som forbedrer sikkerhet og stabilisering i utformingen av reguleringssjiktet og selvoptimerende kontrollstrukturer i utformingen av overordnet styringssjikt. I 1980 foreslo Morari et al. [2] en ny tilnærming til optimalisering av systemer ved å flytte problemet til kontrollsystemet. I tråd med deres ide, søker vi å identifisere kontrollerte variabler som, når de er satt til en konstant verdi, vil sikre nær-optimal ytelse når systemet forstyrres. Ved å gjøre dette kan vi unngå behovet for avanserte optimaliseringsverktøy som er beregningsmessig dyre og utsatt for modellfeil. Forskjellige metoder ble benyttet for å oppnå selvoptimerende kontrollerte variabler ved hjelp av lokale strategier. Disse metodene ble vurdert gjennom en bruteforce-tilnærming og bruken av en Branch and Bound-algoritme i endrende aktive begrensningssoner. Videre ble desentralisert PI-styring brukt for å vurdere tapene ved implementering av de potensielle kontrollerte variablene. Modellen utviklet i denne studien har integrert et resirkulert gassløftsystem, noe som gjør den mer kompleks enn tidligere studier av selvoptimaliserende kontroll i oljeproduksjonssystemer [3] [4] [5]. Kompleksiteten øker på grunn av at leveringstrykket avhenger av kompressorens ytelse og separatortrykket, i motsetning til å være konstant.

Resultatene av denne studien avdekker utfordringene ved å bruke lokale metoder på sterkt ikke-lineære systemer. Kompleksiteten i systemet gjør det vanskelig å nøyaktig bestemme optimale målekombinasjoner. I tillegg viste det seg vanskelig å kontrollere de flere gassløft-chokeventilene ved hjelp av desentralisert styring, noe som begrenset antallet ubegrensede frihetsgrader som ble evaluert i denne studien. Til tross for disse utfordringene viser studien at enkle tilnærminger ofte gir tilstrekkelige resultater i områder med relativt stabile prosessforhold.

Preface

I would like to express my gratitude to my supervisor Sigurd Skogestad and co-supervisor Risvan Dirza for their invaluable guidance throughout my specialization project and master's thesis. I am especially grateful to Risvan for his constant support and assistance, even during late hours of the night.

Contents

List of Figures

List of Tables

Abbreviations

AD	Algorithmic differentiation
CAS	Computer-algebra system
CV	Controlled variable
DAE	Differential-algebraic equation
GOR	Gas-oil ratio
IPOPT	Interior point optimizer
MV	Measured variable
MPC	Model-predictive control
MPFM	Multiphase Flow Meters
PI	proportional-integral
PID	proportional-integralderivative
RTO	Real-time optimization
SIMC	Simple Internal Method Control
SOC	Self-optimizing control
VPC	Valve Position Control

1 Introduction

Production systems in the oil and gas industry are complex and dynamic, involving numerous interconnected components and processes. The development of effective control strategies is crucial for optimizing system performance, maximizing profit, and ensuring stability and safe operation. In recent years, self-optimizing control (SOC) strategies have experienced increasing attention as a viable alternative to advanced optimization tools such as real-time optimization (RTO) or Dynamic RTO [15].

Previous research [3] [4] [5] has explored the application of SOC strategies in oil and gas production systems, demonstrating their potential to improve system performance and economic viability. However, the majority of studies have focused on relatively simplified models that neglect certain aspects of real production systems. Consequently, the efficacy of SOC strategies in more complex production systems remains largely unexplored. This master thesis seeks to address these unexplored aspects by implementing and testing SOC strategies in a more complex model of an oil and gas production system. The complexity of the model is increased by introducing a re-circulated gas lift system as part of the total produced gas handling. This addition presents new challenges and dynamics that need to be considered in the design and optimization of control strategies.

The motivation behind this research lies in the need to explore the effectiveness of SOC strategies in handling the increased complexity of the production system. By incorporating the re-circulated gas lift system, the model more accurately reflects real-world production scenarios, where such systems are commonly employed to enhance production rates and optimize reservoir recovery. Due to the challenges in sourcing gas lift from other fields during offshore operations, the use of recycled gas lift is favored.

This thesis has three main objectives: (i) design a regulatory control structure that is able to ensure the safety of the operation for the given range of disturbances, (ii) obtain self-optimizing controlled variables (CVs) through plantwide control design and comparison with real production systems, and evaluate the losses from the implementation of the CVs and combinations of these, found by local methods, and (iii) increase the number of disturbances and unconstrained manipulated variables (MVs) and use branch and bound algorithms to obtain the measurement combinations with least average loss.

The work done in this thesis is Based on the idea of moving the optimization problem to the control layer proposed by Morari et al.(1980) [2]. Skogestad et al.(1998) [16] introduced the concept of self-optimizing control as a strategy of achieving near optimal operation by controlling CVs at constant setpoints. Further on, in 2000 Skogestad [6] defined strategies and considerations for obtaining a self-optimizing control structure. Halvorsen et al. (2003) [17] introduced the concept of the optimal self-optimizing control variable as the cost function gradient. This idea was subsequently expanded for large-scale systems Dirza et al. (2022) [18], but practical implementation poses challenges due to limited measurement availability. The issue of finding the self-optimizing controlled variables was initially solved by using brute force methods [6] [19]. The proposed procedure involved analyzing the loss related to controlling the CV at a constant setpoint for all possible disturbances, proving a tedious task for systems with a large amount of candidate CVs. Later, local methods based on evaluating the loss around the nominal point where developed. The local methods are based on the assumptions that the process generally operates around this point, and the methods can thus be used to eliminate potential CVs which results in great loss in the early stages of the design process. [17] [20]. To find the optimal measurement combination of CVs, the nullspace method was proposed by Alstad & Skogestad(2007) [21]. The method propose to find the optimal measurement combination by evaluating the left nullspace of the sensitivity matrix F, which is the gain from the disturbance on each CV. The nullspace method assumes no measurement error, which limits the use to CVs with small potential errors. This can however be overcome by using the exact local method, which takes measurement error into account. [17] [20]. To further improve self-optimizing CV

selection, Branch and Bound methods have been developed Cao et al.(2008,2009,2009) [22] [23] [12]. The Branch and Bound algorithms based on the minimum singular value criterion, worst case loss and miminum average loss obtains the CV combinations which results in the least amount loss based on the criterion. For further information about self-optimizing control principals and development, we refer to Jascke et al.(2017) [15]

The outcomes of this research will hopefully contribute to the understanding of SOC strategies in complex oil and gas production systems and provide insights into their potential for enhancing system performance and their limitations. The findings will be valuable for operators and engineers seeking to optimize production processes, improve oil recovery, and achieve higher operational efficiency in real-world oil and gas production systems.

1.1 Thesis structure

Moving forward, the structure of the thesis will be organized into the following sections.

Section 2 will contain the theoretical background used as a basis for the subsequent sections. The section is initiated by introducing concepts of plantwide control and different advanced control structures, followed by an explanation of self-optimizing control, as well as topics related to the implementation of the different local methods.

Section 3 presents the model used and the development of the regulatory control structures.

Section 4 presents the case studies and the methods used.

Section 5 presents and discusses the results obtained from the simulations and the implementation of methods.

Section 6 discusses the simplifications of the model and the general results.

Section 7 provides a summary of the main findings in the report and their implications.

Section 8 provides recommendations for further work related to the study.

Appendix A presents the results of the simulations of the controlled variables.

Appendix B presents the Python code developed and used in this thesis.

2 Theory

2.1 Hierarchical control

The control structure in a chemical plant is organized into several different *layers* to accommodate the complexity of real systems[19]. The layers operate on distinct timescales, independently of each other, but they are connected through setpoints and CVs, as the higher-level optimization layers determine the setpoints for the lower-level control layers. A more detailed description of the different layers can be found in the list below.

- Scheduling(weeks): Economic models, normally offline and manual, manager.
- Site-wide optimization(day): Real-time optimization(RTO), steady stated models, can be manual or fully automated, process engineer.
- Local optimization(hours): on-line, can be manual or automated, RTO/Operator.
- Supervisory control(minutes): Slow actions, operator/advanced control/MPC
- Regulatory(seconds): Stabilization and fast dynamics, PID controllers.

This hierarchical structure can be observed in Figure 2.1.

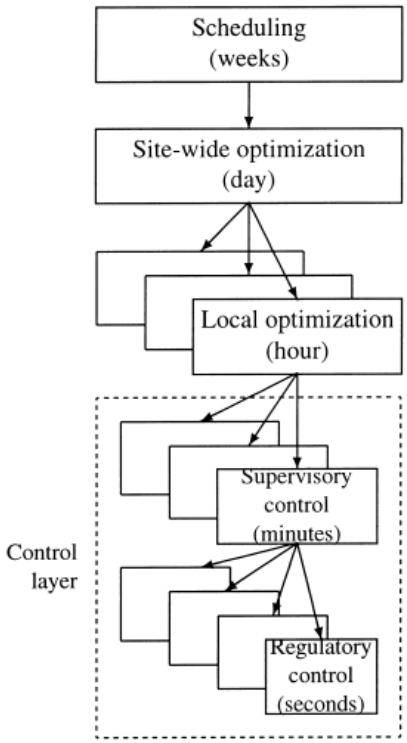


Figure 2.1: Typical control hierarchy[6].

2.2 Real time optimization

Real-time optimization(RTO) is a common method used for optimization of the process inputs. RTO is in the local optimization layer as defined in Section 2.1, and is typically scheduled in the hour range. The RTO based on steady-state detection and or parameter estimation either estimates the different process varibales and disturbances or based on the steady state detection re-optimizes the system to find the optimal inputs for the given process conditions.[24]. For this purpose the RTO needs a steady state model to calculate the most optimal values of the inputs, which leads to

potential error due to imperfect models. There are several disadvantages with the method[25]. The most common challenges of RTO are listed below.

1. Cost related to development and updates of the model
2. The process may not operate at steady state or the settling time is long.
3. Robustness issues, related to computational issues.
4. Frequent grade changes, resulting in steady-state optimization losing it's relevance.
5. Dynamic limitations
6. Incorrect modelling

A flowchart describing tradition RTO-implementation can be observed below.

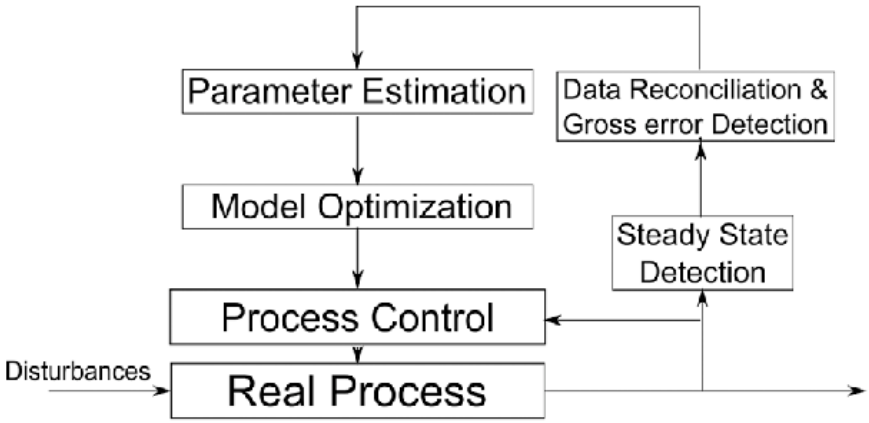


Figure 2.2: Classical RTO-structure. Figure retrieved from[7].

2.3 Plantwide control

Plantwide control handles the structural decisions related to the design of a control system for a chemical plant[6]. Due to the assumption that the different hierarchical layers discussed in Section 2.1 operate on different time scales, we assume immediate implementation of the setpoints calculated by the optimization layers in the control layers. An important question is thus which variable or variables should be controlled? Selecting the CV and its related setpoints is the first step in the procedure for the control structure design problem.[6]:

1. CV and setpoint.
2. MVs.
3. Measurements for control purposes.
4. Control configuration (how the measurements, setpoints, and MVs are connected).
5. Controller type (MPC, PID, decoupler, etc.).

Skogestad[26] proposed a top-down and bottom-up procedure for the control structure design of chemical plants. The focus of the method is to find the best self-optimizing variables for near-optimal operation, and control structure for satisfactory and stable operation.

Top-down analysis

1. Define operational objectives and constraints (scalar cost function for minimization).
2. Identify dynamic and steady-state degrees of freedom.
3. Identify CVs. This includes controlling active constraints and possible CVs for SOC.

4. Define at which point the production rate should be set.

Bottom-up design

1. Regulatory control layer for stabilization and local disturbance rejection.
2. Supervisory control layer, keep variables at optimal setpoints for SOC.
3. Optimization layer, identify active constraints and compute optimal setpoints.
4. Validation with non-linear simulations.

2.4 Self optimizing control

SOC is used to automatically adjust the control parameters of a system to achieve optimal performance in real-time, without needing constant re-optimization. The idea to translate the economic objectives into the control layer was presented by Morari et al.(1980)[2]. The intent was to control a combination of process variables to a constant value, to achieve an optimal adjustment of the MVs, thus resulting in optimal operation. Morari proposed to use simple feedback controllers to achieve this, however, the ideas were somewhat forgotten until Skogestad presented the concept of "SOC" based on Moraris ideas[6]. Skogestad defined the goal of SOC as:

"The goal is to find a set of CVs which, when kept at constant setpoints, indirectly led to near-optimal operating policy." [6]

SOC aims to achieve near-optimal operation by introducing constant setpoints for the CVs. The method eliminates the need for re-optimizing the system when disturbances occur[6]. The problem with different timescales in the control hierarchy can thus be disregarded since the optimization happens in the fast time scale, and the disturbance is dealt with instantly. This differs from RTO, where the time between updating setpoints of the MVs can be hours. Another difference between SOC and RTO is how dependent they are on consistent updates of the model and the re-optimization. While SOC only needs a model for the analysis of the system and implementation, a RTO needs constant updates of the model and computationally costly re-optimizations for each new setpoint calculation.

The implementation of a self-optimizing CV can be analyzed based on the loss compared to the optimal value. This is done by solving the optimization problem for the new disturbance and comparing it to the cost function obtained from controlling our variable to a constant value. The loss function is given by:

$$L = J(u, d) - J_{opt}(d) \tag{2.1}$$

Where L is the difference between the implementation of SOC and the optimal value at the new disturbance, $J(u, d)$ is the cost function value for the SOC case, and $J_{opt}(d)$ is the optimal cost function value at the process conditions. u is the MV and d is the disturbance.

Due to the nature of SOC we expect some loss. Another parameter that can affect the loss is the measurement noise and error. This error will be present in all methods from RTO, MPC, or SOC.

A typical control structure with a measurement combination c_m and measurement noise n can be observed in the figure below.

According to Skogestad et. al (2005)[27], the requirements for good CVs are:

- The CV should be easy to control, that is, the inputs u should have a significant effect (gain) on c.
- The optimal value of c should be insensitive to disturbances.

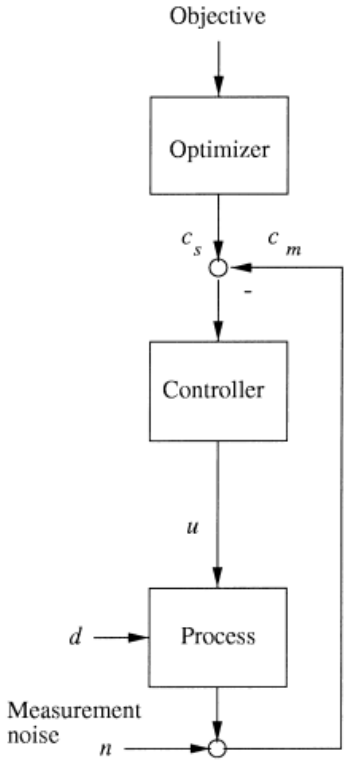


Figure 2.3: Control structure[6].

- The CV should be insensitive to noise.
- In the case of several CVs, the variables should not be closely correlated.

Larsson et al.[28] proposed eight steps to reduce the number of potential CVs for their implementation of SOC on the Tennessee Eastman Process:

Step 1. Eliminate variables with no effect on the economics.

Step 2. Variables directly associated with equality constraints should be controlled.

Step 3. Choose to control active constraints.

Step 4. Eliminate/group closely related variables.

Step 5. Use process insight to eliminate additional variables.

Step 6. Eliminate single variables that, if they had constant setpoints, would yield infeasibility or large losses when disturbances occur.

Step 7. Eliminate combinations of variables that yield infeasibility or large losses.

Step 8. Use local analysis to eliminate variables or combinations that result in a small gain matrix.

After the procedure shown above, the subsequent step is to evaluate disturbance loss and potential implementation loss.

2.5 Nullspace method

The nullspace method is a method developed for the selection of measurement combinations that can be used as CVs for SOC[3]. The method is based on finding a linear combination of the available variables that result in the smallest loss. Equation 2.2 shows how we can find the optimal measurement combination:

$$c = Hy \tag{2.2}$$

where y is a subset of the available measurements, H is a $n_u x n_y$ coefficient matrix and c is the CV that we want to control. The setpoint for c is c_s which corresponds to c at the optimal point. The nullspace method assumes no measurement error, thus the choice of CVs needs to be evaluated carefully. We achieve optimal operation if $c^{opt}(d)$ is independent of the disturbance, which gives Equation 2.3.

$$\Delta c^{opt}(d) = 0 \tag{2.3}$$

To use the Nullspace method we need at least as many measurements as the total number of disturbances and degrees of freedom, as shown in Equation 2.4:

$$n_y \geq n_u + n_d \tag{2.4}$$

where n_y is the number of measurements, n_u is the number of unconstrained degrees of freedom, and n_d is the number of disturbances. We can estimate Δy^{opt} by using the sensitivity matrix F and the disturbance:

$$\Delta y^{opt} = y^{opt}(d) - y^{opt}(d^*) = F(d - d^*) = F\Delta d \tag{2.5}$$

where F is the matrix:

$$F = \left(\frac{dy^{opt}}{dd^T}\right) = \begin{bmatrix} \frac{dy_1^{opt}}{dd_1} & \cdots & \frac{dy_1^{opt}}{dd_{n_d}} \\ \vdots & \ddots & \vdots \\ \frac{dy_{n_y}^{opt}}{dd_1} & \cdots & \frac{dy_{n_y}^{opt}}{dd_{n_d}} \end{bmatrix} \tag{2.6}$$

where n_y is the number of CVs and n_d is the number of disturbances. We can then combine Equation 2.2, 2.3 and 2.5 to find a new expression for c, as shown in Equation 2.7.

$$\Delta c = HF\Delta d = 0 \tag{2.7}$$

We know that Δc should be zero for all disturbances and we thus end up with:

$$HF = 0 \tag{2.8}$$

From Equation 2.8 we can find H by obtaining the left nullspace of F. Which corresponds to the nullspace of F^T.

Figure 2.4 shows a feedback structure related to the measurement combinations.

2.6 Brute Force method

The Brute force method uses continuous evaluations and trials on the total plant to find the best-suited CVs for SOC. The first step of the method is to look at how the different CVs respond to disturbances. Preferably we want a CV that is insensitive to the disturbance. Morari et al. argue that an optimal CV should not be sensitive to any disturbances. Skogestad et al. however, argue that it is necessary that the disturbance affect the CV to a small extent. We must thus evaluate the change in all the CVs to investigate how they are affected by the disturbances.

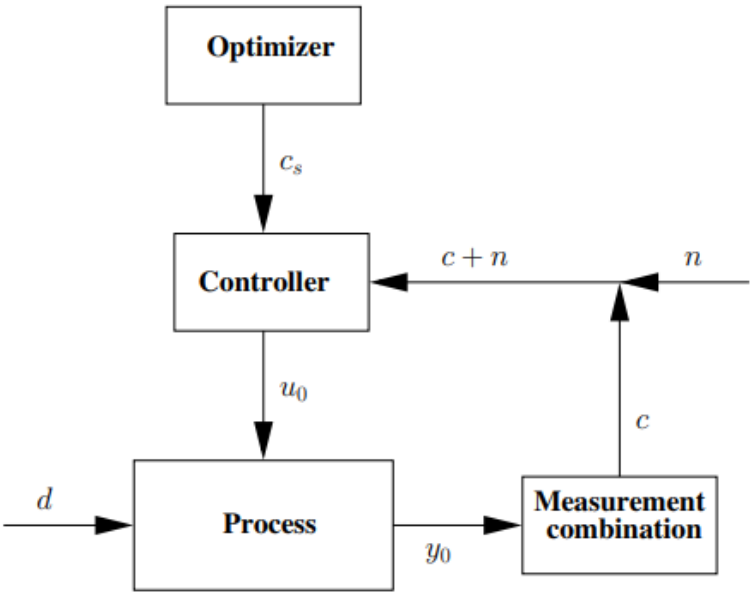


Figure 2.4: Feedback structure with optimizer [3].

The Brute force method is based on evaluating all the potential CVs for all potential disturbances and measurement noise. Thus, the measurement combinations are evaluated by keeping them at their setpoint. Furthermore, the evaluations may be based on either the average loss or worst case loss.

The worst-case loss is calculated as:

$$L_{c_{wc}} = \max_{d\in\mathcal{D},n^y\in\mathcal{N}} L_c \tag{2.9}$$

where $L_{c,wc}$ is the worst case loss, c denotes the particular CV, $\mathcal{D} \subset \mathbb{R}^{n_d}$ contain all disturbances and $\mathcal{N} \subset \mathbb{R}^{n_y}$ contain all noise.

$$L_{c_av} = \mathop{\mathbb{E}}_{d\in\mathcal{D},n^y\in\mathcal{N}}[L_c] \tag{2.10}$$

where $L_{c,av}$ is the average loss and $\mathcal{E}$ is the expectation operator [15]. The brute force method can also be used to find measurement combinations on the form c = Hy. However, the scope of this problem can become huge due to the number of possible combinations that have to be evaluated. The number of possible control structures for the selection of n variables out of m measurements can be calculated from Equation 2.11.

$$C_m^n = \binom{m}{n} = \frac{m!}{(m-n)!n!} \tag{2.11}$$

From this equation, it is clear that when the number of potential CVs increases, the amount of work will become nearly infeasible.

2.7 Valve position control

Valve position control (VPC), also called input resetting, is a technique for controlling one CV with the use of two MVs [29]. The most common area of application of VPC is to improve dynamic

response. Where MV1 has a fast response with limited range, and MV2 is slow with a larger range. The MVs are then configured in parallel where MV1 controls the CV and MV2 controls MV1 to a desired setpoint value, as shown in Figure 2.5. Another use of VPC is to use one of the MVs to extend the steady-state range of the other controller.

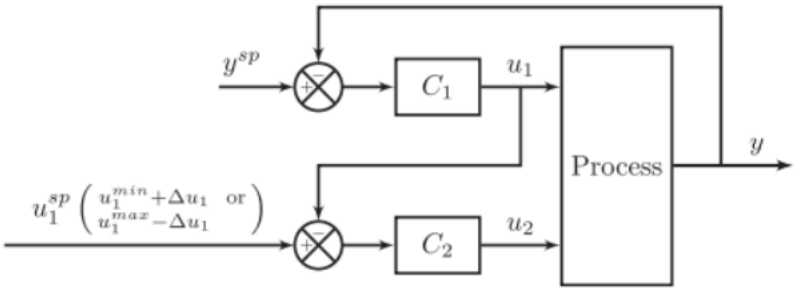


Figure 2.5: Valve position control, figure retrieved from [8].

As we can observed from Figure 2.5, both of the controllers affect the CV, but C_2 will try to control @u_1 to a defined setpoint.

2.8 Selectors

Selectors or override control have been used in the process industry for decades. Selectors are typically used for CV-CV switching, where a number of MVs are used to control a greater number of CVs. The selector typically min/max selectors opens the opportunity to control the most important variable in different active constraint regions. Thus, if CV1 has an active constraint for one disturbance, and CV2 has an active constraint for another disturbance. The selector can decide the CV that should be controlled [30]. Krishnamoorthy et al.(2020) [30] proposed a systematic method for the choice of maximum or minimum selector and the feasibility of implementing a selector. The steps for the design of a selector with one MV and multiple CVs relevant to this thesis are given below.

1. Group the constraints into two sets Y^+(Reduced input favorable for constraint satisfaction) and Y^-(Increased input favorable for constraint satisfaction).
2. Design single input single output(SISO) controllers to compute the input for the CVs.
3. For Y^+ use minimum selector to chose $\overline{u}$ that satisfies the constraint in the set. For Y^- use maximum selector to choose $\underline{u}$ that satisfies the constraint in the set.

Where Y^+ and Y^- can be found by analyzing the response in the CV with a negative and positive change in the MV.

Figure 2.6 shows a typical selector implementation with one MV and two CVs. Based on feedback and the setpoints, each controller calculates a new input to the process. Furthermore, the selector decides, based on configuration, which of the inputs that get implemented. Thus, only one of the inputs can be implemented.

2.9 Split Range control with baton strategy

Split range control can be used when the system has more MVs then CVs. The control method is normally used when there is a need to extend the steady-state control range. For a potential system with two MV's and one CV. The system will switch to MV2 if MV1 saturates. [10] A figure of a typical split range controller can be observed in the figure below.

The split range controller operates with a common controller for all the MVs. The controller settings are found by obtaining the slope α and design the controller parameters for the common

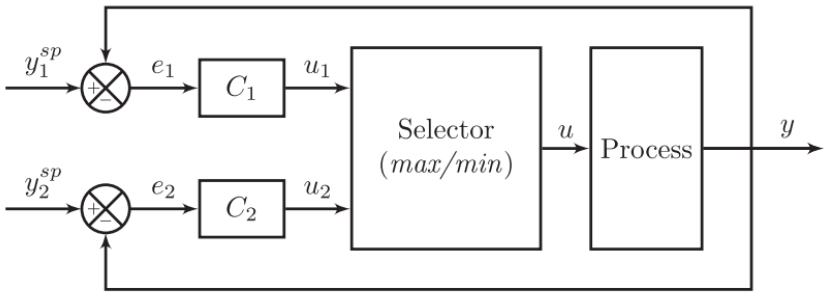


Figure 2.6: Selector block logic, figure retrieved from [9].

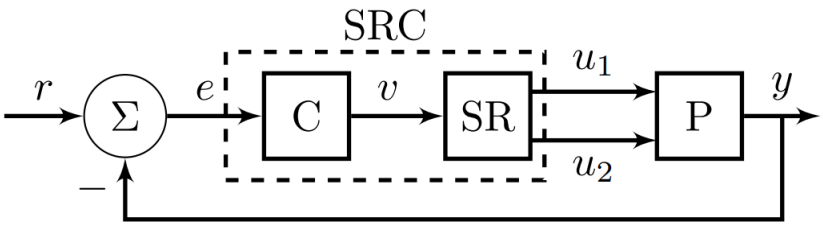


Figure 2.7: Split range control, figure retrieved from [10].

controller. Thus the gain of each MV will be different due to the slope, but the other parameters like the derivative and integral time will be equal. This results in issues relating to tuning of the controllers.

Reyes-Lua et al.(2020) [8] proposed a generalized version of split range control, using the baton strategy. The main difference between the traditional split range controller and the generalized one is that each MV are connected to an individual controller. The resulting change is that each controller can be tuned to match the dynamics of the individual MV. A figure of the baton strategy can be observed below.

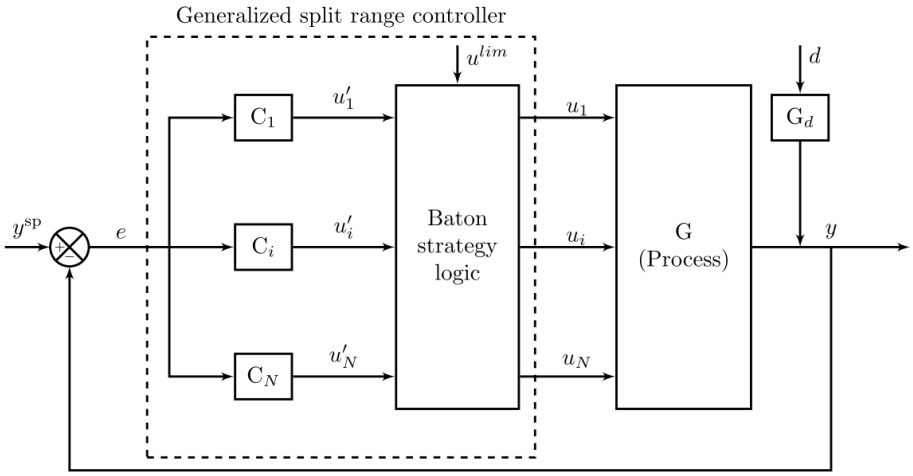


Figure 2.8: Split range control with baton strategy, figure retrieved from [10].

Before Implementing the baton strategy, the sequencing of inputs need to be defined. Reyes-Lua et al.(2020) [8] propose the following steps for choosing the sequence: The first step is to define the maximum and minimum limits for each MV. Furthermore, the inputs should be analyzed by the following criteria:

1. Define the desired operating value for each input.
2. Consider the effect of every input on the output. Then group into:
   (a) Positive gain on output
   (b) Negative gain on output
3. Decide which of the inputs in each group that should be used first based on economics.
4. Test the implementation

After the decision of which active constraints should be active first the baton logic can be implemented. We define i as the the MV that is active(has the baton).

1. The controller of u_i computes the new input u_i^*
2. if $u_i^{min} < u_i^* < u_i^{max}$
   (a) keep u_i active, $u_i \leftarrow u_i^*$
   (b) keep remaining MV's inactive
3. if $u_i^* \leq$ or $u_i^* \geq u_i^{max}$
   (a) set u_i to either it's max or min value depending on limit reached. Furthermore, pass the baton to the next MV j.
   (b) set i = j and start over.

2.10 PID tuning

The proportional-integral-derivative (PID) controller is widely used in industrial applications as one of the most common controller types. The proportional (P) component of the PID controller, known as the P-controller, operates in a manner where the controller's response is directly proportional to the magnitude of the error between the desired and actual values. Equation 2.12 defines the output of the proportional control component.

$$u(t) = K_p e(t), \tag{2.12}$$

where K_p is the proportional gain and $e(t)$ is the error, the difference between the measured CV and the setpoint of the variable. To mitigate steady-state offset resulting from the simplistic nature of the P-controller, the I-controller, which represents the integral (I) component of the controller, is incorporated. By integrating the error signal, the I-controller aims to minimize and correct steady-state deviations between the measured CV and its desired setpoint. This addition enhances the controller's ability to achieve precise and accurate control over time. In Equation 2.13 the integral control output can be observed:

$$u(t) = K_I \int_0^t e(\tau) d\tau, \tag{2.13}$$

where K_I is the integral gain. The D-controller, also known as the derivative (D) component of the controller, utilizes the rate of change of past errors to anticipate and predict the future behavior of the system. By analyzing the temporal variations in the error signal, the D-controller provides valuable insights into the system's dynamics, enabling proactive adjustments and fine-tuning of the control action. This predictive capability enhances the controller's ability to respond swiftly to changing conditions and improves overall system stability and performance. In Equation 2.14 the derivative control component can be observed.

$$u(t) = K_D \frac{de}{dt}, \tag{2.14}$$

where K_D is the derivative gain. The complete PID control output is determined by combining the three individual control outputs: proportional control, integral control, and derivative control. This combination takes into account the contributions of each component to generate the overall control signal that influences the system's behavior. By effectively integrating the proportional, integral, and derivative control actions, the PID controller optimizes the response and stability of the system in order to achieve desired control objectives. The complete PID controller can be observed in Equation 2.15.

$$u(t) = K_p e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de}{dt} \tag{2.15}$$

However, the time constant form is the most commonly employed representation of the PID control output equation, which can be observed in Equation 2.16

$$u(t) = K_p \left(e(t) + \frac{1}{\tau_I} \int_0^t e(\tau)d\tau + \tau_D \frac{de}{dt} \right), \tag{2.16}$$

where τ_D is the integral time and τ_I is the derivative time. From the equation we can observe that the tuning parameters are K_p, τ_I and τ_D.[31]

For the implementation of PID controllers in computers, the controllers need to be discretized. There are several methods of dicretize, however the discrete-time velocity controller eliminates windup in the controllers due to summation of errors are not explicity calculated. The velocity form also has the advantage of the next input being based on the previous, making it suitable for implementation in programming. The discretized velocity form controller can be observed in Equation 2.17

$$u(t_k) = u(t_{k-1}) + Kp\left[e(t_k) - e(t_{k-1})\right] + \frac{K_p h}{T_i} e(t_k) + \frac{KpT_d}{h}\left[e(t_k) - 2e(t_{k-1}) + e(t_{k-2})\right] \tag{2.17}$$

2.11 SIMC method

The SIMC method for controller tuning is systematic approach for tuning PI and PID controllers. As mentioned in the section above, the final controller only has 3 parameters that has to be tuned. However, this can be a time consuming exercise without a proper systematic approach.[11]

The method focuses on obtaining model parameters from either open-loop step response, closed loop setpoint response with P-controller or from detailed model. The first approach and most common in practise is analyzing the CVs response to a step in the MV. A descriptive figure of the approach can be observed in Figure 2.14

From the open-loop step response the dominant lag time constant(τ_1), the plant gain(k) and the effective delay(θ) can be approximated. From this data we can find the initial slope which is given by k' = k/τ_1.

For first order plus delay process which can be approximated as:

$$g_1(s) = \frac{k}{\tau_1 s + 1} e^{-\theta s}, \tag{2.18}$$

We get the following SIMC tuning rules:

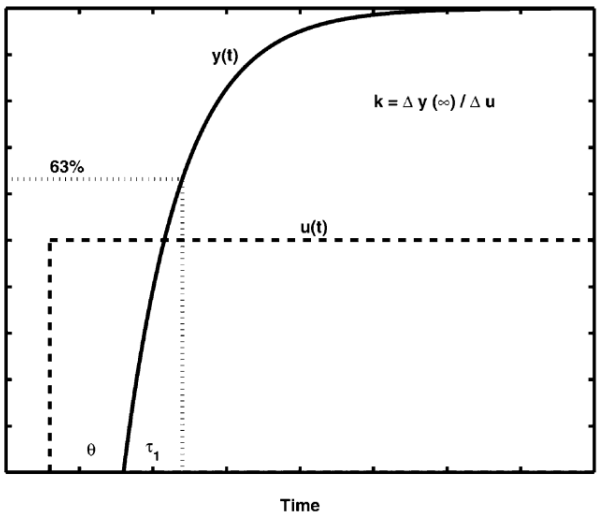


Figure 2.9: Step response of first-order plus time delay process [11].

$$K_c = \frac{1}{k}\frac{\tau_1}{\tau_c + \theta}, \tag{2.19}$$

$$\tau_I = min\{\tau_1, 4(\tau_c + \theta)\}. \tag{2.20}$$

For the second order model plus delay which can be approximated as:

$$g_1(s) = \frac{k}{(\tau_1 s + 1)(\tau_2 s + 1)} e^{-\theta s}, \tag{2.21}$$

We get the following SIMC tuning rules:

$$K_c = \frac{1}{k}\frac{\tau_1}{\tau_c + \theta}, \tag{2.22}$$

$$\tau_I = min\{\tau_1, 4(\tau_c + \theta)\}. \tag{2.23}$$

$$\tau_D = \tau_2 \tag{2.24}$$

2.12 Approximation of loss

The non-linear cost function J is locally approximated using a second-order Taylor expansion around the moving optimal point (u_{opt}, d). For a given disturbance d, this expansion gives J(u, d) as the sum of the cost at the optimal point, the gradient of J at the optimal point dotted with the difference between the current point and the optimal point, the quadratic term involving the Hessian matrix H at the optimal point, and an error term of order O^3.

$$J(u,d) = J(u_{opt}, d) + J_u^T(u - u_{opt}) + \frac{1}{2}(u - u_{opt})^T J_{uu}(u - u_{opt}) + O^3. \tag{2.25}$$

In Equation 2.25 J_{uu} is the Hessian matrix of the cost function and J_u is the Jacobian matrix of the cost function. Due to the second order expansion around the optimum which results in J_u the

Jacobian = 0, we can find the loss by simply deducting J(u_{opt}, d) from each side of Equation 2.25, resulting in,

$$L = J(u, d) - J(u_{\text{opt}}, d)7 \tag{2.26}$$

by defining e = u - u_{opt}, the loss term in Equation 2.26 can be approximated as,

$$L \approx \frac{1}{2} e^T J_{uu} e \tag{2.27}$$

The loss variables is defined as z = $J_{uu}^{\frac{1}{2}}(u - u_{opt})$ resulting in the equation.

$$L = \frac{1}{2} z^T z \tag{2.28}$$

2.13 Exact local method

The exact local method is a local method for evaluating the loss related to a control structure. [17] [20]

The exact local method is derived by using the linear approximation of the plant model. The model is linearized around the optimal nominal operating point. For minor discrepancy from the nominal point we have,

$$\Delta y = G_u^y \Delta u + G_d^y \Delta d + n. \tag{2.29}$$

Where Δy is the deviation in the measurement y compared to the optimal nominal point, Δu is the deviation in the MV u compared to the optimal nominal value, Δd is the deviation in the disturbance d compared to the optimal nominal value, n is the sensor noise and $G_u^y = \frac{\delta y}{\delta u}$ and $G_d^y = \frac{\delta y}{\delta d}$ are the gain from the input to the measurement and the gain from the disturbance to the measurement.

From Equation 2.29 and the optimization matrix H, we can find linear combinations of the measurements by evaluating,

$$\Delta c = H \Delta y \tag{2.30}$$

where c is a measurement combination of the variables in y. From combining Equation 2.30 and Equation 2.29 we get equation,

$$\Delta c = H G_u^y \Delta u + H G_d^y \Delta d + H n. \tag{2.31}$$

To use Equation 2.31 HG_u^y needs to have full rank, corresponding to the number of MVs. [20]

From the assumption of perfect control, resulting in that the measurement combinations Δ c = 0, an expression for Δu can be derived from Equation 2.31 and the expression for the loss,

$$z = -J^{\frac{1}{2}} u u^T (HGy)^{-1} H \left[G_{yd} - G_{yu} J_{uu}^{-1} J_{ud} \Delta d + n \right] \tag{2.32}$$

$$z = -J^{\frac{1}{2}} u u^T (HGy)^{-1} H \left[F \Delta d + n \right] \tag{2.33}$$

In the equation abowe, F is the sensitivity matrix,

$$F = G_{yd} - G_{yu} J_{uu}^{-1} J_{ud}, \tag{2.34}$$

which either can be found from the linearization of the system or by evaluating the optimal change in y by re-optimizing with small changes in the disturbance.

$$F = \left(\frac{dy^{opt}}{dd^T} \right). \tag{2.35}$$

The disturbance and noise are scaled by the use of the scaling matrices W_d and W_n. From the previous equations, the scaling matrices are defined as.

$$\Delta d = W_d d', \tag{2.36}$$

$$n = W_n n', \tag{2.37}$$

where n' and d' are scaled noise and disturbance vectors. When implementing the scaled terms into Equation 2.33 and 2.28 the expression for the loss becomes,

$$L = \frac{1}{2} \left\| J^{\frac{1}{2}} u u^T (HGy)^{-1} H \tilde{F} \begin{bmatrix} d' \\ n' \end{bmatrix} \right\|_2^2 \tag{2.38}$$

Where the term is calculated using the second matrix norm. For simplification of the loss term found in Equation 2.38, matrix M can be introduced,

$$M = J^{\frac{1}{2}} u u^T (HGy)^{-1} H \tilde{F}, \tag{2.39}$$

resulting in the a new expression for the loss defined as,

$$L = \frac{1}{2} \left\| M \begin{bmatrix} d' \\ n' \end{bmatrix} \right\|_2^2. \tag{2.40}$$

In the expression in Equation 2.38,$\tilde{F}$ or Y as it may be defined is represented by.

$$\tilde{F} \triangleq \begin{bmatrix} F W_d & W_n \end{bmatrix} \tag{2.41}$$

2.14 Local loss for normally distributed noise and disturbance

There have been developed several average and worst-case loss methods for different assumptions. In this paper we assume that the noise and disturbance are uniformly distributed over the set. Halvorsen et al.(2003) [17] showed that when the noise and disturbance are uniformly distributed over the set,

$$\mathcal{DN}_2 = \{(d', n') | \parallel \begin{bmatrix} d' & n' \end{bmatrix}^T \parallel_2 \leq 1\}, \tag{2.42}$$

the worst case loss can be calculated by,

$$L_{wc} = \max_{\left\| \begin{bmatrix} d' \\ n' \end{bmatrix} \right\|_2 \leq 1} = L = \frac{1}{2} \left\| M \begin{bmatrix} d' \\ n' \end{bmatrix} \right\|_2^2 = L = \frac{1}{2} \|M\|_2^2 = \frac{1}{2}\overline{\sigma}^2(M), \tag{2.43}$$

where $\overline{\sigma}^2$ is the largest singular value.

Kariwala et al.(2008)[32] showed that the average loss when the noise and disturbance are normally distributed with no unit and mean variance,

$$\mathcal{DN}_2 = \{d' \sim \mathcal{N}(0, I), n' \sim \mathcal{N}(0, I)\} \tag{2.44}$$

becomes,

$$L_{avg} = \mathop{\mathbb{E}}_{d',n' \in \mathcal{DN}_{\mathcal{N}}} = \frac{1}{2} \left[M \begin{bmatrix} d' \\ n' \end{bmatrix} \right] = \frac{1}{2} \|M\|_F^2 \tag{2.45}$$

2.15 Method for minimum loss

For the implementation of the CV c = Hy, which is a optimal linear combination of the available measurements y. The previous terms of loss in Section 2.14 can be used to find H by minimizing the loss terms. In the case of normally distributed noise and disturbance, H can be found from minimizing the average loss found in Equation 2.45.

$$H = arg\min_H \frac{1}{2} \|M\|_F^2 = arg\min_H \frac{1}{2} \left\| J_{uu}^{\frac{1}{2}} (HGy)^{-1} H\tilde{F} \right\|_F^2 \tag{2.46}$$

The resulting expression is a non-convex optimization problem often difficult to solve. However, Kariwala et al.(2008)[32] proposed to exploit the non-unique properties in selecting H. Thus, re-defining the problem from non-convex to a convex optimization problem. The formulation can be found below.

$$\begin{aligned} &\min_H \|H\tilde{F}\|_F \\ &\text{s.t.} \quad HG_{yu} = J_{uu}^{\frac{1}{2}}. \end{aligned} \tag{2.47}$$

To redefine the problem from non-convex to convex, it is proposed to cancel the non-linearity in M by using a non-singular Matrix Q resulting in $HG_{yu} = J_{uu}^{\frac{1}{2}}$. Jascke et al.(2017)[15] stated that the M matrix remains constant when multiplied by any non-singular matrix, and will thus not be effected.

Yelchuru and Skogestad(2012)[33] proposed an analytical solution to the problem,

$$H^T = G_{yu}(\tilde{F}\tilde{F}^T)^{-1}. \tag{2.48}$$

For further reference to the analytical solution in this paper $\tilde{F}$ is changed to Y, the H matrix can thus be found by.

$$H^T = G_{yu}(YY^T)^{-1}. \tag{2.49}$$

2.16 Branch and Bound

The branch and bound method(BAB) is an algorithmic technique used to solve optimization problems. The typical use of the algorithm is when the problem involves searching through a large number of possible solutions for combinatorial optimization problems. There have been developed several BAB methods, with the most common being the upwards and downwards pruning methods depicted in Figure 2.10

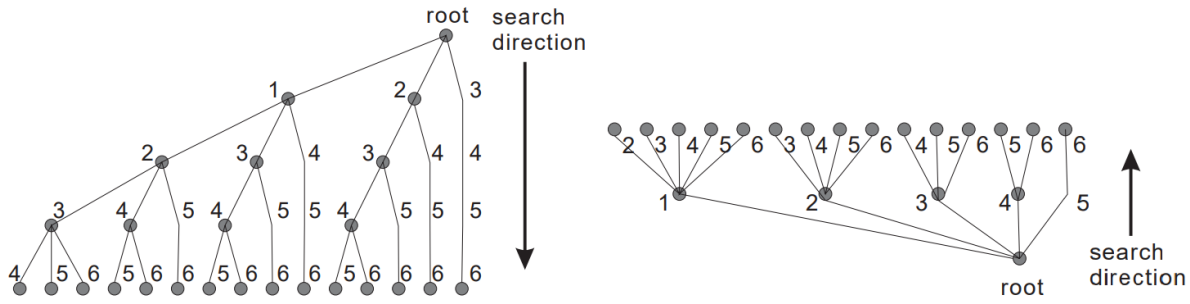


Figure 2.10: Figure depicting upwards and downwards pruning in BAB, figure retrieved from [12].

The figure depicts an example for the application of downwards and upwards BAB for the selection of a subset of 2 elements out of 6 potential measurements.[12] The figure on the left depicts the downwards BAB method. Here, the root consists of all the available measurements in this example 6. Starting from the top each node represents a subset obtained by removing a measurement from the set of all the available measurements. In this example, the number related to each node represents the measurement removed from the total set. The figure on the right depicts the upward BAB method. Compared to the downward method, the set contains no measurements at the start of the implementation. The number related to each node in this case represents the measurement added to the total set. The BAB method work by evaluating the nodes, and pruning(dischard) the branches related to measurement sets that do not lead to a lower optimal loss. The two BAB methods can also be combined, resulting in a Bidirectional BAB method. In this method both upward and downward branching and pruning are used for increased efficiency. In Figure 2.11 an example of bidirectional branching can in the upwards and downward BAB methods.

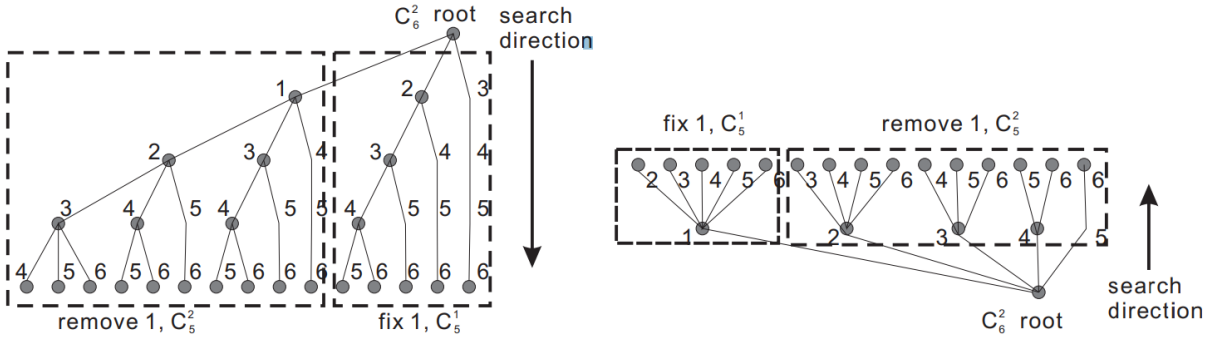


Figure 2.11: Figure depicting binary branching in upwards and downwards BAB, figure retrieved from [12].

In Figure 2.12 an example of binary branching can be observed. In the right node starting from the we can observe that the branches correspond to the right side of the binary tree in figure in the downwards BAB method in 2.11 , while at the left initial node the branching corresponds with the right strategy for the upwards BAB 2.11. This feature can be taken advantage of to increase the efficiency of the method. The method can then choose either the right path or the left path based on which is more efficient.[12].

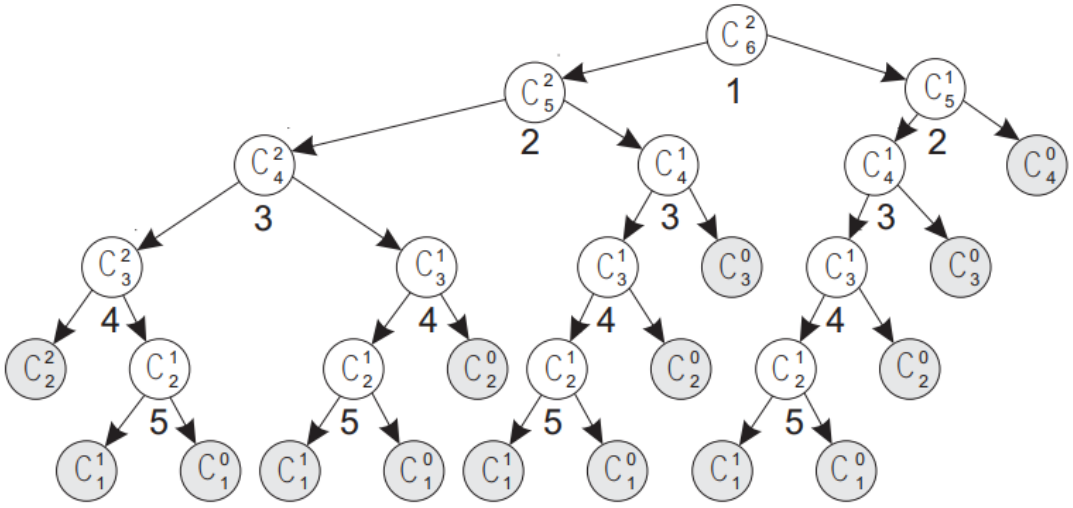


Figure 2.12: Figure depicting bidirectional branching, figure retrieved from [12].

Cao et al.(2009) [22] developed an algorithm for the worst case loss in measurement selection based on the Exact local method. Before further including the evaluation of average loss. [23] For more information about the methods we reefer to these papers. The related matlab files for the methods, which where used in this paper can be found in. [34] [35]

2.17 Active constraint region

Due to the nature of operation and possible disturbances and other effects on the process, different constraints can become active or deactivate. Maarleveld et al.(1970) [36] proposed to control the optimal active constraints. Which from a self optimizing view is logical since the active constraint variable doesn't change with the disturbance. However, when the operation leaves the region where the constraint becomes inactive, a new control structure and a change in CVs needs to be implemented. [13] In practise this means that a self optimizing strategy needs to be implemented separately for the different constraint regions. The change in CVs and constraint control can be implemented with selectors discussed in Section 2.8.

In Figure 2.13 the cost function is depicted for disturbance d_1 and d_2. It can be observed that for disturbance d_1 g is active and for disturbance d_2 g is inactive.

2.18 Anti-windup

Windup in PID control refers to a phenomenon that occurs when the integral term of a PID controller continues to accumulate error even when the system is saturated or unable to respond to the controller's output effectively. This can lead to overshoot, prolonged settling time, and instability in the controlled system.

When a PID controller is unable to achieve the desired control action due to physical limitations, such as a maximum or minimum output constraint, the integral term can keep integrating the error, causing an excessive buildup of the integral term. This accumulated integral term, also known as integrator windup, can result in a significant overshoot or prolonged settling time once the constraints are lifted or the system becomes capable of responding to the controller output.

To mitigate windup, various anti-windup techniques are employed. These techniques aim to limit or reset the integral term when the controller output saturates, preventing excessive integral accumulation. Some common anti-windup methods include back-calculation, clamping, conditional

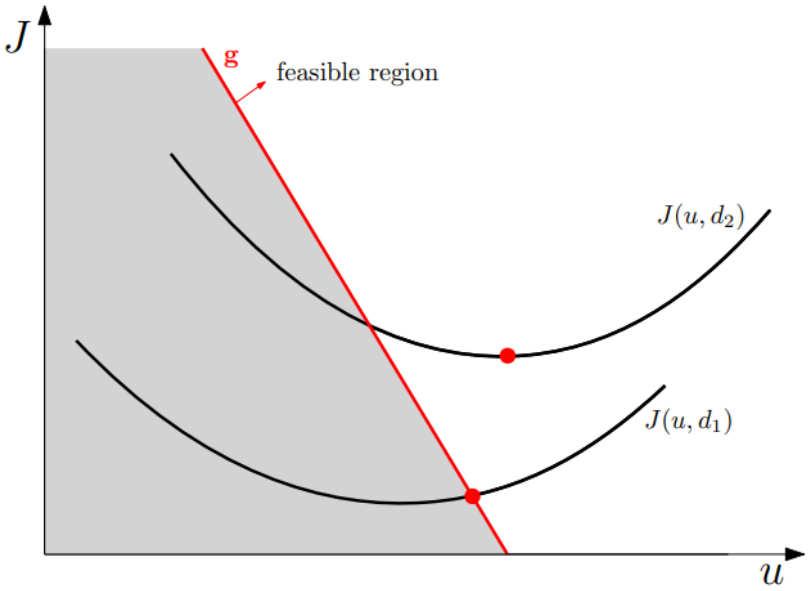


Figure 2.13: Figure depicting how change in disturbance changes cost function(J) and avtive constraint[13].

integration, and gain scheduling.[37]

2.19 Finite difference

Finite difference methods are applicable for estimating derivatives of functions at specific points. Usually, when it comes to approximating analytical solutions to differential equations, either a continuous function or a discrete function is employed. This function, defined within a designated region of space and/or time, fulfills the boundary conditions specified for that particular region. The method is used because analytical solutions to differential equations can be hard to obtain. The method works by reformulating the derivative terms in the differential equation with finite difference approximations.[38]

For the differentiation of one variable the derivative can be approximated as,

$$f'(x) = \lim_{h \to 0} = \frac{f(x+h) - f(x)}{h} \tag{2.50}$$

Where f'(x) is the derivative of the function, f(x + h) is the function value at x plus a small h and f(x) is the function value at the nominal point.

For first-order differentiation of one variable with multiple variables the differential equation can be approximated as.

$$f_x(x, y) \approx \frac{f(x+h, y) - f(x-h, y)}{2h} \tag{2.51}$$

For second-order differentiation of one variable with multiple variables the differential equation can be approximated as.

$$f_{xx}(x, y) \approx \frac{f(x+h, y) - 2f(x, y) + f(x-h, y)}{h^2} \tag{2.52}$$

For second-order differentiation of two variables the differential equation can be approximated as.

$$f_{xy}(x,y) \approx \frac{f(x+h,y+k) - f(x+h,y) - f(x,y+k) + 2f(x,y) - f(x-h,y) - f(x,y-k) + f(x-h,y-k)}{2hk} \tag{2.53}$$

Where h and k are small values added to each variable to perturb the nomainal point.

2.20 Oil and gas operation/GOR effect

The reservoir is the most critical component of an oil production system. Reservoirs are categorized as water-drive, gas-cap drive, or dissolved gas drive reservoirs. In water-drive reservoirs, a decrease in pressure causes the expansion and influx of groundwater into the reservoir, pushing oil and gas to the reservoir's upper portion. If the production rate is kept constant, this type of reservoir maintains its pressure for a longer duration compared to other reservoir types.

In gas-cap drive reservoirs, the gas separates from the oil/gas mixture and accumulates at the top of the reservoir. If the gas in the cap is extracted too quickly, the reservoir pressure experiences a significant drop, thereby reducing the production potential.

In dissolved gas drive reservoirs, the gas remains in a dissolved state within the oil, forming a liquid phase. Early pressure maintenance is often necessary in these reservoirs due to the possibility of two-phase flow formation caused by pressure decline.[39]

In order to extract oil and gas from petroleum reservoirs, wells must be drilled to create pathways for extraction. These production wells are comprised of several components including packers, a production pipe (tubing), casings, and a wellhead equipped with multiple chokes. The packers serve to isolate the annulus at the bottom of the tubing, directing the produced fluid to escape through the perforations and into the lower part of the well. The tubing is responsible for transporting the oil and gas to the surface. Casings are pipes that provide structural support to the well. The wellhead incorporates various chokes, which are employed to regulate the flow from the well. The primary choke used for flow control is referred to as the production choke, which can be adjusted to modify the flow rate. By closing the production choke, the bottom-hole flowing pressure increases, resulting in a reduced pressure difference between the reservoir and the bottom-hole of the well, consequently leading to a decrease in production rate.

Wells can be classified based on their Gas-Oil Ratio (GOR), which quantifies the relationship between oil and gas in the produced fluid. Another classification method involves examining their productivity index, which establishes a relationship between the liquid flow within the well and the pressure difference between the reservoir and the bottom-hole[39]. Wellheads can be positioned either subsea or at topside production facilities. In the case of a subsea wellhead, a riser which is a pipe section can be utilized for transporting the fluid to the topside production facilities. Once at the topside, the production fluid is conveyed to the inlet separator.

Typically, the production fluid in oil wells comprises various compounds, primarily hydrocarbons in both gas and liquid phases, along with water and solids. The flow of production fluid is often turbulent, characterized by irregular movement of the liquid. To separate the different components in the production fluid for further processing, separators are employed. The horizontal separator is the most commonly used type due to its versatility and cost-effectiveness. It separates the components based on their density differences and the force of gravity[39].

Artificial lift refers to methods employed to enhance oil production from wells. One widely used artificial lift method is gas lift. This method involves injecting gas into the annulus of a well. The injected gas travels to an injection valve located in the lower sections of the well and enters the tubing. The gas affects the production fluid by reducing its density, consequently lowering the hydrostatic pressure and increasing the flow rate. Additionally, the injected gas exerts an upward force on the production fluid due to expansion effects[39]. The gas lift system utilizes produced

gas as the injection gas and relies on a compressor system to recompress the produced gas before it can be used for injection. The gas lift system also requires a gas lift manifold with piping and chokes connected to the relevant wells, as well as an injection valve at the bottom of the annulus. According to Hu (2004) [40], gas lift increases the production of a well until the hydrostatic pressure drop can no longer compensate for the increased friction caused by the higher gas mass in the tubing. Injecting more gas beyond this point will actually decrease the oil production.

Gas lift often utilizes a portion of the produced gas. The pressure loss from the reservoir to the production facilities necessitates recompression of the gas. Compressors are commonly employed for this purpose in most production facilities. Various types of compressors are available for meeting this requirement.

2.21 Surge control in compressors

In compressor operation there is an inherent risk of potential damage to the compressor. One major risk factor is the surge phenomenon. Surge can be described as when the gasflow through the compressor is to low relative to the size of the compressor. The pressure ratio will increase until the discharge pressure exceeds the suction pressure.Thus, the compressor becomes unable to deliver energy to the process fluid resulting in reversed flow through the compressor. [41] Due to the risk and potential damage, different anti-surge measures are implemented in most compressor systems. One of these measures is designing the compressors with recycle valves, that can counteract the insufficient flow through the compressor. The valves are implemented with fast-acting controllers, typically measuring the surge flow, that can react fast if the flow closes in on the surge limit. An important observation is that the recycle valves should be closed during normal operation due to economic reasons(cost of compression). Below a typical compressor curve with surge constraint can be observed.

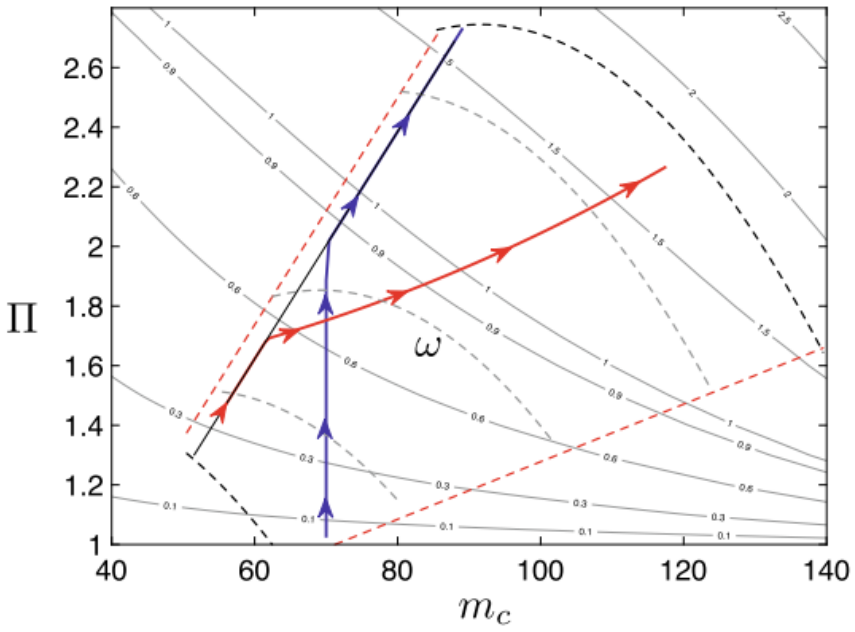


Figure 2.14: Compressor curve, pressure ratio vs massflow [14].

