Dynamic Extremum Seeking Control Using ARX model

Jeongrim Ryu

Chemical Engineering
Submission date: June 2018
Supervisor: Sigurd Skogestad, IKP
Co-supervisor: Dinesh Krishnamoorthy, IKP

Norwegian University of Science and Technology
Department of Chemical Engineering

Summary

With the development of the oil production industry, a gas lift process has become one of the most famous artificial lift techniques. The lift gas rate and oil production rate relationship in the process has a clear maximum point which is called an extremum, and keeping the system in the extremum achieves a maximum profit where the gradient becomes zero. This kind of control strategy is called extremum seeking control(ESC).

In a classic ESC, it assumes a dynamic plant as a static model and estimates the gradient with a local linear static model resulting in a slower convergence to the extremum. This is the main drawback of the classic ESC. Thus, this disadvantage motivated to start this project with an idea that estimating the gradient with a local linear dynamic model will achieve much faster convergence. The new strategy is named a dynamic ESC using ARX model.

In the design of the dynamic ESC, a wave perturbation is decided to be not a sinusoidal wave but a pseudo-random binary sequence(PRBS) wave. The reason behind this is that diverse frequencies in the PRBS wave make the ARX model to estimate the gradient more efficiently. Moreover, it turned out that the ARX model often has numerical spikes on the estimated gradient. This problem could be revised by combining the ARX model with the least square(LS) method and named as a Modified dynamic ESC.

With the newly developed control methods, three case studies with different plant models are performed. The first case study uses parallel heat exchangers with one split model and the second and third case studies are a single gas-lifted oil well and multiple gas-lifted oil wells respectively. The result obtained by the dynamic ESC is compared with that obtained by the classic ESC, verifying the superior performance of the dynamic ESC. Additionally, disturbances are applied to test the control strategy.

Preface

This paper was written for TKP4900 Chemical Process Technology, Master's Thesis at Norwegian University of Science and Technology in spring 2018.

This project is greatly interesting since not only it is deeply related to the Advanced process control course but also the implementation of the ARX model for gradient estimation makes a clear improvement on the extremum seeking control. I would like to appreciate my supervisor, Sigurd Skogestad, and co-supervisor, Dinesh Krishnamoorthy, for offering me the opportunity to do this project. I am grateful for sharing their knowledge and answering my questions throughout this project.

It is a great honor to study and obtain my master's degree in NTNU. Thanks to the curriculum focusing not only on the intellectual development but also on the teamwork and self-development, the 2 years I spent in Trondheim is once in a lifetime experience for me.

06-2018, Trondheim, Norway

Jeongrim Ryu

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In oil production industries, gas lift on oil wells is one of the most popular techniques in artificial lift methods to improve oil production rate in mature fields as studied by Eikrem et al. (2008). A simple sketch of a single gas lift well is illustrated in Figure 1.1. In detail, the compressed lift gas is injected from surface to the bottom of the tubing through the annulus. This lift gas forms a mixture with the fluid from the reservoir and the newly formed fluid has a lower density than the fluid from the reservoir. As a result, the drop in density decreases downhole pressure and consequently increases the oil production rate.

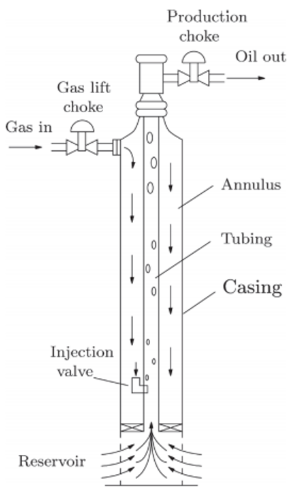

Figure 1.1: A gas lift oil well

However, Peixoto et al. (2015) explained that excessive gas injection has a counter-active effect on oil production rate. This is mainly because the injection of a large amount of gas in the fluid produces a large frictional pressure drop. This leads to the relationship between the gas injection rate and the oil production rate having a maximum point, so-called an extremum. This tendency is well illustrated by Aliev et al. (2015) through their study on mathematical modeling of gas lift process as shown in Figure 1.2.

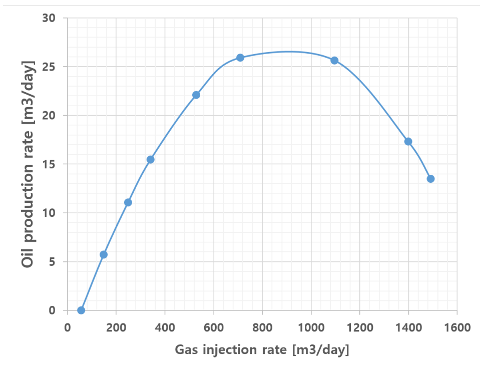

Figure 1.2: The dependence of oil production rate on the gas injection rate

With this background, Aliev et al. (2015) summarized that this kind of problem where the input-to-output relationship has an extremum and keeping the output around the maximum value is the main issue is called extremum control or self-optimizing control, or more precisely, extremum seeking control(ESC). According to Liu and Krstic (2012), extremum seeking can also be well explained as a non-model based real-time optimization method which can be used in a system having a minimum or maximum point where only limited knowledge is approachable.

The concept of ESC was firstly emerged by Leblanc (1922), and it obtained popularity after Krstić and Wang (2000) provided ”The first rigious proof of stability for an extremum seeking feedback scheme”. Diverse developments in ESC towards faster and exact convergence with sufficient stability are studied, see Hunnekens et al. (2014), Peixoto et al. (2015), and Krishnamoorthy et al. (2016). The most popular approach is based on the simple fact that the gradient becomes zero at the extremum as suggested in Krishnamoorthy et al. (2016). Specifically, the gradient estimation of the input-output map can lead the system towards the extremum by a continuous integrating process.

A classic extremum seeking controller optimizes the steady-state gradient in real time using a locally linear static model even though the plant is a nonlinear dynamic system, explained in Krstić and Wang (2000). The main drawback of this approach stems from using transient measurements for the estimation of a steady-state gradient. For an accurate steady-state gradient estimation, the process has to settle down to steady-state before it can be used. Almost all extremum seeking algorithm today assume a local linear static model to estimate the steady-state gradient around the current operating time which makes the convergence to the optimum very slow. This issue is the motivation to start this project to figure out a new algorithm to accomplish faster convergence in the extremum seeking controller.

The main idea to achieve the objective is using a local linear dynamic model to estimate the steady-state gradient providing a faster update of the estimated gradient instead of a local linear static model. Among diverse dynamic models, ARX model is introduced and it is named as a ”Dynamic extremum seeking controller”.

Chapter 2

Basic Theory

In this chapter, a literature review on the existing classic ESC will be structured first. mostly referring to Krstić and Wang (2000) and Krishnamoorthy et al. (2016). The basic feedback control scheme of the classic ESC will be included, and the drawbacks will be studied in more detail providing motivations to develop a new approach. This new approach is named as the dynamic ESC. The reason behind the naming is that the classic ESC assumes a plant as a static map while the dynamic ESC considers a plant as a dynamic model.

2.1 Classic Extremum Seeking Control

The classic ESC suggested by Krstić and Wang (2000) optimizes the steady state performance of the output in real time by adding an external dither signal on the input. Figure 2.1 shows the basic structure of the classic ESC process which is a gradient-based model in a discrete time setting with a sampling time T_s, see Krishnamoorthy et al. (2016).

Overall, the process includes a non-linear plant model, high pass filter(HPF), low pass filter(LPF), gradient estimation unit, integral controller($C(s)$) and a sinusoidal signal($asin\omega t$). The main procedure to find the extremum is estimating the gradient based on y values for each step steps in u and using a controller(integral controller in this case) with a set-point of the gradient at zero.

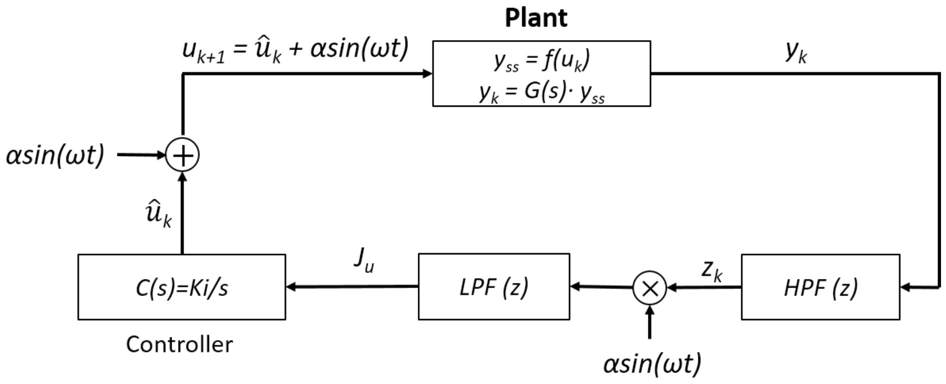

Figure 2.1: Classic ESC scheme implemented in discrete time

In terms of the plant model block, it consists of two parts, $y_{ss} = f(u_k)$ having a static non-linearity and $G(s)$ having a linear dynamic, suggested in Krstić and Wang (2000). This kind of plant model is called Hammerstein and Wiener models in Abd-Elrady and Gan (2008) as shown in Figure 2.2. The static gain of the plant($\frac{\delta y}{\delta u}$) should be maintained to be constant in one of the two blocks.

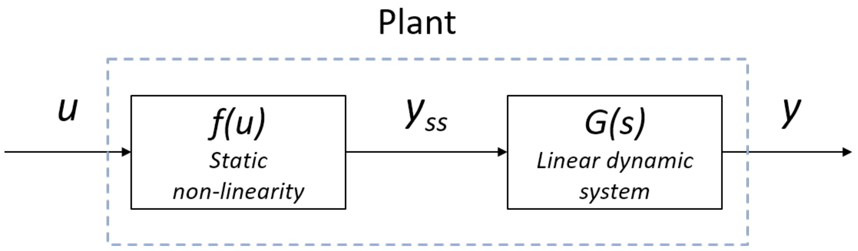

Figure 2.2: Hammerstein and Wiener models

Equation (2.1) to (2.3) are provided by Krishnamoorthy et al. (2016), explaining the high pass filter, the low pass filter, and the estimated optimizing variable respectively.

$$z_k = \frac{T_h}{T_s + T_h}[z_{k-1} + y_k - y_{k-1}] \tag{2.1}$$

$$\xi_{u,k} = (1 - \frac{T_s}{T_s + T_l})\xi_{u,k-1} + \frac{T_s}{T_s + T_l} z_k \alpha sin(\omega t) \tag{2.2}$$

$$\text{Where} \quad J_{u,k} = \frac{\alpha}{2}\xi_{u,k}$$

$$\hat{u}_k = u_{k-1} + T_s K_i J_{u,k} \tag{2.3}$$

In detail, when the changes in the output $y_k - y_{k-1}$ enters the high pass filter, signals with higher frequency than a cut-off time period T_h can only pass the filter removing a low-frequency part resulting in z_k. This can also be explained as the DC component of y is subtracted and y_k is moved to have zero mean according to Krstić and Wang (2000).

