Master's thesis

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

Maren Sofie Lia

Implementation of Real Time Optimization in an Experimental Lab Rig via Output Modifier Adaptation using Gaussian Processes

Master's thesis in Chemical Engineering and Biotechnology
Supervisor: Johannes Jäschke
Co-supervisor: Evren Turan
June 2022

NTNU
Norwegian University of
Science and Technology

Maren Sofie Lia

Implementation of Real Time Optimization in an Experimental Lab Rig via Output Modifier Adaptation using Gaussian Processes

Master's thesis in Chemical Engineering and Biotechnology
Supervisor: Johannes Jäschke
Co-supervisor: Evren Turan
June 2022

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

NTNU
Norwegian University of
Science and Technology

Abstract

In this thesis Real Time Optimization (RTO) via Output Modifier Adaptation with Gaussian Processes (MAy-GP) is implemented in an experimental lab rig representing a subsea oil well network with gas lift. Traditional RTO aims to maximize the production and minimize the operational costs of a continuously operating process plant in real time using a model. The optimized inputs will be optimal for the model, but not for the plant unless the model is a perfect representation of the plant.

The Output Modifier Adaptation (MAy) approach can be a possible solution to the challenges related to traditional RTO. Unlike traditional RTO, MAy relies on a fixed steady-state process model and has the theoretical ability to converge to plant optimum despite plant-model mismatch. This is ensured by modifiers, which modify the original optimization problem in such a way that plant optimum is achieved. This approach depends on accurate plant gradients in order to ensure plant optimality. Obtaining accurate estimates of plant gradients in real systems is challenging due to the presence of noisy measurements.

The gradient estimation step can be replaced by using Gaussian Processes (GPs). The goal of this thesis is to investigate representing the plant-model mismatch by GPs and using these GPs to calculate the modifiers needed for MAy. This is tested by simulations and implementation in an experimental lab rig. There is a gap in the literature when it comes to implementations of MAy in real systems, and this thesis will contribute to fill this gap.

The proposed MAy-GP scheme is first applied to a simulation to study the performance of the algorithm. The simulation results showed that the inputs with the MAy-GP algorithm converged to the optimum when there was no noise present. Measurement noise was then implemented in the simulation to match the noise level in the lab rig. In this simulation, it was clear that the noise hampered the GPs. However, adequate performance was achieved. The experimental results from the rig showed that the MAy-GP algorithm could not predict accurate estimates of the curvature of the plant-model mismatch, due to the high noise level. Thus, the GPs were not able to fully correct the model and the inputs did not converge to the plant optimum.

Sammendrag

I denne oppgaven blir Real Time Optimization (RTO) via Output Modifier Adaptation med Gaussian Processes (MAy-GP) implementert i en eksperimentell lab rigg. Riggen representerer et nettverk bestående av tre oljebrønner med gassløft. Tradisjonell RTO maksimerer profitten i industrielle prosessanlegg i sanntid ved bruk av en modell av systemet. Små avvik i modellen kan føre til at systemet ikke konvergerer til optimalt punkt.

Output Modifier Adaptation (MAy) metoden kan løse noen av utfordringene til tradisjonell RTO. Til forskjell fra tradisjonell RTO bruker MAy en fast modell av systemet som ikke oppdateres underveis. Til tross for at det finnes avvik fra systemet i modellen, vil MAy fortsatt ha mulighet til å konvergere til optimalt punkt for systemet. Denne egenskapen kommer av at MAy bruker modifikatorer til å modifisere det originale optimaliseringsproblemet. Modifikatorene krever nøyaktige målinger og estimater av gradienter i systemet, noe som kan være svært krevende å få til med høye støynivåer.

Estimering av gradienter kan bli erstattet med å bruke Gaussian Processes (GPs). Målet i denne oppgaven er å undersøke muligheten for å bruke GPs til å representere avviket mellom systemet og modellen, og deretter bruke GPs til å estimere modifikatorene som trengs. Dette er undersøkt i en simulering av den eksperimentelle lab riggen og ved eksperimenter i den faktiske lab riggen. I litteraturen finnes det få implementasjoner av MAy i ekte systemer, og denne oppgaven vil forsøke å fylle dette tomrommet.

MAy-GP er først implementert i en simulering av den eksperimentelle riggen. Resultatet fra simuleringen viste at algoritmen var i stand til å optimalisere systemet når det ikke var støy til stede i målingene. Støy i målingene ble deretter implementert i simuleringen for å gjøre simuleringen mer realistisk. I denne simuleringen ble det tydlig at et høyt støynivå påvirket ytelsen til algoritmen, og et avvik fra optimalt punkt ble observert. Resultatene fra implementasjonen i lab riggen viste at støynivået var for høyt til å estimere nøyaktige estimater av avviket mellom modell og system. Som følge at dette var ikke GPs i stand til å korrigere for avviket og input-ene konvergerte ikke til optimalt punkt for systemet.

Preface

This thesis was written in the spring of 2022 as the final part of my M.Sc in Chemical Engineering at the Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisor Johannes Jäschke for the opportunity to work on this topic during the specialization project in the fall of 2021 and the master thesis in the spring of 2022. I would also like to express a special gratitude to my co-supervisor Jose Otavio Assumpcao Matias during the specialization project and Evren Turan as my co-supervisor during the thesis. The support and guidance from all the supervisors have been essential in my work and are very appreciated.

Declaration of Compliance

I, Maren Sofie Lia, hereby declare that this is an independent work according to the exam regulations of the Norwegian University of Science and Technology (NTNU).
Signature:

Place and Date: Trondheim - Gløshaugen, June 2022

Table of Contents

[1]Section taken from the project work presented in [11] by the same author.
[2]Appendix taken from the project work presented in [11] by the same author.

List of Figures

List of Tables

Nomenclature

Acronyms

DAE	Differential algebraic equation
FDA	Finite-difference approximation
GP	Gaussian Process
KKT	Karush-Kuhn-Tucker
LICQ	Linear Independence Constraint Qualification
MA	Modifier Adaptation
MAy	Ouput Modifier Adaptation
MAy-GP	Combined Output Modifier Adaptation and Gaussian Processes
MPC	Model Predictive Controller
NLP	Nonlinear programming
PI	Proportional-integral
PID	Proportional-integral–derivative
RTO	Real Time Optimization

Symbols

δ_{pq}	Kronecker delta	
λ	First order modifier	
$\mathcal{L}$	Lagrange function	
μ	Lagrange multiplier	
ν	Measurement noise	
$\overline{\mathbf{U}}$	Observed inputs for GP prediction	
$\overline{\mathbf{y}}$	Observed outputs for GP prediction	
ρ_g	Gas density	$[\mathrm{kg/m^3}]$
ρ_{mix}	Mixture density	$[\mathrm{kg/m^3}]$
σ_f^2	Scale factor hyperparameter	
σ_n^2	Variance measurement noise	
θ	Model parameter	
ε	Zeroth order modifier	
b	ε filter gain	

d	λ filter gain	
d	Disturbances/plant parameters	
f	Steady-state model	
f	Uknown function	
G	Constraint	
J	Objective function	
k	Covariance function	
k_i	Filter gain for input i	
l	Length scale hyperparameter	
m	Mean function	
m_g	Gas hold up	[kg]
m_l	Liquid hold up	[kg]
m_{gout}	Well outlet gas	[kg/s]
m_{lout}	Well outlet liquid	[kg/s]
n_g	Number of constraints	
n_u	Number of inputs	
n_y	Number of outputs	
p_{bi}	Pressure before injection point	[bar]
p_{pump}	Reservoir pressure	[bar]
p_{rh}	Riser head pressure	[bar]
q	λ filter gain	
Q_g	Gas flowrate	[sL/min]
Q_l	Liquid flowrate	[L/min]
u	Input	
v_o	Valve opening	[-]
w_l	Liquid flowrate	[kg/s]
w_{total}	Total well production rate	[kg/s]
x	State	
y	Output	
z	Algebraic state	

Chapter 1

Introduction

1.1 Motivation

Optimal operation of continuously operating process plants is of major economic importance in the chemical industry, such as the offshore production of oil and gas. Optimal operation involves meeting goals in different time scales ranging from long-term planning to fast regulation for stable operation. Realizing all these goals as a whole is an unrealistic task. Operation is therefore decomposed into different decision making layers. Figure 1.1 illustrates the different decision making layers, where the RTO layer is of interest in this thesis. Real Time Optimization (RTO) has become a standard practice to improve production during the past years, especially in petrochemical processes [1]. RTO seek to minimize the cost or maximize the production of hour-by-hour operation, while satisfying the operational constraints. This layer typically provides setpoints to the lower layer [2]. The lower layer can be, among other things, a Model Predictive Controller (MPC) or a PID controller.

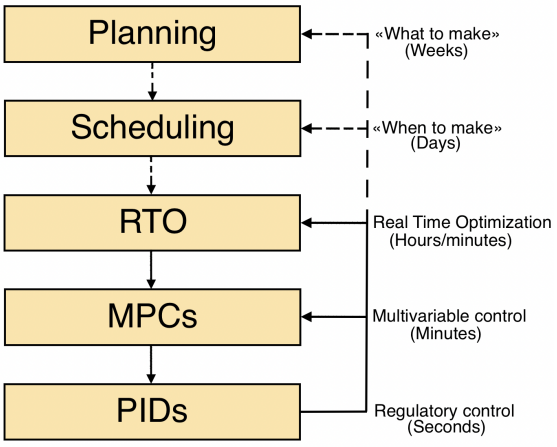


Figure 1.1: Plant decision hierarchy. Figure recreated from [3].

Traditional RTO is typically based on rigorous steady-state process models. The model parameters are updated by incorporating real time measurements of the process plant. After this step, an economic optimization is carried out with the updated model parameters to generate optimal setpoints for the process [4]. This technique will give adequate results if an appropriate process model and algorithm is used. However, optimal operation can be difficult to achieve with inaccurate process models or in the presence of disturbances [5]. Plant models are often based on an oversimplification of the process of interest, which leads to plant-model mismatch and poor results.

An RTO method called Modifier Adaptation (MA) can be used for addressing this issue. The MA approach modifies the optimization problem in every iteration in such a way that a Karush-Kuhn-Tucker (KKT) point for the model coincide with the real optimum of the plant. That is, MA enforce plant optimality despite the presence of parametric and structural plant-model mismatch. The MA scheme uses measurements of the plant and estimates of plant gradients to calculate zeroth-order and first-order modifiers. In standard MA, the modifiers are used for modifying the cost and constraint functions in the optimization problem [5]. An alternative MA strategy has been proposed, wherein the modifiers are applied to the outputs rather than to the cost and constraint functions [4]. This method is called Output Modifier Adaptation (MAy) and is of main interest in this thesis. The gradient estimation step is the main challenge in both implementations, especially in the case of noisy measurements.

Gaussian processes (GPs) is a well known tool to estimate unknown functions in the machine learning community. Moreover, GPs are also gaining attention in the field of control and optimization, due to its simplicity and wide range of uses. This approach is a probabilistic, non-parametric modeling approach which has the ability to capture complex unknown functions using very few variables [6]. This technique can be used for addressing the challenging gradient estimation step in the context of MAy. The idea is to use GPs to represent the plant-model mismatch in presence of noisy measurements. In other words, the usual modifiers in the MAy scheme are replaced by new modifiers calculated from GPs. To the best of the author's knowledge, combined MAy-GP schemes have not been proposed before. However, previous work that combine the standard MA and GPs are for example found in [6], [7] and [8], where the proposed schemes are applied to the Williams–Otto reactor problem.

There is a gap in the literature when it comes to MAy implementations in real systems. The implementations of MAy in the literature are mostly studied in simulations, like in [9] and [10]. Implementing MAy with GPs in a lab-scale setup is valuable to whether this approach can be used for optimizing real process systems in the industry.

1.2 Scope

The main goal of this thesis is to implement Real Time Optimization via Output Modifier Adaptation with Gaussian processes (MAy-GP) in an experimental lab rig. The lab rig represents a subsea oil well network with gaslift. The optimization objective function to be maximized is the profit, which in this case only depends on the production. The thesis covers:

- Implementation of MAy-GP in a simulation of the lab rig. This simulation consists of a dynamic model representing the behaviour of the lab rig. The simulation is used to study the performance of the algorithm, as well as to study the tuning parameters that affect the performance before it is applied to the actual lab rig.
- Implementation of MAy-GP in the actual experimental lab rig. The performance of the algorithm is compared to a previously developed traditional RTO algorithm.

This thesis is built on the specialization project from 2021, presented in [11] by the same author. Some of the sections in Chapter 2 and Chapter 3 are hence based on the work performed in the specialization project. The project work from 2021 covers implementations of standard MA and MAy in the same simulation of the experimental lab rig.

Chapter 2

General Concepts and Theory

In this chapter general concepts and theory will be introduced. This includes a general overview of the traditional steady-state Real Time Optimization approach, the Modifier Adaptation approach and Gaussian Processes. Section 2.2 is rewritten from the work performed in the specialization project presented in [11] by the same author.

2.1 Real Time Optimization

The most commonly used steady-state Real Time Optimization (RTO) method is the two-step approach. This approach is referred to as the traditional RTO approach and consists in solving two optimization problems [12]. The first one uses measurements of the outputs $\mathbf{y}$ of the process plant to estimate model parameters and unmeasured disturbances $\mathbf{d}$, such that the model error is minimized. The second optimization problem minimizes the cost, or alternatively optimizes the revenues of the plant with the updated model parameters to find a new optimal input $\mathbf{u}^*$. A block diagram of the traditional RTO is shown in Figure 2.1. The RTO layer typically provides the input as a setpoint to a lower regularization layer. In order to execute the optimization, steady-state has to be detected. The issue of steady-state detection is not covered in this thesis.

Traditional RTO is depending on rigorous and accurate process models. These models can be very costly to develop and obtaining them requires a solid understanding of the process. Incorrect model structures will lead to plant-model mismatch, which can result in the algorithm converging to a sub-optimal operating point. There are also computational issues associated to solving and simulating detailed and complex models online, which can lead to convergence issues and numerical failure. An RTO variant called Modifier Adaptation allows to overcome these challenges by introducing appropriate corrections to the model in the optimization problem.

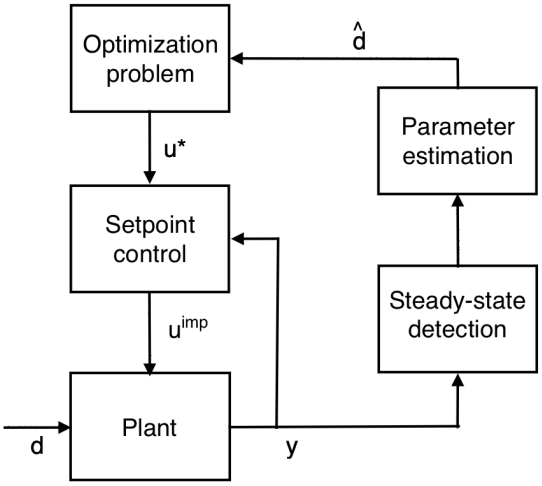


Figure 2.1: Traditional steady-state RTO block diagram. u is the input, y is the output measurements and d is the unmeasured model parameters and disturbances.

2.2 Modifier Adaptation

While traditional RTO uses measurements of the plant to improve the steady-state model, Modifier Adaptation relies on a fixed steady-state process model. A block diagram for MA is shown in Figure 2.2. The gradient estimation block and the modified optimization problem block are different from the traditional RTO block diagram. The idea behind this approach is to modify the optimization problem to enforce plant optimality, despite the presence of structural plant-model mismatch. This is done through modifiers, which are calculated from estimated plant gradients and measurements. The standard MA adds modifiers to the cost and constraint functions. An alternative is Output Modifier Adaptation (MAy), which adds modifiers to the outputs in the optimization problem.

Using an MA approach can solve some of the traditional RTO challenges, but there are still some challenges which need to be addressed:

- Steady-state wait time. MA and MAy are, as traditional RTO, steady-state optimization methods. The plant needs time to stabilize before the next iteration, which can be very time consuming. For instance, in case of frequent disturbances.
- Accurate plant measurements are needed for detecting steady-state, and especially for gradient estimation. The gradients can not be measured directly and they have to be estimated based on noisy plant measurements. There are many methods available for estimating plant gradients. One very common approach is to use Finite-difference approximation (FDA), described in Appendix A. This method is a steady-state perturbation method, which relies on steady-state data. The FDA approach requires usually $n_u + 1$ steady-state operating points to estimate the

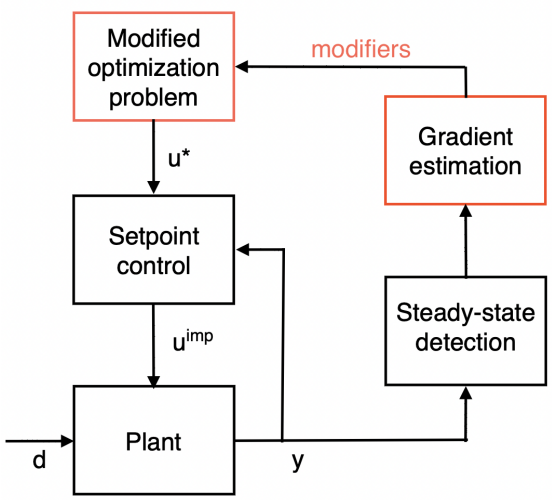


Figure 2.2: Modifier Adaptation block diagram. u is the input, y is the output measurements and d is the disturbances.

gradients, where n_u is the number of inputs [4]. Such methods includes noisy measurements, which may lead to poor gradient estimates.

- Dynamic limitations. Since MA and MAy are steady-state optimization methods, they do not take the dynamics of the process into account. The RTO layer calculates the optimal operating point without considering the transients. That is, the RTO layer does not consider how to actually reach the optimal operating point.

The following sub sections include detailed mathematical expressions relevant to the problem formulation. First, the steady-state optimization problem is introduced, followed by the necessary conditions of optimality. Then, the standard MA and the MAy scheme is presented.