The red dashed lines shows the surge and choke lines, the black dashed lines shows the upper and lower bounds on the speed, the grey dashed lines shows operating points for the implementation of three constant speeds. While the red and blue lines shows operating points for parallel and serial configuration of compressors. [14].

Milosavljevic et al.(2020) proposed to the following condition for modelling the surge in a compressor

model that the model in this work is based on.

$$G_{1,i} = \frac{1}{s_{1,i}}(s_{0,i} + \Pi_i) - m_{c,i} - s_{2,i} \leq 0 \tag{2.54}$$

where $s_{0,i}$, $s_{1,i}$ and $s_{2,i}$ are surge line coefficients for compressor i, which are decision variables. Π_i is the pressure ratio of compressor i and $m_{c,i}$ is the massflow through compressor i.

2.22 Casadi - numerical solver

CasADI, an open-source software framework, is designed for numerical optimization and was initially developed as a tool for algorithmic differentiation (AD) using a computer-algebra system (CAS) syntax. The project was initiated by Joel Andersson and Joris Gillis in 2018.

Built on a symbolic framework, CasADI treats variables as symbolic values and represents them as matrices. This framework enables the solution of various optimization problems associated with optimal control. It provides users with a toolkit that facilitates the implementation of optimization problems, minimizing both the required effort and any potential loss of performance.

CasADI supports a wide range of optimization algorithms, including gradient-based methods, nonlinear programming solvers, and mixed-integer programming solvers. It employs efficient automatic differentiation techniques to compute derivatives, which are crucial for gradient-dependent optimization algorithms.

By utilizing a symbolic approach, CasADI offers an intuitive and concise means of expressing optimization problems. It incorporates an extensive set of mathematical operations and functions, enabling the modeling of complex systems and the formulation of sophisticated optimization objectives and constraints.

One notable feature of CasADI is its ability to generate highly efficient numerical code for optimization problems. It can produce code in multiple programming languages such as C, C++, and Python, allowing seamless integration with existing codebases and leveraging the performance advantages of compiled languages.

CasADI has gained significant popularity among researchers and practitioners in the field of numerical optimization due to its user-friendly nature, efficiency, and extensibility. It has found applications in various domains, including robotics, control systems, machine learning, and energy optimization.

In summary, CasADI is a powerful open-source software framework that provides a comprehensive toolkit for numerical optimization. It simplifies the implementation of optimization problems, offers efficient algorithms, and supports code generation for high-performance computations.[42]

3 Modelling and control

3.1 Model

The model used in this paper is based on previous work performed in the author's specialization project [1], with some additional modifications and new implementations. In this section, the model basis from the specialization project will be described briefly, while the new modifications and implementations will be described in more detail.

The model employed in this study comprises a system of 6 wells, each connected to a distinct reservoir. The wells are linked to a shared subsea production manifold through individual production chokes, as depicted in Figure 3.1.

The fluids produced by the six wells are routed from the manifold to the production platform by a vertical riser. Upon reaching the surface, the production fluid undergoes separation in an inlet separator, which separates the oil from the gas. The oil and the gas depart the separator in the bottom and top sections respectively. The oil is then routed downstream for further processing, which is beyond the scope of this study. The gas is either routed to export for further processing or to be used for gas lift. As with further processing of the oil, the processing of the export gas is outside of the scope. The gas designated to be used for the gas lift is routed to a three-stage compressor train designed in a serial configuration. The discharge gas from the compressor train is transported from the surface facilities to the shared subsea gas lift manifold. In the gas lift manifold, the gas is distributed to the wells based on demand.

For a more detailed description of the overall model, see "Modelling and optimization of recirculated gas lift problem" (2022) [1]. The total production system and the scope of this work can be observed in Figure 3.1.

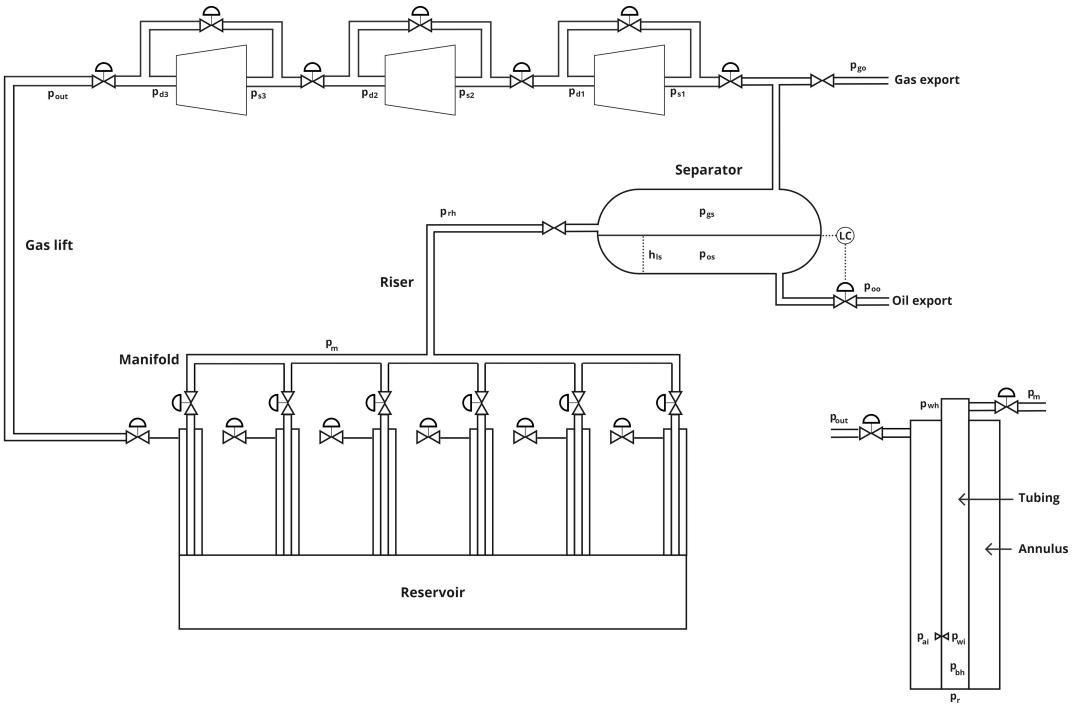


Figure 3.1: Total case model.

3.1.1 Objective

Upon development of the complete model, the final non-linear optimization problem was formulated. The objective function was designed to maximize the the total oil production, specifically

the oil routed to export(w_{os}), while minimizing the cost related to power consumption in the compressors(P_{ci}). The variables within the objective function are multiplied by factors representing the price of oil ($/kg/s) and the price of power ($/kW).

The optimization problem is subject to several constraints, including the amount of produced gas (10 kg/s), the total gas utilized for gas lift (8 kg/s), and the total power consumption (18 kW). These constraints are linked to the capacity limitations of the system. Additionally, the model equations and the upper and lower boundaries of the states and control variables serve as further constraints. Equation 3.1 provides a representation of the objective function and the constraints.

$$\begin{aligned} \min_{\theta} \quad & -0.6w_{os} + 0.03\sum_{i=1}^{3} P_{c_i} \\ \text{s.t.} \quad & g(\theta) = 0 \\ & f(\theta) = 0 \\ & w_{gs} - w_{gs}^{max} \leq 0 \\ & \sum_{i=1}^{6} w_{gl_i} - w_{gl}^{max} \leq 0 \\ & \sum_{i=1}^{3} P_{C_i} - P_{C}^{max} \leq 0 \\ & x^L \leq x \leq x^U \\ & u^L \leq u \leq u^U \\ & z^L \leq z \leq z^U \end{aligned} \tag{3.1}$$

where θ represents the model states and controlled variables. $g(\theta)$ are the algebraic equations of the model, $f(\theta)$ are the differential equations of the model, w_{gs}^{max} is the maximum produced gas, w_{gl}^{max} is the maximum gas lift, P_C^{max} is the maximum power consumption, x^L and x^U are the lower and upper bounds of the differential states, z^L and z^U are the lower and upper bounds of the algebraic states and u^L and u^U are the lower and upper bounds of the manipulated variables. For information about the upper and lower bounds on the states, the reader is refereed to [1].

3.1.2 Nominal point

At the optimal nominal point we find from optimization that the value of w_{gs} is 10 kg/s and is thus and active constraint. However, the total power is found to be 13.0863 KW, thus not active and the total gas lift flow 3.67915 kg/s, thus not active.

3.1.3 Well system with gas lift

As mentioned in 3.1 the well system comprises of 6 wells, with distinct reservoirs. Consequently, any modifications to the parameters within a specific reservoir will solely effect the reservoir parameters in the corresponding well.

The wells are produced due to the pressure differential between the reservoir and bottom section of the well. To facilitate production flow from the reservoir, gas lift is introduced in the lower sections of the well via and injection valve that connects the annular section to the tubing. The gas lift severs to descrease the density of the production fluid, subsequently reducing the bottomhole pressure increasing the flow from the reservoir.

The production fluid eventually reaches the wellhead at the seabed, where the overall flow is determined by the difference in pressure over the production choke and the corresponding choke opening.

The model equations employed are based on differential-algebraic equations (DAE), where the mass within each compartment is calculated based on the differential flow of mass into and out of the compartment. The remaining well variables are calculated algebraically.

A summary of the variables in the well system can be found below for well i, with their related equations found in Appendix B.10 or in [1].

$$x_{well} = [m_{ga_i}, m_{gt_i}, m_{ot_i}]^T \tag{3.2a}$$
$$z_{well} = [p_{ai_i}, p_{wh_i}, p_{wi_i}, p_{bh_i}, \rho_{ai_i}, \rho_{m_i}, w_{iv_i}, w_{pc_i}, w_{pg_i}, w_{po_i}, w_{ro_i}, w_{rg_i}]^T \tag{3.2b}$$
$$p_{well} = [GOR_i, p_{res_i}]^T \tag{3.2c}$$
$$u_{well} = [u_{pc_i}, u_{gl_i}]^T \tag{3.2d}$$

Where x are differential states, z are algebraic states, p are constant parameters and u are manipulated variables. Further on, m_{ga} is the mass rate of gas in the annulus, m_{gt} is the mass rate of gas in the tubing, m_{ot} is the mass rate of oil in the tubing, p_{ai} is the annulus pressure at the injection point, p_{wh} is the wellhead pressure, p_{wi} is the tubing pressure at the gas lift injection point, p_{bh} is the bottomhole pressure, ρ_{ai} is the annulus density, ρ_m is the mixed oil and gas density in the tubing, w_{iv} is the flow through the injection valve, w_{pc} is the flow through the production choke, w_{pg} is the flow of gas through the production choke, w_{po} is the flow of oil through the production choke, w_{rg} is the flow of gas from the reservoir, w_{ro} is the flow of oil from the reservoir, GOR is the gas oil ratio, p_{res} is the reservoir pressure, u_{pc} is the opening of the production choke and u_{gl} is the opening of the gas lift choke.

3.1.4 Riser and manifold system

Production fluid from the 6 wells meet at a common manifold and mix. . Subsequently, the oil and gas are transported from the subsea facilities to the production facilities located above sea level. The transportation process through the riser entails a pressure drop, primarily influenced by friction within the pipes and variations in elevation.

A summary of the variables in the riser and manifold system can be found below, with their related equations found in Appendix B.10 or in [1].

$$x_{riser} = [m_{or}, m_{gr}]^T \tag{3.3a}$$
$$z_{riser} = [p_{rh}, \rho_r, p_m, w_{pr}, w_{to}, w_{tg}]^T \tag{3.3b}$$

Where m_{or} is the mass rate of oil in the riser, Where m_{gr} is the mass rate of gas in the riser, Where p_{rh} is the riserhead pressure, ρ_r is the riser density, Where p_m is the manifold pressure, w_{pr} is the total flow in the riser, w_{to} is the flow of oil in the riser and w_{tg} is the flow of gas in the riser.

3.1.5 Separator system

The separator is responsible for efficiently separating the oil and gas components. When the production fluid enters the separator, the liquid and gas phases are segregated based on the density disparities of the hydrocarbons.

In typical oil and gas production scenarios, the production fluids often contain water. However, in this project, the treatment cost associated with water is not considered. Additionally, an assumption of perfect separation is made, implying that the inclusion of water would not affect the results unless it reduces the oil content in specific wells.

A summary of the process variables related to the separator system can be found below, with their related equations found in Appendix B.10 or in [1].

$$x_{sep} = [p_{gs}, h_{ls}]^T \tag{3.4a}$$
$$z_{sep} = [w_{os}, w_{gs}, \rho_{gs}, p_{os}, V_{os}, V_{gs}]^T \tag{3.4b}$$
$$p_{sep} = [p_{oo}, p_{go}]^T \tag{3.4c}$$
$$u_{sep} = [u_{os}]^T \tag{3.4d}$$

Where p_{gs} is the pressure of the gas in the separator, Where h_{ls} is the oil level, w_{os} is the flow of oil out of the separator, w_{gs} is the flow of gas out of the separator, ρ_{gs} is the density of gas, p_{os} is the pressure of the oil, V_{gs} is the volume of gas, V_{os} is the volume of oil, p_{oo} is the pressure at the oil export, p_{go} is the pressure at the gas export and u_{os} is the opening of the oil choke.

3.1.6 Compressor system

In order to utilize the produced gas for gas lift purposes, it is necessary to recompress the gas. We suggest implementing a compressor system consisting of three centrifugal compressors arranged in series to handle the gas lift operation.

The series configuration is chosen due to the inherent limitations of each individual compressor. Centrifugal compressors typically have a pressure ratio in the range of 2-2.5, which would not be adequate for the intended implementation. The valves positioned between the compressors are considered as pressure drop elements and are not regarded as manipulated variables. Additionally, recycle valves are incorporated in the system to control the surge constraint.

A summary of the process variables in the compressor train for compressor i can be found below, with their related equations found in Appendix B.10 or in [1].

$$x_{comp} = [p_{s_i}, p_{d_i}, m_{c_i}]^T \tag{3.5a}$$
$$z_{comp} = [w_{in_i}, w_{out_i}, \rho_{in_i}, \rho_{d_i}, \Pi_i, P_{c_i}, y_{p_i}, \eta_{p_i}, w_{rec}]^T \tag{3.5b}$$
$$u_{comp} = [u_{rec_i}]^T \tag{3.5c}$$

Where p_s is the suction pressure of the compressor, p_d is the discharge pressure of the compressor, m_c is the flow through the compressor, w_{in} is the flow of gas in to the compressor, w_{out} is the flow of gas out of the compressor, ρ_d is the density of gas at the discharge, Π is the the pressure ratio over the compressor, P_c is the power usage, y_{p_i} is the polytropic head, η_p is the efficiency, w_{rec} is the flow through the recycle line and u_{rec} is choke opening of the recycle valve.

3.2 Control Implementations

Several modifications on the previous model presented in Section 3.1 where implemented on the new model. The main modifications was related to the unimplemented control structures and modifications for the specific case study. The implementation of the regulatory control structures proposed in this thesis are described in the upcoming sections.

3.2.1 Implementation of surge control

As explained in the theoretical Section 2.21, surge refers to a condition where the gas flow through the compressor reaches a point where the compressor is unable to transfer sufficient energy to the

gas through its blades. Consequently, the flow can reverse, potentially causing damage to the compressor.

One of the commonly employed techniques to prevent surge is the inclusion of recycle lines that connect the discharge side to the suction side of the compressor. Additionally, a controller logic that monitors the suction flow should be implemented. By opening the recycle valve, more flow is directed through the compressor, effectively avoiding surge. In this particular case, the compressors are assumed to be identical in design. However, in real-life scenarios, various factors may contribute to deviations between two compressors in series. The following sections elaborate on the incorporation of surge constraints into the model and the development of a control structure to effectively manage surge.

3.2.1.1 Modelling of surge constraint

In this study, the surge limit, determined by the pressure ratio and the flow rate through the compressor, was modeled based on the methodology presented in Section 2.21.

The surge line was subsequently obtained through model fitting, employing a trial and error approach. Using this information, the compressor curve for different radial speeds was derived and incorporated as a constraint within the optimization process. From an optimization perspective, this constraint has a practical impact, as it restricts the optimizer from further reducing the flow when the demand for gas lift falls below the surge limit, unless gas recycling is employed.

The decision variables related to the surge constraint for the compressors, was implemented in the model by the use of Equation 2.54. The related decision variables s_i are outlined in the Table 3.1.

Table 3.1: Surge constraint decision variables.

s_0	0.2063
s_1	0.001906
s_2	1.3948

3.2.1.2 Controller design

In contrast to the optimizer, the solver lacks inherent constraint handling logic, except for handling constant variables. However, if the constraint is kept constant, this can result in divergence or the loss of process dynamics effects during disturbances. Therefore, it is essential to incorporate controller logic to protect the compressor train from surge in the event of system disturbances. To ensure that the controllers respond in a manner that prevents surge from occurring, a certain level of back-off is assumed to be included in the calculation of the surge limit.

In addition to safety considerations, the economic implications of gas recycling are also noteworthy. The compression process in the compressors consumes energy, which is sourced either from onshore electricity or local gas/diesel turbines. Irrespective of the energy source, compressing already compressed gas incurs costs, leading to financial implications. Consequently, the act of re-compressing gas that has already undergone compression is essentially an inefficient use of energy and, consequently, a waste of financial resources.

Taking this into consideration, it is crucial to incorporate a logic that automatically closes the recycle valves when the surge flow exceeds the surge limit. To achieve this, we propose a control structure based on straightforward feedback principles. The control structure effectively regulates the gas flow to the surge limit when it is below the constraint. Additionally, it utilizes a simple switching mechanism to close the valves when the gas flow through the compressor surpasses the surge limit. The proposed feedback-based control structure is illustrated in Figure 3.2.

The subsequent aspect to be addressed is the selection of controller type and its tuning. Considering

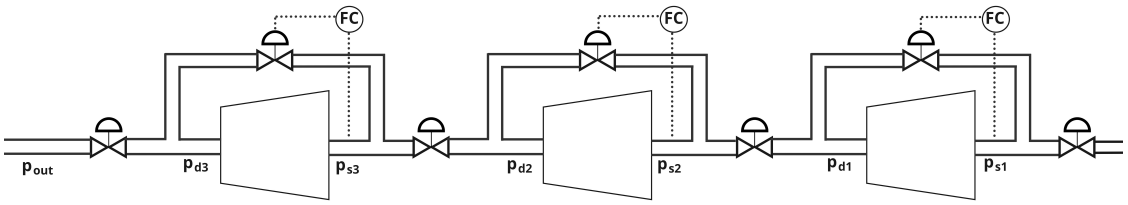


Figure 3.2: Compressor train with surge control.

the aforementioned potential risks associated with surge, a controller with a fast response is desired. Theoretical analysis suggests that either a Proportional-Integral (PI) controller or a pure Integral (I) controller would be well-suited for this purpose (see Section 2.10). For the current implementation, we propose utilizing a PI controller to achieve both rapid and reliable response. By employing the Simple Internal Model Control (SIMC) tuning method described in Section 2.11, we obtain the following tuning parameters for the three recycle valves.

Table 3.2: Controller parameters recycle valves.

Compressor	Kc	τ_I	τ_C
C_1	0.321	10	10
C_2	0.161	10	10
C_3	0.107	10	10

Where C_i denotes the three compressors, K_c is the controller gain, τ_I is the integral time and τ_C is the controller time. The implementation of the controllers with the additional logic can be observed in Appendix B.12.

3.2.2 Implementation of total produced gas control

The case under consideration examines a production system operating nominally within a region where the constraint on the total produced gas is active. In this scenario, we assume that the constraint on the produced gas is a soft constraint, implying that as long as it is satisfied at steady state, the system can operate safely. This assumption allows us to omit the inclusion of back-off, which would be necessary for constraints that must not be exceeded.

For the further implementation of self-optimizing local methods, it is recommended to always control the active constraint when it is optimally active, as this promotes optimal operation as explained in Section 2.17. However, it is important to note that the constraint is always controlled to prevent it from exceeding the constraint value. Additionally, it should be noted that when measurement error is incorporated into the Exact local method, any measurement error associated with the control of produced gas is disregarded.

Based on simulations, it has been observed that a single gas lift choke is inadequate for effectively controlling the produced gas during certain disturbances. Therefore, we suggest implementing split range control with the baton strategy, as described in Section 2.9. To determine the most suitable strategy, we will follow the proposed selection criteria.

1. In the subsequent cases discussed in this paper, the following gas lift chokes are available for controlling the produced gas: GLC1, GLC3, GLC4, and GLC5. The optimal nominal opening of each available input can be found in the table presented in Table 3.3.
2. Given the nature of the gas lift system, where all the gas lift chokes are interconnected through a common manifold, each valve exhibits the same effect on the produced gas.
3. To determine the order in which the valves should be utilized, we can examine the oil production of each well. In this context, it is understood that introducing additional gas into

Table 3.3: Valve opening.

Valve	Opening[0-1]
GLC_1	0.64
GLC_3	0.55
GLC_4	0.37
GLC_5	0.61

the tubing enhances production, while the converse is also true. Based on this rationale, it is apparent that the gas lift chokes will be adjusted to decrease the amount of produced gas. Additionally, we suggest controlling the produced gas using the gas lift chokes associated with the wells exhibiting the lowest oil production. the nominal oil production of each well can be observed in table3.4. From the data production data we choose the following order of active

Table 3.4: Oil production.

Well	Oil production[kg/s]
W_1	11.1919
W_3	13.1433
W_4	14.2117
W_5	12.2615

gas lift chokes: GLC1, GLC5, GLC3, GLC4.

4. The subsequent stage involves determining the controller tunings for the controllers and implementing the algorithm described in Section 2.9. For safety purposes, the control structure will also incorporate the production choke of well 1, in case the gas lift chokes are unable to effectively manage the disturbance.

Based on the selection method, we propose the following control structure, utilizing feedback measurements of the produced gas. It should be noted that the effective time delay in the simulator is assumed to be zero, although in real-life scenarios, delays may occur due to the distances between the inputs and the output. The control structure, incorporating split range baton logic, is shown in Figure 3.3.

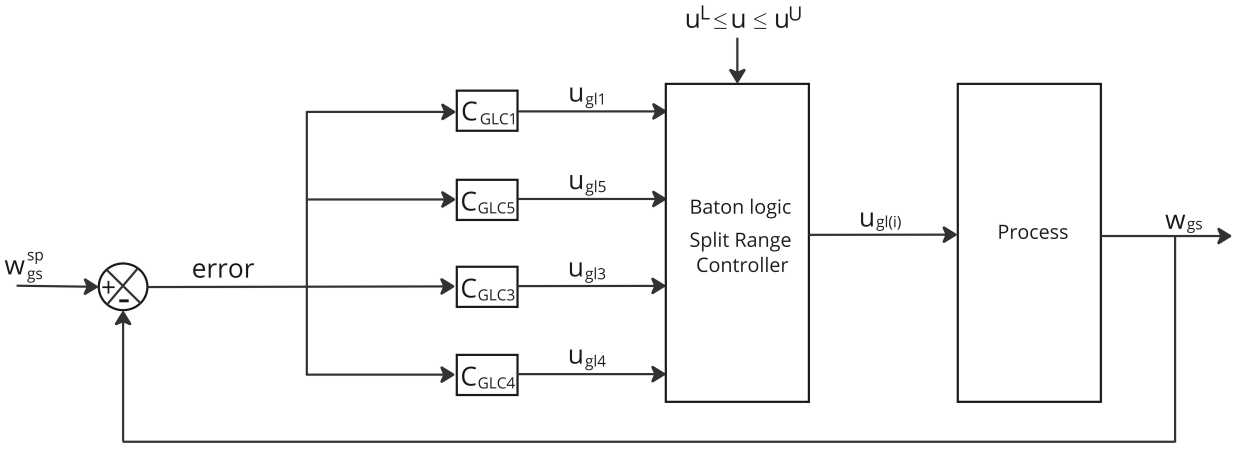


Figure 3.3: Proposed total produced gas control.

The total produced gas is measured, and based on the deviation between the setpoint and the measured values, the controllers will suggest an input to the process. The implemented logic dictates that GLC 1 will be utilized initially, and if it becomes saturated, control will be transferred to the next controller (in this case, GLC 5), while maintaining GLC 1 at its saturated value. This process

will continue until all the valves have reached saturation or until one of the controllers successfully controls the total produced gas. The baton logic, as implemented in this study, can be found in Appendix B.12.

The PI controllers were tuned by analyzing the open-loop step response of the corresponding valve and applying the SIMC tuning rules. The parameters of the controllers are presented in Table 3.5.

Table 3.5: Controller parameters total produced gas control.

Controller	Kc	τ_I	τ_C
C_{gl1}	0.793	572	2000
C_{gl3}	0.777	559	2000
C_{gl4}	0.758	540	2000
C_{gl5}	0.742	523	2000

Where C_{gli} denotes the controller related to gas lift choke i.

3.2.3 Implementation of changing active constraint control

As mentioned earlier, the system operates within an active constraint region during nominal operation. However, certain disturbances can cause a shift in the active constraint region. Following the principle discussed in Section 2.17 of always controlling the optimal nominal active constraint, we need to introduce selector logic to determine whether or not to control the produced gas in different regions.

To address the change in active constraints, we propose implementing a selector with CV-CV switching. The intended logic controls the produced gas at its active constraint when it is deemed optimal to do so, and switches to controlling the valves to their optimal nominal openings when the constraint is violated due to a change in the system. To determine the appropriate type of controller, we follow the procedure outlined in Section 2.8.

1. We begin by categorizing the constraints into two sets based on which input is most effective in satisfying the constraint. In the case of the changing active constraint, which is the total produced gas, we conducted a step change in the gas lift chokes and observed that constraint satisfaction was achieved when the chokes were closed down. Referring to Section 2.8, this set of constraints falls under the category Y^+.

2. The controllers have already been incorporated in the control of the produced gas, and the tuning parameters can be found in Table 3.5. When the constraint is not active, the valve will assess the current valve position and the optimal nominal valve position to suggest a modification in the valve position

3. From the procedure we propose a min selector to change between the constraint regions.

The proposed control structure is depicted in Figure 3.4. The feedback for the active constraint control consists of the produced gas as the measured value and the active constraint as the setpoint. On the other hand, the feedback for the controller comprises the valve position, while the setpoint is the optimal nominal valve position. The controller's setpoint is determined based on the active valve in the baton logic. In scenarios where multiple valves have reached saturation, we suggest implementing a logic where the previous valves are gradually controlled back to their nominal openings once the active valve has been regulated to its nominal opening.

The min selector will always choose the lowest input value computed by the two controllers. As previously discussed, when the total produced gas flow exceeds the active constraint, the split range controller will suggest a smaller input than the previous value to address the constraint. In this case, the split range controller will gradually decrease its input to regulate the gas production.

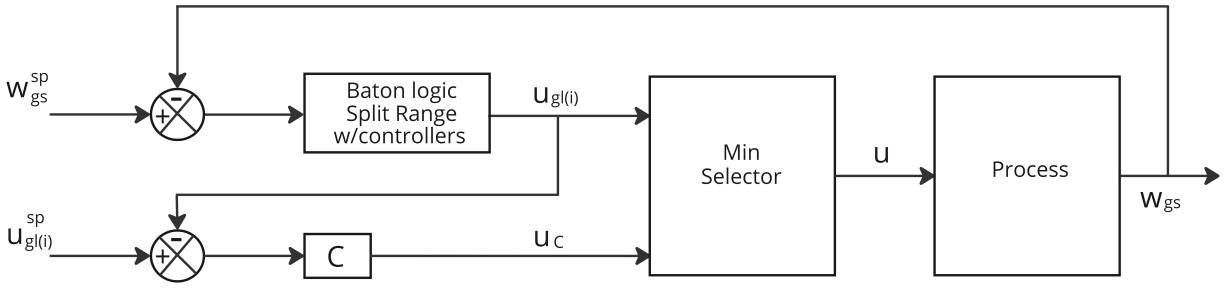


Figure 3.4: Proposed active constraint shifting.

On the other hand, when the total produced gas flow falls below the active constraint, the split range controllers will begin increasing its input to augment gas production from the corresponding well.

At nominal operation, Controller C remains unchanged and maintains its proposed input. Consequently, when the total produced gas increases, Controller C will suggest a larger input than the split range controller. However, when the total produced gas decreases, the split range controller will have the smallest input until the valve reaches its optimal nominal value, at which point Controller C will resume control.

The implementation of the min selector can be observed in Appendix B.12.

3.2.4 Implementation of level control

The control of the liquid level in a separator is essential for various reasons. One crucial aspect is the prevention of flooding, which can result in production issues, equipment failure, and environmental pollution. Additionally, controlling the separator pressure is important for maintaining stability. In this study, we propose the implementation of a controller that ensures a relatively stable liquid level during normal operation, allowing for some deviation.

We also introduce high-high (HH) and low-low (LL) limits that trigger a rapid response when breached. In real-life operations, there are typically multiple alarm limits before reaching HH and LL, alerting operators to take action and address the issue before critical limits are reached. However, for the scope of this project, we focus on two specific limits

During normal operation, the liquid outlet valve is responsible for monitoring the level of the separator using a simple feedback loop. The control scheme for maintaining a constant level in the separator is depicted in Figure 3.5. To achieve this, we suggest implementing a PI (Proportional-Integral) controller to regulate the liquid level.

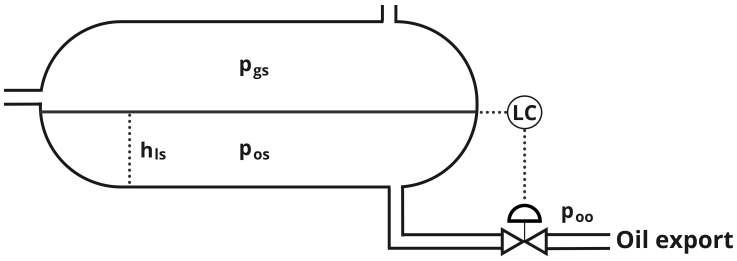


Figure 3.5: Constant control of separator level.

Table 3.6 provides the controller tuning parameters for the constant level control of the separator. The controller settings where obtained by open-loop step response and SIMC.

Table 3.6: Controller parameters constant control.

Valve	Kc	τ_I	τ_C
C_{os}	0.4	2000	500

The control structure for allowing the level of the separator to fluctuate between the defined High-High (HH) and Low-Low (LL) limits is depicted in Figure 3.6.

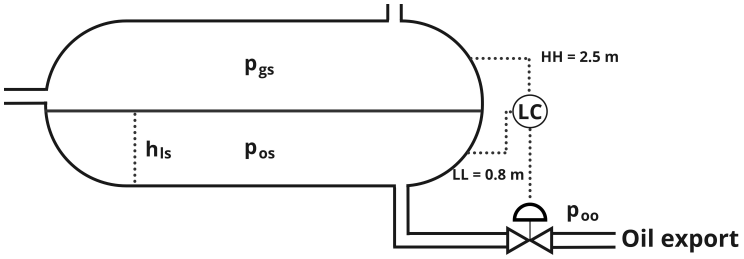


Figure 3.6: Boundary control of separator level.

Table 3.7 provides the controller tuning parameters for the boundary based level control of the separator. The controller settings where obtained by open-loop step response and SIMC.

Table 3.7: Controller parameters HH and LL control.

Valve	Kc	τ_I	τ_C
C_{os}	2	400	100

The related implementations of the control strategies can be observed in Appendix B.12.

3.2.5 Implementation of valve position control

During system operation, it is possible for certain controllers to reach their saturation limits, thereby affecting the range of control action. To mitigate the issue of saturating valves, one potential strategy is to employ valve position control, as detailed in Section 2.7. In our implementation, we introduce an additional valve that controls a target valve either towards a specific setpoint or away from a limiting value.

In our simulation, we incorporate an extra controller whenever the primary manipulated variable saturates. This approach serves two purposes: first, it allows us to obtain steady-state results for potential control structures, and second, it addresses the potential dangers associated with saturated variables in practical applications. To illustrate the significance of this, let us consider an example from the compressor train. If one of the recycle valves becomes fully open and the suction flow decreases, the controller lacks the means to counteract surge.

In typical valve position control, a combination of a large, slow valve and a small, fast valve is utilized. However, due to the constraints of our system model, the only available alternate valves are other gas lift chokes. Therefore, for the mentioned cases, we implement controllers that use the optimal setpoint of the other gas lift choke as the target setpoint. Based on this setpoint and the measurement of the opening of the other chokes, the controllers strive to stabilize the valve position around its nominal opening.

4 Method

4.1 Method Implementation

This section focuses on the implementation of various methods to obtain self-optimizing control structures. It begins by describing strategies for control structure design, which involve conducting a degree of freedom analysis, defining objectives, and identifying controlled variables. Subsequently, two different cases with varying numbers of manipulated variables and disturbances are considered for the implementation of local methods. The resulting control structures are then designed based on these implementations.

4.1.1 Top-down analysis

For the controlled variable selection, a the Top-down analysis for plant-wide control presented in Section 2.3 where utilized. Where the focus in Section 3.2 where on the design of regulatory control structures related to the safety of operation, in this section we focus on identifying potential self-optimizing controlled variables. The Top-down analysis is primarily implemented to get an overview over the problem.

4.1.1.1 Operational objectives and constraints

The operational objectives were previously defined in Section 3.1.1. The primary objective is to maximize oil production while minimizing compression-related costs. Several constraints were considered, including those related to total produced gas, total gas lift, surge, and power consumption in the compressor system. However, in the specific case studies discussed, the only constraint that impacts the system is the total produced gas constraint. It should be noted that, optimally, some of the manipulated variables are saturated at the nominal operation point, which imposes a constraint on further decrease or increase of these variables.

4.1.1.2 Degree of freedom analysis

An important subsequent step involves investigating the manipulated variables associated with our plant and analyzing their dynamic and steady-state effects on the plant. The goal is to identify the manipulated variables that have a steady-state impact on the cost (N_{opt}). These manipulated variables can be determined by subtracting the total number of manipulated variables from the sum of the manipulated variables with no steady-state effect (i.e., no effect on cost), as well as the number of controlled variables that need to be regulated but have no effect on the cost.

In Table 4.1 the optimal nominal valve openings can be observed.

Table 4.1: Valve openings in percent[%]

u_{gl1}	u_{gl2}	u_{gl3}	u_{gl4}	u_{gl5}	u_{gl6}	u_{pc1}	u_{pc2}
64.17	53.79	54.59	36.71	60.81	50.99	100	100
u_{pc3}	u_{pc4}	u_{pc5}	u_{pc6}	u_{rec1}	u_{rec2}	u_{rec3}	u_{os}
100	100	100	100	0	0	0	30.84

Based on the evaluation of Table 4.1 and the corresponding valve openings, it is evident that the production valves (u_{pc_i}) are saturated at the nominal optimum. Therefore, the production chokes should be maintained at a constant position, in accordance with the principle of controlling the active constraints. Thus, the production chokes do not represent degrees of freedom.

It is however important to assess the valve openings under different operating conditions, to determine the optimal values in these scenarios and verify if this holds true for all cases. However, based on the analysis of the objective function, we would anticipate the optimizer to keep the valves

fully open to maximize oil production. Nevertheless, there might be certain scenarios where the production chokes need to be partially closed to ensure constraint control.

Another observation can be made regarding the recycle valves u_{rec_i}. They are optimally saturated fully closed at the optimal nominal operation point. This aligns with the evaluation of the objective function, as recycled gas does not provide cost benefits and is typically only required for security purposes. Therefore, the recycle valves should be utilized for constraint control, specifically related to surge prevention as specified in Section 3.2.1, and should not be considered as a degree of freedom.

The valve at the liquid outlet of the separator, u_{os}, is not saturated at the optimum. However, its control is necessary for maintaining stability and safety by regulating the level in the separator. Since the control of levels does not directly impact the steady-state cost, we can exclude this valve from further evaluation as a degree of freedom.

Based on the reasoning provided above, we can calculate the number of steady-state degrees of freedom with Equation 4.1.

$$N_{ss} = 16 - 10 = 6 \tag{4.1}$$

Therefore, we have a total of six degrees of freedom at nominal operation, which correspond to the six gas lift chokes.

4.1.1.3 Identification of controlled variables

In the previous section, we identified the steady-state degrees of freedom. However, in this section, we will systematically explore all the potential controlled variables and manipulated variables.

The model includes a total of 179 potential controlled variables, which includes the positions of manipulated variables. Among these, 16 are manipulated variables. By employing the concept of combinations, Equation 4.2 reveals the vast number of possible combinations that exist.

$$\frac{179!}{(179-16)!16!} = 6.62 \cdot 10^{19} \tag{4.2}$$

Due to the impracticality of evaluating all of these combinations, it is evident that a reduction of the set of potential controlled variables is necessary for further evaluation. Considering the model's inclusion of 6 identical wells and 3 identical compressors, we can significantly reduce the number of potential controlled variables by initially assessing the system as if it comprised only one well and one compressor. The resulting reduction in potential controlled variables from the simplification can be observed in Equation 4.3.

$$179 - 105 = 74. \tag{4.3}$$

Based on the previous section's discussion on degrees of freedom and the regulatory control design described in Section 3.2.4, it is evident that the level of the separator and the valve at the oil outlet of the separator will be controlled to ensure stability and safety. Consequently, we can exclude these variables from the set, resulting in a reduced set of 72 potential controlled variables.

Building upon the principle of controlling active constraints outlined in Section 2.17, it is observed that at the nominal operation point, the production chokes are fully open, the recycle valves are fully closed, and the total produced gas is constrained to its active limit. Additionally, the recycle flow has a direct correlation with the opening of the recycle valve. Therefore, we can exclude the measurement of recycle flow as well. Consequently, the set of potential controlled variables is further reduced by subtracting the number of production chokes (6), recycle valves (3), the total

produced gas (1), and the recycle flow (1) from the initial count, resulting in a reduced set of 57 variables. Where (i) denotes the number of variables related to the measurements we decide to remove from the set.

Another selection method is to eliminate closely related variables, similar to what was done with the recycle flow. Applying this principle, we can eliminate the oil out of the separator (1) since it is closely related to the valve at the oil outlet of the separator.

Additionally, the volume of gas and oil in the separator (2) is closely related to the height of the separator, which we already control for stability reasons.

The discharge pressure of the third compressor and the pressure in the gas lift line (1) are also closely related, allowing us to choose to disregard one of these variables.

Furthermore, the flow in and out of the compressors can be controlled by managing the total gas lift due to the serial configuration of the compressor train (2). The same logic applies to the valves related to the common line which in this case is considered constant (4). Thus, reducing the set to 51.

The decision of which variable to retain or discard can be made arbitrarily since controlling either of the two variables would affect the system in the same manner. Consequently, we remove the gas lift flow (1) from each well variable.

Based on process insight and the model equations, it is evident that controlling the power of the compressor (1) is equivalent to managing the total gas lift. Additionally, the pressure ratio (1) is influenced by the flow through the compressor (1) and the assumed constant radial speed in this scenario(3). Furthermore, the polytropic head (1) and polytropic efficiency (1) are both dependent on the pressure ratio and the flow through the compressors. As a result, we can eliminate these four variables from the set, reducing it to 41.

We can further eliminate variables that are exclusively used for modeling purposes, such as the surge constraint (1) and maximum pressure ratio (1). This reduces the remaining set of variables to 39.

To further narrow down the potential CVs, we compare the variables in the model with the available measurements in the real production site. The real system primarily consists of temperature and pressure transmitters. Additionally, a multi-phase meter (MPFM) is used in the well system to calculate the flow of oil and gas. However, using density (7) as a potential CV is disregarded due to potential measurement errors when calculating it based on other measurements. This reduces the set of variables to 32.

Furthermore, the model includes mass variables(6) in different regions, which are calculated based on mass flow in versus mass flow out. However, the approximation of mass in certain regions, such as the riser, is prone to measurement error. This leads to a further reduction to 26 variables.

We also note that controlling the suction pressure of the compressor(1) will have the same effect as controlling the pressure in the separator. Hence, we can eliminate these redundant variables, resulting in a reduced set of 25 variables.

In the well system, we lack available measurements for specific variables. These variables include the flow through the injection valve (which is equivalent to the flow through the gas lift choke at steady state), reservoir oil (which corresponds to the produced oil), reservoir gas (which corresponds to the produced gas of the well minus the gas lift of the well), and the injection point pressure of the gas lift in the tubing. Considering these variables, we can reduce the initial set of 25 variables to 21.

Considering that it would be more meaningful to control either the oil flow or the gas flow rather than the total flow (1) from the valve, we further reduce the set to 20 variables.

The pressure in the separator is generally determined by the gas pressure rather than the liquid pressure (1), as they are related. Hence, we can eliminate one of these variables, resulting in a reduced set of 19 variables.

Additionally, there are no flowmeters in the riser system (3), leading to a further reduction to 16 variables.

The manifold pressure and riser head pressure (1) are related, so we can arbitrarily choose to control one of them, resulting in a set of 15 variables.

By removing the degrees of freedom, the gas lift chokes (6) the set of variables is reduced 9 potential controlled variables.

Therefore, the resulting list of potential controlled variables consists of the following 9 variables, found in Equation 4.4.

$$y = \left[p_{wh_i}, p_{bh_i}, p_{ai_i}, p_{gs}, p_m, p_{d3}, w_{po_i}, w_{pg_i}, w_{gl}\right] \tag{4.4}$$

where p_{wh_i} is the wellhead pressure of well i, p_{bh_i} is the bottomhole pressure of well i, p_{ai_i} is the annulus pressure at the injection point of well i, p_{gs} is the separator pressure, p_m is the manifold pressure, p_{d3} is the discharge pressure of compressor 3, w_{po_i} is the produced oil of well i, w_{pg_i} is the produced gas of well i and w_{gl} is the total gas lift.

4.1.2 Case 1

In Case 1, we examine the system's response to a disturbance in the Gas-oil ratio (GOR) of well 2. The magnitude of the disturbance is assumed to range from -3% to +3% of the nominal GOR, which is 0.13 kg/kg. To determine the best combination of variables for self-optimizing control, we employ a brute force method with gas lift choke 2 as the MV. Initially, we assess the loss incurred by controlling individual measurements to their optimal nominal values. Subsequently, we implement local methods such as the nullspace method and exact local method to identify combinations of measurements with minimal loss at the specified disturbances.

To simplify the control strategy, we suggest controlling either individual measurements or combinations of these measurements with the manipulated variable associated with the well experiencing the disturbance. In order to assess the loss, we will consider two scenarios for evaluation. The first scenario involves utilizing all degrees of freedom for optimization, while the second scenario focuses solely on the degrees of freedom that are actually modified during the simulations.

Figure 4.1 illustrates the behavior of well 2, demonstrating variations in the GOR above or below its nominal value, which we define as 100%.

Through our observations of how the active constraint responds to disturbances, we have noted that positive disturbances in the Gas-oil ratio (GOR) yield the same constraint region as the nominal operating point, which is constrained by the total produced gas capacity.

However, since the nominal point lies on the boundary between these regions, we propose perturbing the GOR of the system by a small amount to obtain a new operating point that is unconstrained. This allows us to assess the system's behavior in the unconstrained region. Therefore, we suggest controlling the total produced gas within the active constraint region rather than the unconstrained region as implemented in Section 3.2.3. (Note that the term "unconstrained" is used in this study for simplicity of writing and to distinguish between the two regions. Despite the fact that the region is not truly unconstrained.)

To ensure the validity of the local methods, it is crucial for the active constraints to remain unchanged. As discussed in Section 2.17, separate control structures should be implemented for each region to maintain control effectiveness.

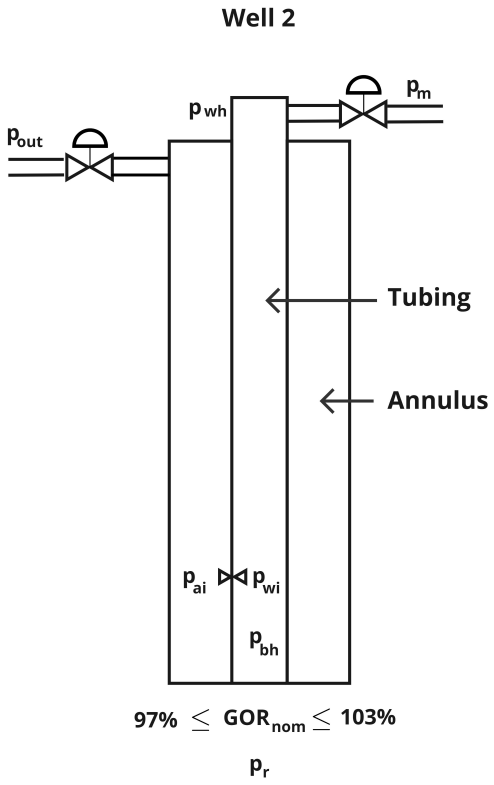


Figure 4.1: Case where the GOR of well 2 can change with ±3%.

To enhance stability and expedite convergence in our simulations, we suggest employing constant separator control, as described and implemented in Section 3.2.4. This approach ensures consistent control of the separator level throughout the simulation process. Moreover, we have integrated surge control, as described in Section 3.2.1, to address potential surge limit violations. Additionally, the split range controller with baton strategy, as developed in Section 3.2.2, along with the associated min selector for CV-CV switching, as described in Section 3.2.3, is implemented.

4.1.2.1 Single measurements for self optimizing control

In this section we will analyze the impact and loss associated with implementing simple feedback controllers to control individual measurements at their nominal optimum for the mentioned disturbances. Due to the presence of two active constraint regions, we will assess the most effective control strategy for when the constraint on total produced gas is active or inactive. This corresponds to positive and negative changes in GOR.

Since we are only controlling a single measurement, there is no need to design different control structures specifically for gas lift choke 2 in each case. This differentiation may however be necessary when implementing the optimal measurement combination of multiple controlled variables.

In this scenario, we disregard any measurement noise or error and focus on evaluating potential controlled variables that are less susceptible to such issues. Flow measurements, in general, are more challenging and tend to have a greater margin of error compared to pressure transmitters. As a result, we will assess the controlled variables related to pressure transmitters listed in Equation 4.4.

The measurements evaluated for self-optimizing control (SOC) properties in this case are shown in Equation 4.5.

$$y = \left[p_{wh_2}, p_{bh_2}, p_{ai_2}, p_{gs}, p_m, p_{d3}\right] \tag{4.5}$$

The next step in the procedure involves performing open-loop step responses by opening gas lift choke 2 by 10% and assessing the response for each variable. The response is evaluated using the SIMC method, as described in Section 2.11. PI controllers are then designed using the velocity form as described in Section 2.10, with the controller parameters obtained from the SIMC method.

To minimize the interaction with the gas lift choke responsible for controlling the total produced gas, additional tuning was performed on the controllers to reduce inter-valve interaction and eliminate or minimize oscillations.

Table 4.2 presents the controller parameters acquired for controlling the measurements mentioned in Equation 4.5.

Table 4.2: Controller parameters single measurement control.

Controller	Kc	τ_I[s]	τ_C[s]
C_{wh2}	0.05165	262	1500
C_{bh2}	-0.03930	690	1900
C_{ai2}	0.005875	64	1300
C_{gs}	2.2150	550	2000
C_m	0.08914	293	2500
C_{d3}	0.01926	225	800

Where C_i is the controller related to measurement i, Kc is the controller gain, τ_I is the integral time and τ_c is the controller time.

In order to address difficulties in controlling the variables to their nominal setpoints, valve position control was implemented for managing the manifold pressure and discharge pressure.

Gas lift choke 3 was employed for this purpose and tuned using open-loop step response on the closed-loop control of the relevant variables. This allowed us to assess the impact of gas lift choke 3 opening on the control of gas lift choke valve 2.

The SIMC method was subsequently employed to determine appropriate tuning parameters for the valve position controllers. The resulting control parameters, where the nominal setpoint of gas lift choke 2 serves as the reference, are presented in Table 4.3.

Table 4.3: Controller parameters VPC using GLC3.

Controller	Kc	τ_I[s]	τ_C[s]
VPC_m	-0.3094	500	1500
VPC_{d3}	-0.07737	500	2000

The system was simulated using the CasAdi integrator, specifically IDAS. The potential disturbances were applied during the simulations. Additionally, the loss was computed by comparing the results of the cost function obtained from the CasAdi optimizer IPOPT, which represents RTO (Real-Time Optimization), with the cost function calculated during the simulations. The controller implementations can be observed in Appendix B.12.

4.1.2.2 Nullspace method Implementation

The next step in the evaluation process involves implementing the nullspace method to determine the optimal combinations of measurements.

Similar to the previous case, the nullspace method does not take measurement noise into account. We will thus consider combinations of measurements related to the same measurements evaluated in Section 4.1.2.1. According to the theory discussed in Section 2.5, a condition for using the nullspace

method is the relationship between the number of controlled variables, manipulated variables, and disturbances. In this case, this condition is satisfied when,

$$n_y \geq n_u + n_d = 2. \tag{4.6}$$

Where n_y is the number of controlled variables, n_u is the number of manipulated variables and n_d is the number of disturbances. From Equation 4.6, we know that we need at least a combination of 2 measurements to implement the method.

Expanding on the findings in Section 4.1.2.1, we opt to form measurement combinations involving the bottomhole pressure of well 2. This particular variable yielded the lowest loss across both constraint regions which can be observed in Section 5.3. Another observation that was made, relates to the unexpected behavior of the separator pressure in the constrained region. Consequently, the separator pressure was excluded from further analysis and consideration within the constrained section.

As we navigate through two distinct constraint regions for opposing disturbances, it is necessary to establish separate control structures for each region. The nominal operating point lies precisely on the border between these active constraints. Consequently, by applying a slight negative perturbation to the GOR, we transition into the region where the constraint becomes inactive, leading to distinct system behavior. To account for this, matrix F will be derived by introducing a negative perturbation for the negative GOR change (inactive constraint) case, where the nominal operating point is perturbed, and a positive perturbation for the positive GOR change (active constraint) case.

The measurement combinations considered in the case can be observed in Table 4.4.

Table 4.4: Measurement combinations for Nullspace method

Combination
p_{bh2}&p_{wh2}
p_{bh2}&p_{ai2}
p_{bh2}&p_{gs}
p_{bh2}&p_{m}
p_{bh2}&p_{d3}
No control

In order to find the optimal sensitivity matrix F, we perturb the system by a small amount, specifically ±0.01% of the nominal GOR, in both constraint regions. Subsequently, we re-optimize the variables and calculate the resulting change in each controlled variable using finite difference. The F matrices for the constrained case, denoted (C) and the unconstrained case, denoted (U) are presented in Table 4.5.

Once matrix F was obtained for the five cases, the subsequent step was to derive matrix H from the equation HF = 0. To calculate H, we found the left nullspace of F, which corresponds to null(F^T). The calculation involved transposing the matrix F using the transpose() function from the numpy library in Python, and then using the null_space() function from the scipy.linalg library to determine the nullspace.

After obtaining matrix H, we further calculated the controlled variables c = Hy. The setpoints for the controlled variables c, denoted as c_{ns}, were then determined using the optimal nominal values of the variables in the combination. The matrices H for the constrained case can be observed in Table 4.6. While for the unconstrained case, the H matrices can be observed in Table 4.7.

The controllers where then designed to control $c = H[0]y_1 + H[1]y_2$ to c_{ns}. The PI controllers where designed using open-loop step response and tuned with the SIMC method. The related controller

Table 4.5: Sensitivity matrices for Nullspace method regions

Combination	F = $\frac{\delta y^{opt}}{\delta d}$(C)	F = $\frac{\delta y^{opt}}{\delta d}$(U)
p_{bh2}&p_{wh2}	$[-21.177 \quad -351.797]^T$	$[0.426 \quad 34.716]^T$
p_{bh2}&p_{ai2}	$[-21.177 \quad -525.742\ ]^T$	$[0.426 \quad 159.169]^T$
p_{bh2}&p_{gs}	-	$[0.426 \quad -3.782]^T$
p_{bh2}&p_{m}	$[-21.177 \quad -332.475]^T$	$[0.426 \quad 32.329]^T$
p_{bh2}&p_{d3}	$[-21.177 \quad 4106.803]^T$	$[0.426 \quad 906.223]^T$
No control	-	-

Table 4.6: Measurement combinations for Nullspace method constrained case

Combination	H = null(F^T)	c_{ns}
p_{bh2}&p_{wh2}	$[0.998 \quad -0.0601]$	132.139
p_{bh2}&p_{ai2}	$[0.999 \quad -0.0403]$	133.033
p_{bh2}&p_{gs}	-	-
p_{bh2}&p_{m}	$[0.998 \quad -0.0636]$	131.952
p_{bh2}&p_{d3}	$[0.999 \quad 0.00516]$	138.050
No control	-	-

tunings for the constrained case can be found in Table 4.8. While for the unconstrained case, the controller tunings can be observed in Table 4.9.

4.1.2.3 Exact local Implementation

The previous methods do not account for the expected measurement errors that can arise from the transmitters. To simulate these potential measurement errors, we have generalized the expected errors for pressure transmitters to be ±0.0025%, and for flow transmitters to be ±1%. It is important to note that these measurement errors are based on highly precise transmitters and the actual errors in real-world transmitters may be larger. However, the primary objective of this case study is to demonstrate the difference between measuring flow and pressure.

We will follow the strategy outlined in Section 2.13 to identify controlled variables, which will be evaluated for loss under different disturbances. The combinations assessed in this section are provided in Table 4.10.

In this case we only consider one disturbance, namely ±3% in the GOR of well 2. Consequently, our W_d found in Equation 4.7 will be equal to the 3% of the nominal GOR,

Table 4.7: Measurement combinations for Nullspace method unconstrained case

Combination	H = null(F^T)	c_{ns}
p_{bh2}&p_{wh2}	[0.999 −0.0123]	136.230
p_{bh2}&p_{ai2}	[0.999 −0.00268]	136.959
p_{bh2}&p_{gs}	[0.994 0.112]	138.819
p_{bh2}&p_{m}	[0.999 −0.0132]	136.182
p_{bh2}&p_{d3}	[0.999 −0.000470]	137.156
No control	-	-

Table 4.8: Controller parameters for Nullspace method constrained region.

Controller	Kc	τ_I[s]	τ_C[s]
p_{bh2}&p_{wh2}	-0.1034	621	1000
p_{bh2}&p_{ai2}	-0.0649	591	1500
p_{bh2}&p_{gs}	-	-	-
p_{bh2}&p_{m}	-0.05303	620	2000
p_{bh2}&p_{d3}	-0.08808	621	1200

Table 4.9: Controller parameters for Nullspace method uconstrained region.

Controller	Kc	τ_I[s]	τ_C[s]
p_{bh2}&p_{wh2}	-0.1062	621	1000
p_{bh2}&p_{ai2}	-0.1065	621	1000
p_{bh2}&p_{gs}	-0.1078	623	1000
p_{bh2}&p_{m}	-0.1068	622	1000
p_{bh2}&p_{d3}	-0.1071	623	1000

Table 4.10: Combinations evaluated with the exact local method

Combination
p_{bh2}&p_{d3}
p_{bh2}&p_{wh2}
p_{bh2}&w_{po2}
p_{bh2}&w_{pg2}
p_{bh2}&w_{gl}
w_{po2}&w_{gl}
w_{pg2}&w_{gl}

$$W_d = 0.0039. \tag{4.7}$$

In the subsequent phase of the process, we aim to determine the potential measurement noise for each controlled variable. Given the minor perturbation in the GOR and the specific local conditions, we opt to derive the measurement error associated with each variable from its nominal value and the error associated with the transmitter type. The errors associated with the transmitters are found in Table 4.11.

Table 4.11: Measurement errors related to the implementation of the exact local method.

Combination	± error
p_{bh2}	0.00343
p_{d3}	0.00398
w_{po2}	0.124
w_{pg2}	0.0234
w_{gl}	0.0433
p_{wh2}	0.00254

In this case, considering the combinations of variables, which amount to two, Wn is constructed as a diagonal matrix where the associated measurement errors are arranged in the same order as the measurements in y. The specific implementation of Wn for the measurement combination p$bh2$ and w$po2$ is provided in Equation 4.8.

$$W_{n_{bh2\&po2}} = \begin{bmatrix} 0.00343 & 0 \\ 0 & 0.124 \end{bmatrix} \tag{4.8}$$

The subsequent step entails obtaining F, which can be accomplished through two approaches: linear approximation around the nominal point or by finite difference involving perturbation of the system with a small disturbance, followed by re-optimization. In this case, the latter method was employed, similar to the nullspace method in Section 4.1.2.2.

The following F matrices for the constrained case, denoted (C) and for the unconstrained case, denoted (U) can be observed in Table 4.12.

Our objective is to compare the combinations derived from the nullspace method that exhibit the lowest loss in each region with the combinations associated with the introduced flow measurements. As a result, the first two combinations are only assessed within a single constraint region. This decision is driven by the underlying rationale of the study.

Once the sensitivity matrix was determined, the gain matrix was obtained through finite difference analysis. This gain matrix illustrates the relationship between each variable and the corresponding changes in the manipulated variable. In the absence of any disturbances being applied to the system, a perturbation equal to the nominal opening times 10^{-5} was introduced to the gas lift choke of well 2. Subsequently, the system was solved using the IDAS integrator.

The gains related to $G_y = \frac{\delta y}{\delta u}|_{d=0}$ can be observed in Table 4.13.

The subsequent stage involves determining Y in order to compute the H matrix. According to the theory presented in Section 2.14, Y is a composition of the F matrix and Wn. Y was computed for each combination. In Equation 4.9 an example illustrating the calculation of Y for the p$bh2$ and w_{po2} combination can be observed.

$$Y = \begin{bmatrix} -21.177 & 0.00343 & 0 \\ 14.824 & 0 & 0.124 \end{bmatrix} \tag{4.9}$$

Table 4.12: Sensitivity matrices for the exact local method

Combination	F = $\frac{\delta y^{opt}}{\delta d}$(C)	F = $\frac{\delta y^{opt}}{\delta d}$(U)
p_{bh2}&p_{d3}	$[-21.177 \quad 4106.800]^T$	-
p_{bh2}&p_{wh2}	-	$[-537450 \quad 6791.001\ ]^T$
p_{bh2}&w_{po2}	$[-21.177 \quad 14.824]^T$	$[-4.329 \cdot 10^5 \quad 2.305 \cdot 10^5]^T$
p_{bh2}&w_{pg2}	$[-21.177 \quad -39.155]^T$	$[-466615 \quad 17771]^T$
p_{bh2}&w_{gl}	$[-21.177 \quad -291.551]^T$	$[-526464 \quad 5189]^T$
w_{po2}&w_{gl}	$[14.824 \quad -291.551]^T$	$[284.980 \quad 26.830]^T$
w_{pg2}&w_{gl}	$[-39.155 \quad -291.551]^T$	$[3016.840 \quad -270.090]^T$
No control	-	-

Table 4.13: Gain from gas lift choke 2 on the controlled variables

Combination	$G_y = \frac{\delta y}{\delta u}\|_{d=0}$
p_{bh2}	-6.292
p_{d3}	-12.232
w_{po2}	4.404
w_{pg2}	1.778
w_{gl}	0.665
p_{wh2}	2.768

At this point, all the essential components required to compute H are readily available. Consequently, we proceed to calculate H for the preceding case by the analytical solution described in Section 2.14. In Equation 2.49, the calculation of H for the measurement combinations can be observed.

$$H^T = G_y(YY^T)^{-1} \tag{4.10}$$

In order to compute the expression, we utilize various functions from the numpy library. Specifically, we employ the concatenate() function to obtain Y, the transpose() function to calculate the transpose of matrices, matmul() function for matrix multiplication, and inv() function from numpy.linalg to find the inverse of matrices.