After that, $asin\omega t$ is multiplied to z_k and it enters the low pass filter with a cut-off time constant T_l. Similarly, a DC component of the two sinusoidals $J_{u,k}$ comes out which is an estimated gradient, see Krstić and Wang (2000).

Finally, the integral controller generates an estimated input $\hat{u}_{k+1}$ using the optimizing variable $J_{u,k}$, looking for the gradient to be zero with a controller gain K_i. As a consequence, the new input u_{k+1} is updated with sine perturbations.

However, there are some drawbacks in the classic ESC suggested in Krishnamoorthy et al. (2016). The first and most important disadvantage in the classic ESC is its slow transients to the new optimum after being disturbed. This slow dynamic is due to the fact that the input(u_k) and output(y_k) data are dynamic data but the classic ESC uses a local linear static model to estimate a steady state gradient. Specifically, the local linear static model uses only parts of data reached a steady state and throws others away. This wastes not only time to reach the steady state but also data obtained before the steady state. This is the reason behind the fact that the local linear static model cannot efficiently estimate a nonlinear dynamic plant system. As briefly mentioned in the chapter 1, this is the motivation to start this project.

The second disadvantage is that the classic ESC loses its robustness when there are disturbances with large amplitudes which changes frequently, which is well motivated by Krishnamoorthy et al.Krishnamoorthy et al. (2016) by introducing a disturbance rejection block. To guarantee the stability of the dynamic ESC, the robustness towards large disturbances has to be studied after development.

The last problem is originated from controller tuning. There are mainly five tuning parameters, the high pass filter frequency, the low pass filter frequency, the amplitude and frequency of sinusoidal perturbations, and the controller gain. In some cases, inadequately decided tuning parameters seriously influence the accuracy and convergence speed of the classic ESC, see Nešić (2009). Additionally, there is a controversial issue on the decision of the perturbation frequency. In specific, Nešić (2009) observed that the sine wave frequency should be sufficiently small or large, while Tan et al. (2013) insisted that it has to be located between the high pass filter frequency and low pass filter frequency. Thus, it will be challenging to select different tuning parameters for achieving a fast and accurate convergence in the classic ESC.

Even though various kinds of the ESC have been introduced such as using 1st-order least-square method for gradient estimation provided in Hunnekens et al. (2014), all of them has the same disadvantage caused from applying a static model to estimate a dynamic system.

Therefore, introducing a dynamic model to estimate a nonlinear dynamic system in the ESC will achieve much faster transients to the new optimum. As already mentioned, this new approach is named as the dynamic extremum seeking controller or simply the dynamic ESC. The dynamic ESC has one more advantage that the ARX model is capable of estimating not only the gradient but also the time constant in transfer functions while the classic ESC can only estimate the gradient. It is well described in 2.2.1.

A deep understanding of the dynamic ESC requires some knowledge of the ARX model which is the most important idea and basis for this project, which will be stated in the chapter 2.2.1.

2.2 Dynamic Extremum Seeking Control

Overall, the dynamic ESC is based on the same idea with the classic ESC that the gradient is estimated first, and a controller updates a new input value which can achieve the zero set-point for the gradient.

The main difference between the dynamic ESC from the classic ESC is the gradient estimation method. In detail, the slow transient to a new optimum in the classic ESC results from the gradient estimation part. As already mentioned in the chapter 2.1, the data of u_k and y_k are dynamic but the classic ESC uses a static model. Based on this fact, using a dynamic model instead of a static model is considered since it can handle the input and output data more efficiently for the gradient estimation. To realize this idea, ARX model, which is a local linear dynamic model is considered in the dynamic ESC which is expected to achieve a faster update of the estimated gradient.

Figure 2.3 illustrates a schematic control structure for the dynamic ESC. As the classic ESC suggested by Krstić and Wang (2000), Hammerstein and Wiener's models are used to describe a plant in a simple way. In terms of the perturbation, $\alpha sin\omega t$ is applied firstly, however, it is replaced with a pseudo-random binary sequence(PRBS) wave a wave with diverse frequencies can estimate the ARX model better.

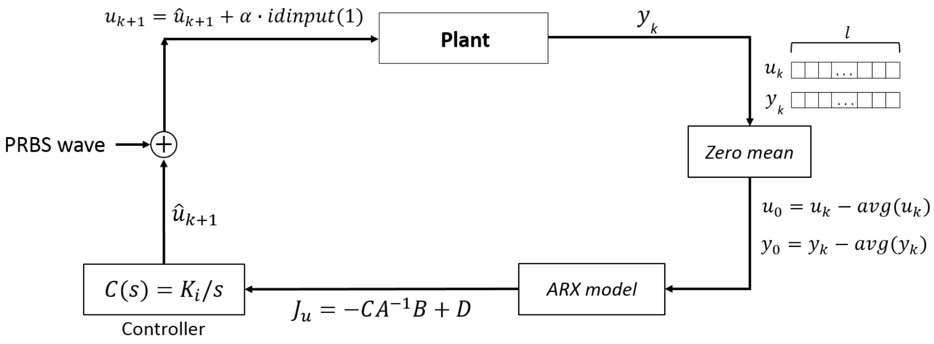

Figure 2.3: Dynamic ESC scheme using ARX method

A deep understanding of the dynamic ESC requires some knowledge of the ARX model which is the most important idea and basis for this study, which will be stated in the chapter 2.2.1. With the basis, the algorithm will be provided in 2.2.2.

2.2.1 ARX model

Selecting models of dynamical systems is well identified in Ljung (1998). A basic linear dynamic model with additive disturbance in discrete time system is specified as:

$$y(t) = G(q)u(t) + H(q)e(t) \tag{2.4}$$

where

$$G(q) = \sum_{k=1}^{\infty} g(k)q^{-k}, \quad \text{and} \quad H(q) = 1 + \sum_{k=1}^{\infty} h(k)q^{-k}$$

In Equation (2.4), $G(q)$ describes a relationship between the input $u(t)$ and the output $y(t)$, and it is called the transfer function. The other term $H(q)e(t)$ is an additive term at the output comes from the disturbances where $e(t)$ is white noise.

In a finite number of values, $g(k)$ and $h(k)$ can be considered as coefficients which should be determined, and they can simply denoted by the vector θ. Thus the model can be described as:

$$y(t) = G(q, \theta)u(t) + H(q, \theta)e(t) \tag{2.5}$$

ARX model is a family of transfer function models which parametrizing $G(q, \theta)$ and $H(q, \theta)$. It starts with specifying the input and output relationship as a linear difference equation:

$$y(t) + a_1 y(t-1) + ... + a_{n_a} y(t-n_a) = b_1 u(t-1) + ... + b_{n_b} u(t-n_b) + e(t) \quad (2.6)$$

where

$$\theta = [a_1 \quad a_2 \quad ... \quad a_{n_a} \quad b_1 \quad b_2 \quad ... \quad b_{n_b}]^T$$

Additionally, by introducing some terms:

$$A(q) = 1 + a_1 q^{-1} + ... + a_{n_a} q^{-n_a}, \quad \text{and} \quad B(q) = b_1 q^{-1} + ... + b_{n_b} q^{-n_b}$$

This makes it possible to represent G and H as rational functions:

$$G(q,\theta) = \frac{B(q)}{A(q)}, \quad H(q,\theta) = \frac{1}{A(q)} \quad (2.7)$$

The polynomial parameter(θ) estimation is performed by a least square method. When the ARX model finishes estimation providing θ, the dynamic discrete-time system model can be converted into the dynamic discrete state-space model and dynamic continuous time state-space system in sequence yielding A, B, C and D matrices in a state space representation:

$$\dot{x} = Ax + Bu, \quad \text{and} \quad y = Cx + Du \quad (2.8)$$

Where $\cdot$ is a notation for differentiation with respect to time. This process is explained algorithmically in detail in 2.2.2.

Thus, the estimated gradient($\hat{J}_u$) can be easily calculated by:

$$\hat{J}_u = \frac{dy}{du} = -CA^{-1}B + D \tag{2.9}$$

Based on this ARX model structure, how the estimated A, B, C and D matrices in Equation 2.8 are linked with variables in transfer functions will be explained. Both the first and second-order transfer functions are introduced to give a better understanding.

First-order transfer function

First-order ARX model is used for estimating a system containing a first-order transfer function as Equation (2.10).

$$G(s) = \frac{k}{\tau s + 1} \tag{2.10}$$

And the corresponding ARX model is

$$y(t) + a_1 y(t-1) = b_1 u(t-1) + e(t) \tag{2.11}$$

where

$$\theta = [a_1 \quad b_1]^T$$

Parameters a_1 and b_1 are the two estimated variables by the 1st order ARX model, and it gives A, B, C, and D matrices, which are $[1 \times 1]$ in this case.

In regards to the plant model structure, the static non-linearity function $f(u)$ gives an input y_{ss} for $G(s)$. Thus, the relationship between y and y_{ss} can be stated in Fourier domain as:

$$y = G(s) \cdot y_{ss} = \frac{1}{\tau s + 1} y_{ss} \tag{2.12}$$

By a transformation from Fourier domain to time domain yields:

$$\tau \dot{y} + y = y_{ss} \tag{2.13}$$

By rearrangement,

$$\dot{y} = \frac{y_{ss} - y}{\tau} \tag{2.14}$$

Put $y = x$,

$$\dot{x} = \frac{y_{ss} - x}{\tau} = -\frac{1}{\tau} x + \frac{k}{\tau} y_{ss} \tag{2.15}$$

Equation (2.8) can be written as a matrix form:

$$\begin{bmatrix} \dot{x} \\ y \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x \\ y_{ss} \end{bmatrix} \tag{2.16}$$

and comparing Equation (2.16) with $y = x$ and Equation (2.15) yields

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} -\frac{1}{\tau} & \frac{k}{\tau} \\ 1 & 0 \end{bmatrix} \tag{2.17}$$

In conclusion, the first-order ARX model estimates 2 parameters a_1 and b_1, or in its final analysis, it estimates the gain of the system k and time constants τ.

