TKP4580 - Chemical Process Technology, Specialization project

Extremum Seeking Control: Implementation on Experimental Lab Rig

Frida Bakken Myrvang

Submission date: 18.12.2020
Supervisor: Johannes Jaschke, IKP
Co-supervisor Jose Otavio Assumpcao Matias, IKP

NTNU
Department of Chemical Engineering
Norwegian University of Science and Technology

Abstract

The purpose of this specialization project is to investigate the opportunity to implement least square extremum-seeking control (LSESC) to a gas lifted oil network, on lab scale. LSESC is a purely data-driven, unconstrained, optimization method, which only relies on measurements of the inputs and the cost function, but that require tuning of some parameters. It is an alternative to traditional model-based optimization, such as Real Time Optimization (RTO). RTO brings up some challenges related to obtaining and updating models, among other things, which LSESC can resolve.

In this specialization project two case studies are performed. First, the optimization method is applied to a simple toy example, with one input, in order to study the tuning parameters. Some guidelines for the parameter tuning were obtained. Next, the method is applied to a model that represent the gas lifted oil network on lab scale. The first challenge, in this case, is that the system has three inputs, that is, the gas lift rate in each of the three wells, which makes the tuning more challenging. In addition, the system has a constraint on the total gas available for gas lift, and a minimum and maximum limit on the inputs, so the second challenge is constraint handling with an unconstrained optimization method. To handle the constraints, active constraint control, along with LSESC and some logic, is used in a step-by-step approach. The simulation results show that this method was able to drive the system to its optimum, without violating the constraints. After the preliminary testes in simulations, LSESC is merited to be implemented in the lab rig.

Table of Contents

Table of Contents v

List of Tables vii

List of Figures x

1 Introduction 1

1.1 Specialization Project Goals 3

2 Theory and Background 5

2.1 Production Optimization 5

2.2 Extremum-Seeking Control 6

2.2.1 Classical ESC 6

2.3 Least Square Extremum-Seeking Control 7

2.3.1 Least Square Estimation 9

2.3.2 Parameter Tuning 10

2.4 Constraint Handling in Extremum Seeking Control 11

3 Case Study 1 - Simplified Bioreactor 13

3.1 Methodology 13

3.1.1 The Model 13

3.1.2 Optimization Using LSE 14

3.1.3 Simulation 15

3.2 Results 17

3.2.1 Base Case 17

3.2.2 Changing Amplitude of Perturbation 18
3.2.3 Changing Frequency of Perturbation 19
3.2.4 Changing Buffer Length 20
3.2.5 Changing Integral Gain 20
3.2.6 Using PBRS signal as dither 21
3.3 Discussion 23
3.3.1 The Control Parameters 23
3.3.2 Applying the Method to a Real Process 24
3.4 Conclusion 25

4 Case Study 2 - Gas Lifted Oil Well Network **27**
4.1 System Description 27
4.1.1 The Model 28
4.2 The Optimization Method 29
4.2.1 The Optimization Problem 29
4.2.2 The Dither 30
4.2.3 The Control 30
4.2.4 The Constraint Handling 31
4.3 Results 34
4.3.1 Tuning Parameters 36
4.4 Discussion 37
4.4.1 The Simulation Result 37
4.4.2 The Tuning 38
4.4.3 Applying the Method to Other Processes 39
4.4.4 Implementation on Lab Rig 39
4.5 Conclusion 40

5 Conclusion **41**

A Gas Lifted Wells Model **45**
A.1 ErosionRigDynModelGrad.m 46

B MATLAB Code: Case study 1 **51**
B.1 caseStudy1.m 51
B.2 Plant.m 55
B.3 LSE.m 57

C MATLAB Code: Case study 2 **59**
C.1 Main.m 59
C.2 LabViewES7.m 65
C.3 LSE2.m 70
C.4 ResultsPlotting.m 70
C.5 ParametersGasLiftModel.m 72
C.6 InitialConditionGasLift3.m 73
C.7 InitializationLabViewRTO.m 74

List of Tables

3.1 Tuning for the nominal case and the range of values studied for each parameter. 16
3.2 Guidelines for the tuning parameters in LSESC 25

4.1 Tuning parameters for Case Study 2 36

List of Figures

2.1 Block diagram of the classical extremum-seeking control method. 7
2.2 Visualization of the linear model fit of the objective function as a function of the inputs, from the last N samples of data. 8
2.3 Block Diagram of LSESC 9

3.1 Flowsheet of the biochemical reactor 14
3.2 Simulation results for the objective function and the manipulated variable for the nominal case. 17
3.3 Simulation results for the objective function and the manipulated variable for different values of the amplitude of the perturbation. 18
3.4 Simulation results for the objective function and the manipulated variable for different values for the frequency of the sinusoidal perturbation. . . . 19
3.5 Simulation results for the objective function and the manipulated variable for different values of the buffer length. 20
3.6 Simulation results for the objective function and the manipulated variable for different values of the integral gain. 21
3.7 Simulation results for the objective function and the manipulated variable using PRBS dither with same tuning as the sinusoidal perturbation. 22
3.8 Simulation results for the objective function and the manipulated variable using PRBS dither with N = 50. 22

4.1 Simple figure of the gas lifted well network. w_{gl_i} is the gas lift rate in well i and $w_{ro,tot}$ is the total oil production from the 3 wells. 28

4.2 Flow diagram for the constraint handling. u_L is the MV with largest flow, while u_{s1} and u_{s1} are the to other MVs with smaller flows. u_{min} and u_{max} correspond to $w_{gl,i}^{min}$ and $w_{gl,i}^{min}$ respectively. 33
4.3 Simulation results for disturbance scenario 1. 34
4.4 Simulation results for disturbance scenario 2. 35
4.5 Simulation result of w_{gl_2} and w_{gl_3} around the time the disturbance occur, in scenario 2. 38

Nomenclature

Acronyms

CV	Controlled variable
DAE	Differential algebraic equations
DOF	Degree of freedom
ESC	Extremum-seeking control
LSE	Least-square estimation
LSESC	Least-square extremum-seeking control
MV	Manipulated variable
NLP	Nonlinear programming
PID	Proportional–integral–derivative
PRBS	Pseudo binary random sequence
RTO	Real time optimization
SOC	Self-optimizing control

Symbol	*Definition*	*Unit*
General:		
a	Amplitude of dither signal	
$\mathbf{d}$	Disturbances	
$\mathbf{f}$	Steady-state model equations	
$\mathbf{g}$	Operational constraints	
J	Objective function	
$\hat{\mathbf{J}}_{\mathbf{u}}$	Estimated gradients form the inputs to J	
K_I	Integral gain	
N	Buffer length	
u	Input	
ω	Frequency of dither signal	
$\mathbf{\Phi}$	Regressor vector	
$\boldsymbol{\theta}$	Estimated parameters	

Case Study 1:

D	Dilution rate	h^{-1}
D_{min}	Minimum dilution rate	h^{-1}
D_{max}	Maximum dilution rate	h^{-1}
F	Volumetric flow rate	m^3/h
k_m	Monod saturation constant	$\frac{\text{mass of cells}}{\text{volume}}$
V	Volume of bioreactor	m^3
x_1	Biomass concentration	$\frac{\text{mass of cells}}{\text{volume}}$
$x_{1,f}$	Biomass concentration in feed	$\frac{\text{mass of cells}}{\text{volume}}$
x_2	Substrate concentration	$\frac{\text{mass of cells}}{\text{volume}}$
$x_{2,f}$	Substrate concentration in feed	$\frac{\text{mass of cells}}{\text{volume}}$
Y	Yield current	$\frac{\text{mass of cells produced}}{\text{mass of substrate consumed}}$
μ	Specific growth rate	h^{-1}
μ_{max}	Maximum specific growth rate	h^{-1}
ψ	Production rate of cells	$\frac{\text{mass of cells}}{\text{volume}\cdot h}$

Case Study 2:

m_{g_i}	Gas hold up	kg
m_{o_i}	Oil hold up	kg
p_{rh_i}	Riser head pressure	bar
p_{bh_i}	Pressure below injection point	bar
w_{gl_i}	Gas lift rate in well i	L/min
$w_{gl,max}$	Total gas lift capacity	L/min
$w_{gl_i}^{max}$	Maximum limit on gas lift in one well	L/min
$w_{gl_i}^{min}$	Minimum limit on gas lift in one well	L/min
w_{gr_i}	Riser head gas production rate	kg/s
w_{lr_i}	Riser head liquid production rate	kg/s
w_{pr_i}	Riser head total production rate	kg/s
w_{ro_i}	Oil rate from reservoir	kg/s
ρ_{r_i}	Mixture density in riser	kg/m^3
ρ_{gr_i}	Density gas	kg/m^3
α_o	Price of oil	$\frac{\$}{\text{kg/s}}$
α_{gl}	Cost of compressing gas for gas lift injection	$\frac{\$}{\text{L/min}}$

Chapter 1

Introduction

Production optimization is a technique that generally seeks to maximize the production, while minimizing the cost of production. Optimization of a process involves decision making in separated time scales to achieve short, medium, and long term objectives. This specialization project will focus on the medium term objectives. They often relate to maximizing daily profit, and the decisions are typically taken in the time scale of hours to days. In current practice, medium-term production optimization is addressed by Real-time optimization (RTO). Traditionally, in RTO, a nonlinear steady-state model, which describes the process, is used to solve an optimization problem online. Most RTO systems also require a model adaption step to update the model parameters, such as feed compositions and efficiencies [5].

Given that you have a perfect model and use an approptiate solver for the optimization problem, the inputs computed by the RTO match the plant optimal inputs. However, there are some drawbacks of the model-based optimization, pointed out in [8], some of them are listed below:

1. In complex processes, making a model for properly representing the behavior of the system can be difficult and time consuming. Additionally it requires a very good understanding of the process.

2. The model needs to be adapted during operation, to ensure that it reflects the current plant behavior, and be compared against online measurements from the process. If the model is not updated, the optimization is based on an inaccurate model, and the inputs computed by the RTO are likely to be sub-optimal.

3. If steady state models are used, the model can not be updated before the plant has

reached steady-state. The process operates sub-optimally during the steady-state wait time.

4. If dynamic optimization is used (this solves item 3), the method requires high computational capacity. Even with todays computers, this is a challenge.

5. The method that is used to solve the optimization problem, affects the computational capacity demand, which is also influenced by the model complexity. There is a trade off on how accurate the problem is solved, how complex the model is, and how much computational capacity required to solve the problem.

In summary, modelling the process can be difficult, time consuming and an ongoing task as conditions change over time. This, along with the computational issues, make model-based optimization expensive [8].

On the other hand, model free optimization methods can be used. They are purely data-driven and do not use a model to optimize the process, but measurements from the process. One of these methods is *Extremum Seeking Control* (ESC) [1]. In ESC, the objective function of the optimization problem needs to be measured. These measurements, along with measurements of the inputs, are used to optimize the process. The gradient of the objective function with relation to the input is estimated, and driven to zero. The objective function is then at a minimum or a maximum [1], which yields optimal operation.

Using ESC can solve some of the problems with the traditional RTO approach, mentioned above. Since there is no need for a model, the problems regarding obtaining and updating the model are gone. In addition, there is no steady-state wait time, since ESC do not require steady state before updating the process. Also, estimating a gradient require much less computational capacity, than solving a non-linear, model-based, optimization problem.

As traditional RTO, ESC also has some challenges:

1. The method requires accurate measurements of the inputs and the cost function, since they affect the gradient estimation and, consequently, the optimization performance.

2. ESC have some tuning parameters, which can be challenging to tune. Simulations of the process optimization is most likely necessary to obtain the right tuning, which will require a model. If the goal is to avoid making a model, the tuning is even more challenging.

3. ESC is in principle an unconstrained optimization method, so constraint handling can be a challenge.

1.1 Specialization Project Goals

The goal of this specialization project is to better understand ESC, and investigate the possibility of implementing it on an experimental lab rig. First, the tuning parameters that effect the performance are studied in a simple system with only one input. Next, ESC is applied to a system, in which a dynamic model is used for representing the experimental lag rig of interest. In this second case study, we investigate how to handle constraints in ESC implementations.

Chapter 2

Theory and Background

2.1 Production Optimization

In production optimization the objective is to maximize the profit. This is done by maximizing the production and minimizing the cost of production, while satisfying the operational constraints. In current practice, production optimization is addressed by RTO. In traditional RTO, the optimization requires [12]

1. An economic objective function to be maximized or minimized, that includes costs and product values.
2. The operating model, which includes a steady-state process model and all constraints on the process variables .

The general form of an optimization problem includes a nonlinear objective function and nonlinear constraints, and is called a nonlinear programming (NLP) problem [12]. In this specialization project, a process where the optimal operating point can be described by the following optimization problem is considered

$$\begin{aligned} & \min_{\mathbf{x},\mathbf{u}}/\max_{\mathbf{x},\mathbf{u}} && J(\mathbf{x},\mathbf{u},\mathbf{d}) \\ \text{s.t} & && \\ & && \mathbf{f(x,\,u,\,d)} = 0 \\ & && \mathbf{g(x,\,u,\,d)} \leq 0 \end{aligned}$$

where J is the objective function, **f** and **g** represent the equality and inequality constraints, respectively. **f** is the steady-state model equations and **g** represents operational limitations for the process. **x** is the vector of state variables, **u** is the vector of manipulated variables

used to optimize J and **d** is the vector of the disturbances. Together **x**, **u** and **d** represent the process variables.

2.2 Extremum-Seeking Control

Extremum-seeking control (ESC) is a model free and adaptive control approach. The method finds a local maximum or minimum of the objective function, despite disturbances, varying system parameters and non-linearity. The objective function is assumed to be a static map [3]. In ESC, there is no need for the model equations, **f**, so the optimization problem can be written as

$$\begin{aligned} & \min_{\mathbf{u}}/\max_{\mathbf{u}} \quad J(\mathbf{u},\mathbf{d}) \\ \text{s.t} \quad & \\ & \mathbf{g(u, d)} \leq 0 \end{aligned}$$

The method finds the inputs, **u**, so that the objective function,J, is at an extremum point. For unconstrained problems, this is done by driving the estimated gradients from the inputs to the objective function, $\mathbf{J_u}$, to zero. When the gradients are zero, the objective function is at a minimum or maximum, which yields optimal operation. The gradients are estimated by perturbing the inputs, and observing how they affect the objective function [3].

Given that there are n_u inputs, the size of the objective function gradient with respect to inputs is (1 x n_u), and can be written as

$$\mathbf{J_u} = [J_{u,1} \;\; J_{u,2} \; ... \; J_{u,n_u}] \tag{2.1}$$

By using measurements we can compute the gradient estimate $\hat{\mathbf{J}}_\mathbf{u}$. Then, n_u integral controllers can be used to drive the gradients to zero. Integral control is used because the control should be slow. In discrete time, the integral control on the input can be written as

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \mathbf{K}_I \hat{\mathbf{J}}_\mathbf{u} \tag{2.2}$$

where $\mathbf{K}_I$ is the diagonal gain matrix. The integral gain determines how aggressively the input changes [3].

2.2.1 Classical ESC

There are different extremum-seeking control approaches. The classical approach is adding a sinusoidal perturbation to the input. The sinusoidal perturbation added to the input can

be written as

$$u_{k+1} = u_k + asin(\omega t) \tag{2.3}$$

where a and w are the amplitude and frequency of the sinusoidal perturbation, respectively.

The perturbed input is implemented in the plant. This results in a sinusoidal output perturbation and a sinusoidal cost function, J, that oscillates around the mean. The mean of J is non-zero, so in order to subtract out the mean, J is passed through a high-pass filter. After this, the input sinusoidal is multiplied to the signal, resulting in a demodulated signal [3]. A low-pass filter is then applied to remove the noise of the signal, and we end up with a signal, ξ [10]. This signal is an approximation of the gradient, and the sign of ξ indicates how the input should be changed in order to move towards the optimal input [10]. A block diagram of the classical ESC approach is shown in Figure 2.1.

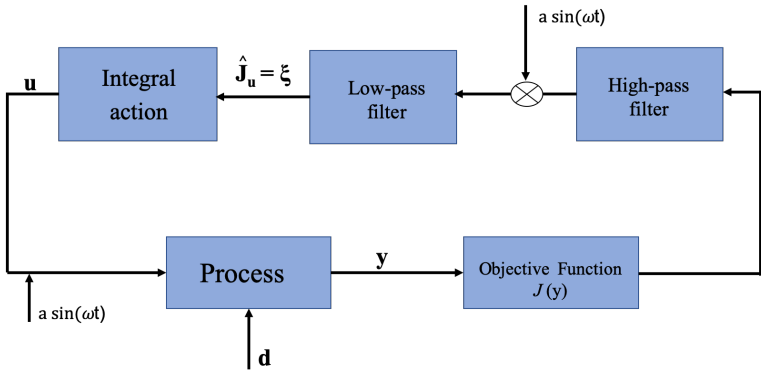

Figure 2.1: Block diagram of the classical extremum-seeking control method.