2.2.1 Steady-State Optimization Problem

The steady-state production optimization problem for the plant can be formulated as:

$$\begin{aligned} \min_{\mathbf{u}} \quad & J_p(\mathbf{u}) := J(\mathbf{u}, \mathbf{y}_p(\mathbf{u}, \mathbf{d})) \\ \text{s.t.} \quad & \mathbf{G}_p(\mathbf{u}) := \mathbf{g}(\mathbf{u}, \mathbf{y}_p(\mathbf{u}, \mathbf{d})) \leq \mathbf{0} \\ & \mathbf{u}^L \leq \mathbf{u} \leq \mathbf{u}^U \end{aligned} \tag{2.1}$$

The notation p is used for variables associated with the plant. $\mathbf{u} \in \mathbb{R}^{n_u}$ denotes the input variables and $\mathbf{y}_p \in \mathbb{R}^{n_y}$ denotes the measured output variables of the plant. n_u and n_y are the number of inputs and outputs respectively. J_p: $\mathbb{R}^{n_u} \times \mathbb{R}^{n_y} \to \mathbb{R}$ is the operational costs of the plants which should be minimized. This function is the objective

function in the optimization problem. $\mathbf{g}$ is a set of n_g inequality constraint functions, where g_i: $\mathbb{R}^{n_u} \times \mathbb{R}^{n_y} \to \mathbb{R}$, $i = 1, \ldots, n_g$. The inequality constraints are often operational limitations. $\mathbf{y}_p(\mathbf{u},\mathbf{d})$ represents the steady-state input-output mapping of the plant, where $\mathbf{d} \in \mathbb{R}^{n_d}$ are representing plant parameters and disturbances. n_d is the number of plant parameters and disturbances. $\mathbf{u}^L$ and $\mathbf{u}^U$ are the lower and upper bounds of $\mathbf{u}$ respectively. These bounds are not dependent on $\mathbf{y}_p$ and are therefore not affected by uncertainty. The objective function and constraint functions are assumed to be known directly from the measurements. Since $\mathbf{y}_p(\mathbf{u},\mathbf{d})$ is typically unknown, a steady-state process model is used for solving the optimization problem (2.1)

$$\begin{aligned} f(\mathbf{u}, \mathbf{x}, \mathbf{d}) &= \mathbf{0} \\ \mathbf{y} &= \mathbf{h}(\mathbf{u}, \mathbf{x}, \mathbf{d}) \end{aligned} \tag{2.2}$$

where f is the steady-state process model and $\mathbf{y} \in \mathbb{R}^{n_y}$ is the output variables predicted by the model. $\mathbf{x} \in \mathbb{R}^{n_x}$ is representing the state variables. For simplicity, $\mathbf{y}$ can be expressed as a function of only $\mathbf{u}$ and $\mathbf{d}$. The solution $\mathbf{u}^*$ to the original optimization problem (2.1) can be obtained by solving to the following NLP problem:

$$\begin{aligned} \mathbf{u}^* = \arg\min_{\mathbf{u}} \quad & J(\mathbf{u}) := J(\mathbf{u}, \mathbf{y}(\mathbf{u}, \mathbf{d})) \\ \text{s.t.} \quad & \mathbf{G}(\mathbf{u}) := \mathbf{g}(\mathbf{u}, \mathbf{y}(\mathbf{u}, \mathbf{d})) \leq \mathbf{0} \\ & \mathbf{u}^L \leq \mathbf{u} \leq \mathbf{u}^U \end{aligned} \tag{2.3}$$

However, it is not guaranteed that the optimal solution to this problem, $\mathbf{u}^*$, coincide with the optimal plant value $\mathbf{u}_p^*$ [5]. In the presence of plant-model mismatch, it could be likely that $\mathbf{u}^*$ converge to a sub-optimal operating point, due to an inaccurate process model.

2.2.2 Necessary Conditions of Optimality

The Necessary Conditions of Optimality (NCO) has to be satisfied in order to obtain a feasible point upon convergence. The local minima of the optimization problem in (2.3) can be characterized by the Karush-Kuhn-Tucker (KKT) conditions. A solution $\mathbf{u}^*$ is a KKT point if there exist unique Langrange multipliers $\boldsymbol{\mu}^* \in \mathbb{R}^{n_g}$ such that the following holds:

$$\begin{aligned} \mathbf{G} \leq \mathbf{0}, \quad \boldsymbol{\mu}^{\boldsymbol{T}} \mathbf{G} = 0, \quad \boldsymbol{\mu} \geq \mathbf{0} \\ \frac{\partial \mathcal{L}}{\partial \mathbf{u}} = \frac{\partial J}{\partial \mathbf{u}} + \boldsymbol{\mu}^{\boldsymbol{T}} \frac{\partial \mathbf{G}}{\partial \mathbf{u}} = \mathbf{0} \end{aligned} \tag{2.4}$$

where $\mathcal{L}(\mathbf{u}, \boldsymbol{\mu}) := \mathrm{J}(\mathbf{u}) + \boldsymbol{\mu}^T\mathbf{G}(\mathbf{u})$ is the Lagrangian function. The KKT conditions alone do not sufficiently charaterize an optimum and satisfy the NCO. Two additional conditions must hold; The Linear Independence Constraint Qualification (LICQ) and a second order necessary condition. LICQ requires that the gradients of the active constraints are linearly independent at $\mathbf{u}^*$. The second order necessary conditions require that the reduced Hessian of the Lagrangian is positive semi-definite at $\mathbf{u}^*$ [4].

2.2.3 Standard MA Scheme

The standard MA scheme uses measurements of the plant constraints and estimates of plant gradients to modify the cost and constraint functions in the optimization problem. The modification is done in such a way that a KKT point for the model coincide with the optimum of the plant. At the k-th RTO iteration with input $\mathbf{u}_k$, the modified cost and constraint functions become:

$$J_{m,k}(\mathbf{u}) := J(\mathbf{u}) + \epsilon_k^J + (\boldsymbol{\lambda}_k^J)^T(\mathbf{u} - \mathbf{u}_k) \tag{2.5}$$

$$G_{m,i,k}(\mathbf{u}) := G_i(\mathbf{u}) + \varepsilon_k^{G_i} + (\boldsymbol{\lambda}_k^{G_i})^T(\mathbf{u} - \mathbf{u}_k) \leq 0, \quad i = 1, \ldots, n_g \tag{2.6}$$

with modifiers $\varepsilon_k^J \in \mathbb{R}$, $\varepsilon_k^{G_i} \in \mathbb{R}$, $\boldsymbol{\lambda}_k^J \in \mathbb{R}^{n_u}$ and $\boldsymbol{\lambda}_k^{G_i} \in \mathbb{R}^{n_u}$ given by:

$$\begin{aligned}
\varepsilon_k^J &= J_p(\mathbf{u}_k) - J(\mathbf{u}_k) \\
\varepsilon_k^{G_i} &= G_{p,i}(\mathbf{u}) - G_i(\mathbf{u}), \quad i = 1, \ldots, n_g \\
(\boldsymbol{\lambda}_k^J)^T &= \frac{\partial J_p}{\partial \mathbf{u}}(\mathbf{u}_k) - \frac{\partial J}{\partial \mathbf{u}}(\mathbf{u}_k) \\
(\boldsymbol{\lambda}_k^{G_i})^T &= \frac{\partial G_{p,i}}{\partial \mathbf{u}}(\mathbf{u}_k) - \frac{\partial G_i}{\partial \mathbf{u}}(\mathbf{u}_k), \quad i = 1, \ldots, n_g
\end{aligned} \tag{2.7}$$

where n_g is the number of constraints. The modifiers ε_k^J and $\varepsilon_k^{G_i}$ represent the differences between the plant values and the predicted values at the k-th RTO iteration. $\boldsymbol{\lambda}_k^J$ and $\boldsymbol{\lambda}_k^{G_i}$ represent the differences between the plant gradients and the model gradients at the k-th iteration. A graphical representation of the first-order modification of the constraint G_i at $\mathbf{u}_k$ is shown in Figure 2.3.

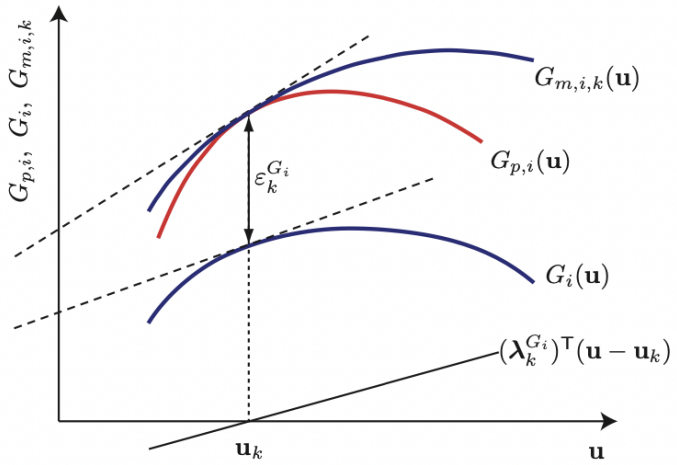


Figure 2.3: Graphical representation of the first-order modification of the constraint G_i at $\mathbf{u}_k$. Figure from [4].

The next optimal input $\mathbf{u}_{k+1}^*$ is found by solving the modified optimization problem:

$$\begin{aligned} \mathbf{u}_{k+1}^* = \arg\min_{\mathbf{u}} \quad & J_{m,k}(\mathbf{u}) := J(\mathbf{u}) + (\boldsymbol{\lambda}_k^J)^T\mathbf{u} \\ \text{s.t.} \quad & G_{m,i,k}(\mathbf{u}) := G_i(\mathbf{u}) + \varepsilon_k^{G_i} + (\boldsymbol{\lambda}_k^{G_i})^T(\mathbf{u}-\mathbf{u}_k) \leq \mathbf{0}, \quad i = 1,\dots,n_g \\ & \mathbf{u}^L \leq \mathbf{u} \leq \mathbf{u}^U \end{aligned} \tag{2.8}$$

The constant term $\varepsilon_k^J - (\boldsymbol{\lambda}_k^J)^T\mathbf{u}_k$ does not affect the solution and hence only the linear term in $\mathbf{u}$ is included in the objective function. Since the modifiers and the optimal inputs are sensitive to measurement noise, first-order filters are often applied to both the modifiers and the optimal inputs, as shown in Eqs. (2.9).

$$\begin{aligned} \varepsilon_{k+1}^{G_i} &= (1-b_i)\varepsilon_k^{G_i} + b_i(G_{p,i}(\mathbf{u}) - G_i(\mathbf{u})), \quad i = 1,\dots,n_g \\ (\boldsymbol{\lambda}_{k+1}^J)^T &= (1-d)(\boldsymbol{\lambda}_k^J)^T + d(\frac{\partial J_p}{\partial \mathbf{u}}(\mathbf{u}_k) - \frac{\partial J}{\partial \mathbf{u}}(\mathbf{u}_k)) \\ (\boldsymbol{\lambda}_{k+1}^{G_i})^T &= (1-q_i)(\boldsymbol{\lambda}_k^{G_i})^T + q_i(\frac{\partial G_{p,i}}{\partial \mathbf{u}}(\mathbf{u}_k) - \frac{\partial G_i}{\partial \mathbf{u}}(\mathbf{u}_k)), \quad i = 1,\dots,n_g \\ \mathbf{u}_{k+1} &= \mathbf{u}_k + K(\mathbf{u}_{k+1}^* - \mathbf{u}_k) \end{aligned} \tag{2.9}$$

The filter coefficients b_i, d and q_i have values (0,1] [5]. Small filter coefficient values give smaller iteration steps, since a higher ratio of the previous value is included in the updated one. The input filter matrix is a diagonal matrix given by $K = diag(k_1,\dots,k_{n_u})$, where the filter values k_i, $i = 1,\dots,n_u$, have values (0,1]. The input filter restricts the new input to move too far from the previous operating point.

2.2.4 MAy Scheme

Output Modifier Adaptation (MAy) is an alternative to the standard Modifier Adaptation method. Instead of modifying the cost and constraint functions, the outputs $\mathbf{y}$ are modified. At the k-th RTO iteration, the expression for the modified outputs are as follows:

$$\mathbf{y}_{m,k}(\mathbf{u}) := \mathbf{y}(\mathbf{u}) + \boldsymbol{\varepsilon}_k^{\mathbf{y}} + (\boldsymbol{\lambda}_k^{\mathbf{y}})^T(\mathbf{u} - \mathbf{u}_k) \tag{2.10}$$

with $\boldsymbol{\varepsilon}_k^{\mathbf{y}} \in \mathbb{R}^{n_y}$ and $\boldsymbol{\lambda}_k^{\mathbf{y}} \in \mathbb{R}^{n_u \times n_y}$ given by:

$$\begin{aligned} \boldsymbol{\varepsilon}_k^{\mathbf{y}} &= \mathbf{y}_p(\mathbf{u}_k) - \mathbf{y}(\mathbf{u}_k) \\ (\boldsymbol{\lambda}_k^{\mathbf{y}})^T &= \frac{\partial \mathbf{y}_p}{\partial \mathbf{u}}(\mathbf{u}_k) - \frac{\partial \mathbf{y}}{\partial \mathbf{u}}(\mathbf{u}_k) \end{aligned} \tag{2.11}$$

The next optimal inputs $\mathbf{u}_{k+1}^*$ are found by solving the following modified optimization problem:

$$\begin{aligned} \mathbf{u}_{k+1}^* = \arg\min_{\mathbf{u}} \quad & J(\mathbf{u}, \mathbf{y}_{m,k}(\mathbf{u})) \\ \text{s.t.} \quad & \mathbf{y}_{m,k}(\mathbf{u}) = \mathbf{y}(\mathbf{u}) + \boldsymbol{\varepsilon}_k^{\mathbf{y}} + (\boldsymbol{\lambda}_k^{\mathbf{y}})^T(\mathbf{u} - \mathbf{u}_k) \\ & G_i(\mathbf{u}, \mathbf{y}_{m,k}(\mathbf{u})) \leq 0 \quad i = 1, \ldots, n_g \\ & \mathbf{u}^L \leq \mathbf{u} \leq \mathbf{u}^U \end{aligned} \tag{2.12}$$

First-order filters are also applied to these modifiers and to the optimal inputs, as shown in Eqs. (2.9).

The gradient estimation step can be replaced by using Gaussian Processes to represent the plant-model mismatch in presence of noisy measurements. A general overview of Gaussian Processes will be presented in the next section to illustrate how it can be applied to represent the plant-model mismatch. Finally, the combined MAy and GP scheme is presented.

2.3 Gaussian Processes

Traditional non-linear regression methods, such as the Least Squares Estimator approach [13], provide parameters for a function that fit the observed data set best. This approach has an obvious problem in that it has to decide a class of functions. If the data is not well modeled by the function, then the predictions will be poor. Overfitting the function

to the data by increasing the flexibility (e.g. higher order functions) may lead to a good fit to the observed data, but perform bad on test predictions [14]. Moreover, there may be an infinite number of functions that fit the observed data points well. Gaussian Processes can be used for addressing this issue.

A fitted Gaussian Process (GP) defines a probability distribution over all possible functions that fit the data set [15]. This section will introduce the most important concepts of GPs. A more comprehensive introduction and mathematical derivations can be found in [14].

2.3.1 General Overview

A Gaussian Process is a non-parametric, Bayesian approach to regression and aims to describe unknown complex functions using very few variables. The Bayesian approach specifies a prior distribution, based on a priori knowledge, and updates this distribution based on observed data using Bayes' rule. The updated distribution is called the posterior distribution [16].

A GP can be defined as follows:

Definition 2.3.1 (Gaussian Process)**.** A Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution [14].

Consider an unknown function $f : \mathbb{R}^{n_u} \to \mathbb{R}$ that is of interest to be approximated by a GP. The function is dependent on the inputs $\mathbf{u} \in \mathbb{R}^{n_u}$, where n_u denotes the number of inputs. The GP prior is specified with a mean function m($\mathbf{u}$) and a covariance function k($\mathbf{u}$,$\mathbf{u}$'). Within the GP prior, prior knowledge about the space of functions can be incorporated [16]. The mean function and the covariance function are defined as:

$$\begin{aligned} m(\mathbf{u}) &= \mathbb{E}[f(\mathbf{u})] \\ k(\mathbf{u}, \mathbf{u}') &= \mathbb{E}[(f(\mathbf{u}) - m(\mathbf{u}))(f(\mathbf{u}') - m(\mathbf{u}'))] \end{aligned} \tag{2.13}$$

The GP can then be written as:

$$f(\mathbf{u}) \sim GP(m(\mathbf{u}), k(\mathbf{u}, \mathbf{u}')) \tag{2.14}$$

The specification of the mean function and the covariance function gives a prior distribution over functions. To illustrate this, n input points are chosen, which is given by $\overline{\mathbf{U}}_* = [\mathbf{u}_1 \ \mathbf{u}_2 \ \dots \ \mathbf{u}_n]^T \in \mathbb{R}^{n \times n_u}$. The inputs carry a subscript because they act as test inputs, and are not actually observed points. Then the corresponding covariance matrix

$\mathbf{K}_* \in \mathbb{R}^{n \times n}$ is made using the chosen covariance function elementwise. The covariance matrix is given by:

$$\mathbf{K}_* = \begin{bmatrix} k(\mathbf{u}_1, \mathbf{u}_1) & k(\mathbf{u}_1, \mathbf{u}_2) & \dots & k(\mathbf{u}_1, \mathbf{u}_n) \\ k(\mathbf{u}_2, \mathbf{u}_1) & k(\mathbf{u}_2, \mathbf{u}_2) & \dots & k(\mathbf{u}_2, \mathbf{u}_n) \\ \vdots & \vdots & \vdots & \vdots \\ k(\mathbf{u}_n, \mathbf{u}_1) & k(\mathbf{u}_n, \mathbf{u}_2) & \dots & k(\mathbf{u}_n, \mathbf{u}_n) \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} & \dots & k_{1n} \\ k_{21} & k_{22} & \dots & k_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ k_{n1} & k_{n2} & \dots & k_{nn} \end{bmatrix} \tag{2.15}$$

where the diagonal elements represent the self covariance of the random variables, which is equal to the variance [17]. A random Gaussian vector $\mathbf{f}_*$, consisting of random generated values, with this covariance matrix can be generated and written as:

$$\mathbf{f}_* \sim N(\mathbf{0}, \mathbf{K}_*) \tag{2.16}$$

where a zero mean function is chosen for simplicity. See [14] for a full introduction to multivariate Gaussian distributions and how to generate random samples.

The generated values can then be plotted as a function of the inputs. The left plot in Figure 2.4 shows three such random samples, where the squared exponential covariance function was used. This covariance function will be further described in the following section. The right plot in Figure 2.4 shows three random functions drawn from the GP posterior distribution. The posterior functions are constructed from observed data points. The figure illustrate how functions that disagree with the observations are rejected. The shaded region in both plots shows a 95% confidence region. This means that the drawn distributions will fall inside this region 95% of the time.

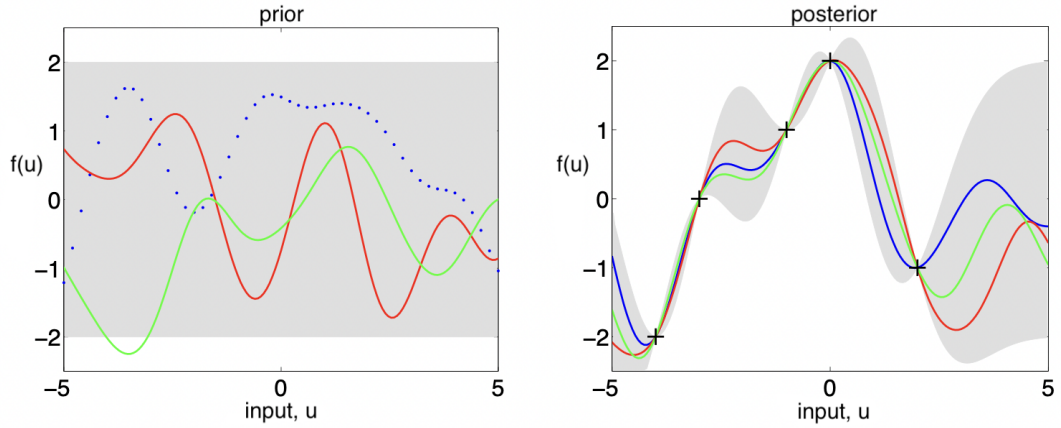


Figure 2.4: Three functions drawn at random from a GP prior distribution and three random functions drawn from the GP posterior after 5 datapoints have been observed. The shaded region in both plots is showing a 95% confidence region. Figure from [14].