Second-order transfer function

Second-order ARX model is used for estimating a system containing a second-order transfer function as Equation (2.18).

$$G(s) = \frac{\tau_a s + 1}{(\tau_1 s + 1)(\tau_2 s + 1)} = \frac{\gamma s + 1}{\alpha s^2 + \beta s + 1} \tag{2.18}$$

And the corresponding second-order ARX model is

$$y(t) + a_1 y(t-1) + a_2 y(t-2) = b_1 u(t-1) + b_2 u(t-2) + e(t) \tag{2.19}$$

where

$$\theta = [a_1 \quad a_2 \quad b_1 \quad b_2]^T$$

Parameters a_1, a_2, b_1, and b_2 are the four estimated variables by the 2nd order ARX model, and it gives A,B,C, and D matrices.

Similar with the 1st order transfer function, y_{ss} is the input for G(s). In Fourier domain,

$$y = G(s) \cdot y_{ss} = \frac{\gamma s + 1}{\alpha s^2 + \beta s + 1} y_{ss} \tag{2.20}$$

Equation (2.20) can be rearranged with respect to $\frac{y}{y_{ss}}$, and $z(s)$ is introduced for simplifying a conversion to time domain,

$$\frac{y}{y_{ss}} = \frac{(\gamma s + 1)z(s)}{(\alpha s^2 + \beta s + 1)z(s)} \tag{2.21}$$

Converting Equation (2.21) to time domain yields:

$$y_{ss} = \alpha z + \beta \dot{z} + z \tag{2.22}$$

and

$$y = r\dot{z} + z \tag{2.23}$$

Put

$$x_1 = \dot{z}, \quad \text{and} \quad x_2 = z$$

where the relationship between $\dot{x_2}$ and x_1 becomes

$$\dot{x_2} = x_1 \tag{2.24}$$

Then, a set of equations can be obtained by applying x_1 and x_2 in Equation (2.22) and (2.23) as

$$y_{ss} = \alpha \dot{x_1} + \beta x_1 + x_2 \tag{2.25}$$

$$y = \gamma x_1 + x_2 \tag{2.26}$$

Equation (2.25) can be rearranged in terms of $\dot{x_1}$ as

$$\dot{x_1} = -\frac{\beta}{\alpha}x_1 - \frac{1}{\alpha}x_2 + \frac{u}{\alpha} \tag{2.27}$$

Meanwhile, Equation (2.15) can be written in a second-order matrix form as

$$x = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = A \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + By_{ss} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} y_{ss} \tag{2.28}$$

$$y = C \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + Dy_{ss} = c_1x_1 + c_2x_2 + dy_{ss} \tag{2.29}$$

Thus, comparing Equation (2.24), (2.26), and (2.27) with Equation (2.28) and (2.29) provides A,B,C, and D matrices as

$$A = \begin{bmatrix} -\frac{\beta}{\alpha} & -\frac{1}{\alpha} \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} -\frac{\tau_1+\tau_2}{\tau_1\cdot\tau_2} & -\frac{1}{\tau_1\cdot\tau_2} \\ 1 & 0 \end{bmatrix} \tag{2.30}$$

$$B = \begin{bmatrix} \frac{1}{\alpha} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\tau_1\cdot\tau_2} \\ 0 \end{bmatrix} \tag{2.31}$$

$$C = \begin{bmatrix} \gamma & 1 \end{bmatrix} = \begin{bmatrix} \tau_a & 1 \end{bmatrix} \tag{2.32}$$

$$D = 0 \tag{2.33}$$

In conclusion, the second-order ARX model estimates 4 parameters a_1, a_2, b_1, and b_2. At the same time, it can also be explained to estimate the system gain k and time constants τ_a, τ_1, and τ_2 in its last analysis.

Therefore, from the analysis on the linkage between estimated variables and parameters in transfer functions, the second advantage of the dynamic ESC, ”it is possible to estimate a controller gain and time constants in transfer functions”, is well explained.

2.2.2 Algorithm

A basic algorithm for the dynamic ESC is displayed in Algorithm 1 which is describing the dynamic ESC procedures performed in Figure 2.3. Specifically, the input and output values in the plant are filtered out by low pass filter and sets of data with a window size(l) are built up. The filtering is not necessary if noise is not considered. The sets of data with a window size(l) pass the zero mean unit where the average value is subtracted from the data sets and they are moved to have a zero mean.

After that, the ARX model estimates parameters in a dynamic discrete time system, and converts them into a state-space model in discrete time with *idss* generating A, B, C, and D matrices. Then, the parameters are converted into a dynamic continuous time state-space system by *d2c*, and the estimated steady-state gradient is calculated by $J_u = -CA^{-1}B + D$.

The estimated gradient is a controlled variable which has a setpoint of zero. A simple integral controller is used to update the input with a controller gain K_i generating $\hat{u}_{k+1}$, and a pseudo-random binary sequence(PRBS) wave perturbation is added yielding u_{k+1}.

While the classic ESC uses a sinusoidal wave perturbations $\alpha sin(\omega t)$, a PRBS wave is applied in the dynamic ESC. The reason will be explained in section 3.1.

Main advantages of the dynamic ESC are (1) it converges faster to a new extremum, (2) it uses all dynamic data for gradient estimation, (3) it is possible to estimate a controller gain and time constants in transfer functions, (4) the tuning process is easier than the classic ESC.

Algorithm 1 The dynamic ESC using ARX model

```
input: $y_k$,$u_k$
1: for $k = 1 \rightarrow n$ do
2:     Data of the input and output values are built up with a window size $l$
3:             vector   $u_k \leftarrow [u_{k-l+1}, u_{k-l+2}, ..., u_k]$
4:             vector   $y_k \leftarrow [y_{k-l+1}, y_{k-l+2}, ..., y_k]$
5:     Moved to have a zero mean
6:             $y_0 \leftarrow y_k - avg(y_k)$
7:             $u_0 \leftarrow u_k - avg(u_k)$
8:     Gradient estimation with the ARX model using data sets of $y_0$ and $u_0$
9:             $data \leftarrow iddata(y_0, u_0, T_s)$
10:            $[A(q) \quad B(q)] \leftarrow arx[data, [1, 1, 0]]$   for 1st-order systems
11:            $[A(q) \quad B(q)] \leftarrow arx[data, [2, 2, 0]]$   for 2nd-order systems
12:            $[A_d \quad B_d \quad C_d \quad D_d] \leftarrow idss[A(q) \quad B(q)]$
13:            $[A \quad B \quad C \quad D] \leftarrow d2c[A_d \quad B_d \quad C_d \quad D_d]$
14:            $J_{u,k} \leftarrow -CA^{-1}B + D$
15:    Integral Controller
16:            $\hat{u}_{k+1} \leftarrow u_k + K_i J_{u,k}$
17:    Perturbation
18:            $u_{k+1} \leftarrow \hat{u}_{k+1} + prbs$
19: end for
```

Chapter 3

Design of the Dynamic ESC

In this chapter, the procedure to design the dynamic ESC will be explained. Since almost all issues are caused by the model in the Case study 2, it is highly recommended to see the process described in the section 5.1 to achieve a better understanding of the process.

In short, the issues happen when the plant system changes from the first-order system to the second-order system. The reasonable explanations stem from the difference in their step responses and the ARX model.

3.1 Decision of wave perturbations

When the dynamic ESC is developed, general steps used in this study can be listed as below.

1. Apply a step change in the input and monitor the output changes.
   This is to check if the plant model works properly.
2. Implement the ARX model and wave perturbations with a zero controller gain.
   This is to check if the ARX model estimates the steady-state gradient as intended.
3. Turn on the controller and adjust tuning parameters such as a controller gain, window size, etc.

The steps are applied in Hammerstein-Weiner plant models with three different transfer functions which are first-order transfer function, second-order transfer function without zero, and second-order transfer function with zero. Please see chapter 5.1 for more information on the plant process.

In brief, a sine wave perturbation works very well in the first-order system but not in the second-order without zero system where the static non-linearity is illustrated as $f(u) = -0.1(u-20)^2 + 45$ and the dynamic part is depicted as $G(s) = \frac{1}{(174s+1)(60s+1)}$. The simulation result of the step 2 where the controller is turned off is illustrated in the Figure 3.1. The blue solid lines are indicating the original input and output value containing noise, and the orange dashed lines mean the steady-state values which are calculated from the plant model.

In the Figure, the input and output move with sine wave perturbations but not updated to a new value and the true steady-state gradient equals to 1. However, it is clearly indicated that the steady-state gradient J_u is estimated as 0 by the ARX model. Even though sine wave perturbations with different frequencies and amplitudes are applied and other tuning parameters are tried to change as well to solve the problem, none of them could make the situation better.

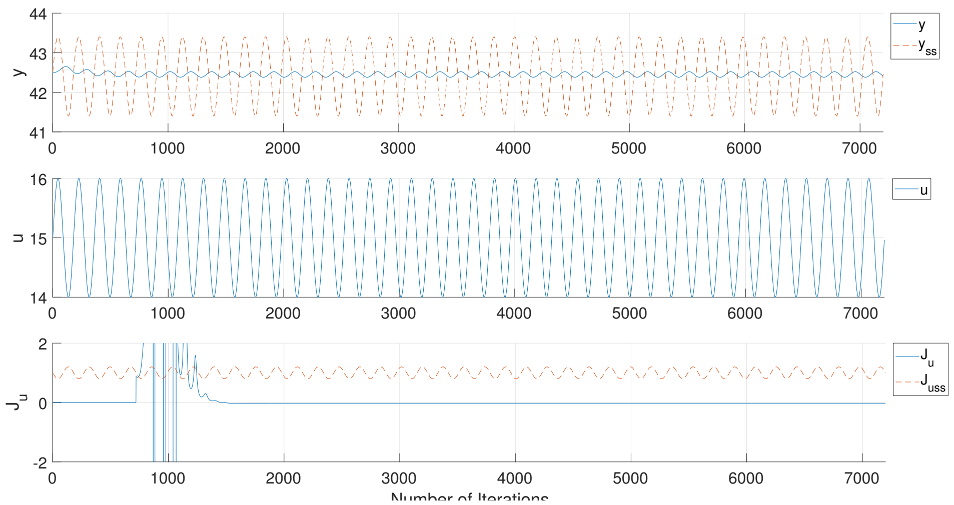

Figure 3.1: Simulation result for a single gas-liftd oil well model, second-order without zero system using a dynamic ESC with sinusoidal perturbation: no control

Based on the fact that there was no such a problem in the first-order system, an approach for figuring out the cause of this matter comes from the difference between the first and second-order system.