2.3 Least Square Extremum-Seeking Control

An alternative approach, to the classical ESC, is least square extremum seeking control (LSESC) [7]. In LSESC, least square estimation is used to estimate the gradients from the inputs, **u**, to the objective function, J [7]. Using this method will result in fewer tuning parameters, since there are no high and low pass filters that require tuning. In addition there can be used other dither signals than a sinusoidal perturbation to excite the system. This could possibly make the LSESC approach more applicable and relevant when implementing the method in a chemical process, due to their inertia and relatively large time constants. Therefore, this approach is used in the specialization project.

In LSESC, a moving window of the last N samples of data measurements of the ob-

jective function and the inputs is used to estimate the gradients [15]. A moving window is used to compute and average and mitigate the influence of noisy measurements. **J** is the vector of the last N samples of the measured values of objective function, and **U** is the last N samples of the measured inputs. At a time k they are given by the following

$$\mathbf{J}_k = [J_k \;\; J_{k-1} \; ... \; J_{k-N+1}]^T \tag{2.4}$$

$$\mathbf{U}_k = [\mathbf{u}_k \;\; \mathbf{u}_{k-1} \; ... \; \mathbf{u}_{k-N+1}]^T \tag{2.5}$$

At every time step, these two buffers are used to fit a linear model of the objective function, as seen from Figure 2.2. The linear model estimation is given by the following equation

$$J = \hat{\mathbf{J}}_\mathbf{u}^T \mathbf{u} + \hat{m} \tag{2.6}$$

where $\hat{\mathbf{J}}_\mathbf{u}$ is the vector of the estimated gradients from **u** to J, and $\hat{m}$ is the bias.

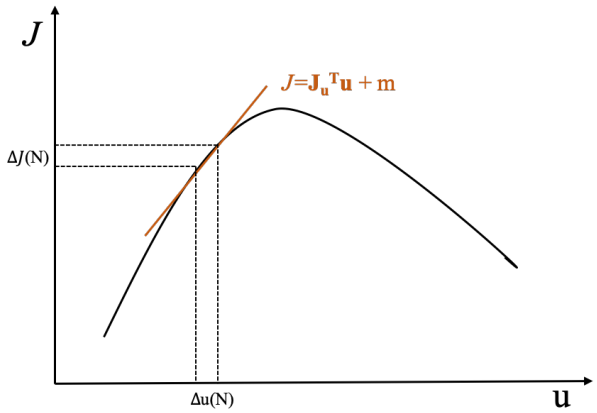

Figure 2.2: Visualization of the linear model fit of the objective function as a function of the inputs, from the last N samples of data.

To estimate $\mathbf{J}_\mathbf{u}$,a dither signal is added to the input. The dither is added in order to observe how input changes affect the objective function. It guarantees that **u** is sufficiently well conditioned to obtain accurate gradient approximations. The most common one in ESC is a sinusoidal perturbation. In LSESC more general dithers can be added [7], such as a PBRS or a square wave. A Block Diagram of LSESC is showed in Figure 2.3.

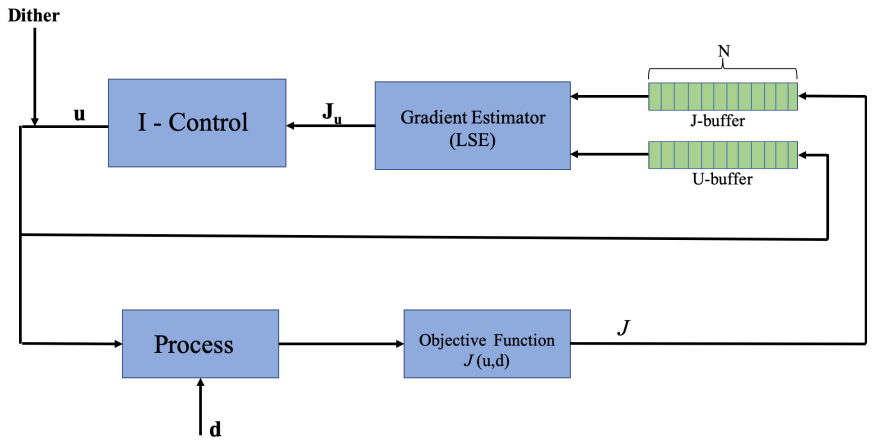

Figure 2.3: Block Diagram of LSESC

2.3.1 Least Square Estimation

Least square estimation (LSE) is a standards approach that is used to fit a set of data to a model. The method is based on minimizing the sum of square errors, that is the 'distance' from the observed values and the estimated values. For a sample with N observations and r unknown model parameters, a general regression model, in matrix form, can be written as [6]

$$\begin{bmatrix} y_1 \\ y_2 \\ \\ y_N \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 & 1 \\ \mathbf{x}_2 & 1 \\ & \\ \mathbf{x}_N & 1 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \\ \theta_r \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \\ e_N \end{bmatrix} \tag{2.7}$$

where y_i is the model output for observation i, $\mathbf{x}_i$ is the vector of the (1xr) dependant model variables for observation i and e_i is the residual between the estimated model and observation i, where $i \in (1, ..., N)$. θ_j is the unknown model parameter for the dependant model parameter x_j, where $j \in (1, ..., r)$.

By introducing $\mathbf{\Phi}$ as the regressor vector, as in [15], Equation 2.7 on matrix form is given by

$$\mathbf{Y} = [\mathbf{X} \;\; 1]\, \boldsymbol{\theta} + \mathbf{e} = \Phi\boldsymbol{\theta} + \mathbf{e}$$

where $\mathbf{Y}$ is a vector of the model output for the N observations, $\mathbf{X}$ is the matrix of the m dependent model variables for the N observations, $\mathbf{e}$ is the residual between the model and

the observation for each of the N observations and $\boldsymbol{\theta}$ is the parameters to be estimated.

As pointed out in [15], $N > r$ to apply this method. The analytical solution to the least square problem above is given by Equation 2.8 [11].

$$\hat{\boldsymbol{\theta}} = [\Phi^T \Phi]^{-1} \Phi^T \mathbf{Y} \tag{2.8}$$

Connecting this notation with the notation from LSESC above, gives the following

$$\mathbf{Y} = \mathbf{J} \tag{2.9}$$

$$\Phi = [\mathbf{U} \;\; 1] \tag{2.10}$$

$$\hat{\boldsymbol{\theta}} = [\hat{\mathbf{J}}_{\mathbf{u}}^T \;\; \hat{m}]^T \tag{2.11}$$

2.3.2 Parameter Tuning

Implementation of LSESC require correct tuning of the parameters. The parameters that need tuning are the buffer length, N, the integral gains, $\mathbf{K}_I$, and the frequency, ω, and amplitude, a, of the perturbation signals added to the inputs.

In ESC there are three time scales

- Slow - convergence to the optimum
- Medium - perturbation frequency
- Fast - the controlled plant dynamics

In order to have a good extremum-seeking controller, there should be a clear time scale separation between these [15], and the parameters should be tuned accordingly. The timescales of interest are the slow and medium. Regulatory controllers, such as PIDs, are responsible for dealing with fast timescale disturbances.

The integral gain should be chosen small enough so that the timescale for the convergence is slower than the timescale for the perturbation signal. And the frequency of the perturbation signal must be significantly slower than the plant dynamics, this is necessary in order to approximate the plant as a static map [15]. In [15], it is also mentioned that in order to estimate the gradient of the objective function with respect to each input, for a multiple inputs system, the inputs should have different perturbation frequencies.

2.4 Constraint Handling in Extremum Seeking Control

ESC is in principle used for unconstrained optimization, but there are ways to handle constraint with this method. One solution is using constraint relaxation and penalise constraint deviations. The constrained problem is converted into an unconstrained problem by relaxing the constraints[4]. Thus, an optimization problem given by

$$\begin{aligned} &\min \quad J \\ \text{s.t} \quad & \\ &\quad g \leq 0 \end{aligned}$$

can be converted into

$$\min \quad J + w \cdot g \tag{2.12}$$

where w is some weighting factor, that penalises breaking the constraints.

Another alternative is using active constraint control, in addition to self-optimizing control (SOC) for the unconstrained degrees of freedom (DOF) [8]. In short, SOC is choosing CVs, that are controlled to a constant setpoint, in order to maintain close to optimal operation without the need of reoptimization when disturbances occur [13]. In this approach, the MV with the largest value is used to control the active constraint. Using a small MV to control the active constraint may quickly saturate, leading to constraint violation or suboptimal operation. The remaining MVs are used for SOC. [8].

The inputs that are not used to control the active constraint, can be used to drive the estimated gradients to zero. The following self-optimizing CVs are used,

$$CV_i := \hat{J}_{u,i-1} - \hat{J}_{u,i} \quad \forall i = 2, ..r \tag{2.13}$$

where r is the number of inputs in the system. The self-optimizing CVs are controlled to a constant setpoint of zero. By doing this, equal value of the gradients is achieved.

In this specialization project, the constraints are handled using the latter method. How this is implemented for a given case is further explained in Chapter 4.2.3.

Chapter 3

Case Study 1 - Simplified Bioreactor

In this chapter Least Square Extremum Seeking Control (LSESC) is applied on a simple biochemical reactor, with only one input. The model used in this case study is a slow process, the timescale is in hours, and extremum seeking control is not a good optimization method for a biochemical reactor. The goal of this case study is not to try and implement extremum seeking on a bioreactor in real life, but to use the model as a toy example to study how the tuning parameters of the method affect the optimization result. As mentioned in Section 2.3.2, the tuning of the parameters is very important for the performance of the optimization.

3.1 Methodology

3.1.1 The Model

The bioreactor is modeled as a CSTR with two components, substrate and biomass. A flowsheet of the process is shown in Figure 3.1. The dynamic model for the biochemical reactor is given by the following two equations, which are derived from the total and component mass balances, as presented in [2].

$$\frac{dx_1}{dt} = Dx_{1,f} - Dx_1 + \mu x_1 \tag{3.1}$$

$$\frac{dx_2}{dt} = Dx_{2,f} - Dx_2 - \frac{\mu x_1}{Y} \tag{3.2}$$

Where the first term of the equations represents the inflow, the middle term the outflow and the last part represents consumption/generation. x_1 is the biomass concentration and x_2 is the substrate concentration given in $\frac{\text{mass of cells}}{\text{volume}}$ and $\frac{\text{mass of substrate}}{\text{volume}}$, respectively. $x_{1,f}$ and $x_{2,f}$ is the concentration of biomass and substrate in the feed. Y is the yield, which is calculated as the mass of cells produced per mass of substrate consumed. D is the dilution rate given by F/V, where F is the volumetric flow rate and V is the volume of the bioreactor. μ is the specific growth rate coefficient, in this case the Monod relation is used. The Monod relation is given by Equation 3.3, where k_m and μ_{max} are constants.

$$\mu_{monod} = \frac{\mu_{max} x_2}{k_m + x_2} \tag{3.3}$$

In this case study Y and V are assumed to be constant, to simplify the model. It is also assumed that there is no biomass in the feed stream, so $x_{1,f} = 0$.

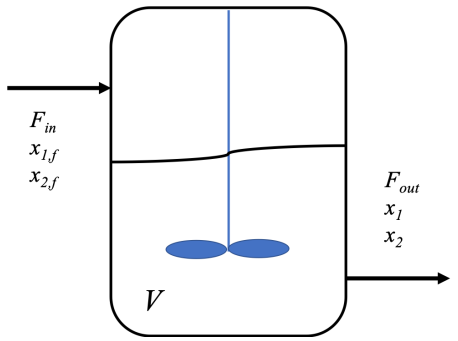

Figure 3.1: Flowsheet of the biochemical reactor

3.1.2 Optimization Using LSE

The economic objective in this case is to maximise the steady-state production rate of cells, ψ. Operational limitations of the reactor are represented by constraints on the dilution rate, D. The optimization problem is written as

$$\begin{aligned} &\max \psi = Dx_1 \\ \text{s.t} \\ &D_{min} \leq D \leq D_{max} \\ &x_1, x_2 \geq 0 \end{aligned}$$

The manipulated variable that is used to maximize ψ in this problem is the dilution rate. Thus, according to the notation from the theory in Chapter 2, the objective function, J, is ψ and the input, u, is D. Since we only have one input, there is no need for writing the equations in vector form.

In this case study we primarily look at a sinusoidal perturbation as the dither added to the input, which is the most commonly used perturbation in ESC. The input in discrete time can then be written as

$$u_{k+1} = u_k + u_k \cdot asin(\omega t) \tag{3.4}$$

where a and ω are the amplitude and period of the sinusoidal wave, respectively. The use of a signed pseudorandom binary sequence signal(PBRS) as dither, instead of a sinusoidal perturbation, was also studied. A PBRS is most likely easier to implement in a real process, due to its discrete nature which is more suitable to be used in computer-based systems. A signed PBRS is a pseudo random binary sequence, seq, of 1 and -1, that repeats itself after every w sampling times. Its amplitude can be adjusted by multiplying the sequence by a factor a. Then, the generated sequence is added to the input as in

$$u_{k+1} = u_k + a \cdot seq(k) \tag{3.5}$$

In both cases, an integral controller is used for adjusting the input, u, and driving the estimated gradient, $\hat{J}_u$, to zero. The integral controller can be written as

$$u_{k+1} = u_k + K_I \hat{J}_u \tag{3.6}$$

These values above, along with the buffer length, N, need to be tuned to be able to have good control of the process.

3.1.3 Simulation

In this case study, a set of tuning parameters is used for the nominal case. They were obtained by testing different values for each of the tuning parameters, until the LSESC gave good performance. In the nominal case, and to study the the tuning parameters, a sinusoidal perturbation was used as a dither. Using a PRBS dither instead of a sinusoidal perturbation was also studied.

To study how each parameter affect the control, one of the tuning parameters was varied at a time, and the other parameters were kept at their nominal values. Table 3.1 shows the tuning for the nominal case in addition to the studied parameters range. Simulations in

MATLAB were used to investigate how each parameter affects the LSESC performance. The code used for the simulations is in Appendix B.

Table 3.1: Tuning for the nominal case and the range of values studied for each parameter.

Parameter	Nominal Case	Values Studied
a	0.0001	[0.00001, 0.0001, 0.001, 0.01]
ω	$2\pi/5$	[$2\pi/10$, $2\pi/5$, $2\pi/3$, π]
N	100	[50, 100, 200]
K_I	0.0002	[0.00005, 0.0002, 0.001]

3.2 Results

In this section the simulation results are presented, beginning with the nominal case. Then the results for each of the parameter variations are presented in the following order, changes in the amplitude of the perturbation, changes in the frequency of the perturbation, changes in the buffer length and changes in the integral gain. Finally the result with PRBS as perturbation is presented.

3.2.1 Base Case

Figure 3.2 shows the simulation results for the nominal case. The figure show that LSESC is able to drive the input to the steady state optimal operating point, which was computed previously. There is a drop in the input as the control starts. The reason for this is poorly estimated gradients.

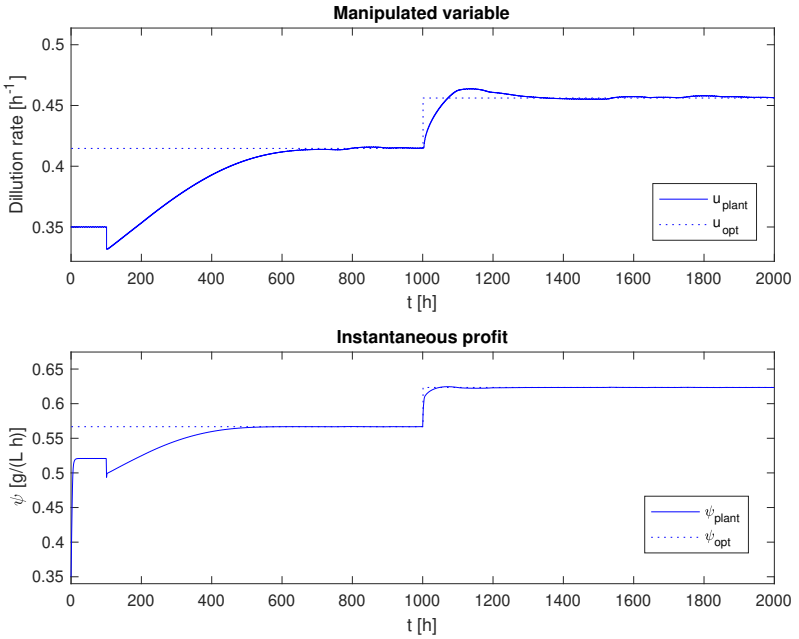

Figure 3.2: Simulation results for the objective function and the manipulated variable for the nominal case.