2.3.2 Covariance Functions

The properties of a GP is highly dependent on the covariance function, as it includes the assumptions about the function of interest. This function defines the similarity between outputs. A basic assumption is that points with close input values will have similar outputs [14]. The covariance function is also often referred to as the kernel function, and is formulated as:

$$\text{cov}(f(\mathbf{u}_p), f(\mathbf{u}_q)) = k(\mathbf{u}_p, \mathbf{u}_q) \tag{2.17}$$

Note that the covariance between the outputs is a function of the inputs. A valid covariance function must give a positive definite covariance matrix, which means that it has to be symmetric and invertible. One well known and widely used covariance function is the squared exponential covariance function. This function is defined as:

$$k_{SE}(\mathbf{u}_p, \mathbf{u}_q) = \sigma_f^2 \text{exp}(-\frac{||\mathbf{u}_p - \mathbf{u}_q||^2}{2l^2}) \tag{2.18}$$

where σ_f and l are the hyperparameters of the covariance function. The output variance σ_f determines the average distance of the function away from its mean and is just a scale factor [18]. The factor $\text{exp}(-\frac{||\mathbf{u}_p - \mathbf{u}_q||^2}{2l^2})$ has a maximum value of 1 and a minimum value of 0. The scale factor hence allows a different output range. l is defined as the characteristic length scale and can be thought of as the distance one can move in input space before the function value would change significantly [14]. Given a set of observations, the hyperparameters are learned by maximizing the log-marginal likelihood [6].

Realistic modelling situations most often contain noisy measurements and not direct access to the true function values. It is assumed that the noisy measurements are given by:

$$y = f(\mathbf{u}) + \nu \tag{2.19}$$

where the measurement noise $\nu \sim \mathcal{N}(0, \sigma_n^2)$ is independent and identically distributed. The noise follows a Gaussian distribution with zero mean and variance σ_n^2. In these cases the covariance function must be combined with a term which accounts for the noise. This allows the GP to not pass through every data point due to noise in the outputs. The covariance between the outputs can now be written as in Eq. (2.20), where δ_{pq} is a Kronecker delta which is one if $p = q$ and zero otherwise [14]. This is because the noise is independent and not correlated between different outputs.

$$\text{cov}(y_p, y_q) = k(\mathbf{u}_p, \mathbf{u}_q) + \sigma_n^2 \delta_{pq} \tag{2.20}$$

The covariance can also be written in matrix form, as in Eq. 2.21. The $\sigma_n^2 I$ term is in this case a diagonal matrix with the variance σ_n^2 of each output on the diagonal.

$$\text{cov}(\mathbf{y}) = \mathbf{K} + \sigma_n^2 I \tag{2.21}$$

2.3.3 Prediction using Noisy Observations

Consider the unknown function f which is of interest to be predicted by GP regression. The most interesting part is not to draw random functions from the prior, but rather the incorporation of observed data. The observed data is often called training data and provides important knowledge about the function of interest. The observations in this case are noisy and are given by Eq. 2.19. Given a data set of n such observations $\mathcal{D} = \{(\mathbf{u}_i, y_i) | i = 1, \ldots, n\}$, where the input vector is given as $\overline{\mathbf{U}} = [\mathbf{u}_1 \ \mathbf{u}_2 \ \ldots \ \mathbf{u}_n]^T \in \mathbb{R}^{n \times n_u}$ and the output vector as $\overline{\mathbf{y}} = [y_1 \ y_2 \ \ldots \ y_n]^T \in \mathbb{R}^n$. It is preferable to make predictions for a new single input $\mathbf{u}_*$ (test input) that is not already observed.

The joint distribution of the observed outputs $\overline{\mathbf{y}}$ and the function value f_* at the test input $\mathbf{u}_*$ according to the prior is:

$$\begin{bmatrix} \overline{\mathbf{y}} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} + \sigma_n^2 I & \mathbf{k}_*) \\ \mathbf{k}_*^T & k_{**} \end{bmatrix} \right) \tag{2.22}$$

where f_* is a scalar. $\mathbf{K} \in \mathbb{R}^{n \times n}$ denotes the matrix of the covariances evaluated at all pairs of observed inputs. k_{**} denotes the self covariance evaluated at the test input, given by $k_{**} = k(\mathbf{u}_*, \mathbf{u}_*)$. $\mathbf{k}_* \in \mathbb{R}^n$ denotes the covariances evaluated at all the observed inputs combined with the test input, given by $\mathbf{k}_* = [k(\mathbf{u}_1, \mathbf{u}_*) \ k(\mathbf{u}_2, \mathbf{u}_*) \ \ldots \ k(\mathbf{u}_n, \mathbf{u}_*)]^T$ [17].

The posterior distribution over functions restricts the joint prior distribution to only contain functions that agree with the observed data set, as illustrated in Figure 2.4. This is done by conditioning the joint Gaussian prior on the observations. For further details, see [14]. The conditioned GP, i.e. the GP posterior, is given by:

$$f_* | \overline{\mathbf{U}}, \overline{\mathbf{y}}, \mathbf{u}_* \sim \mathcal{N}(\bar{f}_*, \mathbb{V}[f_*]) \tag{2.23}$$

with conditioned mean $\bar{f}_*$ and variance $\mathbb{V}[f_*]$ given by:

$$\bar{f}_* = \mathbf{k}_*^T [\mathbf{K} + \sigma_n^2 I] \overline{\mathbf{y}} \tag{2.24}$$

$$\mathbb{V}[f_*] = k_{**} - \mathbf{k}_*^T[\mathbf{K} + \sigma_n^2 I]^{-1}\mathbf{k}_* \tag{2.25}$$

To simplify the notation, a conditioned GP predicting f_* at input $\mathbf{u}_*$, given training data $(\overline{\mathbf{U}}, \overline{\mathbf{y}})$ is shown in Eq. (2.26). This equation will be used throughout the thesis.

$$f_* = (GP)^f(\mathbf{u}_*|\overline{\mathbf{U}}, \overline{\mathbf{y}}) \tag{2.26}$$

2.3.4 MAy Combined with Gaussian Processes

In the proposed MAy-GP scheme in this thesis, GPs are used to represent the plant-model mismatch in the system outputs. The GPs will then be used to calculate alternative modifiers to the ones seen in Section 2.2.4. The functions $\mathbf{f}$ that describe the plant-model mismatch and are of interest to be fitted by GPs are given as:

$$\mathbf{f} = [f_1 \ f_2 \ \dots \ f_{n_y}]^T \tag{2.27}$$

with $f_i = (y_{p,i} - y_i)$. The predicted function value of each function are then given by Eq. (2.26). The GPs need to be trained at every iteration, since the observed data $(\overline{\mathbf{U}}, \overline{\mathbf{y}})$ includes more data points after every iteration. Note that the observed outputs $\overline{\mathbf{y}}$ in this data set are the observed mismatch. The optimal input $\mathbf{u}_{k+1}^*$ is found by solving the following optimization problem:

$$\begin{aligned} \mathbf{u}_{k+1}^* = \arg\min_{\mathbf{u}} \quad & J(\mathbf{u}, \mathbf{y}_{m,k}(\mathbf{u})) \\ \text{s.t.} \quad & \mathbf{y}_{m,k}(\mathbf{u}) = \mathbf{y}(\mathbf{u}) + \boldsymbol{\varepsilon}_k^{\mathbf{y}_{GP}} + (\boldsymbol{\lambda}_k^{\mathbf{y}_{GP}})^T(\mathbf{u} - \mathbf{u}_k) \\ & G_i(\mathbf{u}, \mathbf{y}_{m,k}(\mathbf{u})) \leq 0 \quad i = 1, \dots, n_g \\ & \mathbf{u}^L \leq \mathbf{u} \leq \mathbf{u}^U \end{aligned} \tag{2.28}$$

with the GP modifiers $\boldsymbol{\varepsilon}_k^{\mathbf{y}_{GP}} \in \mathbb{R}^{n_y}$ and $\boldsymbol{\lambda}_k^{\mathbf{y}_{GP}} \in \mathbb{R}^{n_u \times n_y}$ given by:

$$\boldsymbol{\varepsilon}_k^{\mathbf{y}_{GP}} = (GP)^{\mathbf{f}}(\mathbf{u}_k|\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k) = \begin{bmatrix} (GP)^{f_1}(\mathbf{u}_k|\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k) \\ (GP)^{f_2}(\mathbf{u}_k|\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k) \\ \vdots \\ (GP)^{f_{n_y}}(\mathbf{u}_k|\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k) \end{bmatrix} \tag{2.29}$$

$$(\boldsymbol{\lambda}_k^{\mathbf{y}_{GP}})^T = \frac{\partial (GP)^{\mathbf{f}}(\mathbf{u}_k|\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)}{\partial \mathbf{u}_k} = \begin{bmatrix} \frac{\partial (GP)^{f_1}(\mathbf{u}_k|\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)}{\partial \mathbf{u}_k} \\ \frac{\partial (GP)^{f_2}(\mathbf{u}_k|\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)}{\partial \mathbf{u}_k} \\ \vdots \\ \frac{\partial (GP)^{f_{n_y}}(\mathbf{u}_k|\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)}{\partial \mathbf{u}_k} \end{bmatrix} \tag{2.30}$$

where $(\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)$ is the observed data set at the k-th iteration.

The optimal inputs are filtered with a first-order filter, like in the previous sections:

$$\mathbf{u}_{k+1} = \mathbf{u}_k + K(\mathbf{u}_{k+1}^* - \mathbf{u}_k) \tag{2.31}$$

with $K = diag(k_1, \ldots, k_{n_u})$, where the filter values k_i, $i = 1, \ldots, n_u$, have values (0,1]. The filter is useful because it keeps the predictions closer to the region where the GPs have observed data points. Filters on the modifiers should not be necessary since the GPs handle noise internally.

Chapter 3

System Description

This chapter contains an overall discription of the system of interest in this thesis. It includes a general overview of subsea oil well networks, a description of the lab rig setup, a presentation of the optimization problem and the process model and finally, a description of the dynamic model used in the MATLAB simulation.

3.1 Subsea Oil Well Networks

The overall goal in subsea oil production is to maximize the production of oil from the reservoirs. Typically, the reservoir pressure is what drives the fluids from below the seafloor to the top facilities trough risers. Artificial lifting methods, such as a gas lift, can be applied to the system if this pressure is not large enough. A simplified figure of the system of interest is shown in Figure 3.1, where a gas lift is applied to the system. The idea of gas lift consists of injecting excess gas that is produced at the bottom of a well. The injected gas reduces the bulk density of the system and decreases the hydrostatic pressure on the reservoir. Since the reservoirs typically are located several kilometers below sea level, such a decrease on the hydrostatic pressure has a significant effect on the production rate. However, if the gas injection rate becomes too large, the frictional pressure drop effect dominates and the injected gas will have a negative effect [19]. The gain of the oil production by injecting gas is also dependent on the outflow from the reservoir, which can vary.

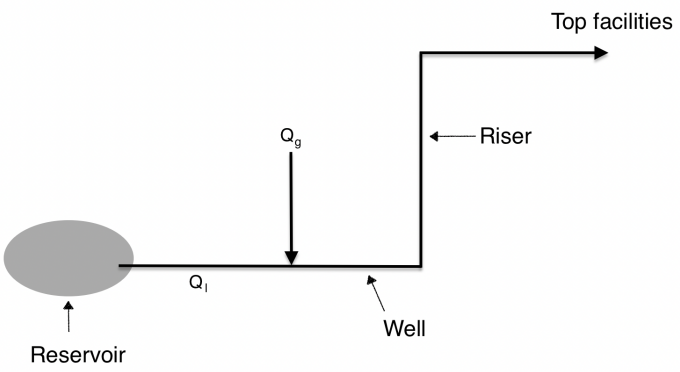


Figure 3.1: Simple model diagram of one gas lifted well. Q_g is the gas lift rate and Q_l is the oil production rate.

3.2 Experimental Lab Rig Setup

The experimental rig represents a subsea oil well network with three parallel gas lifted wells. The setup in the rig uses water and air as working fluids instead of oil and gas. This is a simplification of the system, but the gas lift phenomenon is not influenced. A simplified flowsheet of the rig is shown in Figure 3.2.

The reservoir consists of a tank, a pump and three control valves. The different valve openings represent different behaviours of the reservoir and can be manipulated. The pump outlet pressure PI104 represents the reservoir pressure and the pump rotation is kept constant by a PI controller. The three wells consist of three parallel hoses of 1.5 m with an inner diameter of 2 cm. Air is injected approximately 10cm after the valve openings of the reservoir. Three PI controllers are used for controlling the gas flowrates, whose setpoints are the system inputs. The risers are consisting of three 2.2 m high hoses with and inner diameter of 2 cm, like the well hoses. The top facilities are represented by a separation tank.

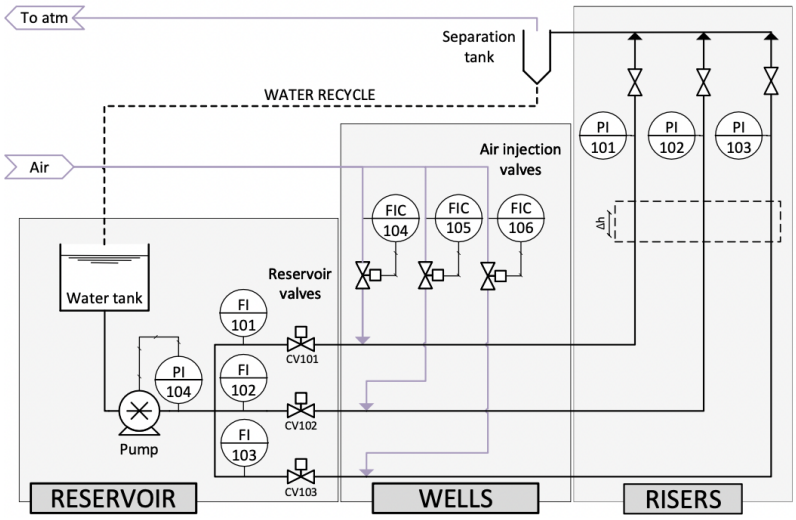


Figure 3.2: Simple flowsheet of the experimental rig. Figure from [19].

The system measurements are the well top pressure (PI101, PI102 and PI103), the pump outlet pressure (PI104), the liquid flowrates (FI101, FI102, FI103) and the gas flowrates (FI104, FI105, FI106). The reservoir valve openings (CV101, CV102, CV103) are known disturbances in the experiments. These valve openings will be referred to as v_{o_1}, v_{o_2} and v_{o_3} respectively throughout the thesis.

3.3 The Optimization Problem

The objective is to maximize the economic profit J by determine the optimal gas injection rate (the setpoints of flow controllers FIC104, FIC105 and FIC106). J is a function of the liquid flowrates. The inputs $\mathbf{u}$ and the outputs $\mathbf{y}$ are:

$$\mathbf{u} = [Q_{g_1}\ Q_{g_2}\ Q_{g_3}]^T$$
$$\mathbf{y} = [Q_{l_1}\ Q_{l_2}\ Q_{l_3}]^T$$

where Q_{l_1}, Q_{l_2} and Q_{l_3} are the liquid flowrates FI101, FI102 and FI103 respectively. Q_{g_1}, Q_{g_2} and Q_{g_3} are the setpoints to the flow controllers FIC104, FIC105 and FIC106 respectively.

The three parallel wells have different priority in the objective function J due to economic reasons. This is indicated by using different weights. A gas availability constraint and gas injection bounds have to be respected. Maximum gas availability is 7.5 sL/min and the upper and lower bounds of each gas injection rate is 1 and 5 sL/min respectively. The optimization problem becomes:

$$\begin{aligned} \max_{Q_{g_1}, Q_{g_2}, Q_{g_3}} \quad & J = 20Q_{l_1} + 10Q_{l_2} + 30Q_{l_3} \\ \text{s.t.} \quad & Q_{g_1} + Q_{g_2} + Q_{g_3} \leq 7.5, \\ & 1 \leq Q_{g_1}, Q_{g_2}, Q_{g_3} \leq 5 \end{aligned} \tag{3.1}$$

3.4 Model Equations

Instead of relying on physical knowledge to obtain the steady-state process model, it is chosen to represent the process by using a simple mathematical expression for describing the input-output relationship in each well. The three wells in the experimental rig are independent, which means that one well is only affected by one gas injection rate and one valve opening. It is chosen to use one polynomial per well, instead of more complex model structures. Polynomials are linear in the parameters, and can represent the first and second order effects of the inputs on the outputs. Thus, this model structure is also

referred to as simple. The idea of using a simplified model of the process is to study if it can be used in the optimization and still give good results, even though it is simple.

Different steady-state models of each well in the experimental rig were obtained from an experiment, described in the project work presented in [11]. the Least Squares Estimator approach was used to fit the models to the data. Hence, all the obtained models were linear with respect to the parameters. Statistical methods were then used for evaluation, and one of the model structures turned out to be preferred over the other ones. The chosen model structure to be implemented in the RTO layer is:

$$Q_l = \theta_1 + \theta_2 Q_g + \theta_3 v_o + \theta_4 Q_g^2 + \theta_5 v_o^2 \tag{3.2}$$

where Q_l is the liquid flow rate, Q_g is the gas flowrate and v_o is the valve opening. The above model is shown for one well, so there are in total three such relationships:

$$Q_{l_i} = \theta_{1_i} + \theta_{2_i} Q_{g_i} + \theta_{3_i} v_{o_i} + \theta_{4_i} Q_{g_i}^2 + \theta_{5_i} v_{o_i}^2, \quad i = 1, 2, 3 \tag{3.3}$$

The parameters θ_{1_i}, θ_{2_i}, θ_{3_i}, θ_{4_i} and θ_{5_i} for $i = 1, 2, 3$ are constant and their numerical values are shown in Table 3.1.

Table 3.1: Estimated model parameters.

$\boldsymbol{\theta}$	**Well 1**	**Well 2**	**Well 3**
θ_1	0.493	-0.274	-0.883
θ_2	0.300	0.459	0.620
θ_3	19.312	20.102	21.313
θ_4	-0.015	-0.037	-0.064
θ_5	-14.338	-14.639	-15.714

3.5 Experimental Rig Simulation Description[1]

Before testing the optimization method in the actual experimental lab rig, some simulation was done using a rigorous first-principles model as the plant, while the simple model of Section 3.4 is used in the RTO layer for economic optimization purposes.

The model (plant in the simulation) of the three gas lifted wells is obtained from [20]. The model has been implemented in MATLAB to test and tune different optimization methods before they have been applied to the experimental rig. A model diagram is shown in Figure 3.3. The model uses mass balances of the different phases, density

[1]Section taken from the project work presented in [11] by the same author.

models, pressure models and flow models, which become the following differential algebraic equation (DAE):

$$\begin{aligned} \dot{\mathbf{x}}_i &= \mathbf{f}_i(\mathbf{x}_i, \mathbf{z}_i, \mathbf{u}_i, \mathbf{p}_i) \\ \mathbf{g}_i(\mathbf{x}_i, \mathbf{z}_i, \mathbf{u}_i, \mathbf{p}_i) &= \mathbf{0} \quad \forall i \in \mathcal{N} = \{1, \ldots, n_w\} \end{aligned} \tag{3.4}$$

where the subscript i is referring to a well in the set $\mathcal{N}$. $\mathbf{x}_i$ is the differential states, $\mathbf{z}_i$ is the algebraic states, $\mathbf{u}_i$ is the decision variables and $\mathbf{p}_i$ is the uncertain variables. n_w is the number of wells, i.e. $n_w = 3$ in this case. $\mathbf{f}_i$ is the set of differential equations and $\mathbf{g}_i$ is the set of algebraic equations. The MATLAB code with these equations is shown in Appendix B.