In specific, the step responses of the three different transfer functions are depicted in Figure 3.2. The red dash-dot line is a step response to the second-order transfer function without zero where the problem happened. With the figure below with zoomed-in in axis, it is shown that there is almost no change in the response at the starting point due to the secondary time lag response. Thus, sine waves with one frequency are discussed not to be sufficient to obtain full information about y, so that the ARX model could not estimate the gradient.

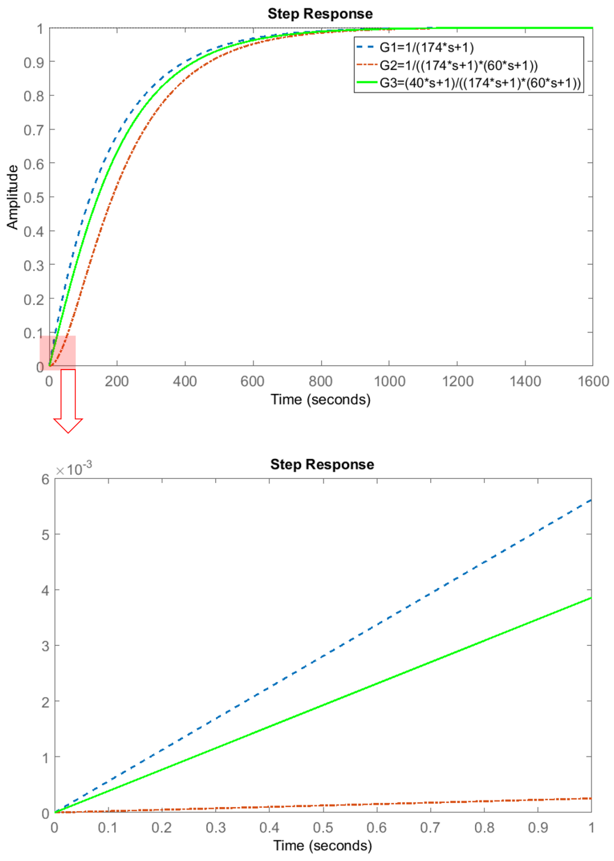

Figure 3.2: Step responses of first and second-order transfer functions

To solve this problem, a pseudo-random binary sequence(PRBS) is introduced instead of a simple sine wave. The PRBS generates a binary sequence with a vector length of N, which is set to be 1 in this project and it can be easily presented by *idinput(N)* in Matlab.

In principle, the PRBS wave perturbation can be implemented in Matlab with a simple reminder logic as below. Since the *rem(a,b)* function in Matlab gives a reminder of a divided b, a logic, "$if \quad rem(sim_k, i) == 0, prbs = 1 * idinput(1);$", updates a new PRBS perturbation in every *i* steps of the simulation iterations.

Figure 5.4 is obtained by the application of the PRBS wave perturbation in the input. Differently from the sine wave perturbation in Figure 3.1, the ARX estimated gradient successfully follows the true gradient.

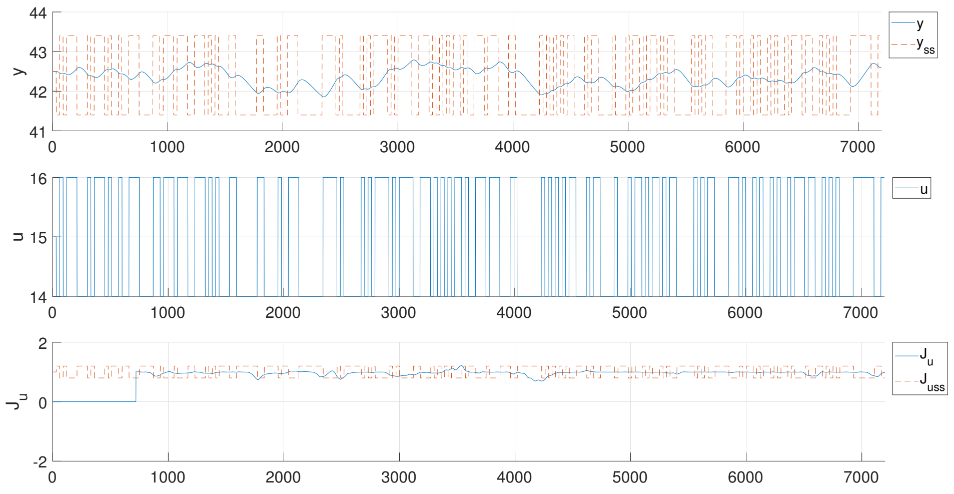

Figure 3.3: Simulation result for a single gas-liftd oil well model, second-order without zero system using a dynamic ESC with PRBS perturbation: no control

This phenomenon results from the fact that the PRBS wave has a diverse range of frequencies compared to a simple sinusoidal wave and this characteristic is more efficient to make the ARX model estimate four parameters in the second-order system. This is the reason behind the fact that the PRBS wave perturbation is suggested in Algorithm 1, and the other case studies use the PRBS wave as well due to its advantage on the ARX estimation.

3.2 Combination of the ARX model with LS method

Another issue on the Figure 3.1 is that there are spikes in the estimated steady-state gradient. The same problem occurred in the PRBS perturbation case as well. The spikes make the system deviate from the optimal so that the estimated gradient converges to a value which is not exactly zero.

After some additional simulations and observations, it became clear that the numerical spikes are prone to firstly happen right after the estimated steady-state reaches zero, and occasionally even after the convergence. Therefore, the ARX model is discussed as a possible reason to explain such phenomena.

In specific, the second-order ARX model is:

$$y(t) + a_1 y(t-1) + a_2 y(t-2) = b_1 u(t-1) + b_2 u(t-2) + e(t) \tag{3.1}$$

$$\text{where} \quad \theta = [a_1 \quad a_2 \quad b_1 \quad b_2]^T$$

The second-order ARX model estimates four parameters stated in the θ vector. When we consider a point near the extremum where J_u is close to zero, there is a change in the input u, but almost no change in the output y which makes only one parameter can be estimated as depicted in Figure 3.4. However, the ARX model still tries to estimate four parameters, and this is the cause of the peaks. They are simply called numerical spikes or numerical peaks.

Therefore, this discussion has an implication that there is a probability of the numerical spikes near zero-gradient when the ARX model estimates more than one parameters, even in a first-order system as well.

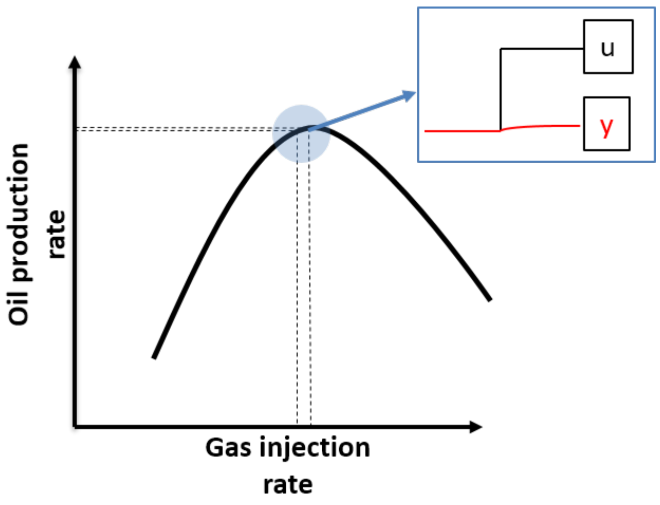

Figure 3.4: The input and output relationship when the gradient is close to zero

Since the reason of the numerical spikes is expected to be the ARX model, a method suggested solving this issue is combining a simple first-order least square(LS) method with the ARX model in gradient estimation.

In specific, the ARX model can be switched with the LS method when J_u is close to zero where numerical peaks may occur. This enables J_u to converge to zero fast enough with the local linear dynamic model by the ARX method, and when it is sufficiently close to zero, the switched LS method gives stability on the gradient estimation avoiding numerical peaks.

In the simulations for each case study, this idea can be applied when serious numerical spikes occur especially making serious deviations from the optimal. It is named as a modified dynamic ESC. The algorithm is stated in Algorithm 2. The threshold to switching the gradient estimation method is defined as ϵ.

Algorithm 2 The modified dynamic ESC combining ARX model with LS method

```
input: $y_k$,$u_k$
 1: for $k = 1 \to n$ do
 2:     Data of the input and output values are built up with a window size $l$
 3:             vector   $u_k \leftarrow [u_{k-l+1}, u_{k-l+2}, ..., u_k]$
 4:             vector   $y_k \leftarrow [y_{k-l+1}, y_{k-l+2}, ..., y_k]$
 5:     Moved to have a zero mean
 6:             $y_0 \leftarrow y_k - avg(y_k)$
 7:             $u_0 \leftarrow u_k - avg(u_k)$
 8:     if $J_{u,k-1} > \epsilon$ then
 9:         Gradient estimation with the ARX model using data sets of $y_0$ and $u_0$
10:                 $data \leftarrow iddata(y_0, u_0, T_s)$
11:                 $[A(q) \quad B(q)] \leftarrow arx[data, [1, 1, 0]]$   for 1st-order systems
12:                 $[A(q) \quad B(q)] \leftarrow arx[data, [2, 2, 0]]$   for 2nd-order systems
13:                 $[A_d \quad B_d \quad C_d \quad D_d] \leftarrow idss[A(q) \quad B(q)]$
14:                 $[A \quad B \quad C \quad D] \leftarrow d2c[A_d \quad B_d \quad C_d \quad D_d]$
15:                 $J_{u,k} \leftarrow -CA^{-1}B + D$
16:     else
17:         Gradient estimation with the LS method using data sets of $y_k$ and $u_k$
18:                 $X \leftarrow [u_k \quad ones(size(u_k))]$
19:                 $b \leftarrow (X'X)^{-1}X'y_k$
20:                 $J_{u,k} \leftarrow b(1, 1)$
21:     end if
22:     Integral Controller
23:             $\hat{u}_{k+1} \leftarrow u_k + K_i J_{u,k}$
24:     Perturbation
25:             $prbs \leftarrow idinput(1)$
26:             $u_{k+1} \leftarrow \hat{u}_{k+1} + prbs$
27: end for
28: $*$   $\epsilon$ is the threshold defined by the user
```

Chapter 4

Case study 1 : Parallel heat exchangers

4.1 Process description

With the successive development of the dynamic ESC in chapter 3, this first case study introduces a general optimization problem as a plant model which is two parallel heat exchangers with one split suggested and studied in Jäschke and Skogestad (2014).