3.2.2 Changing Amplitude of Perturbation

In this section a change in the amplitude of the input sinusoidal perturbation signal is added. Figure 3.3 shows, that for larger amplitudes, the controller is not able to find the optimal operating point, the input keeps on increasing. This happens for both a=0.01 and a=0.001. The deviation increases with increased amplitude.

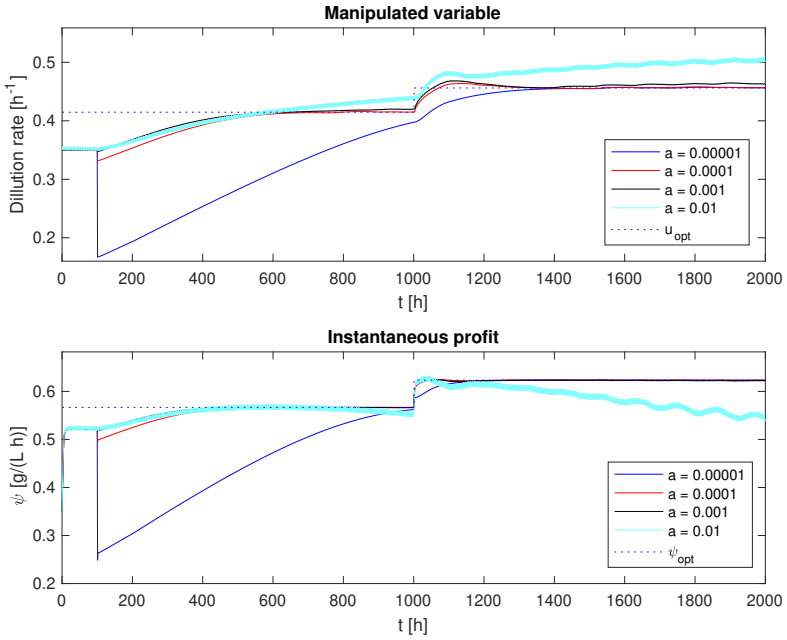

Figure 3.3: Simulation results for the objective function and the manipulated variable for different values of the amplitude of the perturbation.

With smaller amplitudes, a=0.00001 and a=0.0001 (the nominal case), the controller is able to find the optimal operating point. But, in this case, the control is slower, due to the significant decrease of the input value after the control starts at 100h. This variation of the inputs happens for all four simulations, but it is more significant with smaller amplitudes. The reason for this is the poorly estimated gradients.

The figure also shows that the simulation results for a=0.001 and a=0.0001 are quite similar. This indicates that the value for a is not so sensitive.

3.2.3 Changing Frequency of Perturbation

In this section, changes in the frequency of the sinusoidal perturbation are studied. Figure 3.4 shows that for higher frequencies, there is a larger decrease in the input as the control starts, similarly to the results in Figure 3.3. Consequently, higher frequencies lead to slower convergence to the optimum and more economic loss.

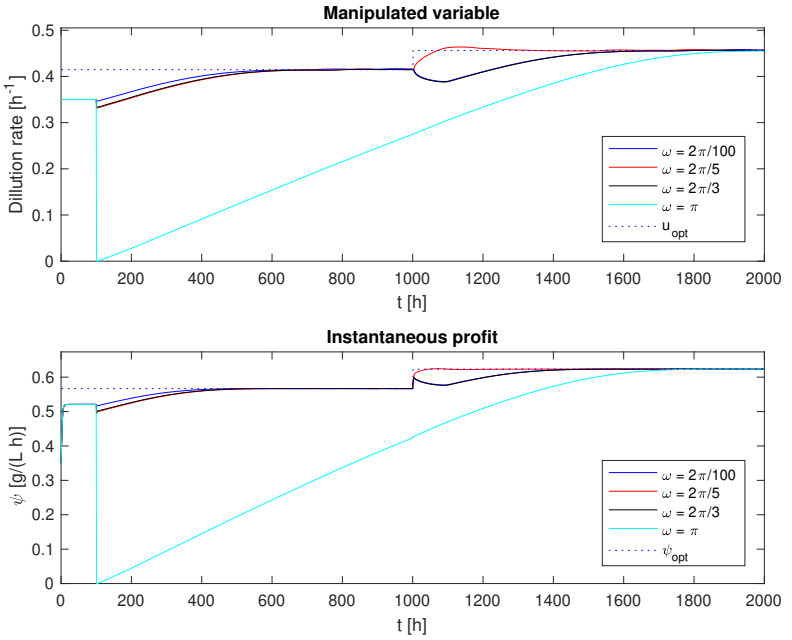

Figure 3.4: Simulation results for the objective function and the manipulated variable for different values for the frequency of the sinusoidal perturbation.

A decrease in the frequency results in faster control in the first part of the simulation interval, but after the disturbance occurs it adjusts the input in the wrong direction. The black line, ω=2π/3, follows the red line, the nominal case, in the first part and and the blue line, ω=2π/100, after the disturbance occurs.

Despite different performances with different frequencies, they all are able to reach the optimal operating point.

3.2.4 Changing Buffer Length

In this section, changes in the buffer length, N, are studied . Figure 3.5 shows that, by using different buffer lengths, the system gradients are properly estimated and drive the system to its optimal value, but different buffer lengths give different performances. Shorter buffer lengths give a bigger decrease in the input after the control starts. When long buffer lengths are used, the control use some time to change in the right direction when the disturbance occurs. This is because the gradient is to a greater extent calculated based on the past.

The shorter the buffer length is, the faster the control starts, since the buffer needs to be filled up for estimating the plant gradients.

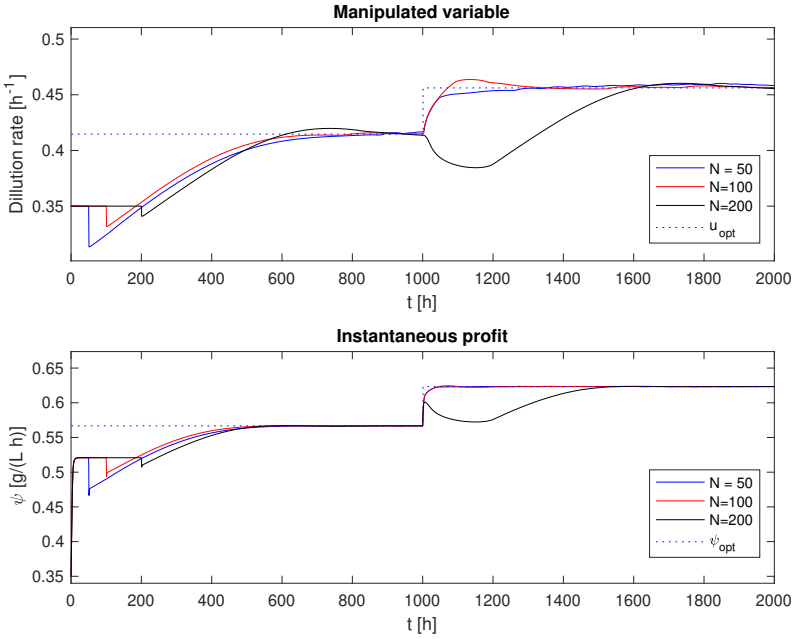

Figure 3.5: Simulation results for the objective function and the manipulated variable for different values of the buffer length.

3.2.5 Changing Integral Gain

In this section, the integral gain, K_I is changed within the range shown in Table 3.1. The results can be seen in Figure 3.6. They show that, if the integral gain is large, the input will oscillate around the optimal value, but the controller is still able to damp the oscillations

and drive the system to its optimum. For integral gains smaller than the base case, the control is slower, i.e. the input moves slower towards the optimum.

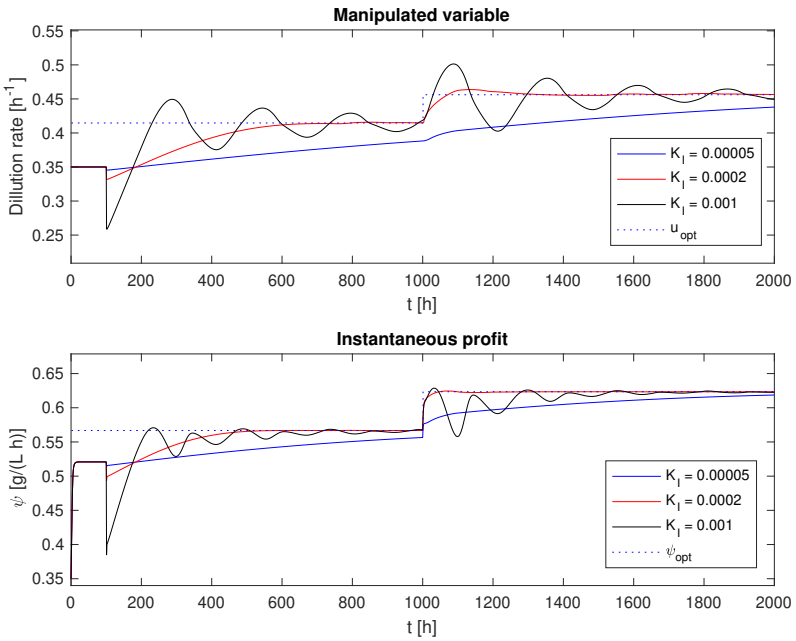

Figure 3.6: Simulation results for the objective function and the manipulated variable for different values of the integral gain.

3.2.6 Using PBRS signal as dither

In this section, a PBRS signal is used as a dither added to the input, instead of a sinusoidal perturbation. The amplitude, integral gain and buffer length are the same as for the sinusoidal perturbation. The results of the simulation are in Figure 3.7.

Figure 3.7 shows a decrease in the input when the disturbance occurs, similar to what we see in Figure 3.5, when the buffer length is a above the base case. Such behavior was a consequence of using long buffers for estimating the gradients. Therefore, the buffer length is decreased to N=50, instead of N=100. The results in Figure 3.8 show that the controller with this new tuning has a better performance.

The change in the performance is probably because the frequency for the PRBS perturbation is not the same as for the nominal case. The PRBS perturbation is added with $\omega = 1[h^{-1}]$, while the sinusoidal perturbation is added with $\omega = 2\pi/5$

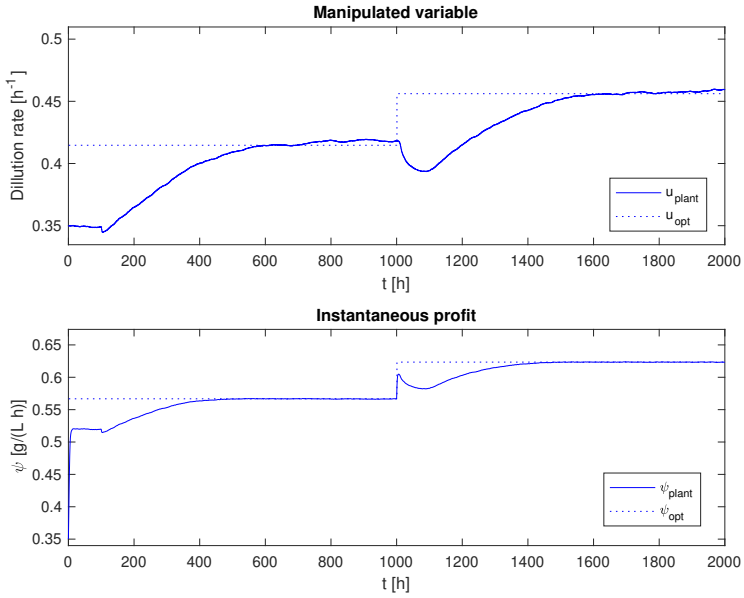

Figure 3.7: Simulation results for the objective function and the manipulated variable using PRBS dither with same tuning as the sinusoidal perturbation.

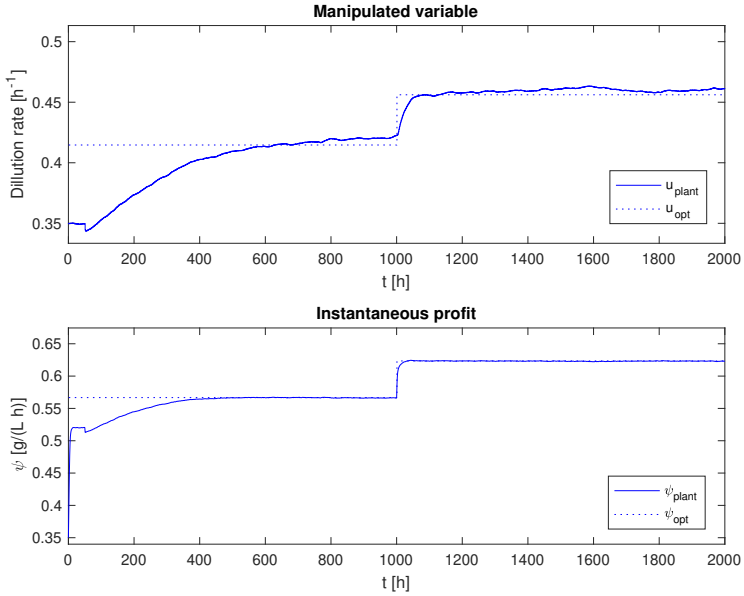

Figure 3.8: Simulation results for the objective function and the manipulated variable using PRBS dither with $N = 50$.

3.3 Discussion

3.3.1 The Control Parameters

From the results it is clear that the tuning parameters value affect the control performance.

Amplitude

Figure 3.3 shows that the amplitude should be small enough so it can find the optimal operating point, but not so small that the control becomes slow. The figure also shows that for smaller amplitude, there is a larger decrease in the input as the control starts.

Freqency

For the frequency, in Figure 3.4, it is hard to see any clear patterns for how the frequency should be tuned. Frequencies higher, ω=2π/100 and lower, ω=2π/3, than the base case give very similar results. For high frequencies, there is a decrease in the input as the control starts, and the control is therefore very slow.

Buffer Length

Figure 3.5 shows that if the buffer length is too long, there is a 'delay' of the response in the input as the disturbance occurs. With longer buffer lengths, more old data is saved in the buffer and used for the gradient estimation. Thus, when a change occur it will take some time for the gradient estimation to be updated to the current value.

A decrease in the input after the control starts occurs for all simulations, and it increases with smaller buffer lengths. One thing to possibly test could be starting the control after a certain time, smaller than the time to fill up the buffer length, to start the control earlier. Sub-optimal control in the start, before the buffer is filled up, can possibly be better than no control. How relevant this is depends on the time scale of the system. This is more relevant to investigate if the time for filling the buffer is 50h vs 100h, than if it is 50s vs 100s.

Integral Gain

For the integral gain, in Figure 3.6, there is a clear pattern. If the integral gain is to large there will be oscillation. The larger the integral gain, the bigger the oscillations. If the integral gain is too small, the control will be slow. Hence, for obtaining a good control performance the integral gain should be small for avoiding oscillations, but not so small

that the controller is too slow. The decrease in the input after the control starts increases with increased integral gain.

3.3.2 Applying the Method to a Real Process

For the control method to be implemented in a real process, there are some things that need to be considered. As seen from the results, some of the simulations have abrupt change in the input, especially after the control starts. This is a consequence of poor gradient estimation from the LSE. These changes can potentially damage equipment or represent a safety issue for the operation. Thus, this should be avoided and the control parameters should be tuned accordingly. One possible solution could be to have a constraint on how much u can change in one step, so that abrupt changes can be avoided.

Using this method in a real process will require that a dither is added to the input. How easy it is to add a dither to the input will depend on the process. The dither do not have to be a sinusoidal perturbation, it could also be a PBRS or a square wave. Figure 3.8 shows that using a PBRS dither yields a similar performance in the case study. The PRBS is easier to implemented in a real process.

Another consideration when implementing LSESC, is if it is desirable to continuously perturb the input and if it is possible to do it. In most cases a stable process and input is preferable. Processes that have continuously small perturbation in the input could possibly take advantage of this control method. The disturbance that perturbs the input continuously could be a replacement for the PRBS dither.

If this optimization method is going to be used in a real process, simulation on the process behavior would be beneficial, and maybe necessary, when tuning the control parameters. In order to simulate the process behavior, a model for the process is necessary. The motivation for using this optimization method is that it is model free, but if a model is necessary for tuning, there is no clear advantage in using LSESC. One solution for this is to use data measurements to approximate a model. There are several methods for doing this, such as *Dynamic Mode Decomposition*, *Kooperman Analysis* and *Sparse Identification of Nonlinear Dynamics* [3]. Then, the data-driven model can then be used for tuning the controller. But again, this model could also be used in optimization methods that require a model.