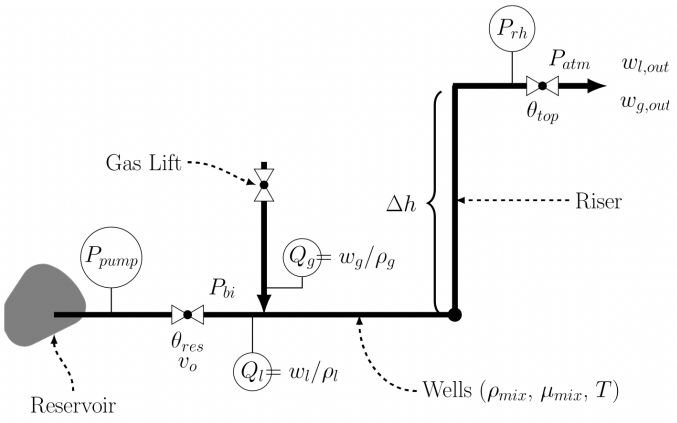


Figure 3.3: Model diagram of one single well. Figure from [19].

The differential states, the algebraic states and the decision variables are given by:

$$\begin{aligned} \mathbf{x}_i &= [m_{g_i} \; m_{l_i}]^T \\ \mathbf{z}_i &= [w_{l_i} \; w_{total_i} \; p_{rh_i} \; p_{bi_i} \; \rho_{mix} \; \rho_{g_i} \; w_{gout_i} \; w_{lout_i}]^T \\ \mathbf{u}_i &= [Q_{g_i} \; v_{o_i} \; p_{pump}]^T \end{aligned} \tag{3.5}$$

where m_{g_i} is the gas hold up and m_{l_i} is the liquid hold up. w_{l_i} is the water rate from the reservoir and w_{total_i} is the total well production rate. p_{rh_i} is the riser head pressure and p_{bi_i} is the pressure before injection point. ρ_{mix} is the mixture density in the system and ρ_{g_i} is the density of the gas. w_{gout_i} and w_{lout_i} is the well outlet gas and liquid outlet flowrate respectively. Q_{g_i} is the gas lift injection rate, v_{o_i} is the valve opening from the reservoir and p_{pump} is the reservoir pressure.

Note that the valve opening v_o is included as a decision variable in the model, but its seen as a (measured) disturbance in the simulation. p_{pump} is held at a constant value in the simulation.

Chapter 4

Methodology

This chapter presents the building blocks for the implementation of Output Modifier Adaptation combined with Gaussian Processes (MAy-GP) in the experimental rig. The model used for optimization purposes is the steady-state model described in Section 3.4 and given by Eq. (3.3). The optimization problem is Problem (3.1). The inputs $\mathbf{u}$ and the outputs $\mathbf{y}$ are:

$$\mathbf{u} = [Q_{g_1}\ Q_{g_2}\ Q_{g_3}]^T$$
$$\mathbf{y} = [Q_{l_1}\ Q_{l_2}\ Q_{l_3}]^T$$

4.1 The MAy-GP Algorithm

A block diagram of MAy-GP is shown in Figure 4.1. In order to optimize the system, GPs representing the mismatch between plant outputs and model outputs have to be trained. The mismatch is given as functions $\mathbf{f}$:

$$\mathbf{f} = [(Q^p_{l_1} - Q_{l_1})\ (Q^p_{l_2} - Q_{l_2})\ (Q^p_{l_3} - Q_{l_3})]^T \tag{4.1}$$

where the notation p is used to represent the plant. The GPs are updated at every iteration, since more data points are added to the observed data sets. There are three independent data sets in this case, which is written as $(\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)_i$, where $i = 1, 2, 3$ represents the well number and k represents the MAy-GP iteration number. This is because the wells are independent, which means that each of the three wells are only depending on one gas injection rate and one valve opening. Each observed data point in these data sets can be written as $(\mathbf{U}_k,\ (Q^p_l - Q_l)_k)_i$. $(Q^p_l - Q_l)_k \in \mathbb{R}$ is the observed mismatch at the k-th iteration. $\mathbf{U}_k \in \mathbb{R}^2$ consists of both the gas injection rate Q_{g_i} and the valve opening v_{o_i} at the k-th iteration. The valve openings are seen as disturbances

and not as inputs, but since they are measured they are introduced as predictors in the GP fitting. The input vector to the GPs is therefore on the form $\overline{\mathbf{U}}_k = [\mathbf{U}_0\ \mathbf{U}_1\ \ldots\ \mathbf{U}_{k-1}\ \mathbf{U}_k]^T$, and the observed mismatch given to the GPs is on the form $\overline{\mathbf{y}}_k = [(Q_l^p - Q_l)_0\ (Q_l^p - Q_l)_1\ \ldots\ (Q_l^p - Q_l)_{k-1}\ (Q_l^p - Q_l)_k]^T$. The squared exponential covariance function and the zero mean function are used in the prediction of GPs.

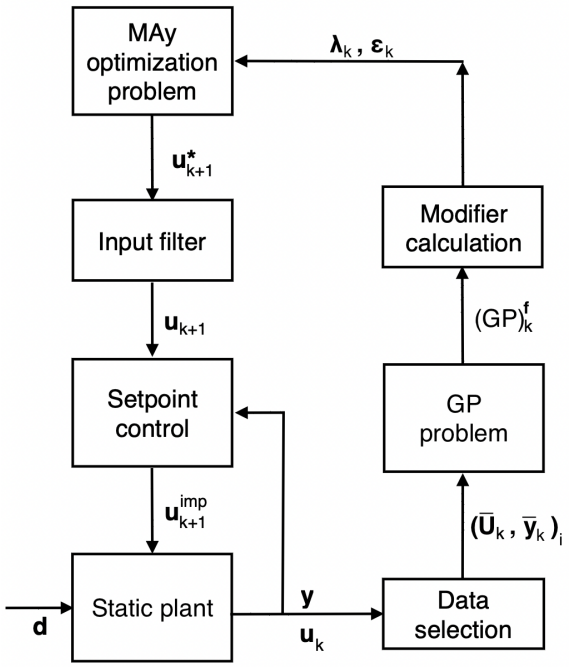


Figure 4.1: MAy-GP block diagram, with $\mathbf{u} = [Q_{g_1}\ Q_{g_2}\ Q_{g_3}]^T$, $\mathbf{y} = [Q_{l_1}\ Q_{l_2}\ Q_{l_3}]^T$ and $\mathbf{d} = [v_{o_1}\ v_{o_2}\ v_{o_3}]^T$. The i notation is associated with well number (1-3).

It is required that the system reaches steady-state before the selection of new observed data. Based on previous knowledge, a sampling time of 60 seconds is enough to guarantee steady-state of the system. Measurements are given every second, and the observed plant outputs is an average of the last 10 measurements to filter some of the noise. The trained GPs will then be used in the modifier calculation. The modifiers $\boldsymbol{\varepsilon}_k^{\mathbf{y}_{GP}} \in \mathbb{R}^3$ and $\boldsymbol{\lambda}_k^{\mathbf{y}_{GP}} \in \mathbb{R}^{3\times 3}$ are calculated as follows:

$$\boldsymbol{\varepsilon}_k^{\mathbf{y}_{GP}} = \begin{bmatrix} (GP)^{(Q_{l_1}^p - Q_{l_1})}(\mathbf{u}_k|(\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)_1) \\ (GP)^{(Q_{l_2}^p - Q_{l_2})}(\mathbf{u}_k|(\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)_2) \\ (GP)^{(Q_{l_3}^p - Q_{l_3})}(\mathbf{u}_k|(\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)_3) \end{bmatrix} = \begin{bmatrix} (\varepsilon_k^{\mathbf{y}_{GP}})_1 \\ (\varepsilon_k^{\mathbf{y}_{GP}})_2 \\ (\varepsilon_k^{\mathbf{y}_{GP}})_3 \end{bmatrix} \tag{4.2}$$

$$(\boldsymbol{\lambda}_k^{\mathbf{y}_{GP}})^T = \begin{bmatrix} \frac{\partial (GP)^{(Q_{l_1}^p - Q_{l_1})}(\mathbf{u}_k|(\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)_1)}{\partial \mathbf{u}_k} \\ \frac{\partial (GP)^{(Q_{l_2}^p - Q_{l_2})}(\mathbf{u}_k|(\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)_2)}{\partial \mathbf{u}_k} \\ \frac{\partial (GP)^{(Q_{l_3}^p - Q_{l_3})}(\mathbf{u}_k|(\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)_3)}{\partial \mathbf{u}_k} \end{bmatrix} = \begin{bmatrix} (\lambda_k^{\mathbf{y}_{GP}})_{11} & 0 & 0 \\ 0 & (\lambda_k^{\mathbf{y}_{GP}})_{22} & 0 \\ 0 & 0 & (\lambda_k^{\mathbf{y}_{GP}})_{33} \end{bmatrix} \tag{4.3}$$

where the gradients are estimated by FDA with step length set to $h = 10^{-8}$. Since the three wells are independent, $\boldsymbol{\lambda}_k^{\mathbf{y}_{GP}}$ becomes a diagonal matrix.

Inserting $\mathbf{u} = [Q_{g_1}\ Q_{g_2}\ Q_{g_3}]^T$, $\mathbf{y} = [Q_{l_1}\ Q_{l_2}\ Q_{l_3}]^T$ and the simple steady-state process model given in Eq. (3.3), the modified economic optimization problem becomes:

$$\begin{aligned} \max_{Q_{g_1}, Q_{g_2}, Q_{g_3}} \quad & J = 20Q_{l_{1,m,k}} + 10Q_{l_{2,m,k}} + 30Q_{l_{3,m,k}} \\ \text{s.t.} \quad & Q_{l_{i,m,k}} = \theta_{1_i} + \theta_{2_i} Q_{g_i} + \theta_{3_i} v_{o_{i,k}} + \theta_{4_i} Q_{g_i}^2 + \theta_{5_i} v_{o_{i,k}}^2 + (\varepsilon_k^{\mathbf{y}_{GP}})_i \\ & \qquad + (\lambda_k^{\mathbf{y}_{GP}})_{ii} (Q_{g_i} - Q_{g_{i,k}}), \quad i = 1, 2, 3 \\ & Q_{g_1} + Q_{g_2} + Q_{g_3} \leq 7.5 \\ & 1 \leq Q_{g_1}, Q_{g_2}, Q_{g_3} \leq 5 \end{aligned} \tag{4.4}$$

The optimal inputs are filtered with a first-order filter, as shown in Eq. (2.31) before they are implemented by the algorithm. The input filter values are all set to $k_i = 0.55$. The implemented inputs by MAy-GP are setpoints to PI controllers used for controlling the gas flowrates.

Algorithm 1 summarizes the main steps in the MAy-GP algorithm. It also includes the *initialization* step, which is needed for initial training of the GPs. In this step, an initial $\mathbf{u}_0$ is set. Then, two additional input sequences around $\mathbf{u}_0$ are implemented to give the GPs an initial data set consisting of 3 data points. $\mathbf{u}_0 = [2,\ 2.5,\ 3]^T$ is chosen as the initial point and the next input sequences for generating initial observed data sets are $[3,\ 2,\ 2.5]^T$ and $[2.5,\ 3,\ 2]^T$.

Algorithm 1 MAy-GP

Initialize:

Set initial $\mathbf{u}_0$ and input filter K. Set initial modifiers $\boldsymbol{\varepsilon}_0^{\mathbf{y}_{GP}}$ and $\boldsymbol{\lambda}_0^{\mathbf{y}_{GP}}$ to zero. Set two additional input sequences around $\mathbf{u}_0$ to obtain initial training dataset. Train GPs with initial dataset and optimize hyperparameters.

for $k = 1 \rightarrow \infty$ **do**

1. Get steady-state measurements from the plant and model at $\mathbf{u}_k$. These measurements include the outputs $\mathbf{y}$ and the valve opening v_o of each well.
2. Update the observed data set $(\overline{\mathbf{U}}_k, \overline{\mathbf{y}}_k)_i$ for each GP and optimize hyperparameters.
3. Predict the plant-model mismatch with the fitted GPs at $\mathbf{u}_k$ and update the modifier $\boldsymbol{\varepsilon}_k^{\mathbf{y}_{GP}}$.
4. Calculate the modifier $\boldsymbol{\lambda}_k^{\mathbf{y}_{GP}}$ with FDA.
5. Solve modified optimization problem to obtain $\mathbf{u}_{k+1}^*$.
6. Apply filter on the optimal inputs to obtain $\mathbf{u}_{k+1}$ and implement as setpoints for the controllers.

end for

4.2 Programming Environment

For implementation of the MAy-GP scheme, MATLAB version 9.9 (R2020b) is used. CasADi version 3.5.5, which is an open-source tool for nonlinear optimization and algorithmic differentiation [21] is used for solving the NLP. The IPOPT solver is chosen as the optimization method.

For the Gaussian Processes, the GPML Toolbox version 4.2 is used. This toolbox is an implementation of inference and prediction in GPs [22] and is based on algorithms presented in [14].

Chapter 5

Results and Discussion

The MAy-GP algorithm is first implemented in a simulation of the experimental rig to study the performance of the algorithm before implementing in the actual lab rig. The plant in the simulation is the dynamic model described in Section 3.5. The algorithm is then implemented in the actual lab rig. This chapter presents and discuss the results from the simulation in addition to the experimental results from the experimental lab rig.

5.1 Simulation Case Studies

In this section, two different simulation case studies are presented. It is chosen to keep the valve openings at a constant value in the simulations and focus on the influence of noise in the system. The first case study is noise free, while noise is introduced in the second to match the noise in the experimental rig. The measurement noise in the simulation is drawn from a Gaussian distribution with zero mean and variance σ^2, which was computed to represent the actual noise level in the experimental rig. The second case study also includes "controller action" for the PI controllers, which is implemented as a 5 seconds delay to the inputs.

The global optimum for the plant in the simulations is $\mathbf{u} = [2.59, 1.00, 3.91]^T$. The model optimum is $\mathbf{u} = [2.78, 1.00, 3.72]^T$, which means that a plant-model mismatch is present. The plant and the model optimum is found by optimization of the plant and model separately in Problem 3.1.

In the following sections, the GPs have been given numbers which refers to the well number. That is, $(\text{GP})^{(Q^p_{l_1} - Q_{l_1})}$ is referred to as GP1, $(\text{GP})^{(Q^p_{l_2} - Q_{l_2})}$ is referred to as GP2 and $(\text{GP})^{(Q^p_{l_3} - Q_{l_3})}$ is referred to as GP3.

5.1.1 Simulation without Noise

Figure 5.1 shows the simulation result without measurement noise in the system. The left plots show the valve openings, the output measurements and the instantaneous profit (objective function). The valve opening of each well are held at a constant value during the simulation. The upper right plot is showing the inputs, as well as the pre-calculated optimal inputs. The inputs start at a sub-optimal operating point and the MAy-GP algorithm begins the initialization step at time 2 minutes. At time 2 and 3 minutes the inputs hence change in order to obtain initial observed data, as described in the pseudo code shown in Algorithm 1. The first optimization takes place after 4 minutes. After approximately 14 number of iterations, the inputs converge to the optimum. As seen from the bottom right plot, the gas availability constraint is respected and the total input usage is held at 7.5 sL/min.

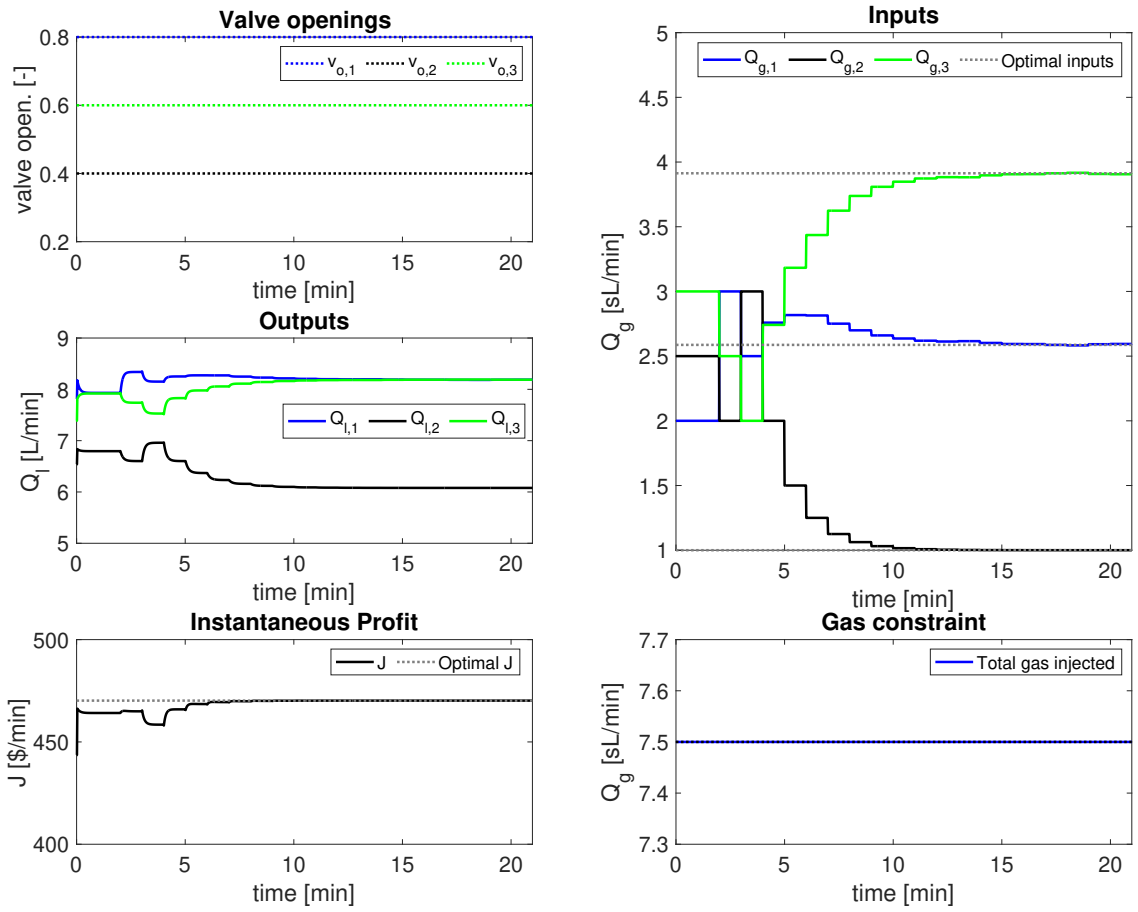


Figure 5.1: Simulation result without noise. The figure shows the valve openings, the outputs, the instantaneous profit (objective function), the inputs and the total gas injected.

The inputs converge to the plant optimum after approximately 14 minutes. However, when examining the objective function it is clear that the system is already close to the

optimum at time 5 minutes. This means that the objective function is not sensitive to changes in the inputs around the optimum.