The resulting H matrices for the measurement combinations in the constrained case are presented in Table 4.14, while the H matrices for the unconstrained region are presented in Table 4.15.

The final phase entails obtaining the controller settings for the measurement combinations. Similar to the previous implementations, we initiate by assessing the combinations through an open-loop

Table 4.14: Optimal combination for the exact local method positive GOR

Combination	H	c_{ns}
p_{bh2}&p_{d3}	$[-539914.347 \quad -2784.256]$	-7.45·10^7
p_{bh2}&p_{wh2}	$[-5.348 \cdot 10^5 \quad -1.416]$	-7.34·10^7
p_{bh2}&w_{po2}	$[-920.434 \quad 0.491]$	-1.26·10^5
p_{bh2}&w_{pg2}	$[-42892.005 \quad 22740.062]$	-5.83·10^6
p_{bh2}&w_{gl}	$[-293089.719 \quad 21259.285]$	-4.01·10^7
w_{po2}&w_{gl}	$[286.779 \quad 15.074]$	3.63·10^3
w_{pg2}&w_{gl}	$[2900.464 \quad -388.455]$	5.11·10^3

Table 4.15: Optimal combination for the exact local method negative GOR

Combination	H	c_{ns}
p_{bh2}&p_{d3}	$[-539914.347 \quad -2784.253]$	-7.45·10^7
p_{bh2}&p_{wh2}	$[-5.346 \cdot 10^5 \quad -1.416]$	-7.34·10^7
p_{bh2}&w_{po2}	$[-4.329 \cdot 10^5 \quad 230.539]$	-5.94·10^7
p_{bh2}&w_{pg2}	$[-466615.861 \quad 17771.120]$	-6.39·10^7
p_{bh2}&w_{gl}	$[-526464.962 \quad 5189.135]$	-7.22·10^7
w_{po2}&w_{gl}	$[284.981 \quad 26.839]$	3.66·10^3
w_{pg2}&w_{gl}	$[3016.845 \quad -270.098]$	5.89·10^3

step response in gas lift choke 2. Subsequently, the SIMC method is employed to determine the controller settings for our PI controllers. The controller tunings for the constrained case can be observed in Table 4.16, while for the unconstrained case, the controller tunings can be observed in 4.17.

4.1.3 Case 2

In Case 2, similar to Case 1, we investigate the system's response to a disturbance in the Gas-Oil Ratio (GOR) of well 2. The magnitude of the disturbance is assumed to vary from -3% to +3% of the nominal GOR, which is 0.13 kg/kg. Additionally, we introduce a disturbance in the GOR of well 6. The magnitude of the GOR disturbance in well 6 is assumed to range from -2% to +2% of the nominal GOR, which is 0.135 kg/kg.

Table 4.16: Controller parameters for Exact local method constrained case.

Controller	Kc	τ_I[s]	τ_C[s]
p_{bh2}&p_{d3}	$6.794 \cdot 10^{-8}$	431	2000
p_{bh2}&p_{wh2}	$1.397 \cdot 10^{-7}$	434	1000
p_{bh2}&w_{po2}	$4.054 \cdot 10^{-7}$	434	2000
p_{bh2}&w_{pg2}	$7.012 \cdot 10^{-7}$	404	2000
p_{bh2}&w_{gl}	$2.515 \cdot 10^{-7}$	432	1000
w_{po2}&w_{gl}	0.000186	110	500
w_{pg2}&w_{gl}	$8.615 \cdot 10^{-5}$	202	500

Table 4.17: Controller parameters for Exact local method unconstrained case.

Controller	Kc	τ_I[s]	τ_C[s]
p_{bh2}&p_{d3}	$6.794 \cdot 10^{-8}$	431	2000
p_{bh2}&p_{wh2}	$1.397 \cdot 10^{-7}$	434	1000
p_{bh2}&w_{po2}	$1.723 \cdot 10^{-7}$	434	1000
p_{bh2}&w_{pg2}	$1.574 \cdot 10^{-7}$	432	1000
p_{bh2}&w_{gl}	$1.416 \cdot 10^{-7}$	434	1000
w_{po2}&w_{gl}	0.000233	110	400
w_{pg2}&w_{gl}	$8.058 \cdot 10^{-5}$	200	500

To simplify the control strategy, we suggest controlling the measurement combinations with gas lift choke 2 and 6. In order to assess the loss, we will consider two scenarios for evaluation. The first scenario involves utilizing all degrees of freedom for optimization, while the second scenario focuses solely on the degrees of freedom that are actually modified during the simulations.

Figure 4.2 illustrates the behaviour of well 2 and 6, demonstrating variations in the GOR above or below its nominal value, which is defined as 100%.

To enhance stability and expedite convergence in our simulations, we suggest employing constant separator control, as described and implemented in Section 3.2.4. This approach ensures consistent control of the separator level throughout the simulation process. Moreover, we have integrated surge control, as described in Section 3.2.1, to address potential surge limit violations. Additionally, the split range controller with baton strategy, as developed in Section 3.2.2, along with the associated min selector for CV-CV switching, as described in Section 3.2.3, is implemented.

4.1.3.1 Linear approximation of the system

To determine the optimal combination of measurements, the first step is to identify the potential controlled variables. In accordance with the measurements outlined in Section 4.1.1.3, we select the measurements as potential controlled variables. The list of potential controlled variables for this case is provided in Equation 4.11.

$$y = [p_{wh_2}, p_{bh_2}, p_{ai_2}, p_{wh_6}, p_{bh_6}, p_{ai_6}, p_m, p_{d3}, w_{po_2}, w_{pg_2}, w_{po_6}, w_{pg_6}, w_{os}, w_{gs}, w_{gl}] \tag{4.11}$$

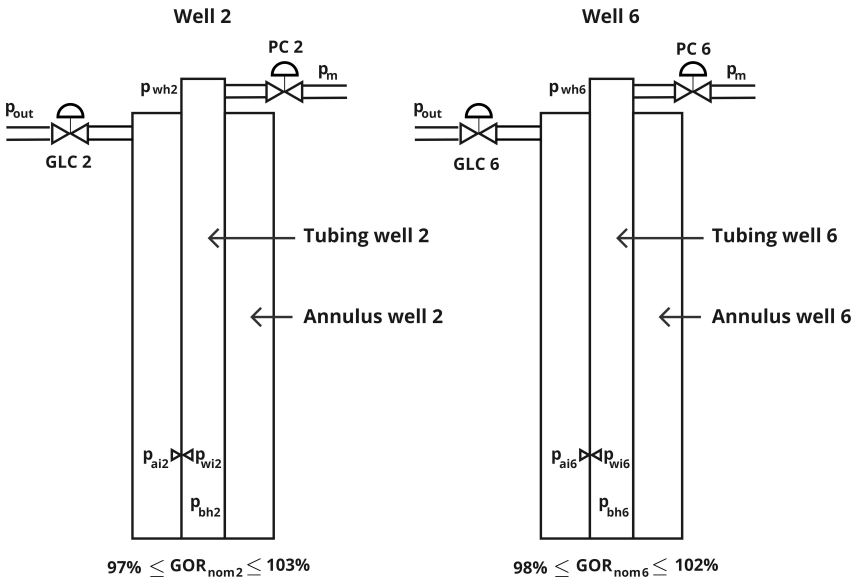


Figure 4.2: Case where the GOR of well 2 can change with ±3% and the GOR of well 6 can change with ±2%.

In this case, the manipulated variables under consideration are the gas lift choke valves of well 2 and well 6. In Equation 4.12, we define u as a list containing the manipulated variables.

$$u = \left[u_{glc2}, u_{glc6}\right]. \tag{4.12}$$

Furthermore, the disturbances considered in this case are the GOR of well 2 and 6. In Equation 4.13, we define d as a list containing the disturbances.

$$d = \left[GOR_2, GOR_6\right] \tag{4.13}$$

To implement the bracket and bounds algorithm for subset selection, the first step involves finding a linearized version of the model around the nominal operating point. This process is described in detail in Section 2.13. The linearized model can be observed in Equation 4.14.

$$y = G_u^y u + G_d^y W_d d + W_n e \tag{4.14}$$

To obtain this linearized model we need to calculate G_u^y, G_d^y, W_d and W_n for our system of measurements.

To begin, we obtain G_u^y, which represents the Jacobian matrix evaluated at the nominal operating point and signifies the gain from the inputs to the outputs. To calculate the gains from the two manipulated variables to the outputs, we employ finite difference by simulating the system and perturbing the manipulated variables with a small fraction (10^{-8}) of their original values. The resulting G_u^y matrix can be observed in Equation 4.15.

$$G_u^y = \begin{bmatrix} \frac{\delta y_1}{\delta u_1} & \frac{\delta y_1}{\delta u_2} \\ \vdots & \vdots \\ \frac{\delta y_{13}}{\delta u_1} & \frac{\delta y_{13}}{\delta u_2} \end{bmatrix} = \begin{bmatrix} 2.768 & 0.487 \\ 5.347 & 0.168 \\ -6.292 & 1.249 \\ 0.436 & 2.938 \\ 0.181 & 5.165 \\ 1.156 & -6.008 \\ 0.768 & 0.809 \\ -12.232 & -12.218 \\ 4.404 & -0.874 \\ 1.778 & -0.223 \\ -0.867 & 4.506 \\ -0.221 & 1.825 \\ 0.101 & 0.112 \\ 0.0106 & 0.0378 \\ 0.665 & 0.669 \end{bmatrix} \tag{4.15}$$

The procedure for obtaining G_u^y is outlined in Appendix B.1.

We proceed to determine G_d^y, which is the Jacobian matrix evaluated at the nominal point, illustrating the influence of disturbances on the outputs. To calculate the gains from the two disturbances on the outputs, we employ finite difference by simulating the system and introducing a perturbation of 10^{-8} of the nominal value to the Gas-Oil Ratio (GOR) of well 1 and well 6. The resulting G_d^y matrix can be observed in Equation 4.16.

$$G_d^y = \begin{bmatrix} \frac{\delta y_1}{\delta d_1} & \frac{\delta y_1}{\delta d_2} \\ \vdots & \vdots \\ \frac{\delta y_{13}}{\delta d_1} & \frac{\delta y_{13}}{\delta d_2} \end{bmatrix} = \begin{bmatrix} 36.819 & 15.118 \\ -54.128 & 13.694 \\ -61.279 & 12.935 \\ 13.864 & 39.808 \\ 12.773 & -53.501 \\ 12.159 & -60.032 \\ 16.628 & 17.726 \\ 22.845 & 24.832 \\ 42.895 & -9.054 \\ 18.555 & -1.052 \\ -9.119 & 45.024 \\ -1.123 & 19.588 \\ 3.512 & 3.772 \\ 12.248 & 13.061 \\ 1.111 & 1.159 \end{bmatrix} \tag{4.16}$$

The procedure for obtaining G_d^y is outlined in Appendix B.2.

The subsequent step involves determining W_n, which represents the potential error associated with the variable measurements. The measurement errors considered for this analysis are ± 0.0025% for the pressure transmitters and ± 1% for the flow transmitters. In Equation 4.17, the structure of matrix W_n is provided.

$$W_n = \begin{bmatrix} y_{1_{err}} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & y_{15_{err}} \end{bmatrix} \tag{4.17}$$

The procedure of obtaining W_n is outlined in Appendix B.6.

Further on, W_d represents the magnitude of each disturbance. In Equation 4.18 the structure of W_d can be observed.

$$W_d = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix} \tag{4.18}$$

The implementation of finding W_d can be observed in appendix B.5.

To obtain the sensitivity matrix F and calculate the loss, we need to determine Juu, which is the Hessian matrix of the cost function with respect to the manipulated variables. Juu was computed using finite difference by keeping all manipulated variables constant except for the perturbed manipulated variables. The structure of Juu in this study can be observed in Equation 4.19.

$$J_{uu} \approx \begin{bmatrix} \frac{\delta^2 J}{\delta u_1^2} & \frac{\delta^2 J}{\delta u_1 \delta u_2} \\ \frac{\delta^2 J}{\delta u_2 \delta u_1} & \frac{\delta^2 J}{\delta u_2^2} \end{bmatrix} \tag{4.19}$$

The implementation of finding J_{uu} can be observed in appendix B.3.

Additionally, we need to determine Jud for the evaluation. Jud is the Hessian matrix of the cost function with respect to both the manipulated variables and disturbances. Jud was computed using finite difference by perturbing the manipulated variables and disturbances shown in Equation 4.20.

$$J_{ud} \approx \begin{bmatrix} \frac{\delta^2 J}{\delta u_1 \delta d_1} & \frac{\delta^2 J}{\delta u_1 \delta d_2} \\ \frac{\delta^2 J}{\delta u_2 \delta d_1} & \frac{\delta^2 J}{\delta u_2 \delta d_2} \end{bmatrix} \tag{4.20}$$

The implementation of finding J_{ud} can be observed in appendix B.4.

Using the derived matrices, we can compute the average loss and worst-case loss as discussed in Section 2.14. To achieve this, we employed the Branch and Bound algorithm to determine the measurement sets that yield the lowest average and worst-case loss. Subsequently, the calculated variables were exported to CSV files before being utilized in the Matlab functions for minimizing average loss Cao(2003) [34] and worst-case loss Cao(2003) [35].

The proposed measurement sets were then assessed using the simulator for the disturbances. However, due to the non-linearity of the model, perturbing the system did not yield satisfactory results due to sensitivity to initial values. Consequently, the optimizer faced difficulties in finding solutions when introducing a new nominal point in the unconstrained region. Therefore, the proposed controlled variables of Case 2 were evaluated solely in the active constraint region and not in the unconstrained region. Nevertheless, the proposed measurement combinations and their respective weights were tested for both negative and positive GOR changes, despite the shift in constraint regions. The optimal measurement set for different Branch and Bound (BAB) methods can be found in Table 4.20.

For each measurement combination, the sensitivity matrix F was calculated using Equation 2.34. The resulting sensitivities of each variable, as considered by the Branch and Bound algorithm, are presented in Table 4.19.

When the sensitivity matrix was obtained the H matrices where found using Equation 2.49. The H matrix of each measurement combination can be observed in table.

The procedure for obtaining H and F is outlined in Appendix B.9.

The controller tunings for GLC 2 and GLC 6 was found by open-loop step response and the SIMC method. The related controller tunings can be observed in the Table 4.21.

Table 4.18: Measurement combinations proposed by Branch and Bounds

Combination	BAB Method
p_{ai2} & p_{bh2}	Worst Case loss
p_{ai2} & p_{ai6}	Average loss
p_{d3} & p_{ai6}	Worst Case partial
p_{d3} & p_{ai6} & p_{bh6}	Worst Case partial
p_{d3} & p_{ai6} & p_{bh6} & p_{ai2}	Worst Case partial
p_{d3} & p_{ai2}	Average loss partial
p_{d3} & p_{ai2} & p_{bh2}	Average loss partial
p_{d3} & p_{wh6} & p_{bh6} & p_{ai2}	Average loss partial

Table 4.19: Sensitivity matrices found from linearized model.

Variables	F(d1)	F(d2)
p_{ai2}	-33.384	20.589
p_{bh2}	-87.774	8.683
p_{ai6}	6.034	-39.497
p_{d3}	-7.539	-22.520
p_{bh6}	25.358	-74.672
p_{wh6}	11.340	48.174

Table 4.20: Measurement combinations proposed by Branch and Bounds with related setpoints

Combination	H	c_{ns1}	c_{ns2}
p_{ai2} & p_{bh2}	$\begin{bmatrix} 3583 & -1449 \\ -146 & 67 \end{bmatrix}$	$1.65 \cdot 10^5$	$-5.57 \cdot 10^3$
p_{ai2} & p_{ai6}	$\begin{bmatrix} 413 & 326 \\ 305 & 663 \end{bmatrix}$	$7.51 \cdot 10^4$	$9.83 \cdot 10^4$
p_{d3} & p_{ai6}	$\begin{bmatrix} -6976 & 3401 \\ -8339 & 4481 \end{bmatrix}$	$-7.66 \cdot 10^5$	$-8.73 \cdot 10^5$
p_{d3} & p_{ai6} & p_{bh6}	$\begin{bmatrix} -234922 & 476847 & -181679 \\ -549333 & 1128132 & -431189 \end{bmatrix}$	$-1.41 \cdot 10^7$	$-3.24 \cdot 10^7$
p_{d3} & p_{ai6} & p_{bh6} & p_{ai2}	$\begin{bmatrix} -805763 & 131567 & 291161 & 427066 \\ -691151 & 1042352 & -313718 & 106098 \end{bmatrix}$	$-3.15 \cdot 10^7$	$-3.68 \cdot 10^7$
p_{d3} & p_{ai2}	$\begin{bmatrix} -2704 & 327 \\ -2675 & 68 \end{bmatrix}$	$-3.97 \cdot 10^5$	$-4.19 \cdot 10^5$
p_{d3} & p_{ai2} & p_{bh2}	$\begin{bmatrix} -41674 & -52288 & 23376 \\ -403800 & -541499 & 240611 \end{bmatrix}$	$-8.74 \cdot 10^6$	$-8.63 \cdot 10^7$
p_{d3} & p_{wh6} & p_{bh2} & p_{ai2}	$\begin{bmatrix} -686015 & -613181 & -334371 & 825709 \\ -659260 & -243106 & 98775 & -193401 \end{bmatrix}$	$-1.21 \cdot 10^8$	$-1.31 \cdot 10^8$

Table 4.21: Controller parameters Branch and Bound

Combination	K_C(GLC1, GLC6)	τ_I(GLC1, GLC6)	τ_C(GLC1, GLC6)
p_{ai2} & p_{bh2}	($1.1 \cdot 10^{-6}$,0.001)	(101,185)	(3000,3000)
p_{ai2} & p_{ai6}	($8.4 \cdot 10^{-5}$,$5.2 \cdot 10^{-5}$)	(67,65)	(300,300)
p_{d3} & p_{ai6}	($7.9 \cdot 10^{-7}$,$3.2 \cdot 10^{-7}$)	(264,163)	(4000,4000)
p_{d3} & p_{ai6} & p_{bh6}	($2.6 \cdot 10^{-8}$,$4.2 \cdot 10^{-9}$)	(274,133)	(4000,2000)
p_{d3} & p_{ai6} & p_{bh6} & p_{ai2}	($6.7 \cdot 10^{-9}$,$5.5 \cdot 10^{-9}$)	(169,137)	(2000,1500)
p_{d3} & p_{ai2}	($3.3 \cdot 10^{-6}$,$4.0 \cdot 10^{-6}$)	(226,258)	(2000,2000)
p_{d3} & p_{ai2} & p_{bh2}	($3.1 \cdot 10^{-5}$,$1.4 \cdot 10^{-8}$)	(1553,251)	(1553,3500)
p_{d3} & p_{wh6} & p_{bh6} & p_{ai2}	($9.3 \cdot 10^{-9}$,$5.5 \cdot 10^{-8}$)	(120,221)	(1000,600)

5 Results

This section will present the results of the implementation of the control structure and the findings from the case studies. The results will be discussed in this section, while the following section, Section 6, will provide a more general discussion.

To ensure a structured presentation of the results, the results section will be divided into four distinct parts. The first part will focus on examining and discussing the impact of the primary disturbance, namely the GOR, on the cost function. The second part will present the results obtained from the implementation of the regulatory control structures outlined in Section 3. The third part will address the findings obtained from the implementation of Case 1 described in Section 4.1.2. Lastly, the fourth part will delve into the implementation of Case 2 as discussed in Section 4.1.3.

5.1 Objective function change with GOR

In order to evaluate the potential economic losses associated with the implementation of the control structures, it is important to visualize the variation of the objective function with respect to the disturbance. In this particular scenario, the objective function is represented as the cost, while the disturbance corresponds to the fluctuations in the GOR.

To investigate the response of the cost function to the disturbance, we utilized IPOPT to solve the optimization problem. The disturbances considered ranged from -3.5% to +3.5% in the GOR of well 2, with a 0.1% interval, leading to a total of 70 iterations. It is assumed that any decrease or increase in the GOR of any well will influence the direction of the cost function similarly. Specifically, a decrease in GOR will result in a decrease in the cost function, while an increase in GOR will lead to an increase in the cost function.

Figure 5.1 displays a plotted curve illustrating the relationship between the cost function and GOR. The nominal GOR value is marked by a black X, nearly positioned at the boundary delineated by the orange dotted line, separating two regions with significantly distinct GOR-cost relationships. When the GOR value is smaller than the nominal point, a positive step generates a minor increase in cost, indicating a relatively low condition number for the function. However, upon examining the plot for GOR values larger than the nominal value, it becomes evident that a positive step yields a substantially greater increase in cost. This observation emphasizes a significantly higher condition number for the function within this particular region.

Figure 5.1 displays that the relationship between the cost function and the disturbance seems not linear and not flat in the constrained region. Below the nominal GOR value of well 2, the GOR-cost relationship exhibits close to linear behavior. When the GOR value reaches the constraint region around 0.130 kg/kg this close-to-linear relationship ceases. Once the system becomes constrained, meaning that the total produced gas is equal to 10 kg/s, the slope of the cost function decreases for a slim interval of GOR values before it gradually starts to increase again with increasing values of GOR. This behavior may be the result of the constraint affecting the feasible space. .

By analyzing how the cost function changes from left to right in Figure 5.1, we can observe that the cost increases as the GOR of the well increases. From a minimization perspective, this means that we are losing profit. This is expected from the definition of GOR, which is defined as the ratio between the produced gas and oil. An increase in the GOR will result in a production ratio of more gas and less oil, which results in a decrease in profit. Another effect of an increased GOR is the resulting increased pressure in the manifold. Such an increase in the manifold pressure can lead to reduced production of oil. However, we can observe that the relative change to the cost function is small compared to the change in the constrained region.

In the constrained region of the plot, the cost function exhibits non-linear behavior. When the produced gas constraint becomes active, and more gas is gradually introduced, the system will take

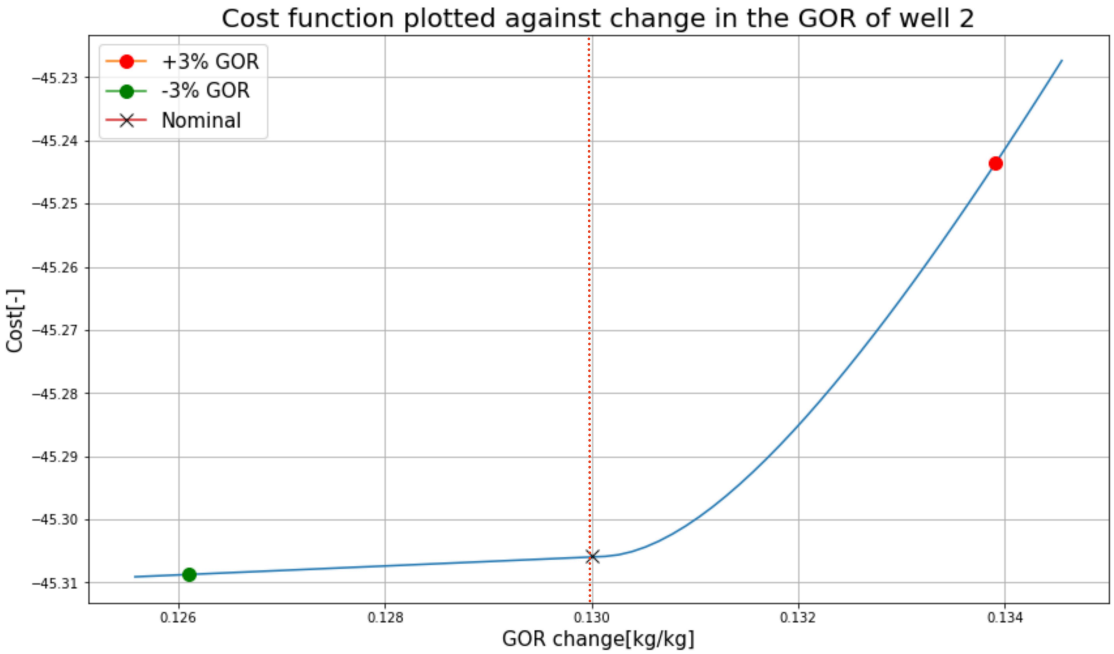


Figure 5.1: Objective function vs GOR change well 2.

measures to prevent the gas from being sent to export. In this model, the system will start by closing down the gas lift chokes to reduce gas production from the wells, which in turn will reduce oil production. The resulting production loss will have a great effect on the cost.

Another aspect, unrelated to the specific discussion on cost versus GOR, is the inherent non-linear nature of the system, which presents challenges in implementing local methods such as the nullspace method and the exact local method outlined in this paper. These local methods either linearize the function around the nominal operating point or estimate its behavior using analytical techniques. However, to ensure the validity of these methods, it is essential for the active constraint to remain constant, which is not the case in the study under consideration. Consequently, different measurements and combinations of measurements may be preferable in one region but not in another. To address this issue, the implementation of logical switchers or CV-CV selectors is recommended to ensure that the measurement combination yielding the least amount of loss is utilized, even when the constraint region undergoes changes.

5.2 Regulatory control results

5.2.1 Surge control

The implementation of surge control is crucial for the construction of a realistic model. As described in Section 2.21, the surge phenomena can damage the compressors, making it an undesired event. However, if we look at this from a modeling and optimization point of view, excluding surge constraints could potentially result in almost no flow through the compressor train, depending on the weights assigned to the variables in the objective function. If this were to happen, it would reduce the reality of the model and greatly limit the scope of work.

The implementation of the surge controllers for each of the three compressors is explained in Section 3.2.1. The system was simulated for 20000 iterations to test the proposed implementation using the IDAS integrator with CasAdi. At $t = 5000s$ the GOR was perturbed with a negative value, while at $t = 10000s$, the GOR was perturbed with a positive value of the same magnitude. Figure 5.2 consists of two plots showing the surge control implementation. The upper plot shows how the three PI controllers react to the disturbances, while the bottom plot shows how the flow through the compressors reacts to the disturbances and the control action.

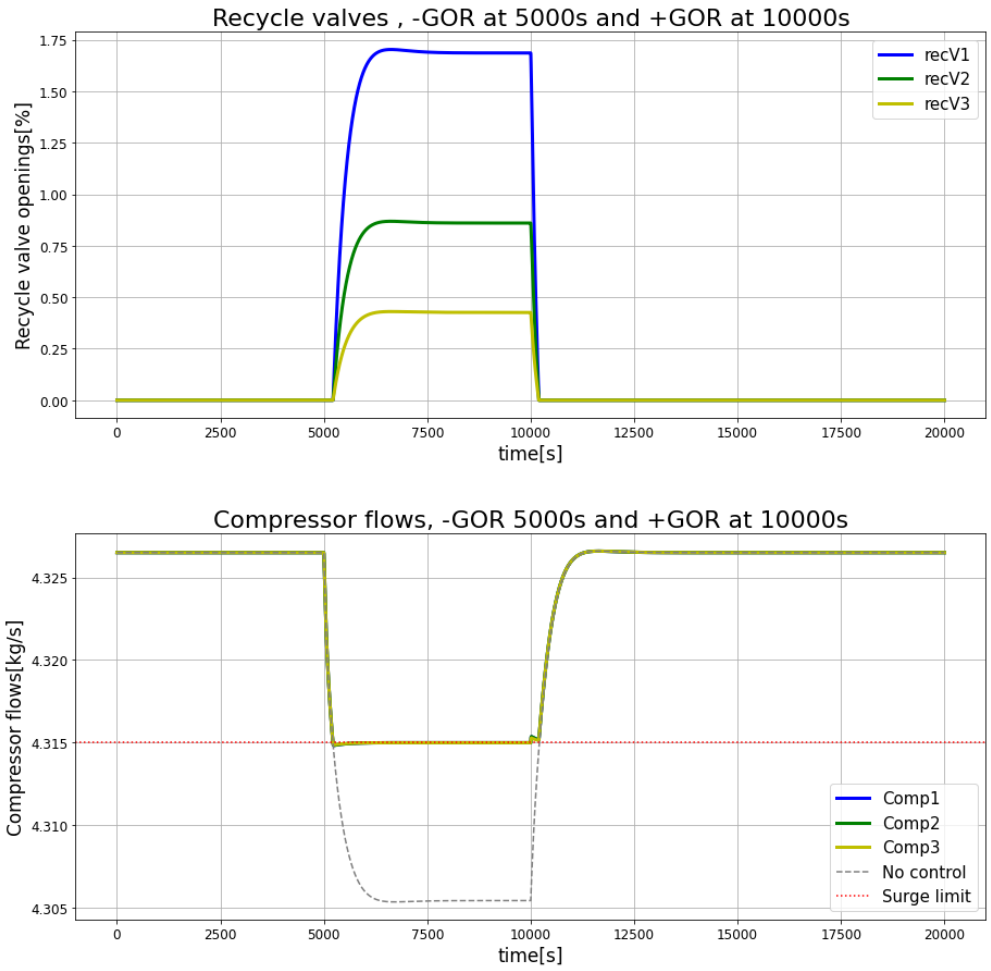


Figure 5.2: Results of surge implementation.

In the lower plot, the surge limit is depicted as a red dotted line. Based on the feedback received from the suction flows, the controllers promptly respond by opening the recycle valves if the flow through the compressors drops below the threshold determined by the relationship between the pressure ratio and the flow. In a real plant setting, a back-off strategy would be employed to ensure that the constraint is never violated, thereby mitigating the risk of compressor damage. However, for simulation purposes, we implement fast-responding controllers that react swiftly before stabilizing at the new steady-state, even though a slight violation of the constraint may occur within a small time window as can be observed in Figure 5.2.

In the simulations, we assume that a slight back-off is incorporated in the calculations of the surge limit. This is done to prevent the risk of surge in the event of a slight flow drop below the limit. While implementing back-off may result in unnecessary loss of recycled gas prior to technical necessity, it is crucial to prioritize safety by mitigating the potential risks and associated costs associated with surge.

From the bottom plot in Figure 5.1 we can also observe that the response of the controllers is as expected from the implementation described in Section 3.2.1. The first disturbance occurs at time $t = 5000s$, and represents a decrease in GOR, meaning that the fraction of gas in the mixture decreases. Consequently, the total amount of gas in the system will decrease, and since the gas lift functions as a recycle of the produced gas, the gas flow into the compressor train will decrease.

When the gas flow through the compressor train falls below the surge limit, the controllers will react instantly by opening the recycle valves, to avoid damage to the compressors. We can observe from the plot that this control response is rapid, and the controllers manage to control the flow back to

the surge limit value. We can verify that the controls are effective by comparing the results of the controlled functions with the grey dotted line, which shows how the flow would change without any control. Since recycling compressed gas demands a great deal of energy, it is ideal to keep recycle flow as small as possible, to minimize the use of energy and economic losses. For this reason, the setpoints of the controllers will be the surge limit in this case.

At $t = 10000s$ the system faces a new disturbance, which represents a positive change in GOR with the same magnitude as the previous negative disturbance. The system will then experience an increase in the total amount of gas in the system, which leads to increased flow through the gas lift recycle system. Since the magnitude of the positive disturbance in this case is equal to the previous negative experience, the increased level of gas through the compressor train is higher than the surge limit. There is no longer a need for the recycle valves to be open, and to save economic losses the valves are closed by the controllers. We can observe from the plots in Figure 5.1 that the controllers close the recycle valves instantly after the compressor flow exceeds the surge limit. This is achieved through the implementation of a logic block that utilizes information from the valve position and measured flow to determine whether the valves should be closed down.

As we can observe from the bottom plot, at the time when the second disturbance is introduced to the system the flow through the compressors will increase and decrease slightly before the flow increases in a significant matter. This reverse effect is the result of the recycle valves closing down and thus reducing the flow through the compressors, which for a short time period decreases the gas flow through the compressors before the flow will gradually increases back to the starting point.

In the upper plot in Figure 5.1 we can observe that the different recycle valves open to different extents. Recycle valve 1 opens approximately twice as much as recycle value 2, and recycle valve 2 opens approximately twice as much as recycle valve 3. This effect is explained by the fact that each compressor and valve are designed equally. Since the compressors are arranged in a series, the pressure and suction flow will vary for each compression stage due to the output of one compressor becoming the input of the next compressor. The pressure ration of the compressors is approximately 2, meaning that the relationship between the suction pressure and the discharge pressure of one compressor will be approximately 1:2. The density of a gas can be found from the ideal gas law. From this equation we can observe that if the pressure increases by a factor of two, the density will also change by the same magnitude, given constant temperature and mass flow.

Based on the results obtained from the implementation of surge control, we can conclude that the controllers associated with the recycle valves effectively protect the compressor train from the potential hazards of surge. Moreover, the controller logic ensures that there is no unnecessary recompression of already compressed gas, ensuring no waste of energy.

5.2.2 Produced gas control

In the case study described in Section 3.1.1 the optimal nominal operating point is constrained by, among others, the total amount of produced gas, at 10 kg/s. Based on both safety considerations and the theory on active constraint regions presented in Section 2.17, it is considered necessary to control the total amount of produced gas. This is because we do not want the flow to exceed the capacity of the equipment in the "gas export" part of the production system, which is responsible for further processing and exportation of the gas.

From a self-optimizing perspective, controlling the active constraint is beneficial if it is optimally active for the new operating point. Section 2.4 explains that good self-optimizing variables should not be sensitive to changes in the disturbance, and consequently, the active constraint will be a good CV when the constraint region stays constant.

To show how the implementation of produced gas control works, a test was simulated with the use of split-range control with the batton strategy. The system was simulated with the IDAS integrator

for $t = 30000s$, with a disturbance of increased GOR in well 2 at $t = 5000s$. The result of the simulation is shown in Figure 5.3.

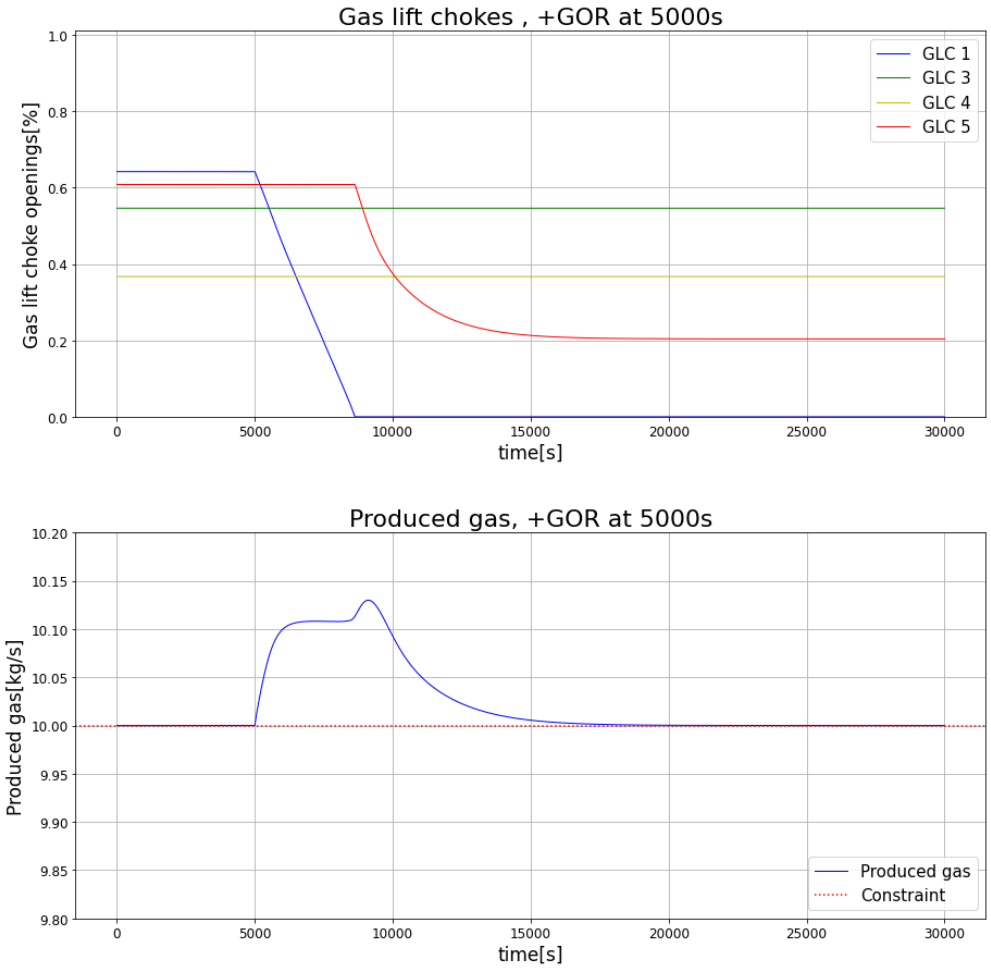


Figure 5.3: Results of produced gas control.

In the bottom plot in Figure 5.3 we can observe that produced gas starts to increase at $t = 5000s$. From the upper plot, we can observe that the controllers react to the disturbance almost instantly by acting on the gas lift chokes to counteract the increase in total produced gas. The chokes start to close down one by one to reduce the amount of gas lift going into the wells, resulting in a decrease in production.

The order in which the gas lift chokes are manipulated is according to the order defined by the method in Section 3.2.2. This order ensures that the wells with the lowest oil production rate will be manipulated first, to minimize the economic losses from the disturbance as much as possible. We can observe from the upper plot in the figure that the first gas lift choke that reacts is the gas lift choke of well 1 (GLC 1), which is the well with the lowest oil production rate. Since GLC 1 saturates and closes completely, the second gas lift choke in the predefined order, GLC 5, must supplant the manipulation, while GLC 1 is kept closed.

In the bottom plot, we can observe that the total produced gas is at steady-state before the disturbance occurs at $t = 5000s$. The disturbance makes the total amount of produced gas increase, which triggers the control of GLC 1. GLC 1 is manipulated to counteract this disturbance and manages to stop the increase in produced gas. However, GLC 1 saturates before the total amount of produced gas reaches the constraint limit, which forces GLC 5 to take over further manipulation at approximately $t = 8000s$. At this time, we can also observe a sudden increase in the total amount of produced gas in the bottom plot. This inverse response occurs when GLC 1 saturates and GLC

5 becomes active, which is expected due to the recycle effect of the gas in the system. Moreover, GLC 5 successfully controls the produced gas and brings it back to the constraint, demonstrating the effectiveness of the implemented approach.

To ensure that the control structures are controllable the total amount of produced gas is considered a soft constraint in the simulations. This is both due to the initial inverse response of the gas lift chokes on the total produced gas, which will be discussed in the paragraph below, and to reduce the interaction between the different controllers. The interaction between the controllers that control the produced gas and the controllers that control measurement combinations poses significant difficulties in implementing decentralized control. In order to prevent these difficulties from affecting the control system, the controller time of the different controller types has been manipulated to operate on different time scales.

Based on the results obtained from the produced gas control, it is evident that the implemented split-range controller with the baton strategy successfully controls the total produced gas by transferring control from one manipulated variable to another. However, it is important to note that for cases where the constraint is considered "hard", a more suitable choice would likely be to use a production choke for controlling the active constraint.

5.2.3 Level Control

The implementation of level control was described in Section 3.2.4. In this work, we proposed two methods for controlling the level in the separator, which is an inherit unstable system. We can either control the level by controlling it at a constant setpoint or by defining boundaries where the level is allowed to change. Despite which method of level control we choose, it is important to ensure stable and safe operation. The results from each type of level control are presented below.

5.2.3.1 Level control - constant setpoint

To be able to control the level in the separator to a constant setpoint, a feedback control structure measuring the level in the separator was implemented to manipulate the valve at the oil outlet. The result of level control of the separator using a PI controller and a constant setpoint can be observed in Figure 5.4. The upper plot shows the controller's response to the disturbance, and the bottom plot shows the flows in and out of the separator. The system was simulated for $t = 50000s$, using the IDAS integrator with a positive change in the GOR of well 2 as a disturbance at $t = 10000s$.

In the upper plot, we can observe how the controller responds to the disturbance by manipulating the valve at the oil outlet of the separator. Due to the valve being physically close to the separator, the controller will be efficient in controlling the liquid level by adjusting the flow of oil going out of the separator. We can observe from the upper plot that when the disturbance occurs, the controller reacts by starting to close the valve.

The bottom plot shows the change in oil flow in and out of the separator, and the change in the flow through the OSC. When the disturbance occurs at $t = 10000s$, we can observe that both the oil flow in and out of the separator increases for a small time period, before they decrease, and gradually settle at a steady-state value lower than the initial value. The inverse response that occurs when the disturbance is introduced is a result of a change in the manifold pressure, which in turn leads to changing effects on the production rates from the wells. However, after an initial reverse response, the total oil flow will decrease, as expected. The controller will force the valve to adjust so that the outlet flow of oil from the separator always matches the inlet flow of oil from the riser.

The results of the implementation shows that the controller manages to efficiently control the level at the provided setpoint.

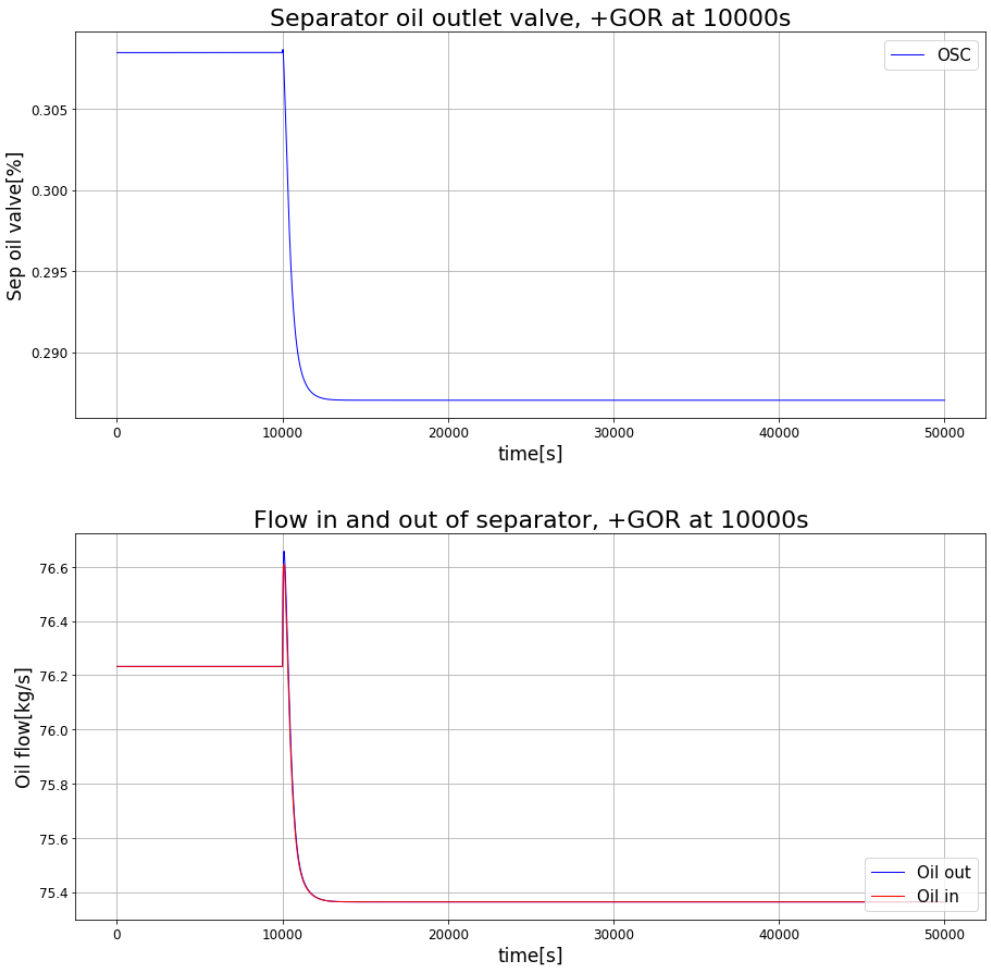


Figure 5.4: Control of the active constraint on total produced gas.

5.2.3.2 Level control - Boundaries

The second proposed control structure only controls the level of the separator if the level approaches defined high-high (HH) or low-low (LL) limits. The level in the separator is allowed to move "freely" between these limits. The result of controlling the separator level to be inbetween two defined boundaries using a PI controller is shown in Figure 5.5. The controller's response to the disturbance is shown in the upper plot, and the flow in and out of the separator is shown in the bottom plot. The system was simulated for $t = 50000s$, using the IDAS integrator with a positive change in the GOR of well 2 at $t = 10000s$ and a negative change in the GOR of well 2 at $t = 30000s$.

We can observe from Figure 5.5 that the controller shown in the upper plot remains inactive until the level of the separator reaches the defined LL at 0.8 meters. At the LL, the controller reacts by decreasing the opening of the valve, so the level in the separator doesn't decrease below the lower permitted boundary. Both the valve opening and the level in the separator reach a new steady-state quickly, where the oil in and out of the separator are equal. At $t = 30000s$ a second disturbance occurs. This disturbance has a negative change in the GOR of well 2 with a larger magnitude than the first disturbance. The controller remains inactive until the level reaches the defined HH limit, and proceeds to control the level to not exceed this constraint.

Based on the implementation of boundary-based control for the separator level, the results demonstrate that the controller effectively maintains the level of the separator within the permissible limits.

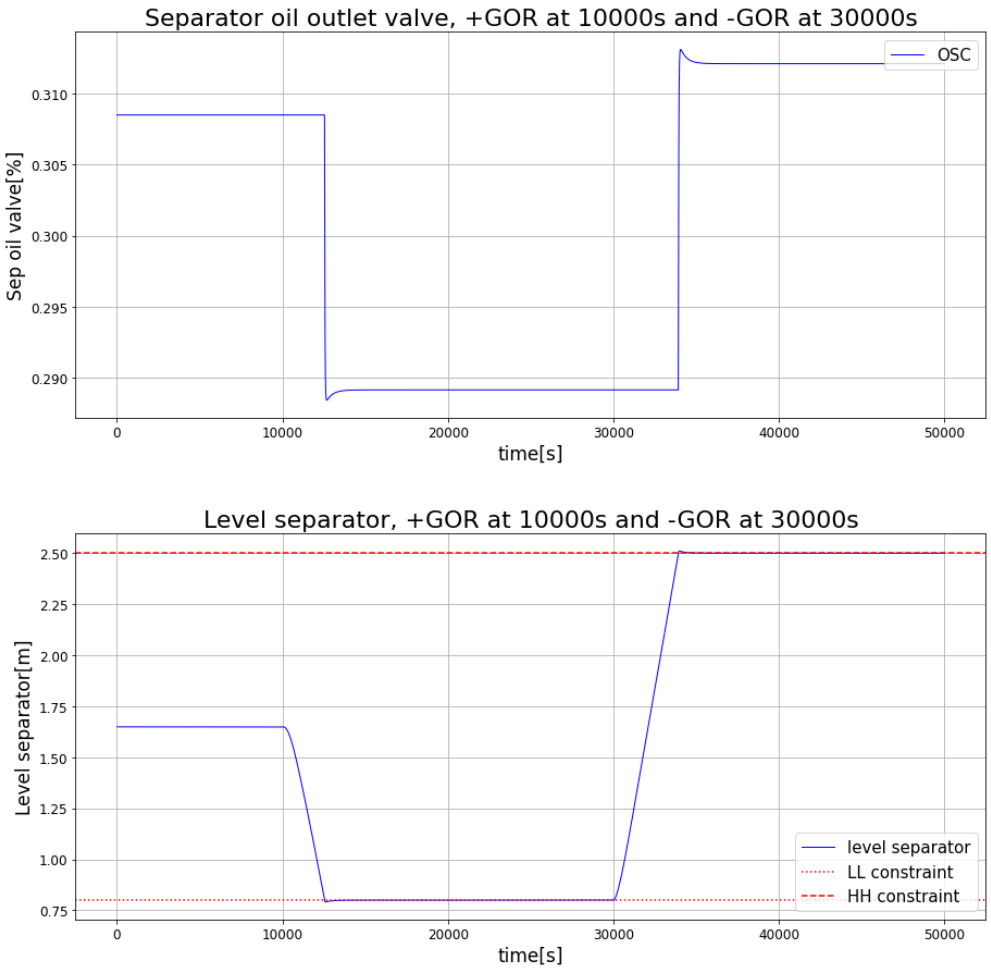


Figure 5.5: Constant control of separator pressure.

5.2.4 Valve position control

The implementation of valve position control (VPC) was proposed as a solution for controlling measurement combinations, as discussed in Section 3.2.5. This approach was necessary because the MV experienced saturation before reaching the desired setpoint when attempting to control certain measurements and combinations. In this study, it was observed that the saturation of the MV could be mitigated by introducing a secondary MV that would control the primary MV towards its optimal nominal value. This resulted in either resetting the valve position or allowing the secondary MV to saturate before reaching the optimal value, effectively preventing saturation of the primary MV.

To illustrate the effectiveness of the VPC implementation in controlling the proposed measurements, we conducted tests on the discharge pressure of compressor 3. The results of using VPC in the control of the discharge pressure of compressor 3 are presented in Figure 5.6. The system was simulated for a duration of $t = 50000s$, with a positive change in the gas-oil ratio (GOR) of well 2 occurring at $t = 10000s$

Figure 5.6 demonstrates the implementation of valve position control (VPC) by GLC 3 to prevent the saturation of GLC 2. The resulting control of the discharge pressure in compressor 3 is depicted in the bottom plot. At $t = 10000s$, a positive disturbance occurs in the GOR of well 2. As a consequence, the discharge pressure increases due to the rise in system pressure caused by the increased gas flow.

Upon examining the upper plot, it is evident that GLC 2 initiates an opening action to reduce the

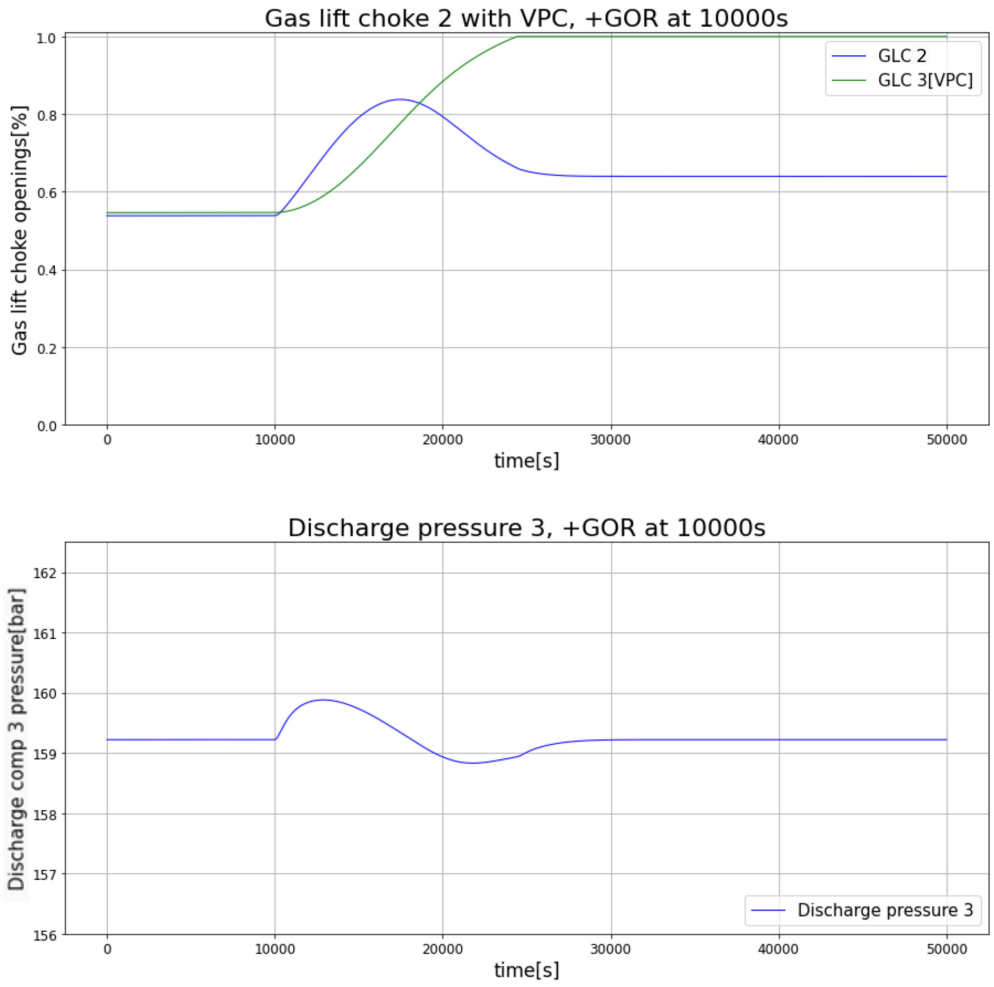


Figure 5.6: Result of implementing VPC to control the discharge pressure of compressor 3.

discharge pressure. Simultaneously, GLC 3, which is designed with a controller time approximately four times longer than GLC 2 to minimize interaction and oscillations, responds by countering the opening of GLC 2 and attempting to regulate it back to the setpoint. Around $t = 17000s$, GLC 2 begins to return to its nominal setpoint.

The combined efforts of GLC 2 and GLC 3 result in a slight undershoot in the discharge pressure before GLC 3 reaches saturation. Subsequently, GLC 2 takes over and successfully controls the discharge pressure, guiding it towards the setpoint value. Ultimately, the system stabilizes at a new steady-state with the discharge pressure of compressor 3 under the control of GLC 2.

To validate the implementation, it is necessary to examine the behavior of the discharge pressure control using GLC 2 without the utilization of VPC. A new simulation was conducted, following a similar setup as the one depicted in Figure 5.6, but excluding the implementation of VPC. The comparative scenario is presented in Figure 5.7.

The comparative simulation shown in Figure 5.7 provides insight into the consequences of not employing VPC in controlling the discharge pressure of compressor 3. It is evident from the figure that GLC 2 reaches saturation at approximately $t = 18000s$, resulting in a loss of control over the discharge pressure.

In this specific case, the utilization of VPC was necessary to effectively control the amount of produced gas and maintain it at its initial value. Based on these findings, it can be concluded that the implementation of VPC may be essential for controlling certain CVs when there is limited control gain from the MV or conflicting control requirements between multiple controllers.

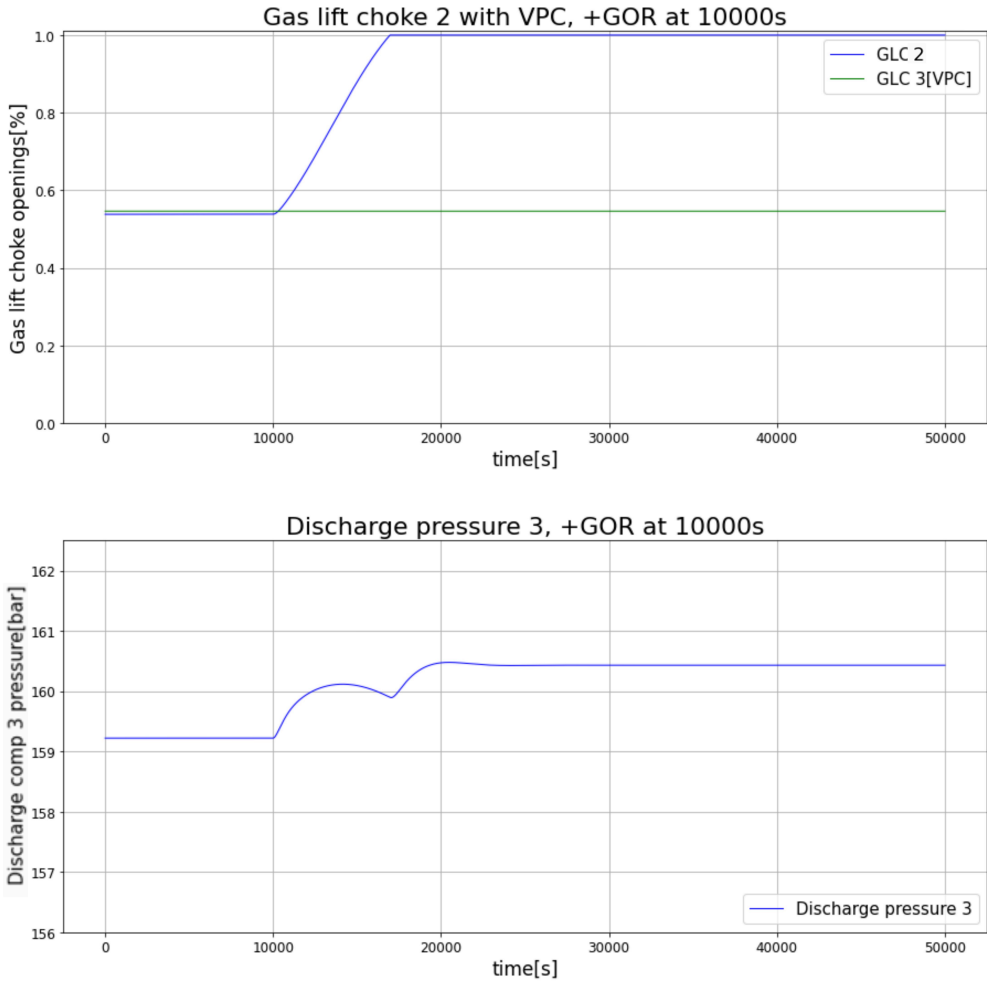


Figure 5.7: Result of only using GLC 2 in the control of the discharge pressure of compressor 3.

5.2.5 Changing constraint regions

Given that oil production is a dynamic process that is susceptible to various challenges and changes, it is likely that the system will need to operate within different active constraint regions. The regions relevant in this thesis are defined based on the activation and deactivation of the constraint on total produced gas. The nominal operating point is located on the boundary between the regions where the constraint is active.

When operating within an active constraint region, it is generally advisable to control the active constraints. However, it is not desirable to do so if the constraint is not optimally active, as it can result in significant losses. In Section 3.2.3, we propose a CV-CV switching mechanism utilizing a minimum selector. This mechanism aims to address this issue and ensure optimal control of the active constraints when necessary.

To show how the min selector and the change between constraint regions are implemented in the model, a simulation of the system for $t = 60000s$ is solved with the IDAS integrator. In the simulation, a positive change in the GOR of well 2 occurs at $t = 5000s$, and a negative change of the same magnitude is introduced at $t = 30000s$. The simulation results can be observed in Figure 5.8.

The result of the control with min selector logic can be observed in the upper plot of Figure 5.8. From $t = 0s$ to $t = 30000s$, the controller output will be identical to the produced gas control discussed in Section 5.2.2. However, at $t = 30000s$, a negative GOR change affects the system.