As the simulations show, extremum seeking is a relative slow optimization method. It is best to use if the controlled process has fast dynamics. From the simulations of the nominal case, in Figure 3.2, we see that the process reaches the optimal operating point after approximately 600h, which is 25 days. As mentioned in the intro, this system is not appropriate for ESC, but it was just used as a toy example to study the tuning parameters.

3.4 Conclusion

Tuning the parameters for least square extremum control can be difficult, experience with the method and the process facilitates this task. From this case study some guidelines have been found, they are presented in Table 3.2 below.

In a real process a sinusoidal perturbation can be hard to implement, an alternative is using a PRBS dither instead. Figure 3.8, show that this works in the simulations.

Table 3.2: Guidelines for the tuning parameters in LSESC

Parameter	Guideline
a	The value should be small enough so that the optimal operating point is obtained, but not so small that the control is too slow.
ω	It was difficult to see any clear guidelines for the frequency.
N	The length should be long enough so the LSE give a good estimate for the gradient, but not so long that there is too much old data saved in the buffer.
K_I	The value should be so small that oscillations in the input are avoided. It should not be too small, because a decrease in K_I gives slower control.

Chapter 4

Case Study 2 - Gas Lifted Oil Well Network

In this chapter least square extremum seeking control is applied to a gas lifted oil well network model, containing 3 wells. A model is used for representing a real system, on lab scale. The goal with this case study is to investigate the possibility to implement LSESC on the experimental lab rig that the model represents. The main challenges with applying LSESC to this system, is that it is a multiple input system and that there are constraints that must be handled, despite fact that ESC originally is an unconstrained optimization method.

4.1 System Description

The system we want to optimize is a gas lifted well network with 3 wells. A simple figure of the gas lifted well network is shown in Figure 4.1.

The overall objective is to get oil up from the reservoirs, to the top facilities, in an economical way. If the reservoir pressure is not sufficient, artificial lift methods can be used to increase the production. In this system gas injection is used [14]. Compressed gas is injected in the bottom of the well, through the annulus, that is the void between two concentric objects where fluid can flow, and thereby reducing the fluid mixture density. The hydrostatic pressure drop in the well is reduced, and the pressure at the bottom of the well decreases, leading to increased flow from the reservoir [9]. The gain in production by injecting gas is dependent on the outflow of the reservoir, decreased outflow give an

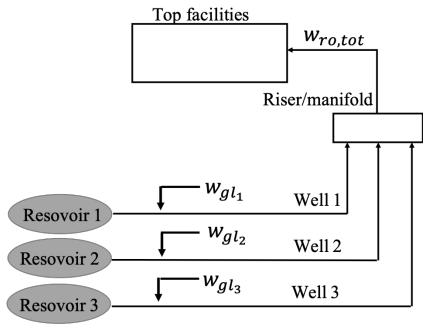

Figure 4.1: Simple figure of the gas lifted well network. w_{gl_i} is the gas lift rate in well i and $w_{ro,tot}$ is the total oil production from the 3 wells.

increase in the gain.

In the system, there is a limited amount of the total gas available for gas lift. The total gas lift injection rate, for the 3 wells, can not exceed the limit $w_{gl,max}$. Thus, to have optimal operation, that is maximizing the total oil production, the optimal gas lift injection rate, for each well, must be found. The gas lift in each well has an upper limit $w_{gl,i}^{max}$ and a lower limit $w_{gl,i}^{min}$. The values of $w_{gl,max}$, $w_{gl,i}^{max}$ and $w_{gl,i}^{min}$ in the system are 5 $\frac{L}{min}$, 3 $\frac{L}{min}$ and 1 $\frac{L}{min}$, respectively.

The gas lift method described, is commonly used in oil and gas production. In this project we look at an experimental lab rig, that represent the gas lifted well system. The method is tested on a model that represent the rig, in order to investigate the opportunity to implement LSESC.

4.1.1 The Model

The production of the gas lifted wells is described and modelled using mass balances of the different phases , density models, pressure models and flow models [9], which results in a differential algebraic equation (DAE) on the form

$$\dot{x}_i = f_i(x_i, z_i, u_i, p_i) \tag{4.1}$$

$$g_i(x_i, z_i, u_i, p_i) = 0 \qquad \forall i \in \mathcal{N} = \{1, .., n_w\} \tag{4.2}$$

where $f_i(x_i, z_i, u_i, p_i)$ is a set of differential equations, and $g_i(x_i, z_i, u_i, p_i)$ is a set of algebraic equations. The subscript i refers to a well i from a set of $N = \{1, .., n_w\}$ wells. In this model there are three wells, so $n_w = 3$. x_i is the differential states, z_i is the

algebraic states, u_i is the decision variables and p_i is the uncertain parameters. They are given by

$$
\begin{aligned}
x_i &= [m_{g_i} \;\; m_{o_i}]^T \\
z_i &= [\rho_{ro_i} \;\; \rho_{gr_i} \;\; p_{rh_i} \;\; p_{bh_i} \;\; w_{ro_i} \;\; w_{pr_i} \;\; w_{gr_i} \;\; w_{lr_i}]^T \\
u_i &= [w_{gl_i}]^T
\end{aligned}
$$

where m_{g_i} and m_{o_i} is the gas and oil hold up, respectively. $\rho_{ro,i}$ is the mixture density in the riser, $\rho_{gr,i}$ is the gas density, p_{rh_i} is the riser head pressure and p_{bh_i} is the pressure below injection point. w_{ro_i} is the oil rate from the resovoir, w_{gr_i} is the riser head gas production rate, w_{lr_i} is the riser head liquid production rate and w_{pr_i} is the riser head total production rate. w_{gl_i} is the gas lit rate, and the index i, refers to well i.

These equations are for one well, but the wells are connected, as they produce oil and gas to the same top facilities. The full model is given in Appendix A, including the code for the dynamic model in Appendix A.1.

4.2 The Optimization Method

In this section the optimization method is explained. This includes a description of the optimization problem, the dithers added to the inputs, the control and the constraint handling.

4.2.1 The Optimization Problem

The economic objective in this case study, is to maximize the profit,J. This is done by finding the optimal gas lift injection rate, w_{gl_i}, for each of the three wells. Thus, the MVs for the optimization problem are given by

$$\mathbf{u} = [u_1 \;\; u_2 \;\; u_3]^T = [w_{gl_1} \;\; w_{gl_2} \;\; w_{gl_3}]^T$$

The optimization problem has a constraint on the total gas available for gas lift, $w_{gl,max}$. In addition, the gas lift injection rate for each well have an upper and a lower limit, $w_{gl,i}^{max}$ and $w_{gl,i}^{min}$. The optimization problem is written as

$$
\begin{aligned}
&\max \; J = \alpha_o \sum_{i=1}^{n_w} w_{ro_i} - \alpha_{gl} \sum_{i=1}^{n_w} w_{gl_i} \\
\text{s.t} \\
&\sum_{i=1}^{n_w} w_{gl_i} \leq w_{gl,max} \\
&w_{gl,i}^{min} \leq w_{gl_i} \leq w_{gl,i}^{max} \quad \forall i \in \mathcal{N} = \{1, .., n_w\}
\end{aligned}
$$

J is the objective function describing the cost, where α_o is the price of oil, α_{gl} is the cost of compressing the gas for gas lift injection and w_{ro_i} is the oil rate from reservoir i.

4.2.2 The Dither

In this case study, a square wave is used as dither added to the inputs, **u**. A square wave is a periodic waveform, where the amplitude alternates between a fixed minimum and maximum, with the same duration at minimum and maximum. In order to estimate the gradient, $\mathbf{J_u}$, with respect to each input, the dither signals should have different frequencies [15]. The frequency of the wave is the number of times the waveform repeats itself within a given time period. The dither signals could also have different amplitudes. In discrete time the inputs, with added dither, can be written as

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \mathbf{a} \cdot \textbf{sq.wave} = \begin{bmatrix} w_{gl_1,k} \\ w_{gl_2,k} \\ w_{gl_3,k} \end{bmatrix} + \begin{bmatrix} a_1 & 0 & 0 \\ 0 & a_2 & 0 \\ 0 & 0 & a_3 \end{bmatrix} \begin{bmatrix} \text{sq.wave}(\omega_{sq,1}) \\ \text{sq.wave}(\omega_{sq,2}) \\ \text{sq.wave}(\omega_{sq,3}) \end{bmatrix} \tag{4.3}$$

where **a**· **sq.wave** represent the dither added to each of the inputs. **a** is the amplitude of each of the signals, and $\omega_{sq,i}$ is the frequency of the square wave added to input $w_{gl_i,k}$.

4.2.3 The Control

Integral control is used to control the inputs, and drive the gradients, $\mathbf{J_u}$, to zero. Each input can have an individual integral gain, k_{I_i}. The integral gain must be chosen small enough so the time scale of the gradient estimation is slower then the dither signal, as mentioned in Section 2.3.2.

ESC is originally an unconstrained optimization method. In the case where the optimum is unconstrained, the control can be written as

$$\mathbf{u}_{k+1} = \begin{bmatrix} w_{gl_1,k} \\ w_{gl_2,k} \\ w_{gl_3,k} \end{bmatrix} + \begin{bmatrix} k_{I,1} & 0 & 0 \\ 0 & k_{I,2} & 0 \\ 0 & 0 & k_{I,3} \end{bmatrix} \begin{bmatrix} \hat{J}_{u,1} \\ \hat{J}_{u,2} \\ \hat{J}_{u,3} \end{bmatrix} \tag{4.4}$$

where the gradients are controlled to a constant set point of zero. This can be done by using another approach, as done in [8], where the controlled variables are

$$CV_1 := \hat{J}_{u,1} \tag{4.5}$$

controlled to a setpoint of zero, and

$$CV_i := (\hat{J}_{u,i} - \hat{J}_{u,i-1}) \ \ \forall i = 2, 3 \tag{4.6}$$

controlled to a setpoint of zero, which yields controlling all three gradients to zero. By doing this, the control can be written as

$$\mathbf{u}_{k+1} = \begin{bmatrix} u_{k,1} \\ u_{k,2} \\ u_{k,3} \end{bmatrix} + \begin{bmatrix} k'_{I,1} & 0 & 0 \\ 0 & k'_{I,2} & 0 \\ 0 & 0 & k'_{I,3} \end{bmatrix} \begin{bmatrix} \hat{J}_{u,1} \\ \hat{J}_{u,2} - \hat{J}_{u,1} \\ \hat{J}_{u,3} - \hat{J}_{u,2} \end{bmatrix} \tag{4.7}$$

When writing the control like this, it is more suitable in regards to handling the constraints.

4.2.4 The Constraint Handling

In this section the constraint handling, for the gas lifted well network, is explained. For the controller to handle the constraints, active constraint control is used, along with ESC for the two unconstrained DOF [8], as explained briefly in Section 2.4. Since there are both constraints on the inputs and on the total gas available for gas lift, the constraint handling can become more complicated. For example, if more than one of the input bounds become active, there will no longer be two unconstrained DOF. When a constraint becomes active, one degree of freedom is lost. To handle this, the method in [8] is applied, along with some logical thinking, in order to control and optimize the process without violating any of the constraints. An overview of the approach can be seen in Figure 4.2 and consists of 5 steps, explained below.

1. **Check flows:** As mentioned in the theory, the MV with the largest flow should be used to control the active constraint. Thus, which MV that is the largest should be checked for every iteration. Let the largest input be called u_L, and the two other inputs be called u_{s1} and u_{s2}

2. **Treat problem as unconstrained:** New input, $\mathbf{u_{k+1}}$, is calculated by treating the problem as an unconstrained problem, using the approach described in Equation 4.7. Which MV that is largest should be the taken into account. u_L should be the input controlled by only one gradient (the gradient with respect to u_L), as shown in Figure 4.2.

3. **Check minumum and maximum constraint on inputs:** Both $w_{gl,i}^{min}$ and $w_{gl,i}^{max}$ must be checked. If one of the inputs is below the minimum or above the maximum, the input should be changed to the minimum or maximum, respectively.

4. **Check constraint with maximum gas lift capacity:** If the constraint is violated, u_L is used to control the constraint by setting

$$u_L = w_{gl,max} - u_{s1} - u_{s2} \tag{4.8}$$

However, this can possibly violate the minimum constraint on u_L. To avoid this the the following is done:

- If the minimum constraint on u_L is not going to be violated, the following must be attained

$$u_{s1} + u_{s2} \leq w_{gl,max} - u_{min} \tag{4.9}$$

If not, the amount, $(u_{s1}+u_{s2})-(w_{gl,max}-u_{min})$, needs to be subtracted from u_{s1} and u_{s2}. How the this should be divided between the two MVs, must be taken into consideration. It must be subtracted in such a way that the minimum and maximum constraint on the inputs are not violated. In this case, it is done by setting

$$u'_{s1} = u_{s1} - \Big[(u_{s1} + u_{s2}) - (w_{gl,max} - u_{min})\Big] \frac{u_{s1}}{u_{s1} + u_{s2}} \tag{4.10}$$

and

$$u'_{s2} = u_{s2} - \Big[(u_{s1} + u_{s2}) - (w_{gl,max} - u_{min})\Big] \frac{u_{s2}}{u_{s1} + u_{s2}} \tag{4.11}$$

By doing this, the amount subtracted from one input, dependents on its size. A larger value, results in a larger amount subtracted from the input.

One drawback with this approach is that it can cause the minimum constraint to be violated. If one input is close to, or at the this constraint, the subtraction will make the value of the input become lower than $w_{gl,i}^{min}$. This must be checked, and taken care of. If this is the case for u_{s1}, the following must be done

$$u''_{s2} = u'_{s2} - (w_{gl,i}^{min} - u'_{s1}) \tag{4.12}$$

$$u''_{s1} = w_{gl,i}^{min} \tag{4.13}$$

By doing this, $u_{s1} + u_{s2} = w_{gl,max} - w_{gl,i}^{min}$, without violating the constraints on the inputs. The approach is the same if u_{s2} is the input violating the minimum constraint. In this case u_L is going to be equal to $w_{gl,i}^{min}$, when doing the next step.

- The constraint with maximum gas lift capacity can now be checked. If the constraint is violated, i.e

$$\sum_{i=1}^{n_w} u_i \geq w_{gl,max} \tag{4.14}$$

u_L must be set to

$$u_L = w_{gl,max} - u_{s1} - u_{s2} \tag{4.15}$$

5. **Update the input:** Now, all the constraints are handled, and the input,$\mathbf{u_{k+1}}$, can be updated.

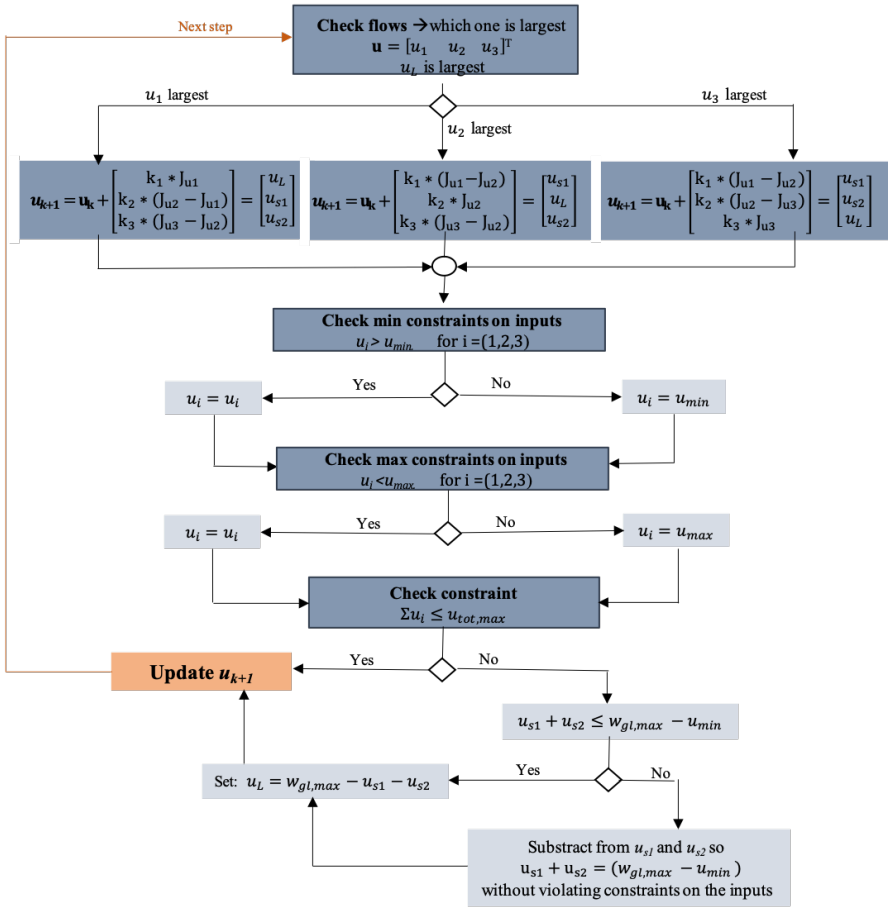

Figure 4.2: Flow diagram for the constraint handling. u_L is the MV with largest flow, while u_{s1} and u_{s1} are the to other MVs with smaller flows. u_{min} and u_{max} correspond to $w_{gl,i}^{min}$ and $w_{gl,i}^{min}$ respectively.