Table 5.1: GP hyperparameters in the simulation without noise.

	GP1	**GP2**	**GP3**
l	1.97	1.61	1.54
σ_f	0.592	0.570	0.379
σ_n	0.000	0.000	0.000

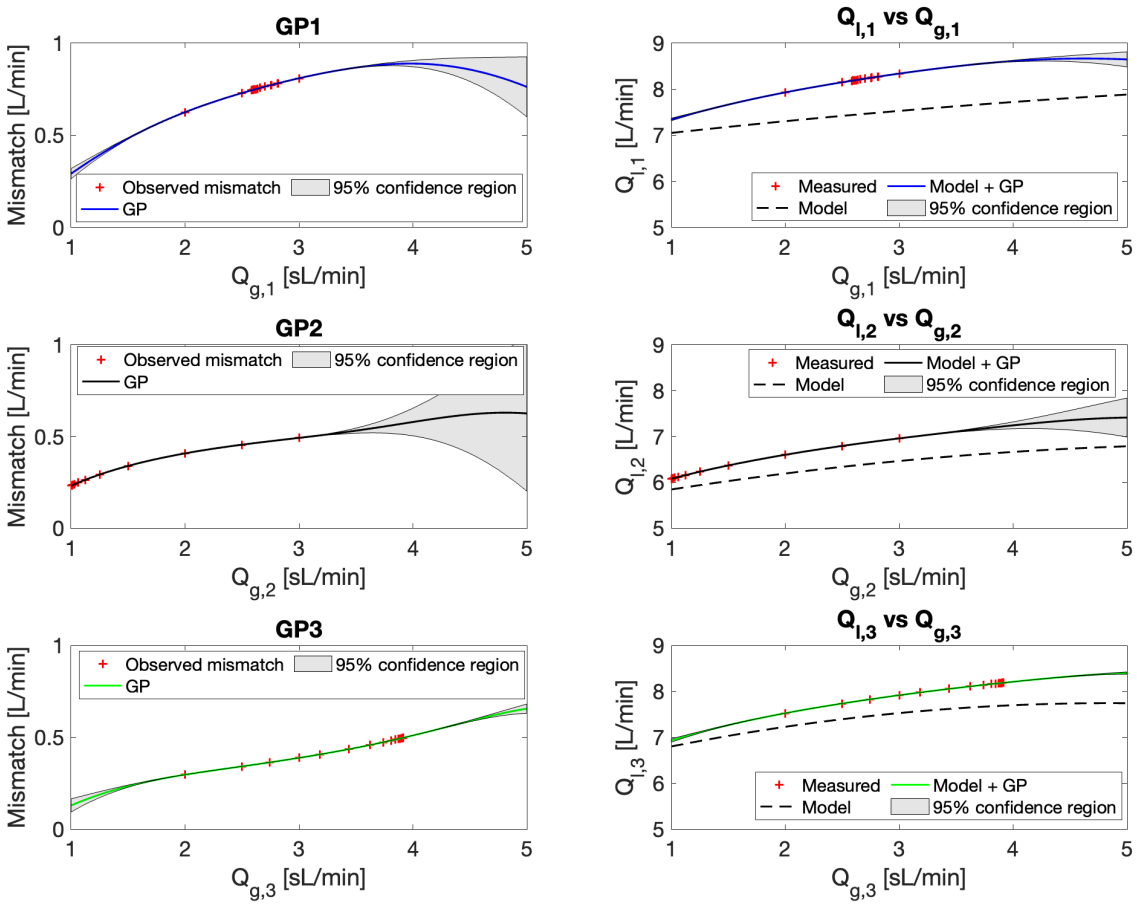


Figure 5.2: The first column shows the three GP's in the end of the simulation, which represent the plant-model mismatch in the outputs in the simulation. The second column shows the relationship between the inputs and the outputs. Noise is not present in this simulation.

Figure 5.2 shows the GPs from the simulation, as well as the relationship between the gas flowrates and the liquid flowrates in each well. The plots are shown without the valve openings, since they are held constant during the simulation. The left plots show the GPs together with the observed data points (mismatch) and a 95% confidence region. As seen from the GP plots, the confidence regions get larger away from the observed data points. There is no uncertainty in the regions with observed data points,

which can be explained by the hyperparameters of the GPs. Table 5.1 shows the hyperparameters of the GPs. The σ_n parameter is zero in all the GPs, because of no measurement noise, which means that all the uncertainty in the predicted GPs is due to lack of observed data. The l parameter in all the GPs are relatively large compared to the range of the inputs. The inputs range from 1 to 5, and the length scales are 1.97, 1.61 and 1.54 in GP1, GP2 and GP3 respectively. The predicted GPs are therefore smooth, and areas less than l units away from the observed data have low uncertainty, i.e. between the observed data points.

The right plots in Figure 5.2 show the relationship between the inputs and outputs. The first plot is associated with well 1, the second with well 2 and the third with well 3. The plots show the measured liquid flowrate, the simple process model and the sum of the simple process model and the GP of the respective well. The shaded region in the plots shows a 95% confidence region, which is predicted as the uncertainty of the GPs. Since the GP is representing the plant-model mismatch in the output, adding the model should give a proper estimate of the plant output. From the figures, it can be seen that adding the model and the GP of the three wells gives a perfect estimate of the plant outputs.

5.1.2 Simulation with Noise

The result from the simulation case study with implemented measurement noise and controller delay is shown in Figure 5.3. The left plots show the valve openings, the output measurements and the instantaneous profit (objective function). The upper right plot is showing the inputs, as well as the pre-calculated optimal inputs. From the bottom right plot, it can be seen that the total gas injected is 7.5 sL/min throughout the simulation. It may look like the gas constraint is violated, but this is due to measurement noise in the gas injection rates. The setpoints of the gas flowrates are adding up to 7.5 sL/min throughout the simulation.

As before, the inputs start at a sub-optimal operating point. At time 2 and 3 minutes they are set to their decided values in order to generate initial data sets. The first optimization is taking place after 4 minutes. Q_{g_2} and Q_{g_3} are approaching their optimal values, while Q_{g_1} is moving away from its optimal value the first iterations. At time 8 minutes, Q_{g_1} drops from 2.88 to 2.35 sL/min and Q_{g_2} jumps from 1.12 to 1.45 sL/min. These changes are relatively large, considering that the inputs are filtered with a filter value equal to 0.55. This behaviour can be explained from the predicted GPs in Figure 5.4 and will be further discussed. Q_{g_1} and Q_{g_2} have a similar behaviour also at time 12 minutes, before they move in the right direction and towards the optimum. The plant optimum is $\mathbf{u} = [2.59, 1.00, 3.91]^T$, which is the desired ending point the inputs should converge to. The inputs do not converge to the optimum values. The last MAy-GP iteration gives $\mathbf{u} = [2.75, 1.00, 3.75]^T$, which means that there is an offset in the end of the simulation.

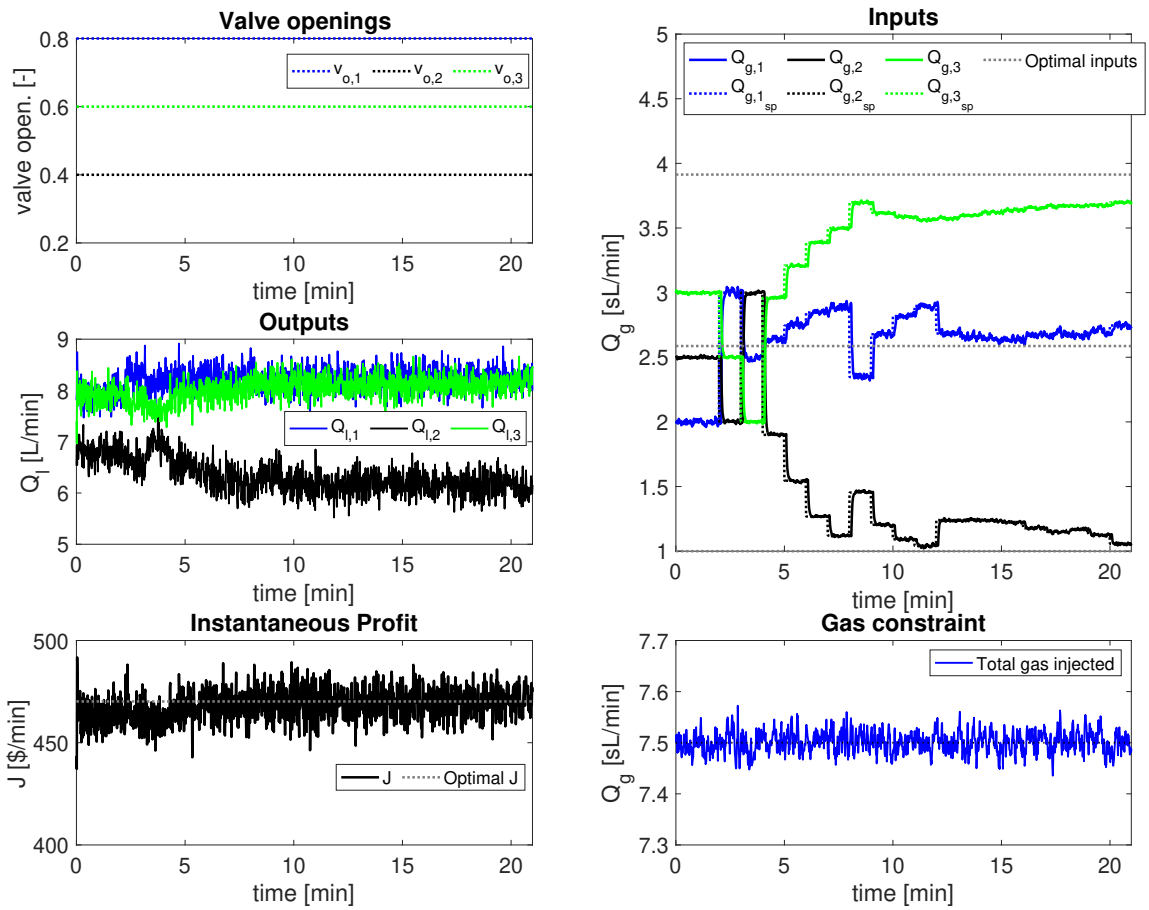


Figure 5.3: Simulation result with noise. The figure shows the valve openings, the outputs, the instantaneous profit (objective function), the inputs and the total gas injected.

Figure 5.4 shows the GPs from the simulation, as well as the relationship between the gas flowrates and the liquid flowrates in each well. The right plots show the measured liquid flowrate, the simple process model and the sum of the model and the GP of each well. The sum of the model and the GP represents an estimate of the outputs. The uncertainty, shown as a 95% confidence region, is predicted as the uncertainty of the GP.

The GPs are shown on the left side in the figure and their hyperparameters are shown in Table 5.2. The shaded region in the plots shows a 95% confidence region. In comparison to the first simulation case study where all the σ_n parameters were zero, the σ_n parameter of all the GPs in this case study is non zero, due to the presence of noise in the measurements. The length scale l of GP1 and GP3 are 6.14 and 7.75 respectively, which means that they are larger than the input range. The result is smooth, almost linear, functions with uncertainty almost entirely due to the GPs estimate of the noise level. The length scale of GP2 is also relatively large, which makes GP2 smooth. It is reasonable to compare these length scales to the ones in the former case study without

noise, since the inputs in the former case study converged to optimum. When comparing, one can see that the length scales in the case of noise are approximately double or triple the value of the length scales from the former case study, depending on which GP is considered.

Table 5.2: GP hyperparameters in the simulation with noise.

	GP1	**GP2**	**GP3**
l	6.14	2.96	7.75
σ_f	0.680	0.418	0.475
σ_n	0.0412	0.0344	0.0360

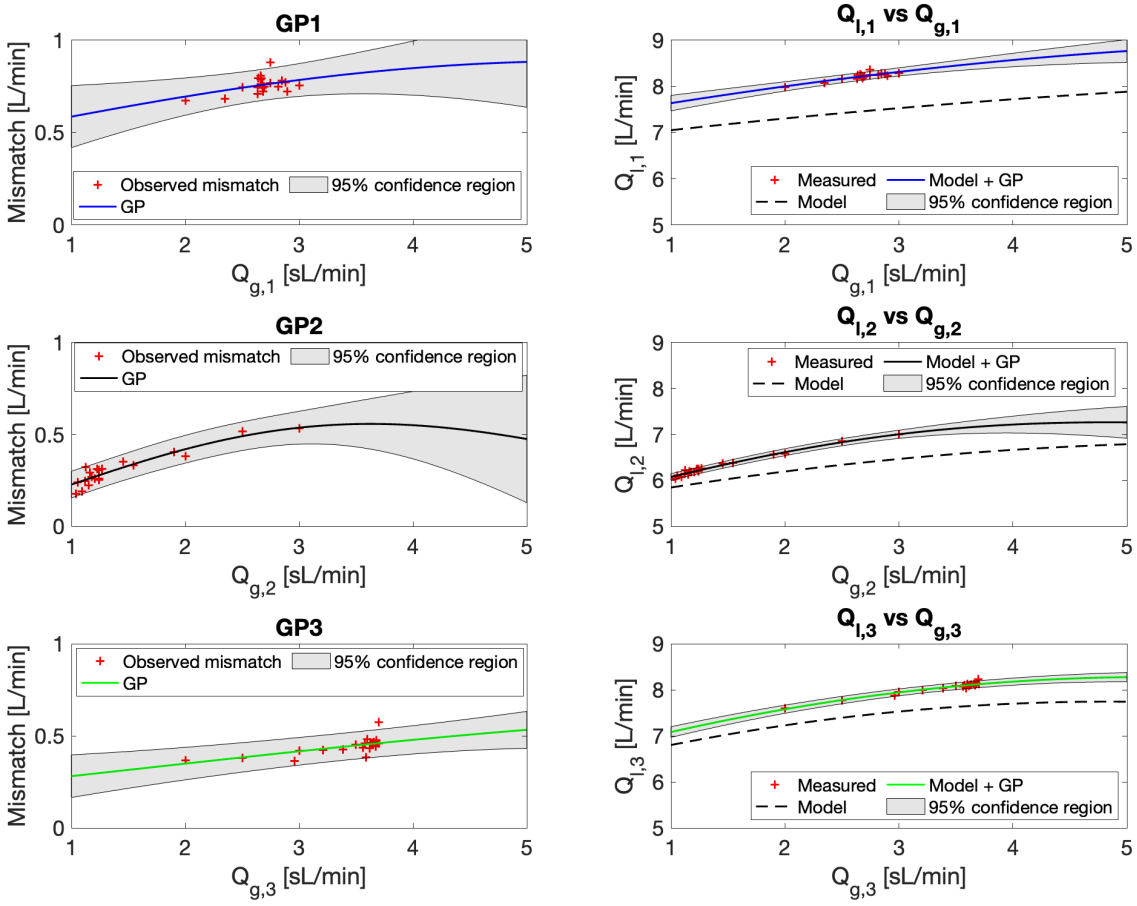


Figure 5.4: The first column shows the three GP's in the end of the simulation, which represent the plant-model mismatch in the outputs. The second column shows the relationship between the inputs and the outputs. Noise is included in this simulation.

The convergence issue, in addition to the undesired behaviour of the inputs at time 8 minutes, can be explained by the predicted GPs. At time 8 minutes, it was seen that Q_{g_2} jumps from 1.12 to 1.45 sL/min. Looking at GP2 in Figure 5.4 where the input Q_{g_2}

range from 1 to 2 sL/min, the uncertainty is approximately equal to the overall increase in the mismatch in this region. This means that the GP is not able to predict the true mismatch. The slope of the GP is slightly changing each time a new data point is added to the observed data set. When the GP do not predict the true response accurately in this region, it will not be able to consistently predict the same input at every iteration. This is also the reason why the inputs do not converge. If the predicted GPs did not change based on new observed data they would have converged.

Comparing the numerical values of the predicted mismatch in this case study with the former case study, one can see that the predicted plant-model mismatch have slightly different values. As an example, looking at GP3 in Figure 5.2 with no noise the predicted mismatch at $Q_{g3} = 4$ sL/min is equal to 0.5 L/min. Looking at GP3 in Figure 5.4 the predicted mismatch is lower. This means that the noise is hiding the true response of the system, resulting in a more "flat" response.

5.2 Experimental Runs in Lab Rig

This section shows the results with MAy-GP implemented in the experimental lab rig. The rig uses LabVIEW [23] as an interface between MATLAB and the rig. The MATLAB codes used in the experimental rig is shown in Appendix C.

Several experiments were carried out in the experimental rig with constant valve openings and a duration of 20 minutes. Different runs gave slightly different results, and the most typical result is chosen to be presented and discussed in this section. The reason why different runs gave different outcomes can be explain by a combination of the high noise level in the rig and the small values of observed mismatch. This will be further discussed.

The experimental result from the chosen experiment is shown in Figure 5.5. The left plots show the valve openings, the output measurements and the instantaneous profit (objective function). The bottom right plot shows the total gas injected. The inputs are shown in the upper right plot, where only the actual gas flowrates are shown and not the setpoints to the PID controllers. The first action of the algorithm is approximately at time 1 minutes, where the inputs change in order to obtain initial observed data. The first optimization is at around time 3 minutes. At first glance, it looks like the inputs are converging perfectly to an optimum. However, they are converging to the model optimum $\mathbf{u} = [2.78, 1.00, 3.72]^T$.

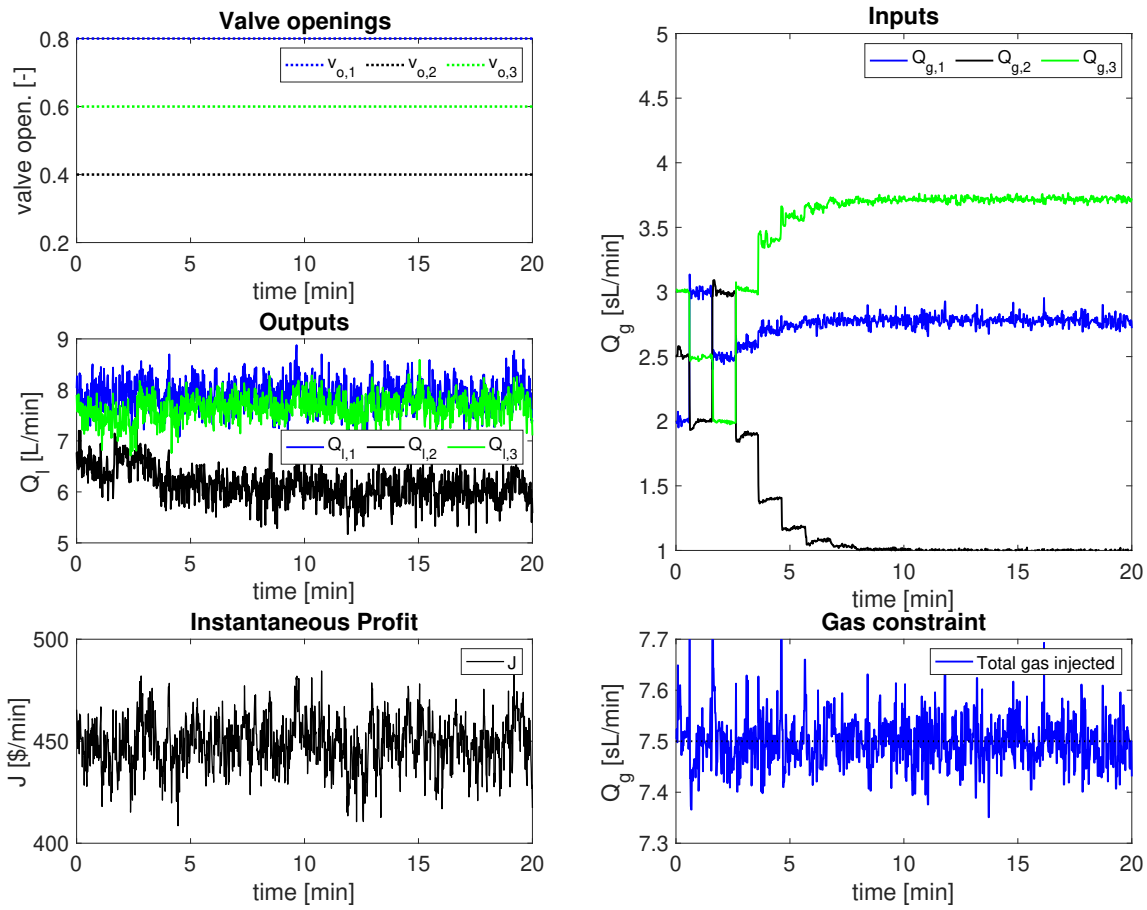


Figure 5.5: Experimental result from the lab rig. The figure shows the valve openings, the outputs, the instantaneous profit, the inputs and the total gas injected.