In detail, the system has one inlet flow and it is divided into two streams with a split ratio α. Each stream is heated by heat exchangers respectively, and they are merged together yielding the outlet temperature, which is illustrated in 4.1.

In this process, the manipulated variable is the split ratio and the control objective is maximizing the outlet temperature. Since this system has an obvious maximum point, it is a great candidate for applying ESC.

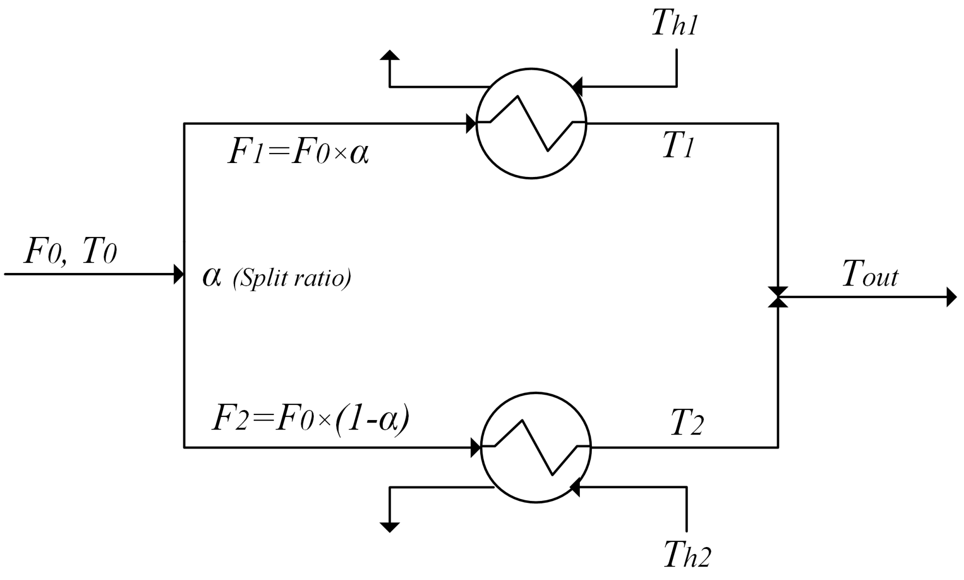

Figure 4.1: A schematic process diagram of a parallel two heat exchangers network with one split

The heat transfer rate is defined with heat transfer coefficient(U), surface area(A) and temperature difference in a heat exchanger. With an assumption that the energy is entirely transferred into the single-phase fluid without heat loss, the energy balance can be written as (4.1) and (4.2) for each heat exchanger.

$$Q_1 = U \cdot A_1 \cdot (T_{h1} - \frac{T_1 + T_0}{2}) = F_1 \cdot cp_1 \cdot (T_1 - T_0) \tag{4.1}$$

$$Q_2 = U \cdot A_2 \cdot (T_{h2} - \frac{T_2 + T_0}{2}) = F_2 \cdot cp_2 \cdot (T_2 - T_0) \tag{4.2}$$

Based on the fact that usually cold water is supplied and heated, T_0 can be set as zero. Additionally, F_1 and F_2 can be expressed as $F_0 \cdot \alpha$ and $F_0 \cdot (1 - \alpha)$ respecitvely as shown in Figure 4.1. Thus, (4.1) and (4.2) can be formulated as below.

$$U \cdot A_1 \cdot (T_{h1} - \frac{T_1}{2}) = F_0 \cdot \alpha \cdot cp_1 \cdot T_1 \tag{4.3}$$

$$U \cdot A_2 \cdot (T_{h2} - \frac{T_2}{2}) = F_0 \cdot (1-\alpha) \cdot cp_2 \cdot T_2 \tag{4.4}$$

By rearrangement, T_1 and T_2 are written as (4.5) and (4.6).

$$T_1 = \frac{T_{h1}}{\alpha \cdot k_1 + 0.5} \tag{4.5}$$

$$T_2 = \frac{T_{h2}}{(1-\alpha) \cdot k_2 + 0.5} \tag{4.6}$$

Where

$$\begin{aligned} k_1 &= \frac{F_0 \cdot cp_1}{U \cdot A_1} \\ k_2 &= \frac{F_0 \cdot cp_2}{U \cdot A_2} \end{aligned}$$

Based on these temperatures, the outlet temperature T_{out} can be obtained by (4.7)

$$\begin{aligned} T_{out} &= \frac{\alpha \cdot F_0 \cdot cp1 \cdot T_1 + (1-\alpha) \cdot F_0 \cdot cp_2 \cdot T_2}{\alpha \cdot F_0 \cdot cp1 + (1-\alpha) \cdot F_0 \cdot cp_2} \\ &= \frac{\alpha \cdot cp1 \cdot T_1 + (1-\alpha) \cdot cp_2 \cdot T_2}{\alpha \cdot cp1 + (1-\alpha) \cdot cp_2} \end{aligned} \tag{4.7}$$

Data for the parallel heat exchangers are provided in 4.1. T_0 and F_0 values are chosen, and all other values related to the heat exchangers refer to Jäschke and Skogestad (2014).

Variable	Value	Unit	Description
T_0	0	$^\circ C$	Cold feed temperature
F_0	12	kg/s	Total flow rate
cp_1	30	kW/K	Hot stream 1 heat capacity
cp_2	50	kW/K	Hot stream 2 heat capacity
T_{h1}	120	$^\circ C$	Hot stream 1 temperature
T_{h2}	220	$^\circ C$	Hot stream 2 temperature
$U \cdot A_1$	50	$^\circ C$	Heat transfer coefficient· Heat exchanger 1 area
$U \cdot A_2$	80	$^\circ C$	Heat transfer coefficient· Heat exchanger 2 area

Table 4.1: Data for Case study 1

The dynamics of the two exchangers and mixing point are first order and the time constant is set to be $600s$ for all of them. All tuning parameters are suggested in Appendix 8.1.

The initial values of the variables in simulations are:

$$u = 0.9, \qquad [T_1, T_2, T_{out}] = [180.97, 818.83, 244.81]$$

4.2 Dynamic ESC using ARX model

Figure 4.2 represents the simulation result of the dynamic ESC on the two parallel heat exchagners model. To see the gradual changes of the temperatures in each stream (T_1 and T_2) and the outlet temperature(T_{out}) at the same time, they are all illustrated in the first subplot with a yellow, purple, and blue solid line respectively. The second subplot includes information of the input value which is the split ratio α, and the last plot contains information of the estimated gradient and steady-state gradient with a blue solid and orange dashed line.

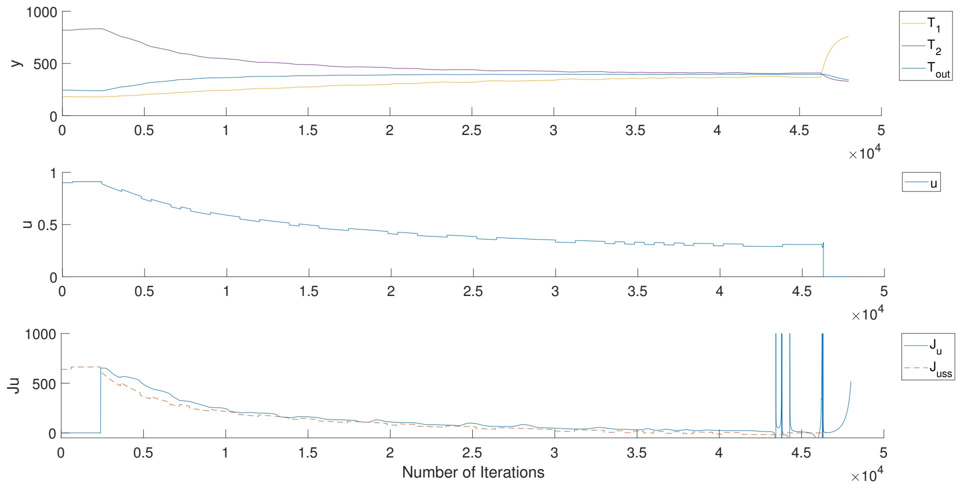

Figure 4.2: Simulation result for parallel heat exchangers with one split model using a dynamic ESC

The result shows that the system is converging to the optimum at the first time, however, sudden numerical spikes happen in the gradient estimation and the system starts to deviate after $4.5 \cdot 10^4$ iterations. As stated in the section 3.2, the idea to combine the ARX model with the LS method is applied to remove the spikes and make the system converge to the right extremum.

4.3 Modified Dynamic ESC combining ARX model with LS method

Figure 4.3 is obtained by applying the modified dynamic ESC mechanism. In detail, the ARX method estimates the steady state gradient when the previously estimated gradient is larger than 50, and the LS method is used when it is same or smaller than 50.