4.3 Results

In this section the simulation results are presented. The optimization method is applied to two scenarios of disturbances. This is done to better validate the method, and to make sure that the tuning is not tailored to one particular set of disturbances. The simulation results for the two scenarios are shown in Figure 4.3 and 4.4. The figures show the control of the three inputs, w_{gl_1}, w_{gl_2} and w_{gl_3}, the value of the cost function and the total gas lift.

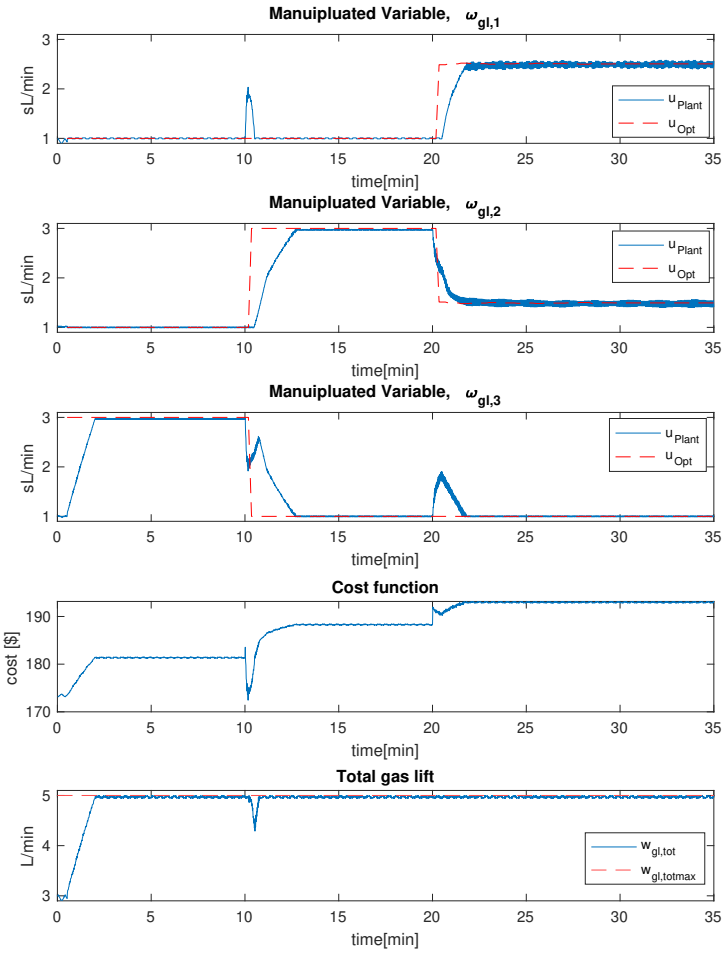

Figure 4.3: Simulation results for disturbance scenario 1.

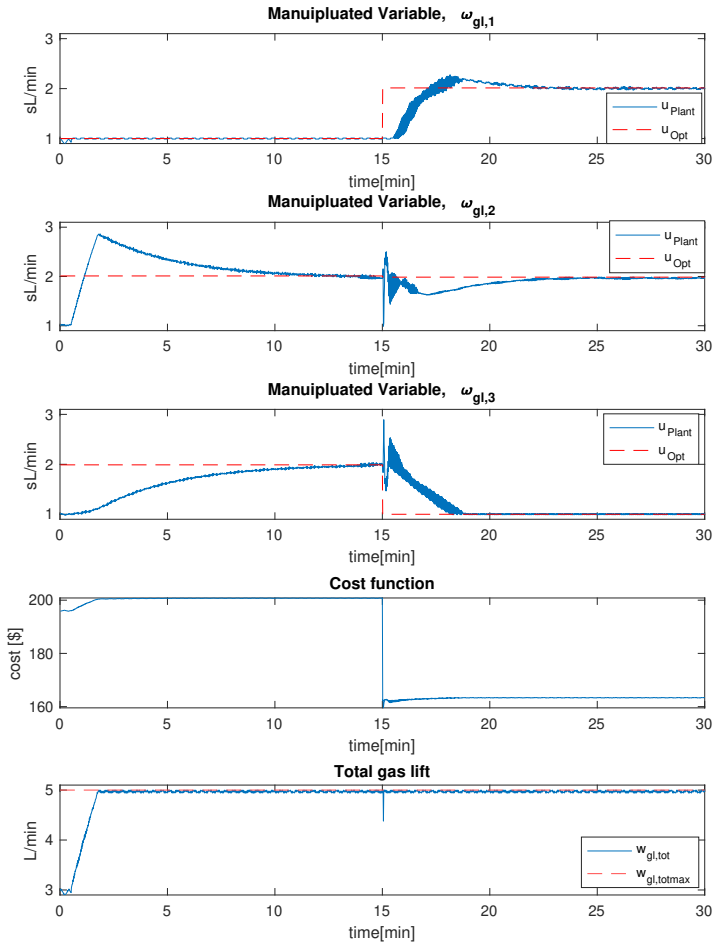

Figure 4.4: Simulation results for disturbance scenario 2.

In scenario 1 the disturbances enters the system at 10 min and 20 min, and in scenario 2 the disturbance enters at 15 min. The disturbances represent an increase or decrease in the outflow of the reservoirs. The disturbance scenarios illustrates a typical operating scenario, which arises from changes in the reservoir natural pressure. With smaller reservoir outflow the gain in the well production with respect to gas injection increases. As a consequence, the optimal distribution of the available gas lift changes and a new operating strategy needs to be found.

In both disturbance scenarios the controller is able to drive the inputs to the optimal operating point, which was previously computed, without violating the constraints. Thus, LSESC works for this process with multiple inputs and constraints, in simulations. However, comparing these simulations results with the simulation result from the simple example in Figure 3.2, from Case Study 1, the MVs oscillate more and the control actions are less smooth.

4.3.1 Tuning Parameters

The tuning that was used for the parameters in this case study are shown in Table 4.1 below. It is worth to mention that another set of tuning parameters can work better for one scenario of disturbances, and worse for the other. In this case, the goal was to find a set of tuning that gave acceptable performance in both scenarios.

Table 4.1: Tuning parameters for Case Study 2

Parameter	Description	Value
a	Amplitude for the sqaure waves	0.01
ω_1	Frequency of square wave added to $w_{gl,1}$	6
ω_2	Frequency of square wave added to $w_{gl,2}$	14
ω_3	Frequency of square wave added to $w_{gl,3}$	10
N	Buffer length	30
k_{I_1}	Integral gain for w_{gl_1}	0.02
k_{I_2}	Integral gain for w_{gl_2}	0.02
k_{I_3}	Integral gain for w_{gl_3}	0.01

4.4 Discussion

4.4.1 The Simulation Result

Scenario 1

Figure 4.3 is the simulation results for disturbance scenario 1. It shows that when the disturbance happens, at 10 minutes and 20 minutes, and the optimal value of the inputs changes, some oscillation occurs. At 10 minutes, the optimal value changes for both w_{gl_2} and w_{gl_3}, but not for w_{gl_1}. And at 20 minutes, the optimal value changes for both w_{gl_1} and $w_{gl,2}$, but not for w_{gl_3}. In both these cases, the input that do not change its setpoint, has an increase before it goes back down to its optimal value. This is because the gradient estimation is less accurate as the disturbance occur. The consequence of this is less smooth control, in addition to an economical loss, which can be seen as a drop in the cost function as the disturbances occur. The economic loss is also related to the time the controller takes to reach the new setpoint. The two inputs that changes their setpoint, at 10 and 20 minutes, are able to move relatively smooth against their optimal value. The exception is w_{gl_3} at 10 minutes, where the input has a little jump in the wrong direction as it moves to its optimum. The reason for this can be the less accurate gradient estimation as the disturbance occur.

Scenario 2

Figure 4.4 is the simulation results for disturbance scenario 2. It shows that there is an overshoot in w_{gl_2}, in the beginning of the interval, before it settles to its optimum. However, this do not give a visible loss in the cost function. The range of values of w_{gl_2} and w_{gl_3}, from around 2 minutes, and until they reach their optimal value, give nearly the same value of the cost function. This indicates that the cost functions optimum, with respect to the inputs, is relative flat, and that is probably why the overshoot happens. With respect to the value of the inputs it looks like the system is operating sub-optimally, but by looking at the cost function one can see that this is not the case.

When the disturbance happens in scenario 2, some oscillation occurs, which was the case in scenario 1 as well, but its amplitude is larger in this scenario. Once again, for scenario 2, the economical consequence of this is very small, the value of the cost function is very similar after the disturbance occur. By looking at the inputs, the operation looks sub-optimal, but not by looking at the cost function. This may be the reason why the inputs uses more time to find their optimal value in this scenario, compared with scenario 1, where we see a clear connection between the cost function and the inputs not being at their optimal value.

Which MV that is the largest alternates, between w_{gl_2} and w_{gl_3}, in the time right after the disturbance occur, meaning that which input that is used to control the constraint with the maximum gas lift capacity, also alternates. This can be another reason for the noise. One possible way to reduce the noise, as mentioned in Section 3.3.2, is to have a constraint on how much the inputs can change in one step. A plot of the two inputs around the time the disturbance take place is shown in Figure 4.5. This plot is obtained by zooming in Figure 4.4.

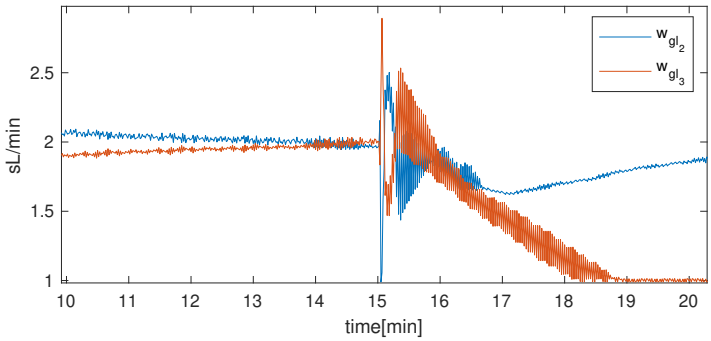

Figure 4.5: Simulation result of w_{gl_2} and w_{gl_3} around the time the disturbance occur, in scenario 2.

4.4.2 The Tuning

As mentioned in Case Study 1, in Chapter 3, tuning the controller is a challenging task. With three inputs in the system, it is more challenging, since there are more parameters to tune. From Case Study 1, it was shown that there are different values for the tuning parameters that still yield an acceptable optimization performance. When a set of tuning parameters that is able to drive the system to its optimum is found, one can start looking at how each parameter affect the control, in order to improve the performance of the controller.

From Case Study 1, we also saw that the performance was not so sensitive to the value of the integral gain. So a good place to start with the integral gains, is setting all three equal. When a set of parameters that works is obtained, you can start tuning each of them separately. This approach was done in this case study.

The amplitude of the dither signals can also be set as the same for the three inputs, but the frequencies must be set to different values, in order to estimate the gradient of the objective function with respect to each input, as explained in the theory. In Case Study 1, it was hard to see any clear guidelines for the frequency. A suggestion is to calculate the

real value of the gradients, and compare them against the estimated gradients. Studying how they match can be a valuable tool when tuning the buffer length and the amplitude and frequency of the dither signals. The purpose of the dither signals and the buffer is to estimate the gradients. If the gradient estimation is good, so is the tuning of the buffer and dither signals. This suggestion is only possible if a model is used to represent the plant, otherwise, it is not possible to calculate the 'true' gradient.

4.4.3 Applying the Method to Other Processes

From the results, we can see that the constraint handling strategy performs satisfactorily for both disturbances scenarios. Despite being defined for this specific case study, this strategy can be extended for systems with different constraint sets, which contain upper and lower bounds on the inputs as well as linear constraints involving two or more inputs.

However, in this case, adjustments in the strategy devised in Figure 4.2 are most likely necessary. Since the logic of the constraint handling reflects the process optimization at hand, different problems will require different strategies. Also, if there are several complicating constraints that involve more than one input, the constraint handling can become more complicated.

4.4.4 Implementation on Lab Rig

From the simulations, LSESC is now merited for implementation in the lab. However, the simulations does not necessarily correspond to what would happen in the real system. The model is not necessarily perfect, and unexpected disturbances can occur. The chances of having to change the tuning, or that the controller will not work at all, are present.

There can be noise in the measurements. If the input values are noisy, the input perturbations can become less significant, especially if the amplitude is small compared to the noise. This can result in poor estimation of the gradients. If this is the case, one would probably have to increase the amplitude of the perturbation, in order to make it more significant and get a better estimation of the gradients.

The considerations mentioned in Section 3.3.2, with applying the method to a real process, should also be thought through. For further work the controller should be implemented and tested on the experimental lab rig.

4.5 Conclusion

LSESC as an optimization approach in the gas lifted oil well network model, with multiple input and constraints, works in the simulations. More inputs make the tuning more challenging, since the number of tuning parameters is increased. Despite the fact that ESC originally is an unconstrained optimization method, it can handle constraints. By using active constraint control, along with ESC and some logic, the system is driven to its optimum, without violating the constraint, for two scenarios of disturbances. It should be further investigated if the approach works for all set of disturbances

Assuming that the constraint handling method works, not only for the two scenarios of disturbances, the method can be used in similar processes, where there are constraints involving more than one input, in addition to upper an lower bounds on the inputs. In this case, adjustments to the logical approach, in Figure 4.2, are likely necessary for fitting the method to the new system.

From the simulations, the prospect of implementing LSESC on the experimental lab rig, which is represented by the model in Section 4.1.1, is good. But even though the method works in simulations, there is no guarantee that it would work on the actual rig, and the possibility of needing to re-tune the parameters is present. This should be tested and further investigated.

Chapter 5

Conclusion

In this specialization project there was performed two case studies. From both case studies it was clear that tuning the parameters for LSESC can be a challenge. In the first case study a simple system with one input was studied, and some guidelines for the tuning parameters was found, they are presented in Table 3.2. In Case Study 2, a gas lifted well network, with three inputs was studied. More inputs made the tuning more difficult, since the number of tuning parameters was increased.

In traditional ESC, a sinusoidal perturbation is used to excite the system, which can be hard to implement. In LSESC, other dither signals can be used, such as a PRBS or a square wave. It was shown that using a PRBS or a square wave as a dither gave good performance, in Case Study 1 and Case Study 2, respectively.

Despite the fact that ESC originally is an unconstrained optimization method, it can handle constraints. By using active constraint control, along with ESC and some logic, the gas lifted well network, in Case Study 2, was driven to its optimum, without violating its constraints, for two scenarios of disturbances. A step-by-step approach for the constraint handling was established. This approach can be used in similar processes, but some adjustments may be necessary for fitting the method to a new system.

From the simulation results in Case Study 2, one can conclude that the prospect of implementing LSESC on the experimental lab rig is good. However, there are no guarantee that the it will work on the actual rig. This should be tested and further investigated.

Bibliography

[1] K. B. Ariyur. Real-Time Optimization by Extremum-Seeking Control. Technical report, 2003.

[2] W. Bequette. *Dynamics Process: Modeling, Analysis, and Simulation*. Prentice Hall PTR, Upper Saddle River, NJ, 1998.

[3] S. L. Brunton and L. J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamics Systems, and Control*. Cambrige University Press, 2019.

[4] R. Dechter and J. Pearl. Network-Based Heuristics for Constraint-Satisfaction Problems. In *Search in Artificial Intelligence*. Springer New York, New York, NY, 1988.

[5] S. Dutta. *Optimization in Chemical Engineering*. Cambridge University Press, 2016.

[6] C. Heij, P. d. Boer, P. H. Franses, T. Kloek, and H. K. van Dijk. *Econometric Methods with Applications in Business and Economics*. OUP Oxford, 2004.

[7] B. G. Hunnekens, M. A. Haring, N. Van De Wouw, and H. Nijmeijer. A dither-free extremum-seeking control approach using 1st-order least-squares fits for gradient estimation. *53rd IEEE Conference on Decision and Control*, pages 2679–2684, 2014.

[8] D. Krishnamoorthy, K. Fjalestad, and S. Skogestad. Optimal operation of oil and gas production using simple feedback control structures. *Control Engineering Practice*, 91, 10 2019.

[9] D. Krishnamoorthy, B. Foss, and S. Skogestad. Real-time optimization under uncertainty applied to a gas liftedwell network. *Processes*, 4(4), 12 2016.

[10] M. Krstich and H.-H. Wang. Stability of extremum seeking feedback for general nonlinear dynamic systems. In *Automatica*, volume 36, pages 595–601, 2000.