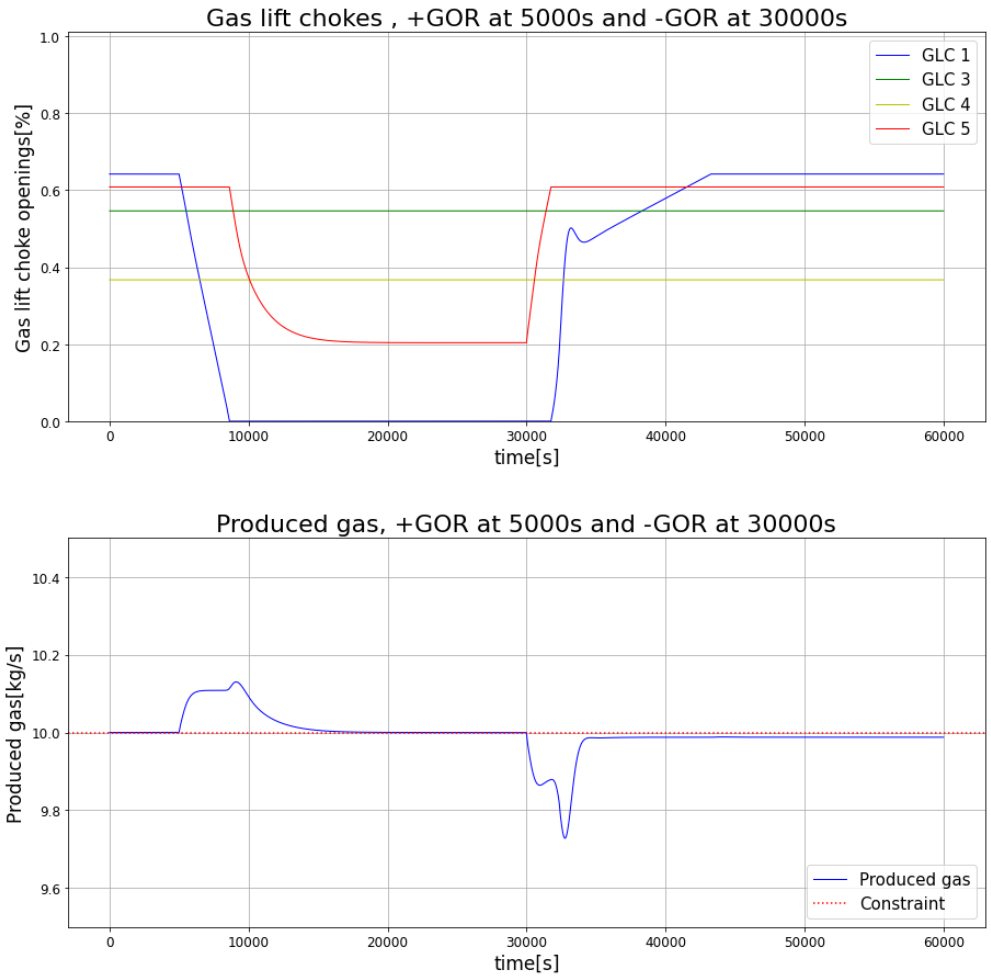


Figure 5.8: Control of changing active constraint regions.

Based on the min selector logic described in Section 3.2.3, the controller with the smallest magnitude output is chosen for implementation in the system. When the constraint is active, GLC 5 is responsible for controlling the total produced gas, and consequently, it will have the smallest proposed input among the controllers.

At $t = 30000s$, when the negative disturbance in GOR affects the system, GLC 5 will initiate control to bring the produced gas back to 10 kg/s by opening up and increasing gas production from the well. This process continues until GLC 5 reaches its nominal point, at which point the controller with the smallest proposed input will be the one with the nominal opening as its setpoint. This leads to a reset of the valve opening to its optimal nominal value. Once this condition is met, GLC 5 passes control back to GLC 1, which then aims to control the produced gas to its constraint. The same logic is implemented, and GLC 1 returns to its optimal nominal value. We can observe an inverse response behavior once again, where the controller of GLC 1 briefly moves the valve opening in the wrong direction. However, it quickly corrects itself and brings the valve back to its optimal nominal value.

If the overall amount of total gas produced falls below the active constraint value, the min selector will thus release control of the total produced gas.

If gas production is not controlled optimally, it can lead to significant losses. The related loss of controlling the bottom hole pressure to its nominal value in the unconstrained region, both with and without control of the active constraint, can be observed in Table 5.1. The simulation was run for $t = 100000s$ to ensure that the system converged with a disturbance of -3% decrease in the GOR of well 2. At the disturbance, the optimal nominal value of the total produced gas is not

constrained.

Table 5.1: Table of related loss to controlling the active constraint and not in the unconstrained region.

Active constraint control	objective function	loss[$/month]	loss[%]
Yes	-45.2386	42310	0.1544
No	-45.3086	28	0.0001039

From Table 5.1 it is clear that giving up the control of the active constraint is not economically beneficial. Another observation from the table is that the loss is small in the unconstrained region. This can also be observed in Figure 5.1, where the objective function has a linear change of small magnitude. Based on this small loss we can also conclude that controlling the valve openings back to their optimal nominal setpoints is sufficient in this case. However, in other cases where the change in the objective function in the unconstrained region has a larger magnitude, controlling other potential CVs with new unconstrained MVs should be considered.

Based on the findings presented in this section, it can be inferred that the min selector effectively controls the total produced gas within the active constraint region. In the unconstrained region, it efficiently regulates the valve openings to their optimal nominal values. Furthermore, the implementation of this control logic has demonstrated significant economic advantages.

5.3 Results of Case 1

As described in Section 4.1.2 we have studied the effect of three different control methods and their performance compared to the effect of using optimizing control. The first method is an initial study, which evaluates only single variables. The two subsequent methods use the results from the single controlled variable evaluation to achieve the most effective control structure.

We have considered two different scenarios to assess the losses we get from applying a control structure instead of using an optimizer. The first scenario, where the optimizer considers all six GLCs, will be denoted (1). The second scenario, where the optimizer only considers the GLCs that are modified during the simulation, will be denoted (2). The cost-value used to evaluate the losses is the value of the solution when the system has reached steady-state after experiencing a disturbance. The negative terms of the cost function are because the problem considered is a minimization problem.

5.3.1 Single controlled variable

To test the implementation of the controllers obtained in Section 4.5, the system was simulated for $t = 100000s$. This was done to ensure convergence and that the system reached its new steady-state. The level was controlled to a constant setpoint to ensure both stability in the system and quicker convergence. The initial tuning parameters found from the SIMC method described in Section 2.11 were manipulated to achieve better control and less interaction. The result of the control of the proposed CVs can be observed in Appendix A.2.

5.3.1.1 GOR increase in well 2

The results of the simulations for an increase in the GOR of well 2 with +3% can be found in Table 5.2. The optimal cost at this operating point was found to be:

(1) -45.2406 $/s

(2) -45.1587 $/s

Based on the simulation results presented in Table 5.2, it is evident that maintaining most variables at their optimal nominal points leads to greater losses compared to keeping GLC 2 at its designated setpoint. To elucidate these findings, we can assess the optimal adjustments of various variables

Table 5.2: The results of controlling the proposed CVs to their optimal nominal value facing a disturbance of +3% in the GOR of well 2. Variables with(*) are assisted with VPC. L denotes the loss compared to optimal operating points.

Variable	Cost [$/s]	L [$/w] (1)	L [%] (1)	L [$/w] (2)	L [%] (2)	Tot. gas [kg/s]
p_{wh2}	-44.9505	175414	0.6411	125627	0.46	10
p_{bh2}	-45.0051	142413	0.5204	92692	0.34	10
p_{ai2}	-44.9619	168542	0.6159	118768	0.43	10
p_{gs}*	-44.9369	183657	0.6712	133851	0.49	10
p_m*	-44.8059	262882	0.9608	212913	0.78	10
p_{d3}*	-44.9582	170745	0.6240	120958	0.44	10
GLC2const	-44.9709	163119	0.5961	113357	0.42	10

from the nominal operating point to the new operating point with a +3% GOR for well 2. Table 5.3 presents these results.

Table 5.3: The optimal values of the CVs at the nominal operating point and at the new operating point (GOR W2(+3%)).

Variable	Nominal op. point	GOR W2(+3%)	Gain from MV	Unit
p_{wh2}	80.6189	79.5110	2.7678	bar
p_{bh2}	137.2310	137.5890	-6.2921	bar
p_{ai2}	101.5360	100.0610	5.3474	bar
p_{gs}	21.8954	21.8954	0.0037	bar
p_m	78.6830	77.7475	0.7678	bar
p_{d3}	159.2200	167.4810	12.2317	bar

From the data in Table 5.3, it is evident that, apart from the separator pressure (p_{gs}), the bottomhole pressure exhibits the smallest variation. The control of separator pressure proves challenging due to its limited sensitivity to GLC 2, as well as its dependence on VPC, which may introduce greater losses. Additionally, the oversized separator might obscure the results. The value of p_{gs} relies on the interplay between gas and oil flows into and out of the separator. Although we initially anticipated a greater increase in pressure during optimization, the observed change remains minimal in comparison to the simulation results within the active constraint region. This discrepancy may be attributed to the fact that elevating the separator gas pressure reduces production, including oil output, consequently resulting in a profit loss. Nonetheless, upon completing the simulations, it became evident that the separator pressure does indeed exhibit a more substantial magnitude of change. Therefore, due to these peculiar dynamics, we excluded the separator pressure from further considerations.

Generally, variables that exhibit lower sensitivity to disturbances tend to yield better CVs. However, the impact of MVs on the CV is also a crucial consideration. We observe a correlation between variables requiring VCP and the gain from the MV. This implies that changes in the MV will have less impact on these CVs and may necessitate additional assistance. Consistent with the pair-close rule and the system model, the well parameters and discharge pressure exhibit the highest gains. Conversely, the manifold and separator pressures, located differently from the gas lift choke, pose greater control challenges. The previous section already discussed the case of the discharge pressure, where GLC 2 demonstrates significant gain on the variable. The difficulty arises from all gas lift chokes having the same gain on it, making it challenging to control when multiple controllers manipulate the valves in different directions.

Based on the results, the bottomhole pressure is the most controllable variable. However, the overall loss is significant due to constrained operating points when the GOR increases. To compare with methods like Dynamic RTO and MPC, which utilize all gas lift valves, using only one manipulated

variable for SOC results in substantial losses during rapid system changes. Employing multiple MVs for control and its limitations will be discussed later.

5.3.1.2 GOR decrease in well 2

The results of the simulations for a decrease in the GOR of well 2 with -3% can be found in Table 5.4. The optimal cost at this operating point was found to be:

(1) -45.30861 $/s

(2) -45.30857 $/s

Table 5.4: The results of controlling the proposed CVs to their optimal nominal value facing a disturbance of -3% in the GOR of well 2. Variables with(*) are assisted with VPC. L denotes the loss compared to optimal operating points.

Variable	Cost [$/s]	L [$/w] (1)	L [%] (1)	L [$/w] (2)	L [%] (2)	Tot. gas [kg/s]
p_{wh2}	-45.3085	58	0.00021	32	0.00012	9.9525
p_{bh2}	-45.3085	28	0.00010	3	0.00001	9.9526
p_{ai2}	-45.3045	2371	0.00865	2344	0.00855	9.9524
p_{gs}*	-	-	-	-	-	-
p_m*	-45.3037	2942.6	0.01074	2916	0.01100	9.9531
p_{d3}	-45.3077	512	0.00187	487	0.00180	9.9526
GLC2const	-45.3078	514	0.00187	488	0.00178	9.9527

From the results in Table 5.4, it is evident that the loss in the constrained region surpasses that in the unconstrained region. Furthermore, certain implementations lead to larger losses compared to having no control. Specifically, attempting to control the separator pressure using the GLC for a negative disturbance proved infeasible in this case. Despite employing VPC, no significant effect was observed. This failure can be attributed to the "pair-close rule" arising from the low gain of the MVs on gas production, resulting in minimal influence of GLC 2 on the CVs. The challenge stems from the requirement to increase separator pressure, which necessitates increased gas production to compensate for the disturbance-induced decrease. Consequently, GLC 2 opens up to enhance well production. However, this causes an increase in manifold pressure, thereby reducing production from other wells. Introducing additional VPCs or placing an MV closer to the separator may resolve this issue. However, controlling the separator pressure becomes infeasible when the GOR of well 2 decreases. The dimensions of the separator may also contribute to this limitation. From a practical standpoint, attempting to control the separator pressure by increasing it for a negative GOR is not economically viable, as it would potentially decrease overall production. This observation aligns with simulations where increasing the separator pressure coincides with a decrease in GOR.

The optimal value of the variables at nominal operation and the new operation point(-3% GOR well 2), can be observed in Table 5.5.

Table 5.5: The CVs optimal values at the nominal operating point and at the new operating point.

Variable	Nominal	GOR W2(-3%)	Gain from MV	Unit
p_{wh2}	80.6189	80.6214	2.7678	bar
p_{bh2}	137.2310	137.2220	-6.2921	bar
p_{ai2}	101.5360	102.0160	5.3474	bar
p_{gs}	21.8954	21.8791	0.0037	bar
p_m	78.6830	78.6827	0.7678	bar
p_{d3}	159.2200	158.3930	12.2317	bar

From Table 5.5, it is evident that the variables least sensitive to the disturbance are the manifold pressure (p_m) and bottomhole pressure (p_{bh2}). However, controlling the manifold pressure results

in the highest loss among all potential control structures. This can be attributed to the inherent difficulty in controlling manifold pressure, which requires VPC. Notably, the loss associated with not implementing any control is relatively low, and the MV values remain close to their original openings. When two MVs experience significant changes, such as opening or closing, the interconnection within the system, where gas lift is shared among multiple wells from the same manifold, can lead to increased gas lift in other wells.

Based on the results, it can be concluded that controlling the bottomhole pressure at its optimal nominal value is the most effective control strategy. However, the overall loss is relatively insignificant. It is important to note that if only GLC1, GLC2, and the oil outlet valve are available as MVs, the objective function value is -45.30858049.

5.3.1.3 Proposed overall control structure single CV

From the results of both negative and positive change in the GOR of well 2 around the optimal nominal point we can conclude that controlling the bottomhole pressure is the best implementation in both regions. To show how the proposed control structure for the whole plant in the case where we propose controlling the single measurement with one MV can be observed in Figure 5.9.

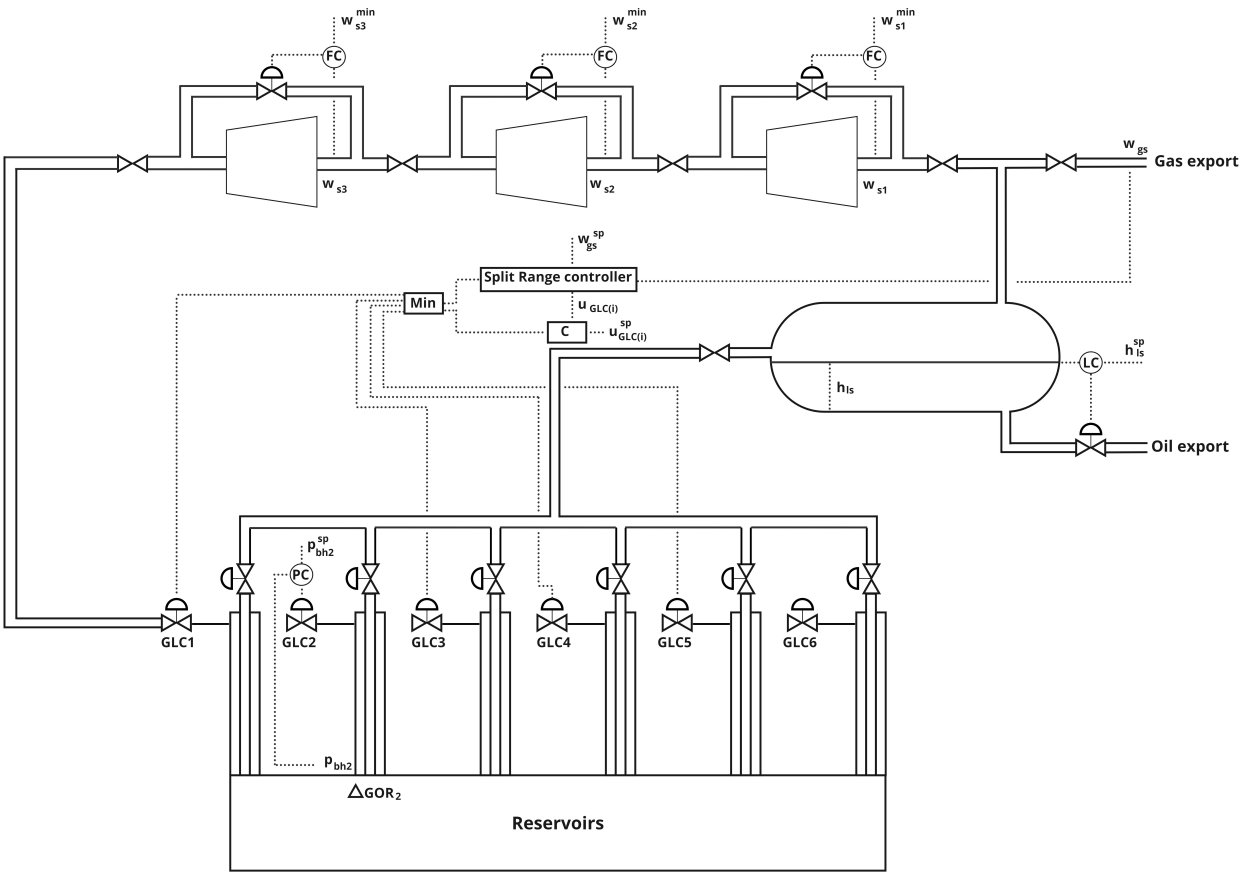


Figure 5.9: Control of the productionsystem with one MV used for SOC.

5.3.2 Null space method

The next step in implementing a self-optimizing CV was to introduce combinations of measurements between the potential CVs discussed in Section 4.1.2.2. Based on the results from controlling a single measurement, as presented in Section 5.3, we proposed using the measurement with the least loss to form combinations of two variables. This choice aligns with the conditions of the nullspace method outlined in Section 2.5, which does not account for measurement errors. The measurement combinations for the bottomhole pressure can be found in Table 4.4.

To test the controllers identified in Tables 4.8 and 4.9, along with the corresponding setpoints from Tables 4.6 and 4.7, simulations of the system were conducted for a duration of $t = 100000s$. This was done to ensure convergence and reach a new steady-state of the system. Similar to the previous case, the separator level was controlled to a constant setpoint. The initial controller tunings were adjusted to achieve improved performance. The results of controlling the proposed CVs can be observed in Appendix A.3.

5.3.2.1 GOR increase in well 2

The results of the simulations for an increase in the GOR of well 2 with +3% can be found in Table 5.6. The optimal cost at this operating point was found to be:

(1) -45.2406 \$/s

(2) -45.1587 \$/s

Table 5.6: The results of controlling the measurement combinations to their optimal nominal value facing a disturbance of +3% in the GOR of well 2. L denotes the loss compared to optimal operating points.

Variable	Cost [\$/s]	L [\$/w] (1)	L [%] (1)	L [\$/w] (2)	L [%] (2)	Tot. gas [kg/s]
p_{bh2} & p_{wh2}	-45.0034	143464	0.5243	93736	0.34	10
p_{bh2} & p_{ai2}	-45.0035	143417	0.5241	93681	0.34	10
p_{bh2} & p_m	-45.0034	143467	0.5243	93739	0.34	10
p_{bh2} & p_{d3}	-45.0038	143255	0.5235	93517	0.34	10
No control	-44.9709	0.59616	113357	113357	0.42	10

Table 5.6 shows that controlling all measurement combinations leads to lower losses compared to having no control. This aligns with the H matrices obtained from the combinations presented in Tables 4.6 and 4.7. The H matrix derived from the left nullspace of the sensitivity matrix determines the weighting assigned to each individual measurement in the combination. Since the bottomhole pressure exhibits lower sensitivity to disturbances than the other potential measurements, as evident in Table 4.5, it dominates the expression.

Another observation is that all measurement combinations result in higher losses compared to solely controlling the bottomhole pressure. This can be attributed to the local nature of the nullspace method, as explained in Section 2.5. This method identifies the combination of available measurements that minimizes the loss based on the local conditions around the nominal point. However, this highly nonlinear model, as discussed in Section 5.1, may exhibit different behavior as the operating point deviates further from the nominal point. This is the case for the system evaluated in this paper.

The best combination proposed by the nullspace method is the bottomhole pressure-discharge pressure of compressor 3 combination. However, the losses for each combination are nearly indistinguishable due to the dominant influence of the bottomhole pressure.

5.3.2.2 GOR decrease in well 2

The results of the simulations for a decrease in the GOR of well 2 with -3% can be found in Table 5.7. The optimal cost at this operating point was found to be:

(1) -45.3086 \$/s

(2) -45.30857 \$/s

It is important to note that the nullspace method requires the active constraints to remain unchanged. However, due to the nominal point being located exactly at the boundary, even a slight decrease in the GOR can cause the operating point to move out of the constraint region. To address

this issue, we propose adjusting the nominal operating region through small parameter changes. This allows us to start in the unconstrained region, making the method valid.

Additionally, the plot in Figure 5.1 demonstrates that in the unconstrained region, the change is approximately linear up to the nominal point where the constraint becomes active. Therefore, we suggest obtaining the sensitivity matrix F in this case by introducing a small negative disturbance in the GOR. From a practical standpoint, the system should be analyzed in all different constraint regions, and the most optimal measurement combination should be implemented for each region. However, identifying all possible constraint regions is beyond the scope of this project.

Table 5.7: The results of controlling the measurement combinations to their optimal nominal value facing a disturbance of -3% in the GOR of well 2. L denotes the loss compared to optimal operating points.

Variable	Cost [$/s]	L [$/w] (1)	L [%] (1)	L [$/w] (2)	L [%] (2)	Tot. gas [kg/s]
p_{bh2}&p_{wh2}	-45.308570	26.0000	0.000099	1.73	0.000006	9.95260
p_{bh2}&p_{ai2}	-45.308570	28.8415	0.000105	3.40	0.000012	9.95262
p_{bh2}&p_{gs}	-45.308570	29.0000	0.000106	3.70	0.000013	9.95260
p_{bh2}&p_{m}	-45.308572	28.3554	0.000103	2.80	0.000010	9.95230
p_{bh2}/p_{d3}	-45.308572	28.4167	0.000104	3.00	0.000011	9.95260
No control	-45.307760	514	0.0019	488	0.001780	9.95270

Table 5.7 demonstrates a continuation of the previous trend, where the dominance of the bottomhole pressure in the H matrix is observed. The losses for most combinations are approximately equal, except for the bottomhole pressure-wellhead pressure combination, which exhibits an even smaller loss. Therefore, we propose controlling the bottomhole pressure-wellhead pressure combination in this region.

5.3.2.3 Proposed overall control structure nullspace method

Based on the results obtained from both negative and positive changes in the GOR of well 2 around the optimal nominal point, it can be concluded that controlling the bottomhole pressure-discharge pressure combination is the best implementation in the active constraint region. Conversely, in the unconstrained region, the optimal alternative is the bottomhole pressure-wellhead pressure combination. The proposed control structure for the entire plant, where we suggest controlling the measurement combination with a single MV, is depicted in Figure 5.10.

To switch between the unconstrained (u) and constrained (c) SOC controllers, we propose a 2-point controller based on the total produced gas value. This switching mechanism is represented by the "Logic" block in the figure.

5.3.3 Exact local method

The implementation of the exact local method, as described in Section 4.1.2.3, was further enhanced by considering the measurement error associated with different variables. The measurement combination with the lowest loss, obtained from the nullspace method for various active constraint regions, was compared with newly introduced measurements that are more susceptible to measurement error.

The controllers obtained in Section 4.1.2.3 were tested through system simulations for a duration of $t = 100000s$ to ensure convergence and the attainment of a new steady-state. Similar to the previous case, the separator level was controlled to a constant setpoint. In line with the nullspace method, the nominal point was perturbed to determine a new unconstrained nominal point for evaluating the measurement combinations in the unconstrained region. The initial controller tunings were adjusted to improve performance. The results of controlling the proposed controlled variables can be found in Appendix A.4.

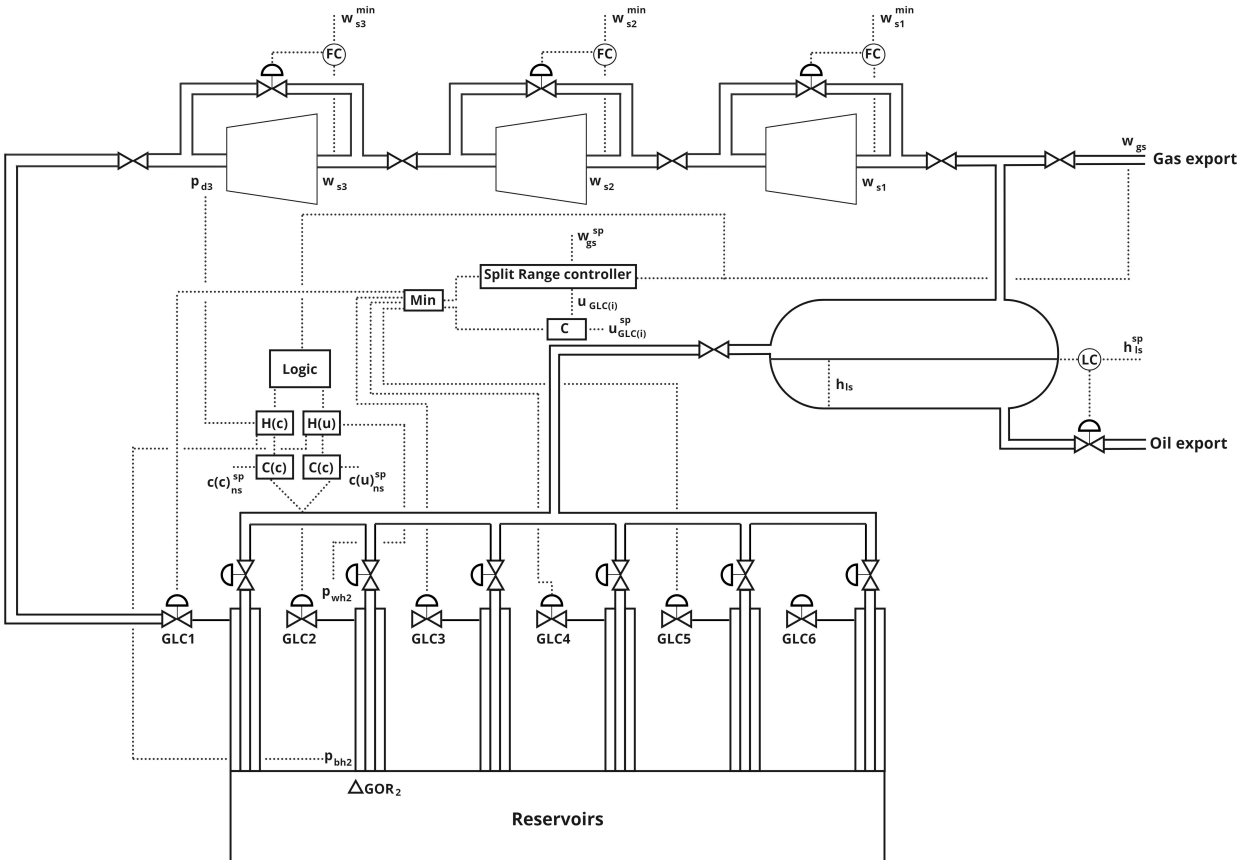


Figure 5.10: Resulting overall control structure from the results of the nullspace method.

5.3.3.1 GOR increase in well 2

The results of the simulations for an increase in the GOR of well 2 with +3% can be found in Table 5.8. The optimal cost at this operating point was found to be:

(1) -45.2406 \$/s

(2) -45.1587 \$/s

Table 5.8: The results of controlling the measurement combinations to their optimal nominal value facing a disturbance of +3% in the GOR of well 2. L denotes the loss compared to optimal operating points.

Variable	Cost [\$/s]	L [\$/w] (1)	L [%] (1)	L [\$/w] (2)	L [%] (2)	Tot. gas [kg/s]
p_{bh2} &p_{d3}	-45.0037	143256	0.52356	93533	0.34	10
p_{bh2} & w_{po2}	-45.0051	142413	0.52048	92692	0.33	10
p_{bh2} & w_{pg2}	-45.0035	143420	0.52416	93697	0.34	10
p_{bh2} & w_{gl}	-45.0040	143103	0.52300	93381	0.34	10
w_{po2} & w_{gl}	-45.0039	143128	0.52300	93406	0.34	10
w_{pg2} & w_{gl}	-44.9993	187059	0.68263	136967	0.50	10

The dominance of bottomhole pressure in the optimal measurement matrix H is evident, as shown in Table 4.14, similar to the findings of the nullspace method. However, if the bottomhole pressure transmitter is unavailable, controlling the oil flow of well 2 and the total gas lift could be a viable alternative. Since fixing the bottomhole pressure transmitter can present practical challenges, an alternative control structure should be considered.

Furthermore, it is worth noting that controlling the gas flow of well 2 and the gas lift leads to greater loss compared to other combinations. On the other hand, controlling the bottomhole pressure in

conjunction with the oil flow of well 2 yields the same loss as controlling only the bottomhole pressure, making it the measurement combination with the lowest amount of loss.

5.3.3.2 GOR decrease in well 2

The results of the simulations for a decrerase in the GOR of well 2 with -3% can be found in Table 5.9. The optimal cost at this operating point was found to be:

(1) -45.3086 $/s

(2) -45.30857 $/s

Table 5.9: The results of controlling the measurement combinations to their optimal nominal value facing a disturbance of -3% in the GOR of well 2. L denotes the loss compared to optimal operating points.

Variable	Cost [$/s]	L [$/w] (1)	L [%] (1)	L [$/w] (2)	L [%] (2)	Tot. gas [kg/s]
p_{bh2} &p_{wh2}	-45.30857	28.4940	0.000104	3.10	0.000011	9.95
p_{bh2}&w_{po2}	-45.30857	28.4941	0.0001040	3.10	0.000013	9.95
p_{bh2}&w_{pg2}	-45.30857	28.4403	0.0001038	3.00	0.000011	9.95
p_{bh2}&w_{gl}	-45.30857	28.5560	0.0001040	3.10	0.000011	9.95
w_{po2}&w_{gl}	-45.30857	29.3740	0.0001072	4.00	0.000015	9.95
w_{pg2}&w_{gl}	-45.30857	25.5046	0.0000931	0.11	0.000001	9.95

The results in Table 5.9 show that the proposed combination from the nullspace method results in more loss when implemented with the exact local method. The H matrix in this case is strongly dominated by the bottomhole pressure, unlike the nullspace method. Consequently, the loss is nearly identical to controlling only the bottomhole pressure. The exact local method determines the weights of the controlled variables using the gain from the MVs and Y, as explained in Section 4.1.2.3. The gain from the MV on the bottomhole pressure is more significant than on the wellhead pressure, leading to an increased weight for the bottomhole pressure. Controlling the produced oil of well 2 and the total gas lift results in slightly higher loss compared to other potential combinations, while the combination with the least loss in this case is the produced gas and total gas lift combination.

5.3.3.3 Proposed overall control structure exact local method

Based on the results of both negative and positive changes in the GOR of well 2 around the optimal nominal point, we can conclude that controlling the bottomhole pressure and produced oil of well 2 is the best implementation in the active constraint region. In the unconstrained region, the best alternative is to control the produced gas of well 2 and the total gas lift. Figure 5.11 illustrates the proposed control structure for the entire plant, where we suggest controlling the measurement combination with one MV for the unconstrained region (u) and the constrained region (c).

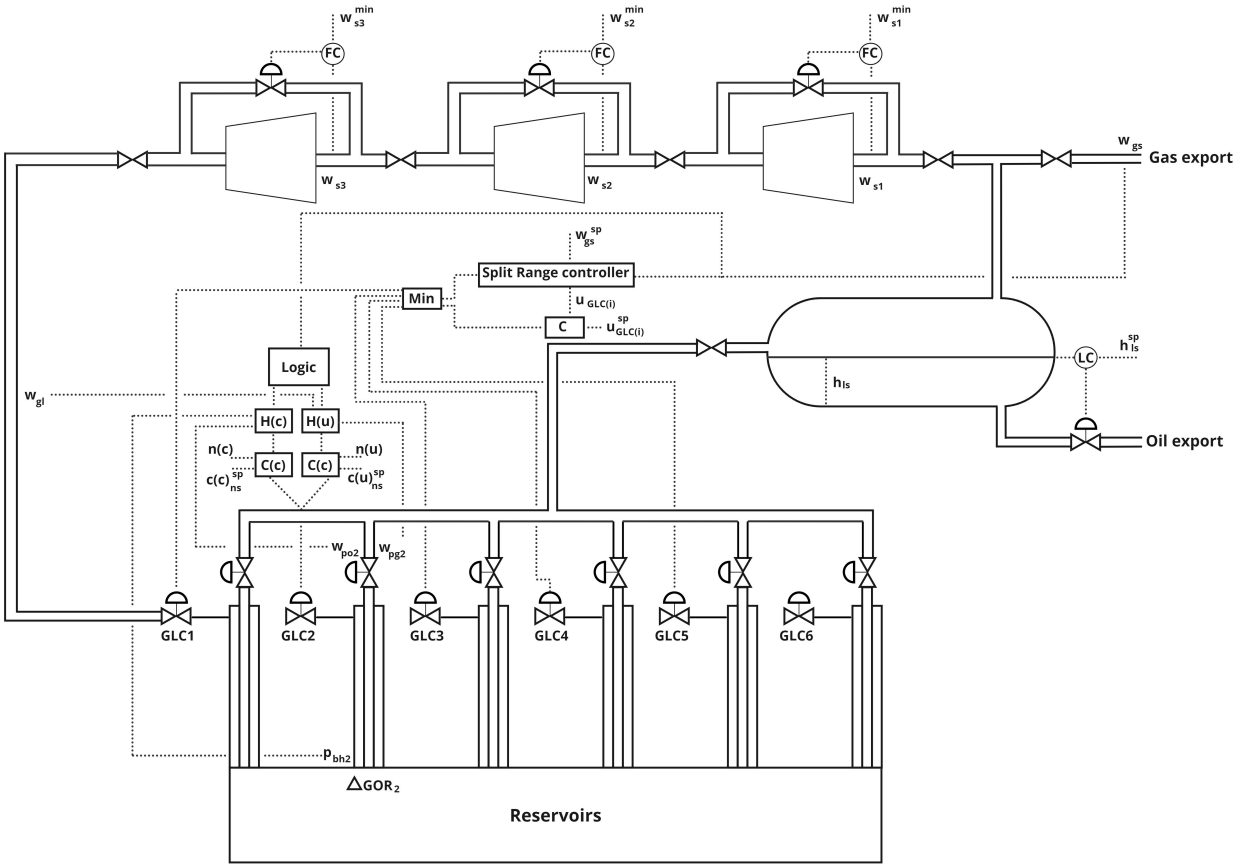


Figure 5.11: Resulting overall control structure from the results of the exact local method.

5.4 Results of Case 2

In case 2, we explored the use of 2 MVs to control measurement combinations in the presence of disturbance in the GOR of well 2 and 6. To linearize the system around the nominal point, we employed finite difference, as described in Section 4.1.3. However, due to the non-linearity of the model, solver difficulties were encountered. Nonetheless, for extremely small perturbations in the variables (of magnitude 10^{-8}), the optimizer was able to handle the computations. Table A.1 in Appendix A.1 presents the proposed top 4 controlled measurement sets, along with their corresponding loss calculated by the algorithms.

Table A.1 in Appendix A.1 provides an overview of the measurement combinations associated with either the worst-case loss or the minimal average loss suggested by the different bracket and bound implementations. The results indicate that incorporating more CVs generally leads to lower loss. However, when comparing the case of two disturbances and two MVs to previous cases with one disturbance and one MV for SOC, there are noticeable differences. Although some similarities in the selection of optimal measurement combinations exist, it is expected that the bottomhole pressure would have a more dominant role, as indicated by the results from the more brute force approach.

By examining the cost function in Figure 5.1, it becomes evident that the dynamics change at the nominal operating point, which lies precisely at the constraint boundary. As the GOR increases, the system behavior undergoes rapid changes due to the nonlinear nature of the model. Based on observations and the obtained results, it is clear that perturbing the disturbance (d) in the brute force approach of case 1 captures the dynamics of the region change more accurately. However, since the methods are defined as local, their objective is to capture the behavior around the nominal point. The issue arises from the non-linearity of the model and the placement of the nominal point on the border. It becomes apparent that the system's behavior is not comparable to how it changes

further into the region. The limitations of local approaches become evident when dealing with systems exhibiting changing behavior.

It is expected that the behavior in the unconstrained region differs from that in the constrained region. However, when faced with nonlinear behavior within the same constraint region, more complex control structures should be considered. One possibility for further implementation is to analyze the system's behavior in all potential constraint regions and design control structures tailored to the specific behaviors observed in each region. This could be achieved through CV-CV switching or gain-scheduling. Alternatively, dynamic RTO could be introduced to adaptively adjust the control strategy based on changing conditions.

The Branch and Bound method yielded measurement combinations with minimal loss, which were further examined for evaluation. Two control structures were implemented to assess the difference between obtaining the sensitivity matrix by re-optimizing the system and using the linearized version obtained from the implementation.

In practice, a small perturbation was introduced to the disturbance to obtain the sensitivity matrices in case 1. However, due to the nonlinearity of the system, the sensitivity of the measurements varies depending on the magnitude of the perturbation. Although the perturbations should be small, in this case even perturbations on the order of 10^{-6}% compared to the nominal value cause changes in the sensitivity values.

5.4.1 Proposed overall control structure Branch and Bounds average loss

The Branch and Bound method identified the combination of p_{ai2}&p_{wh6}&p_{bh2}&p_{d3} as the one with the lowest average loss. However, due to the limitations of the solver, a solution in the unconstrained region could not be obtained. The proposed total control structure for this case is depicted in Figure 5.12.

5.4.2 Case 2 linear approach

5.4.2.1 Positive GOR change

The results of the simulations for an increase in the GOR of well 2 with +3% and well 6 with +2% can be found in Table 5.10. The optimal cost at this operating point was found to be:

(1) -45.1652 $/s

(2) -45.0984 $/s

Table 5.10: The results of controlling the measurement combinations to their optimal nominal value facing a disturbance of +3% in the GOR of well 2 and +2% in the GOR of well 6. Combinations marked (NC) were not controllable.

Variable	Cost [$/s]	L [$/w] (1)	L [%] (1)	L [$/w] (2)	L [%] (2)	Method
p_{ai2} &p_{bh2}(NC)	-	-	-	-	-	WCBnB
p_{ai2} &p_{ai6}	-44.7169	271131	0.9925	230404	0.84	AVBnB
p_{ai6}&p_{d3}	-44.6773	295024	1.0800	254270	0.93	WCPBnB
p_{ai6}&p_{bh6}&p_{d3}	-44.6907	286925	1.0500	246087	0.90	WCPBnB
p_{ai6}&p_{bh6}&p_{d3}&p_{ai2}	-44.6923	285944	1.0468	245214	0.89	WCPBnB
p_{ai2}&p_{d3} (NC)	-	-	-	-	-	AVPBnB
p_{ai2}&p_{bh2}&p_{d3}	-44.7584	246019	0.9006	205338	0.75	AVPBnB
p_{wh6}&p_{bh2}&p_{d3}&p_{ai2}	-44.7530	249247	0.9125	208573	0.76	AVPBnB
No control	-44.7468	252999	0.9269	212266	0.78	-

Table 4.20 presents the loss associated with each measurement combination obtained from simulating the control structure with a positive GOR change in wells 2 and 6. It is evident that in two of

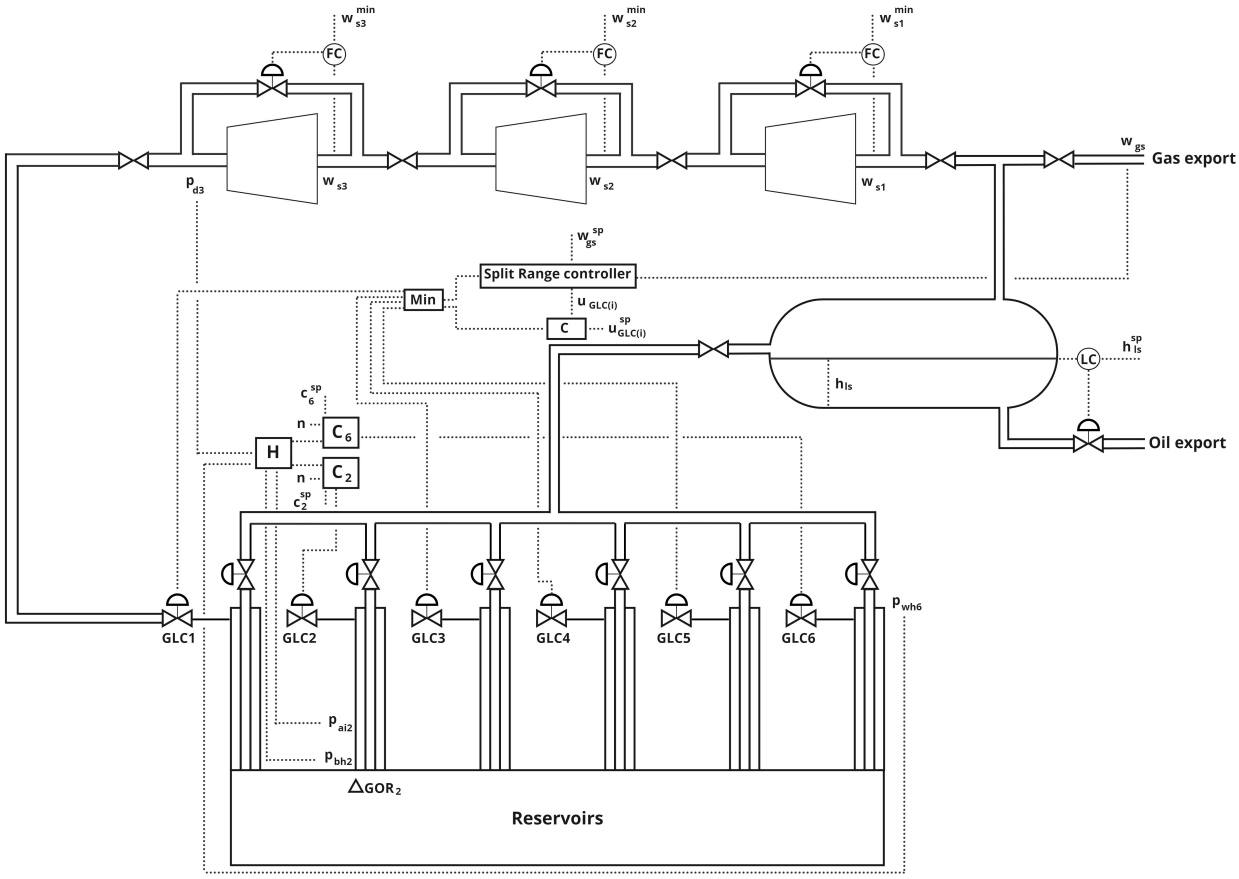


Figure 5.12: Resulting overall control structure from the results of Branch and Bound implementation.

the combinations, the controllers are unable to regulate their respective optimal combinations to their setpoints (c_{ns}). This lack of control can be attributed to interactions between the controllers. In the simulations, three gas lift chokes are responsible for controlling a variable or measurement combination at all times, leading to changes in one choke affecting variables in other wells. For the p_{ai2} & p_{bh2} and p_{ai2} & p_{d3} combinations, simulations have shown that only one of the controllers is able to regulate the measurement combination to its setpoint. Introducing a VPC only assists the valve it controls while hindering the others from achieving their objectives. As a result, the controllers operate effectively only in certain operating regions where one controller successfully controls the measurement combination, rendering them invalid. The interactions between the controllers highlight the reason why this thesis limited the number of MVs to a few. Although utilizing all the GLCs would minimize loss, it would require more complex control structures. While this issue has been partially addressed by operating the controllers on different timescales in this thesis, it remains insufficient for all combinations.

In general, most of the combinations exhibit higher loss compared to having no control implemented. This outcome is anticipated based on the system's behavior differing in regions other than around the nominal point. However, a noticeable trend is observed wherein the inclusion of additional measurements generally leads to reduced loss, as evident from Table 4.20 in Appendix A.1. In this specific case, the optimal combination for control is found to be p_{ai2} & p_{bh2}& p_{d3}.

5.4.2.2 Negative GOR change

The results of the simulations for a decrease in the GOR of well 2 with -3% and well 6 with -2% can be found in Table 5.11. The optimal cost at this operating point was found to be:

(1) -45.3106 $/s

(2) -45.3105 $/s

Table 5.11: The results of controlling the measurement combinations to their optimal nominal value facing a disturbance of -3% in the GOR of well 2 and -2% in the GOR of well 6. Combinations marked (NC) where not able to be controlled.

Variable	Cost [$/s]	L [$/w] (1)	L [%] (1)	L [$/w] (2)	L [%] (2)	Method
p_{ai2}&p_{bh2}	-45.2715	23651	0.0863	23594	0.0861	WCBnB
p_{ai2}&p_{ai6}	-45.3067	2369	0.0086	2302	0.0084	AVBnB
p_{ai6}&p_{d3}	-45.3087	1125	0.0041	1068	0.0039	WCPBnB
p_{ai6}&p_{bh6}&p_{d3}	-45.3087	1163	0.0042	1096	0.0040	WCPBnB
p_{ai6}&p_{bh6}&p_{d3}&p_{ai2}	-45.3084	1346	0.0049	1288	0.0047	WCPBnB
p_{ai2}&p_{d3}	-45.3071	2112	0.0077	2055	0.0075	AVPBnB
p_{ai2}&p_{bh2}&p_{d3}	-45.3087	1121	0.0041	1064	0.0039	AVPBnB
p_{wh6}&p_{bh2}&p_{d3}&p_{ai2}	(NC)	-	-	-	-	AVPBnB
No control	-45.3083	1383	0.0050	1315	0.0048	-

From the results presented in Table 5.11, it is evident that the measurement combinations which were previously uncontrollable in the case of positive GOR change are now controllable. This can be attributed to the gas lift choke controlling the produced gas not being active at this particular operating point, thereby reducing the interaction between the controllers. However, it should be noted that due to the active constraint change, the obtained solution may not be applicable in the unconstrained region. Notably, the combination of four variables proposed by the average loss BAB method could not be effectively controlled. This can be attributed to conflicting control objectives, resulting in both controllers eventually saturating in different directions. Furthermore, it is observed that the combination of p_{ai2} & p_{bh2} exhibits significantly higher loss compared to the other combinations. The remaining methods do not exhibit substantial variations, and the combination yielding the least loss corresponds to the same combination as in the constrained region.

5.4.2.3 Comparing with F found from model

To compare the solutions obtained from the linearized sensitivity matrix and the re-optimized system, the H matrices were derived using both approaches. The linearized model was based on perturbations of magnitude 10^{-8}, while the analytical method used perturbations of 10^{-4} to determine F, due to solver limitations. This discrepancy in perturbation values resulted in different outcomes. This comparison was undertaken to satisfy curiosity, assess the differences between the cases, and evaluate the challenges posed by the non-linear model.

The first set of results of controlling the measurement combinations where the H is found analytically are presented in Table 5.12. The optimal cost at this operating point was found to be:

(1) -45.1652 $/s

(2) -45.0984 $/s

The second set of results of controlling the measurement combinations where the H is found analytically are presented in Table 5.13. The optimal cost at this operating point was found to be:

(1) -45.3106 $/s

(2) -45.3105 $/s

As anticipated, the utilization of F obtained through re-optimization of the system yielded lower loss compared to F derived from the linearized model. This outcome can be attributed to the nonlinear

Table 5.12: The results of controlling the measurement combinations to their optimal nominal value facing a disturbance of +3% in the GOR of well 2 and +2% in the GOR of well 6. Combinations marked (NC) where not able to be controlled.

Variable	Cost [$/s]	L [$/w] (1)	L [%] (1)	L [$/w] (2)	L [%] (2)	Method
p_{ai2}-p_{bh2}	-44.7061	277628	1.0165	236912	0.87	WCBnB
p_{ai2}-p_{ai6}	-44.7169	271131	0.9925	230404	0.84	AVBnB
p_{ai6}-p_{d3}	-44.7301	263095	0.9632	222398	0.82	WCPBnB
p_{ai6}-p_{bh6}-p_{d3}	-44.7881	228040	0.8348	187390	0.69	WCPBnB
p_{ai6}-p_{bh6}-p_{d3}-p_{ai2}	-44.7973	222462	0.8100	180626	0.66	WCPBnB
p_{ai2}-p_{d3}	-44.7178	270578	0.99055	229872	0.8400	AVPBnB
p_{ai2}-p_{bh2}-p_{d3}	-44.7584	246019	0.9006	205338	0.75	AVPBnB
p_{wh6}-p_{bh2}-p_{d3}-p_{ai2}	(NC)	-	-	-	-	AVPBnB
No control	-44.7468	252999	0.9260	212266	0.78	-

Table 5.13: The results of controlling the measurement combinations to their optimal nominal value facing a disturbance of -3% in the GOR of well 2 and -2% in the GOR of well 6. Combinations marked (NC) where not able to be controlled.

Variable	Cost [$/s]	L [$/w] (1)	L [%] (1)	L [$/w] (2)	L [%] (2)	Method
p_{ai2}-p_{bh2}	-45.3017	5402	0.0197	5346	0.019	WCBnB
p_{ai2}-p_{ai6}	-45.3067	2369	0.0086	2302	0.0084	AVBnB
p_{ai6}-p_{d3}	-45.3088	1096	0.0040	1041	0.0038	WCPBnB
p_{ai6}-p_{bh6}-p_{d3}	-45.3054	3173	0.0116	3118	0.0110	WCPBnB
p_{ai6}-p_{bh6}-p_{d3}-p_{ai2}	-45.3086	1197	0.0044	1142	0.0042	WCPBnB
p_{ai2}-p_{d3}	-45.3071	2112	0.0077	2055	0.0075	AVPBnB
p_{ai2}-p_{bh2}-p_{d3} (NC)	-	-	-	-	-	AVPBnB
p_{wh6}-p_{bh2}-p_{d3}-p_{ai2}	-45.3084	1288	0.0049	1288	0.0047	AVPBnB
No control	-45.3083	1383	0.0050	1315	0.0048	-

behavior exhibited by the system in the vicinity of the nominal operating point. Consequently, even slight variations in step lengths can lead to significant alterations in system dynamics.

6 Discussion

The results have already been addressed and analyzed in the Results section. Therefore, this section will focus on exploring more general observations and the underlying assumptions made in the study.

6.1 Model assumptions and limitations

Several simplifications have been made to model the oil and gas production system. These simplifications and assumptions are however necessary to be able to model the system practically. Given that the purpose of this model is to look at several subsystems as a whole, the decision was made to not delve to deeply into each systems specifics. To justify the decision, it should be noted that the detailed modelling of any of these subsystems could be a thesis on it's own.

We have made the assumption that the production fluid consists of only oil and gas to simplify the system. In real oil and gas production systems there are generally always water in the production fluids. However, in this model where the dynamics of the fluid are not modelled in detail, the introduction of water in to the system would essentially only change the composition of the production fluid. An implication of this would be that the density of the production fluid would increase, due to oil being less dens than water. In the separator another outlet for the produced water would need to be introduced. However, due to the assumption of perfect separation the water would just be removed from the other fluids without any further processing. Due to the scope of this project where we don't consider further treatment of any of the produced fluids, the introduction of water would not make a significant difference. It should be noted that for real systems where the separation is far from perfect, considering the water would be important. This is due to the related treatment and installation cost of removing oil from the water, removing water from the gas and multiple separation stages to remove the water from the oil.

To limit the model and define the scope of the work, constant parameters at the battery limits was introduced. This included the export gas pressure, export oil pressure, productivity index and the reservoir pressure. These constant parameters may effect relationships between variables in the simulations compared to the real scenarios. An example of this is the relationship between the GOR and the reservoir pressure in the well. Typically, when a well is new the reservoir pressure is high and the GOR is low. As the well is produced the reservoir pressure drops due to depletion of fluid in the reservoir and the GOR increases. As the reservoir pressure is assumed constant in this study, the relationship between the variables are not considered when the GOR changes and will thus introduce a margin of error. However, since modelling is an approximation of the real system, limits will need to be introduced to any model.

Another variable considered constant is the temperature in the different regions. The temperatures have been modelled to mimic the temperature drop from the different regions. The lack of explicit consideration of temperature in this case may result in the calculation of different system properties in different systems being less realistic.

The modelling of the compressor train introduces another source of uncertainty. The compressor curves were approximated through trial and error to replicate the behavior of an actual system. In a real-world compressor system, these curves would typically be provided by the supplier or obtained through data analysis and model fitting techniques.

6.2 General observations about the results

In the results in Section 5 the results of the control implementations was shown and discussed. The main observations was that the different control structures for the purpose of regulatory control worked as they should for the potential disturbances considered.

For the results of implementing the different control structures for SOC, the uncertainty of not completely converged models should be considered. Due to the share number of evaluations a standard evaluation time of t = 100000s was implemented. Measures as evaluating the difference between the two last values of the most relevant variables was implemented, but due to the share number of evaluations and model variables this was not possible for the almost 200 process variables.

Another factor that makes the implementation of SOC strategies difficult in this case, is the location of the nominal operating point at the border between the constraint regions. Due to limitations in the solver and difficulties related to initial values, implementing the nominal point further into the constrained region became tedious.

6.2.1 Case 1

The general findings from case 1 was that the best controlled variable essentially for all the methods and the two constraint regions where controlling the bottomhole pressure with some exceptions.

In the constrained region all the potential controlled variables with the exception of the single controlled variable resulted in less loss than not controlling anything. Based on this we can determine that implementing a SOC strategy is a better alternative then doing nothing.

However, it is important to note that the overall loss within the constrained region is relatively significant. This can be attributed to the impact of the total produced gas constraint on the system. When the constraint is active, it restricts the total production, leading the system to gradually limit production to counteract the increased gas within the system. Even in the base case where all the MVs are available for optimization, as observed in Figure 5.1, there is a considerable loss, even when compared to methods like RTO, which serves as a point of comparison for our implementation.

If we consider only the available MVs used in the case, the objective function value determined by the optimizer would be -45.1587. While this results in less loss for the implementations, it is still significant. The non-linear nature of the system presents challenges in obtaining accurate magnitudes for each controlled variable in the optimal measurement combinations, primarily due to the local methods employed in the implementation.

At the nominal operating point, it is evident that controlling bottomhole pressure is the most favorable variable and dominates the measurement combinations. However, as dynamic changes occur further within the constrained region, the weights of the measurements would undoubtedly shift. Given the non-linear behavior within the constrained region, approaches such as feedback-optimizing control, gain-scheduling or dynamic RTO should be considered.

In the unconstrained region, the bottomhole pressure was identified as the optimal controlled variable for most combinations, except for the nullspace method where a combination of bottomhole pressure and wellhead pressure exhibited lower loss. Additionally, controlling the produced gas of well 2 and the total gas lift using the exact local method resulted in reduced loss. It is noteworthy that in the unconstrained region, the rate of change is more linear compared to the constrained region. This linear behavior contributes to lower loss and allows for more accurate estimation of the magnitudes of variables in the optimal measurement combination, particularly further away from the nominal operating point.

To ensure consistency with the implementation of local methods and in line with the theory proposing different SOC control structures for different active constraint regions, the optimal nominal point was moved into the unconstrained region during evaluation of the best strategies. The selection of the active control structure was determined by measuring the total produced gas and deciding based on the presence or absence of the constraint. The optimizer was employed to find the optimal point in cases where only the MVs used in the simulations were available for control, resulting in an objective function value of -45.3085, which is nearly equal to the case where all MVs are available for control.

Given the aim of this paper to identify simple control structures, it is concluded that solely controlling the manipulated variable of the well experiencing a disturbance would be beneficial in achieving the desired outcomes.

6.2.2 Case 2

In case two, we applied linearization to the model and utilized the bi-directional branch and bound algorithm to evaluate measurement combinations with the least worst-case loss and least average loss. To achieve this, we perturbed the linearization with a significantly smaller value compared to the brute force approach employed in case one. However, due to the non-linear nature of the system, the local conditions were not the same, leading the branch and bound method to identify measurement combinations that were not suitable for the specific system under consideration. This outcome highlighted the vulnerabilities associated with utilizing local linear approximations when dealing with highly non-linear systems.

Due to the solver's difficulties, the method was solely evaluated around the nominal point within the unconstrained region, which deviated from the condition related to constraint changes defined by the local methods employed for SOC. However, it is important to note that the proposed methods were tested for operation in both the constrained and unconstrained regions

During the implementation of the methods, a gradual increase in the number of manipulated variables was proposed in an attempt to achieve control over all of them. However, due to the substantial workload involved and the challenges associated with the coupling between the gas lift chokes, achieving this goal using decentralized control proved difficult. The tuning process for many controllers became time-consuming in order to minimize the interaction among them. It was observed that even with the utilization of only two manipulated variables for SOC, some control structures failed to work effectively.

For future investigations, more advanced and complex control strategies could be explored to address these challenges and improve the system's control performance.

The results obtained from both approaches, namely using the F matrix obtained by perturbing it with a small disturbance and re-optimizing, as well as using the F matrix derived from the linearized model, clearly demonstrated that the first option yielded better outcomes. Despite both perturbations being relatively small, this highlights the challenges associated with the non-linearity of the system.

One potential strategy to address these challenges in a non-linear system is to analyze the system across the entire operating range and subsequently design different control structures corresponding to similar regions. However, if the system's behavior varies within the same constraint regions, it may become challenging to devise reliable switching logic between the different implementations.

7 Conclusion

In conclusion, this master's thesis has highlighted the potential benefits and challenges associated with designing self-optimizing control structures for an oil and gas production system with recycled gas lift. The inherent non-linearity of the system and the dynamic changes in active constraints present significant difficulties during the implementation of control structures. To address changes in operating conditions and disturbances, the use of CV-CV switching to control the optimal measurement combination and active constraints in each region is recommended. The thesis has demonstrated that the careful selection of controlled variables using heuristics can yield comparable or even better results than more complex methods. Local control methods generally perform well in seemingly linear regions but may result in suboptimal performance in non-linear regions. In highly non-linear regions, it is crucial to evaluate more sophisticated control structures based on the trade-off between potential benefits and disadvantages associated with their implementation. Additionally, the study has identified limitations in controlling multiple gas lift chokes connected to the same feed source using decentralized control strategies.

Regarding the regulatory control layer, the implemented surge control structure effectively managed flows below the surge limit, and the logic ensured the closure of valves when additional recycle flow was unnecessary. Furthermore, the split range controller with the baton strategy demonstrated its capability to control the active constraint on total produced gas, while the logic based on one active manipulated variable at a time minimized interactions between the gas lift chokes. Both implemented level control strategies proved effective in their respective control targets; however, for operational stability and convergence of simulations, constant control of the separator level was preferred. The addition of valve position control played a critical role in controlling multiple controlled variables. The implementation of CV-CV switching demonstrated significant economic advantages compared to controlling the active constraint in unconstrained regions.

In the first case study, it was found that using only one manipulated variable for control in the constrained region was insufficient. However, in the unconstrained region, controlling single variables or combinations derived from the nullspace method or exact local method resulted in minimal loss. The bottomhole pressure of well 2 emerged as the overall preferred controlled variable, showing potential self-optimizing qualities compared to other variables investigated. In the unconstrained region, controlling the total gas lift and the gas flow of well 2 resulted in the least amount of loss.

The results of the second case study suggested that the measurement combination of annulus pressure of well 2, wellhead pressure of well 6, bottomhole pressure of well 2, and the discharge pressure of compressor 3 yielded the least amount of average loss. However, simulations revealed the vulnerabilities of the local method when disturbances moved far away from the nominal operating point. This case study highlighted how even small perturbations, when obtaining the linearized model, can lead to significant changes in the optimal combination matrices and related weights.