The convergence to the model optimum can be explained by the predicted GPs shown on the left side in Figure 5.6. The hyperparmeters of the GPs are shown in Table 5.3. The GPs have such large length scales l that they become flat and predict a constant mismatch in the whole input range. Because the slope of the GPs is zero, the first order modifier $\boldsymbol{\lambda}^{\mathbf{y}_{GP}}$ becomes zero. The value of the entries in the zeroth order modifer $\boldsymbol{\varepsilon}^{\mathbf{y}_{GP}}$ will not affect the optimization problem since they appear as constant terms in the objective function. The result is no correction by the GPs and the inputs converge to the model (Eq. 3.3) optimum.

The right plots in Figure 5.6 show the measured liquid flowrate, the simple process model, and the sum of the model and the GP of each well. The uncertainty is shown as the 95% confidence region of the GPs. The combined model and GP in each plot gives a reasonable estimate for the outputs, but due to the noise level the uncertainty is large.

Table 5.3: GP hyperparameters in the experimental run.

	GP1	GP2	GP3
l	53.2	7700	850
σ_f	0.452	0.209	0.000
σ_n	0.113	0.108	0.105

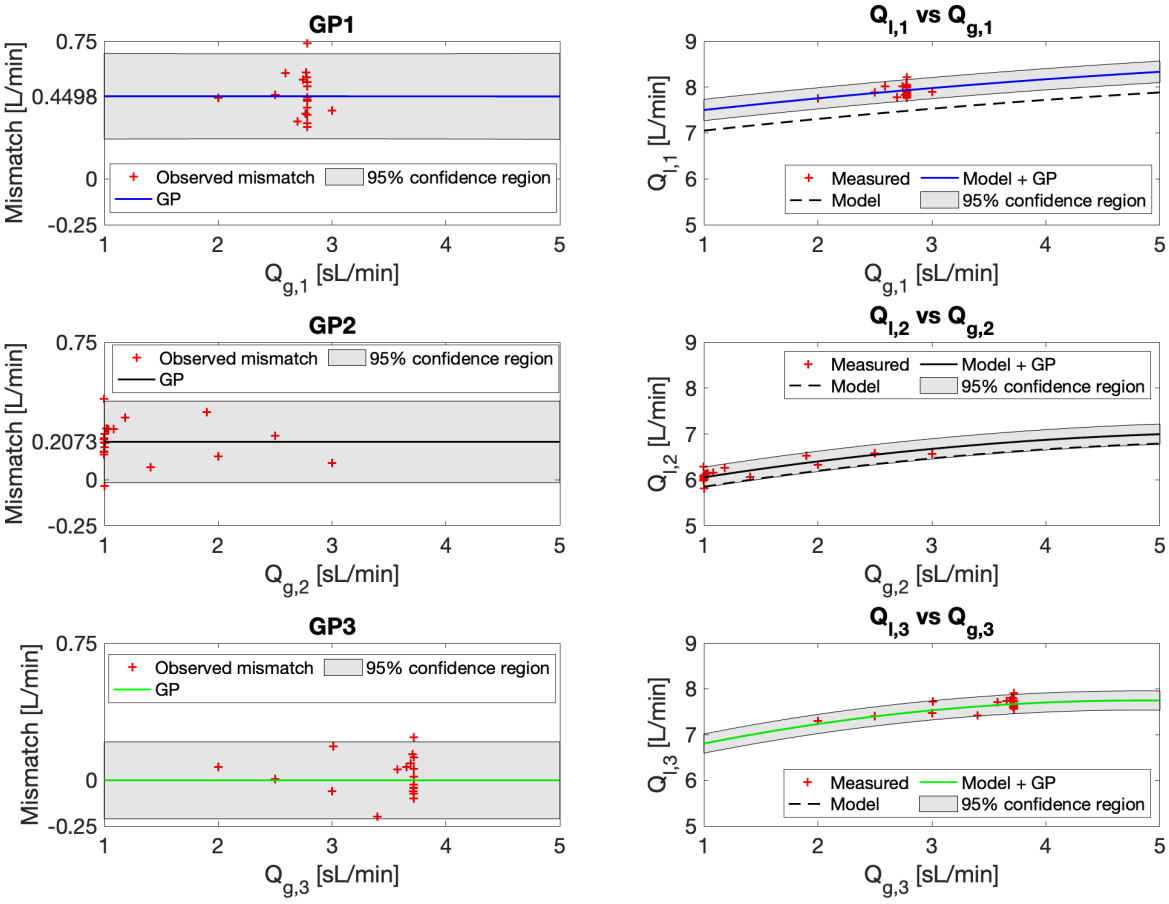


Figure 5.6: The first column shows the three GP's in the end of the experiment, which represent the plant-model mismatch in the outputs. The second column shows the relationship between the inputs and the outputs.

The uncertainty in the GPs is entirely due to the GPs estimate of the noise level as we observe the same uncertainty in the areas with and without observed data. Looking at the σ_n parameters shown in Table 5.3, it can be observed that these values are approximately triple the value of the σ_n parameters in the simulation case study described in Section 5.1.2. Even though the noise level in the simulation was calculated to match the noise in the rig, the noise is more complex in the rig. The noise in the rig is not perfectly Gaussian distributed, unlike the noise in the simulation. The GPs predict an uncertainty approximately $\pm$ 0.2 L/min in the whole input range, while the predicted

mismatch by GP1, GP2 and GP3 is 0.45, 0.21 and 0 L/min respectively. As observed in the simulation case study with noise, the high noise level is hiding the true response of the system resulting in flat GPs.

The observed mismatch in the experiment is smaller than in the simulation. This is because the simple process model was obtained from rig data, and not simulation data. The model used in the optimization was obtained by the intention to be simple and to contain a larger mismatch than rigorous first-principle models. However, the predicted mismatch is small compared to the uncertainty in the GPs due to the noise level, which makes the prediction poor.

5.2.1 Comparison with Traditional RTO

The optimal inputs of the rig are not known. Therefore, a previously made traditional steady-state RTO [19] is used to give an idea of where the optimum is. A comparison of experimental runs with traditional RTO and MAy-GP is shown in Figure 5.7. The traditional RTO is optimizing every 60 seconds and converges to $\mathbf{u} = [2.65, 1.00, 3.85]^T$ after 7-8 iterations. The MAy-GP algorithm converges to the model optimum $\mathbf{u} = [2.78, 1.00, 3.72]^T$, as seen before.

The bottom plot in Figure 5.7 shows a comparison of the instantaneous profit in both experiments. The red line shows the objective function with RTO, while the black line shows the objective function with MAy-GP. There is no visible loss in the objective function, despite the difference in the inputs. This was also observed in the simulation case study with noise. The objective function is not sensitive to changes in the inputs around the optimum.

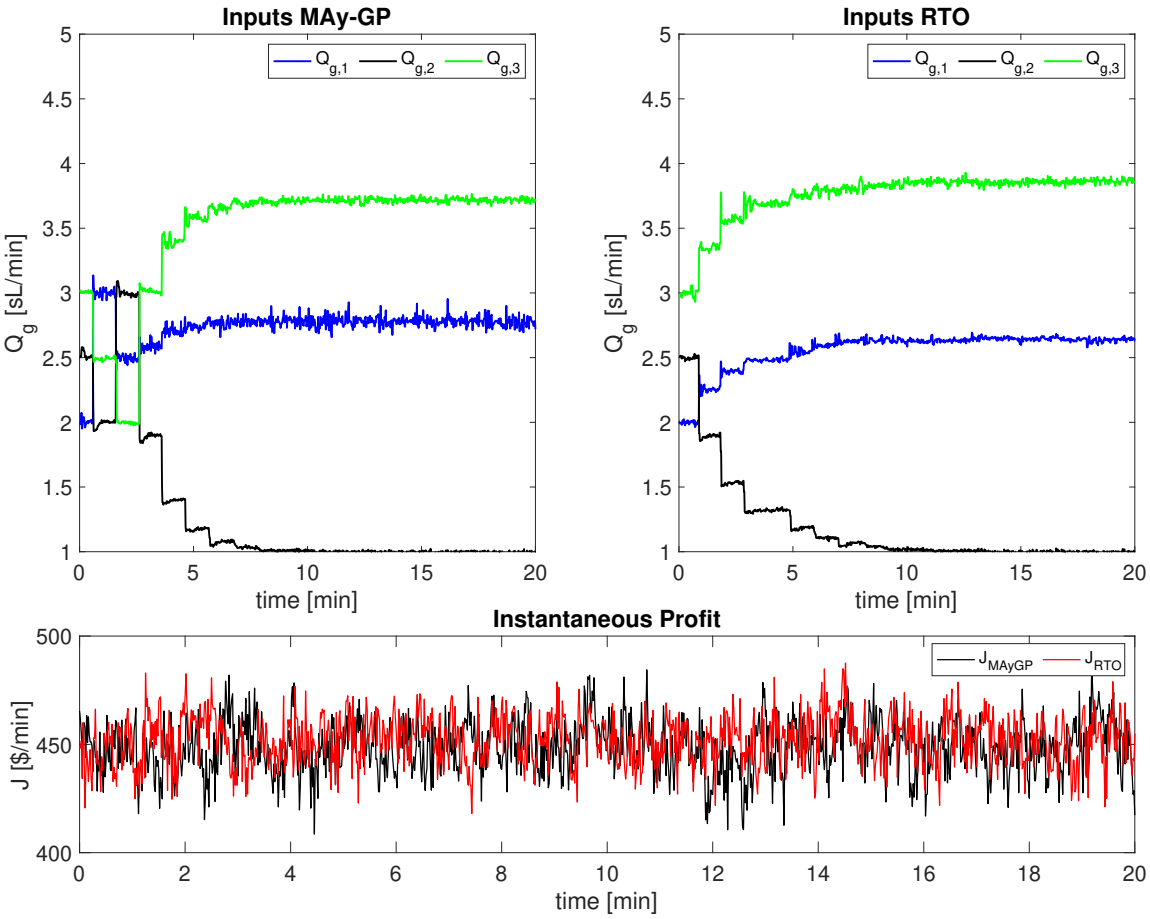


Figure 5.7: Comparison of MAy-GP and traditional RTO. The inputs with both MAy-GP and traditional RTO is shown, as well as the instantaneous profit in both experiments.

5.2.2 Attempts for Improvement

One additional observed data point at each iteration

To improve the performance of the MAy-GP algorithm experiments with one extra observed data point at each $\mathbf{u}_k$ was carried out. In practice this means staying at each $\mathbf{u}_k$ for a longer time period in order to observe another data point. The outcome of these experiments were the same as with only one observed data point at each input sequence. That is, the GPs were flat and thus the first order modifier became zero. This can be explained by looking at the upper left plot in Figure 5.5. Q_{g_1} converges to 2.78 sL/min, so at this point there is a larger density of observed data. The observed mismatch at this input range from approximately 0.25 to 0.75 L/min. Giving the GPs only one extra measurement at each $\mathbf{u}_k$ will therefore not be sufficient to give an accurate estimate of the curvature of the mismatch, as this is too small compared to the uncertainty.

Experimental run with larger plant-model mismatch

Since the observed mismatch were so small compared to the uncertainty in the experiments, it was chosen to do some experimental runs with a different model with larger plant-model mismatch. It was chosen to change all the model parameters (Table 3.1) to 1, in order to make the mismatch large. The optimal inputs with this model is $\mathbf{u} = [1.50, 1.00, 5.00]^T$.

The experimental result of one chosen experimental run with this model is shown in Figure 5.8 and 5.9. The GPs are able to correct for the major mismatch of the model. However, it is not able to converge to the system optimum. Instead the inputs exhibits a cyclic behaviour with the gas injection not fully used, as shown in the bottom right plot in Figure 5.8. After 10 number iterations all the inputs change alternately to approximately 1.5, 2.1 and 3.4 sL/min.

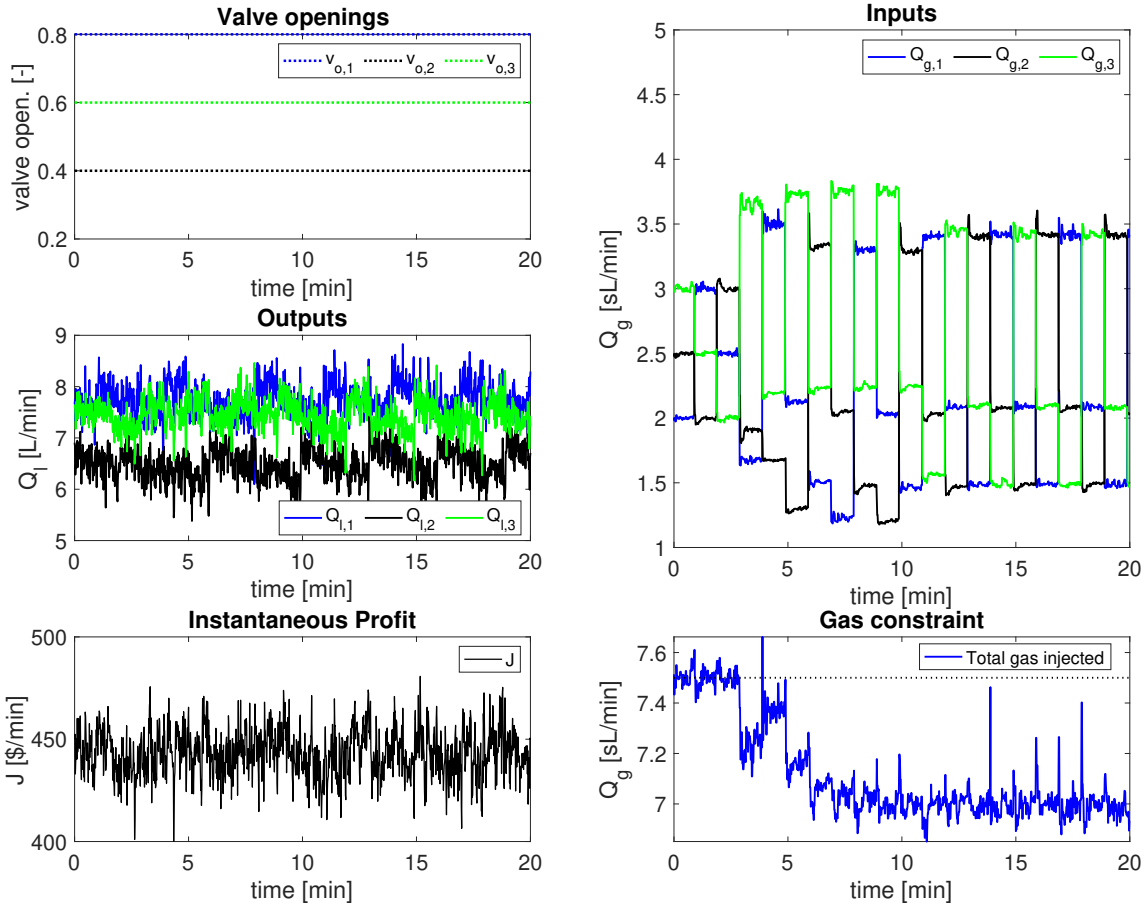


Figure 5.8: Experimental result with "new" model and larger mismatch. The figure shows the valve openings, the outputs, the instantaneous profit, the inputs and the total gas injected.

The cycling behaviour is explained by the predicted GPs. Looking at the right plots in Figure 5.9 the combined model and GP is close to flat in the interval where Q_g range

from 1.5 to 3.4 sL/min. This means that it predicts that the liquid flowrates are nearly independent from the gas injected. The actual slope in this interval will change a little each time new data points are observed. The algorithm with this model is hence not able to predict the same inputs at every iteration and the inputs alternate between approximately 1.5, 2.1 and 3.4 sL/min.

Again, it is observed that the noise level is too large to ensure proper corrections by the GPs. However, this experiment did show a positive result. It is observed that even with a huge plant-model mismatch the GPs are still able to offer useful corrections. Even though the GPs in this experiment were unable to correct to the system optimum, it did some corrections and the inputs did not converge to the model optimum, as in the previous experiment with the original model.

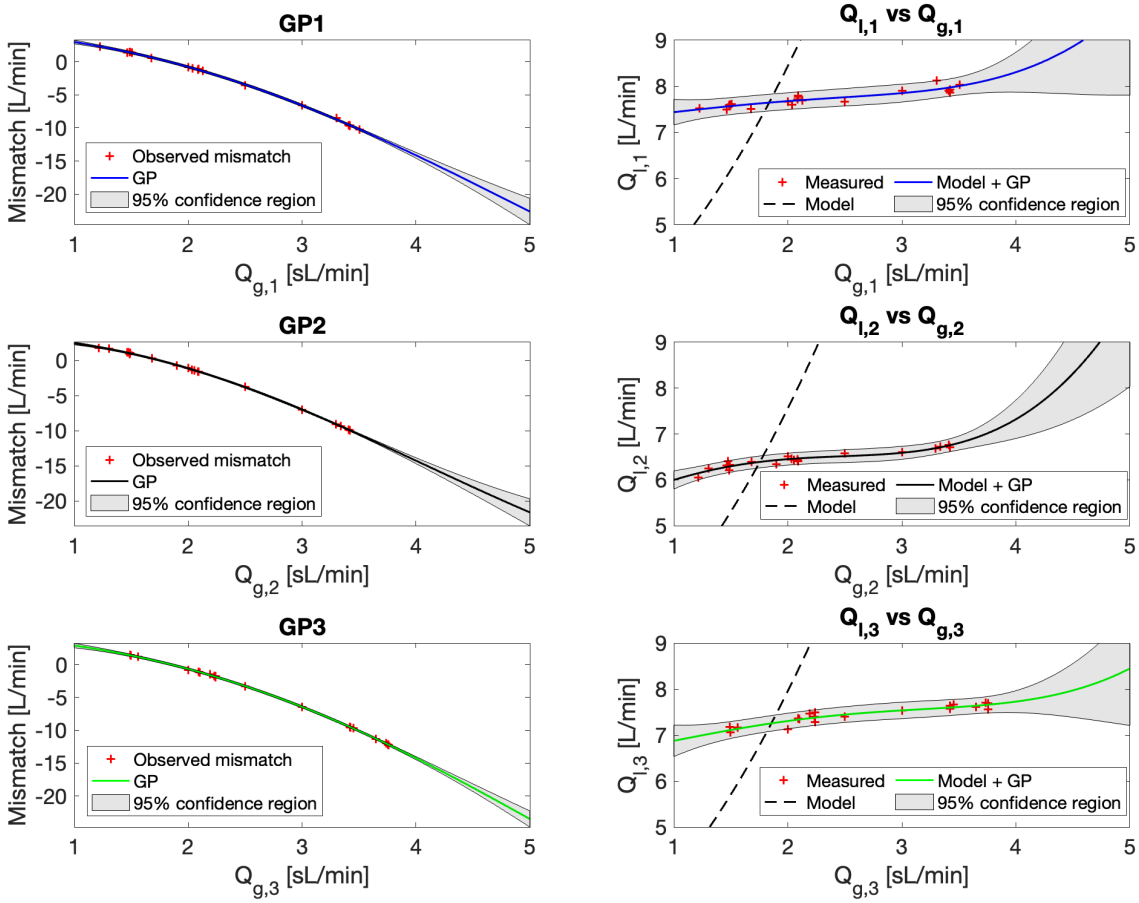


Figure 5.9: The left plots show the three GP's in the end of the experiment with larger mismatch. The right plots show the relationship between the inputs and the outputs.