[11] L. Ljung. *System Identification*. Prentice Hall, Upper Saddle River, NJ, 1999.

[12] D. E. Seborg, T. F. Edgar, D. A. Mellichamp, and F. J. Doyle III. *Process Dynamics and Control*. John Wiley & Sons, Inc, 3rd edition, 2011.

[13] S. Skogestad. Self-optimizing control: The missing link between steady-state optimization and control. 24(2-7):569–575, 2000.

[14] S. Skogestad and E. Storkaas. Stabilization of severe slugging based on a low-dimensional nonlinear model. Trondheim, Norway, 2002. In Norwegian University of Science and Technology, Norwegian University of Sience and Technology.

[15] J. Straus, D. Krishnamoorthy, and S. Skogestad. On combining self-optimizing control and extremum-seeking control – Applied to an ammonia reactor case study. *Journal of Process Control*, 78:78–87, 6 2019.

Appendix A

Gas Lifted Wells Model

The modelling for the gas lifted wells are based mass balances of the different phases, density models, pressure models and flow models [9]. This results in a DAE on the form

$$\dot{x}_i = f_i(x_i, z_i, u_i, p_i) \tag{A.1}$$

$$g_i(x_i, z_i, u_i, p_i) = 0 \qquad \forall i \in \mathcal{N} = \{1, .., n_w\} \tag{A.2}$$

where $f_i(x_i, z_i, u_i, p_i)$ is a set of differential equations, and $g_i(x_i, z_i, u_i, p_i)$ is a set of algebraic equations. The subscript i refers to a well i from a set of $N = \{1, .., n_w\}$ wells. In this model there are three wells, so $n_w = 3$. x_i is the differential states, z_i is the algebraic states, u_i is the decision variables and p_i is the uncertain parameters. They are given by

$$\begin{aligned} x_i &= [m_{g_i} \;\; m_{o_i}]^T \\ z_i &= [\rho_{ro_i} \;\; \rho_{gr_i} \;\; p_{rh_i} \;\; p_{bh_i} \;\; w_{ro_i} \;\; w_{pr_i} \;\; w_{gr_i} \;\; w_{lr_i}]^T \\ u_i &= [w_{gl_i}]^T \end{aligned}$$

where m_{g_i} and m_{o_i} is the gas and oil hold up, respectively. $\rho_{ro,i}$ is the mixture density in the riser, $\rho_{gr,i}$ is the gas density, p_{rh_i} is the riser head pressure and p_{bh_i} is the pressure below injection point. w_{ro_i} is the oil rate from the resovoir, w_{gr_i} is the riser head gas production rate, w_{lr_i} is the riser head liquid production rate and w_{pr_i} is the riser head total production rate. w_{gl_i} is the gas lit rate, and the index i, refers to well i.

The differential equations are given by

$$\mathbf{f} = [df_1 \;\; df_2] \tag{A.3}$$

and the algebraic equations are given by

$$\mathbf{g} = [f_1 \;\; f_2 \;\; f_3 \;\; f_4 \;\; f_5 \;\; f_6 \;\; f_7 \;\; f_8] \tag{A.4}$$

These equations are shown in the code for the dynamic model below.

A.1 ErosionRigDynModelGrad.m

```
function [F,S_xx,S_xp,L_xz,L_p] = ErosionRigDynModelGrad(par)
%    Simulates SS model of the rig with the data-driven reservoir
%    model

% Inputs:
%    par = system parameters
%
% Outputs:
%   F: system integrator
%   F_xk,F_pk = system sensitivities
%
% Other m-files required: none
% Subfunctions: none
% MAT-files required: none
%
% Author: Julio Paez
% email: julio.paez.oliveira@usp.br
% June 2020; Last revision:

%addpath('<yourpath>/casadi-matlabR2014a-v3.5.1')
import casadi.*

%% Parameters
%number of wells
n_w = par.n_w; %[]
%gas constant
R = par.R; %[m3 Pa K^?1 mol^?1]
%molecular weigth
Mw = par.Mw; %[kg/mol?]

%properties
```

```
%density of oil - dim:  nwells x 1
rho_o = par.rho_o; %[kg/m3]
%lcP oil viscosity
mu_oil = par.mu_oil;% [Pa s]
%density of gas
rho_g = par.rho_g; %[kg/m3]

%project
%well parameters - dim:  nwells x 1
L_w = par.L_w; %[m]
H_w = par.H_w; %[m]
D_w = par.D_w; %[m]
A_w = par.A_w;%[m2]

%well below injection - [m]
L_bh = par.L_bh;
H_bh = par.H_bh;
D_bh = par.D_bh;
A_bh = par.A_bh;%[m2]

%riser - [m]
L_r = par.L_r;
H_r = par.H_r;
D_r = par.D_r;
A_r = par.A_r;%[m2]


%% System states
%gas holdup
m_g = MX.sym('m_gr',n_w);        % 1:3 [kg]
%oil holdup
m_o = MX.sym('m_or',n_w);        % 4:6 [kg]

%oil rate from reservoir
w_ro = MX.sym('w_ro',n_w);       % 7:9 [kg/s]
%riser head total production rate
w_pr = MX.sym('w_pr',n_w);       % 10:12 [kg/s]

%riser head pressure
p_rh = MX.sym('p_rh',n_w);       % 13:15 [bar]
%pressure - below injection point (bottom hole)
p_bh = MX.sym('p_bh',n_w);       % 16:18 [bar]

%mixture density in riser
rho_r =  MX.sym('rho_r',n_w);    % 19:21 [100 kg/m3]
%density gas
```

```
rho_gr= MX.sym('rho_gr',n_w);    % 22:24 [kg/m3]

%riser head gas production rate gas
w_gr = MX.sym('w_gr',n_w);       % 25:27 [kg/s]
%riser head gas production rate gas
w_lr = MX.sym('w_lr',n_w);       % 28:30 [kg/s]

%% System input
%gas lift rate
Q_gl = MX.sym('w_gl',n_w);       % 1:3 [L/min]
%valve oppening
vo=MX.sym('vo',n_w);             % 4:6 [0-1]
%velocity pump
vpump=MX.sym('vpump',1);        % 7 [%]

%% parameters
% fixed
%riser temperature
T_r = MX.sym('T_r',1); %[oC]
%separator pressure
p_s = MX.sym('p_s',1); %[bar]
%time transformation: CASADI integrates always from 0 to 1 and the USER
    does the time
%scaling with T.
T = MX.sym('T',1); %[s]

% estimable
%reservoir data-driven model
res_theta = MX.sym('res_theta',n_w);
%riser valve characteristics
val_theta = MX.sym('val_theta',n_w);%[m2]

%% Modeling
%conversion
CR = 60*10^3; % [L/min] -> [m3/s]
%oil from reservoir flowrate [kg/s]
f1 = -(w_ro.*1e-2).*CR./rho_o + (1e-2*(res_theta.*1e1)).*(Q_gl)  + (vo
    .^0.002485526-0.994065).*((vpump*1e2).^2 - 0.003760888.*(Q_gl .*(vpump
    *1e2)).^2);

% total riser production [kg/s]
f2 = - (w_pr.*1e-2) + ((w_gr.*1e-5) + (w_lr.*1e-2));
%riser head pressure [Pa]
f3 = -p_rh.*1e5 + (w_pr.*1e-2).^2./((1e-4*val_theta).^2.*(rho_r.*1e2)) +
    p_s.*1e5 ; %25
%bottom hole pressure [Pa]
```

```
f4 = -p_bh.*1e5 + (p_rh.*1e5 + (rho_r.*1e2).*9.81.*H_r + 128.*mu_oil.*(L_w
    +L_r).*(w_ro.*1e-2)./(3.14.*D_bh.^4.*(rho_r.*1e2)));
%riser density [1e2 kg/m3]
f5 = -(rho_r.*1e2) + (((m_g.*1e-4) + m_o).*p_rh.*1e5.*Mw.*rho_o)./(m_o.*
    p_rh.*1e5.*Mw + rho_o.*R.*T_r.*(m_g.*1e-4));
%density gas with pressure [kg/m3]
f6 = -rho_gr + p_rh.*1e5.*Mw/(R*T_r);

% production distribution
xL = (m_o./((m_g.*1e-4) + m_o));
%xG = 1 - xL;
%Gas Mass production [kg/s] / %Oil Mass production [kg/s]
f7 = -(w_lr.*1e-2) + xL.*(w_pr.*1e-2);
% f7 = -w_gr + xG.*w_pr;
% Volume [V]
f8 = -(A_w.*L_w + A_r.*L_r) + (m_o./rho_o + (m_g.*1e-4)./rho_gr);

%dynamic: change these to differential
%gas [kg]
df1= -(w_gr.*1e-5) + Q_gl./(CR./rho_gr);
%liquid [kg]
df2= -(w_lr.*1e-2) + (w_ro.*1e-2);

%objective function
 J = -(- 10*((w_ro(1)*1e-2)*CR/rho_o(1)) - 10*((w_ro(2)*1e-2)*CR/rho_o(2))
     - 10*((w_ro(3)*1e-2)*CR/rho_o(3))...
     + 0.5.*(Q_gl(1)) + 0.5.*(Q_gl(2)) + 0.5.*(Q_gl(3)));
% Form the DAE system
diff = vertcat(df1,df2);
alg = vertcat(f1,f2,f3,f4,f5,f6,f7,f8);

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

% concatenate the differential and algebraic states
x_var = vertcat(m_g,m_o);
z_var = vertcat(w_ro,w_pr,p_rh,p_bh,rho_r,rho_gr,w_gr,w_lr);
u_var = vertcat(Q_gl,vo,vpump);
p_var = vertcat(res_theta,val_theta,T);

%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*diff,'
```

```
    alg',alg);
dae = struct('x',x_var,'z',z_var,'p',vertcat(u_var,p_var),'ode',T*diff,'
    alg',alg, 'quad', J);
%calling the integrator, the necessary inputs are: label; integrator;
    function with IO scheme of a DAE (formalized); struct (options)
F = integrator('F','idas',dae);

%%integration results
% Fend = F('x0',dxk,'z0',zk,'p',[uk;thetak;par.T]);
%
%%extracting the results (from symbolic to numerical)
% xk_1 = full(Fend.xf);
% zk_1 = full(Fend.zf);

    % ================================================
    %       Calculating sensitivity matrix (theta)
    % ================================================

% S_xx = Function('sensStaStates',{x_var,z_var,u_var,p_var},{jacobian(
    vertcat(T*diff,alg),vertcat(x_var,z_var))});
% S_xp = Function('sensStaStates',{x_var,z_var,u_var,p_var},{jacobian(
    vertcat(T*diff,alg),vertcat(u_var,p_var))});
%% S_zz = F.factory('sensStaStates',{'x0','z0','p'},{'jac:zf:z0'});
%% S_xz = F.factory('sensStaStates',{'x0','z0','p'},{'jac:xf:z0'});
%% S_zx = F.factory('sensStaStates',{'x0','z0','p'},{'jac:zf:x0'});
%
%% S_xp = F.factory('sensParStates',{'x0','z0','p'},{'jac:xf:p'});
%% S_zp = F.factory('sensParStates',{'x0','z0','p'},{'jac:zf:p'});
%%
% L_xz = Function('sensStaStates',{x_var,z_var,u_var,p_var},{jacobian(J,
    vertcat(x_var,z_var))});
% L_p = Function('sensStaStates',{x_var,z_var,u_var,p_var},{jacobian(J,
    vertcat(u_var,p_var))});

S_xx = Function('sensStaStates',{x_var,z_var,u_var,p_var},{jacobian(
    vertcat(T*diff,alg),vertcat(x_var,z_var))});
S_xp = Function('sensStaStates',{x_var,z_var,u_var,p_var},{jacobian(
    vertcat(T*diff,alg),vertcat(u_var,p_var))});
L_xz = Function('sensStaStates',{x_var,z_var,u_var,p_var},{jacobian(J,
    vertcat(x_var,z_var))});
L_p = Function('sensStaStates',{x_var,z_var,u_var,p_var},{jacobian(J,
    vertcat(u_var,p_var))});
end
```

Appendix B

MATLAB Code: Case study 1

In this appendix the code for the simulations in Case Study 1 is presented. `caseStudy1.m` is the main script, Plant.m represent the biochemical reactor and `LSE.m` is the Least Square Estimation with one input.

B.1 caseStudy1.m

```
clear
close all
clc
%% Setting the system
%declaring integration parameters
T = 2000; % time horizon
dt = 1; %sampling time
N = T*dt;

%initial conditions
%states
x10 = 1;     %[g/L] - biomass
x20 = 1;    %[g/L] - substrate
x0 = [x10;x20];
%inputs
u0 = 0.35; %[h^-1] - Dilution Rate: F/V

%disturbances
mumax1 = 0.5;
mumax2 = 0.55;
[uOpt1, psiOpt1] = SteadyStateOptimization(x0, mumax1);
```

```
[uOpt2, psiOpt2] = SteadyStateOptimization(x0, mumax2);

mumax = [mumax1*ones(1,N*dt/2), mumax2*ones(1,N*dt/2+1)];
psiOpt = [psiOpt1*ones(1,N*dt/2), psiOpt2*ones(1,N*dt/2+1)];
uOpt = [uOpt1*ones(1,N*dt/2), uOpt2*ones(1,N*dt/2+1)];

%Control parameters
N_buffer = 100;      %Buffer lenth
KI = 0.0002;         %Integral gain
a = 0.0001;           %Amplitude pertrubation
w = 2*pi*1/5;


%making PRBS dither
cinit = 9;
[seq,cinit] = nrPRBS(cinit, T, 'MappingType','signed');
seq = a*seq;

%% For plotting
%time array
timeSim = [];
%states array
xSim = [];
%optimal RTO inputs array - dillution rate [h^-1]
uSim = [];
%plant profit - dillution rate [g/(L h)]
psiSim = [];

%% Simulation initialization
time = 0;
xk = x0;
uk = u0;

%for plotting
timeSim = [timeSim, time];
xSim = [xSim, xk];
uSim = [uSim, uk];
psiSim = [psiSim, uk*xk(1)];


%% Simulation
J_buffer = [];
u_buffer = [];
Ju_hatSim =[0];
Ju_hat2Sim =[0];
Ju_Sim =[uk*xk(1)];
```

```
68
69 Jk = uk*xk(1);
70
71 for k=1:N
72
73     % Updating/making buffer list
74     if k<=N_buffer
75         J_buffer = [J_buffer, Jk];
76         u_buffer = [u_buffer, uk];
77     else
78         J_buffer = [J_buffer(2:end), Jk];
79         u_buffer = [u_buffer(2:end), uk];
80     end
81
82     %Simulating the plant behavior during dt
83     [xk,Jk,dqdu] = Plant(xk, uk, dt, mumax(k));
84
85     %% LSE - Gradient estimation
86     if k>N_buffer
87         Ju_hat = LSE(J_buffer,u_buffer);
88     else
89         Ju_hat = 0;
90         Ju_hat2 = 0;
91     end
92
93     Ju_hat2Sim = [Ju_hat2Sim,Ju_hat2];
94     Ju_hatSim = [Ju_hatSim,Ju_hat];
95     Ju_Sim = [Ju_Sim, dqdu];
96
97     % I-controll
98     uk = uk + KI*Ju_hat;
99
100    %Adding dither to input signal, chose one of them
101    %Sinsoidal pertrubation
102    uk = uk*(1 + a*sin(w*k));
103    %PRBS
104    %uk = uk + seq(k);
105    if uk<0
106        uk = 0;
107    end
108
109    %Constraints on D
110    if uk<0
111        uk = 0;
112    elseif uk>1
113        uk = 1;
```

```
114     end
115
116     %for plotting
117     timeSim = [timeSim, k*dt];
118     xSim = [xSim, xk];
119     uSim = [uSim, uk];
120     psiSim = [psiSim, uk*xk(1)];
121
122     sum_prof = sum(psiSim); %use to compare profit
123
124 end
125 %% plot data
126 figure(1)%plot states
127     plot(timeSim, xSim(1,1:end), 'b',timeSim, xSim(2,1:end), 'r')
128
129     xlim([0, timeSim(end)])
130     xlabel('t [h]')
131     ylabel('Concentration [g/L]')
132     legend({'x_1: Biomass','x_2: Substrate'});
133     title('Measured variables')
134
135 figure(2)
136 %sgtitle('Simulation Reults for the Base Case')
137 subplot(2,1,1) %plot inputs and opt inputs
138     stairs(timeSim, uSim, 'b')
139     hold on
140     stairs(timeSim(2:end), uOpt(1,1:length(timeSim)-1), 'b:')%previously
        calculated
141
142      xlabel('t [h]')
143      ylabel('Dillution rate [h^{-1}]')
144      legend({'u_{plant}','u_{opt}'});
145      title('Manipulated variable')
146
147      ylim([min(uSim)-0.01,max(uSim)+0.05])
148
149 subplot(2,1,2)%plot profit and opt profit
150      plot(timeSim, psiSim, 'b')
151      hold on
152      stairs(timeSim(2:end), psiOpt(1,1:length(timeSim)-1), 'b:')
153
154       xlabel('t [h]')
155       ylabel('\psi [g/(L h)]')
156       legend({'\psi_{plant}','\psi_{opt}'});
157       title('Instantaneous profit')
158       ylim([psiSim(1)-0.01,max(psiSim)+0.05])
```

```
159
160 figure(4)
161     plot(timeSim, Ju_hatSim)
162     hold on
163     plot(timeSim, Ju_Sim)
164     legend('Ju_hat', 'Ju')
```