In summary, this research has shed light on the potential benefits and challenges associated with designing self-optimizing control structures for oil and gas production systems with recycled gas lift. It has emphasized the importance of carefully selecting controlled variables, considering the non-linear nature of the system and active constraint changes. The implementation of CV-CV switching has shown economic advantages, and the study has provided insights into the limitations and performance of various control strategies.

8 Further work

For further work we propose implementing more complex control structures in the active constraint region, due to the highly non-linear nature of the system. The issue related to control of multiple gas lift chokes should also be addressed with more complex control structures as feedback-optimizing control or dynamic-RTO.

A natural next step in the modelling of the system is to use real life data from a production facility and re-model it based on this. The model should then be evaluated based on the behaviour be compared to the real life system. The model could also be extended with an export compressor and a oil export pump to consider the cost of power in these components as well.

References

[1] Kristian Ødegård. Modelling and optimization of recirculated gas lift problem, 2022. URL https://folk.ntnu.no/skoge/diplom/prosjekt22/odegard/Specialization_Project_Kristian_Odegard.pdf.

[2] Manfred Morari, Yaman Arkun, and George Stephanopoulos. Studies in the synthesis of control structures for chemical processes: Part i: Formulation of the problem. process decomposition and the classification of the control tasks. analysis of the optimizing control structures. *AIChE Journal*, 26(2):220–232, 1980.

[3] Vidar Alstad. Studies on selection of controlled variables. *Norwegian University of Science and Technology*, 2005.

[4] Dinesh Krishnamoorthy, Kjetil Fjalestad, and Sigurd Skogestad. Optimal operation of oil and gas production using simple feedback control structures. *Control Engineering Practice*, 91, 08 2019. doi: 10.1016/j.conengprac.2019.104107.

[5] Esmaeil Jahanshahi, Dinesh Krishnamoorthy, Andrés Codas, Bjarne Foss, and Sigurd Skogestad. Plantwide control of an oil production network. *Computers Chemical Engineering*, 136:106765, 2020. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2020.106765. URL https://www.sciencedirect.com/science/article/pii/S0098135419306271.

[6] Sigurd Skogestad. Plantwide control: the search for the self-optimizing control structure. *Journal of Process Control*, 10(5):487–507, 2000. ISSN 0959-1524. doi: https://doi.org/10.1016/S0959-1524(00)00023-8. URL https://www.sciencedirect.com/science/article/pii/S0959152400000238.

[7] Diego Fernando Mendoza, José Eduardo Alves Graciano, Fabio dos Santos Liporace, and Galo Antonio Carrillo Le Roux. Assessing the reliability of different real-time optimization methodologies. *The Canadian Journal of Chemical Engineering*, 94(3):485–497, 2016.

[8] Adriana Reyes-Lúa and Sigurd Skogestad. Multi-input single-output control for extending the operating range: Generalized split range control using the baton strategy. *Journal of Process Control*, 91:1–11, 2020.

[9] Adriana Reyes-Lua and Sigurd Skogestad. Systematic design of active constraint switching using classical advanced control structures. *Industrial & Engineering Chemistry Research*, 59 (6):2229–2241, 2019.

[10] Adriana Reyes-Lúa, Cristina Zotică, Krister Forsman, and Sigurd Skogestad. Systematic design of split range controllers. *IFAC-PapersOnLine*, 52(1):898–903, 2019.

[11] Sigurd Skogestad. Simple analytic rules for model reduction and pid controller tuning. *Journal of process control*, 13(4):291–309, 2003.

[12] Yi Cao and Vinay Kariwala. Bidirectional branch and bound for controlled variable selection: Part i. principles and minimum singular value criterion. *Computers & Chemical Engineering*, 32(10):2306–2319, 2008.

[13] Dinesh Krishnamoorthy and Sigurd Skogestad. Online process optimization with active constraint set changes using simple control structures. *Industrial & Engineering Chemistry Research*, 58(30):13555–13567, 2019.

[14] Predrag Milosavljevic, Alejandro G. Marchetti, Andrea Cortinovis, Timm Faulwasser, Mehmet Mercangöz, and Dominique Bonvin. Real-time optimization of load sharing for gas compressors in the presence of uncertainty. *Applied Energy*, 272:114883, 2020. ISSN 0306-2619. doi: https://doi.org/10.1016/j.apenergy.2020.114883. URL https://www.sciencedirect.com/science/article/pii/S0306261920303950.

[15] Johannes Jäschke, Yi Cao, and Vinay Kariwala. Self-optimizing control–a survey. *Annual Reviews in Control*, 43:199–223, 2017.

[16] Sigurd Skogestad, Ivar J Halvorsen, and John C Morud. *Self-optimizing control: The basic idea and taylor series analysis*. American Institute of Chemical Engineers, 1998.

[17] Ivar J Halvorsen, Sigurd Skogestad, John C Morud, and Vidar Alstad. Optimal selection of controlled variables. *Industrial & Engineering Chemistry Research*, 42(14):3273–3284, 2003.

[18] Risvan Dirza, Jose Matias, Sigurd Skogestad, and Dinesh Krishnamoorthy. Experimental validation of distributed feedback-based real-time optimization in a gas-lifted oil well rig. *Control Engineering Practice*, 126:105253, 2022.

[19] Truls Larsson and Sigurd Skogestad. Plantwide control-a review and a new design procedure. 2000.

[20] Vidar Alstad, Sigurd Skogestad, and Eduardo S Hori. Optimal measurement combinations as controlled variables. *Journal of Process Control*, 19(1):138–148, 2009.

[21] Vidar Alstad and Sigurd Skogestad. Null space method for selecting optimal measurement combinations as controlled variables. *Industrial & engineering chemistry research*, 46(3):846–853, 2007.

[22] Vinay Kariwala and Yi Cao. Bidirectional branch and bound for controlled variable selection. part ii: Exact local method for self-optimizing control. *Computers & chemical engineering*, 33 (8):1402–1412, 2009.

[23] Vinay Kariwala and Yi Cao. Bidirectional branch and bound for controlled variable selection part iii: Local average loss minimization. *IEEE Transactions on Industrial Informatics*, 6(1): 54–61, 2010.

[24] Mark L Darby, Michael Nikolaou, James Jones, and Doug Nicholson. Rto: An overview and assessment of current practice. *Journal of Process control*, 21(6):874–884, 2011.

[25] André D Quelhas, Normando José Castro de Jesus, and José Carlos Pinto. Common vulnerabilities of rto implementations in real chemical processes. *The Canadian Journal of Chemical Engineering*, 91(4):652–668, 2013.

[26] Sigurd Skogestad. Control structure design for complete chemical plants. *Computers Chemical Engineering*, 28(1):219–234, 2004. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2003.08.002. URL https://www.sciencedirect.com/science/article/pii/S0098135403001984. Escape 12.

[27] Sigurd Skogestad and I Postlethwaite. *Multivariable Feedback Control: Analysis and Design*, volume 2. 01 2005.

[28] Truls Larsson, Kristin Hestetun, Espen Hovland, and Sigurd Skogestad. Self-optimizing control of a large-scale plant: The tennessee eastman process. *Industrial & Engineering Chemistry Research*, 40(22):4889–4901, 2001. doi: 10.1021/ie000586y. URL https://doi.org/10.1021/ie000586y.

[29] Adriana Reyes-Lúa, Cristina Zotică, and Sigurd Skogestad. Optimal operation with changing active constraint regions using classical advanced control. *IFAC-PapersOnLine*, 51(18):440–445, 2018.

[30] Dinesh Krishnamoorthy and Sigurd Skogestad. Systematic design of active constraint switching using selectors. *Computers Chemical Engineering*, 143:107106, 2020. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2020.107106. URL https://www.sciencedirect.com/science/article/pii/S0098135420307274.

[31] James Crowe, GR Chen, R Ferdous, DR Greenwood, MJ Grimble, HP Huang, JC Jeng, Michael A Johnson, MR Katebi, S Kwong, et al. *PID control: new identification and design methods*. Springer, 2005.

[32] Vinay Kariwala, Yi Cao, and S Janardhanan. Local self-optimizing control with average loss minimization. *Industrial & Engineering Chemistry Research*, 47(4):1150–1158, 2008.

[33] Ramprasad Yelchuru and Sigurd Skogestad. Convex formulations for optimal selection of controlled variables and measurements using mixed integer quadratic programming. *Journal of Process control*, 22(6):995–1007, 2012.

[34] Yi Cao. Bidirectional branch and bound for average loss minimization, 2023. URL https://www.mathworks.com/matlabcentral/fileexchange/25870-bidirectional-branch-and-bound-for-average-loss-minimization.

[35] Yi Cao. Bidirectional branch and bound solvers for worst case loss minimization, 2023. URL https://www.mathworks.com/matlabcentral/fileexchange/22632-bidirectional-branch-and-bound-solvers-for-worst-case-loss-minimization.

[36] A. Maarleveld and J.E. Rijnsdorp. Constraint control on distillation columns. *Automatica*, 6 (1):51–58, 1970. ISSN 0005-1098. doi: https://doi.org/10.1016/0005-1098(70)90074-9. URL https://www.sciencedirect.com/science/article/pii/0005109870900749.

[37] Karl Johan Åström and Richard M Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2021.

[38] Randall LeVeque. Finite difference methods for differential equations, 1998.

[39] Boyun Gou, William C Lyons, and Ali Ghalambor. Petroleum production engineering, 2007.

[40] Bin Hu. Characterizing gas-lift instabilities. *Master of Science Thesis, NTNU*, 2004.

[41] Jim Cahill and Mikhail Ilchenko. Controlling surge in centrifugal compressors, Nov 2020. URL https://www.emersonautomationexperts.com/2019/control-safety-systems/controlling-surge-centrifugal-compressors/.

[42] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019. doi: 10.1007/s12532-018-0139-4.

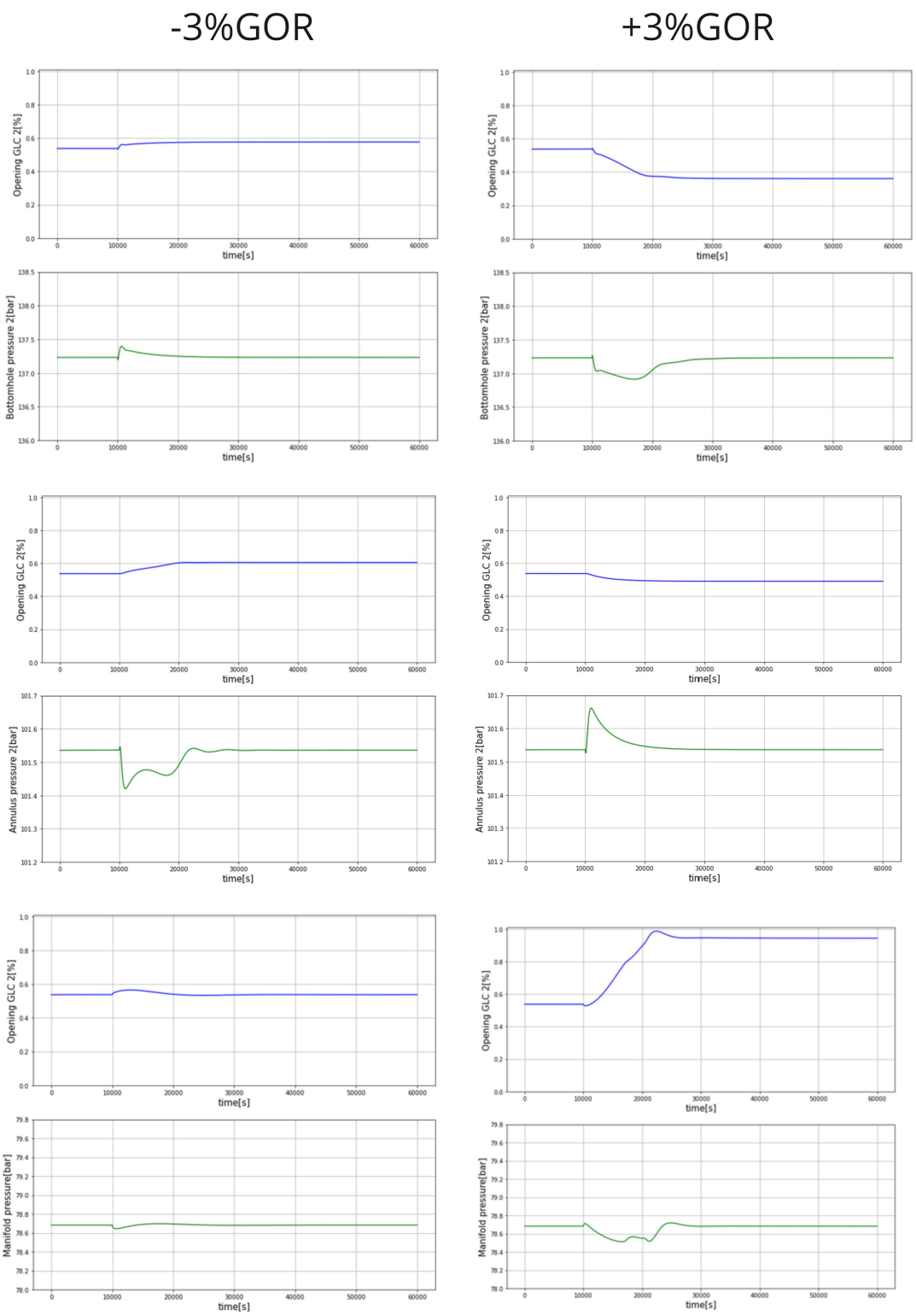
A Appendix A

A.1 Mearument combinations with related loss, proposed by Branch and Bounds.

Table A.1: The proposed measurement sets proposed by the different bracket and bounds methods.

Worst case loss BAB	Loss
p_{ai2}&p_{bh2}	323.79
Average loss BAB	Loss
p_{ai2}&p_{ai6}	2.5597
p_{ai6}&p_{d3}	2.5603
p_{ai2}&p_{d3}	3.3297
p_{bh6}&p_{d3}	3.9807
Worst case loss partial BAB 2	Loss
p_{ai6}&p_{d3}	30.5995
p_{ai2}&p_{d3}	39.5692
p_{bh6}&p_{d3}	47.5047
w_{pg2}&p_{d3}	59.0685
Worst case loss partial BAB 3	Loss
p_{ai6}&p_{bh6} &p_{d3}	1.4000
p_{ai6}&p_{d3} &w_{pg6}	1.6766
p_{ai6}&p_{d3} &p_{wh6}	2.9605
p_{ai6}&p_{d3} &w_{po6}	24.3740
Worst case loss partial BAB 4	Loss
p_{ai2}&p_{ai6}&p_{bh6} &p_{d3}	0.0050
p_{ai2}&p_{ai6}&p_{wh6} &p_{d3}	0.0253
p_{ai2}&p_{ai6}&w_{pg6} &p_{d3}	1.2274
p_{ai2}&p_{ai6}&w_{po6} &p_{d3}	4.3604
Average loss partial BAB 2	Loss
p_{ai2}&p_{d3}	0.0061
p_{wh6}&p_{d3}	0.5523
p_{bh6}&p_{d3}	0.8757
p_{ai2}&p_{ai6}	2.5597
Average loss partial BAB 3	Loss
p_{ai2}&p_{bh2} &p_{d3}	0.0585
p_{ai2}&p_{bh2} &p_{ai6}	0.7006
p_{ai2}&p_{ai6} &p_{d3}	1.6250
p_{ai2}&p_{d3} &p_{gs}	1.9595
Average loss partial BAB 4	Loss
p_{ai2}&p_{wh6}&p_{bh2} &p_{d3}	0.0001
p_{ai2}&p_{wh6}&p_{bh2} &p_{ai6}	0.0004
p_{ai2}&p_{wh6}&p_{ai6} &p_{gs}	0.0008
p_{ai2}&p_{wh6}&p_{bh2} &p_{gs}	0.0178

A.2 One manipulated variable, single controlled variable simulation results.



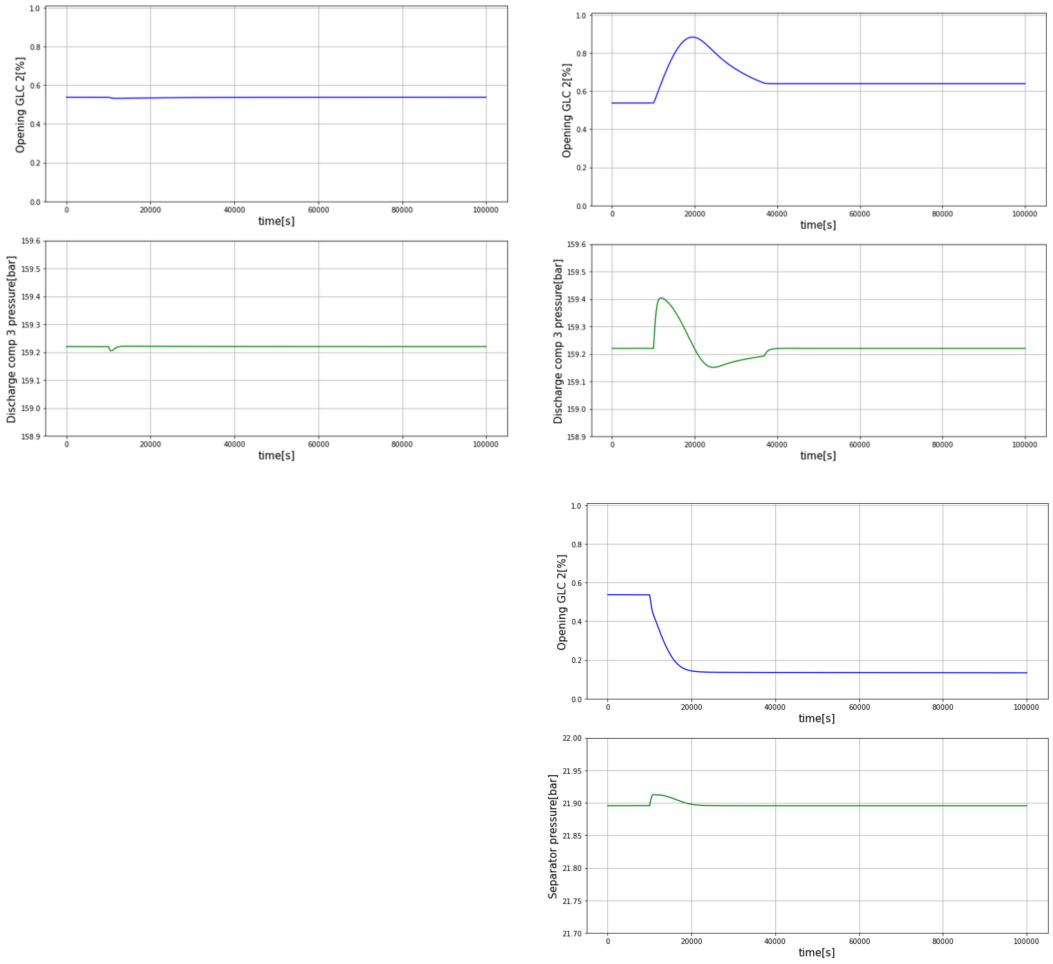
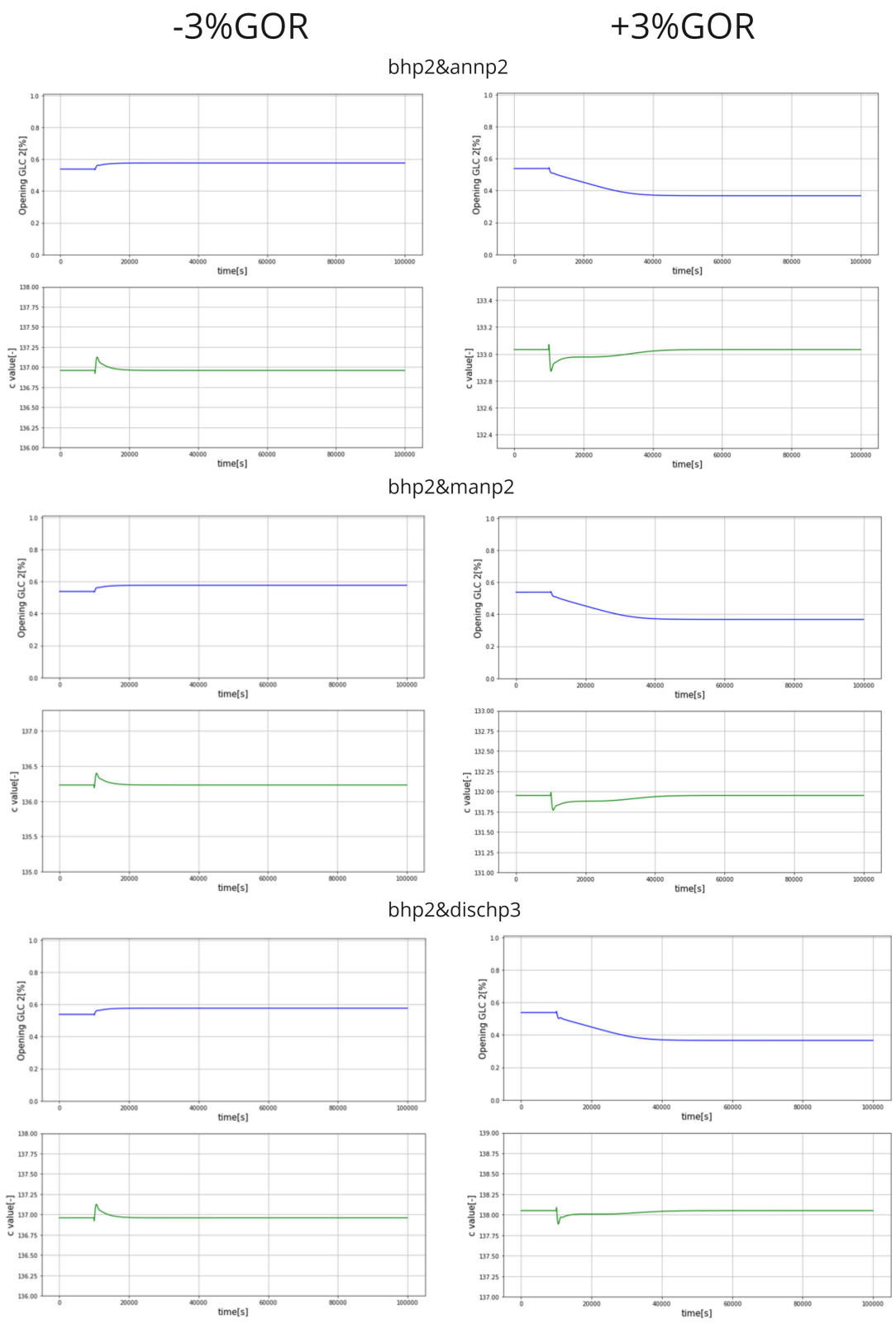


Figure A.1: Simulation results for one manipulated variable controlling single measurement

A.3 Nullspace method simulations results.



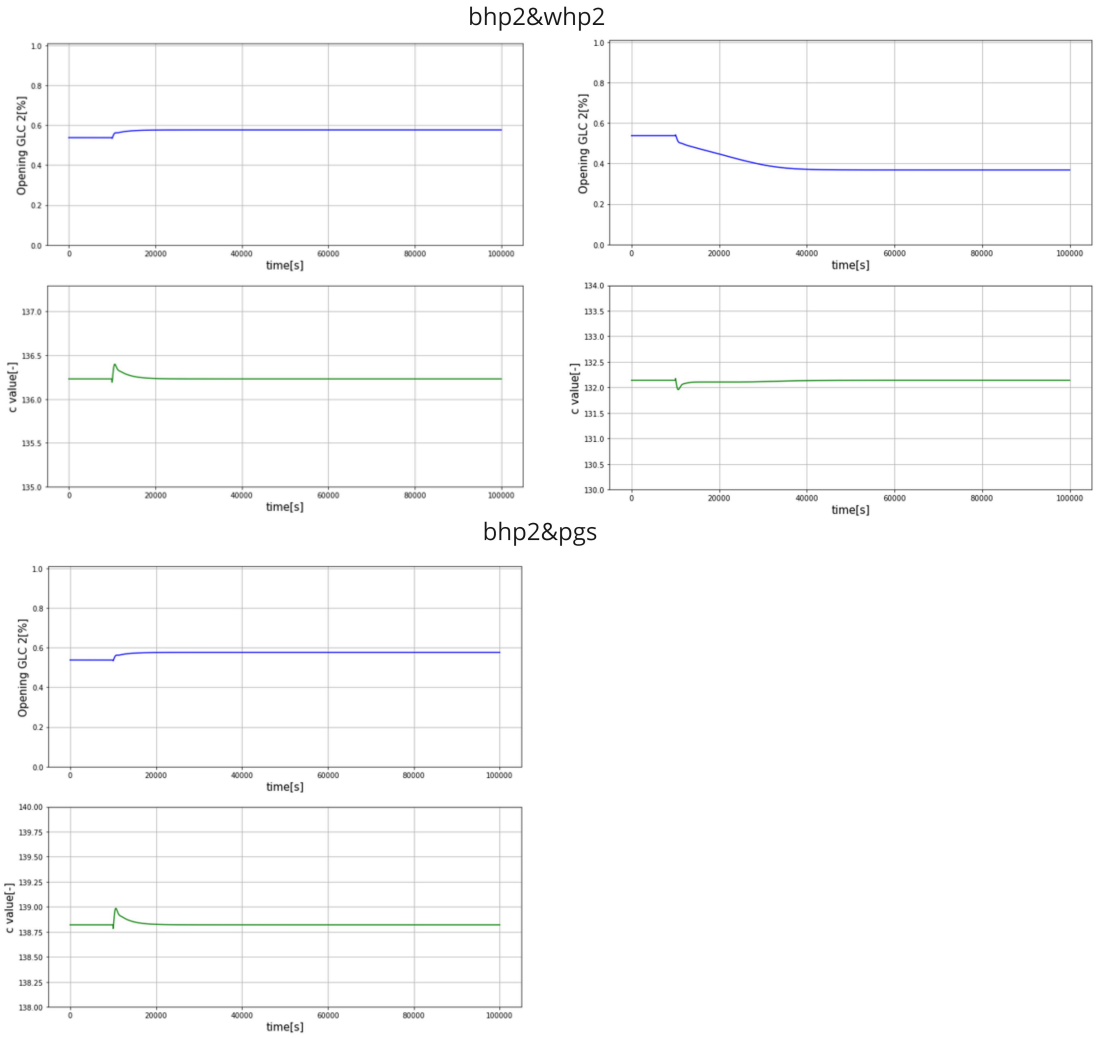
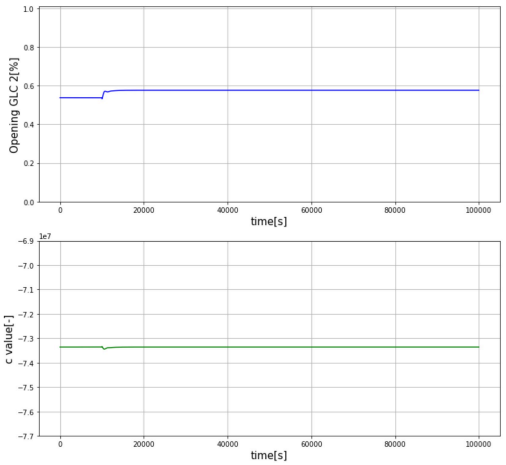


Figure A.2: Simulation results for nullspace implementation

A.4 Exact local method simulations results.

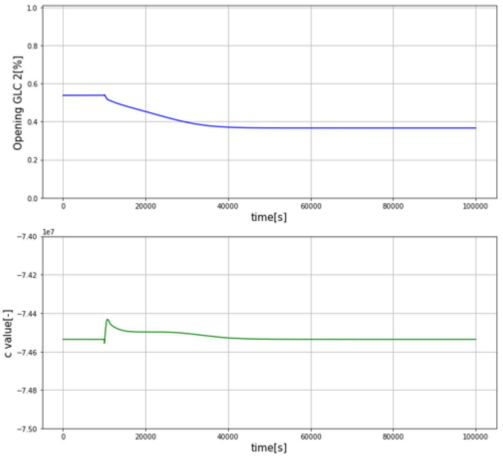
-3%GOR

bhp2&whp2

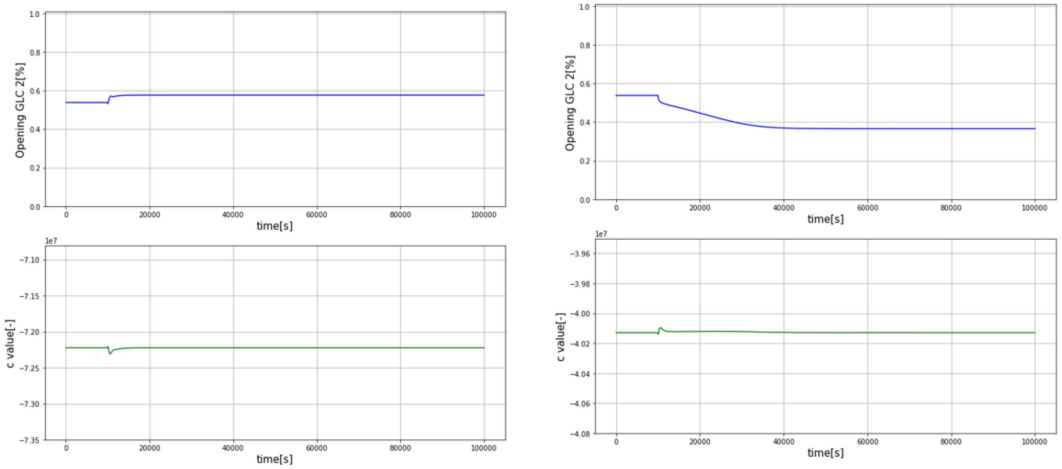


+3%GOR

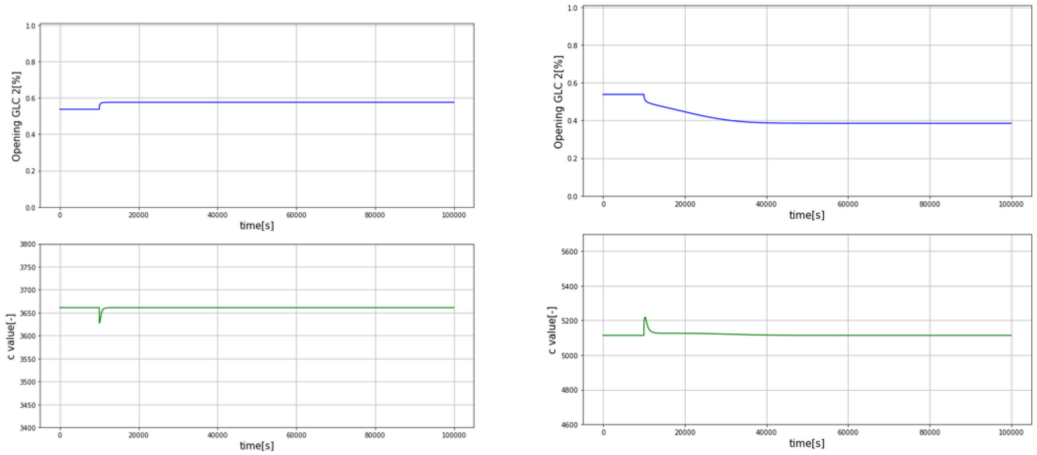
bhp2&dischp3



bhp2&totgGl



pg2&totgGl



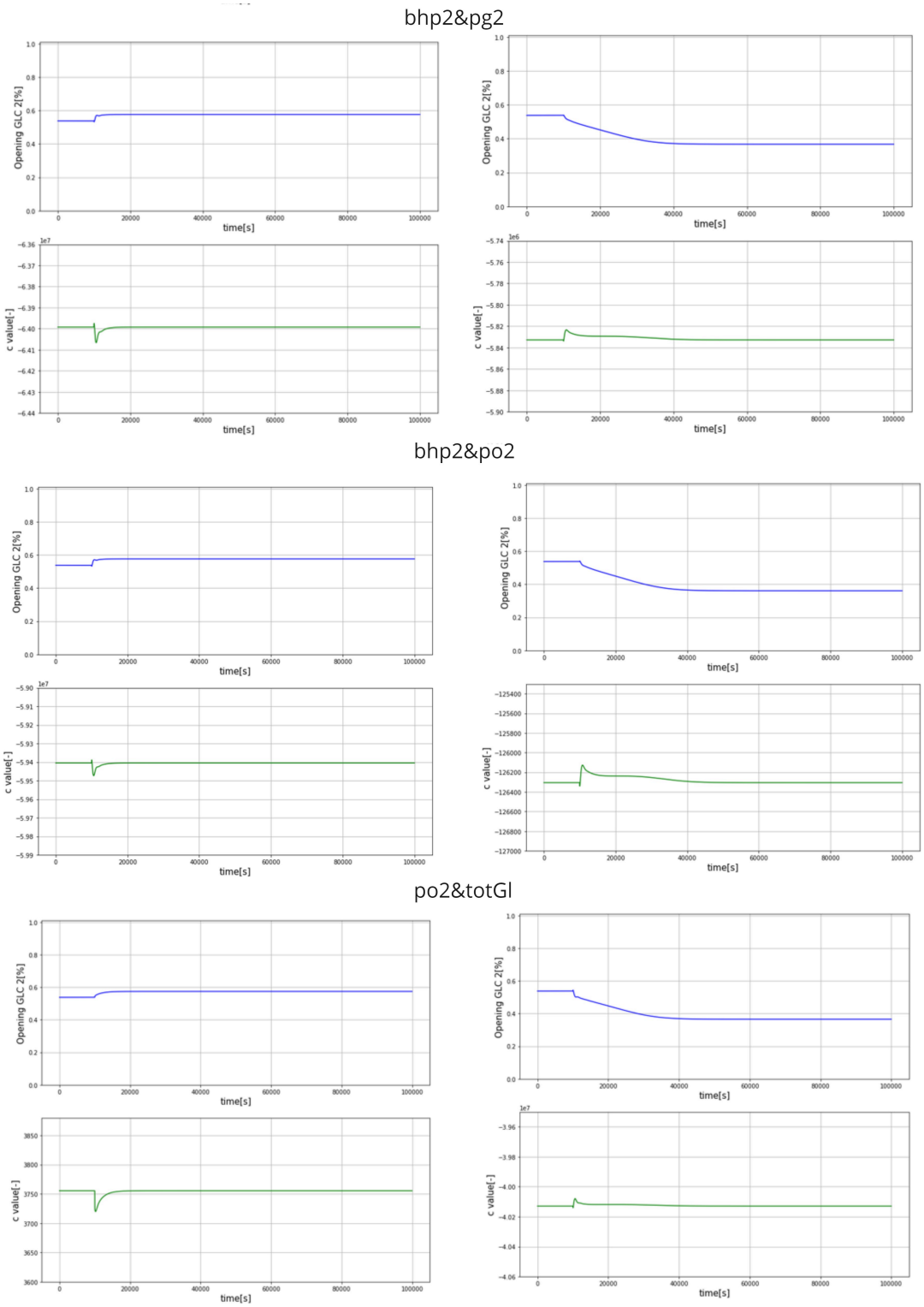


Figure A.3: Simulation results for exact local method implementation

B Appendix B

B.1 GyuImplemetation.py

This code calculates the G_u^y.

```
1
2  import numpy as np
3  from sys import path
4  path.append(r"C:/Users/Bruker/Documents/CASADIPython/casadi-windows-py38-v3.5.5-64
       bit")
5  from casadi import *
6  import casadi as ca
7  from tabulate import tabulate
8  from texttable import Texttable
9  import latextable
10 from decimal import Decimal
11 import xlsxwriter
12
13 # Call the parameters
14 import ParameterSOCN
15 import SimulatorSOCN
16 #par now represents the dictionary defined in parameter function
17 par = ParameterSOCN.Params_6wells()
18
19
20
21
22 import pandas as pd
23
24
25 #Retrieve the lower and upper bounds for the differential states(x), algebraic
       states(z) and
26 #controlled variables(u). Data listed in excel, comma separated files.
27 lbx = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbx6Sep.csv',header=None).values.
       reshape(-1)
28 lbz = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbz6Sep.csv',header=None).values.
       reshape(-1)
29 lbu = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbu6Sep.csv',header=None).values.
       reshape(-1)
30 ubx = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubx6Sep.csv',header=None).values.
       reshape(-1)
31 ubz = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubz6Sep.csv',header=None).values.
       reshape(-1)
32 ubu = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubu6Sep.csv',header=None).values.
       reshape(-1)
33
34
35
36
37
38 #Define the parameter intial values(constant, if not manually changed)
39 p0 = ca.vertcat(par['GOR'],par['wmax_gl'],par['wmax_pg'],par['Powmax_glcom']
40                 ,par['p_go'],par['p_oo'],par['omega1'],par['omega2'],par['omega3'
       ])
41
42 ep = 1e-8 #Perturbation factor
43
44
45 x_store = []
46 z_store = []
47 u_store = []
48
49
50
```

```
#Retrieve the model equations from the simulator
F,x_var, z_var, u_var, p_var, alg, dif, L, g_var =  SimulatorSOCN.
    CentralizedSimulator_F(par)
t_span = np.arange(40000)


#Function returning the data from pertrubating the valve opening,
#small change in either GLC2 or GLC6
def Optimizer(valve, eps):
    #Retrieve the initial values of the states
    x0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/x06Sep.csv',header=None).
    values.reshape(-1)
    z0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/z06Sep.csv',header=None).
    values.reshape(-1)
    u0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/u06Sep.csv',header=None).
    values.reshape(-1)
    uk = u0
    xf = x0
    zk = z0

    #Make list to store all the data
    GLC2case = []
    #Change in openings
    ##Change GLC##

    for k in t_span:


        #Solving the initial value problem
        inputs = ca.vertcat(uk, p0)
        Fk = F(x0 = xf, z0 = zk, p = inputs)
        #Retrieving the differential states
        xf = (Fk['xf']).full()
        #Retrieving the algebraic states
        zk = (Fk['zf']).full()
        zk[79] = 10 #Make sure the active constraint on the produced gas is active

        #Append results
        x_store.append(xf)
        u_store.append(uk)
        z_store.append(zk)
        #Perturbation of the valve opening
        if k == 1:
            diff = uk[valve]*eps
            uk[valve] = uk[valve] + diff

    GLC2case.append(zk[7]) #Wellhead pressure 2   1
    GLC2case.append(zk[1]) #Annulus pressure 2    2
    GLC2case.append(zk[19]) #Bottomhole pressure 2  3
    GLC2case.append(zk[11]) #Wellhead pressure 6   4
    GLC2case.append(zk[5]) #Annulus pressure 6    5
    GLC2case.append(zk[23]) #Bottomhole pressure 6   6
    GLC2case.append(zk[74]) #Manifold pressure   7
    GLC2case.append(xf[29]) #Discharge pressure comp 3   8
    GLC2case.append(xf[20]) #Separator pressure   9
    GLC2case.append(zk[55]) #Oil flow 2   10
    GLC2case.append(zk[49]) #Gas flow 2   11
    GLC2case.append(zk[59]) #Oil flow 6   12
    GLC2case.append(zk[53]) #Gas flow 6   13
    GLC2case.append(zk[78]) #Oil out separator  14
    GLC2case.append(zk[79]) #Produced gas   15
    GLC2case.append(zk[107]) #Tot gaslift  16

    return GLC2case, diff
```

```
111
112
113 #Use finite difference to obtain the how the CV's change with perturbing the
    valves
114 def GetGy(valve1, valve2, eps):
115     #finite difference for first valve
116     Delta_y1 = []
117     Delta_y2 = []
118     Listu1, Delta_u1 = Optimizer(valve1, eps)
119     Listu2, Delta_u2 = Optimizer(valve2, eps)
120     ListNom, Delta_nom = Optimizer(valve1, 0)
121     for i in range(len(Listu1)):
122         Delta_y1.append((Listu1[i][0] - ListNom[i][0])/Delta_u1)
123     for i in range(len(Listu2)):
124         Delta_y2.append((Listu2[i][0] - ListNom[i][0])/Delta_u2)
125
126     return Delta_y1, Delta_y2
127
128
129 #From the data, get the Guy matrix on correct form
130 def GetGyMatrix(valve1, valve2, eps):
131
132     GyMat = np.zeros((16,2)) #16
133     array11, array22 = GetGy(valve1,valve2,eps)
134     #Reshape arrays
135     array1 = np.array(array11)
136     array2 = np.array(array22)
137     array1 = array1.reshape((16,1))
138     array2 = array2.reshape((16,1))
139
140     GyMat[:, 0] = array1[:, 0]
141     GyMat[:, 1] = array2[:, 0]
142
143     return GyMat
144
145 mat = Gymat(1,5,ep)
146 #Export to Excel for further use in BAB methods in matlab
147 workbook = xlsxwriter.Workbook('DataForBandB/Gyu.xlsx')
148
149 #add the workbook
150 worksheet = workbook.add_worksheet()
151
152 row = 0
153 col = 0
154
155 # Iterate over the data and write it out row by row.
156 for u1, u2 in (mat):
157     worksheet.write(row, col, u1)
158     worksheet.write(row, col + 1, u2)
159     row += 1
160
161 workbook.close()
162 # Close the Excel file via close method
```

B.2 GydImplementation.py

This code calculates the G_d^y.

```
1
2  import numpy as np
3  from sys import path
4  path.append(r"C:/Users/Bruker/Documents/CASADIPython/casadi-windows-py38-v3.5.5-64
       bit")
5  from casadi import *
6  import casadi as ca
7  from tabulate import tabulate
8  from texttable import Texttable
9  import latextable
10 from decimal import Decimal
11 import xlsxwriter
12
13 # Call the parameters
14 import ParameterSOCN
15 import SimulatorSOCN
16 #par now represents the dictionary defined in parameter function
17 par = ParameterSOCN.Params_6wells()
18
19
20
21
22 import pandas as pd
23 #Retrieve initial guesses for the differential states(x0), algebraic states(z0)
       and
24 #controlled variables(u0). Data listed in excel, comma separated files.
25 x0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/x06Sep.csv',header=None).values.
       reshape(-1)
26 z0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/z06Sep.csv',header=None).values.
       reshape(-1)
27 u0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/u06Sep.csv',header=None).values.
       reshape(-1)
28
29 #Retrieve the lower and upper bounds for the differential states(x), algebraic
       states(z) and
30 #controlled variables(u). Data listed in excel, comma separated files.
31 lbx = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbx6Sep.csv',header=None).values.
       reshape(-1)
32 lbz = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbz6Sep.csv',header=None).values.
       reshape(-1)
33 lbu = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbu6Sep.csv',header=None).values.
       reshape(-1)
34 ubx = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubx6Sep.csv',header=None).values.
       reshape(-1)
35 ubz = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubz6Sep.csv',header=None).values.
       reshape(-1)
36 ubu = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubu6Sep.csv',header=None).values.
       reshape(-1)
37
38
39 #Define the parameter intial values(constant, if not manually changed)
40 p0 = ca.vertcat(par['GOR'],par['wmax_gl'],par['wmax_pg'],par['Powmax_glcom'],
41                 par['p_go'],par['p_oo'],par['omega1'],par['omega2'],par['omega3'])
42
43
44
45 ep = 1e-8 #perturbation factor
46
47 x_store = []
48 z_store = []
49 u_store = []
```

```
50
51 t_span = np.arange(40000)
52
53
54 #Return how the CV's change with perturbation for the disturbances
55 def Optimizer(disturbance, eps):
56     #Retrieve return variables of the integrator function
57     F,x_var, z_var, u_var, p_var, alg, dif, L, g_var =  SimulatorSOCN.
       CentralizedSimulator_F(par)
58
59     #Retrieve the initial data
60     x0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/x06Sep.csv',header=None).
       values.reshape(-1)
61     z0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/z06Sep.csv',header=None).
       values.reshape(-1)
62     u0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/u06Sep.csv',header=None).
       values.reshape(-1)
63
64     #Define the parameter values locally
65     p0Real = [0.125  , 0.13  , 0.13 , 0.14 , 0.125 , 0.135 , 8, 10, 19, 20, 18,
       20, 20, 20]
66     p0Real1 = p0Real
67
68     uk = u0
69     xf = x0
70     zk = z0
71
72     GLC2case = []
73
74
75     for k in t_span:
76
77
78         #Solving the initial value problem
79         inputs = ca.vertcat(uk, p0Real1)
80         Fk = F(x0 = xf, z0 = zk, p = inputs)
81         #Retrieving the differential states
82         xf = (Fk['xf']).full()
83         #Retrieving the algebraic states
84         zk = (Fk['zf']).full()
85
86         zk[79] = 10 #make sure the constraint is active
87         #Append results
88         x_store.append(xf)
89         u_store.append(uk)
90         z_store.append(zk)
91         #Perturb the disturbances
92         if k == 1:
93             diff = p0Real1[disturbance]*eps
94             p0Real1[disturbance] = p0Real1[disturbance] + diff
95
96     GLC2case.append(zk[7]) #Wellhead pressure 2   1
97     GLC2case.append(zk[1]) #Annulus pressure 2    2
98     GLC2case.append(zk[19]) #Bottomhole pressure 2  3
99     GLC2case.append(zk[11]) #Wellhead pressure 6   4
100    GLC2case.append(zk[5]) #Annulus pressure 6    5
101    GLC2case.append(zk[23]) #Bottomhole pressure 6   6
102    GLC2case.append(zk[74]) #Manifold pressure   7
103    GLC2case.append(xf[29]) #Discharge pressure comp 3   8
104    GLC2case.append(xf[20]) #Separator pressure   9
105    GLC2case.append(zk[55]) #Oil flow 2   10
106    GLC2case.append(zk[49]) #Gas flow 2   11
107    GLC2case.append(zk[59]) #Oil flow 6   12
108    GLC2case.append(zk[53]) #Gas flow 6   13
```

```
109     GLC2case.append(zk[78]) #Oil out separator  14
110     GLC2case.append(zk[79]) #Produced gas   15
111     GLC2case.append(zk[107]) #Tot gaslift  16
112 
113     return GLC2case,diff
114 
115 #Use finite difference to obtain the how the CV's change with perturbing the
        disturbances
116 def GetGyd(disturbance1, disturbance2, eps):
117     #finite difference for first disturbance
118     Delta_y1 = []
119     Delta_y2 = []
120     Listu1, Delta_u1 = Optimizer(disturbance1, eps)
121     Listu2, Delta_u2 = Optimizer(disturbance2, eps)
122     ListNom, Delta_nom = Optimizer(disturbance1, 0)
123     for i in range(len(Listu1)):
124         Delta_y1.append((Listu1[i][0] - ListNom[i][0])/Delta_u1)
125     for i in range(len(Listu2)):
126         Delta_y2.append((Listu2[i][0] - ListNom[i][0])/Delta_u2)
127 
128     return Delta_y1, Delta_y2
129 
130 
131 #From the data, get the Gyd matrix on correct form
132 def GetGydMatrix(disturbance1, disturbance2, eps):
133 
134     GyMat = np.zeros((16,2)) #16
135     array11, array22 = GetGyd(disturbance1,disturbance2,eps)
136     #Reshape arrays
137     array1 = np.array(array11)
138     array2 = np.array(array22)
139     array1 = array1.reshape((16,1)) #16
140     array2 = array2.reshape((16,1)) #16
141 
142     GyMat[:, 0] = array1[:, 0]
143     GyMat[:, 1] = array2[:, 0]
144 
145     return GyMat
146 
147 #Retireve Gymat
148 matd = GetGydMatrix(1,5,ep)
149 
150 
151 workbook = xlsxwriter.Workbook('DataForBandB/Gyd.xlsx')
152 
153 #Add workbook to worksheet
154 worksheet = workbook.add_worksheet()
155 
156 # Use the worksheet object to write
157 # data via the write() method.
158 
159 row = 0
160 col = 0
161 
162 # Iterate over the data and write it out row by row.
163 for d1, d2 in (matd):
164     worksheet.write(row, col, d1)
165     worksheet.write(row, col + 1, d2)
166     row += 1
167 
168 workbook.close()
169 # Close the Excel file via the close() method.
```

B.3 JuuImplementation.py

This code calculates the J_{uu}.

```
1
2  import numpy as np
3  from sys import path
4  path.append(r"C:/Users/Bruker/Documents/CASADIPython/casadi-windows-py38-v3.5.5-64
       bit")
5  from casadi import *
6  import casadi as ca
7  from tabulate import tabulate
8  from texttable import Texttable
9  import latextable
10 from decimal import Decimal
11 import xlsxwriter
12
13
14 ep = 1e-8 #perturbation variable
15 import FiniteDiffJuu
16
17
18 #Use finite finite difference for multivariable to obtain how the cost function
       changes with valve change
19 def Juu(valve1, valve2, eps):
20     Juu = np.zeros((2,2))
21     nom, deltanom = FiniteDiffJuu.OptimizerSameu(valve2,0)
22     for i in range(2):
23         for j in range(2):
24             if i == j and i == 0:
25                 f1, delta1 = FiniteDiffJuu.OptimizerSameu(valve1,eps)
26                 f2, delta2 = FiniteDiffJuu.OptimizerSameu(valve1,-eps)
27                 Juu[i][j] = (f1 - 2*nom + f2)/(delta1**2)
28             if i == j and i == 1:
29                 f3, delta3 = FiniteDiffJuu.OptimizerSameu(valve2,eps)
30                 f4, delta4 = FiniteDiffJuu.OptimizerSameu(valve2,-eps)
31                 Juu[i][j] = (f3 - 2*nom + f4)/(delta3**2)
32             if i != j:
33                 f5, h1, k1 =  FiniteDiffJuu.OptimizerDiffu(valve1,valve2, eps,eps)
34                 f6, h,k =  FiniteDiffJuu.OptimizerDiffu(valve1,valve2, eps,0)
35                 f7, h, k =  FiniteDiffJuu.OptimizerDiffu(valve1,valve2, 0,eps)
36                 f8, h, k =  FiniteDiffJuu.OptimizerDiffu(valve1,valve2, -eps,0)
37                 f9, h, k =  FiniteDiffJuu.OptimizerDiffu(valve1,valve2, 0,-eps)
38                 f10, h, k =  FiniteDiffJuu.OptimizerDiffu(valve1,valve2, -eps,-eps
       )
39                 Juu[i][j] = (f5 - f6 - f7 + 2*nom - f8 - f9 + f10)/(2*h1*k1)
40             ###
41             #if the same manipulated variable is looked at(e.g u1 and u1)
42             # f''(x) = ( f(x+h) -2f(x) +f(x-h) )/h**2 (Second-order central)
43
44             #if different manipulated variables are looked at e.g u2 and u1
45             #f''(x,y) = ( f(x+h,y+k) - f(x+h,y) - f(x,y+k) + 2f(x,y) - f(x-h,y) -
       f(x,y-k)
46             #+ f(x-h,y-k) )/2hk
47
48     return Juu
49
50 #Retrieve Juu
51 matJuu = Juu(1,5,ep)
52 workbook = xlsxwriter.Workbook('DataForBandB/Juu.xlsx')
53
54 #Add workbook
55 worksheet = workbook.add_worksheet()
56
57 # Use the worksheet object to write
```

```
58 # data via the write() method.
59
60 row = 0
61 col = 0
62
63 # Iterate over the data and write it out row by row.
64 for u1, u2 in (matJuu):
65     worksheet.write(row, col, u1)
66     worksheet.write(row, col + 1, u2)
67     row += 1
68
69 workbook.close()
70 # Close the file with close()
```

B.4 JudImplementation.py

This code calculates the J_{ud}.

```
1
2  import numpy as np
3  from sys import path
4  path.append(r"C:/Users/Bruker/Documents/CASADIPython/casadi-windows-py38-v3.5.5-64
       bit")
5  from casadi import *
6  import casadi as ca
7  from tabulate import tabulate
8  from texttable import Texttable
9  import latextable
10 from decimal import Decimal
11 import xlsxwriter
12
13 ep = 1e-8 #perturb value
14 import FiniteDiffJud
15
16
17 def Jud(valve1, valve2, disturbance1, disturbance2,eps):
18     Jud = np.zeros((2,2))
19     nom, deltanom, deltanom1 = FiniteDiffJud.OptimizerDiffud(1,1,0,0)
20     for i in range(2):
21         for j in range(2):
22                 if i == 0 and j == 0:
23                     f1, h1, k1 =  FiniteDiffJud.OptimizerDiffud(valve1,
       disturbance1, eps,eps)
24                     f2, h,k =  FiniteDiffJud.OptimizerDiffud(valve1,disturbance1,
       eps,0)
25                     f3, h, k =  FiniteDiffJud.OptimizerDiffud(valve1,disturbance1,
        0,eps)
26                     f4, h, k =  FiniteDiffJud.OptimizerDiffud(valve1,disturbance1,
        -eps,0)
27                     f5, h, k =  FiniteDiffJud.OptimizerDiffud(valve1,disturbance1,
        0,-eps)
28                     f6, h, k =  FiniteDiffJud.OptimizerDiffud(valve1,disturbance1,
        -eps,-eps)
29                     Jud[i][j] = (f1 - f2 - f3 + 2*nom - f4 - f5 + f6)/(2*h1*k1)
30                 if i == 0 and j == 1:
31                     f1, h1, k1 =  FiniteDiffJud.OptimizerDiffud(valve1,
       disturbance2, eps,eps)
32                     f2, h,k =  FiniteDiffJud.OptimizerDiffud(valve1,disturbance2,
       eps,0)
33                     f3, h, k =  FiniteDiffJud.OptimizerDiffud(valve1,disturbance2,
        0,eps)
34                     f4, h, k =  FiniteDiffJud.OptimizerDiffud(valve1,disturbance2,
        -eps,0)
35                     f5, h, k =  FiniteDiffJud.OptimizerDiffud(valve1,disturbance2,
        0,-eps)
36                     f6, h, k =  FiniteDiffJud.OptimizerDiffud(valve1,disturbance2,
        -eps,-eps)
37                     Jud[i][j] = (f1 - f2 - f3 + 2*nom - f4 - f5 + f6)/(2*h1*k1)
38                 if i == 1 and j == 0:
39                     f1, h1, k1 =  FiniteDiffJud.OptimizerDiffud(valve2,
       disturbance1, eps,eps)
40                     f2, h,k =  FiniteDiffJud.OptimizerDiffud(valve2,disturbance1,
       eps,0)
41                     f3, h, k =  FiniteDiffJud.OptimizerDiffud(valve2,disturbance1,
        0,eps)
42                     f4, h, k =  FiniteDiffJud.OptimizerDiffud(valve2,disturbance1,
        -eps,0)
43                     f5, h, k =  FiniteDiffJud.OptimizerDiffud(valve2,disturbance1,
        0,-eps)
```

```
                    f6, h, k =  FiniteDiffJud.OptimizerDiffud(valve2,disturbance1,
    -eps,-eps)
                    Jud[i][j] = (f1 - f2 - f3 + 2*nom - f4 - f5 + f6)/(2*h1*k1)

                if i == 1 and j == 1:
                    f1, h1, k1 =  FiniteDiffJud.OptimizerDiffud(valve2,
    disturbance2, eps,eps)
                    f2, h,k =  FiniteDiffJud.OptimizerDiffud(valve2,disturbance2,
    eps,0)
                    f3, h, k =  FiniteDiffJud.OptimizerDiffud(valve2,disturbance2,
    0,eps)
                    f4, h, k =  FiniteDiffJud.OptimizerDiffud(valve2,disturbance2,
    -eps,0)
                    f5, h, k =  FiniteDiffJud.OptimizerDiffud(valve2,disturbance2,
    0,-eps)
                    f6, h, k =  FiniteDiffJud.OptimizerDiffud(valve2,disturbance2,
    -eps,-eps)
                    Jud[i][j] = (f1 - f2 - f3 + 2*nom - f4 - f5 + f6)/(2*h1*k1)
                #Different so will use
            #f''(x,y) = ( f(x+h,y+k) - f(x+h,y) - f(x,y+k) + 2f(x,y) - f(x-h,y) -
    f(x,y-k)
            #+ f(x-h,y-k) )/2hk

    return Jud
#retrieve Jud
matJud = Jud(1,5,1,5,ep)


workbook = xlsxwriter.Workbook('DataForBandB/Jud.xlsx')

#Add workbook
worksheet = workbook.add_worksheet()


row = 0
col = 0

# Iterate over the data and write it out row by row.
for u1, u2 in (matJud):
    worksheet.write(row, col, u1)
    worksheet.write(row, col + 1, u2)
    row += 1

workbook.close()
# Close the workbook
```

B.5 Wd.py

This code calculates the W_d.

```
1
2  import numpy as np
3  from sys import path
4  path.append(r"C:/Users/Bruker/Documents/CASADIPython/casadi-windows-py38-v3.5.5-64
       bit")
5  from casadi import *
6  import casadi as ca
7  from tabulate import tabulate
8  from texttable import Texttable
9  import latextable
10 from decimal import Decimal
11 import xlsxwriter
12
13 # Call the parameters
14 import ParameterSOCN
15
16 #par now represents the dictionary defined in parameter function
17 par = ParameterSOCN.Params_6wells()
18
19
20 #Define the parameter intial values(constant, if not manually changed)
21 p0 = ca.vertcat(par['GOR'],par['wmax_gl'],par['wmax_pg'],par['Powmax_glcom'],
22                 par['p_go'],par['p_oo'],par['omega1'],par['omega2'],par['omega3'])
23
24 #Define the disturbance matrix
25 def Wd():
26     Wd = np.zeros((2,2))
27     Wd[0][0] =  (p0[1]/100)*3
28     Wd[1][1] = (p0[5]/100)*2
29     return Wd
30
31 #retireve Wd
32 matWd = Wd()
33
34 workbook = xlsxwriter.Workbook('DataForBandB/Wd.xlsx')
35
36 #Add the workbook
37 worksheet = workbook.add_worksheet()
38
39 # Use the worksheet object to write
40 # data via the write() method.
41
42 row = 0
43 col = 0
44
45 # Iterate over the data and write it out row by row.
46 for d1, d2 in (matWd):
47     worksheet.write(row, col, d1)
48     worksheet.write(row, col + 1, d2)
49     row += 1
50
51 workbook.close()
```