Chapter 6

Conclusion

The inputs with MAy-GP converged to the optimum in the simulation case study without implemented measurement noise. This case study is not realistic, but gives an intuition on how the method works. Noise was implemented in the second simulation case study to represent a more realistic simulation of the experimental rig. In this simulation the inputs did not converge to the optimum, but were close in the end of the simulation. It was clear that the presence of noise hampered the GPs, which lead to convergence issues. Slight changes in the prediction of the GPs at every iteration made it difficult for the algorithm to predict the same inputs every iteration. However, the performance in second simulation case study was acceptable and the algorithm was implemented in the actual experimental rig.

The experimental runs in the lab rig gave slightly different results each run. The most typical result consisted of predicted GPs with slopes equal to zero. This lead to no correction by the first order modifier and the inputs converged to the model optimum. The noise level was too large for the deterministic trend of the mismatch to be identified. This indicates that the noise level in the rig is hiding the true response of the system. It was also observed that giving the GPs one additional data point at each input was not sufficient to give an accurate estimate of the curvature of the mismatch. The experimental result with a larger mismatch in the model showed that the GPs were able to correct for large errors. However, they were not able to correct all the way to the optimum due to the high noise level.

Even though combining MAy with GPs eliminates the challenging gradient estimation step, there are still challenges to overcome before it can be applied to real systems with high noise levels. Thus, the proposed MAy-GP scheme seems to be more suited for systems with less measurement noise or to correct for systems with large plant-model mismatch.

Chapter 7

Future Work

The noise tolerance of the algorithm should be improved. Adding more data is unlikely to improve the performance as the true curvature of the mismatch is hidden by the large noise level. Instead, other approaches should be considered. There is a gap in the literature when it comes to Modifier Adaptation variants implemented in real systems. Future work could involve testing other MA variants from the literature on this experimental system.

Paper [24] suggests to combine Modifier Adaptation with Stochastic Approximation (SA) in processes with model uncertainties and noisy measurements. SA is a collective term used for model-free stochastic algorithms which are able to find the extrema of unknown functions, given only noisy observations. The SA approach uses estimates of noisy plant gradients directly in the calculation of the new input sequence and has a proven ability to converge to a critical point [24]. The SA procedure is a slow procedure, and are hence only used when the inputs are near the optimal solution or after a number of unsuccessful iterations of standard MA. This combined MA and SA scheme is tested in a simulation of the Williams-Otto reactor problem. The result shows a good performance in the case of a high noise level, where the proposed algorithm outperforms the standard MA. Since this approach gives such promising results in the simulation, it would be interesting to implement it in the experimental rig.

Bibliography

[1] Rodrigo Curvelo et al. 'Investigation of the use of transient process data for steady-state Real-Time Optimization in presence of complex dynamics'. In: *31st European Symposium on Computer Aided Process Engineering*. Ed. by Metin Türkay and Rafiqul Gani. Vol. 50. Computer Aided Chemical Engineering. Elsevier, 2021, pp. 1299–1305. DOI: https://doi.org/10.1016/B978-0-323-88506-5.50200-X.

[2] Dinesh Krishnamoorthy. 'Novel approaches to online process optimization under uncertainty'. In: *Doktoravhandlinger ved NTNU.* (2019).

[3] Mark Darby et al. 'RTO—An Overview and Assessment of Current Practice'. In: *Journal of Process Control* 21 (July 2011), pp. 874–884. DOI: 10.1016/j.jprocont.2011.03.009.

[4] Alejandro Marchetti et al. 'Modifier Adaptation for Real-Time Optimization – Methods and Applications'. In: *Processes* 4 (Dec. 2016). DOI: 10.3390/pr4040055.

[5] A. Marchetti, B. Chachuat and D. Bonvin. 'Modifier-Adaptation Methodology for Real-Time Optimization'. In: *Industrial & Engineering Chemistry Research* 48.13 (2009), pp. 6022–6033. DOI: 10.1021/ie801352x. eprint: https://doi.org/10.1021/ie801352x. URL: https://doi.org/10.1021/ie801352x.

[6] Tafarel de Avila Ferreira et al. 'Real-Time optimization of Uncertain Process Systems via Modifier Adaptation and Gaussian Processes'. In: *2018 European Control Conference (ECC)*. 2018, pp. 465–470. DOI: 10.23919/ECC.2018.8550397.

[7] E. Bradford E.A. del Rio Chanona J.E Alves Granciano and B. Chachuay. 'Modifier-Adaptation Schemes Employing Gaussian Processes and Trust Regions for Real-Time Optimization'. In: *IFAC PapersOnLine* 52-1 (2019), pp. 52–57.

[8] Victor S. P. Ruela and Michel Bessani. 'Investigation of Initial Data and Optimizer in Real-Time Optimization Performance via Modifier Adaptation with Gaussian Processes'. In: *2021 IEEE Conference on Control Technology and Applications (CCTA)*. 2021, pp. 208–213. DOI: 10.1109/CCTA48906.2021.9658960.

[9] A. Papasavvas et al. 'Analysis of output modifier adaptation for real-time optimization'. In: *Computers & Chemical Engineering* 121 (2019), pp. 285–293. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2018.09.028.

[10] José Matias and Johannes Jäschke. 'Online Model Maintenance via Output Modifier Adaptation'. In: *Industrial & Engineering Chemistry Research* 58.30 (2019), pp. 13750–13766. DOI: 10.1021/acs.iecr.9b00267. URL: https://doi.org/10.1021/acs.iecr.9b00267.

[11] Maren Sofie Lia. 'Modifier Adaptation: implementation in Experimental Lab Rig'. In: *TKP4580 Chemical Process Technology, Specialization project* (2021).

[12] Ryad Bousbia-Salah et al. 'Dynamic Real-time Optimization of a Batch Polymerization Process'. In: *27th European Symposium on Computer Aided Process Engineering*. Ed. by Antonio Espuña, Moisès Graells and Luis Puigjaner. Vol. 40. Computer Aided Chemical Engineering. Elsevier, 2017, pp. 1741–1746. DOI: https://doi.org/10.1016/B978-0-444-63965-3.50292-0.

[13] George E. P. Box and Norman R. Draper. *Response Surfaces, Mixtures, and Ridge Analyses*. Hoboken, New Jersey: Wiley, 2007.

[14] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. the MIT Press, 2006.

[15] Jie Wang. *An Intuitive Tutorial to Gaussian Processes Regression*. 2021. URL: https://arxiv.org/pdf/2009.10862.pdf (visited on 02/03/2022).

[16] Hilarie Sit. *Quick Start to Gaussian Process Regression*. 2019. URL: https://towardsdatascience.com/quick-start-to-gaussian-process-regression-36d838810319 (visited on 17/02/2022).

[17] Nando de Freitas. *CPSC540, Gaussian Processes*. 2013. URL: https://www.cs.ubc.ca/~nando/540-2013/lectures/l6.pdf (visited on 20/01/2022).

[18] David Duvenaud. *The Kernel Cookbook: Advice on Covariance functions*. URL: https://www.cs.toronto.edu/~duvenaud/cookbook/ (visited on 14/03/2022).

[19] José Matias. 'Implementation of RTO using transient measurements on experimental rig'. In: (2020).

[20] Dinesh Krishnamoorthy, Bjarne Foss and Sigurd Skogestad. 'Real-Time Optimization under Uncertainty Applied to a Gas Lifted Well Network'. In: *Processes* 4.4 (2016). ISSN: 2227-9717. DOI: 10.3390/pr4040052. URL: https://www.mdpi.com/2227-9717/4/4/52.

[21] *CasADi*. URL: https://web.casadi.org (visited on 17/03/2022).

[22] Carl Edward Rasmussen and Hannes Nickisch. *The GPML Toolbox Manual*. 2018. URL: http://www.gaussianprocess.org/gpml/code/matlab/doc/manual.pdf (visited on 10/02/2022).

[23] National Instrument Corp. *What Is LabVIEW?* 2022. URL: https://www.ni.com/en-no/shop/labview.html# (visited on 13/05/2022).

[24] Reinaldo Hernández and Sebastian Engell. 'Stochastic Approximation in Online Steady State Optimization Under Noisy Measurements'. In: *27th European Symposium on Computer Aided Process Engineering*. Ed. by Antonio Espuña, Moisès Graells and Luis Puigjaner. Vol. 40. Computer Aided Chemical Engineering. Elsevier, 2017, pp. 1747–1752. DOI: https://doi.org/10.1016/B978-0-444-63965-3.50293-2.

[25] *The finite difference method*. URL: https://www.ljll.math.upmc.fr/frey/cours/UdC/ma691/ma691_ch6.pdf (visited on 17/03/2022).

Appendix A

Finite Difference Approximation

For a smooth function $f(x)$, the derivative at point $x \in \mathbb{R}$ is defined by:

$$\frac{\partial f(x)}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{A.1}$$

, where h is the step length in x [25]. An approximation of the derivative can therefore be expressed as shown in Eq. (A.2). This approximation is called the forward finite difference approximation.

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x)}{h}, \quad h > 0 \tag{A.2}$$

The approximation requires a sufficiently small h to be a good estimate of the gradient. However, there is a trade-off in the choice of h in the presence of noisy function outputs.

Appendix B

Experimental Rig Modeling[1]

The model uses mass balances of the different phases, density models, pressure models and flow models, which become the following differential algebraic equation (DAE)

$$\begin{aligned} \dot{\mathbf{x}}_i &= \mathbf{f}_i(\mathbf{x}_i, \mathbf{z}_i, \mathbf{u}_i, \mathbf{p}_i) \\ \mathbf{g}_i(\mathbf{x}_i, \mathbf{z}_i, \mathbf{u}_i, \mathbf{p}_i) &= \mathbf{0} \quad \forall i \in \mathcal{N} = \{1, \ldots, n_w\} \end{aligned} \tag{B.1}$$

where $f_i(x_i, z_i, u_i, p_i)$ is the set of differential equations and $g_i(x_i, z_i, u_i, p_i)$ is the set of algebraic equations. n_w is the number of wells, i.e. n_w=3 in this case. The subscript i is referring to a well in the set $\mathcal{N}$. x_i is the differential states, z_i is the algebraic states, u_i is the decision variables.

The differential states, the algebraic states and the decision variables are given by

$$\begin{aligned} \mathbf{x}_i &= [m_{g_i} \; m_{l_i}]^T \\ \mathbf{z}_i &= [w_{l_i} \; w_{total_i} \; p_{rh_i} \; p_{bi_i} \; \rho_{mix} \; \rho_{g_i} \; w_{gout_i} \; w_{lout_i}]^T \\ \mathbf{u}_i &= [Q_{g_i} \; v_{o_i} \; p_{pump}]^T \end{aligned} \tag{B.2}$$

where m_{g_i} is the gas hold up and m_{l_i} is the liquid hold up. w_{l_i} is the water rate from the reservoir and w_{total_i} is the total well production rate. p_{rh_i} is the riser head pressure and p_{bi_i} is the pressure before injection point. ρ_{mix} is the mixture density in the system and ρ_{g_i} is the density of the gas. w_{gout_i} and w_{lout_i} is the well outlet gas and liquid outlet flowrate respectively. Q_{g_i} is the gas lift injection rate, v_{o_i} is the valve opening from the reservoir and p_{pump} is the reservoir pressure.

In addition, the model contains some constant parameters and some uncertain parameters. These parameters are given by

[1] Appendix taken from the project work presented in [11] by the same author.

$$\mathbf{p}_i = [p_{atm}\ T\ R\ M_w\ \rho_l\ \mu_{mix}\ L_w\ A_w\ L_r\ H_r\ D_{bh}\ \theta_{res}\ \theta_{top}]^T$$

where p_{atm} is the atmospheric pressure, T is the room temperature, R is the gas constant, M_w is the molecular weigth of air, ρ_l is the desity of water, μ_{mix} is the mixture viscosity, L_w is the well length, A_w is the well pipes cross section area, L_r is the riser length, A_r is the riser pipes cross section area, H_r is the riser height and D is well below injection. θ_{res} is the reservoir valve flow coefficient and θ_{top} is the top valve flow coefficient [19].

The code for the dynamic model is shown in the section below. The differential equations are $\mathbf{f} = [df_1\ df_2]^T$, while the algebraic equations are $\mathbf{g} = [f_1\ f_2\ f_3\ f_4\ f_5\ f_6\ f_7\ f_8]^T$.

B.1 ErosionRigDynModel.m

```
function [F,S_xx,S_zz,S_xz,S_xp,S_zp,x_var,z_var,u_var,p_var,dif,alg,L]
        = ErosionRigDynModel(par)
%    Creates a dynamic model of the rig and computes the sensitivity
%    matrices for EKF

% Inputs:
%    par = system parameters
%
% Outputs:
%   F: system integrator
%   S's: system sensitivities
%   x_var,z_var,u_var,p_var, diff,alg,L: Model in CasADi form

% Other m-files required: none
% MAT-files required: none

addpath('/Applications/casadi-osx-matlabR2014b-v3.5.5')
import casadi.*

%% Parameters
%number of wells
n_w = par.n_w; %[]
%gas constant
R = par.R; %[m3 Pa K^-1 mol^-1]
%air molecular weigth
Mg = par.Mw; %[kg/mol] -- Attention: this unit is not usual
```

```
%properties
%density of water - dim:  nwells x 1
rho_l = par.rho_o; %[kg/m3]
%mixture viscosity
mu_mix = par.mu_oil;% [Pa s]

%project - dim:  nwells x 1
% well length
L_w = par.L_w; %[m]
% well pipes cross section area
A_w = par.A_w;%[m2]

% riser length
L_r = par.L_r; %[m]
% riser pipes cross section area
A_r = par.A_r;%[m2]
%riser height
H_r = par.H_r; %[m]
%well below injection
D = par.D_bh; %[m]

%% System states
% differential
%gas holdup
m_g = MX.sym('m_g',n_w);          % 1:3 [1e-4 kg]
%water holdup
m_l = MX.sym('m_l',n_w);          % 4:6 [kg]

% algebraic
%water rate from reservoir
w_l = MX.sym('w_l',n_w);          % 1:3 [1e-2 kg/s]
%total well production rate
w_total = MX.sym('w_total',n_w);    % 4:6 [1e-2 kg/s]

%riser head pressure
p_rh = MX.sym('p_rh',n_w);        % 7:9 [bar]
%pressure - before injection point (bottom hole)
p_bi = MX.sym('p_bi',n_w);        % 10:12 [bar]

%mixture density in system
```

```
rho_mix = MX.sym('rho_mix',n_w);  % 13:15 [1e2 kg/m3]
%density gas
rho_g= MX.sym('rho_g',n_w);       % 16:18 [kg/m3]

%well outlet flowrate (gas)
w_gout = MX.sym('w_gout',n_w);    % 19:21 [1e-5 kg/s]
%riser head gas production rate gas
w_lout = MX.sym('w_lout',n_w);    % 22:24 [1e-2 kg/s]

%% System input
%gas lift rate
Q_gl = MX.sym('Q_gl',n_w);        % 1:3 [sL/min]
%valve oppening
vo = MX.sym('vo',n_w);            % 4:6 [0-1]
%pump outlet pressure
Ppump = MX.sym('Ppump',1);        % 7 [bar]

%% parameters
%%%%%%%%%
% fixed %
%%%%%%%%%
%room temperature
T = MX.sym('T',1); %[K]
%atmospheric pressure
p_atm = MX.sym('p_atm',1); %[bar]

%time transformation: CASADI integrates always from 0 to 1 and the USER
↪  does the time
%scaling with T --> sampling time
t_samp = MX.sym('t_samp',1); %[s]

% estimable
%scaled reservoir valve parameters
res_theta = MX.sym('res_theta',n_w);
%scaled top valve parameters
top_theta = MX.sym('top_theta',n_w);

%% Modeling
% Algebraic
%conversion
CR = 60*10^3; % [L/min] -> [m3/s]
```

```
%reservoir outflow
f1 = -Ppump*ones(n_w,1)*1e5 + (w_l.*1e-2).^2.*(res_theta.*1e9)./(vo.^2.*rho_l) ...
+ p_bi.*1e5 ;
% total system production
f2 = - (w_total.*1e-2) + ((w_gout.*1e-5) + (w_lout.*1e-2));
%riser head pressure
f3 = -p_rh.*1e5 + (w_total.*1e-2).^2.*(top_theta.*1e8)./(rho_mix.*1e2) + p_atm.*1e5 ;
%before injection pressure
f4 = -p_bi.*1e5 + (p_rh.*1e5 + (rho_mix.*1e2).*9.81.*H_r + ...
128.*mu_mix.*(L_w+L_r).*(w_l.*1e-2)./(3.14.*D.^4.*(rho_mix.*1e2)));
%mixture density
f5 = -(rho_mix.*1e2) + (((m_g.*1e-4) + m_l).* ...
p_bi.*1e5.*Mg.*rho_l)./(m_l.*p_bi.*1e5.*Mg + rho_l.*R.*T.*(m_g.*1e-4));
%gas density (ideal gas law)
f6 = -rho_g + p_bi.*1e5.*Mg/(R*T);

% Simplifying assumption!
% liquid fraction in the mixture
xL = (m_l./((m_g.*1e-4) + m_l));
%Liquid outlet flowrate
f7 = -(w_lout.*1e-2) + xL.*(w_total.*1e-2);

% Total volume constraint
f8 = -(A_w.*L_w + A_r.*L_r) + (m_l./rho_l + (m_g.*1e-4)./rho_g);

% Differential
% gas mass balance
df1= 1e4*(-(w_gout.*1e-5) + Q_gl.*rho_g/CR);
% liquid mass balance
df2= -(w_lout.*1e-2) + (w_l.*1e-2);

% Form the DAE system
diff = vertcat(df1,df2);
alg = vertcat(f1,f2,f3,f4,f5,f6,f7,f8);

% give fixed parameter values
alg = substitute(alg,p_atm,par.p_s);
alg = substitute(alg,T,par.T_r);

% concatenate the differential and algebraic states
x_var = vertcat(m_g,m_l);
```

```
z_var = vertcat(w_l,w_total,p_rh,p_bi,rho_mix,rho_g,w_gout,w_lout);
u_var = vertcat(Q_gl,vo,Ppump);
p_var = vertcat(res_theta,top_theta,t_samp);

%objective function
L = 20*((w_lout(1)*1e-2)*CR/rho_l(1)) + 10*((w_lout(2)*1e-2)*CR/rho_l(2)) + ...
30*((w_lout(3)*1e-2)*CR/rho_l(3));
%end modeling

%% Casadi commands
%declaring function in standard DAE form (scaled time)
dae = struct('x',x_var,'z',z_var,'p',vertcat(u_var,p_var),'ode',t_samp*diff,'alg',alg);

%calling the integrator, the necessary inputs are: label; integrator;
↪  function with IO scheme of a DAE (formalized); struct (options)
F = integrator('F','idas',dae);

% ================================================
%       Calculating sensitivity matrices
% ================================================

S_xx = F.factory('sensStaStates',{'x0','z0','p'},{'jac:xf:x0'});
S_zz = F.factory('sensStaStates',{'x0','z0','p'},{'jac:zf:z0'});
S_xz = F.factory('sensStaStates',{'x0','z0','p'},{'jac:xf:z0'});

S_xp = F.factory('sensParStates',{'x0','z0','p'},{'jac:xf:p'});
S_zp = F.factory('sensParStates',{'x0','z0','p'},{'jac:zf:p'});

end
```

Appendix C

MATLAB Codes in the Lab Rig.