B.2 Plant.m

```
1   function [xend,Jend,dqdu] = Plant(x0,D,dt,mumax)
2 % dynamic equations for bioreactor
3 % Bequette book pg. 534
4
5 % state variables
6 % x(1) = biomass - "bugs" that consume the substrate
7 % x(2) = substrate
8 %
9 % D     = dilution rate (F/V, time^-1)
10 % Y     = yield biomass/substrate
11 % mu    = specific growth rate
12 % mumax = parameter (both Monod and Substrate Inhibition)
13 % km    = parameter (both Monod and Substrate Inhibition)
14 % k1    = parameter (Substrate Inhibition only, k1 = 0 for Monod)
15 % sf    = substrate feed concentration
16
17 % calling CasADi
18 addpath('/Users/frida/Documents/MATLAB/casadi-osx-matlabR2015a-v3.5.5')
19 import casadi.*
20
21 %%%%%%%%%%%%%%%%%%%%%%
22 % parameter values
23 %%%%%%%%%%%%%%%%%%%%%%
24    Y = 0.4;
25    km = 0.12;
26    sf = 4.0;
27 %    k1 = 0.4545;
28
29 %%%%%%%%%%%%%%%%%%%%%%
30 % Declare model variables
31 %%%%%%%%%%%%%%%%%%%%%%
32
33 x1 = SX.sym('x1');    %[g/L] - biomass
34 x2 = SX.sym('x2');    %[g/L] - substrate
35 x = [x1;x2];
36
```

```
%%%%%%%%%%%%%%%%%%%%%%
% Declare system inputs
%%%%%%%%%%%%%%%%%%%%%%
u = SX.sym('D'); %[h^-1] - Dilution Rate: F/V

%%%%%%%%%%%%%%%%%%%%%%
%  Substrate Inhibition expression for specific growth rate

%%%%%%%%%%%%%%%%%%%%%%
%  Monod
%%%%%%%%%%%%%%%%%%%%%%
mu = mumax*x2/(km+x2);


%%%%%%%%%%%%%%%%%%%%%%
%  dynamic equations
%%%%%%%%%%%%%%%%%%%%%%

    xdot = [(mu - u)*x1;
            (sf - x2)*u - mu*x1/Y];

%%%%%%%%%%%%%%%%%%%%%%
%  quadrature (OF)
%%%%%%%%%%%%%%%%%%%%%%
L = x1*u;

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

% Formulate discrete time dynamics
% CVODES from the SUNDIALS suite
ode = struct('x',x,'p',u,'ode',xdot,'quad',L);

% building the integrator
opts = struct('tf',dt);
F = integrator('F', 'cvodes', ode, opts);
sim = F('x0',x0,'p',D);

%Getting the states and cost function
xend = full(sim.xf);
Jend = full(sim.qf);
%%%%%%%%%%%%%%%%%%%%%%
% Calculating sensitivities with CASADI MY WAY
%%%%%%%%%%%%%%%%%%%%%%
%calculating jacobian w.r.t. u
```

```
F_u = Function('sensitivity',{x,u},{jacobian(xdot,u)});
%calculating jacobian w.r.t. x
F_x = Function('sensitivity',{x,u},{jacobian(xdot,x)});

%calculating SS sensitivities
S = -full(F_x(xend,D))\full(F_u(xend,D));

%calculating OF steady-state gradient
dqdu = D*S(1) + xend(1);
```

B.3 LSE.m

```
function Ju_hat = LSE2(J_buffer, u_buffer)
x = u_buffer';
y = J_buffer';
theta = polyfit(x,y,1);
Ju_hat = theta(1);
```

Appendix C

MATLAB Code: Case study 2

In this appendix the codes used for the simulations in Case Study 2 are presented. The codes in this case study are made to fit the actual lab rig, so the optimization method can easily be tested on the rig. `Main.m` is the main script, in this file you can set up and change the disturbances. The optimization is in `LabViewES7.m` and `LSE2.m` is the least square estimation with multiple inputs. The model is in Appendix A.1.

`ResultsPlotting.m` is the plotting of the results, `ParametersGasLiftModel.m` is the parameters used in the model, `InitializationLabViewRTO.m` is a configuration file and `InitialConditionGasLift3.m` sets the initial conditions.

C.1 Main.m

```
% Runs a mock-up RTO problem, where the model and the system are equal and
    the disturbance setup is pre-determined.
% Other m-files required: none
% Subfunctions: none
% MAT-files required: none
clear
close all
clc

%noise seed
rng('default')
%% Simulation tuning
%parameters
parPlant = ParametersGasLiftModel;
```

```
14
15 %initial condition
16 [dxPlant0,zPlant0,uPlant0,thetaPlant0] = InitialConditionGasLift3(parPlant
       );
17
18 % setting up the disturbances
19 thetaPlantArray = load('thetaPlant'); %previously computed
20
21 % model integrators (scaled variables)
22 %[F,F_xk,F_pk, F_u, F_x] = ErosionRigDynModel(parPlant);
23 [F,S_xx,S_xp,L_x,L_u] = ErosionRigDynModelGrad(parPlant);
24
25 %% Run configuration file
26 InitializationLabViewRTO %here we use the same syntax as in the rig
27
28 %% creating simulation vector
29 %      d1      d2      d3
30 % w1: 45% -> 45% -> 35%
31 % w2: 45% -> 35% -> 35%
32 % w3: 35% -> 35% -> 35%
33 % 10 minutes each step -> 10*60 = 600 points
34 d_length = 600;
35
36 % time limits found manually
37 d45_theta = mean(thetaPlantArray.value(:,1:114),2);
38 d35_theta = mean(thetaPlantArray.value(:,741:834),2);
39
40 %creating disturbance array
41 theta_d1 = [d45_theta(1);
42              d45_theta(2);
43              d35_theta(3);
44              d45_theta(4);
45              d45_theta(5);
46              d35_theta(6)].*ones(6,d_length);
47
48 theta_d2 = [d45_theta(1);
49              d35_theta(2);
50              d35_theta(3);
51              d45_theta(4);
52              d35_theta(5);
53              d35_theta(6)].*ones(6,d_length);
54
55 theta_d3 = [d35_theta(1);
56              d35_theta(2);
57              d35_theta(3);
58              d35_theta(4);
```

```
59                  d35_theta(5);
60                  d35_theta(6)].*ones(6,d_length+300);
61
62  thetaArray = [theta_d1,theta_d2,theta_d3];
63
64  distArray = [[0.45;0.45;0.35;0.4].*ones(4,d_length),
        [0.45;0.35;0.35;0.4].*ones(4,d_length), [0.35;0.35;0.35;0.4].*ones(4,
        d_length + 1+300)];
65
66  %% Simulation initialization
67  %simulation parameters
68  nInit = 1; %[s]
69  nFinal = size(thetaArray,2); %[s]
70  deltaT = 1; %[s]
71  parPlant.T = 1; %[s]
72  tgrid = (nInit:parPlant.T:nFinal)/60; %one measurements per second
73  %% Control parameter
74  KI1 = 0.02;
75  KI2 = 0.02;
76  KI3 = 0.01;
77  N = 30;
78  %Setting up square wave dither
79  a = 0.01;
80  freq1=6;
81  freq2=14;
82  freq3=10;
83
84  offset=0;
85  amp=a;
86  duty=50;
87  t=0:deltaT:nFinal+10;
88  sq_wave1=offset+amp*square(freq1.*t,duty);
89  sq_wave2=offset+amp*square(freq2.*t,duty);
90  sq_wave3=offset+amp*square(freq3.*t,duty);
91
92  dither1 = sq_wave1;
93  dither2 = sq_wave2;
94  dither3 = sq_wave3;
95  %% Control limits
96  w_gl_totmax = 5;
97  w_gl_max = 3;
98  w_gl_min = 1;
99
100 %to make sure that the pertrubation do not get us over max limit
101 w_gl_totmax = 5 - 3*a;
102 %% Run mock-up loop
```

```
103 % arrays for plotting
104 flagArray = [];
105 SSDArray = [];
106 ofArray = [];
107 thetaHatArray = [];
108 xEstArray = [];
109 xOptArray = [];
110 uOptArray = [];
111 uImpArray = [];
112
113 xPlantArray = [];
114 zPlantArray = [];
115 measPlantArray = [];
116 ofPlantArray = [];
117
118 JPlantArray = [];
119 tot_gas = [];
120 O_vectorArray =[];
121 Ju_hatArray = [];
122 d_ofPlantArray = [];
123 sumOArray = [];
124 % initializing simulation
125 % states
126 dxk = dxPlant0;
127 zk = zPlant0;
128 Ju_hat = [0;0;0;0];
129
130 % inputs
131 O_vector = uPlant0(1:3)'; % we use this name for consistency with file
        LabViewMPC.m
132 uk = [O_vector';distArray(:,1)];
133 uPlantArray = uk;% in order to avoid shifting the input array one step
        ahead
134
135 for kk = 1:nFinal
136
137     % printing the loop evolution
138       fprintf('      kk >>> %6.4f [min]\n',tgrid(kk))
139
140     % integrating the system (the model sampling time is defined by
141     Fend = F('x0',dxk,'z0',zk,'p',[uk;thetaArray(:,kk);parPlant.T]); %
        model with the updated parameters
142
143     %extracting the results (from symbolic to numerical)
144     dxk = full(Fend.xf);
145     zk = full(Fend.zf);
```

```
    J= full(Fend.qf); %- (sum(O_vector)-8);

   %computing the gradients
    S = -full(S_xx(dxk,zk,uk,[thetaArray(:,kk);parPlant.T]))\full(S_xp(dxk,zk,uk,[thetaArray(:,kk);parPlant.T]));
    dqdu = full(L_x(dxk,zk,uk,[thetaArray(:,kk);parPlant.T]))*S + full(L_u(dxk,zk,uk,[thetaArray(:,kk);parPlant.T]));

    xPlantArray = [xPlantArray, dxk];
    zPlantArray = [zPlantArray, zk];
    measPlantArray = [measPlantArray, par.H*zk + 0.01*[0.1;0.1;0.1;0.001;0.001;0.001].*randn(6,1)]; %adding artificial noise to the measurements
    ofPlantArray = [ofPlantArray, 2];
    JPlantArray = [JPlantArray, J];
    O_vectorArray =[O_vectorArray, O_vector'];
    Ju_hatArray = [Ju_hatArray, Ju_hat];
    d_ofPlantArray = [d_ofPlantArray, dqdu(1:3)'];
    sumOArray = [sumOArray, sum(O_vector)];

    if kk>N
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %!!! I'm rearranging all the vectors here, so can the be exactly%
        %the same as in the actual rig                                  %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % values of the measured variables for the last 30 seconds (dim = ny[12] X 30)
         % 9: FI-101 [l/min]
         %11: FI-102 [l/min]
         %13: FI-103 [l/min]
         % 3: FIC-104 [sl/min]
         % 5: FIC-105 [sl/min]
         % 7: FIC-106 [sl/min]
         %14: dP-101 [mbar D]
         %15: dP-102 [mbar D]
         %16: dP-103 [mbar D]
         %18: PI-101 [mbar G]
         %20: PI-102 [mbar G]
         %22: PI-103 [mbar G]

        I_vector = [measPlantArray(1,(kk-(N-1)):kk);
                    measPlantArray(2,(kk-(N-1)):kk);
                    measPlantArray(3,(kk-(N-1)):kk);
                    O_vectorArray(1,(kk-(N-1)):kk);
                    O_vectorArray(2,(kk-(N-1)):kk);
                    O_vectorArray(3,(kk-(N-1)):kk);
```

```
187                     %uPlantArray(3,(kk-29):kk);
188                     ones(3,N); %dummy values --> in the actual rig, they
    will be DP
189                     measPlantArray(4,(kk-(N-1)):kk);
190                     measPlantArray(5,(kk-(N-1)):kk);
191                     measPlantArray(6,(kk-(N-1)):kk)];
192         % values of the inputs (gas lift) of the last optimization run (
    dim = nQg[3] X 1)
193         O_vector = uPlantArray(1:3,kk - 1);
194
195         % values of the controlled variables (dim = nu[7] X 1)
196         P_vector = [uk(4);
197                     uk(5);
198                     uk(6);
199                     uk(7)
200                     uk(1);
201                     uk(2);
202                     uk(3)];
203
204         %Adding values into buffer
205         u_buffer = [uPlantArray(1,(kk-(N-1)):kk);
206                     uPlantArray(2,(kk-(N-1)):kk);
207                     uPlantArray(3,(kk-(N-1)):kk)];
208
209         J_buffer = JPlantArray(1,(kk-(N-1)):kk);
210
211         % running optimization code
212         LabViewES7
213
214     else
215         Ju_hat = [0;0;0;0];
216     end
217
218     %Adding dither to input
219     input = [O_vector(1) + dither1(kk), O_vector(2)+ dither2(kk), O_vector
    (3) + dither3(kk)];
220     O_vector = input;
221
222     uk = [O_vector'; distArray(:,kk + 1)];
223
224     if kk ~= length(tgrid) % in order to avoid shifting the input array
    one step ahead
225         uPlantArray = [uPlantArray, uk];
226     end
227 end
228
```

```
229 tgridRTO = (31:10:nFinal)/60; %one measurements per second
230
231 %Previous calculated optimum
232 load('RTO_results_unc2.mat');
233 save(name,'tgridRTO','xEstArray','xOptArray','uPlantArray','thetaHatArray'
        ,'SSDArray','flagArray','uOptArray','uImpArray','ofArray');
234
235 tgridRTO = (31:10:nFinal+10)/60;
236 r1= [uImpArray(1,:), uImpArray(1,end)*ones(1,31)];
237 r2= [uImpArray(2,:), uImpArray(2,end)*ones(1,31)];
238 r3= [uImpArray(3,:), uImpArray(3,end)*ones(1,31)];
239 uImpArray = [r1;r2;r3];
240 %%
241 ResultsPlotting
```