B.6 Wn.py

This code calculates the W_n.

```
1
2  import numpy as np
3  from sys import path
4  path.append(r"C:/Users/Bruker/Documents/CASADIPython/casadi-windows-py38-v3.5.5-64
       bit")
5  from casadi import *
6  import casadi as ca
7  from tabulate import tabulate
8  from texttable import Texttable
9  import latextable
10 from decimal import Decimal
11 import xlsxwriter
12
13 # Call the parameters
14 import ParameterSOCN
15 import SimulatorSOCN
16 #par now represents the dictionary defined in parameter function
17 par = ParameterSOCN.Params_6wells()
18
19
20
21
22 import pandas as pd
23 #Retrieve initial guesses for the differential states(x0), algebraic states(z0)
       and
24 #controlled variables(u0). Data listed in excel, comma separated files.
25 x0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/x06Sep.csv',header=None).values.
       reshape(-1)
26 z0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/z06Sep.csv',header=None).values.
       reshape(-1)
27 u0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/u06Sep.csv',header=None).values.
       reshape(-1)
28
29 #Retrieve the lower and upper bounds for the differential states(x), algebraic
       states(z) and
30 #controlled variables(u). Data listed in excel, comma separated files.
31 lbx = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbx6Sep.csv',header=None).values.
       reshape(-1)
32 lbz = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbz6Sep.csv',header=None).values.
       reshape(-1)
33 lbu = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbu6Sep.csv',header=None).values.
       reshape(-1)
34 ubx = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubx6Sep.csv',header=None).values.
       reshape(-1)
35 ubz = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubz6Sep.csv',header=None).values.
       reshape(-1)
36 ubu = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubu6Sep.csv',header=None).values.
       reshape(-1)
37
38 x00 = x0
39 z00 = z0
40 u00 = u0
41
42
43
44 #Define the parameter intial values(constant, if not manually changed)
45 p0 = ca.vertcat(par['GOR'],par['wmax_gl'],par['wmax_pg'],par['Powmax_glcom'],
46                 par['p_go'],par['p_oo'],par['omega1'],par['omega2'],par['omega3'])
47
48 def Wn():
49     #Define error related to the measurement in %
```

```
    errorP = 0.0025
    errorF = 1
    GLC2case = []
    F,x_var, z_var, u_var, p_var, alg, dif, L, g_var =  SimulatorSOCN.
CentralizedSimulator_F(par)
    nu = u_var.shape[0]
    nx = x_var.shape[0]

    #Define equality constraints of nlp, equals the model equations
    eqcon = ca.vertcat(alg, dif)
    x = ca.vertcat(u_var, x_var, z_var)

    #Define the optimization problem(x = states, L= objective function, g =
inequality constraints, p = parameters)
    nlp = {
           'x': ca.vertcat(u_var, x_var, z_var),
           'f': L,
           'g': ca.vertcat(eqcon, g_var),
           'p': p_var,
           }

    #Define upper/lower bounds for the inequality constraints
    lbg = ca.vertcat(np.full(eqcon.shape, 0), np.full(g_var.shape, -ca.inf))
    ubg = ca.vertcat(np.full(eqcon.shape, 0), np.full(g_var.shape, 0))

    #Use IPOPT(interior point optimizer) fromt the CasADI framework to solve the
nlp
    opt_inst = ca.nlpsol('opt_inst', 'ipopt', nlp)

    #Extract the solution of the optimization, feed ipopt initial values, lower
and upper bounds
    opt_res = (opt_inst(p=p0,x0 = ca.vertcat(u0,x0,z0), lbx = ca.vertcat(lbu,lbx,
lbz), ubx = ca.vertcat(ubu,ubx,ubz), lbg=lbg, ubg=ubg))
    #u0,x0,z0
    states = opt_res['x']
    cost = opt_res['f']
    Wn = np.eye((15))

    GLC2case.append(states[59].full()) #Wellhead pressure 2
    GLC2case.append(states[53].full()) #Annulus pressure 2
    GLC2case.append(states[71].full()) #Bottomhole pressure 2
    GLC2case.append(states[63].full()) #Wellhead pressure 6
    GLC2case.append(states[57].full()) #Annulus pressure 6
    GLC2case.append(states[75].full()) #Bottomhole pressure 6
    GLC2case.append(states[126].full()) #Manifold pressure
    GLC2case.append(states[49].full()) #Discharge pressure comp 3
    GLC2case.append(states[40].full()) #Separator pressure
    GLC2case.append(states[107].full()) #Oil flow 2
    GLC2case.append(states[101].full()) #Gas flow 2
    GLC2case.append(states[111].full()) #Oil flow 6
    GLC2case.append(states[105].full()) #Gas flow 6
    GLC2case.append(states[130].full()) #Oil out separator
    GLC2case.append(states[131].full()) #Produced gas
    GLC2case.append(states[136].full()) #Tot gaslift
    #Use the nominal state to find the measurement error related to the variables
    for i in range(len(GLC2case)):
        for j in range(len(GLC2case)):
            if i == j and i <= 8: #8
                Wn[i][j] = (GLC2case[i]/100)*errorP
            if i == j and i > 7:
                Wn[i][j] = (GLC2case[i]/100)*errorF

    return Wn

```

```
109
110 matWn = Wn()
111
112 workbook = xlsxwriter.Workbook('DataForBandB/Wn.xlsx')
113
114 #Add workbook
115 worksheet = workbook.add_worksheet()
116
117 row = 0
118 col = 0
119
120 # Iterate over the data and write it out row by row.
121 for d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14,d15 in (matWn):
122     worksheet.write(row, col, d1)
123     worksheet.write(row, col + 1, d2)
124     worksheet.write(row, col + 2, d3)
125     worksheet.write(row, col + 3, d4)
126     worksheet.write(row, col + 4, d5)
127     worksheet.write(row, col + 5, d6)
128     worksheet.write(row, col + 6, d7)
129     worksheet.write(row, col + 7, d8)
130     worksheet.write(row, col + 8, d9)
131     worksheet.write(row, col + 8, d9)
132     worksheet.write(row, col + 9, d10)
133     worksheet.write(row, col + 10, d11)
134     worksheet.write(row, col + 11, d12)
135     worksheet.write(row, col + 12, d13)
136     worksheet.write(row, col + 13, d14)
137     worksheet.write(row, col + 14, d15)
138
139     row += 1
140
141 #Close workbook
142 workbook.close()
```

B.7 FiniteDiffJuu.py

This code provides the necessary step data for obtaining J_{uu}.

```

import numpy as np
from sys import path
path.append(r"C:/Users/Bruker/Documents/CASADIPython/casadi-windows-py38-v3.5.5-64 bit")
from casadi import *
import casadi as ca
from tabulate import tabulate
from texttable import Texttable
import latextable
from decimal import Decimal

# Call the parameters
import ParameterSOCN
import SimulatorSOCN
#par now represents the dictionary defined in parameter function
par = ParameterSOCN.Params_6wells()




import pandas as pd
#Retrieve initial guesses for the differential states(x0), algebraic states(z0) and
#controlled variables(u0). Data listed in excel, comma separated files.
x0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/x06Sep.csv',header=None).values.reshape(-1)
z0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/z06Sep.csv',header=None).values.reshape(-1)
u0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/u06Sep.csv',header=None).values.reshape(-1)

#Retrieve the lower and upper bounds for the differential states(x), algebraic states(z) and
#controlled variables(u). Data listed in excel, comma separated files.
lbx = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbx6Sep.csv',header=None).values.reshape(-1)
lbz = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbz6Sep.csv',header=None).values.reshape(-1)
lbu = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbu6Sep.csv',header=None).values.reshape(-1)
ubx = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubx6Sep.csv',header=None).values.reshape(-1)
ubz = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubz6Sep.csv',header=None).values.reshape(-1)
ubu = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubu6Sep.csv',header=None).values.reshape(-1)

x00 = x0
z00 = z0
u00 = u0



#Define the parameter intial values(constant, if not manually changed)
p0 = ca.vertcat(par['GOR'],par['wmax_gl'],par['wmax_pg'],par['Powmax_glcom'],par['p_go'],par['p_oo'],par['omega1'],par['omega2'],par['omega3'])



#funtion if evaluating same manipulated variable
```

```
49 def OptimizerSameu(valve, eps):
50     #Retrieve return variables of the integrator function
51     F,x_var, z_var, u_var, p_var, alg, dif, L, g_var =  SimulatorSOCN.
       CentralizedSimulator_F(par)
52     #Change in openings
53     ##Change GLC##
54     ubu1 = []
55     lbu1 = []
56     for c in u0: #Make local variables and keep all other manipulated constant
57         lbu1.append(c)
58     for d in u0:
59         ubu1.append(d)
60 
61     GLC2case = []
62     lbufunc = lbu1
63     ubufunc = ubu1
64     #Perturb the valve in metione
65     Delta_u = u0[valve]*eps
66     const = u0[valve]*(1+eps) #Pretrubation of valve opening for finite difference
67     #if eps != 0: #If change is 0 we want nominal bounds
68     lbufunc[valve] = const
69     ubufunc[valve] = const
70     #Call the simulator
71     ################### Optimizer ###################
72     #addCons =
73     #Get shape of controlled, and differential states
74     nu = u_var.shape[0]
75     nx = x_var.shape[0]
76 
77     #Define equality constraints of nlp, equals the model equations
78     eqcon = ca.vertcat(alg, dif)
79     x = ca.vertcat(u_var, x_var, z_var)
80 
81     #Define the optimization problem(x = states, L= objective function, g =
       inequality constraints, p = parameters)
82     nlp = {
83            'x': ca.vertcat(u_var, x_var, z_var),
84            'f': L,
85            'g': ca.vertcat(eqcon, g_var),
86            'p': p_var,
87            }
88 
89     #Define upper/lower bounds for the inequality constraints
90     lbg = ca.vertcat(np.full(eqcon.shape, 0), np.full(g_var.shape, -ca.inf))
91     ubg = ca.vertcat(np.full(eqcon.shape, 0), np.full(g_var.shape, 0))
92 
93     #Use IPOPT(interior point optimizer) fromt the CasADI framework to solve the
       nlp
94     opt_inst = ca.nlpsol('opt_inst', 'ipopt', nlp)
95 
96     #Extract the solution of the optimization, feed ipopt initial values, lower
       and upper bounds
97     opt_res = (opt_inst(p=p0,x0 = ca.vertcat(u0,x0,z0), lbx = ca.vertcat(lbufunc,
       lbx,lbz), ubx = ca.vertcat(ubufunc,ubx,ubz), lbg=lbg, ubg=ubg))
98     #u0,x0,z0
99     states = opt_res['x']
100    cost = opt_res['f']
101 
102    return cost, Delta_u
103 
104 #funtion of evaluating different manipulated variables
105 def OptimizerDiffu(valve1, valve2, eps1, eps2):
106    #Retrieve return variables of the integrator function
```

```
107     F,x_var, z_var, u_var, p_var, alg, dif, L, g_var =  SimulatorSOCN.
        CentralizedSimulator_F(par)
108     #Change in openings
109     ##Change GLC##
110     ubu1 = []
111     lbu1 = []
112     for c in u0: #Make local variables and keep all other manipulated constant
113         lbu1.append(c)
114     for d in u0:
115         ubu1.append(d)
116 
117     GLC2case = []
118     lbufunc = lbu1
119     ubufunc = ubu1
120     Delta_u1 = u0[valve1]*eps1
121     Delta_u2 = u0[valve1]*eps2
122     const1 = u0[valve1]*(1+eps1) #Pretrubation of valve opening for finite
        difference
123     const2 = u0[valve2]*(1+eps2)
124     #if eps != 0: #If change is 0 we want nominal bounds
125     lbufunc[valve1] = const1
126     ubufunc[valve1] = const1
127     lbufunc[valve2] = const2
128     ubufunc[valve2] = const2
129     #Call the simulator
130     ################### Optimizer ###################
131     #addCons =
132     #Get shape of controlled, and differential states
133     nu = u_var.shape[0]
134     nx = x_var.shape[0]
135 
136     #Define equality constraints of nlp, equals the model equations
137     eqcon = ca.vertcat(alg, dif)
138     x = ca.vertcat(u_var, x_var, z_var)
139 
140     #Define the optimization problem(x = states, L= objective function, g =
        inequality constraints, p = parameters)
141     nlp = {
142            'x': ca.vertcat(u_var, x_var, z_var),
143            'f': L,
144            'g': ca.vertcat(eqcon, g_var),
145            'p': p_var,
146            }
147 
148     #Define upper/lower bounds for the inequality constraints
149     lbg = ca.vertcat(np.full(eqcon.shape, 0), np.full(g_var.shape, -ca.inf))
150     ubg = ca.vertcat(np.full(eqcon.shape, 0), np.full(g_var.shape, 0))
151 
152     #Use IPOPT(interior point optimizer) fromt the CasADI framework to solve the
        nlp
153     opt_inst = ca.nlpsol('opt_inst', 'ipopt', nlp)
154 
155     #Extract the solution of the optimization, feed ipopt initial values, lower
        and upper bounds
156     opt_res = (opt_inst(p=p0,x0 = ca.vertcat(u0,x0,z0), lbx = ca.vertcat(lbufunc,
        lbx,lbz), ubx = ca.vertcat(ubufunc,ubx,ubz), lbg=lbg, ubg=ubg))
157     #u0,x0,z0
158     states = opt_res['x']
159     cost = opt_res['f']
160 
161 
162     return cost, Delta_u1, Delta_u2
```

B.8 FiniteDiffJud.py

This code provides the necessary step data for obtaining J_{ud}.

```
1
2  import numpy as np
3  from sys import path
4  path.append(r"C:/Users/Bruker/Documents/CASADIPython/casadi-windows-py38-v3.5.5-64
       bit")
5  from casadi import *
6  import casadi as ca
7  from tabulate import tabulate
8  from texttable import Texttable
9  import latextable
10 from decimal import Decimal
11
12 # Call the parameters
13 import ParameterSOCN
14 import SimulatorSOCN
15 #par now represents the dictionary defined in parameter function
16 par = ParameterSOCN.Params_6wells()
17
18
19
20
21 import pandas as pd
22 #Retrieve initial guesses for the differential states(x0), algebraic states(z0)
       and
23 #controlled variables(u0). Data listed in excel, comma separated files.
24 x0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/x06Sep.csv',header=None).values.
       reshape(-1)
25 z0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/z06Sep.csv',header=None).values.
       reshape(-1)
26 u0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/u06Sep.csv',header=None).values.
       reshape(-1)
27
28 #Retrieve the lower and upper bounds for the differential states(x), algebraic
       states(z) and
29 #controlled variables(u). Data listed in excel, comma separated files.
30 lbx = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbx6Sep.csv',header=None).values.
       reshape(-1)
31 lbz = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbz6Sep.csv',header=None).values.
       reshape(-1)
32 lbu = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbu6Sep.csv',header=None).values.
       reshape(-1)
33 ubx = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubx6Sep.csv',header=None).values.
       reshape(-1)
34 ubz = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubz6Sep.csv',header=None).values.
       reshape(-1)
35 ubu = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubu6Sep.csv',header=None).values.
       reshape(-1)
36
37 x00 = x0
38 z00 = z0
39 u00 = u0
40
41
42
43 #Define the parameter intial values(constant, if not manually changed)
44 p0 = ca.vertcat(par['GOR'],par['wmax_gl'],par['wmax_pg'],par['Powmax_glcom'],par['
       p_go'],par['p_oo'],par['omega1'],par['omega2'],par['omega3'])
45 p0Real = [0.125  , 0.13 , 0.13, 0.14, 0.125, 0.135, 8, 10, 19, 20, 18, 20, 20, 20]
46 ep = 1e-8
47
48
```

```
#Function for perturbing manipulated variable and disturbance at same time
def OptimizerDiffud(valve, disturbance, eps1, eps2):
    #Retrieve return variables of the integrator function
    F,x_var, z_var, u_var, p_var, alg, dif, L, g_var =  SimulatorSOCN.
    CentralizedSimulator_F(par)
    #Change in openings
    ##Change GLC##
    ubu1 = []
    lbu1 = []
    dist = []
    for c in u0: #Make local variables and keep all other manipulated constant
        lbu1.append(c)
    for d in u0:
        ubu1.append(d)
    for e in p0Real:
        dist.append(e)

    GLC2case = []
    lbufunc = lbu1
    ubufunc = ubu1



    #perturb the disturbance and valve opening
    Delta_d = dist[disturbance]*eps2
    Delta_u = u0[valve]*eps1

    constu = u0[valve]*(1+eps1) #Pretrubation of valve opening for finite
    difference
    constd = dist[disturbance]*(1+eps2) #Pretrubation of disturbance opening for
    finite difference

    #if eps != 0: #If change is 0 we want nominal bounds
    lbufunc[valve] = constu
    ubufunc[valve] = constu
    dist[disturbance] = constd


    ################### Optimizer ###################

    #Get shape of controlled, and differential states
    nu = u_var.shape[0]
    nx = x_var.shape[0]

    #Define equality constraints of nlp, equals the model equations
    eqcon = ca.vertcat(alg, dif)
    x = ca.vertcat(u_var, x_var, z_var)

    #Define the optimization problem(x = states, L= objective function, g =
    inequality constraints, p = parameters)
    nlp = {
           'x': ca.vertcat(u_var, x_var, z_var),
           'f': L,
           'g': ca.vertcat(eqcon, g_var),
           'p': p_var,
           }

    #Define upper/lower bounds for the inequality constraints
    lbg = ca.vertcat(np.full(eqcon.shape, 0), np.full(g_var.shape, -ca.inf))
    ubg = ca.vertcat(np.full(eqcon.shape, 0), np.full(g_var.shape, 0))

    #Use IPOPT(interior point optimizer) fromt the CasADI framework to solve the
    nlp
    opt_inst = ca.nlpsol('opt_inst', 'ipopt', nlp)
```

```
108
109     #Extract the solution of the optimization, feed ipopt initial values, lower
        and upper bounds
110     opt_res = (opt_inst(p= dist,x0 = ca.vertcat(u0,x0,z0), lbx = ca.vertcat(
        lbufunc,lbx,lbz), ubx = ca.vertcat(ubufunc,ubx,ubz), lbg=lbg, ubg=ubg))
111     #u0,x0,z0
112     states = opt_res['x']
113     cost = opt_res['f']
114
115
116     return cost, Delta_u, Delta_d
```

B.9 H from linearized model

This code provides the necessary step data for obtaining H in case 2.

```
# -*- coding: utf-8 -*-

import numpy as np
from numpy.linalg import inv
import GydImplementation
import GyuImplementation
import JuuImplementation
import JudImplementation
import Wd
import Wn
ep = 1e-8 #perturb value

#Get Gyu
Gmat = GyuImplementation.GetGyMatrix(1,5,ep)
#Get Juu
Juu = JuuImplementation.Juu(1,5, ep)
#Get Jud
Jud = JudImplementation.Jud(1,5,1,5,ep)
#Get Gyd
Gmatd = GydImplementation.GetGydMatrix(1,5,ep)
#Obtain F from the linear realationship
Fmat = np.dot(-Gmat,np.dot(inv(Juu), Jud)) + Gmatd
#Get Wd
Wd = Wd.Wd()
#Get Wn
Wn = Wn.Wn()




######################Optimal measurement combinations
    ##############################
#Worst case Branch and Bound, annulus 2 and bottomhole 2
F1 = np.array([[Fmat[1][0], Fmat[1][1]],[Fmat[2][0], Fmat[2][1]]])
FWd1 = np.matmul(F1, Wd)
Wn1 = np.array([[Wn[1][1], 0], [0, Wn[2][2]]])
Y1 = np.concatenate((FWd1, Wn1.T), axis = 1)
#Y = [[-0.13019906  0.05559134  0.00253839  0.        ]
 #   [-0.34231843  0.02344421  0.          0.00343078]]
Gy1 = np.array([[Gmat[1][0], Gmat[1][1]], [Gmat[2][0], Gmat[2][1]]])
Yt1 = np.transpose(Y1)
h1 = np.matmul(inv(np.matmul(Y1,Yt1)),Gy1).transpose()
print(h1)
print(Y1)
#       ai2          Bh2
#[[ 3583.10465771 -1449.41787407]
# [ -146.18640435    67.56021995]]

#Average loss Branch and Bound, annulus 2 and annulus 6
F2 = np.array([[Fmat[1][0], Fmat[1][1]],[Fmat[4][0], Fmat[4][1]]])
print(F2)
FWd2 = np.matmul(F2, Wd)
Wn2 = np.array([[Wn[1][1], 0], [0, Wn[4][4]]])
Y2 = np.concatenate((FWd2, Wn2.T), axis = 1)
Gy2 = np.array([[Gmat[1][0], Gmat[1][1]], [Gmat[4][0], Gmat[4][1]]])
Yt2 = np.transpose(Y2)
h2 = np.matmul(inv(np.matmul(Y2,Yt2)),Gy2).transpose()
#Y = [[-0.13019906  0.05559134  0.00253839  0.        ]
 #[ 0.02353136 -0.10664106  0.          0.0025349 ]]
print(h2)
print(Y2)
```

```
#      ai2             ai6
#[[413.16600486 326.50490982]
 #[305.9743484  663.475804  ]]

#Worst case Partial Branch and Bound(2var), discharge pressure and annulus 6
F3 = np.array([[Fmat[7][0], Fmat[7][1]],[Fmat[4][0], Fmat[4][1]]])
FWd3 = np.matmul(F3, Wd)
Wn3 = np.array([[ Wn[7][7], 0], [0,  Wn[4][4]]])
Y3 = np.concatenate((FWd3, Wn3.T), axis = 1)
Gy3 = np.array([[Gmat[7][0], Gmat[7][1]], [Gmat[4][0], Gmat[4][1]]])
Yt3 = np.transpose(Y3)
h3 = np.matmul(inv(np.matmul(Y3,Yt3)),Gy3).transpose()
#Y = [[-0.02940493 -0.06080479  0.00396529  0.        ]
# [ 0.02353136 -0.10664106  0.          0.0025349 ]]
print(h3)
print(Y3)
#     pdisch         ai6
#[[-6976.59161527  3401.78133543]
# [-8339.24832552  4481.00108404]]

#Worst case Partial Branch and Bound(3var), discharge pressure and annulus 6 and
    bottomhole 6
F4 = np.array([[Fmat[7][0], Fmat[7][1]],[Fmat[4][0], Fmat[4][1]], [Fmat[5][0],
    Fmat[5][1]]])
FWd4 = np.matmul(F4, Wd)
Wn4 = np.array([[Wn[7][7], 0,0], [0, Wn[4][4],0], [0, 0, Wn[5][5]]])
Y4 = np.concatenate((FWd4, Wn4.T), axis = 1)
Gy4 = np.array([[Gmat[7][0], Gmat[7][1]], [Gmat[4][0], Gmat[4][1]], [Gmat[5][0],
    Gmat[5][1]]])
Yt4 = np.transpose(Y4)
h4 = np.matmul(inv(np.matmul(Y4,Yt4)),Gy4).transpose()
#Y = [[-0.02940493 -0.06080479  0.00396529  0.          0.        ]
 #[ 0.02353136 -0.10664106  0.          0.0025349   0.        ]
 #[ 0.09889795 -0.2016167   0.          0.          0.00344215]]
print(h4)
print(Y4)
#        DischP          ai6             bh6
#[[-234922.1903769   476847.10061302 -181679.62737955]
# [-549333.90812753 1128132.39347296 -431189.32210637]]

#Worst case Partial Branch and Bound(4var), discharge pressure and annulus 6 and
    bottomhole 6 and annulus 2
F5 = np.array([[Fmat[7][0], Fmat[7][1]],[Fmat[4][0], Fmat[4][1]], [Fmat[5][0],
    Fmat[5][1]],[Fmat[1][0], Fmat[1][1]]])
FWd5 = np.matmul(F5, Wd)
Wn5 = np.array([[Wn[7][7], 0,0,0], [0, Wn[4][4],0,0], [0, 0, Wn[5][5],0], [0, 0,
    0, Wn[1][1]]])
Y5 = np.concatenate((FWd5, Wn5.T), axis = 1)
Gy5 = np.array([[Gmat[7][0], Gmat[7][1]], [Gmat[4][0], Gmat[4][1]], [Gmat[5][0],
    Gmat[5][1]], [Gmat[1][0], Gmat[1][1]]])
Yt5 = np.transpose(Y5)
h5 = np.matmul(inv(np.matmul(Y5,Yt5)),Gy5).transpose()
print(h5)
#Y = [[-0.02940493 -0.06080479  0.00396529  0.          0.          0.        ]
# [ 0.02353136 -0.10664106  0.          0.0025349   0.          0.        ]
# [ 0.09889795 -0.2016167   0.          0.          0.00344215  0.        ]
# [-0.13019906  0.05559134  0.          0.          0.          0.00253839]]
print(Y5)
#      DischP             ai6             bh6              ai2
#[[-805763.43006082  131567.44007523  291161.30688082  427066.88982393]
# [-691151.33978905 1042352.54566528 -313718.6801493   106098.72807374]]

#Average loss Partial Branch and Bound(2var), discharge pressure and annulus 2
F6 = np.array([[Fmat[7][0], Fmat[7][1]],[Fmat[1][0], Fmat[1][1]]])
```

```
FWd6 = np.matmul(F6, Wd)
Wn6 = np.array([[Wn[7][7], 0], [0, Wn[1][1]]])
Y6 = np.concatenate((FWd6, Wn6.T), axis = 1)
Gy6 = np.array([[Gmat[7][0], Gmat[7][1]], [Gmat[1][0], Gmat[1][1]]])
Yt6 = np.transpose(Y6)
h6 = np.matmul(inv(np.matmul(Y6,Yt6)),Gy6).transpose()
print(Y6)
#Y = [[-0.02940493 -0.06080479  0.00396529  0.        ]
# [-0.13019906  0.05559134  0.          0.00253839]]
print(h6)
#    DischP        ai2
#[[-2704.12302706   327.18670482]
# [-2675.80879344    68.22602697]]


#Average loss Partial Branch and Bound(3var), discharge pressure and annulus 2 and bottomhole 2
F7 = np.array([[Fmat[7][0], Fmat[7][1]],[Fmat[1][0], Fmat[1][1]], [Fmat[2][0], Fmat[2][1]]])
FWd7 = np.matmul(F7, Wd)
Wn7 = np.array([[Wn[7][7], 0,0], [0, Wn[1][1],0], [0, 0, Wn[2][2]]])
Y7 = np.concatenate((FWd7, Wn7.T), axis = 1)
Gy7 = np.array([[Gmat[7][0], Gmat[7][1]], [Gmat[1][0], Gmat[1][1]], [Gmat[2][0], Gmat[2][1]]])
Yt7 = np.transpose(Y7)
h7 = np.matmul(inv(np.matmul(Y7,Yt7)),Gy7).transpose()
print(Y7)
#Y = [[-0.02940493 -0.06080479  0.00396529  0.          0.        ]
# [-0.13019906  0.05559134  0.          0.00253839  0.        ]
# [-0.34231843  0.02344421  0.          0.          0.00343078]]
print(h7)
#      DischP           ai2            bh2
#[[ -41674.87829334  -52288.11647798   23376.24685595]
# [-403800.65877812 -541499.5586333   240611.02868619]]

#Worst case Partial Branch and Bound(4var), discharge pressure and wellhead 6 and bottomhole 2 and annulus 2
F8 = np.array([[Fmat[7][0], Fmat[7][1]],[Fmat[3][0], Fmat[3][1]], [Fmat[2][0], Fmat[2][1]],[Fmat[1][0], Fmat[1][1]]])
FWd8 = np.matmul(F8, Wd)
Wn8 = np.array([[Wn[7][7], 0,0,0], [0, Wn[3][3],0,0], [0, 0, Wn[2][2],0], [0, 0, 0, Wn[1][1]]])
Y8 = np.concatenate((FWd8, Wn8.T), axis = 1)
Gy8 = np.array([[Gmat[7][0], Gmat[7][1]], [Gmat[3][0], Gmat[3][1]], [Gmat[2][0], Gmat[2][1]], [Gmat[1][0], Gmat[1][1]]])
Yt8 = np.transpose(Y8)
h8 = np.matmul(inv(np.matmul(Y8,Yt8)),Gy8).transpose()
print(Y8)
#Y = [[-0.02940493 -0.06080479  0.00396529  0.          0.          0.        ]
 #[ 0.04422688  0.13007002  0.          0.00201958  0.          0.        ]
 #[-0.34231843  0.02344421  0.          0.          0.00343078  0.        ]
 #[-0.13019906  0.05559134  0.          0.          0.          0.00253839]]
print(h8)
#    DischP            Wh6              BH2            ai2
#[[-686015.79742427 -613181.02538563 -334371.87860306  825709.36229966]
# [-659260.97418188 -243106.42626252   98775.47820129 -193401.98141469]]
```

B.10 SimulatorSOCN.py

This code is primarily based on the previous work, related to the modelling of the system done in the authors project thesis[1]. The file includes all the model equations and the integrator.

```
#Simulation file
#Constructs the integrator


import numpy as np
from casadi import *

def CentralizedSimulator_F(par):

    ## Retriving the parameters from Param function ##

    #Wells
    n_w = par['n_w']
    L_w = par['L_w']
    H_w = par['H_w']
    D_w = par['D_w']
    L_bh = par['L_bh']
    H_bh = par['H_bh']
    D_bh = par['D_bh']
    L_a = par['L_a']
    H_a = par['H_a']
    D_a = par['D_a']
    rho_o = par['rho_o']
    C_iv = par['C_iv']
    C_pc = par['C_pc']
    mu_oil = par['mu_oil']
    A_w = par['A_w']
    A_bh = par['A_bh']
    V_a = par['V_a']
    p_res = MX.sym('p_res',n_w)
    PI = MX.sym('PI',n_w)
    T_a = MX.sym('T_a',n_w)
    T_w = MX.sym('T_w',n_w)
    R = par['R']
    Mw = par['Mw']

    #Riser
    T_r = par['T_r']
    L_r = par['L_r']
    A_r = par['A_r']
    H_r = par['H_r']
    D_r = par['D_r']
    C_pr = par['C_pr']
    rho_ro = par['rho_ro']

    #Separator
    L_s = par['L_s']
    r_s = par['r_s']
    T_s = par['T_s']
    C_gs = par['C_gs']
    C_os = par['C_os']
    v_s = par['V_s']


    #Compressor
    n_c = par['n_c']
    C_in = par['C_in']
    C_out = par['C_out']
    C_rec = par['C_rec']
    T_in = par['T_in']
```

```
61      T_d = par['T_d']
62      Z_in = par['Z_in']
63      n_v = par['n_v']
64      #alphas
65      alpha_1 = par['alpha_1']
66      alpha_2 = par['alpha_2']
67      alpha_3 = par['alpha_3']
68      alpha_4 = par['alpha_4']
69      alpha_5 = par['alpha_5']
70      alpha_6 = par['alpha_6']
71      #beta
72      beta_1 = par['beta_1']
73      beta_2 = par['beta_2']
74      beta_3 = par['beta_3']
75      beta_4 = par['beta_4']
76      beta_5 = par['beta_5']
77      beta_6 = par['beta_6']
78      #gammas1(Further implementation)
79      gamma_11 = par['gamma_11']
80      gamma_21 = par['gamma_21']
81      gamma_31 = par['gamma_31']
82      #gammas2(Further implementation)
83      gamma_12 = par['gamma_12']
84      gamma_22 = par['gamma_22']
85      gamma_32 = par['gamma_32']
86      #gammas3(Further implementation)
87      gamma_13 = par['gamma_13']
88      gamma_23 = par['gamma_23']
89      gamma_33 = par['gamma_33']
90      #Dynamic Coefficients for compressor dynamic equations
91      Coef_1 = par['Coef_1']
92      Coef_2 = par['Coef_2']
93      Coef_3 = par['Coef_3']
94
95      #Gaslift
96      C_iv = par['C_iv']
97      C_gl = par['C_gl']
98      L_gl = par['L_gl']
99      r_gl = par['r_gl']
100
101
102     #Differential states
103     #Well system
104     m_ga = MX.sym('m_ga',n_w) #Mass gas in annulus[ton] (23-28) x
105     m_gt = MX.sym('m_gt',n_w) #Mass gas in tubing[ton]  (29-34) x
106     m_ot = MX.sym('m_ot',n_w) #Mass oil in tubing[ton] (35-40) x
107     #Riser
108     m_gr = MX.sym('m_gr',1) #Mass gas in riser[ton] (41) x
109     m_or = MX.sym('m_or',1) #Mass oil in riser[ton] (42) x
110     #Separator
111     p_gs = MX.sym('p_gs',1) #Pressure of gas in separator[bar] (43)
112     h_ls = MX.sym('h_ls',1) #Height of oil in separator[bar] (44) #NO SOC x
113     #Compressor 1
114     p_s1 = MX.sym('p_s1',n_c) #Suction Pressure Gas lift Compressor 1[bar] (45)x
115     p_d1 = MX.sym('p_d1',n_c) #Discharge Pressure Gaslift Compressor 1[bar] (46)x
116     w_c1 = MX.sym('w_c1',n_c) #Gas massflow rate Gas-lift Compressor 1[kg/s] (47)x
117     #Compressor 2
118     p_s2 = MX.sym('p_s2',n_c) #Suction Pressure Gas lift Compressor 2[bar] (48) x
119     p_d2 = MX.sym('p_d2',n_c) #Discharge Pressure Gas lift Compressor 2[bar] (49)
        x
120     w_c2 = MX.sym('w_c2',n_c) #Gas massflow rate Gas lift Compressor 2[kg/s] (50)
        x
121     #Compressor 3
122     p_s3 = MX.sym('p_s3',n_c) #Suction Pressure Gas lift Compressor 3[bar] (51) x
```

```
123     p_d3 = MX.sym('p_d3',n_c) #Discharge Pressure of Gas lift Compressor 3[bar]
        (52)
124     w_c3 = MX.sym('w_c3',n_c) #Gas massflow rate in Gas-lift Compressor 3[kg/s]
        (53) x
125     #Gas Lift
126     m_gl = MX.sym('m_gl',1) #Mas gas in gas line[ton] (54)x
127
128
129     #Algebraic states
130     #Well
131     p_ai = MX.sym('p_ai',n_w) #Annulus pressure at injection[bar] (55-60)
132     p_wh = MX.sym('p_wh',n_w) #Wellhead pressure[bar] (61-66)
133     p_wi = MX.sym('p_wi',n_w) #Injection point pressure in tubing[bar] (67-72) x
134     p_bh = MX.sym('p_bh',n_w) #Bottom-hole pressure[bar] (73-78)
135     rho_ai = MX.sym('rho_ai',n_w) #Density of gas annulus injection point[bar]
        (79-84) x
136     rho_m = MX.sym('rho_m',n_w) #Density mixed oil/gas in tubing[kg/m^3] (85-90) x
137     w_iv = MX.sym('w_iv',n_w) #Flow gas through injection valve[kg/s] (91-96) x
138     w_pc = MX.sym('w_pc',n_w) #Flow through production choke[kg/s] (97-102) #NO
        SOC x
139     w_pg = MX.sym('w_pg',n_w) #Flow gas through production choke[kg/s] (103-108) #
        NO SOC
140     w_po = MX.sym('w_po',n_w) #Flow oil through production choke[kg/s] (109-114) #
        NO SOC
141     w_ro = MX.sym('w_ro',n_w) #Flow oil from reservoir[kg/s] (115-120) #NO SOC x
142     w_rg = MX.sym('w_rg',n_w) ##Flow gas from reservoir[kg/s] (121-126) #NO SOC x
143     #Riser
144     p_rh = MX.sym('p_rh', 1) #Pressure riser head[bar] (127) x
145     rho_r = MX.sym('rho_r',1) #density oil/gas riser[kg/m^3] (128) x
146     p_m = MX.sym('p_m', 1) #Manifold pressure[bar] (129)
147     w_pr = MX.sym('w_pr', 1) #Flow through riser valve[kg/s] (130) #NO SOC x
148     w_to = MX.sym('w_to', 1) #Flow oil through riser valve[kg/s] (131) #NO SOC x
149     w_tg = MX.sym('w_tg', 1) #Flow gas through riser valve[kg/s] (132) #NO SOC x
150     #Separator
151     w_os = MX.sym('w_os',1) #Produced oil out of separator[kg/s] (133) #NO SOCx
152     w_gs = MX.sym('w_gs',1) #Produced gas out of separator[kg/s] (134) x
153     rho_gs = MX.sym('rho_gs', 1) #Gas density in separator[kg/m^3] (135) x
154     p_os = MX.sym('p_os', 1) #Separator oil pressure[bar] (136)x
155     v_os = MX.sym('v_os', 1) #Volume of oil in separator[m^3] (137)x
156     v_gs = MX.sym('v_gs',1) #Volume of gas in separator[m^3] (138)x
157     #Compressor 1
158     w_in1 = MX.sym('w_in1',n_c) #Flow gas in compressor 1[kg/s] (139)x
159     w_out1 = MX.sym('w_out1',n_c) #Flow gas out compressor 1[kg/s] (140) #NO SOCx
160     rho_in1 = MX.sym('rho_in1',n_c) #Density gas in compressor 1[kg/m^3] (141) x
161     rho_d1 = MX.sym('rho_d1',n_c) #Density gas out compressor 1[kg/m^3] (142) x
162     Phi1 = MX.sym('Phi1',n_c) #Pressure Ratio compresor 1[-] (143)x
163     Pow1 = MX.sym('Pow1',n_c) #Power consumption compressor 1[kW] (144)x
164     y_p1 = MX.sym('y_p1',n_c) #Polytropic Head compressor 1[m] (145)x
165     n_p1 = MX.sym('n_p1',n_c)  #Polytropic Efficiency 1[%] (146)x
166     w_rec1 = MX.sym('w_rec1', n_c) #Recycle mass flow[kg/s] (147) #NO SOCx
167     #Further implementation
168     Phi_max1 = MX.sym('Phi_max1',n_c) #Max Pressure ratio (148) #NO SOCx
169     gamma_2_dummy1 = MX.sym('gamma_2_dummy1',n_c) #constraint (149) #NO SOCx
170     #Compressor 2
171     w_in2 = MX.sym('w_in2',n_c) #Flow gas in compressor 2[kg/s] (150) #NO SOCx
172     w_out2 = MX.sym('w_out2',n_c) #Flow gas out compressor 2[kg/s] (151) #NO SOCx
173     rho_in2 = MX.sym('rho_in2',n_c) #Density gas in compressor 2[kg/m^3] (152)x
174     rho_d2 = MX.sym('rho_d2',n_c) #Density gas out compressor 2[kg/m^3] (153)x
175     Phi2 = MX.sym('Phi2',n_c) #Pressure Ratio compresor 2[-] (154)x
176     Pow2 = MX.sym('Pow2',n_c) #Power consumption compressor 2[kW] (155)x
177     y_p2 = MX.sym('y_p2',n_c) #Polytropic Head compressor 2[m] (156)x
178     n_p2 = MX.sym('n_p2',n_c) #Polytropic Efficiency 2[%] (157)x
179     w_rec2 = MX.sym('w_rec2', n_c) #Recycle mass flow 2[kg/s] (158) #NO SOCx
180     #Further implementation
```

```
    Phi_max2 = MX.sym('Phi_max2',n_c) #Max pressure ratio (159) #NO SOCx
    gamma_2_dummy2 = MX.sym('gamma_2_dummy2',n_c) #constraint (160) #NO SOCx
    #Compressor 3
    w_in3 = MX.sym('w_in3',n_c) #Flow gas in compressor 2[kg/s] (161) #NO SOCx
    w_out3 = MX.sym('w_out3',n_c) #Flow gas out compressor 2[kg/s] (162) #NO SOCx
    rho_in3 = MX.sym('rho_in3',n_c) #Density gas in compressor 2[kg/m^3] (163)x
    rho_d3 = MX.sym('rho_d3',n_c) #Density gas out compressor 2[kg/m^3] (164)x
    Phi3 = MX.sym('Phi3',n_c) #Pressure Ratio compresor 2[-] (165)x
    Pow3 = MX.sym('Pow3',n_c) #Power consumption compressor 2[kW] (166)x
    y_p3 = MX.sym('y_p3',n_c) #Polytropic Head compressor 2[m] (167)x
    n_p3 = MX.sym('n_p3',n_c) #Polytropic Efficiency 2[%] (168)x
    w_rec3 = MX.sym('w_rec3', n_c) #Recycle mass flow 2[kg/s] (169) #NO SOCx
    #Further implementation
    Phi_max3 = MX.sym('Phi_max3',n_c) #Max pressure ratio (170) #NO SOCx
    gamma_2_dummy3 = MX.sym('gamma_2_dummy3',n_c) #constraint (171) #NO SOC x
    #Gl system
    w_gl = MX.sym('w_gl',n_w) #Flow through gas lift choke[kg/s] (172-177)x
    p_out = MX.sym('p_out',1) #Pressure in gas lift line[bar] (178)x
    rho_out = MX.sym('rho_out',1) #density of gas in gas lift line[kg/m^3] (179)x


    #Control input

    #Gas lift
    u_gl = MX.sym('u_gl', n_w) #Valve opening gas lift chokes[0-1] (0-5)

    #Separator
    z_ov = MX.sym('z_ov', 1) #Valve opening separator oil out[0-1] (6)

    #Compressor
    u_1 = MX.sym('u_1',n_c) #Valve opening inlet compressor 1[0-1] (7)

    #Wells
    u_pc = MX.sym('u_pc', n_w) #Valve opening production chokes[0-1] (8-13)

    #Compressor
    u_2 = MX.sym('u_2',n_c) #Valve opening inlet compressor 2, outlet 1[0-1] (14)
    u_3 = MX.sym('u_3',n_c) #Valve opening inlet compressor 3, outlet 2[0-1] (15)
    u_4 = MX.sym('u_4',n_c) #Valve opening outlet compressor 3[0-1] (16)
    u_rec1 = MX.sym('u_rec1',n_c) #Valve opening recycle compressor 1[0-1] (17)
    u_rec2 = MX.sym('u_rec2',n_c) #Valve opening recycle compressor 2[0-1] (18)
    u_rec3 = MX.sym('u_rec3',n_c) #Valve opening recycle compressor 3[0-1] (19)
    omega1 =MX.sym('omega1',n_c) #Speed of compressor 1[rad/s] (20)
    omega2 = MX.sym('omega2',n_c) #Speed of compressor 2[rad/s] (21)
    omega3 = MX.sym('omega2',n_c) #Speed of compressor 3[rad/s] (22)

    #Parameters(Possible to introduce more)
    #Wells
    GOR = MX.sym('GOR',n_w)
    #Separator
    p_go = MX.sym('p_go',1)
    p_oo = MX.sym('p_oo', 1)

    #Constraints
    wmax_gl = MX.sym('wmax_gl',1)
    wmax_pg = MX.sym('wmax_pg',1)
    Powmax_glcom = MX.sym('Powmax_glcom',1)



    #Algebraic equations
    #Wells
    f1 = -p_ai*1e5 + ((R*T_a/(V_a*Mw) + 9.81*H_a/V_a)*m_ga*1e3) #Bernoulli
```

```
244     f2 = -p_wh*1e5 + ((R*T_w/Mw)*(m_gt*1e3/(L_w*A_w + L_bh*A_bh - m_ot*1e3/rho_o))
        ) - ((m_gt*1e3+m_ot*1e3 )/(L_w*A_w))*9.81*H_w/2 #Bernoulli
245     f3 = -p_wi*1e5 + (p_wh*1e5 + 9.81/(A_w*L_w)*fmax(0,(m_ot*1e3+m_gt*1e3-rho_o*
        L_bh*A_bh))*H_w + 128*mu_oil*L_w*w_pc/(3.14*D_w**4*((m_gt*1e3 + m_ot*1e3)*p_wh
        *1e5*Mw*rho_o)/(m_ot*1e3*p_wh*1e5*Mw + rho_o*R*T_w*m_gt*1e3))) #Bernoulli/
        Hagen-Poiseuille
246     f4 = -p_bh*1e5 + (p_wi*1e5 + rho_o*9.81*H_bh + 128*mu_oil*L_bh*w_ro/(3.14*D_bh
        **4*rho_o)) #Bernoulli/Hagen-Poiseuille
247     f5 = -rho_ai*1e2 +(Mw/(R*T_a)*p_ai*1e5) #Ideal gas law    x
248     f6 = -rho_m*1e2 + ((m_gt*1e3 + m_ot*1e3)*p_wh*1e5*Mw*rho_o)/(m_ot*1e3*p_wh*1e5
        *Mw + rho_o*R*T_w*m_gt*1e3)#Relationship oil/gas  x
249     f7 = -w_iv + C_iv*sqrt(rho_ai*1e2*fmax(0.001,(p_ai*1e5 - p_wi*1e5)))#Valve
        equation
250     f8 = -w_pc + u_pc*C_pc*sqrt(rho_m*1e2*fmax(0.001,(p_wh*1e5 - p_m*1e5)))#Valve
        equation
251     f9 = -w_pg + ((m_gt*1e3)/fmax(1e-3,(m_gt*1e3+m_ot*1e3)))*w_pc #massfraction of
         gas
252     f10 = -w_po + ((m_ot*1e3)/fmax(1e-3,(m_gt*1e3+m_ot*1e3)))*w_pc #massfraction
        oil
253     f11 = -w_ro + PI*1e-6*(p_res*1e5 - p_bh*1e5)#From definition of Productivity
        index
254     f12 = -w_rg + GOR*w_ro #From definition of Gas-oil ratio
255     #Riser system
256     f15 = -p_rh*1e5 + ((R*T_r/Mw))*(m_gr*1e3/(L_r*A_r)) - ((m_gr*1e3+m_or*1e3)/(
        L_r*A_r))*9.81*H_r/2 #Bernoulli
257     f16 = -rho_r*1e2 + ((m_gr*1e3 + m_or*1e3)*p_rh*1e5*Mw*rho_ro)/(m_or*1e3*p_rh*1
        e5*Mw +rho_ro*R*T_r*m_gr*1e3)
258     f17 = -p_m*1e5 + (p_rh*1e5 + 9.81/(A_r*L_r)*(m_or*1e3+m_gr*1e3)*H_r + 128*
        mu_oil*L_r*w_pr/(np.pi*D_r**4*((m_gr*1e3+m_or*1e3) * p_rh*1e5*Mw*rho_ro) / (
        m_or*1e3*p_rh*1e5*Mw+rho_ro*R*T_r*m_gr*1e3))) #Realationship oil/gas
259     f18 = -w_pr + 1*C_pr * np.sqrt(rho_r*1e2*fmax(0.001,(p_rh*1e5-p_gs*1e5))) #
        Valve equation
260     f19 = -w_to + (m_or*1e3/(m_gr*1e3 + m_or*1e3))*w_pr #massfraction oil
261     f20 = -w_tg + (m_gr*1e3/(m_gr*1e3 + m_or*1e3))*w_pr #massfraction gas
262     #Separator system
263     f21 = -w_os + z_ov*C_os*sqrt(rho_ro*1e2*fmax(0.001,(p_os*1e5 - p_oo*1e5))) #
        Valve equation
264     f22 = -w_gs + C_gs*np.sqrt(rho_gs*1e2 *fmax(0.001,(p_gs*1e5 - p_go*1e5))) #
        Valve equation
265     f23 = -rho_gs*1e2 + (Mw/(T_s * R) * p_gs*1e5) #Ideal gas law x
266     f24 = -p_os*1e5 + p_gs*1e5 + rho_ro * 9.81 * h_ls #Bernoulli equation
267     f25 = -v_os  + ((0.5*r_s**2)*((2*np.arccos(fmax(0,(r_s-h_ls)/r_s)))-np.sin((2*
        np.arccos(fmax(0,(r_s-h_ls)/r_s))))))*L_s #Based on equation of segment/circle
        , derivative of the Area
268     f26 = -v_gs  + fmax(0,(v_s - v_os)) #Based on relation volume of gas/oil in
        separator
269     #Compressor 1
270     f27 = -w_in1 + C_in*u_1*np.sqrt(rho_in1*1e2*fmax(0.001,(p_gs*1e5 - p_s1*1e5)))
        #Valve equation x
271     f28 = -w_out1 + C_in*u_2*np.sqrt(rho_d1*1e2*fmax(0.001,(p_d1*1e5-p_s2*1e5)))#
        Valve equation x
272     f29 = -rho_in1*1e2 + (Mw/(R*T_in)*p_gs*1e5)#Ideal gas law x
273     f30 = -rho_d1*1e2 + (Mw/(R*T_d)*p_d1*1e5)#Ideal gas law x
274     f31 = -Phi1 + alpha_1 + alpha_2*omega1 + alpha_3*w_c1 + alpha_4*omega1*w_c1 +
        alpha_5*omega1*omega1 + alpha_6*w_c1*w_c1# xPolynomial realationship/
        approximation
275     f32 = -Pow1 + (y_p1/(n_p1))*w_c1#Based on how much of the potential power that
         can be utilized x
276     f33 = -y_p1*1e5 + (Z_in *R *T_in/(Mw))*(n_v/(n_v-1)) *((Phi1**((n_v-1)/n_v))
        -1)# xEquation for polytropic head
277     f34 = -n_p1*1e2 + beta_1 + beta_2*omega1 + beta_3*Phi1 + beta_4*omega1*Phi1 +
        beta_5*omega1*omega1 + beta_6*Phi1*Phi1# xPolynomial realationship/
        approximation
278     f35 = -Phi_max1 + gamma_11*(w_c1-gamma_21) + gamma_31 #Further work
```

```
    f36 = - gamma_2_dummy1 + w_c1 - ((Phi1 - gamma_31)/gamma_11) #Further work
    f37 = -w_rec1 + C_rec*u_rec1*np.sqrt(rho_d1*1e2*fmax(0.0001,(p_d1*1e5 - p_s1*1
    e5))) #Valve equation x
    #Compressor 2
    f38 = -w_in2 + C_in*u_2*np.sqrt(rho_in2*1e2*fmax(0.001,(p_d1*1e5 - p_s2*1e5)))
    #Valve equation
    f39 = -w_out2 + C_in*u_3*np.sqrt(rho_d2*1e2*fmax(0.001,(p_d2*1e5-p_s3*1e5)))#
    Valve equation
    f40 = -rho_in2*1e2 + (Mw/(R*T_in)*p_d1*1e5)#Ideal gas law x
    f41 = -rho_d2*1e2 + (Mw/(R*T_d)*p_d2*1e5)#Ideal gas law
    f42 = -Phi2 + alpha_1 + alpha_2*omega2 + alpha_3*w_c2 + alpha_4*omega2*w_c2 +
    alpha_5*omega2*omega2 + alpha_6*w_c2*w_c2#Polynomial realationship/
    approximation
    f43 = -Pow2 + (y_p2/n_p2)*w_c2#Based on how much of the potential power that
    can be utilized
    f44 = -y_p2*1e5 + (Z_in *R *T_in/(Mw))*(n_v/(n_v-1)) *((Phi2**((n_v-1)/n_v))
    -1)#Equation for polytropic head
    f45 = -n_p2*1e2 + beta_1 + beta_2*omega2 + beta_3*Phi2 + beta_4*omega2*Phi2 +
    beta_5*omega2*omega2 + beta_6*Phi2*Phi2#Polynomial realationship/approximation
    f46 = -Phi_max2 + gamma_12*(w_c2-gamma_22) + gamma_32 #Further work
    f47 = - gamma_2_dummy2 + w_c2 - ((Phi2 - gamma_32)/gamma_12) #Further work
    f48 = -w_rec2 + C_rec*u_rec2*np.sqrt(rho_d2*1e2*fmax(0.0001,(p_d2*1e5 - p_s2*1
    e5)))#Valve equation
    #Compressor 3
    f49 = -w_in3 + C_in*u_3*np.sqrt(rho_in3*1e2*fmax(0.001,(p_d2*1e5 - p_s3*1e5)))
    #Valve equation
    f50 = -w_out3 + C_in*u_4*np.sqrt(rho_d3*1e2*fmax(0.001,(p_d3*1e5-p_out*1e5)))#
    Valve equation
    f51 = -rho_in3*1e2 + (Mw/(R*T_in)*p_d2*1e5)#Ideal gas law
    f52 = -rho_d3*1e2 + (Mw/(R*T_d)*p_d3*1e5)#Ideal gas law
    f53 = -Phi3 + alpha_1 + alpha_2*omega3 + alpha_3*w_c3 + alpha_4*omega3*w_c3 +
    alpha_5*omega3*omega3 + alpha_6*w_c3*w_c3#Polynomial realationship/
    approximation
    f54 = -Pow3 + (y_p3/n_p3)*w_c3#Based on how much of the potential power that
    can be utilized
    f55 = -y_p3*1e5 + (Z_in *R *T_in/(Mw))*(n_v/(n_v-1)) *((Phi3**((n_v-1)/n_v))
    -1)#Equation for polytropic head
    f56 = -n_p3*1e2 + beta_1 + beta_2*omega3 + beta_3*Phi3 + beta_4*omega3*Phi3 +
    beta_5*omega3*omega3 + beta_6*Phi3*Phi3#Polynomial realationship/approximation
    f57 = -Phi_max3 + gamma_13*(w_c3-gamma_23) + gamma_33 #Further work
    f58 = - gamma_2_dummy3 + w_c3 - ((Phi3 - gamma_33)/gamma_13) #Further work
    f59 = -w_rec3 + C_rec*u_rec3*np.sqrt(rho_d3*1e2*fmax(0.001,(p_d3*1e5 - p_s3*1
    e5)))#Valve equation
    #Gas Lift
    f60 = -w_gl + C_gl*u_gl*np.sqrt(rho_out*1e2*fmax(0.001,(p_out*1e5 - p_ai*1e5))
    )#Valve equation
    f61 = -p_out*1e5 + R*T_d*m_gl*1e3/(Mw*np.pi*r_gl*r_gl*L_gl)#Ideal gas law x
    f62 = -rho_out*1e2 + (Mw/(R*T_d)*p_out*1e5)#Ideal gas law

    #Differential equations
    #Wells
    df1 = (w_gl - w_iv)*1e-3 #m_ga, massbalance of gas annulus
    df2 = (w_iv + w_rg - w_pg)*1e-3 #m_tg, massbalance of gas tubing
    df3 = (w_ro - w_po)*1e-3 #m_to, massbalance of oil tubing
    #Riser
    df4 = (sum(w_pg.nz) - w_tg)*1e-3#*1e-3 #m_gt, massbalance gas riser
    df5 = (sum(w_po.nz) - w_to)*1e-3#*1e-3 #m_ot, massbalance oil riser
    #Separator
    df6 = ((R*T_s/(v_gs*Mw))*(w_tg - w_gs - w_in1)) + (p_gs/(v_gs*rho_ro))*((w_to
    - w_os)*1e-4)# p_gs, based on derivative of ideal gas law
    df7 = (((w_to - w_os))/rho_ro)/(2*L_s * np.sqrt(h_ls* fmax(0,((2 *r_s)-h_ls)))
    )#h_ls, based on equation of segment/circle, derivative of the Area
    #Compressor system(diff equations)
    #Compressor 1
```

```
323     df8 = (w_in1  - w_c1 + w_rec1) * Coef_1 #p_s1, based on gas in/out of system
324     df9 = (w_c1 - w_out1 - w_rec1) *Coef_2 #p_d1, based on gas in/out of system
325     df10 = (p_s1*Phi1 - p_d1) * Coef_3 #w_c1, based on pressure difference between
         in/out
326     # Define variables for combined systems (needed only for decomposition case)
327     #Compressor 2
328     df11 = (w_in2  - w_c2 + w_rec2) * Coef_1 #p_s2, based on gas in/out of system
329     df12 = (w_c2 - w_out2 - w_rec2) *Coef_2 #p_d2, based on gas in/out of system
330     df13 = (p_s2*Phi2 - p_d2) * Coef_3 #w_c2, based on pressure difference between
         in/out
331     #Compressor 3
332     df14 = (w_in3  - w_c3 + w_rec3) * Coef_1#p_s3, based on gas in/out of system
333     df15 = (w_c3 - w_out3 - w_rec3) *Coef_2#p_d3, based on gas in/out of system
334     df16 = (p_s3*Phi3 - p_d3) * Coef_3#w_c3, based on pressure difference between
        in/out
335     #Gas lift(diff equations)
336     df17 = (w_out3 - sum(w_gl.nz))*1e-3 #m_gl, based on massbalance
337
338     #Form the DAE system
339     dif = vertcat(df1,df2,df3,df4, df5,df6,df7,df8,df9,df10,df11,df12,df13,df14,
        df15,df16,df17) #Differential equations
340     alg = vertcat(f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f15,f16,f17,f18,f19,f20,
        f21,f22,f23,f24,f25,f26,f27,f28,f29,f30,f31,f32,f33,f34,f35,f36,f37,f38,f39,
        f40,f41,f42,f43,f44,f45,f46,f47,f48,f49,f50,f51,f52,f53,f54,f55,f56,f57,f58,
        f59,f60,f61,f62) #Algebraic equations
341     x_var = vertcat(m_ga,m_gt,m_ot,m_gr, m_or,p_gs,h_ls,p_s1,p_d1, w_c1, p_s2,p_d2
        ,w_c2,p_s3,p_d3,w_c3,m_gl) #Differential states
342     z_var = vertcat(p_ai,p_wh,p_wi,p_bh,rho_ai,rho_m,w_iv,w_pc,w_pg,w_po,w_ro,w_rg
        , p_rh,rho_r,p_m,w_pr,w_to,w_tg,w_os,w_gs,rho_gs,p_os,v_os,v_gs,w_in1,w_out1,
        rho_in1,rho_d1, Phi1,Pow1,y_p1,n_p1,Phi_max1,gamma_2_dummy1,w_rec1,w_in2,
        w_out2,rho_in2,rho_d2, Phi2,Pow2,y_p2,n_p2,Phi_max2,gamma_2_dummy2,w_rec2,
        w_in3,w_out3,rho_in3,rho_d3,Phi3,Pow3,y_p3,n_p3,Phi_max3,gamma_2_dummy3,w_rec3
        ,w_gl,p_out,rho_out)#Algebraic states
343     u_var = vertcat(u_gl,z_ov,u_1,u_pc,u_2,u_3,u_4,u_rec1,u_rec2,u_rec3)#Control
        variables
344     p_var = vertcat(GOR,wmax_gl,wmax_pg,Powmax_glcom,p_go,p_oo,omega1,omega2,
        omega3)#Parameters/constraints
345
346
347     #Inequality constraints
348     g_var = vertcat((w_gs-wmax_pg),((Pow1 + Pow2 + Pow3)-Powmax_glcom),(sum(w_gl.
        nz)- wmax_gl))
349
350     #Objective function(Whant to maximize oil pruduction and minimize power consum
        ption)
351     L = -0.6*w_os + 0.03*(Pow1 + Pow2 + Pow3)
352
353     #Free variables need to be added
354     alg = substitute(alg,p_res,par['p_res'])
355     alg = substitute(alg,PI,par['PI'])
356     alg = substitute(alg,T_a,par['T_a'])
357     alg = substitute(alg,T_w,par['T_w'])
358
359     #Constructing the total DEA system, into CasADI framework
360     dae = {'x': x_var,'z': z_var,'p': vertcat(u_var,p_var),'ode': dif,'alg': alg,'
        quad': L}
361
362     #Define integration time
363     opts = {'tf': par['tf']}
364
365     #Create IDAS integrator for the DAE system
366     F = integrator('F','idas',dae,opts)
367
368     #Returns values
```

```
369     return F,x_var, z_var, u_var, p_var, alg, dif, L, g_var
```