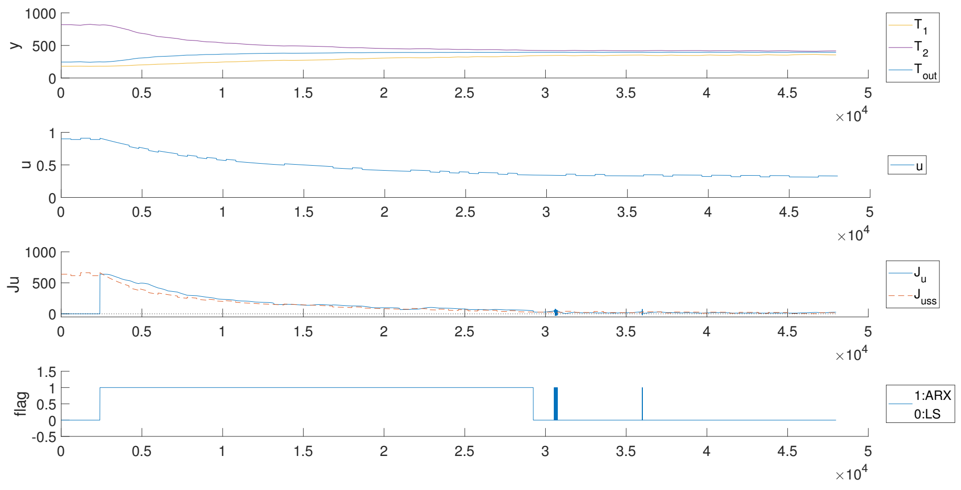

Figure 4.3: Simulation result for parallel heat exchangers with one split model using a modified dynamic ESC with a threshold of 50

Overall, the estimated gradient converges to zero without numerical spikes. It is clearly shown that most of the system is estimated by the ARX method, and it achieves a faster convergence as intended. After the gradient becomes close to zero, the LS method makes the convergence more stable not affecting the speed of the convergence.

4.4 Comparison

In this section, the result obtained in the Figure 4.3 will be compared with the result simulated by the classic ESC. The main objective of the comparison is to see the speed of the convergence. Both cases are tuned as best as possible, please see the Appendix for the information of tuning parameters.

Figure 4.4 is comparing the results obtained by the dynamic ESC and classic ESC with blue and green solid line respectively. In the first subplot, only the outlet temperature is indicated to show the result in a clear way.

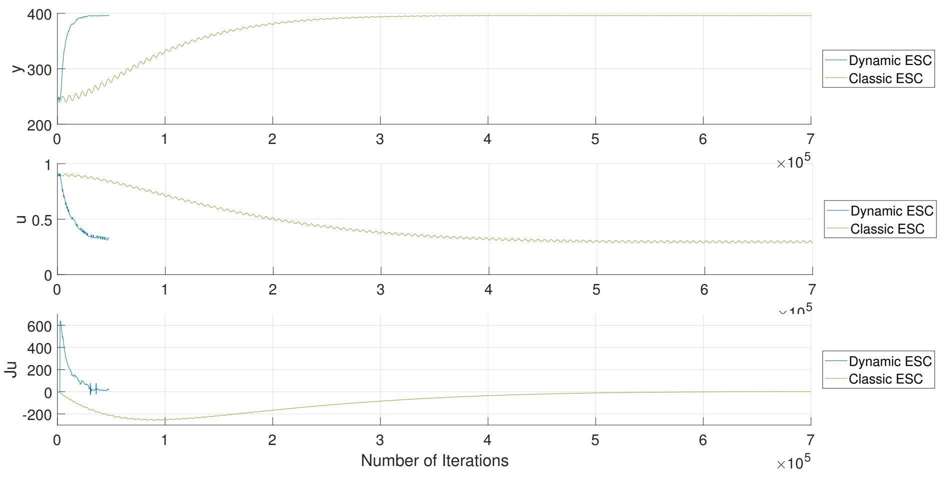

Figure 4.4: Simulation result comparison of a dynamic and classic ESC for two parallel heat exchangers with one split model

In specific, the system converges to the optimal around $5 \cdot 10^5$ iterations with the classic ESC while the dynamic ESC requires only 10% of the iterations. The system moves to the extremum very gradually and slowly in the case of the classic ESC. This fact proves the usefulness of the dynamic ESC.

4.5 Application of disturbance

Figure 4.5 shows a simulation result when a disturbance is applied. Since the main role of a controller is counter-reacting to a disturbance and placing a process variable in a set-point, this section will illustrate a performance of dynamic ESC in regards to the disturbance.

In detail, the disturbance is applied in the inlet temperature of stream 1, T_{h1}. Thus, the equation (4.5) becomes (4.8) as below. The disturbance is set to be $40°$C.

$$T_1 = \frac{T_{h1} + d}{\alpha \cdot k_1 + 0.5} \tag{4.8}$$

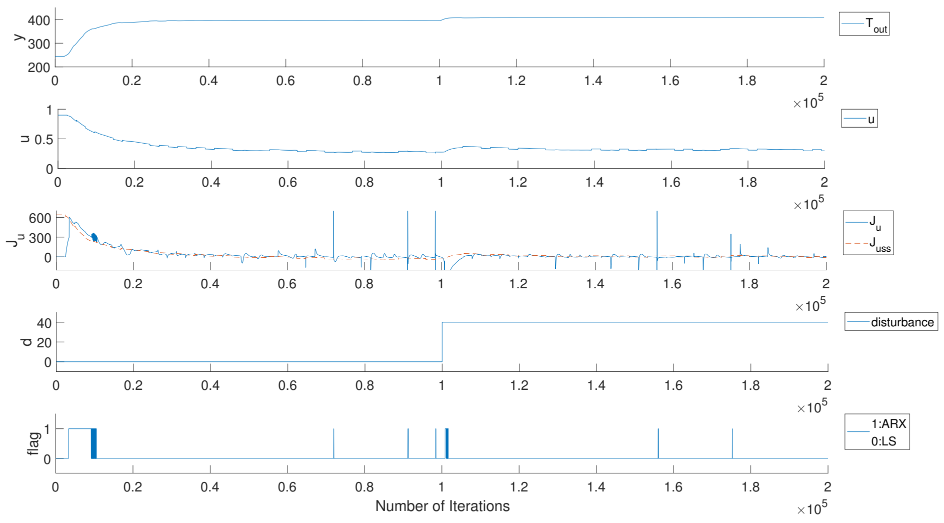

Figure 4.5: Simulation result for disturbed parallel heat exchangers with one split model using a modified dynamic ESC with a threshold of 50

The result indicates that the input and output values are successfully converging to a new extremum where the gradient becomes zero after the disturbance happens.

Chapter 5

Case study 2 : Single gas-lifted oil well

5.1 Process description

As shown in Figure 1.2, the relationship between the oil production rate and the gas injection rate can be simplified as a polynomial equation. In Hammerstein and Weiner model, this relationship represents a static non-linear part of the plant in Figure 2.3, $y_{ss} = f(u)$. The use of such simplified Hammerstein models for gas-lifted well is justified in Peixoto et al. (2015) and Plucenio et al. (2009). Empirical models such as the one used in this case study are also used in practice Hamedi et al. (2011). Therefore, Hammerstein and Weiner's model is assumed and the gas-lift model is simplified as a quadratic equation in this second case study.

Specifically, Equation (5.1) can be used as a simple static non-linear quadratic system with unknown parameters a, b, and c, which has an extremum at $(u, y_{ss}) = (a, b)$ when the c is negative.

$$f(u) = c(u-a)^2 + b = y_{ss} \tag{5.1}$$

The constants in the Equation (5.1) are dependent not only on different wells but also on operation situations and disturbances. In this project, the $f(u)$ is not describing a real system, and the constants are manually selected to have a clear maximum and small values for the input and output. This is because when the values for u or y is large, it requires too much computational time to converge. Using a simple model will provide a clear and fast result and it will also help to develop the dynamic ESC model faster. Thus, the a, b, and c are set to be 20, 45, and -0.1 respectively as Equation (5.2). Tuning parameters are suggested in Appendix.

$$f(u) = -0.1(u-20)^2 + 45 = y_{ss} \tag{5.2}$$

In this chapter, monitoring if the dynamic ESC algorithm is working well is the first focus. For this, plant processes with different dynamics are tested, especially a first-order transfer function, a second-order transfer function without zero, and a second-order transfer function with zero are applied to add more complexity as Table 5.1. Each system with a different transfer function is simulated one by one in section 5.2, 5.3, and 5.4 respectively.

The initial values of the variables are the same in all of the cases as:

$$u = 15, \qquad y = 42.5$$

first-order transfer function	**second-order transfer function without zero**	**second-order transfer function with zero**
$G(s) = \frac{1}{\tau s+1}$ $= \frac{1}{174s+1}$	$G(s) = \frac{\tau_a s+1}{(\tau_1 s+1)(\tau_2 s+1)}$ $= \frac{1}{(174s+1)(60s+1)}$	$G(s) = \frac{\tau_a s+1}{(\tau_1 s+1)(\tau_2 s+1)}$ $= \frac{40s+1}{(174s+1)(60s+1)}$

Table 5.1: Transfer functions for Case study 2

5.2 Dynamic ESC using ARX model for first-order system

Figure 5.1 is illustrating the application of the dynamic ESC strategy on the first-order system. The input, output, and estimated gradient are drawn with blue solid lines, and orange dashed lines indicate steady-state values which is calculated from the plant model.

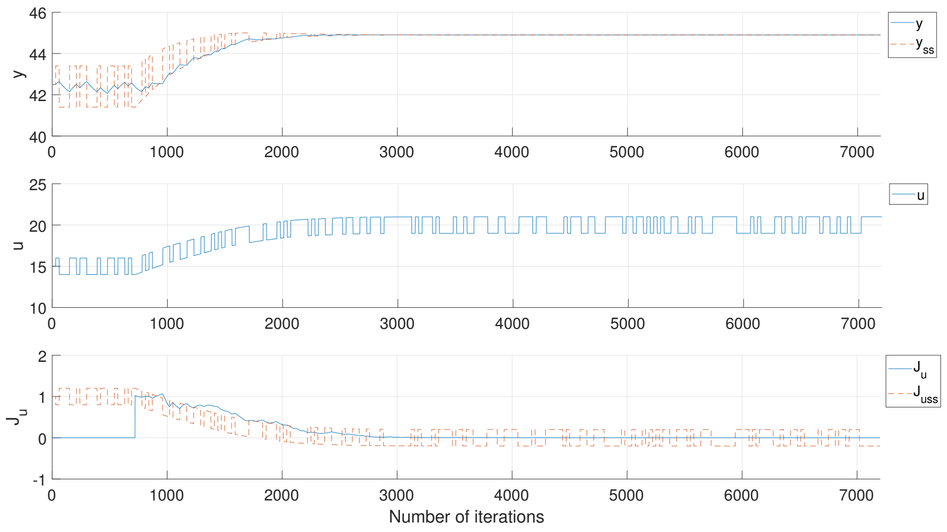

Figure 5.1: Simulation result for a single gas-liftd oil well model, first-order system using a dynamic ESC

In the figure, it is clearly indicated that the estimated gradient J_u follows the true gradient, verifying the fact that the dynamic ESC using ARX model is successfully implemented.