C.2 LabViewES7.m

```
1  % Main program
2  % Run Initialization file first
3
4  %%%%%%%%%%%%%%%%%%%
5  % Get Variables
6  %%%%%%%%%%%%%%%%%%%
7  % % setpoint of liquid flowrate
8  % ys1 = vector(1);
9  % ys2 = vector(2);
10 % ys3 = vector(3);
11
12 % disturbances
13 % if you want to convert to 0 (fully closed) to 1 (fully open)
14 % measurement already in between [0, 1]
15 cv101 = P_vector(1);
16 cv102 = P_vector(2);
17 cv103 = P_vector(3);
18 % changing measurement to %
19 pRate = P_vector(4)*100;
20
21
22 % always maintain the inputs greater than 0.5
23 %inputs computed in the previous MPC iteration
24 % Note that the inputs are the setpoints to the gas flowrate PID's
25 fic104sp = O_vector(1);
26 fic105sp = O_vector(2);
27 fic106sp = O_vector(3);
28 %current inputs of the plant
```

```
u0old=[P_vector(5);P_vector(6);P_vector(7)];

% liquid flowrates [L/min]
fi101 = I_vector(1,:);
fi102 = I_vector(2,:);
fi103 = I_vector(3,:);

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

% DP of the erosion boxes [mbar]
dp101 = I_vector(7,:);
dp102 = I_vector(8,:);
dp103 = I_vector(9,:);

% top pressure [mbar g]
% for conversion [bar a]-->[mbar g]
% ptop_n = ptop*10^-3 + 1.01325;
pi101 = (I_vector(10,:) - 1.01325)*10^3;
pi102 = (I_vector(11,:) - 1.01325)*10^3;
pi103 = (I_vector(12,:) - 1.01325)*10^3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CODE GOES HERE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
yPlantArray = [fi101;fi102;fi103];


    yPlant = [fi101;
              fi102;
              fi103;
              ];

    uPlant = [fic104; %conversion [L/min] --> [kg/s]
              fic105;
              fic106];

    O_vector = vertcat(fic104sp,fic105sp,fic106sp)';

    %GRADIENT ESTIMATIOn
    Ju_hat = LSE2(J_buffer, u_buffer,N);

    %If the gradient estimation does not give a result
    if isnan(Ju_hat)==1
```

```
75          Ju_hat = [0; 0; 0;0];
76      end
77
78
79      flagEst =1;
80
81      if flagEst == 1
82              %Checking maximum flow
83              [M,I]=max(O_vector);
84
85
86              if I==1
87                  O_vector = [fic104sp + KI1 * Ju_hat(1);
88                          fic105sp + KI2 *(Ju_hat(2)-Ju_hat(1));
89                          fic106sp + KI3 *(Ju_hat(3)-Ju_hat(2))]';
90
91                  %Check max and min constraint on input
92                  for i=1:3
93                      if O_vector(i)<w_gl_min
94                          O_vector(i)=w_gl_min;
95                      elseif O_vector(i)>w_gl_max
96                          O_vector(i) = w_gl_max;
97                      end
98                  end
99
100                 %Check max gas cap constraint
101                 if O_vector(3)+O_vector(2)>w_gl_totmax
102                     tot = O_vector(3)+O_vector(2);
103                     O_vector(3) = O_vector(3) - (tot-(w_gl_totmax -
    w_gl_min))*O_vector(3)/tot;
104                     O_vector(2) = O_vector(2) - (tot-(w_gl_totmax -
    w_gl_min))*O_vector(2)/tot;
105                 end
106
107                 if O_vector(3)<w_gl_min
108                     O_vector(2)=O_vector(2)-(w_gl_min-O_vector(3));
109                     O_vector(3)=w_gl_min;
110                 elseif O_vector(2)<w_gl_min
111                     O_vector(3)=O_vector(3)-(w_gl_min-O_vector(2));
112                     O_vector(2)=w_gl_min;
113                 end
114
115                 if sum(O_vector)>w_gl_totmax
116                     O_vector(1) = w_gl_totmax - O_vector(2)-O_vector(3);
117                 end
118
```

```
119             elseif I==2
120                 O_vector = [fic104sp + KI1 * (Ju_hat(1)-Ju_hat(3));
121                             fic105sp + KI2 * Ju_hat(2);
122                             fic106sp + KI3 * (Ju_hat(3)-Ju_hat(2))]';
123                 %Check max and min constraint on input
124                 for i=1:3
125                     if O_vector(i)<w_gl_min
126                         O_vector(i)=w_gl_min;
127                     elseif O_vector(i)>w_gl_max
128                         O_vector(i) = w_gl_max;
129                     end
130                 end
131 
132                 %Check max gas cap constraint
133                 if O_vector(1)+O_vector(3)>w_gl_totmax - w_gl_min
134                     tot = O_vector(1)+O_vector(3);
135                     O_vector(3) = O_vector(3) - (tot-(w_gl_totmax -
    w_gl_min))*O_vector(3)/tot;
136                     O_vector(1) = O_vector(1) - (tot-(w_gl_totmax -
    w_gl_min))*O_vector(1)/tot;
137                 end
138 
139                 if O_vector(3)<w_gl_min
140                     O_vector(1)=O_vector(1)-(w_gl_min-O_vector(3));
141                     O_vector(3)=w_gl_min;
142                 elseif O_vector(1)<w_gl_min
143                     O_vector(3)=O_vector(3)-(w_gl_min-O_vector(1));
144                     O_vector(1)=w_gl_min;
145                 end
146 
147                 if sum(O_vector)>w_gl_totmax
148                     O_vector(2) = w_gl_totmax - O_vector(1)-O_vector(3);
149                 end
150 
151             elseif I==3
152                 O_vector = [fic104sp + KI1 * (Ju_hat(1) - Ju_hat(2));
153                             fic105sp + KI2 *(Ju_hat(2)-Ju_hat(3));
154                             fic106sp + KI3 * Ju_hat(3)]';
155                 %Check max and min constraint on input
156                 for i=1:3
157                     if O_vector(i)<w_gl_min
158                         O_vector(i)=w_gl_min;
159                     elseif O_vector(i)>w_gl_max
160                         O_vector(i) = w_gl_max;
161                     end
162                 end
```

```
163
164             %Check max gas cap constraint
165             if O_vector(1)+O_vector(2)> w_gl_totmax-w_gl_min
166                 tot = O_vector(1)+O_vector(2);
167                 O_vector(1) = O_vector(1) - (tot-(w_gl_totmax -
    w_gl_min))*O_vector(1)/tot;
168                 O_vector(2) = O_vector(2) - (tot-(w_gl_totmax -
    w_gl_min))*O_vector(2)/tot;
169             end
170
171             if O_vector(1)<w_gl_min
172                 O_vector(2)=O_vector(2)-(w_gl_min-O_vector(1));
173                 O_vector(1)=w_gl_min;
174             elseif O_vector(2)<w_gl_min
175                 O_vector(1)=O_vector(1)-(w_gl_min-O_vector(2));
176                 O_vector(2)=w_gl_min;
177             end
178
179
180             if sum(O_vector)>w_gl_totmax
181                 O_vector(3) = (w_gl_totmax) - O_vector(2)-O_vector(1);
182             end
183         end
184
185         SS = 0;
186         Estimation = flagEst;
187         Optimization = 1;
188         Parameter_Estimation = thetaHat';
189         State_Variables_Estimation = (par.H*zEstHat)';
190
191 else
192     %%%%%%%%%%
193     %(dummy)%
194     %%%%%%%%%%
195     % compute new values for the gas flow rate setpoints
196     O_vector = vertcat(fic104sp,fic105sp,fic106sp)';
197
198     SS = 0;
199     Estimation = flagEst;
200     Optimization = 0;
201     Result = 0;
202     Parameter_Estimation = [0,0,0,0,0,0];
203     State_Variables_Estimation = [0,0,0,0,0,0];
204     State_Variables_Optimization = [0,0,0,0,0,0];
205     Optimized_Air_Injection = [0,0,0];
206 end
```

C.3 LSE2.m

```
function Ju_hat = LSE2(J_buffer, u_buffer,N)
N_buffer = N;
x = u_buffer';
y = J_buffer';

phi = [x ones(N_buffer,1)];
theta = inv(phi'*phi)*phi'*y;

Ju_hat = theta;
```

C.4 ResultsPlotting.m

```
%%%%%%%%%%%%%%%
% Plotting %
%%%%%%%%%%%%%%%
time = linspace(1,30,1800);
figure(1)
subplot(5,1,1)
    plot(tgrid,O_vectorArray(1,:))
    hold on
    plot(tgridRTO,uImpArray(1,:),'Linestyle','--', 'Color','r')
    title('Manuipluated Variable, \omega_{gl,1}')
    xlabel('time[min]')
    ylabel('L/min');
    legend('u_{Plant}', 'u_{Opt}')
    xlim([0,35])
    ylim([0.9 3.1])

subplot(5,1,2)
    plot(tgrid,O_vectorArray(2,:))
    hold on
    plot(tgridRTO,uImpArray(2,:),'Linestyle','--', 'Color','r')
    title('Manuipluated Variable, \omega_{gl,2}')
    xlabel('time[min]')
    ylabel('L/min');
    legend('u_{Plant}', 'u_{Opt}')
    xlim([0,35])
    ylim([0.9 3.1])

subplot(5,1,3)
    plot(tgrid,O_vectorArray(3,:))
    hold on
    plot(tgridRTO,uImpArray(3,:),'Linestyle','--', 'Color','r')
```

```
32      title('Manuipluated Variable,    \omega_{gl,3}')
33      xlabel('time[min]')
34      ylabel('L/min');
35      legend('u_{Plant}', 'u_{Opt}')
36      xlim([0,35])
37      ylim([0.9  3.1])
38
39  subplot(5,1,4)
40      plot(tgrid, JPlantArray)
41      hold on
42      title('Cost function')
43      xlabel('time[min]')
44      ylabel('cost')
45      xlim([0,35])
46
47  subplot(5,1,5)
48      plot(tgrid, sumOArray)
49      hold on
50      yline(5, 'Linestyle','--', 'Color','r')
51      title('Total gas lift')
52      legend('w_{gl,tot}','w_{gl,totmax}')
53      xlabel('time[min]')
54      ylabel('L/min')
55      xlim([0,tgrid(end)])
56      ylim([2.9  5.1])
57  %%
58  figure(11)
59      subplot(3,1,1)
60      plot(tgrid, d_ofPlantArray(1,:))
61      hold on
62      plot(tgrid, Ju_hatArray(1,:))
63      legend('real', 'estimate')
64      ylim([-10,10])
65
66      subplot(3,1,2)
67      plot(tgrid, d_ofPlantArray(2,:))
68      hold on
69      plot(tgrid, Ju_hatArray(2,:))
70      legend('real', 'estimate')
71      ylim([-10,10])
72
73      subplot(3,1,3)
74      plot(tgrid, d_ofPlantArray(3,:))
75      hold on
76      plot(tgrid, Ju_hatArray(3,:))
77      legend('real', 'estimate')
```

```
78      ylim([-10,10])
```

C.5 ParametersGasLiftModel.m

```
1  function  par = ParametersGasLiftModel
2
3  %number of wells
4  par.n_w = 3;
5  %gas constant
6  par.R = 8.314; %[m3 Pa/(K mol)]
7  %molecular weigth
8  par.Mw = 0.029; %[kg/mol] -- Attention: this unit is not usual
9
10 %% Properties
11 %density of oil - dim:  nwells x 1
12 par.rho_o = 996.57*ones(par.n_w,1); %[kg/m3] - water
13 %1cP oil viscosity
14 par.mu_oil = 1*0.000853*ones(par.n_w,1); %[Pa s or kg/(m s)]  - water
15 %Density Gas
16 par.rho_g = 1.2041*ones(par.n_w,1); %[kg/m3]
17 %riser temperature
18 par.T_r = 23+273; %[K]
19 %separator pressure
20 par.p_s = 1.01; %[bar]
21
22 %% Project
23 %well parameters - dim:  nwells x 1
24 %length
25 par.L_w = 1.8*ones(par.n_w,1); %[m]
26 %height
27 par.H_w  = 0*ones(par.n_w,1); %[m]
28 %diameter
29 par.D_w = 0.02*ones(par.n_w,1); %[m]
30 %well transversal area
31 par.A_w = pi.*(par.D_w/2).^2;%[m2]
32
33 %well below injection - [m]
34 par.L_bh = 0.4*ones(par.n_w,1);
35 par.H_bh = 0*ones(par.n_w,1);
36 par.D_bh = 0.02*ones(par.n_w,1);
37 par.A_bh = pi.*(par.D_bh/2).^2;%[m2]
38
39 %riser - [m]
40 par.L_r = 2.2*ones(par.n_w,1);
41 par.H_r = 2.2*ones(par.n_w,1);
```

```
42 par.D_r = 0.02*ones(par.n_w,1);
43 %riser areas
44 par.A_r = pi.*(par.D_r/2).^2;%[m2]
45
46 %% Upper limits
47 %Max gas lift flow
48 % par.qGLMax = 4; % [L/min]
49 %Max wellhead gas production rate
50 %par.QgMax = 7.5; % [L/min]  %CONSTRAINED
51 par.QgMax = 1000; % [L/min]  %UNCONSTRAINED
52
53 %% Estimation
54 %Previously calculated covariance
55 cov = [5.91227126883972e-06,-2.06202609409489e-07,-1.89554128806828e
       -07,-5.05804983673889e-08,2.01691604531746e-08,-1.46870984879245e
       -08;...
56       -2.06202609409489e-07,8.98504731564069e-06,-1.57793863671730e
       -07,-9.45999577341072e-09,1.58393325004350e-07,3.99881900108853e
       -08;...
57       -1.89554128806828e-07,-1.57793863671730e-07,5.86527519642299e
       -06,8.42373098367610e-09,-3.18596115962588e-08,1.02162653653725e
       -07;...
58       -5.05804983673889e-08,-9.45999577341072e-09,8.42373098367610e
       -09,1.07111470982306e-06,-6.58726935128263e-08,2.19056938187727e
       -08;...
59       2.01691604531746e-08,1.58393325004350e-07,-3.18596115962588e
       -08,-6.58726935128263e-08,1.30222960427062e-06,-5.39703493095107e
       -08;...
60       -1.46870984879245e-08,3.99881900108853e-08,1.02162653653725e
       -07,2.19056938187727e-08,-5.39703493095107e-08,1.10557288974228e-06];
61 %Sigma = eye(6)\cov;  %inv function sometimes is not the best for
       inverting matrix
62 par.Sigma = inv(cov);
```

C.6 InitialConditionGasLift3.m

```
1 %Initial Condition Simplified
2 function [dx0,z0,u0,theta0] = InitialConditionGasLift3(par)
3
4
5 %% Inputs
6 %gas lift rate
7 Q_gl_0 = [0.985530864197531;0.998111111111111;1.00266666666667]; %[L/min]
8 %valve oppening
9 vo_0 = [0.399999999999999;0.399999999999999;0.399999999999999]; %[0-1]
```

```
%velocity pump
vpump_0 = 35*1e-2;% [0-1]

%% States
%gas holdup %%%%%%%%%%%%%%%%%%
m_g_0 = [0.000301299301829417;0.000301176749723761;0.000304470148224803]*1
    e4;%[1e4 kg]
%oil holdup
m_o_0 = [1.00146717930556;1.00451794910589;0.998642151393710];%[kg]

%oil rate from reservoir %%%%%%%%%%%%%%%%%%%%%%%
w_ro_0 = [0.0653480683694890;0.0672010939853007;0.0655584758661665]*1e2; %
    [1e2 kg/s]
%riser head total production rate %%%%%%%%%%%%%%%%%%%%%%%%%
w_pr_0 = [0.0653677288514124;0.0672212423630201;0.0655784636053547]*1e2;%
    [1e2 kg/s]
%riser head pressure
p_rh_0 = [1.01573148143340;1.02781790122580;1.01499074068967];%[bar]
%pressure - below injection point (bottom hole)
p_bh_0 = [1.08871176278397;1.10260160139600;1.08839756290436];%[bar]
%mixture density in riser
rho_r_0 = [7.97182026020070;7.99609653965992;7.94936463516739];%[100 kg/m3
    ]
%density gas
rho_gr_0= [1.19694771443676;1.21119046737951;1.19607481844367];%[kg/m3]
% total gas
w_gr_0 = [1.96604819234686e-05;2.01483777193892e-05;1.99877391882142e
    -05]*1e5; %[1e5 kg/s]
% total liquid
w_lr_0 = [0.0653480683694890;0.0672010939853007;0.0655584758661665]*1e2; %
    [1e2 kg/s]

%% Parameters
res_theta_0 = [-54.0722272955321;-42.1710050834622;-51.8274028690567]*1e
    -1;
val_theta_0 = [0.967055060728985;0.563169656106628;1.041148135717951];

dx0 = vertcat(m_g_0,m_o_0);
z0 = vertcat(w_ro_0,w_pr_0,p_rh_0,p_bh_0,rho_r_0,rho_gr_0,w_gr_0,w_lr_0);
u0 = vertcat(Q_gl_0,vo_0,vpump_0);
theta0  = vertcat(res_theta_0,val_theta_0);
```

C.7 InitializationLabViewRTO.m

```
% Operate the rig under the initial inputs ussm
```

```
% Wait for the rig to become stable
%clear
%clc

%% SS detection Configuration
SSConf.sspn = 5; %Number of points in SS to assure that system is at SS

%SS detection for the whole plant (MPA)
SSConf.dss = 30; %length of the moving window used for SS identification
SSConf.ny = 3;
SSConf.rThres = [2.2;2.2;2.2];
SSConf.lambda1 = [0.05;0.1;0.1];
SSConf.lambda2 = [0.1;0.1;0.1];
SSConf.lambda3 = [0.1;0.1;0.1];

%% Model tuning
%paramters
par = ParametersGasLiftModel;
%initial condition
[x0,z0,u0,theta0] = InitialConditionGasLift3(par);

%states to measurement mapping function
par.nMeas = 6;
H = zeros(6,24);
H(1,1) = 1e-2*60*1e3/par.rho_o(1); %wro-oil rate from reservoir, well 1 [1
    e2 kg/s] --> [L/min]
H(2,2) = 1e-2*60*1e3/par.rho_o(2); %wro-oil rate from reservoir, well 2
H(3,3) = 1e-2*60*1e3/par.rho_o(3); %wro-oil rate from reservoir, well 3
H(4,7) = 1; %prh - riser head pressure well 1
H(5,8) = 1; %prh - riser head pressure well 2
H(6,9) = 1; %prh - riser head pressure well 3
par.H = H;

%% Optimization tuning
OptConf.lim = 0.1; %check proximity between new and old inputs, if too
    close, dont change
OptConf.ku = 1; % input filter

%% initializing estimation
xEstHat = x0;
zEstHat = z0;
thetaHat = theta0;
```