B.11 ParameterSOCN.py

This code is primarily based on the previous work, related to the modelling of the system done in the authors project thesis[1]. The file includes all the models constant parameters in a dictionary.

```
#Parameter file
#Function returns a dictionary with the models constant parameters

import numpy as np

def Params_6wells():
    par = {}  #Dictionary to store the parameters
    par['n_w'] = 6 #Number of wells

    ##### Well Parameters ####

    #Length, height and diameter of wells[m]
    par['L_w'] = np.array([1500, 1500, 1500, 1500, 1500, 1500])
    par['H_w'] = np.array([1000,1000,1000,1000,1000,1000])
    par['D_w'] = np.array([0.121,0.121,0.121,0.121,0.121,0.121])

    #Length, height and diameter of bottom hole[m]
    par['L_bh'] = np.array([500, 500, 500, 500, 500, 500])
    par['H_bh'] = np.array([500, 500, 500, 500, 500, 500])
    par['D_bh'] = np.array([0.121,0.121,0.121,0.121,0.121,0.121])

    #Length, height and diameter of annuluses[m]
    par['L_a'] = par['L_w'] # Lenght of annuls equals length of well
    par['H_a'] = par['H_w'] # Height of annuls equals length of well
    par['D_a'] = np.array([0.189, 0.189, 0.189, 0.189, 0.189, 0.189])

    #Density oil, injection valve char and production choke valve char
    par['rho_o'] = np.array([8,8,7.9,8,8.2,8.05]) *1e2 #[kg/m^3]
    par['C_iv'] = np.array([0.1e-3,0.1e-3,0.1e-3,0.1e-3,0.1e-3,0.1e-3]) #[m^2]
    par['C_pc'] = np.array([2e-3,2e-3,2e-3,2e-3,2e-3,2e-3])  #[m^2]

    #Gas-oil ratio of wells[kg/kg], possible disturbance
    #par['GOR'] = np.array([0.1 ,0.12,0.09,0.108,0.115,0.102]) + 0.01
    #par['GOR'] = np.array([0.11 ,0.12 ,0.12,0.12,0.12,0.13]) + 0.0329
    #par['GOR'] = np.array([0.11  ,0.115 + 0.02 ,0.125  ,0.11,0.132,0.13]) +
    0.0329 #Manages + 0.025
    #par['GOR'] = np.array([0.11  ,0.115  ,0.125   ,0.111,0.131,0.13]) + 0.0329 #
    Manages + 0.055
    #par['GOR'] = np.array([0.13  ,0.12  ,0.13  , 0.12, 0.13, 0.12])
    #par['GOR'] = np.array([0.13  ,0.12 + 0.002875 * 4   ,0.13  , 0.115, 0.14,
    0.125]) + 0.0279
    #par['GOR'] = np.array([0.13  ,0.122    ,0.135  , 0.115 - 0.002875 * 4 ,
    0.127, 0.125]) + 0.027
    #par['GOR'] = np.array([0.133  ,0.128   ,0.135  , 0.115 + 0.002875 * 4 ,
    0.132, 0.13]) + 0.0263
    #par['GOR'] = np.array([0.133  ,0.128   ,0.135  , 0.125  , 0.132, 0.13]) +
    0.0263
    #par['GOR'] = np.array([0.13  ,0.125 + 0.003125 *1   ,0.13  , 0.12  , 0.129,
    0.125]) + 0.026
    #par['GOR'] = np.array([0.154  ,0.152   ,0.156   ,0.148 + 0.00379 * 1
    ,0.153, 0.154]) #Manages up to 10% increase and 2.5, 5 and 10% decrease
    #par['GOR'] = np.array([0.154  ,0.152 + 0.00152*3   ,0.156   ,0.1488   ,0.153,
     0.154]) #Manages decrease and increase of 2.5, 5 and 10% well 4 + 0.00379*4
    #par['GOR'] = np.array([0.154  ,0.152   ,0.156   ,0.1488 + 0.00379*2   ,0.153,
     0.154]) + 0.0028
    par['GOR'] = np.array([0.125    , 0.13  ,0.13  ,0.14   ,0.125  , 0.135 ])
    #par['GOR'] = np.array([0.125 , 0.13     ,0.129  ,0.142   ,0.1252 , 0.135])
    #4.36397
    #0.00013
    #0.537969
```

```
51      par['p_res'] = np.array([150,155,155,160,155,155]) #Reservoir pressure[bar]
52      par['PI'] = np.array([15,14,15,14,14,15])* 0.5 #Productvity index wells[kg s
        ^-1 bar^-1]
53      #par['PI'] = np.array([7,7,7,7,7,7])* 0.5 #Productvity index wells[kg s^-1 bar
        ^-1]
54      par['T_a'] = np.array([273, 273, 273, 273, 273, 273]) + 28 #Annulus
        temperature[K]
55      par['T_w'] = np.array([273, 273, 273, 273, 273, 273]) + 32 #Well temperature[K
        ]
56
57      #Area of well, bottom hole and volume of annulus
58      par['A_w'] = np.pi*(par['D_w']/2)**2 #[m^2]
59      par['A_bh'] = np.pi*(par['D_bh']/2)**2 #[m^2]
60      par['V_a'] = par['L_a']*(np.pi*(par['D_a']/2)**2 - np.pi*(par['D_w']/2)**2) #[
        m^3]
61      #Volume of annulus will equal the area of the total well and annulus minus the
         well
62
63
64      #Constraints
65      par['wmax_gl'] = np.array([8]) #Max gas lift
66      #par['wmax_pg'] = np.array([10]) #Max produced gas
67      par['wmax_pg'] = np.array([10]) #Max produced gas
68      #par['Powmax_glcom'] = np.array([19]) #Max power #Nominal
69      par['Powmax_glcom'] = np.array([19]) #Max power
70
71      #General parameters
72      par['R'] = 8.314 #Gas constant [m^3 Pa K^-1 mol^-1]
73      par['Mw'] = 20e-3 #Molar weighgt kg/mol
74      par['tf'] = 1 #Simulation time
75      par['mu_oil'] = 0.001 #Oil viscosity[kg m^  1   s^  1  ]
76
77
78      #### Riser System ####
79      par['L_r'] = 500 #Length of riser[m]
80      par['H_r'] = 500 #Height of riser[m]
81      par['D_r'] = 0.121 #Diameter of riser[m]
82      A_r = np.pi*(par['D_r']/2)**2
83      par['A_r'] = A_r #Area of riser[m^2]
84      par['T_r'] = 30+273 #Temperature riser[K]
85      par['C_pr'] = 0.003 #Valve char riser valve[m^2]
86      rho_ro = np.sum(par['rho_o'])/6
87      par['rho_ro'] = rho_ro #Density of oil in riser[kg/m^3]
88
89
90      #### Separator ####
91      par['L_s'] = 5 #10length Separator[m] #Oversized
92      par['r_s'] = 1.65 #radius Separator[m] #Oversized
93      par['T_s'] = 29 + 273 #Temperature Separator[K]
94      V_sep = np.pi * par['r_s']**2 * par['L_s']
95      par['V_s'] = V_sep #Volume of Separator[m3]
96      par['C_gs'] = 5.5*0.001 #Valve char gas outlet[m^2]
97      par['C_os'] = 5.5*0.001*0.5*0.5#5.5*0.001*0.5*0.5 #Valve char oil outlet[m^2]
98      par['p_go'] = 20 #pressure gas out[bar]
99      par['p_oo'] = 20 - 2 #pressure oil out[bar]
100
101     #### Compressors ####
102     par['n_c'] = 1  #This parameter is just an early implementation error.
103     par['T_d'] = 298 #Temperature out of compessor(assume heat is removed)[K]
104     par['C_in'] = 9e-4*2.93#Valve char is equal for all in/out valves[m^2]
105     par['T_in'] = 298 #Temperature inlet compressors[K]
106     par['Z_in'] = 0.9 #Compression factor(difference from ideal behaviour)[-]
107     par['n_v'] = 1.27 #Polytropic coefficient[-]
108     par['C_out'] = 1.201e-3 #Valve char out, not used at this impelementation[m^2]
```

```
109     par['C_rec'] = 1.1*3.5e-5 *2 #Recycle valve char[m^2]
110     par['omega1'] = 20
111     par['omega2'] = 20
112     par['omega3'] = 20
113     #alpha values for the approximation of pressure ratio
114     par['alpha_1'] = 1.05 * 0.745 *2.3
115     par['alpha_2'] = 0.7 * -1.4e-2 *-1
116     par['alpha_3'] = 0.3 * 0.11 * -4.09e-2
117     par['alpha_4'] = 1.75 * 0.13* 9.86e-4
118     par['alpha_5'] = 1.0 * 0.5* -4.25e-4 *-1
119     par['alpha_6'] = 300* (-0.15)* 2.45e-5 *2
120
121     #beta values for the approximation of efficiency
122     par['beta_1'] = 0.7* 9*5.91e-2 *200
123     par['beta_2'] = -2.13e-1 *2
124     par['beta_3'] = 2.93e-1
125     par['beta_4'] = 2.97e-3
126     par['beta_5'] = -2.68e-5
127     par['beta_6'] = -1.1e1 *(-0.1)*1.2 *2
128
129     #Dynamic coefficients for compressor dynamic equations
130     par['Coef_1'] = 1e4
131     par['Coef_2'] = 1e5
132     par['Coef_3'] = 1
133
134     a = 0.1#0.073
135     #gamma values for further implementation of surge and choke constraints
136     #Comp1
137     par['gamma_11'] = 250*0.015* 0.55 * a#0.085#0.085 #0.08
138     par['gamma_21'] = 0.5* 3.812e-2 * 0.1
139     par['gamma_31'] = 1.08* 0.615* 3 * 0.7
140
141     #Comp2
142     #gamma values for further implementation of surge and choke constraints
143     par['gamma_12'] = 250*0.015* 0.55 * a#0.085#0.085 #0.06
144     par['gamma_22'] = 0.5* 3.812e-2 * 0.1
145     par['gamma_32'] = 1.08* 0.615* 3 * 0.7
146
147     #Comp3
148     #gamma values for further implementation of surge and choke constraints
149     par['gamma_13'] = 250*0.015* 0.55 * a#0.085#0.085#0.09  #0.058
150     par['gamma_23'] = 0.5* 3.812e-2 * 0.1
151     par['gamma_33'] = 1.08* 0.615* 3 * 0.7
152
153     #### Gas lift ####
154     par['L_gl'] = 500 #Length gas lift line[m]
155     par['r_gl'] = 0.15 #radius gas lift line[m]
156     par['C_gl'] = np.array([5e-5,5e-5,5e-5,5e-5,5e-5,5e-5]) #Valve char gas lift
        valves[m^2]
157     par['C_iv'] = np.array([0.1e-3,0.1e-3,0.1e-3,0.1e-3,0.1e-3,0.1e-3]) * 1.35  #
        Valve char injection valves[m^2]
158
159     return par
```

B.12 Controlimplementations.py

This file shows the controller implementations related to self-optimizing and regulatory control.

```
#Main file
#Coding based on and inspired by model made by Risvan Dirza(NTNU).
#Integrates the system of equations with the use of the CasADI framework IDAS integrator.
#Optimize the system of equations with the use of the CasADI framework IPOPT nlp solver.


import numpy as np
from sys import path
path.append(r"C:/Users/Bruker/Documents/CASADIPython/casadi-windows-py38-v3.5.5-64 bit")
from casadi import *
import casadi as ca
from tabulate import tabulate
from texttable import Texttable
import latextable
from decimal import Decimal

# Call the parameters
import ParameterSOCN

#par now represents the dictionary defined in parameter function
par = ParameterSOCN.Params_6wells()




import pandas as pd
#Retrieve initial guesses for the differential states(x0), algebraic states(z0) and
#controlled variables(u0). Data listed in excel, comma separated files.
x0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/x06Sep.csv',header=None).values.reshape(-1)
z0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/z06Sep.csv',header=None).values.reshape(-1)
u0 = pd.read_csv('DatafolderSOCProdGasLimitWhN5/u06Sep.csv',header=None).values.reshape(-1)

#Retrieve the lower and upper bounds for the differential states(x), algebraic states(z) and
#controlled variables(u). Data listed in excel, comma separated files.
lbx = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbx6Sep.csv',header=None).values.reshape(-1)
lbz = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbz6Sep.csv',header=None).values.reshape(-1)
lbu = pd.read_csv('DatafolderSOCProdGasLimitWhN5/lbu6Sep.csv',header=None).values.reshape(-1)
ubx = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubx6Sep.csv',header=None).values.reshape(-1)
ubz = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubz6Sep.csv',header=None).values.reshape(-1)
ubu = pd.read_csv('DatafolderSOCProdGasLimitWhN5/ubu6Sep.csv',header=None).values.reshape(-1)



x00 = x0
z00 = z0
u00 = u0