The system starts the gradient estimation after building some data sets, however, it can be ignored to see the convergence speed since a process continuously works in a real plant system.

5.2.1 Comparison

Figure 5.2 is comparing simulation results for the first-order system obtained by the dynamic and classic ESC. The blue solid lines illustrate the simulation with dynamic ESC model. For the classic ESC, the values are stated with a green line.

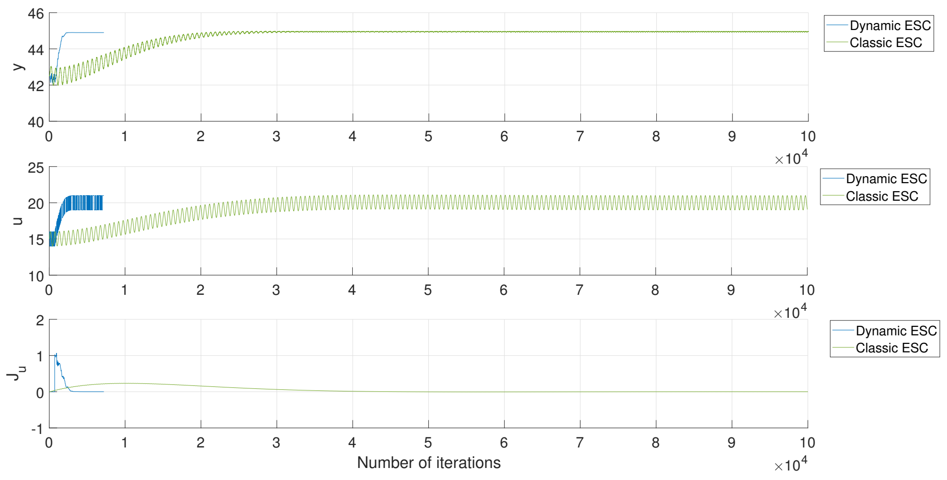

Figure 5.2: Simulation result comparison of a dynamic and classic ESC for a single gas-liftd oil well with first-order system

In specific, the classic ESC converges to the extremum around $5 \cdot 10^4$ iterations while the dynamic ESC only requires 3100 iterations. As already mentioned that there is no need to wait to build data since a process is continuous and always build data, the window size(720) can be subtracted from iterations to converge in the case of the dynamic ESC, then it only takes 2380 iterations to the new extremum.

This result is extremely meaningful because a faster convergence is the main motivation to start this work. Numerically, only around 4% of the transient iteration steps are needed for convergence with the application of the dynamic ESC compared to the classic ESC.

5.2.2 Application of disturbance

Figure 5.3 depicts the simulation result of a disturbed first-order single gas-liftd oil well. The disturbance is applied in the static non-linear part and the equation (5.2) becomes (5.3) as below, and the disturbance is set to be 2. The Modified dynamic ESC strategy is used to avoid serious numerical spikes.

$$f(u,d) = -0.1(u-20)^2 + 45 + d \tag{5.3}$$

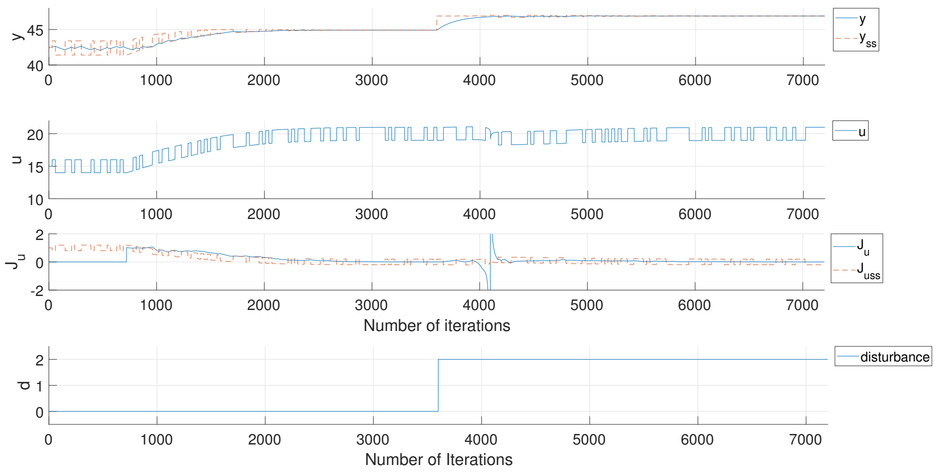

Figure 5.3: Simulation result for a disturbed single gas-liftd oil well model, first-order system using a dynamic ESC

When the disturbance occurs, the system converges to the new extremum where $(u,y) = (20,47)$ and the gradient equals to zero.

5.3 Dynamic ESC using ARX model for second-order system without zero

Figure 5.4 is obtained by the application of the dynamic ESC with PRBS wave perturbation. As the previous first-order system, the estimated gradient follows the true gradient as well. However, numerical spikes happen in the gradient estimation and they make a deviation from the true extremum as expected in section 3.2. In specific, the optimal value of the input equals to 20 and the spikes firstly appear after 2000 iterations when the updated input approaches to the optimal.

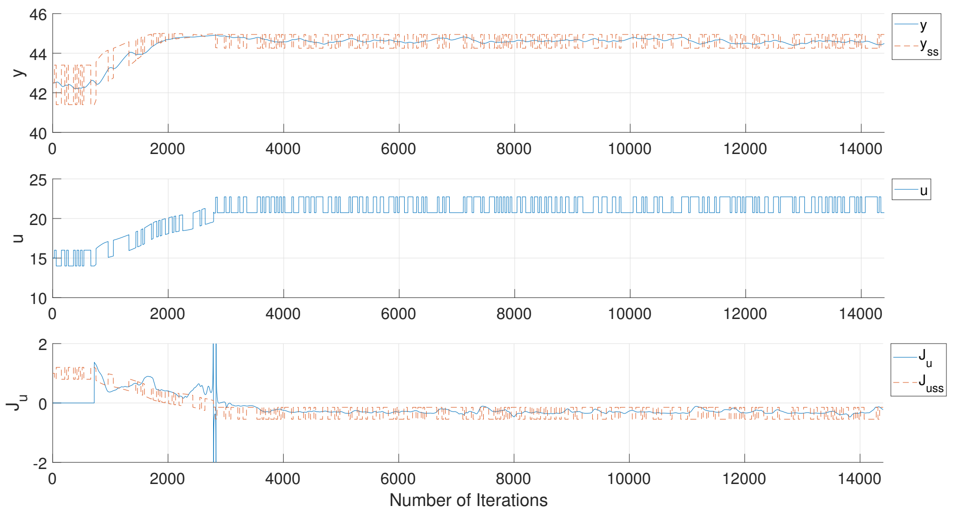

Figure 5.4: Simulation result for a single gas-liftd oil well, second-order without zero system using a dynamic ESC

As discussed in the 3.2, the strategy to combine the ARX model with the LS method for the steady-state gradient estimation can be applied to remove the numerical spikes.

5.3.1 Modified Dynamic ESC combining ARX model with LS method

Since the reason of the numerical spikes is expected to be the ARX model, a method suggested solving this issue is combining a simple first-order least square(LS) method with the ARX model in gradient estimation as section 3.2. Based on this idea, simulations with different thresholds of the switching point are performed, and the best result is illustrated in Figure 5.5. A flag is stating which method of them is used to estimate the gradient, 1 for the ARX model and 0 for the LS method.

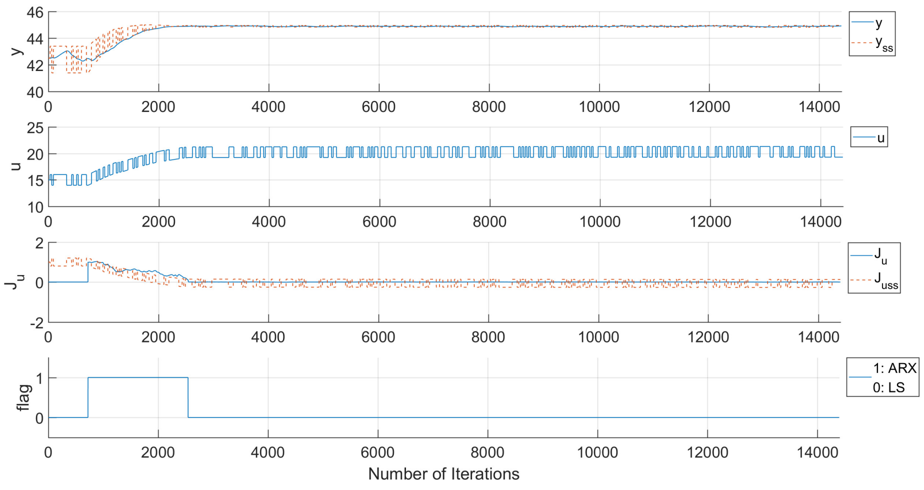

Figure 5.5: Simulation result for a single gas-liftd oil well, second-order without zero system using a modified dynamic ESC with a threshold of 0.1

Figure 5.5 is obtained with a tolerance of 0.1. In detail, the ARX model is used when the mean value of the estimated gradient in the previous 10 steps is larger than 0. and the Least square method is used when that is smaller than 0.30.

By applying the modified ESC, the numerical spikes are cleared out and the estimated steady-state gradient successfully converges to zero.

5.3.2 Comparison

The result obtained by a combination of the ARX model with LS method is compared with the classic ESC in Figure 5.6. The tendency is similar to the first-order system that the classic ESC requires $5 \cdot 10^4$ iterations while the dynamic ESC needs only 2500 iterations, or when the window size is subtracted, 1780 transient iterations. This result means that only 3.6% of the iterations for the classic ESC are needed in the dynamic ESC for a transition to a new optimum.