InitializationLabViewMain.m is the initialization file, which is ran before the main file. This file sets the sampling time and the model parameters. *ssModel.m* contains the model used in the optimization. *LabViewMain.m* is the main file ran by LabVIEW.

C.1 InitializationLabViewMain.m

```
%% Path to GPML toolbox
addpath(genpath(pwd))

%% Model Parameters

theta_well1 = [0.492764910641654; 0.300443533757929; 19.3122131197183;
               -0.0153700130556438; -14.3380841612070];
theta_well2 = [-0.273738047663130; 0.459291990662657; 20.1022897064108;
               -0.0373720166243209; -14.6387961216287];
theta_well3 = [-0.883020415785438; 0.619837772512701; 21.3127342608823;
               -0.0640840666704814; -15.7140573084694];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BE CAREFUL - it should match sampling time in LABVIEW interface %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Optimization sampling time
nExec = 60; %[s] 10
```

C.2 ssModel.m

```
function y_model_ss = ssModel(u, v0)
    % u = [FIC104; FIC105; FIC106]
    % v0 = [CV101; CV102; CV103]

    % Model parameters
    theta_well1 = [0.492764910641654; 0.300443533757929; 19.3122131197183;
                -0.0153700130556438; -14.3380841612070];
    theta_well2 = [-0.273738047663130; 0.459291990662657; 20.1022897064108;
                -0.0373720166243209; -14.6387961216287];
    theta_well3 = [-0.883020415785438; 0.619837772512701; 21.3127342608823;
                -0.0640840666704814; -15.7140573084694];

    % Model value
    y_well1_ss = theta_well1(1) + theta_well1(2)*u(1) + theta_well1(3)*v0(1) ...
              + theta_well1(4)*u(1)^2 + theta_well1(5)*v0(1)^2;

    y_well2_ss = theta_well2(1) + theta_well2(2)*u(2) + theta_well2(3)*v0(2) ...
              + theta_well2(4)*u(2)^2 + theta_well2(5)*v0(2)^2;

    y_well3_ss = theta_well3(1) + theta_well3(2)*u(3) + theta_well3(3)*v0(3) ...
              + theta_well3(4)*u(3)^2 + theta_well3(5)*v0(3)^2;

    y_model_ss = [y_well1_ss; y_well2_ss; y_well3_ss];

end
```

C.3 LabViewMain.m

```
% Main program
% Run Initialization file first
addpath ('C:\Users\lab\Documents\casadi-windows-matlabR2016a-v3.4.5')
import casadi.*

%%%%%%%%%%%%%%%%
% Get Variables
%%%%%%%%%%%%%%%%
% disturbances
%valve opening [-]
```

```
cv101 = P_vector(1);
cv102 = P_vector(2);
cv103 = P_vector(3);
% if value is [A] from 0.004 to 0.020
% if you want to convert to 0 (fully closed) to 1 (fully open)
% vo_n = (vo - 0.004)./(0.02 - 0.004);

%pump rotation [%]
pRate = P_vector(4);
% if value is [A] from 0.004 to 0.020
% if you want to convert to (min speed - max speed)
% goes from 12% of the max speed to 92% of the max speed
% pRate = 12 + (92 - 12)*(P_vector(4) - 0.004)./(0.02 - 0.004);

% always maintain the inputs greater than 0.5
% inputs computed in the previous MPC iteration
% Note that the inputs are the setpoints to the gas flowrate PID's
fic104sp = P_vector(5);
fic105sp = P_vector(6);
fic106sp = P_vector(7);

%current inputs of the plant
u0old=[P_vector(5);P_vector(6);P_vector(7)];

% cropping the data vector
nd = size(I_vector,2);
dataCrop = (nd - BufferLength + 1):nd;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MOST RECENT VALUE IS THE LAST ONE! %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% liquid flowrates [L/min]
fi101 = I_vector(1,dataCrop);
fi102 = I_vector(2,dataCrop);
fi103 = I_vector(3,dataCrop);

% actual gas flowrates [sL/min]
fic104 = I_vector(4,dataCrop);
fic105 = I_vector(5,dataCrop);
fic106 = I_vector(6,dataCrop);
```

```
% pressure @ injection point [mbar g]
pi105 = I_vector(7,dataCrop);
pi106 = I_vector(8,dataCrop);
pi107 = I_vector(9,dataCrop);

% reservoir outlet temperature [oC]
ti101 = I_vector(10,dataCrop);
ti102 = I_vector(11,dataCrop);
ti103 = I_vector(12,dataCrop);

% DP @ erosion boxes [mbar]
dp101 = I_vector(13,dataCrop);
dp102 = I_vector(14,dataCrop);
dp103 = I_vector(15,dataCrop);

% top pressure [mbar g]
% for conversion [bar a]-->[mbar g]
% ptop_n = ptop*10^-3 + 1.01325;
pi101 = I_vector(16,dataCrop);
pi102 = I_vector(17,dataCrop);
pi103 = I_vector(18,dataCrop);

% reservoir pressure [bar g]
% for conversion [bar g]-->[bar a]
% ptop_n = ptop + 1.01325;
pi104 = I_vector(19,dataCrop);

% number of measurements in the data window
dss = size(pi104,2);

% check for first iteration
if ~exist('h','var')
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % BE CAREFUL - it should match sampling time in LABVIEW interface %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Optimization sampling time
    nExec = 60; %[s]

    % Modifier Adaptation
    % Filters; high values mean less information about previous iterations
    K = 0.55; % filter on u
```

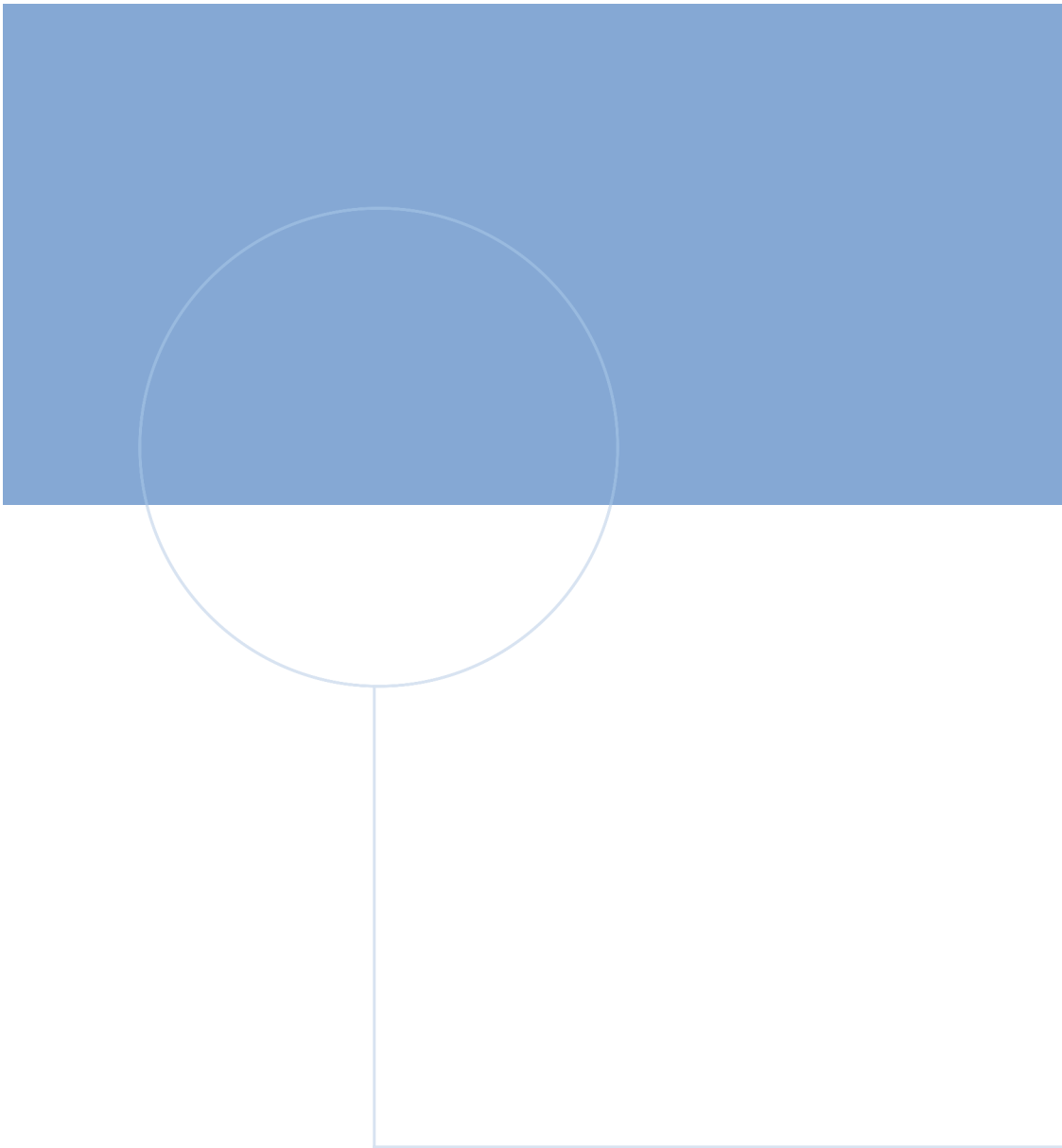
```
    % Modifiers
    lam_k = zeros(3,3);
    eps_k = zeros(3,1);

    % Finite difference step
    h = 1e-8;

    % Define initial
    count = 1;
    intitial_inputs = [3, 2.5; % u1
                       2, 3; % u2
                       2.5, 2]; % u3

end

% relevant measurements are only the liquid flowrates
yPlant = [fi101;fi102;fi103];

uPlant = [fic104;
          fic105;
          fic106;
          cv101*ones(1,dss); %workaround - just have the last measurement
↪  here. Since it is the disturbance, it doesn't really matter
          cv102*ones(1,dss);
          cv103*ones(1,dss)];

% assume that half of the period between the current and past RTO
↪  execution is at SS (mean filtering)
uEst = mean(uPlant(:,end - nExec/1.5:end),2); % 40 seconds average
yEst1 = mean(yPlant(:,50:end),2); % 10 seconds average

y_model = ssModel(u0old,[cv101;cv102;cv103]);

if count <= 2
    % Pass setpoints to the rig
    O_vector = intitial_inputs(:,count)';

    count = count + 1;
```

```
    % Dont mind what these variables are called. These variables can be
↪    obtained
    % from the rig
    SS = 1;
    Estimation = 0;
    Optimization = 0;
    Result = 0;
    Parameter_Estimation = [u0old(1), u0old(2), u0old(3),yEst1(1)-y_model(1), ...
    yEst1(2)-y_model(2),yEst1(3)-y_model(3)];
    State_Variables_Estimation = [0,0,0,cv101,cv102,cv103];
    State_Variables_Optimization = [0,0,0,0,0,0];
    Optimized_Air_Injection = [yEst1(1),yEst1(2),yEst1(3)];

else
    fileRTO = 'C:\ErosionRig\LabVIEW\Log\RTOlog\RTO.txt';
    Bs = [];
    B = [];
    fileID = fopen(fileRTO);

    %scaning the data file to find the number of data points
    Mread=textscan(fileID,'%s','delimiter','\n');
    %compacting everything in a string
    Mlin=string(Mread{:});
    %M is the number of data points (changes from file to file) header is
    %included
    M=length(Mlin);

    %Move file position indicator to beginning of open file
    frewind(fileID);

    for i=1:M
        B_text = textscan(fileID,'%s',25,'Delimiter',' ');
        Baux=string(B_text{:});
        Baux=strrep(Baux,',','.');
        Bs=[Bs Baux];

        if i >= 2
            B=[B str2double(Bs(1:25,i))];
        end
    end
```

```
fclose(fileID);

B = B';

u1_observed = [B(:,5),B(:,14);u0old(1),cv101];
u2_observed = [B(:,6),B(:,15);u0old(2),cv102];
u3_observed = [B(:,7),B(:,16);u0old(3),cv103];

y1_observed = [B(:,8);yEst1(1)-y_model(1)];
y2_observed = [B(:,9);yEst1(2)-y_model(2)];
y3_observed = [B(:,10);yEst1(3)-y_model(3)];

% Set the mean function, covariance function and likelihood function
meanfunc1 = [];
covfunc1 = @covSEiso;
likfunc1 = @likGauss;
% Initialization of hyperparameters
hyp1 = struct('mean', [], 'cov', [0 0], 'lik', -1);
% Optimization of hyperparameters
hyp1_opt = minimize(hyp1, @gp, -100, @infGaussLik, meanfunc1, covfunc1, ...
likfunc1, u1_observed, y1_observed);

% Set the mean function, covariance function and likelihood function
meanfunc2 = [];
covfunc2 = @covSEiso;
likfunc2 = @likGauss;
% Initialization of hyperparameters
hyp2 = struct('mean', [], 'cov', [0 0], 'lik', -1);
% Optimization of hyperparameters
hyp2_opt = minimize(hyp2, @gp, -100, @infGaussLik, meanfunc2, covfunc2, ...
likfunc2, u2_observed, y2_observed);

% Set the mean function, covariance function and likelihood function
meanfunc3 = [];
covfunc3 = @covSEiso;
likfunc3 = @likGauss;
% Initialization of hyperparameters
hyp3 = struct('mean', [], 'cov', [0 0], 'lik', -1);
% Optimization of hyperparameters
hyp3_opt = minimize(hyp3, @gp, -100, @infGaussLik, meanfunc3, covfunc3, ...
likfunc3, u3_observed, y3_observed);
```

```
% Obtain values for finite differences
[GP1,~] = gp(hyp1_opt, @infGaussLik, meanfunc1, covfunc1, likfunc1, ...
u1_observed, y1_observed, [u0old(1),cv101]);
[GP2,~] = gp(hyp2_opt, @infGaussLik, meanfunc2, covfunc2, likfunc2, ...
u2_observed, y2_observed, [u0old(2),cv102]);
[GP3,~] = gp(hyp3_opt, @infGaussLik, meanfunc3, covfunc3, likfunc3, ...
u3_observed, y3_observed, [u0old(3),cv103]);

[GP1h,~] = gp(hyp1_opt, @infGaussLik, meanfunc1, covfunc1, likfunc1, ...
u1_observed, y1_observed, [u0old(1)+h,cv101]);
[GP2h,~] = gp(hyp2_opt, @infGaussLik, meanfunc2, covfunc2, likfunc2, ...
u2_observed, y2_observed, [u0old(2)+h,cv102]);
[GP3h,~] = gp(hyp3_opt, @infGaussLik, meanfunc3, covfunc3, likfunc3, ...
u3_observed, y3_observed, [u0old(3)+h,cv103]);

% Update modifiers
lam_k(1,1) = (GP1h - GP1)/h;
lam_k(2,2) = (GP2h - GP2)/h;
lam_k(3,3) = (GP3h - GP3)/h;

eps_k(1) = GP1;
eps_k(2) = GP2;
eps_k(3) = GP3;

% Symbols to the optimization problem
u = MX.sym('u',3); % FIC104SP, FIC105SP, FIC106SP
y = MX.sym('x',3); % FI101, FI102, FI103

% Ss model
ss_model_well1 = theta_well1(1) + theta_well1(2)*u(1) + theta_well1(3)*uEst(4) ...
        + theta_well1(4)*u(1)^2 + theta_well1(5)*uEst(4)^2 - y(1);
ss_model_well2 = theta_well2(1) + theta_well2(2)*u(2) + theta_well2(3)*uEst(5) ...
        + theta_well2(4)*u(2)^2 + theta_well2(5)*uEst(5)^2 - y(2);
ss_model_well3 = theta_well3(1) + theta_well3(2)*u(3) + theta_well3(3)*uEst(6) ...
        + theta_well3(4)*u(3)^2 + theta_well3(5)*uEst(6)^2 - y(3);

% Modify output
y_m_k = y + eps_k + lam_k*(u-u0old);

J = 20*y_m_k(1) + 10*y_m_k(2) + 30*y_m_k(3);
```

```
    gas_constraint = u(1) + u(2) + u(3);

    nlp = struct('x', [u;y], 'f', -J, 'g', [ss_model_well1;ss_model_well2; ...
    ss_model_well3;gas_constraint]);
    solver = nlpsol('solver','ipopt',nlp);

    sol = solver('x0', [u0old;yEst1], 'lbx', [1;1;1;0;0;0], 'ubx', ...
    [5;5;5;inf;inf;inf], 'lbg', [0;0;0;0], 'ubg', [0;0;0;7.5]);
    opt = full(sol.x);
    u_opt = opt(1:3);

    J_opt = -full(sol.f);

    u_opt = u0old + K*(u_opt-u0old);

    % Pass setpoints to the rig
    O_vector = u_opt';

    % Dont mind what these variables are called. These variables can be
↪   obtained
    % from the rig
    SS = 1;
    Estimation = lam_k(1,1);
    Optimization = length(y1_observed);
    Result = J_opt;
    Parameter_Estimation = [u0old(1), u0old(2), u0old(3),y1_observed(end), ...
    y2_observed(end),y3_observed(end)]; % training data
    State_Variables_Estimation = [hyp1_opt.cov(1),hyp1_opt.cov(2),hyp1_opt.lik, ...
    cv101,cv102,cv103]; % GP1 and observed v_o
    State_Variables_Optimization = [hyp2_opt.cov(1),hyp2_opt.cov(2),hyp2_opt.lik, ...
    hyp3_opt.cov(1),hyp3_opt.cov(2),hyp3_opt.lik]; % GP2 and GP3
    Optimized_Air_Injection = [yEst1(1),yEst1(2),yEst1(3)];
end
```



NTNU

Norwegian University of
Science and Technology