```

```
#Define the parameter intial values(constant, if not manually changed)
p0 = ca.vertcat(par['GOR'],par['wmax_gl'],par['wmax_pg'],par['Powmax_glcom'],par['p_go'],par['p_oo'],par['omega1'],par['omega2'],par['omega3'])
p00 = ca.vertcat(par['GOR'],par['wmax_gl'],par['wmax_pg'],par['Powmax_glcom'],par['p_go'],par['p_oo'],par['omega1'],par['omega2'],par['omega3'])
#Call the simulator
import SimulatorSOCN

#Retrieve return variables of the integrator function
F,x_var, z_var, u_var, p_var, alg, dif, L, g_var =  SimulatorSOCN.CentralizedSimulator_F(par)

###For plotting of integration results ####
#t_span = np.arange(10000)
#x-values containers

m_ga1plot = []
m_ga2plot = []
m_ga3plot = []
m_ga4plot = []
m_ga5plot = []
m_ga6plot = []
m_gt1plot = []
m_gt2plot = []
m_gt3plot = []
m_gt4plot = []
m_gt5plot = []
m_gt6plot = []
m_ot1plot = []
m_ot2plot = []
m_ot3plot = []
m_ot4plot = []
m_ot5plot = []
m_ot6plot = []
m_grplot = []
m_orplot = []
p_gsplot = []
h_lsplot = []
p_s1plot = []
p_d1plot = []
w_c1plot = []
p_s2plot = []
p_d2plot = []
w_c2plot = []
p_s3plot = []
p_d3plot = []
w_c3plot = []
m_glplot = []
#u-value containers
u_gl1plot = []
u_gl2plot = []
u_gl3plot = []
u_gl4plot = []
u_gl5plot = []
u_gl6plot = []
z_ovplot = []
u_1plot = []
u_2plot = []
u_3plot = []
u_4plot = []
u_rec1plot = []
u_rec2plot = []
u_rec3plot = []
u_pc1plot = []
```

```
109 u_pc2plot = []
110 u_pc3plot = []
111 u_pc4plot = []
112 u_pc5plot = []
113 u_pc6plot = []
114 #z-calue containers
115 p_ai1plot = []
116 p_ai2plot = []
117 p_ai3plot = []
118 p_ai4plot = []
119 p_ai5plot = []
120 p_ai6plot = []
121 p_wh1plot = []
122 p_wh2plot = []
123 p_wh3plot = []
124 p_wh4plot = []
125 p_wh5plot = []
126 p_wh6plot = []
127 p_wi1plot = []
128 p_wi2plot = []
129 p_wi3plot = []
130 p_wi4plot = []
131 p_wi5plot = []
132 p_wi6plot = []
133 p_bh1plot = []
134 p_bh2plot = []
135 p_bh3plot = []
136 p_bh4plot = []
137 p_bh5plot = []
138 p_bh6plot = []
139 rho_ai1plot = []
140 rho_ai2plot = []
141 rho_ai3plot = []
142 rho_ai4plot = []
143 rho_ai5plot = []
144 rho_ai6plot = []
145 rho_m1plot = []
146 rho_m2plot = []
147 rho_m3plot = []
148 rho_m4plot = []
149 rho_m5plot = []
150 rho_m6plot = []
151 w_iv1plot = []
152 w_iv2plot = []
153 w_iv3plot = []
154 w_iv4plot = []
155 w_iv5plot = []
156 w_iv6plot = []
157 w_pc1plot = []
158 w_pc2plot = []
159 w_pc3plot = []
160 w_pc4plot = []
161 w_pc5plot = []
162 w_pc6plot = []
163 w_pg1plot = []
164 w_pg2plot = []
165 w_pg3plot = []
166 w_pg4plot = []
167 w_pg5plot = []
168 w_pg6plot = []
169 w_po1plot = []
170 w_po2plot = []
171 w_po3plot = []
172 w_po4plot = []
```

```
w_po5plot = []
w_po6plot = []
w_ro1plot = []
w_ro2plot = []
w_ro3plot = []
w_ro4plot = []
w_ro5plot = []
w_ro6plot = []
w_rg1plot = []
w_rg2plot = []
w_rg3plot = []
w_rg4plot = []
w_rg5plot = []
w_rg6plot = []
p_rhplot = []
rho_rplot = []
p_mplot = []
w_prplot = []
w_toplot = []
w_tgplot = []
w_osplot = []
w_gsplot = []
rho_gsplot = []
p_osplot = []
v_osplot = []
v_gsplot = []
w_in1plot = []
w_out1plot = []
rho_in1plot = []
rho_d1plot = []
Phi1plot = []
Pow1plot = []
y_p1plot = []
n_p1plot = []
Phi_max1plot = []
gamma_2_dummy1plot = []
w_rec1plot = []
w_in2plot = []
w_out2plot = []
rho_in2plot = []
rho_d2plot = []
Phi2plot = []
Pow2plot = []
y_p2plot = []
n_p2plot = []
Phi_max2plot = []
gamma_2_dummy2plot = []
w_rec2plot = []
w_in3plot = []
w_out3plot = []
rho_in3plot = []
rho_d3plot = []
Phi3plot = []
Pow3plot = []
y_p3plot = []
n_p3plot = []
Phi_max3plot = []
gamma_2_dummy3plot = []
w_rec3plot = []
w_gl1plot = []
w_gl2plot = []
w_gl3plot = []
w_gl4plot = []
w_gl5plot = []
```

```
237 w_gl6plot = []
238 p_outplot = []
239 rho_outplot = []
240 CostVsOpening = []
241 C1 = []
242 C2 = []
243 C3 = []
244 C4 = []
245 C5 = []
246 C6 = []
247 C7 = []
248 C8 = []
249 C9 = []
250 C10 = []
251 C11 = []
252 C12 = []
253 C13 = []
254 C14 = []
255 C15 = []
256 C16 = []
257 C17 = []
258 C18 = []
259 C19 = []
260 C20 = []
261 C21 = []
262 C22 = []
263 C23 = []
264 C24 = []
265 C25 = []
266 C26 = []
267
268 #Exact local method 2 MV/2 dist
269 B11 = []
270 B12 = []
271 B21 = []
272 B22 = []
273 B31 = []
274 B32 = []
275 B41 = []
276 B42 = []
277 B51 = []
278 B52 = []
279 B61 = []
280 B62 = []
281 B71 = []
282 B72 = []
283 B81 = []
284 B82 = []
285 B91 = []
286 B92 = []
287 B101 = []
288 B102 = []
289 B111 = []
290 B112 = []
291 B121 = []
292 B122 = []
293 B131 = []
294 B132 = []
295 B141 = []
296 B142 = []
297 B151 = []
298 B152 = []
299 B161 = []
300 B162 = []
```

```
301
302
303 #Define time span of simulation
304 t_span = np.arange(100000)
305
306 #Initialize initial values
307 uk = u0
308 xf = x0
309 zk = z0
310
311 #Make containers for storing integrator/control output
312 x_store =[]
313 z_store = []
314 u_store = []
315 p_store = []
316 error_store1 = [0]
317 error_store2 = [0,0]
318 error_store3 = [0,0]
319 error_store4 = [0]
320 error_store5 = [0]
321 error_store6 = [0]
322 error_store7 = [0]
323 error_store8 = [0]
324 error_storeA = [0]
325 error_storeB = [0]
326 error_storeC = [0]
327 error_storeD = [0]
328 error_store11 = [0]
329 error_store12 = [0]
330 error_store13 = [0]
331 error_store14 = [0]
332 error_storeC1 = [0]
333 error_storeC2 = [0]
334 error_storeC3 = [0]
335 error_storeC11 = [0]
336 error_storeC22 = [0]
337 error_storeC33 = [0]
338 B = []
339 u_1plot = []
340 u_2plot = []
341 u_3plot = []
342 u_4plot = []
343 u_5plot = []
344 u_6plot = []
345 u_7plot = []
346 u_8plot = []
347 u_11plot = []
348 u_12plot = []
349 u_13plot = []
350 u_14plot = []
351 lowestpoint = []
352 u_Aplot = []
353 u_Bplot = []
354 u_Cplot = []
355 u_Dplot = []
356 cplot = []
357 error_store = [0,0]
358 timer = 1000
359
360 ################### Integrator ###################
361 Condition = 0
362 for k in t_span:
363     #Change to simulate disturbance in GOR(Possible to implement disturbance in
    more variables)
```

```
364     #p0[1]  += 0.0013
365     #Simulate change in GOR for different wells
366     #if k == 5000:
367         #p0[1]   += 0.01
368         #p0[1]   -= 0.02
369         #p0[1]   += 0.0013*3
370         #p0[5]   += 0.00135*2
371         #p0[2] += 0.1
372         #p0[3] += 0.1
373         #p0[4] += 0.1
374         #p0[5] -= 0.00135*2
375 
376 
377     #Simulate change in valve opening for different wells
378     #if k == 10000:
379         #uk[0] -= 0.4
380         #uk[1] += uk[1]*10**(-5)
381         #uk[2] += uk[5]*10**(-5)
382         #uk[3] += 0.1
383         #uk[4] += 0.1
384         #uk[5] += 0.1
385 
386 
387     #Solving the initial value problem
388     inputs = ca.vertcat(uk, p0)
389     Fk = F(x0 = xf, z0 = zk, p = inputs)
390     #Retrieving the differential states
391     xf = (Fk['xf']).full()
392     #Retrieving the algebraic states
393     zk = (Fk['zf']).full()
394 
395     #zk[79] = 10
396     #Append results
397     x_store.append(xf)
398     u_store.append(uk)
399     z_store.append(zk)
400 
401 
402     #Exact local method 2MV/2d from branch and bound F caluclated from linearized
        model
403 
404 
405     #Annulus P and bottomhole P well 2 GLC2
406     #PI controller, tuned with SIMC rules
407     h2 = 3583.10465771*zk[1] - 1449.41787407*zk[19]#-7.98501232*zk[1] -
        64.82349538*zk[19]
408     h_sp2 =  1.64907832*10**(5)#-9.70655713*10**(3)#72.5945 #Nominal optimal
        wellhead pressure
409     tau12 = 101#720
410     tauC2 = 200#4000#300#Controller time, can be changed up or down depending on
        needs for fast control or smooth control
411     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
412     Kp2 = 1/(308.12053295 *tauC2)#Proportional gain 0.45315772
413     Ki2 = Kp2/tauI2 #Integral gain
414     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
415     #Calculate new controller output
416     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[-1] ))))
417     #Update the controller output for z_ov(oil valve separator)
418     uk[1] = u2
419     #Store all errors, to be used for previous errors
420     error_store2.append(error2)
421     u_2plot.append(u2)
```

```
422
423     #Annulus P and bottomhole P well 2 GLC6
424     #PI controller, tuned with SIMC rules
425     h3 = -146.18640435*zk[1] + 67.56021995*zk[19]#2.60067824*zk[1] + 16.08713207*
        zk[19]
426     h_sp3 =  -5.57177646*10**(3)#2.47171494*10**(3)#72.5945 #Nominal optimal
        wellhead pressure
427     tau13 = 185#424
428     tauC3 = 200#4000#300#Controller time, can be changed up or down depending on
        needs for fast control or smooth control
429     tauI3 = ca.fmin(tau13, 4*tauC3) #Integral time, corresponding to SIMC rules
        for integration processes.
430     Kp3 = 1/(0.33032675*tauC3)#Proportional gain 0.04527007
431     Ki3 = Kp3/tauI3 #Integral gain
432     error3 = (h_sp3 - h3) #Difference between setpoint and measured   value
433     #Calculate new controller output
434     u3 = ca.fmax(0, ca.fmin(1, (u_store[k-1][5] + (Kp3*error3 + Ki3*error3 - Kp3*
        error_store3[-1] ))))
435     #Update the controller output for z_ov(oil valve separator)
436     uk[5] = u3
437     #Store all errors, to be used for previous errors
438     error_store3.append(error3)
439     u_5plot.append(u3)
440
441
442
443     #Annulus P2 and annulus P6 GLC2
444     #PI controller, tuned with SIMC rules
445     h2 = 413.16600486*zk[1] + 326.50490982*zk[5]#22.88897358*zk[1] - 22.67672343*
        zk[5]
446     h_sp2 = 7.50573395*10**(4)#24.7206142#72.5945 #Nominal optimal wellhead
        pressure
447     tau12 = 67
448     tauC2 = 63#300#Controller time, can be changed up or down depending on needs
        for fast control or smooth control
449     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
450     Kp2 = 1/(39.72975464 *tauC2)#Proportional gain 2.2514496
451     Ki2 = Kp2/tauI2 #Integral gain
452     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
453     #Calculate new controller output
454     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[-1] ))))
455     #Update the controller output for z_ov(oil valve separator)
456     uk[1] = u2
457     #Store all errors, to be used for previous errors
458     error_store2.append(error2)
459     u_2plot.append(u2)
460
461     #Annulus P2 and annulus P6 GLC6
462     #PI controller, tuned with SIMC rules
463     h3 = 305.9743484*zk[1] + 663.475804*zk[5]#-22.18074525*zk[1] + 24.49319288*zk
        [5]
464     h_sp3 = 9.83410265*10**(4)#2.31372351*10**(2)#72.5945 #Nominal optimal
        wellhead pressure
465     tau13 = 65
466     tauC3 = 65#300#Controller time, can be changed up or down depending on needs
        for fast control or smooth control
467     tauI3 = ca.fmin(tau13, 4*tauC3) #Integral time, corresponding to SIMC rules
        for integration processes.
468     Kp3 = 1/(63.61388852 *tauC3)#Proportional gain 2.39501819
469     Ki3 = Kp3/tauI3 #Integral gain
470     error3 = (h_sp3 - h3) #Difference between setpoint and measured   value
471     #Calculate new controller output
```

```
472     u3 = ca.fmax(0, ca.fmin(1, (u_store[k-1][5] + (Kp3*error3 + Ki3*error3 - Kp3*
        error_store3[-1] ))))
473     #Update the controller output for z_ov(oil valve separator)
474     uk[5] = u3
475     #Store all errors, to be used for previous errors
476     error_store3.append(error3)
477     u_5plot.append(u3)
478 
479     #
480 
481     #Annulus P6 and DischargeP GLC2
482     #PI controller, tuned with SIMC rules
483     h2 = -6976.59161527*xf[29] + 3401.78133543*zk[5]#-4.3656668*xf[29] -
        26.84908555*zk[5]
484     h_sp2 = -7.65887664*10**(5)#-3.41748920*10**(3)#72.5945 #Nominal optimal
        wellhead pressure
485     tau12 = 264
486     tauC2 = 6000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
487     tauI2 = 500#ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC
        rules for integration processes.
488     Kp2 = 1/(318.34137393 *tauC2)#Proportional gain 0.2417497
489     Ki2 = Kp2/tauI2 #Integral gain
490     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
491     #Calculate new controller output
492     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[-1] ))))
493     #Update the controller output for z_ov(oil valve separator)
494     uk[1] = u2
495     #Store all errors, to be used for previous errors
496     error_store2.append(error2)
497     u_2plot.append(u2)
498 
499     #Annulus P6 and DischargeP GLC6
500     #PI controller, tuned with SIMC rules
501     h3 = -8339.24832552*xf[29] + 4481.00108404*zk[5]#10.22381382*xf[29] +
        65.47016409*zk[5]
502     h_sp3 = -8.73421692*10**(5)#8.26624307*10**(3)#72.5945 #Nominal optimal
        wellhead pressure
503     tau13 = 163
504     tauC3 = 6000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
505     tauI3 = 500#ca.fmin(tau13, 4*tauC3) #Integral time, corresponding to SIMC
        rules for integration processes.
506     Kp3 = 1/(782.41556659*tauC3)#Proportional gain 5.99665696
507     Ki3 = Kp3/tauI3 #Integral gain
508     error3 = (h_sp3 - h3) #Difference between setpoint and measured   value
509     #Calculate new controller output
510     u3 = ca.fmax(0, ca.fmin(1, (u_store[k-1][5] + (Kp3*error3 + Ki3*error3 - Kp3*
        error_store3[-1] ))))
511     #Update the controller output for z_ov(oil valve separator)
512     uk[5] = u3
513     #Store all errors, to be used for previous errors
514     error_store3.append(error3)
515     u_5plot.append(u3)
516 
517 
518     #Annulus P6 and DischargeP GLC2 and discharge pressure 6
519     #PI controller, tuned with SIMC rules
520     h2 = -234922.1903769*xf[29] + 476847.10061302*zk[5]  - 181679.62737955*zk[23]#
        1290.90504975*xf[29] + 16765.46161926*zk[5] + 17129.3708444*zk[23]
521     h_sp2 = -1.40687276*10**(7)#4.26395815*10**(6)#72.5945 #Nominal optimal
        wellhead pressure
522     tau12 = 274#744
```

```
523     tauC2 = 4000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
524     tauI2 = 600#ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC
        rules for integration processes.
525     Kp2 = 1/(9792.10072892*tauC2)#Proportional gain 7.27672269
526     Ki2 = Kp2/tauI2 #Integral gain
527     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
528     #Calculate new controller output
529     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[-1] ))))
530     #Update the controller output for z_ov(oil valve separator)
531     uk[1] = u2
532     #Store all errors, to be used for previous errors
533     error_store2.append(error2)
534     u_2plot.append(u2)
535
536     #Annulus P6 and DischargeP GLC6
537     #PI controller, tuned with SIMC rules
538     h3 = -549333.90812753*xf[29] + 1128132.39347296*zk[5]  - 431189.32210637*zk
        [23]#-1676.90699719*xf[29] - 21807.04221199*zk[5]  - 22311.54379972*zk[23]
539     h_sp3 = -3.24457153*10**(7)#-5.55012462*10**(6)#72.5945 #Nominal optimal
        wellhead pressure
540     tau13 = 133#1454
541     tauC3 = 2000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
542     tauI3 = 600#ca.fmin(tau13, 4*tauC3) #Integral time, corresponding to SIMC
        rules for integration processes.
543     Kp3 = 1/(119828.95678892*tauC3)#Proportional gain 6.35087093
544     Ki3 = Kp3/tauI3 #Integral gain
545     error3 = (h_sp3 - h3) #Difference between setpoint and measured   value
546     #Calculate new controller output
547     u3 = ca.fmax(0, ca.fmin(1, (u_store[k-1][5] + (Kp3*error3 + Ki3*error3 - Kp3*
        error_store3[-1] ))))
548     #Update the controller output for z_ov(oil valve separator)
549     uk[5] = u3
550     #Store all errors, to be used for previous errors
551     error_store3.append(error3)
552     u_5plot.append(u3)
553
554
555     #Annulus P6 and DischargeP GLC2 and discharge pressure 6, annulus pressure 2
556     #PI controller, tuned with SIMC rules
557     h2 = -805763.43006082*xf[29] + 131567.44007523*zk[5] +  291161.30688082*zk[23]
         +  427066.88982393*zk[1]#61119.14498099*xf[29] + 147027.68286197*zk[5] -
        58183.45153308*zk[23] + 201082.69102819*zk[1]
558     h_sp2 = -3.15021186*10**(7)#3.70454342*10**(7)#72.5945 #Nominal optimal
        wellhead pressure
559     tau12 = 169#52#52#120#28 #28
560     tauC2 = 2000#3000#300#Controller time, can be changed up or down depending on
        needs for fast control or smooth control
561     theta2 = 0
562     tauI2 = ca.fmin(tau12, 4*(tauC2 + theta2)) #Integral time, corresponding to
        SIMC rules for integration processes.
563     Kp2 =  1/((75093.770713) *(tauC2 + theta2))#Proportional gain 17809.50303836
564     Ki2 = Kp2/tauI2 #Integral gain
565     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
566     #Calculate new controller output
567     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Ki2*error2 + Kp2*error2 -  Kp2*
        error_store2[-1] )))) #Kp2*error2 - Kp2*error_store2[-1]
568     #Update the controller output for z_ov(oil valve separator)
569     uk[1] = u2
570     #Store all errors, to be used for previous errors
571     error_store2.append(error2)
572     u_2plot.append(u2)
```

```
573
574
575     #Annulus P6 and DischargeP GLC6
576     #PI controller, tuned with SIMC rules
577     h3 = -691151.33978905*xf[29] + 1042352.54566528*zk[5]  - 313718.6801493*zk[23]
         +  106098.72807374*zk[1]#62702.16245358*xf[29] + 118363.56280388*zk[5]  -
        103353.02902385*zk[23] + 216378.02726489*zk[1]
578     h_sp3 = -3.67767947*10**(7)#2.97248713*10**(7)#72.5945 #Nominal optimal
        wellhead pressure
579     tau13 = 137#120 #120
580     tauC3 = 1500#6000#300#Controller time, can be changed up or down depending on
        needs for fast control or smooth control
581     tauI3 = ca.fmin(tau13, 4*tauC3) #Integral time, corresponding to SIMC rules
        for integration processes.
582     Kp3 = 1/(120198.22672229*tauC3)#Proportional gain 5746.51735918
583     Ki3 = Kp3/tauI3 #Integral gain
584     error3 = (h_sp3 - h3) #Difference between setpoint and measured   value
585     #Calculate new controller output
586     u3 = ca.fmax(0, ca.fmin(1, (u_store[k-1][5] + (Kp3*error3 + Ki3*error3 - Kp3*
        error_store3[-1] ))))
587     #Update the controller output for z_ov(oil valve separator)
588     uk[5] = u3
589     #Store all errors, to be used for previous errors
590     error_store3.append(error3)
591     u_5plot.append(u3)
592
593
594     # discharge pressure, annulus pressure 2 GLC2
595     #PI controller, tuned with SIMC rules
596     h2 = -2704.12302706*xf[29] +  327.18670482*zk[1]#10.56433836*xf[29] +
        64.592046*zk[1]
597     h_sp2 =  -3.97329886*10**(5)#8.24045229*10**(3)#72.5945 #Nominal optimal
        wellhead pressure
598     tau12 = 226#48 #28
599     tauC2 = 226#2000#300#Controller time, can be changed up or down depending on
        needs for fast control or smooth control
600     tauI2 = 100#ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC
        rules for integration processes.
601     Kp2 = 1/(152.50011205*tauC2)#Proportional gain 5.88649612
602     Ki2 = Kp2/tauI2 #Integral gain
603     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
604     #Calculate new controller output
605     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[-1] ))))
606     #Update the controller output for z_ov(oil valve separator)
607     uk[1] = u2
608     #Store all errors, to be used for previous errors
609     error_store2.append(error2)
610     u_2plot.append(u2)
611
612     #Annulus P6 and DischargeP GLC6
613     #PI controller, tuned with SIMC rules
614     h3 =  -2675.80879344*xf[29] + 68.22602697*zk[1]#-4.68023771*xf[29] -
        27.54634754*zk[1]
615     h_sp3 = -4.19115433*10**(5)#77569*10**(3)#-3.54212505*10**(3)#72.5945 #Nominal
         optimal wellhead pressure
616     tau13 = 258#201 #120
617     tauC3 = 258#2000#300#Controller time, can be changed up or down depending on
        needs for fast control or smooth control
618     tauI3 = ca.fmin(tau13, 4*tauC3) #Integral time, corresponding to SIMC rules
        for integration processes.
619     Kp3 = 1/(124.23138298 *tauC3)#Proportional gain 0.26182904
620     Ki3 = Kp3/tauI3 #Integral gain
621     error3 = (h_sp3 - h3) #Difference between setpoint and measured   value
```

```
622     #Calculate new controller output
623     u3 = ca.fmax(0, ca.fmin(1, (u_store[k-1][5] + (Kp3*error3 + Ki3*error3 - Kp3*
        error_store3[-1] ))))
624     #Update the controller output for z_ov(oil valve separator)
625     uk[5] = u3
626     #Store all errors, to be used for previous errors
627     error_store3.append(error3)
628     u_5plot.append(u3)
629
630
631     # discharge pressure, annulus pressure 2 GLC2
632     #PI controller, tuned with SIMC rules
633     h2 = -41674.87829334*xf[29] - 52288.11647798*zk[1] + 23376.24685595*zk[19]#
        -2855.6340659*xf[29] - 19423.56983834*zk[1] - 16311.40372815*zk[19]
634     h_sp2 = -8.73664507*10**(6)#-4.66529004*10**(6)#72.5945 #Nominal optimal
        wellhead pressure
635     tau12 = 10#1553#1553#78#1540#28
636     tauC2 = 1553#5000#78#300#Controller time, can be changed up or down depending
        on needs for fast control or smooth control
637     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
638     Kp2 = 0.01*1/(20.70448516  *tauC2)#Proportional gain 3.90348885
639     Ki2 = Kp2/tauI2 #Integral gain
640     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
641     #Calculate new controller output
642     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[-1] ))))
643     #Update the controller output for z_ov(oil valve separator)
644     uk[1] = u2
645     #Store all errors, to be used for previous errors
646     error_store2.append(error2)
647     u_2plot.append(u2)
648
649     #Annulus P6 and DischargeP GLC6
650     #PI controller, tuned with SIMC rules
651     h3 =  -403800.65877812*xf[29] - 541499.5586333*zk[1]  + 240611.02868619*zk[19]
        #-6405.52726445*xf[29] - 43548.86813336*zk[1]  - 36426.92700546*zk[19]
652     h_sp3 =  -8.62554440*10**(7)#-1.04405564*10**(7)#72.5945 #Nominal optimal
        wellhead pressure
653     tau13 = 251#78 #120
654     tauC3 = 5000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
655     tauI3 = ca.fmin(tau13, 4*tauC3) #Integral time, corresponding to SIMC rules
        for integration processes.
656     Kp3 = 100*1/(20122.01897968  *tauC3)#Proportional gain 366.28016462
657     Ki3 = Kp3/tauI3 #Integral gain
658     error3 = (h_sp3 - h3) #Difference between setpoint and measured   value
659     #Calculate new controller output
660     u3 = ca.fmax(0, ca.fmin(1, (u_store[k-1][5] + (Kp3*error3 + Ki3*error3 - Kp3*
        error_store3[-1] ))))
661     #Update the controller output for z_ov(oil valve separator)
662     uk[5] = u3
663     #Store all errors, to be used for previous errors
664     error_store3.append(error3)
665     u_5plot.append(u3)
666
667
668     # discharge pressure, annulus pressure 2 GLC2
669     #PI controller, tuned with SIMC rules
670     h2 = -686015.79742427*xf[29] - 613181.02538563*zk[11] - 334371.87860306*zk[23]
         + 825709.36229966*zk[1]#-3716.69625337*xf[29] - 4798.98978458*zk[11] -
        15641.39990574*zk[23] - 20136.44541368*zk[1]
671     h_sp2 = -1.20961591*10**(8)#-5.17761673*10**(6)#72.5945 #Nominal optimal
        wellhead pressure
```

```
672     tau12 = 120#152#700#500#28
673     tauC2 = 120#1000#300#Controller time, can be changed up or down depending on
        needs for fast control or smooth control
674     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
675     Kp2 = 0.1*1/(107137.28766212 *tauC2)#Proportional gain
676     Ki2 = 10*Kp2/tauI2 #Integral gain
677     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
678     #Calculate new controller output
679     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[-1] ))))
680     #Update the controller output for z_ov(oil valve separator)
681     uk[1] = u2
682     #Store all errors, to be used for previous errors
683     error_store2.append(error2)
684     u_2plot.append(u2)
685
686     #Annulus P6 and DischargeP GLC6
687     #PI controller, tuned with SIMC rules
688     h3 =  -659260.97418188*xf[29] - 243106.42626252*zk[11]  + 98775.47820129*zk
        [23] - 193401.98141469*zk[1]#10994.18774889*xf[29] + 96974.47619766*zk[11]  -
        49965.87429334*zk[23] - 29143.59981315*zk[1]
689     h_sp3 =  -1.30643789*10**(8)#-2.54303615*10**(5)#72.5945 #Nominal optimal
        wellhead pressure
690     tau13 = 221#500#502 #120
691     tauC3 = 221#300#Controller time, can be changed up or down depending on needs
        for fast control or smooth control
692     tauI3 = ca.fmin(tau13, 4*tauC3) #Integral time, corresponding to SIMC rules
        for integration processes.
693     Kp3 = 0.1*1/(30081.59183107 *tauC3)#Proportional gain
694     Ki3 = 10*Kp3/tauI3 #Integral gain
695     error3 = (h_sp3 - h3) #Difference between setpoint and measured   value
696     #Calculate new controller output
697     u3 = ca.fmax(0, ca.fmin(1, (u_store[k-1][5] + (Kp3*error3 + Ki3*error3 - Kp3*
        error_store3[-1] ))))
698     #Update the controller output for z_ov(oil valve separator)
699     uk[5] = u3
700     #Store all errors, to be used for previous errors
701     error_store3.append(error3)
702     u_5plot.append(u3)
703
704
705     ###############Implementation Exact local method positive
        ##########################
706     #BHP&PDIsch3
707     #PI controller, tuned with SIMC rules
708     h2 = -539914.34688897*zk[19] - 2784.25256079*xf[29]#-446413.25092053*zk[19] +
        19345.66174978*xf[29]
709     h_sp2 = -7.45362992*10**(7)#-58181520.23667923#72.5945 #Nominal optimal
        wellhead pressure
710     tau12 = 431#645
711     tauC2 = 2000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
712     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
713     Kp2 = 1/(7359.86887271 *tauC2)#Proportional gain 3664.2726547185534
714     Ki2 = Kp2/tauI2 #Integral gain
715     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
716     #Calculate new controller output
717     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
718     #Update the controller output for z_ov(oil valve separator)
719     uk[1] = u2
720     #Store all errors, to be used for previous errors
```

```
721     error_store2.append(error2)
722     u_2plot.append(u2)
723
724
725
726     #BHP&Prod oil w2
727     #PI controller, tuned with SIMC rules
728     h2 = -9.20434088*10**(2)*zk[19] + 0.491188339*zk[55]#-45.41318981*zk[19] -
        0.08392059*zk[55]
729     h_sp2 = -1.26305988*10**(5)#-6233.1416364#72.5945 #Nominal optimal wellhead
        pressure
730     tau12 = 434#624
731     tauC2 = 2000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
732     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
733     Kp2 = 1/(12.33380599*tauC2)#Proportional gain 0.42254083285255406
734     Ki2 = Kp2/tauI2 #Integral gain
735     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
736     #Calculate new controller output
737     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
738     #Update the controller output for z_ov(oil valve separator)
739     uk[1] = u2
740     #Store all errors, to be used for previous errors
741     error_store2.append(error2)
742     u_2plot.append(u2)
743
744
745
746     #BHP&Prod gas w2
747     #PI controller, tuned with SIMC rules
748     h2 = -42892.00474827*zk[19] + 22740.06185649*zk[49]#-2193.44492237*zk[19] -
        3680.37324392*zk[49]
749     h_sp2 = -5.83284406*10**(6)#-309629.99426986#72.5945 #Nominal optimal wellhead
         pressure
750     tau12 = 404#774
751     tauC2 = 2000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
752     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
753     Kp2 = 1/(713.05330544*tauC2)#Proportional gain 8.377426299741053
754     Ki2 = Kp2/tauI2 #Integral gain
755     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
756     #Calculate new controller output
757     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
758     #Update the controller output for z_ov(oil valve separator)
759     uk[1] = u2
760     #Store all errors, to be used for previous errors
761     error_store2.append(error2)
762     u_2plot.append(u2)
763
764
765
766     #BHP&tot gaslift
767     #PI controller, tuned with SIMC rules
768     h2 = -293089.71870215*zk[19] + 21259.28465932*zk[107]#-19877.79126375*zk[19] -
         9122.48640609*zk[107]
769     h_sp2 = -4.01290192*10**(7)#-2767317.74641531#72.5945 #Nominal optimal
        wellhead pressure
770     tau12 = 432#634
771     tauC2 = 1000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
```

```
772     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
773     Kp2 = 1/(3975.79288856 *tauC2)#Proportional gain 173.0044692063084
774     Ki2 = Kp2/tauI2 #Integral gain
775     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
776     #Calculate new controller output
777     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
778     #Update the controller output for z_ov(oil valve separator)
779     uk[1] = u2
780     #Store all errors, to be used for previous errors
781     error_store2.append(error2)
782     u_2plot.append(u2)
783
784
785
786     #Prod oil w2&Tot gaslift
787     #PI controller, tuned with SIMC rules
788     h2 = 286.77917524*zk[55] + 15.07388955*zk[107]#257.76699191*zk[55] -
        81.78981924*zk[107]
789     h_sp2 = 3.63226099*10**(3)#2852.31828537#72.5945 #Nominal optimal wellhead
        pressure
790     tau12 = 110#211
791     tauC2 = 500#300#Controller time, can be changed up or down depending on needs
        for fast control or smooth control
792     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
793     Kp2 = 1/( 10.69748668 *tauC2)#Proportional gain 4.721775281990528
794     Ki2 = Kp2/tauI2 #Integral gain
795     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
796     #Calculate new controller output
797     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
798     #Update the controller output for z_ov(oil valve separator)
799     uk[1] = u2
800     #Store all errors, to be used for previous errors
801     error_store2.append(error2)
802     u_2plot.append(u2)
803
804
805     #Prod gas w2&Tot gaslift
806     #PI controller, tuned with SIMC rules
807     h2 = 2900.46415236*zk[49] - 388.45472794*zk[107]#2804.15788752*zk[49] -
        750.21433487*zk[107]
808     h_sp2 =  5.11373807*10**(3)#3322.98685992#72.5945 #Nominal optimal wellhead
        pressure
809     tau12 = 202#337
810     tauC2 = 500#300#Controller time, can be changed up or down depending on needs
        for fast control or smooth control
811     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
812     Kp2 = 1/(23.21338918*tauC2)#Proportional gain 12.736480716320484
813     Ki2 = Kp2/tauI2 #Integral gain
814     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
815     #Calculate new controller output
816     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
817     #Update the controller output for z_ov(oil valve separator)
818     uk[1] = u2
819     #Store all errors, to be used for previous errors
820     error_store2.append(error2)
821     u_2plot.append(u2)
822
823
```

```
824
825
826
827     ###############Implementation Exact local method negative
        ##########################
828     #BHP&Pwh2
829     #PI controller, tuned with SIMC rules
830     h2 = -5.34574111*10**(5)*zk[19] - 1.41642159*zk[7]#-483783.15402796*zk[19] -
        9043.21957475*zk[1]
831     h_sp2 = -7.33602582*10**(7)#-67308259.10694613#72.5945 #Nominal optimal
        wellhead pressure
832     tau12 = 434#645
833     tauC2 = 1000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
834     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
835     Kp2 = 1/(7160.60340766*tauC2)#Proportional gain 3664.2726547185534
836     Ki2 = Kp2/tauI2 #Integral gain
837     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
838     #Calculate new controller output
839     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
840     #Update the controller output for z_ov(oil valve separator)
841     uk[1] = u2
842     #Store all errors, to be used for previous errors
843     error_store2.append(error2)
844     u_2plot.append(u2)
845
846
847
848     #BHP&Prod oil w2
849     #PI controller, tuned with SIMC rules
850     h2 = -4.32905553*10**(5)*zk[19] + 2.30539895*10**(2)*zk[55]#
        -6.26485343*10**(4)*zk[19] + 26.0408009*zk[55]
851     h_sp2 = -5.94051978*10**(7)#-8596997.59926734#72.5945 #Nominal optimal
        wellhead pressure
852     tau12 = 434#624
853     tauC2 = 1000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
854     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
855     Kp2 = 1/(5800.92483758*tauC2)#Proportional gain 0.42254083285255406
856     Ki2 = Kp2/tauI2 #Integral gain
857     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
858     #Calculate new controller output
859     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
860     #Update the controller output for z_ov(oil valve separator)
861     uk[1] = u2
862     #Store all errors, to be used for previous errors
863     error_store2.append(error2)
864     u_2plot.append(u2)
865
866
867
868     #BHP&Prod gas w2
869     #PI controller, tuned with SIMC rules
870     h2 = -466615.86146777*zk[19] + 17771.12021649*zk[49]#-62520.82233189*zk[19] +
        613.91453191*zk[49]
871     h_sp2 = -6.39925358*10**(7)#-8578357.35508944#72.5945 #Nominal optimal
        wellhead pressure
872     tau12 = 432#774
873     tauC2 = 1000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
```

```
874     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
875     Kp2 = 1/(6349.30100954*tauC2)#Proportional gain 8.377426299741053
876     Ki2 = Kp2/tauI2 #Integral gain
877     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
878     #Calculate new controller output
879     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
880     #Update the controller output for z_ov(oil valve separator)
881     uk[1] = u2
882     #Store all errors, to be used for previous errors
883     error_store2.append(error2)
884     u_2plot.append(u2)
885
886
887
888     #BHP&tot gaslift
889     #PI controller, tuned with SIMC rules
890     h2 = -526464.96151215 *zk[19] + 5189.13501704*zk[107]#-125435.8109666*zk[19] -
         12587.91389531*zk[107]
891     h_sp2 = -7.22248665*10**(7)#-17268144.34149321#72.5945 #Nominal optimal
        wellhead pressure
892     tau12 = 434#634
893     tauC2 = 1000#300#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
894     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
895     Kp2 = 1/(7059.68830538 *tauC2)#Proportional gain #173.0044692063084*20
896     Ki2 = Kp2/tauI2 #Integral gain
897     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
898     #Calculate new controller output
899     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
900     #Update the controller output for z_ov(oil valve separator)
901     uk[1] = u2
902     #Store all errors, to be used for previous errors
903     error_store2.append(error2)
904     u_2plot.append(u2)
905
906
907
908     #Prod oil w2&Tot gaslift
909     #PI controller, tuned with SIMC rules
910     h2 = 284.98125313*zk[55] + 26.83981752*zk[107]#259.40899698*zk[55] +
        122.2241566*zk[107]
911     h_sp2 = 3.66080317*10**(3)#3755.40804699#72.5945 #Nominal optimal wellhead
        pressure
912     tau12 = 110#211
913     tauC2 = 400#300#Controller time, can be changed up or down depending on needs
        for fast control or smooth control
914     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
915     Kp2 = 1/(10.69984226 *tauC2)#Proportional gain 4.721775281990528
916     Ki2 = Kp2/tauI2 #Integral gain
917     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
918     #Calculate new controller output
919     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
920     #Update the controller output for z_ov(oil valve separator)
921     uk[1] = u2
922     #Store all errors, to be used for previous errors
923     error_store2.append(error2)
924     u_2plot.append(u2)
925
```

```
926
927     #Prod gas w2&Tot gaslift
928     #PI controller, tuned with SIMC rules
929     h2 = 3016.84474179*zk[49] - 270.09782426*zk[107]#3051.23091538*zk[49] +
        57.97844937*zk[107]
930     h_sp2 = 5.89843246*10**(3)#7398.40393725#72.5945 #Nominal optimal wellhead
        pressure
931     tau12 = 200#337
932     tauC2 = 500#300#Controller time, can be changed up or down depending on needs
        for fast control or smooth control
933     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
934     Kp2 = 1/(24.81747272 *tauC2)#Proportional gain 12.736480716320484
935     Ki2 = Kp2/tauI2 #Integral gain
936     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
937     #Calculate new controller output
938     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
939     #Update the controller output for z_ov(oil valve separator)
940     uk[1] = u2
941     #Store all errors, to be used for previous errors
942     error_store2.append(error2)
943     u_2plot.append(u2)
944
945     #
        ##########################################################################################
946
947
948
949
950     ######################Single controlled variable SOC
        ###################################
951     #Wellhead pressure case 7#(POSITIVE GAIN WITH Prod gas constraint)
952     #PI controller, tuned with SIMC rules
953     h2 = z_store[k][7]
954     h_sp2 = 80.6189#72.5865#72.5945 #Nominal optimal wellhead pressure
955     tau12 = 262
956     tauC2 = 1500#400#Controller time, can be changed up or down depending on needs
         for fast control or smooth control
957     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
958     Kp2 = 1/(0.012908143129770892*tauC2)#Proportional gain
959     Ki2 = Kp2/tauI2 #Integral gain
960     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
961     #Calculate new controller output
962     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
963     #Update the controller output for z_ov(oil valve separator)
964     uk[1] = u2
965     #Store all errors, to be used for previous errors
966     error_store2.append(error2)
967     u_2plot.append(u2)
968
969     hD = u_store[k][1]
970     h_spD = 0.64172 #
971     tauCD = 1200#1600#2575 #Controller time, can be changed up or down depending
        on needs for fast control or smooth control
972     tau1D = 500#65
973     thetaD = 0
974     #Opening = 0.291193 optimal nominal
975     tauID = ca.fmin(tau1D, 4*(tauCD + thetaD)) #Integral time, corresponding to
        SIMC rules for integration processes.
```

```
976     KpD = (1/(0.006462045398132644 ))*(1/(tauCD + thetaD)) #Proportional gain
        #0.0007929643656716369
977     KiD = KpD/tauID #Integral gain
978     errorD = (h_spD - hD) #Difference between setpoint and measured   value
979     #Calculate new controller output
980     uD= ca.fmax(0, ca.fmin(1, (u_store[k-1][2] - (KpD*errorD + KiD*errorD - KpD*
        error_store3[-1] ))))
981     #Update the controller output for z_ov(oil valve separator)
982     uk[2] = uD
983     #Store all errors, to be used for previous errors
984     error_store3.append(errorD)
985     u_Dplot.append(uD)
986
987
988     #Bottomhole pressure#(POSITIVE GAIN WITH Prod gas constraint)
989     #PI controller, tuned with SIMC rules
990     h2 = z_store[k][19]
991     h_sp2 = 137.231#123.725 #Nominal optimal bottomhole pressure
992     tau12 = 690
993     tauC2 = 1900#1300 #Controller time, can be changed up or down depending on
        needs for fast control or smooth control
994     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
995     Kp2 = 1/(0.01339127782608702*tauC2)#Proportional gain
996     Ki2 = Kp2/tauI2 #Integral gain
997     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
998     #Calculate new controller output
999     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] - (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
1000    #Update the controller output for z_ov(oil valve separator)
1001    uk[1] = u2
1002    #Store all errors, to be used for previous errors
1003    error_store2.append(error2)
1004    u_2plot.append(u2)
1005
1006
1007    #Annulus pressure#(POSITIVE GAIN WITH Prod gas constraint)
1008    #PI controller, tuned with SIMC rules
1009    h2 = z_store[k][1]
1010    h_sp2 = 101.536#87.8149 #Nominal optimal annulus pressure
1011    tau12 = 64
1012    tauC2 = 1300#64#1300 #Controller time, can be changed up or down depending on
        needs for fast control or smooth control
1013    tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
1014    Kp2 = 1/(0.13092841875000039*tauC2)#Proportional gain
1015    Ki2 = Kp2/tauI2 #Integral gain
1016    error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
1017    #Calculate new controller output
1018    u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
1019    #Update the controller output for z_ov(oil valve separator)
1020    uk[1] = u2
1021    #Store all errors, to be used for previous errors
1022    error_store2.append(error2)
1023    u_2plot.append(u2)
1024
1025    #Separator pressure#(POSITIVE GAIN WITH Prod gas constraint)
1026    #PI controller, tuned with SIMC rules
1027    h2 = x_store[k][20]
1028    h_sp2 = 21.8954 #Nominal optimal Sep pressure
1029    tau12 = 550
1030    theta2 = 289
```

```
1031     tauC2 = 2000#1000#3000#1500#1300 #Controller time, can be changed up or down
    depending on needs for fast control or smooth control
1032     tauI2 = ca.fmin(tau12, 4*(tauC2 + theta2)) #Integral time, corresponding to
    SIMC rules for integration processes.
1033     Kp2 = (1/(0.00022573436363638227))*(1/(tauC2 + theta2))#Proportional gain
1034     Ki2 = Kp2/tauI2 #Integral gain
1035     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
1036     #Calculate new controller output
1037     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] - (Kp2*error2 + Ki2*error2 - Kp2*
    error_store2[k-1] ))))
1038     #Update the controller output for z_ov(oil valve separator)
1039     uk[1] = u2
1040     #Store all errors, to be used for previous errors
1041     error_store2.append(error2)
1042     u_2plot.append(u2)
1043
1044     hD = u_store[k][1]
1045     h_spD = 0.537969 #
1046     tauCD = 3000#293#2575 #Controller time, can be changed up or down depending on
     needs for fast control or smooth control
1047     tau1D = 500#65
1048     thetaD = 0
1049     #Opening = 0.291193 optimal nominal
1050     tauID = ca.fmin(tau1D, 4*(tauCD + thetaD)) #Integral time, corresponding to
    SIMC rules for integration processes.
1051     KpD = (1/(0.006462045398132644 ))*(1/(tauCD + thetaD)) #Proportional gain
    #0.0007929643656716369
1052     KiD = KpD/tauID #Integral gain
1053     errorD = (h_spD - hD) #Difference between setpoint and measured   value
1054     #Calculate new controller output
1055     uD= ca.fmax(0, ca.fmin(1, (u_store[k-1][2] - (KpD*errorD + KiD*errorD - KpD*
    error_store3[-1] ))))
1056     #Update the controller output for z_ov(oil valve separator)
1057     uk[2] = uD
1058     #Store all errors, to be used for previous errors
1059     error_store3.append(errorD)
1060     u_Dplot.append(uD)
1061
1062
1063     hD = u_store[k][1]
1064     h_spD = 0.537969 #
1065     tauCD = 1000#1600#2575 #Controller time, can be changed up or down depending
    on needs for fast control or smooth control
1066     tau1D = 500#65
1067     thetaD = 0
1068     #Opening = 0.291193 optimal nominal
1069     tauID = ca.fmin(tau1D, 4*(tauCD + thetaD)) #Integral time, corresponding to
    SIMC rules for integration processes.
1070     KpD = (1/(0.006462045398132644 ))*(1/(tauCD + thetaD)) #Proportional gain
    #0.0007929643656716369
1071     KiD = KpD/tauID #Integral gain
1072     errorD = (h_spD - hD) #Difference between setpoint and measured   value
1073     #Calculate new controller output
1074     uD= ca.fmax(0, ca.fmin(1, (u_store[k-1][2] + (KpD*errorD + KiD*errorD - KpD*
    error_store3[-1] ))))
1075     #Update the controller output for z_ov(oil valve separator)
1076     uk[2] = uD
1077     #Store all errors, to be used for previous errors
1078     error_store3.append(errorD)
1079     u_Dplot.append(uD)
1080
1081     #Manifold pressure#
1082     #PI controller, tuned with SIMC rules
1083     h2 = z_store[k][74]
```

```
1084     h_sp2 = 78.683 #Nominal optimal man pressure
1085     tau12 = 293
1086     tauC2 = 2500 #Controller time, can be changed up or down depending on needs
         for fast control or smooth control
1087     tauI2 = ca.fmin(tau12, 4*(tauC2)) #Integral time, corresponding to SIMC rules
         for integration processes.
1088     Kp2 = (1/(0.004487439590443815))*(1/(tauC2))#Proportional gain
1089     Ki2 = Kp2/tauI2 #Integral gain
1090     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
1091     #Calculate new controller output
1092     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
         error_store2[k-1] ))))
1093     #Update the controller output for z_ov(oil valve separator)
1094     uk[1] = u2
1095     #Store all errors, to be used for previous errors
1096     error_store2.append(error2)
1097     u_2plot.append(u2)
1098
1099     hD = u_store[k][1]
1100     h_spD = 0.537969 #
1101     tauCD = 500#293#2575 #Controller time, can be changed up or down depending on
         needs for fast control or smooth control
1102     tau1D = 500#65
1103     thetaD = 0
1104     #Opening = 0.291193 optimal nominal
1105     tauID = ca.fmin(tau1D, 4*(tauCD + thetaD)) #Integral time, corresponding to
         SIMC rules for integration processes.
1106     KpD = (1/(0.006462045398132644 ))*(1/(tauCD + thetaD)) #Proportional gain
         #0.0007929643656716369
1107     KiD = KpD/tauID #Integral gain
1108     errorD = (h_spD - hD) #Difference between setpoint and measured   value
1109     #Calculate new controller output
1110     uD= ca.fmax(0, ca.fmin(1, (u_store[k-1][2] - (KpD*errorD + KiD*errorD - KpD*
         error_store3[-1] ))))
1111     #Update the controller output for z_ov(oil valve separator)
1112     uk[2] = uD
1113     #Store all errors, to be used for previous errors
1114     error_store3.append(errorD)
1115     u_Dplot.append(uD)
1116
1117
1118     #Discharge comp 3 pressure#Negative gain from prod gas.
1119     #PI controller, tuned with SIMC rules
1120     h2 = x_store[k][29]
1121     h_sp2 = 159.22 #Nominal optimal man pressure
1122     tau12 = 225
1123     tauC2 = 800#225#1300 #Controller time, can be changed up or down depending on
         needs for fast control or smooth control
1124     tauI2 = ca.fmin(tau12, 4*(tauC2)) #Integral time, corresponding to SIMC rules
         for integration processes.
1125     Kp2 = (1/(0.06490573644444389))*(1/(tauC2))#Proportional gain
1126     Ki2 = Kp2/tauI2 #Integral gain
1127     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
1128     #Calculate new controller output
1129     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] - (Kp2*error2 + Ki2*error2 - Kp2*
         error_store2[k-1] ))))
1130     #Update the controller output for z_ov(oil valve separator)
1131     uk[1] = u2
1132     #Store all errors, to be used for previous errors
1133     error_store2.append(error2)
1134     u_2plot.append(u2)
1135
1136     hD = u_store[k][1]
1137     h_spD = 0.537969 #
```

```
    tauCD = 2000#1600#2575 #Controller time, can be changed up or down depending on needs for fast control or smooth control
    tau1D = 500#65
    thetaD = 0
    #Opening = 0.291193 optimal nominal
    tauID = ca.fmin(tau1D, 4*(tauCD + thetaD)) #Integral time, corresponding to SIMC rules for integration processes.
    KpD = (1/(0.006462045398132644 ))*(1/(tauCD + thetaD)) #Proportional gain #0.0007929643656716369
    KiD = KpD/tauID #Integral gain
    errorD = (h_spD - hD) #Difference between setpoint and measured   value
    #Calculate new controller output
    uD= ca.fmax(0, ca.fmin(1, (u_store[k-1][2] - (KpD*errorD + KiD*errorD - KpD*error_store3[-1] ))))
    #Update the controller output for z_ov(oil valve separator)
    uk[2] = uD
    #Store all errors, to be used for previous errors
    error_store3.append(errorD)
    u_Dplot.append(uD)
```

```
    ##############Case 7 Nullspace Positive#######################################
    #BHP&WHP
    #PI controller, tuned with SIMC rules
    h2 = 0.99819296*zk[19] - 0.06009004*zk[7]#0.95155469*z_store[k][19] + 0.30747954*z_store[k][7]#0.99819296*zk[19] - 0.06009004*zk[7]#
    h_sp2 =  132.1386341894136#155.37146395069#72.5945#132.13866341894136# # Nominal optimal wellhead pressure
    tau12 =  621#668 #621
    tauC2 = 1000#300#Controller time, can be changed up or down depending on needs for fast control or smooth control
    tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules for integration processes.
    Kp2 = (1/(0.00967549 *tauC2))#Proportional gain     #0.00967549 0.007077228592814475
    Ki2 = Kp2/tauI2 #Integral gain
    error2 = (h_sp2 - h2) #Difference between setpoint and measured   value

    #Calculate new controller output
    u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] - (Kp2*error2 + Ki2*error2 - Kp2*error_store2[-1])))) #- 0.1*Kp2*(error2 - error_store2[k-1]))))
    #Update the controller output for z_ov(oil valve separator)
    uk[1] = u2
    #Store all errors, to be used for previous errors
    error_store2.append(error2)
    u_2plot.append(u2)

    #BHP&Annulus P
    #PI controller, tuned with SIMC rules
    h2 = 0.99918968*zk[19] - 0.04024904*zk[1]#0.97178607*z_store[k][19] + 0.23586401*z_store[k][1]
    h_sp2 = 133.03309302#157.3078622915#72.5945 #Nominal optimal wellhead pressure
    tau12 = 591#814
    tauC2 = 1500#3000#Controller time, can be changed up or down depending on needs for fast control or smooth control
    tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules for integration processes.
    Kp2 = 1/(0.01025999 *tauC2)#Proportional gain 0.01025999 #0.005105094594594611
    Ki2 = Kp2/tauI2 #Integral gain
    error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
    #Calculate new controller output
```

```
1189     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] - (Kp2*error2 + Ki2*error2 - Kp2*
         error_store2[k-1] ))))
1190     #Update the controller output for z_ov(oil valve separator)
1191     uk[1] = u2
1192     #Store all errors, to be used for previous errors
1193     error_store2.append(error2)
1194     u_2plot.append(u2)
1195 
1196     #BHP&Manifold P#With negative gain
1197     #PI controller, tuned with SIMC rules
1198     h2 = 0.99797747*zk[19] - 0.06356854*zk[74]#0.93394895*z_store[k][19] +
         0.35740644*z_store[k][74]
1199     h_sp2 = 131.95168963#156.2885592759#72.5945 #Nominal optimal wellhead pressure
1200     tau12 = 620#641
1201     tauC2 = 2000#1500#Controller time, can be changed up or down depending on
         needs for fast control or smooth control
1202     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
         for integration processes.
1203     Kp2 = 1/(0.00942779*tauC2)#Proportional gain #0.008088352730108911
1204     Ki2 = Kp2/tauI2 #Integral gain
1205     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
1206     #Calculate new controller output
1207     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] - (Kp2*error2 + Ki2*error2 - Kp2*
         error_store2[k-1] ))))
1208     #Update the controller output for z_ov(oil valve separator)
1209     uk[1] = u2
1210     #Store all errors, to be used for previous errors
1211     error_store2.append(error2)
1212     u_2plot.append(u2)
1213 
1214     #BHP&Disch comp3
1215     #PI controller, tuned with SIMC rules
1216     h2 = 0.9999867*zk[19] +  0.00515668*xf[29]#0.99906231*z_store[k][19] -
         0.04329552*x_store[k][29]
1217     h_sp2 = 138.05023113#130.2088071692100#72.5945 #Nominal optimal wellhead
         pressure
1218     tau12 = 621#645
1219     tauC2 = 1200#Controller time, can be changed up or down depending on needs for
          fast control or smooth control
1220     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
         for integration processes.
1221     Kp2 = 1/(0.00946077 *tauC2)#Proportional gain 0.008200548527131894
1222     Ki2 = Kp2/tauI2 #Integral gain
1223     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
1224     #Calculate new controller output
1225     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] - (Kp2*error2 + Ki2*error2 - Kp2*
         error_store2[k-1] ))))
1226     #Update the controller output for z_ov(oil valve separator)
1227     uk[1] = u2
1228     #Store all errors, to be used for previous errors
1229     error_store2.append(error2)
1230     u_2plot.append(u2)
1231 
1232     hD = u_store[k][1]
1233     h_spD = 0.419185 #
1234     tauCD = 1600#2575 #Controller time, can be changed up or down depending on
         needs for fast control or smooth control
1235     tau1D = 500#65
1236     thetaD = 0
1237     #Opening = 0.291193 optimal nominal
1238     tauID = ca.fmin(tau1D, 4*(tauCD + thetaD)) #Integral time, corresponding to
         SIMC rules for integration processes.
1239     KpD = (1/(0.006462045398132644 ))*(1/(tauCD + thetaD)) #Proportional gain
         #0.0007929643656716369
```

```
1240     KiD = KpD/tauID #Integral gain
1241     errorD = (h_spD - hD) #Difference between setpoint and measured   value
1242     #Calculate new controller output
1243     uD= ca.fmax(0, ca.fmin(1, (u_store[k-1][2] - (KpD*errorD + KiD*errorD - KpD*
         error_store3[-1] ))))
1244     #Update the controller output for z_ov(oil valve separator)
1245     uk[2] = uD
1246     #Store all errors, to be used for previous errors
1247     error_store3.append(errorD)
1248     u_Dplot.append(uD)
1249 
1250     #############Case 7 Nullspace Negative#######################################
1251     #BHP&WHP
1252     #PI controller, tuned with SIMC rules
1253     h2 =  0.99992464*zk[19] - 0.01227675*zk[7]#0.26764386*z_store[k][19] +
         0.96351791*z_store[k][7]
1254     h_sp2 = 136.23092774#114.40678858615#72.5945 #Nominal optimal wellhead
         pressure
1255     tau12 =  621#85
1256     tauC2 = 1000#300#Controller time, can be changed up or down depending on needs
          for fast control or smooth control
1257     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
         for integration processes.
1258     Kp2 = 1/(0.00941242 *tauC2)#Proportional gain 0.011343201176470408
1259     Ki2 = Kp2/tauI2 #Integral gain
1260     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
1261     #Calculate new controller output
1262     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] - (Kp2*error2 + Ki2*error2 - Kp2*
         error_store2[k-1] ))))
1263     #Update the controller output for z_ov(oil valve separator)
1264     uk[1] = u2
1265     #Store all errors, to be used for previous errors
1266     error_store2.append(error2)
1267     u_2plot.append(u2)
1268 
1269 
1270     #BHP&ANnnulus P
1271     #PI controller, tuned with SIMC rules
1272     h2 = 0.99999641*zk[19] - 0.00267783*zk[1]#0.99982427*z_store[k][19] +
         0.0187467*z_store[k][1]
1273     h_sp2 = 136.9586203#139.1103493275#72.5945 #Nominal optimal wellhead pressure
1274     tau12 =  621#638
1275     tauC2 = 1000#300#Controller time, can be changed up or down depending on needs
          for fast control or smooth control
1276     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
         for integration processes.
1277     Kp2 = 1/(0.0093887 *tauC2)#Proportional gain 0.06698454823529186
1278     Ki2 = Kp2/tauI2 #Integral gain
1279     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
1280     #Calculate new controller output
1281     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] - (Kp2*error2 + Ki2*error2 - Kp2*
         error_store2[k-1] ))))
1282     #Update the controller output for z_ov(oil valve separator)
1283     uk[1] = u2
1284     #Store all errors, to be used for previous errors
1285     error_store2.append(error2)
1286     u_2plot.append(u2)
1287 
1288     #BHP&Sep pressure
1289     #PI controller, tuned with SIMC rules
1290     h2 = 0.99371152*zk[19] +  0.11197063*xf[20]#0.87581411*z_store[k][19] -
         0.48264858*x_store[k][20]
1291     h_sp2 = 138.81967765#109.6210472557125#72.5945 #Nominal optimal wellhead
         pressure
```

```
1292     tau12 =  623#624
1293     tauC2 = 1000#300#Controller time, can be changed up or down depending on needs
          for fast control or smooth control
1294     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
1295     Kp2 = 1/(0.0092726 *tauC2)#Proportional gain 0.06698454823529186
1296     Ki2 = Kp2/tauI2 #Integral gain
1297     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
1298     #Calculate new controller output
1299     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] - (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
1300     #Update the controller output for z_ov(oil valve separator)
1301     uk[1] = u2
1302     #Store all errors, to be used for previous errors
1303     error_store2.append(error2)
1304     u_2plot.append(u2)
1305
1306     #BHP&Man pressure
1307     #PI controller, tuned with SIMC rules
1308     h2 = 0.9999131*zk[19]  - 0.01318273*zk[74]#0.03331483*z_store[k][19] -
        0.99944491*z_store[k][74]
1309     h_sp2 = 136.18182529#-74.067496417#72.5945 #Nominal optimal wellhead pressure
1310     tau12 =  622#294
1311     tauC2 = 2000#300#Controller time, can be changed up or down depending on needs
          for fast control or smooth control
1312     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
1313     Kp2 = 1/(0.00936008 *tauC2)#Proportional gain 0.002987406462584945
1314     Ki2 = Kp2/tauI2 #Integral gain
1315     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
1316     #Calculate new controller output
1317     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] - (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
1318     #Update the controller output for z_ov(oil valve separator)
1319     uk[1] = u2
1320     #Store all errors, to be used for previous errors
1321     error_store2.append(error2)
1322     u_2plot.append(u2)
1323
1324     #BHP&Disch comp 3 pressure
1325     #PI controller, tuned with SIMC rules
1326     h2 = 0.999999889*zk[19] - 4.70335652*10**(-4)*xf[29]#0.99994079*z_store[k][19]
          - 0.01088206*x_store[k][29]
1327     h_sp2 = 137.15610524#135.4512098921#72.5945 #Nominal optimal wellhead pressure
1328     tau12 = 623 #629
1329     tauC2 = 1000#300#Controller time, can be changed up or down depending on needs
          for fast control or smooth control
1330     tauI2 = ca.fmin(tau12, 4*tauC2) #Integral time, corresponding to SIMC rules
        for integration processes.
1331     Kp2 = 1/(0.00934041*tauC2)#Proportional gain 0.009034566454690212
1332     Ki2 = Kp2/tauI2 #Integral gain
1333     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
1334     #Calculate new controller output
1335     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] - (Kp2*error2 + Ki2*error2 - Kp2*
        error_store2[k-1] ))))
1336     #Update the controller output for z_ov(oil valve separator)
1337     uk[1] = u2
1338     #Store all errors, to be used for previous errors
1339     error_store2.append(error2)
1340     u_2plot.append(u2)
1341
1342     hD = u_store[k][1]
1343     h_spD = 0.537969 #
```

```
1344     tauCD = 1600#2575 #Controller time, can be changed up or down depending on
         needs for fast control or smooth control
1345     tau1D = 500#65
1346     thetaD = 0
1347     #Opening = 0.291193 optimal nominal
1348     tauID = ca.fmin(tau1D, 4*(tauCD + thetaD)) #Integral time, corresponding to
         SIMC rules for integration processes.
1349     KpD = (1/(0.006462045398132644 ))*(1/(tauCD + thetaD)) #Proportional gain
         #0.0007929643656716369
1350     KiD = KpD/tauID #Integral gain
1351     errorD = (h_spD - hD) #Difference between setpoint and measured   value
1352     #Calculate new controller output
1353     uD= ca.fmax(0, ca.fmin(1, (u_store[k-1][2] - (KpD*errorD + KiD*errorD - KpD*
         error_store3[-1] ))))
1354     #Update the controller output for z_ov(oil valve separator)
1355     uk[2] = uD
1356     #Store all errors, to be used for previous errors
1357     error_store3.append(errorD)
1358     u_Dplot.append(uD)
1359
1360
1361     #BHP&Sep pressure
1362     #PI controller, tuned with SIMC rules
1363     h2 = 5.55555562*10**(-6)*z_store[k][19] + x_store[k][20]
1364     h_sp2 = 21.8915762#72.5945 #Nominal optimal wellhead pressure
1365     tau12 =  319
1366     theta1S = 189
1367     tauC2 = 4000#300#Controller time, can be changed up or down depending on needs
          for fast control or smooth control
1368     tauI2 = ca.fmin(tau12, 4*(tauC2 + theta1S)) #Integral time, corresponding to
         SIMC rules for integration processes.
1369     Kp2 = 1/(0.00014624952978058362 *(tauC2 + theta1S ))#Proportional gain
1370     Ki2 = Kp2/tauI2 #Integral gain
1371     error2 = (h_sp2 - h2) #Difference between setpoint and measured   value
1372     #Calculate new controller output
1373     u2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][1] + (Kp2*error2 + Ki2*error2 - Kp2*
         error_store2[k-1] ))))
1374     #Update the controller output for z_ov(oil valve separator)
1375     uk[1] = u2
1376     #Store all errors, to be used for previous errors
1377     error_store2.append(error2)
1378     u_2plot.append(u2)
1379
1380     hD = u_store[k][1]
1381     h_spD = 0.537969 #
1382     tauCD = 4000#2575 #Controller time, can be changed up or down depending on
         needs for fast control or smooth control
1383     tau1D = 500#65
1384     thetaD = 0
1385     #Opening = 0.291193 optimal nominal
1386     tauID = ca.fmin(tau1D, 4*(tauCD + thetaD)) #Integral time, corresponding to
         SIMC rules for integration processes.
1387     KpD = (1/(0.006462045398132644 ))*(1/(tauCD + thetaD)) #Proportional gain
         #0.0007929643656716369
1388     KiD = KpD/tauID #Integral gain
1389     errorD = (h_spD - hD) #Difference between setpoint and measured   value
1390     #Calculate new controller output
1391     uD= ca.fmax(0, ca.fmin(1, (u_store[k-1][2] - (KpD*errorD + KiD*errorD - KpD*
         error_store3[-1] ))))
1392     #Update the controller output for z_ov(oil valve separator)
1393     uk[2] = uD
1394     #Store all errors, to be used for previous errors
1395     error_store3.append(errorD)
1396     u_Dplot.append(uD)
```

```
1397
1398
1399
1400    ##################################################################
1401    ##################Surge Control#####################################
1402
1403    Scon = 3.25895
1404    ###############Conmpressor
        1#####################################################
1405    if x_store[k][24] < Scon:#Scon:#3.79601:
1406        #PI controller, tuned with SIMC rules
1407        hC1 = x_store[k][24]
1408        h_spC1  = Scon#72.5945 #Nominal optimal wellhead pressure
1409        tau1C1  = 1
1410        tauCC1  = 10#100#Controller time, can be changed up or down depending on
        needs for fast control or smooth control
1411        tauIC1  = ca.fmin(tau1C1 , 4*tauCC1 ) #Integral time, corresponding to
        SIMC rules for integration processes.
1412        KpC1  = 1/(0.3114571*tauCC1 )#1/(0.012908143129770892*tauC2)#Proportional
        gain
1413        KiC1  = KpC1 /tauIC1  #Integral gain
1414        errorC1  = (h_spC1  - hC1) #Difference between setpoint and measured
        value
1415        #Calculate new controller output
1416        uC1  = ca.fmax(0, ca.fmin(1, (u_store[k-1][17] + (KpC1 *errorC1 + KiC1*
        errorC1 - KpC1*error_storeC1[-1] ))))
1417        #Update the controller output for z_ov(oil valve separator)
1418        uk[17] = uC1
1419        #Store all errors, to be used for previous errors
1420        error_storeC1.append(errorC1)
1421        #u_2plot.append(u6)
1422
1423    if x_store[k][24] > Scon and u_store[k-1][17] >= 0:
1424        #PI controller, tuned with SIMC rules
1425        hC1 = x_store[k][24]
1426        h_spC1  = Scon#72.5945 #Nominal optimal wellhead pressure
1427        tau1C1  = 10
1428        tauCC1  = 10#100#Controller time, can be changed up or down depending on
        needs for fast control or smooth control
1429        tauIC1  = ca.fmin(tau1C1 , 4*tauCC1 ) #Integral time, corresponding to
        SIMC rules for integration processes.
1430        KpC1  = 1/(0.3114571*tauCC1 )#1/(0.012908143129770892*tauC2)#Proportional
        gain
1431        KiC1  = KpC1 /tauIC1  #Integral gain
1432        errorC1  = (h_spC1  - hC1) #Difference between setpoint and measured
        value
1433        #Calculate new controller output
1434        uC1  = ca.fmax(0, ca.fmin(1, (u_store[k-1][17] + (KpC1 *errorC1 + KiC1*
        errorC1 - KpC1*error_storeC11[-1] )))) #+ Ki6*error6 - Kp6*error_store6[-1] ))
        ))
1435        #Update the controller output for z_ov(oil valve separator)
1436        uk[17] = uC1
1437        #Store all errors, to be used for previous errors
1438        error_storeC11.append(errorC1)
1439        #u_2plot.append(u6)
1440    #
        ##########################################################################

1441    ###############Conmpressor
        2#####################################################
1442
1443
1444    if x_store[k][27] < Scon:#3.79601:
1445        #PI controller, tuned with SIMC rules
```

```
1446        hC2 = x_store[k][27]
1447        h_spC2 = Scon#72.5945 #Nominal optimal wellhead pressure
1448        tau1C2 = 10
1449        tauCC2 = 10#200#Controller time, can be changed up or down depending on
    needs for fast control or smooth control
1450        tauIC2 = ca.fmin(tau1C2, 4*tauCC2) #Integral time, corresponding to SIMC
    rules for integration processes.
1451        KpC2 = 1/(2*0.3114571*tauCC2)#1/(0.012908143129770892*tauC2)#Proportional
    gain
1452        KiC2 = KpC2/tauIC2 #Integral gain
1453        errorC2 = (h_spC2 - hC2) #Difference between setpoint and measured   value
1454        #Calculate new controller output
1455        uC2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][18] + (KpC2*errorC2 + KiC2*
    errorC2 - KpC2*error_storeC2[-1] ))))
1456        #Update the controller output for z_ov(oil valve separator)
1457        uk[18] = uC2
1458        #Store all errors, to be used for previous errors
1459        error_storeC2.append(errorC2)
1460        #u_2plot.append(u6)
1461
1462    if x_store[k][27] > Scon and u_store[k-1][18] >= 0:
1463        #PI controller, tuned with SIMC rules
1464        hC2 = x_store[k][27]
1465        h_spC2 = Scon#72.5945 #Nominal optimal wellhead pressure
1466        tau1C2 = 10
1467        tauCC2 = 10#200#Controller time, can be changed up or down depending on
    needs for fast control or smooth control
1468        tauIC2 = ca.fmin(tau1C2, 4*tauCC2) #Integral time, corresponding to SIMC
    rules for integration processes.
1469        KpC2 = 1/(2*0.3114571*tauCC2)#1/(0.012908143129770892*tauC2)#Proportional
    gain
1470        KiC2 = KpC2/tauIC2 #Integral gain
1471        errorC2 = (h_spC2 - hC2) #Difference between setpoint and measured   value
1472        #Calculate new controller output
1473        uC2 = ca.fmax(0, ca.fmin(1, (u_store[k-1][18] + (KpC2*errorC2 + KiC2*
    errorC2 - KpC2*error_storeC22[-1] ))))
1474        #Update the controller output for z_ov(oil valve separator)
1475        uk[18] = uC2
1476        #Store all errors, to be used for previous errors
1477        error_storeC22.append(errorC2)
1478        #u_2plot.append(u6)
1479
1480        #
    ##########################################################################################

1481        ###############Conmpressor
    3#####################################################
1482
1483    if x_store[k][30] < Scon:
1484        #PI controller, tuned with SIMC rules
1485        hC3 = x_store[k][30]
1486        h_spC3 = Scon#72.5945 #Nominal optimal wellhead pressure
1487        tau1C3 = 10
1488        tauCC3 = 10#400#Controller time, can be changed up or down depending on
    needs for fast control or smooth control
1489        tauIC3 = ca.fmin(tau1C3, 4*tauCC3) #Integral time, corresponding to SIMC
    rules for integration processes.
1490        KpC3 = 1/(3*0.3114571*tauCC3)#1/(0.012908143129770892*tauC2)#Proportional
    gain
1491        KiC3 = KpC3/tauIC3 #Integral gain
1492        errorC3 = (h_spC3 - hC3) #Difference between setpoint and measured   value
1493        #Calculate new controller output
1494        uC3 = ca.fmax(0, ca.fmin(1, (u_store[k-1][19] + (KpC3*errorC3 + KiC3*
    errorC3 - KpC3*error_storeC3[-1] ))))
```

```
1495        #Update the controller output for z_ov(oil valve separator)
1496        uk[19] = uC3
1497        #Store all errors, to be used for previous errors
1498        error_storeC3.append(errorC3)
1499        #u_2plot.append(u6)
1500
1501    if x_store[k][30] > Scon and u_store[k-1][19] >= 0:
1502        #PI controller, tuned with SIMC rules
1503        hC3 = x_store[k][30]
1504        h_spC3 = Scon#72.5945 #Nominal optimal wellhead pressure
1505        tau1C3 = 10
1506        tauCC3 = 10#400#Controller time, can be changed up or down depending on
    needs for fast control or smooth control
1507        tauIC3 = ca.fmin(tau1C3, 4*tauCC3) #Integral time, corresponding to SIMC
    rules for integration processes.
1508        KpC3 = 1/(3*0.3114571*tauCC3)#1/(0.012908143129770892*tauC2)#Proportional
    gain
1509        KiC3 = KpC3/tauIC3 #Integral gain
1510        errorC3 = (h_spC3 - hC3) #Difference between setpoint and measured   value
1511        #Calculate new controller output
1512        uC3 = ca.fmax(0, ca.fmin(1, (u_store[k-1][19] + (KpC3*errorC3 + KiC3*
    errorC3 - KpC3*error_storeC33[-1] ))))
1513        #Update the controller output for z_ov(oil valve separator)
1514        uk[19] = uC3
1515        #Store all errors, to be used for previous errors
1516        error_storeC33.append(errorC3)
1517        #u_2plot.append(u6)
1518    #
    ##############################################################################
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531    #Prod gas control Split Range control with Batton strategy
1532    ###################################################################
1533    #if z_store[k][79] > 10.0001:
1534    if Condition == 0:
1535        ###### Test with multiple controllers controlling active constraint
    ###########################################################################
1536        #PI controller, tuned with SIMC rules(integration process) well 4
1537        h4 = z_store[k][79]
1538        h_sp4 = 10.0000 #
1539        tauC4 = 2000#2000#5000#5000#5000#1500#2000#1500#800#572 #Controller time,
    can be changed up or down depending on needs for fast control or smooth
    control
1540        tau14 = 572#2000#1000#572
1541        #theta4 = 272
1542        #Opening = 0.291193 optimal nominal
1543        tauI4 = ca.fmin(tau14, 4*(tauC4 + theta4)) #Integral time, corresponding
    to SIMC rules for integration processes.
1544        Kp4 = (1/(0.000630722569930093))*(1/(tauC4 + theta4)) #Proportional gain
    #0.0007929643656716369
1545        Ki4 = Kp4/tauI4 #Integral gain
1546        error4 = (h_sp4 - h4) #Difference between setpoint and measured   value
1547        #Calculate new controller output
```

```
1548         #tauD = 2
1549         u4 = ca.fmax(0, ca.fmin(1, (u_store[k-1][0] + (Kp4*error4 + Ki4*error4 -
    Kp4*error_store[-1] + Kp4*0*(error4 - 2*error_store[-1] + error_store[-2]) )))
    )
1550         #Update the controller output for z_ov(oil valve separator)
1551         uk[0] = u4
1552         #Store all errors, to be used for previous errors
1553         error_store.append(error4)
1554         u_4plot.append(u4)
1555 
1556         if u4 == 1 or u4 == 0:
1557             Condition = 1
1558     if Condition == 1:
1559         #PI controller, tuned with SIMC rules(integration process) well 4
1560         hA = z_store[k][79]
1561         h_spA = 10.0000 #
1562         tauCA = 2000#3000#5000#800#559#800#Controller time, can be changed up or
    down depending on needs for fast control or smooth control
1563         tau1A = 559
1564         #thetaA = 288
1565         #Opening = 0.291193 optimal nominal #0.0006445851520572363
1566         tauIA = ca.fmin(tau1A, 4*(tauCA + thetaA)) #Integral time, corresponding
    to SIMC rules for integration processes.
1567         KpA = (1/(0.0006445851520572363))*(1/(tauCA + thetaA)) #Proportional gain
    #0.0007929643656716369
1568         KiA = KpA/tauIA #Integral gain
1569         errorA = (h_spA - hA) #Difference between setpoint and measured   value
1570         #Calculate new controller output
1571         uA = ca.fmax(0, ca.fmin(1, (u_store[k-1][4] + (KpA*errorA + KiA*errorA -
    KpA*error_store[-1] + KpA*0*(errorA - 2*error_store[-1] + error_store[-2])))))
    #+
1572         #Update the controller output for z_ov(oil valve separator)
1573         uk[4] = uA
1574         #Store all errors, to be used for previous errors
1575         error_store.append(errorA)
1576         u_Aplot.append(uA)
1577         if uA == 1 or uA == 0:
1578             Condition = 2
1579     if Condition == 2:
1580         #PI controller, tuned with SIMC rules(integration process) well 4
1581         hB = z_store[k][79]
1582         h_spB = 10.0000 #
1583         tauCB = 2000#572 #Controller time, can be changed up or down depending on
    needs for fast control or smooth control
1584         tau1B = 572
1585         #thetaB = 296
1586         #Opening = 0.291193 optimal nominal
1587         tauIB = ca.fmin(tau1B, 4*(tauCB + thetaB)) #Integral time, corresponding
    to SIMC rules for integration processes.
1588         KpB = (1/(0.00065996849848586))*(1/(tauCB + thetaB)) #Proportional gain
    #0.0007929643656716369
1589         KiB = KpB/tauIB #Integral gain
1590         errorB = (h_spB - hB) #Difference between setpoint and measured   value
1591         #Calculate new controller output
1592         uB= ca.fmax(0, ca.fmin(1, (u_store[k-1][2] + (KpB*errorB + KiB*errorB- KpB
    *error_store[-1] ))))
1593         #Update the controller output for z_ov(oil valve separator)
1594         uk[2] = uB
1595         #Store all errors, to be used for previous errors
1596         error_store.append(errorB)
1597         u_Bplot.append(uB)
1598         if uB == 1 or uB == 0:
1599             Condition == 3
1600 
```

```
1601     if Condition == 3:
1602          #PI controller, tuned with SIMC rules(integration process) well 4
1603         hC = z_store[k][79]
1604         h_spC = 10.0000 #
1605         tauCC = 2000#572 #Controller time, can be changed up or down depending on
     needs for fast control or smooth control
1606         tau1C = 572
1607         #thetaC = 500#296
1608         #Opening = 0.291193 optimal nominal
1609         tauIC = ca.fmin(tau1C, 4*(tauCC + thetaC)) #Integral time, corresponding
     to SIMC rules for integration processes.
1610         KpC = (1/(0.0006739857389595 32))*(1/(tauCC + thetaC)) #Proportional gain
     #0.0007929643656716369
1611         KiC = KpC/tauIC #Integral gain
1612         errorC = (h_spC - hC) #Difference between setpoint and measured   value
1613         #Calculate new controller output
1614         uC= ca.fmax(0, ca.fmin(1, (u_store[k-1][3] + (KpC*errorC + KiC*errorC- KpC
     *error_store[-1] ))))
1615         #Update the controller output for z_ov(oil valve separator)
1616         uk[3] = uC
1617         #Store all errors, to be used for previous errors
1618         error_store.append(errorC)
1619         u_Cplot.append(uC)
1620         if uC == 1 or uC == 0:
1621             Condition = -1
1622 
1623     if Condition == -1:
1624         uk[0] == 0
1625 
1626     #Min selector, with logic to open previous valves
1627     openings = [uk[0] - u_store[k-1][0] , uk[4] - u_store[k-1][4], uk[2] - u_store
     [k-1][2],uk[3] - u_store[k-1][3]] #Find the valve from the split range
     controller that is active
1628     relatedtoop = [0,4,2,3] #The position in u_gl
1629     diffvalve, activevalve = next(((num, idx) for idx, num in enumerate(openings)
     if num != 0), (0, Condition)) #Find change in active valve and it's related
     index
1630     opt_open = [0.64172, 0.60811, 0.545984, 0.367095] #Nominal openings
1631     difffromopt = opt_open[activevalve] - u_store[k][relatedtoop[activevalve]] #
     meaures difference from nominal opening
1632     if diffvalve > difffromopt: #Min selector looks at which of the inputs is the
     smalles
1633         uk[relatedtoop[activevalve]] = opt_open[activevalve]
1634         #Restart the valve openings when the constraint is not active
1635         if uk[relatedtoop[activevalve]] == opt_open[activevalve] and Condition !=
     0:
1636             Condition -= 1
1637 
1638 
1639     #
     ##########################################################################################
1640 
1641 
1642     ############### Constant Level control
     ########################################
1643     #PI controller, tuned with SIMC rules(integration process) level separator
1644     h8 = x_store[k][21]
1645     h_sp8 = 1.64904#1.64915  #
1646     tauC8 = 500#500#50 #Controller time, can be changed up or down depending on
     needs for fast control or smooth control
1647     tauI8 = 4*tauC8 #Integral time, corresponding to SIMC rules for integration
     processes.
1648     Kp8 = 1/(0.005*tauC8)#Proportional gain
```

```
1649     Ki8 = Kp8/tauI8 #Integral gain
1650     error8 = (h_sp8 - h8) #Difference between setpoint and measured   value
1651     #Calculate new controller output
1652     u8 = ca.fmax(0, ca.fmin(1, (u_store[k-1][6] - (Kp8*error8 + Ki8*error8 - Kp8*
    error_store8[k-1] ))))
1653     #Update the controller output for z_ov(oil valve separator)
1654     uk[6] = u8
1655     #Store all errors, to be used for previous errors
1656     error_store8.append(error8)
1657     u_8plot.append(u8)
1658 
1659 
1660     ############### HH and LL level control
    ########################################
1661     #PI controller, tuned with SIMC rules(integration process) level separator
1662     if x_store[k][21] < 0.8:#1.484:
1663         h8 = x_store[k][21]
1664         h_sp8 = 0.8#1.484#1.64904#1.64915  #
1665         tauC8 = 100 #Controller time, can be changed up or down depending on needs
     for fast control or smooth control
1666         tauI8 = 4*tauC8 #Integral time, corresponding to SIMC rules for
    integration processes.
1667         Kp8 = 1/(0.005*tauC8)#Proportional gain 0.00005
1668         Ki8 = Kp8/tauI8 #Integral gain
1669         error8 = (h_sp8 - h8) #Difference between setpoint and measured   value
1670         #Calculate new controller output
1671         u8 = ca.fmax(0, ca.fmin(1, (u_store[k-1][6] - (Kp8*error8 + Ki8*error8 -
    Kp8*error_store8[-1] ))))
1672         #Update the controller output for z_ov(oil valve separator)
1673         uk[6] = u8
1674         #Store all errors, to be used for previous errors
1675         error_store8.append(error8)
1676         u_8plot.append(u8)
1677     if x_store[k][21] > 2.5:
1678         h8 = x_store[k][21]
1679         h_sp8 = 2.5#1.814#1.64904#1.64915  #
1680         tauC8 = 100 #Controller time, can be changed up or down depending on needs
     for fast control or smooth control
1681         tauI8 = 4*tauC8 #Integral time, corresponding to SIMC rules for
    integration processes.
1682         Kp8 = 1/(0.005*tauC8)#Proportional gain 0.00005
1683         Ki8 = Kp8/tauI8 #Integral gain
1684         error8 = (h_sp8 - h8) #Difference between setpoint and measured   value
1685         #Calculate new controller output
1686         u8 = ca.fmax(0, ca.fmin(1, (u_store[k-1][6] - (Kp8*error8 + Ki8*error8 -
    Kp8*error_store8[-1] ))))
1687         #Update the controller output for z_ov(oil valve separator)
1688         uk[6] = u8
1689         #Store all errors, to be used for previous errors
1690         error_store8.append(error8)
1691         u_8plot.append(u8)
1692 
1693 
1694     ################################
1695     #appending the resulting x values from the integration
1696     ################################
1697 
1698     m_ga1plot.append(xf[0])
1699     m_ga2plot.append(xf[1])
1700     m_ga3plot.append(xf[2])
1701     m_ga4plot.append(xf[3])
1702     m_ga5plot.append(xf[4])
1703     m_ga6plot.append(xf[5])
1704     m_gt1plot.append(xf[6])
```

```
1705     m_gt2plot.append(xf[7])
1706     m_gt3plot.append(xf[8])
1707     m_gt4plot.append(xf[9])
1708     m_gt5plot.append(xf[10])
1709     m_gt6plot.append(xf[11])
1710     m_ot1plot.append(xf[12])
1711     m_ot2plot.append(xf[13])
1712     m_ot3plot.append(xf[14])
1713     m_ot4plot.append(xf[15])
1714     m_ot5plot.append(xf[16])
1715     m_ot6plot.append(xf[17])
1716     m_grplot.append(xf[18])
1717     m_orplot.append(xf[19])
1718 
1719     p_gsplot.append(xf[20])
1720     h_lsplot.append(xf[21])
1721     p_s1plot.append(xf[22])
1722     p_d1plot.append(xf[23])
1723     w_c1plot.append(xf[24])
1724     p_s2plot.append(xf[25])
1725     p_d2plot.append(xf[26])
1726     w_c2plot.append(xf[27])
1727     p_s3plot.append(xf[28])
1728     p_d3plot.append(xf[29])
1729     w_c3plot.append(xf[30])
1730     m_glplot.append(xf[31])
1731 
1732     ################################
1733     #appending u-values
1734     ################################
1735     u_gl1plot.append(uk[0])
1736     u_gl2plot.append(uk[1])
1737     u_gl3plot.append(uk[2])
1738     u_gl4plot.append(uk[3])
1739     u_gl5plot.append(uk[4])
1740     u_gl6plot.append(uk[5])
1741     z_ovplot.append(uk[6])
1742     u_1plot.append(uk[7])
1743     #u_2plot.append(uk[14])
1744     #u_3plot.append(uk[15])
1745     #u_4plot.append(uk[16])
1746     u_rec1plot.append(uk[17])
1747     u_rec2plot.append(uk[18])
1748     u_rec3plot.append(uk[19])
1749     u_pc1plot.append(uk[8])
1750     u_pc2plot.append(uk[9])
1751     u_pc3plot.append(uk[10])
1752     u_pc4plot.append(uk[11])
1753     u_pc5plot.append(uk[12])
1754     u_pc6plot.append(uk[13])
1755     ################################
1756     #appending z-values
1757     ################################
1758 
1759     p_ai1plot.append(zk[0])
1760     p_ai2plot.append(zk[1])
1761     p_ai3plot.append(zk[2])
1762     p_ai4plot.append(zk[3])
1763     p_ai5plot.append(zk[4])
1764     p_ai6plot.append(zk[5])
1765     p_wh1plot.append(zk[6])
1766     p_wh2plot.append(zk[7])
1767     p_wh3plot.append(zk[8])
1768     p_wh4plot.append(zk[9])
```

```
1769     p_wh5plot.append(zk[10])
1770     p_wh6plot.append(zk[11])
1771     p_wi1plot.append(zk[12])
1772     p_wi2plot.append(zk[13])
1773     p_wi3plot.append(zk[14])
1774     p_wi4plot.append(zk[15])
1775     p_wi5plot.append(zk[16])
1776     p_wi6plot.append(zk[17])
1777     p_bh1plot.append(zk[18])
1778     p_bh2plot.append(zk[19])
1779     p_bh3plot.append(zk[20])
1780     p_bh4plot.append(zk[21])
1781     p_bh5plot.append(zk[22])
1782     p_bh6plot.append(zk[23])
1783 
1784     rho_ai1plot.append(zk[24])
1785     rho_ai2plot.append(zk[25])
1786     rho_ai3plot.append(zk[26])
1787     rho_ai4plot.append(zk[27])
1788     rho_ai5plot.append(zk[28])
1789     rho_ai6plot.append(zk[29])
1790     rho_m1plot.append(zk[30])
1791     rho_m2plot.append(zk[31])
1792     rho_m3plot.append(zk[32])
1793     rho_m4plot.append(zk[33])
1794     rho_m5plot.append(zk[34])
1795     rho_m6plot.append(zk[35])
1796     w_iv1plot.append(zk[36])
1797     w_iv2plot.append(zk[37])
1798     w_iv3plot.append(zk[38])
1799     w_iv4plot.append(zk[39])
1800     w_iv5plot.append(zk[40])
1801     w_iv6plot.append(zk[41])
1802     w_pc1plot.append(zk[42])
1803     w_pc2plot.append(zk[43])
1804     w_pc3plot.append(zk[44])
1805     w_pc4plot.append(zk[45])
1806     w_pc5plot.append(zk[46])
1807     w_pc6plot.append(zk[47])
1808     w_pg1plot.append(zk[48])
1809     w_pg2plot.append(zk[49])
1810     w_pg3plot.append(zk[50])
1811     w_pg4plot.append(zk[51])
1812     w_pg5plot.append(zk[52])
1813     w_pg6plot.append(zk[53])
1814     w_po1plot.append(zk[54])
1815     w_po2plot.append(zk[55])
1816     w_po3plot.append(zk[56])
1817     w_po4plot.append(zk[57])
1818     w_po5plot.append(zk[58])
1819     w_po6plot.append(zk[59])
1820     w_ro1plot.append(zk[60])
1821     w_ro2plot.append(zk[61])
1822     w_ro3plot.append(zk[62])
1823     w_ro4plot.append(zk[63])
1824     w_ro5plot.append(zk[64])
1825     w_ro6plot.append(zk[65])
1826     w_rg1plot.append(zk[66])
1827     w_rg2plot.append(zk[67])
1828     w_rg3plot.append(zk[68])
1829     w_rg4plot.append(zk[69])
1830     w_rg5plot.append(zk[70])
1831     w_rg6plot.append(zk[71])
1832     p_rhplot.append(zk[72])
```

```
1833    rho_rplot.append(zk[73])
1834
1835    p_mplot.append(zk[74])
1836    w_prplot.append(zk[75])
1837    w_toplot.append(zk[76])
1838    w_tgplot.append(zk[77])
1839    w_osplot.append(zk[78])
1840    w_gsplot.append(zk[79])
1841    rho_gsplot.append(zk[80])
1842    p_osplot.append(zk[81])
1843    v_osplot.append(zk[82])
1844    v_gsplot.append(zk[83])
1845    w_in1plot.append(zk[84])
1846    w_out1plot.append(zk[85])
1847    rho_in1plot.append(zk[86])
1848    rho_d1plot.append(zk[87])
1849    Phi1plot.append(zk[88])
1850    Pow1plot.append(zk[89])
1851    y_p1plot.append(zk[90])
1852    n_p1plot.append(zk[91])
1853    Phi_max1plot.append(zk[92])
1854    gamma_2_dummy1plot.append(zk[93])
1855    w_rec1plot.append(zk[94])
1856    w_in2plot.append(zk[95])
1857    w_out2plot.append(zk[96])
1858    rho_in2plot.append(zk[97])
1859    rho_d2plot.append(zk[98])
1860    Phi2plot.append(zk[99])
1861    Pow2plot.append(zk[100])
1862    y_p2plot.append(zk[101])
1863    n_p2plot.append(zk[102])
1864    Phi_max2plot.append(zk[103])
1865    gamma_2_dummy2plot.append(zk[104])
1866    w_rec2plot.append(zk[105])
1867    w_in3plot.append(zk[106])
1868    w_out3plot.append(zk[107])
1869    rho_in3plot.append(zk[108])
1870    rho_d3plot.append(zk[109])
1871    Phi3plot.append(zk[110])
1872    Pow3plot.append(zk[111])
1873    y_p3plot.append(zk[112])
1874    n_p3plot.append(zk[113])
1875    Phi_max3plot.append(zk[114])
1876    gamma_2_dummy3plot.append(zk[115])
1877    w_rec3plot.append(zk[116])
1878    w_gl1plot.append(zk[117])
1879    w_gl2plot.append(zk[118])
1880    w_gl3plot.append(zk[119])
1881    w_gl4plot.append(zk[120])
1882    w_gl5plot.append(zk[121])
1883    w_gl6plot.append(zk[122])
1884    p_outplot.append(zk[123])
1885    rho_outplot.append(zk[124])
1886    ######
1887
1888    B.append(-0.6*zk[78] + 0.03*(zk[89] + zk[100] + zk[111]))
1889
1890    #Nullspace positive
1891    C1.append(0.95155469*zk[19] + 0.30747954*zk[7])
1892    C2.append(0.97178607*zk[19] + 0.23586401*zk[1])
1893    C3.append(0.93394895*zk[19] + 0.35740644*zk[74])
1894    C4.append(0.99906231*zk[19] - 0.04329552*xf[29])
1895    #Nullspace negative
1896    C5.append(0.26764386*zk[19] + 0.96351791*zk[7])
```

```
    C6.append(0.99982427*zk[19] + 0.0187467*zk[1])
    C7.append(0.87581411*zk[19] - 0.48264858*xf[20])
    C8.append(0.03331483*zk[19] - 0.99944491*zk[74])
    C9.append(0.99994079*zk[19] - 0.01088206*xf[29])

    #Nullspace positive New
    C1.append(0.99819296*zk[19] - 0.06009004*zk[7]) #132.1386341894136
    C2.append(0.99918968*zk[19] - 0.04024904*zk[1]) #133.03309302
    C3.append(0.99797747*zk[19] - 0.06356854*zk[74]) #c_ns = 131.95168963
    C4.append(0.9999867*zk[19] +  0.00515668*xf[29]) #c_ns = 138.05023113
    #Nullspace negative New
    C5.append( 0.99992464*zk[19] - 0.01227675*zk[7])  #c_ns = 136.23092774
    C6.append(0.99999641*zk[19] - 0.00267783*zk[1]) #c_ns = 136.9586203
    C7.append(0.99371152*zk[19] +  0.11197063*xf[20]) #c_ns = 138.81967765
    C8.append(0.9999131*zk[19]  - 0.01318273*zk[74]) #c_ns = 136.18182529
    C9.append(0.999999889*zk[19] - -4.70335652*10**(-4)*xf[29]) #c_ns =
    137.15610524



    #Test multiple controllers
    C10.append(5.55555562*10**(-6)*zk[19] + 1*xf[20])
    C11.append(0.04200139*zk[7] + 0.99843881*zk[19] + 0.00643228*zk[1] +
    0.00407959*zk[74] - 0.03602514*xf[29])#ugl2
    C12.append(-0.17305043*zk[7] + 0.00643228*zk[19] + 0.97349828*zk[1] -
    0.01680838*zk[74] + 0.1484276*xf[29])#u1
    C13.append(-0.10975503*zk[7] +  0.00407959*zk[19] - 0.01680838*zk[1] +
    0.9893395*zk[74] + 0.09413832*xf[29])#ugl3
    C14.append(0.96919972*zk[7] - 0.03602514*zk[19] + 0.1484276*zk[1] +
    0.09413832*zk[74] + 0.16870481*xf[29])#ugl4

    #Positive Exact local method
    C15.append(-446413.25092053*zk[19] + 19345.66174978*xf[29])
    C16.append(-45.41318981*zk[19] - 0.08392059*zk[55])
    C17.append(-2193.44492237*zk[19] - 3680.37324392*zk[49])
    C18.append(-19877.79126375*zk[19] - 9122.48640609*zk[107])
    C19.append(257.76699191*zk[55] - 81.78981924*zk[107])
    C20.append(2804.15788752*zk[49] - 750.21433487*zk[107])

    #Negative Exact local method
    C21.append(-483783.15402796*zk[19] - 9043.21957475*zk[1])

    C22.append(-6.26485343*10**(4)*zk[19] + 26.0408009*zk[55])
    C23.append(-62520.82233189*zk[19] + 613.91453191*zk[49])
    C24.append(-125435.8109666*zk[19] - 12587.91389531*zk[107])
    C25.append(259.40899698*zk[55] + 122.2241566*zk[107])
    C26.append(3051.23091538*zk[49] + 57.97844937*zk[107])

    #Positive Exact local method new
    C15.append(-539914.34688897*zk[19] - 2784.25256079*xf[29])
    C16.append(-9.20434088*10**(2)*zk[19] + 0.491188339*zk[55])
    C17.append(-42892.00474827*zk[19] + 22740.06185649*zk[49])
    C18.append(-293089.71870215*zk[19] + 21259.28465932*zk[107])
    C19.append(286.77917524*zk[55] + 15.07388955*zk[107])
    C20.append(2900.46415236*zk[49] - 388.45472794*zk[107])

    #Negative Exact local method new
    C21.append(-5.34574111*10**(5)*zk[19] - 1.41642159*zk[7])

    C22.append(-4.32905553*10**(5)*zk[19] + 2.30539895*10**(2)*zk[55])
    C23.append(-466615.86146777*zk[19] + 17771.12021649*zk[49])
    C24.append(-526464.96151215 *zk[19] + 5189.13501704*zk[107])
```

```
1956     C25.append(284.98125313*zk[55] + 26.83981752*zk[107])
1957     C26.append(3016.84474179*zk[49] - 270.09782426*zk[107])
1958
1959
1960
1961
1962
1963     #Exact local method 2MV/2 dist
1964     #Positive GOR
1965     B11.append(-7.98501232*zk[1] - 64.82349538*zk[19])
1966     B12.append(2.60067824*zk[1] + 16.08713207*zk[19])
1967
1968     B21.append(22.88897358*zk[1] - 22.67672343*zk[5])
1969     B22.append(-22.18074525*zk[1] + 24.49319288*zk[5])
1970
1971     B31.append(-4.3656668*xf[29] - 26.84908555*zk[5])
1972     B32.append(10.22381382*xf[29] + 65.47016409*zk[5])
1973
1974     B41.append(1290.90504975*xf[29] + 16765.46161926*zk[5] + 17129.3708444*zk[23])
1975     B42.append(-1676.90699719*xf[29] - 21807.04221199*zk[5]  - 22311.54379972*zk
         [23])
1976
1977     B51.append(61119.14498099*xf[29] + 147027.68286197*zk[5] - 58183.45153308*zk
         [23] + 201082.69102819*zk[1])
1978     B52.append(62702.16245358*xf[29] + 118363.56280388*zk[5]  - 103353.02902385*zk
         [23] + 216378.02726489*zk[1])
1979
1980     B61.append(10.56433836*xf[29] + 64.592046*zk[1])
1981     B62.append(-4.68023771*xf[29] - 27.54634754*zk[1])
1982
1983     B71.append(-2855.6340659*xf[29] - 19423.56983834*zk[1] - 16311.40372815*zk
         [19])
1984     B72.append(-6405.52726445*xf[29] - 43548.86813336*zk[1]  - 36426.92700546*zk
         [19])
1985
1986     B81.append(-3716.69625337*xf[29] - 4798.98978458*zk[11] - 15641.39990574*zk
         [23] - 20136.44541368*zk[1])
1987     B82.append(10994.18774889*xf[29] + 96974.47619766*zk[11]  - 49965.87429334*zk
         [23] - 29143.59981315*zk[1])
1988
1989     #Negative GOR
1990     B91.append(-8361.73566402*zk[1] - 447667.21081462*zk[19])
1991     B92.append(1732.29646306*zk[1] + 92410.72240929*zk[19])
1992
1993     B101.append(27.44028606*zk[1] + 0.88681124*zk[5])
1994     B102.append(0.19466741*zk[1] + 56.00386487*zk[5])
1995
1996     B111.append(-17.41643364*xf[29] - 12.97245266*zk[5])
1997     B112.append(-10.17879649*xf[29] + 47.6991263*zk[5])
1998
1999     B121.append(-478.12873683*xf[29] + 2230.28224625*zk[5] + 96372.60899258*zk
         [23])
2000     B122.append(2147.79026662*xf[29] - 10459.67002223*zk[5]  - 451407.75988817*zk
         [23])
2001
2002     B131.append(-25887.46368923*xf[29] - 17940.01178791*zk[5] + 96464.81364318*zk
         [23] - 43658.25180635*zk[1])
2003     B132.append(-173556.7739217*xf[29] - 149936.46976775*zk[5]  - 450770.1683109*
         zk[23] - 301895.11536725*zk[1])
2004
2005     B141.append(-15.291117*xf[29] + 1.09866488*zk[1])
2006     B142.append(-169.14956472*xf[29] - 290.9096779*zk[1])
2007
```

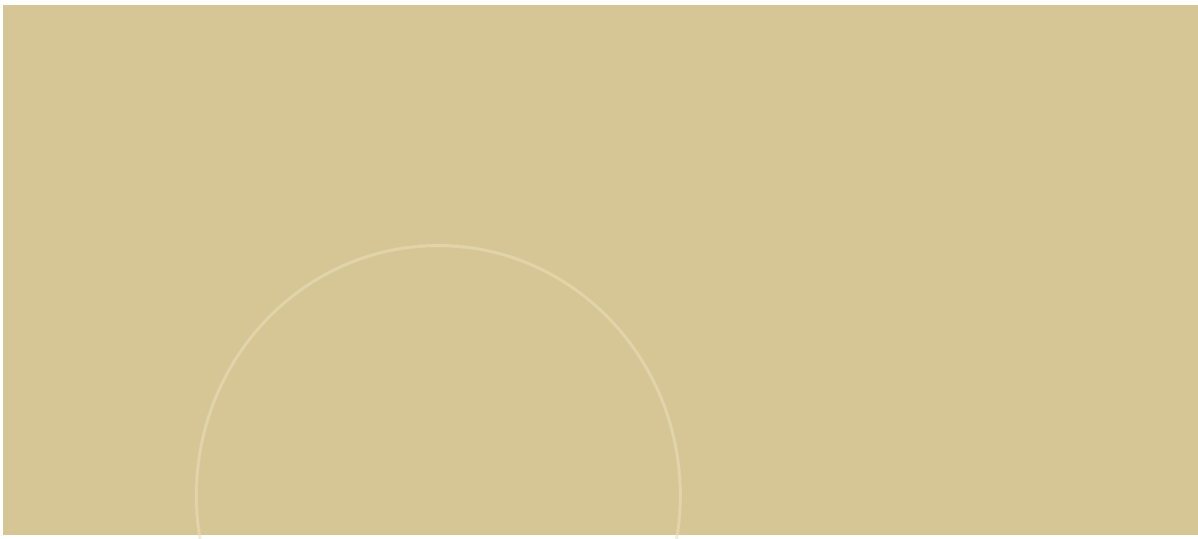
```
2008    B151.append(1771.19498418*xf[29] - 5980.18444265*zk[1] - 483434.7839778*zk
        [19])
2009    B152.append(-551.82431496*xf[29] + 990.31272722*zk[1]  + 103554.28183372*zk
        [19])
2010
2011    B161.append(3101.32595098*xf[29] + 83580.47286321*zk[11] - 483442.38333034*zk
        [23] - 3881.96886745*zk[1])
2012    B162.append(9522.55485732*xf[29] + 633036.44230348*zk[11]  + 103496.72452484*
        zk[23] + 16882.14580032*zk[1])
2013
2014
2015
2016    #Exact local method 2MV/2 dist new F = dy/du
2017    #Positive GOR
2018    B11.append(262.73305154*zk[1] - 5405.75910492*zk[19])
2019    B12.append(-49.55775679*zk[1] + 1024.08822321*zk[19])
2020
2021    B21.append(21.3202415*zk[1] - 17.0239858*zk[5])
2022    B22.append(-16.24253356*zk[1] + 13.61745927*zk[5])
2023
2024    B31.append(-2.73047731*xf[29] - 18.56622293*zk[5])
2025    B32.append(5.69386534*xf[29] + 39.39328154*zk[5])
2026
2027    B41.append(-4776.52311651*xf[29] + 13848.85934925*zk[5] + 134604.1345925*zk
        [23])
2028    B42.append(14989.20015347*xf[29] - 43486.3050394*zk[5]  - 422482.0911806*zk
        [23])
2029
2030    B51.append(44144.59427475*xf[29] + 186246.51824044*zk[5] +  127291.28221963*zk
        [23] +  197772.20401727*zk[1])
2031    B52.append(52545.97703735*xf[29] + 88863.50125175*zk[5]  - 428096.17318851*zk
        [23] + 151829.86277082*zk[1])
2032
2033    B61.append(6.97905258*xf[29] + 59.01674559*zk[1])
2034    B62.append(-2.34535959*xf[29] - 19.47440931*zk[1])
2035
2036    B71.append(-8217.86437896*xf[29] - 43636.39030824*zk[1] - 510211.17724758*zk
        [19])
2037    B72.append(1451.93134491*xf[29] + 7706.53437138*zk[1]  + 90213.05215366*zk
        [19])
2038
2039    B81.append(-18668.46283178*xf[29] - 88207.07033211*zk[11] - 509535.79352571*zk
        [23] - 53203.53374925*zk[1])
2040    B82.append(42011.02824372*xf[29] + 342334.37720436*zk[11]  + 87591.86697055*zk
        [23] + 44836.91495394*zk[1])
2041
2042
2043
2044    #Negative GOR
2045    B91.append(5.66780409*10**(2)*zk[1] - 1.29304852*10**(5)*zk[19])
2046    B92.append(-1.08955450*10**(2)*zk[1] + 92410.72240929*10**(4)*zk[19])
2047    B101.append(19.34328063*zk[1] - 12.68412868*zk[5])
2048    B102.append(-12.60618675*zk[1] +  28.29496047*zk[5])
2049    B111.append(-0.92565692*xf[29] + 1.59970784*zk[5])
2050    B112.append(-1.29256536*xf[29] + 20.86229131*zk[5])
2051    B121.append(1124.0888743*xf[29] + 13506.02748707*zk[5] + 84996.94427602*zk
        [23])
2052    B122.append(-5895.11552797*xf[29] - 70727.29636203*zk[5]  - 445289.30784748*zk
        [23])
2053    B131.append(-113289.79962532*xf[29] - 426206.37623219*zk[5] + 124911.1956689*
        zk[23] +  810049.2431397*zk[1])
2054    B132.append(13751.40257385*xf[29] +  4777.67783779*zk[5]  - 452143.16163039*zk
        [23] - 139097.16143212*zk[1])
2055    B141.append(-14.89941033*xf[29] + 91.88326244*zk[1])
```

```
2056     B142.append(-4.83431855*xf[29] - 25.85243427*zk[1])
2057     B151.append(-2.04592345*10**(3)*xf[29] + 1.31255197*10**(4)*zk[1] -
         5.49032199*10**(5)*zk[19])
2058     B152.append(3.72185169*10**(2)*xf[29] - 2.39358468*10**(3)*zk[1]  +
         1.01916981*10**(5)*zk[19])
2059     B161.append(-27598.25690498*xf[29] - 507211.84911125*zk[11] - 545266.45637924*
         zk[23] + 293076.88722314*zk[1])
2060     B162.append(18765.22060244*xf[29] + 365100.33533589*zk[11]  + 99206.33061025*
         zk[23] - 203907.68245748*zk[1])
2061
2062
2063     #Exact local method 2MV/2 dist new F = dy/du
2064     #Positive GOR
2065     B11.append(3583.10465771*zk[1] - 1449.41787407*zk[19])
2066     B12.append(-146.18640435*zk[1] + 67.56021995*zk[19])
2067
2068     B21.append(413.16600486*zk[1] + 326.50490982*zk[5])
2069     B22.append(305.9743484*zk[1] + 663.475804*zk[5])
2070
2071     B31.append(-6976.59161527*xf[29] + 3401.78133543*zk[5])
2072     B32.append(-8339.24832552*xf[29] + 4481.00108404*zk[5])
2073
2074     B41.append(-234922.1903769*xf[29] + 476847.10061302*zk[5]  - 181679.62737955*
         zk[23])
2075     B42.append(-549333.90812753*xf[29] + 1128132.39347296*zk[5]  -
         431189.32210637*zk[23])
2076
2077     B51.append(-805763.43006082*xf[29] + 131567.44007523*zk[5] +  291161.30688082*
         zk[23] +  427066.88982393*zk[1])
2078     B52.append(-691151.33978905*xf[29] + 1042352.54566528*zk[5]  - 313718.6801493*
         zk[23] +  106098.72807374*zk[1])
2079
2080     B61.append(-2704.12302706*xf[29] +  327.18670482*zk[1])
2081     B62.append(-2675.80879344*xf[29] + 68.22602697*zk[1])
2082
2083     B71.append(-41674.87829334*xf[29] - 52288.11647798*zk[1] + 23376.24685595*zk
         [19])
2084     B72.append(-403800.65877812*xf[29] - 541499.5586333*zk[1]  + 240611.02868619*
         zk[19])
2085
2086     B81.append(-686015.79742427*xf[29] - 613181.02538563*zk[11] - 334371.87860306*
         zk[23] + 825709.36229966*zk[1])
2087     B82.append(-659260.97418188*xf[29] - 243106.42626252*zk[11]  + 98775.47820129*
         zk[23] - 193401.98141469*zk[1])
2088
2089
2090 print('Power 2:', Pow2plot[-1])
2091 print(Power2)
2092 print('Power 3:', Pow3plot[-1])
2093 print(Power3)
2094 print('Oil prod :', w_osplot[-1])
2095 print(Oil)
2096 print('Cost caluclated in the simulations',B[-1])
2097 print('Recycle flow 1', w_rec1plot[-1])
2098 print('Recycle flow 2', w_rec2plot[-1])
2099 print('Recycle flow 3', w_rec3plot[-1])
2100 print('Flow through compressor 1', w_c1plot[-1])
2101 print('Flow through compressor 2', w_c2plot[-1])
2102 print('Flow through compressor 3', w_c3plot[-1])
2103
2104 print('Separator pressure sec last: ', p_gsplot[-2])
2105 print('Separator level last: ', h_lsplot[-1])
2106 print('Separator level sec last: ', h_lsplot[-2])
2107 print('Oil flow into separator: ', w_toplot[-2])
```

```
2108 print('Oil out sep:', w_osplot[-1])
```

B.13 Calculations.py

This



NTNU
Norwegian University of
Science and Technology