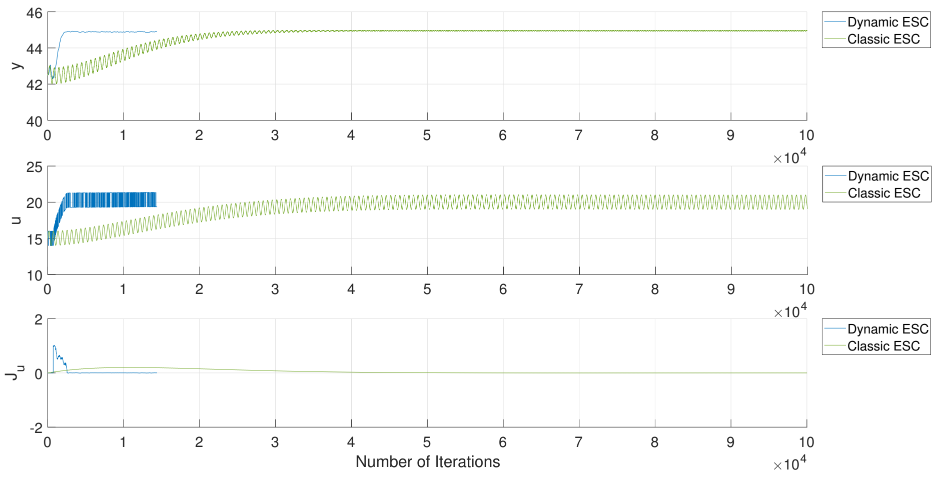

Figure 5.6: Simulation result comparison of a dynamic and classic ESC for a single gas-liftd oil well model with second-order without zero system

Additionally, the percentage 3.6% is similar to that in the first-order system which is 4%. With regards to the ARX model is combined with the LS method, this combination is proven not to affect the overall number of iterations to converge. It is because most of the important part is estimated by the ARX model and the LS method acts around the zero gradient where the ARX model becomes problematic. It can be concluded that the combination provides a faster and more robust result.

5.3.3 Application of disturbance

As same as the first-order system, a disturbance is applied in the static non-linear part, and it is set to be 10. The result is stated in Figure 5.7. In the simulation, the PRBS wave is updated less frequently since the estimated gradient often could not follow the steady-state gradient when only a small disturbance is applied.

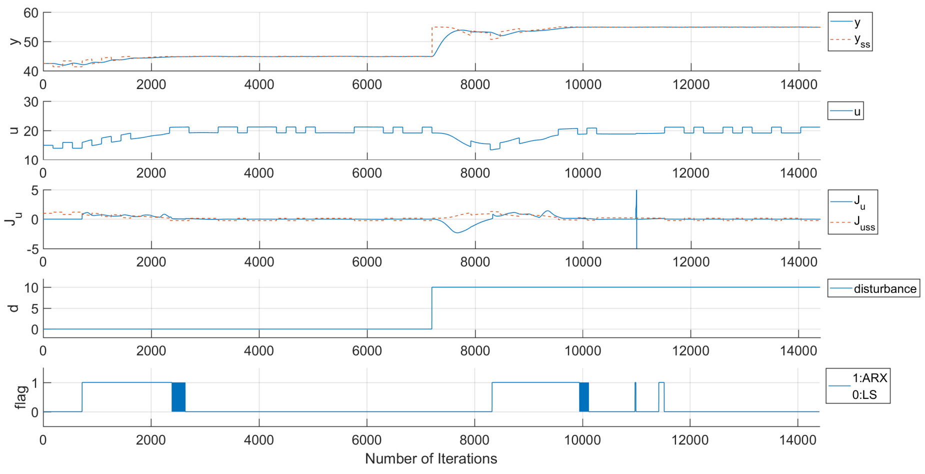

Figure 5.7: Simulation result for a disturbed single gas-liftd oil well, second-order without zero system using a modified dynamic ESC with a threshold of 0.1

As shown in the Figure, the system goes to the next optimum $(u, y) = (20, 55)$ fast, and the estimated gradient converges to zero as well after being disturbed.

5.4 Dynamic ESC using ARX model for second-order system with zero

Similar steps as the previous study are performed for this second-order with zero system. The main difference of this process from the previous system is the existence of zeros in the transfer function, and the idea is to see if the dynamic ESC works in the system with more complexity. Figure 5.8 illustrates the simulation result using the second-order ARX model.

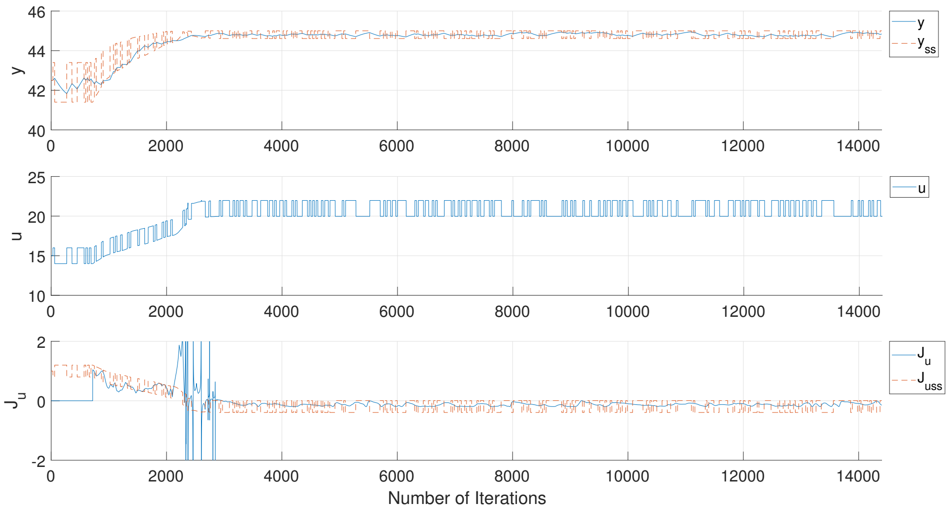

Figure 5.8: Simulation result for a single gas-liftd oil well, second-order with zero system using a dynamic ESC

In detail, the dynamic ESC works very well and the gradient is successfully estimated as zero. Compared with the result obtained for the second-order without zero system, the numerical spikes are not serious and they do not make deviations from the extremum. However, to guarantee safety and robustness, the combination of the ARX model with the least square method is tried as well in section 5.4.1.

5.4.1 Modified Dynamic ESC combining ARX model with LS method

Figure 5.9 is depicting the simulation result when the ARX model is combined with the LS method. A tolerance is 0.001 which means that the ARX model is used when the mean gradient in the last steps is larger than the tolerance. Since the spikes were not serious and not frequent at the same time, it was possible to remove the spikes only with a small tolerance.

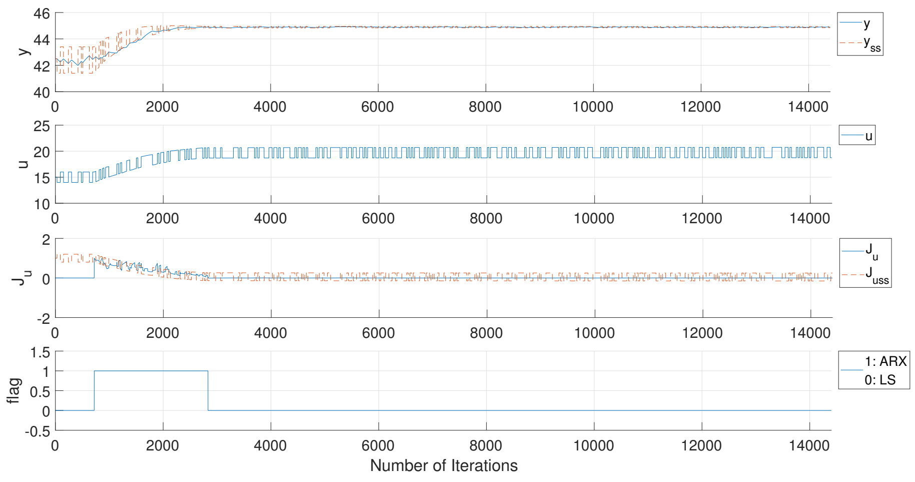

Figure 5.9: Simulation result for a single gas-liftd oil well, second-order with zero system using a modified dynamic ESC with a threshold of 0.05

As the previous case, the numerical peaks are removed out. The estimated gradient smoothly converges to zero.

5.4.2 Comparison

The result of the modified dynamic ESC is compared with the classic ESC as in Figure 5.10. The classic ESC takes around $5 \cdot 10^4$ iterations to converge while the dynamic ESC requires 3500 iterations or 2780 transient iterations which is 5.6% of the classic ESC.

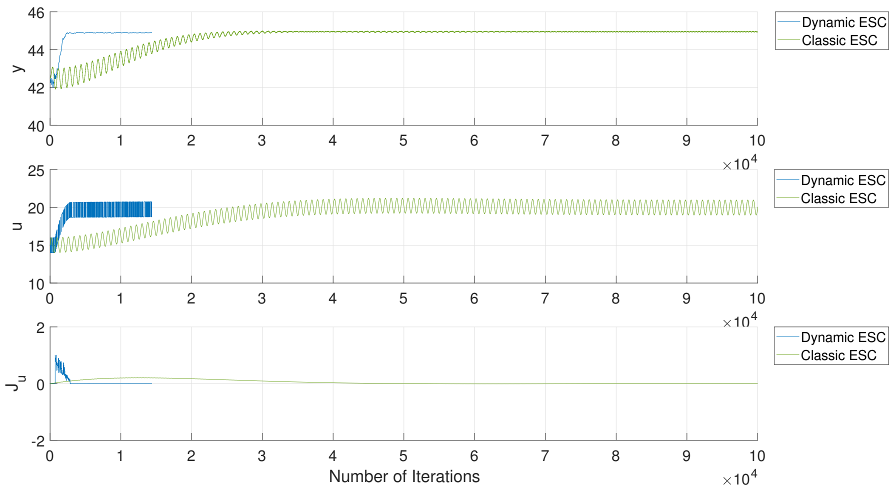

Figure 5.10: Simulation result comparison of a dynamic and classic ESC for a single gas-liftd oil well with second-order with zero system

Overall, the classic ESC requires a significant number of iterations compared to the dynamic ESC, verifying the superior ability of the dynamic ESC developed in this project. The idea that estimating the dynamic system by the local linear dynamic model is resulted to be more efficient than the local linear static model.

It is proven that the dynamic ESC is successfully working for both the first and second-order systems, and the suggestion to combine the ARX model with the LS method is worthwhile.

5.4.3 Application of disturbance

A disturbance is applied as same as the previous case which is illustrated in Figure 5.11. The Modified dynamic ESC strategy is used to avoid serious numerical spikes, and this case converges to the next optimum as well.

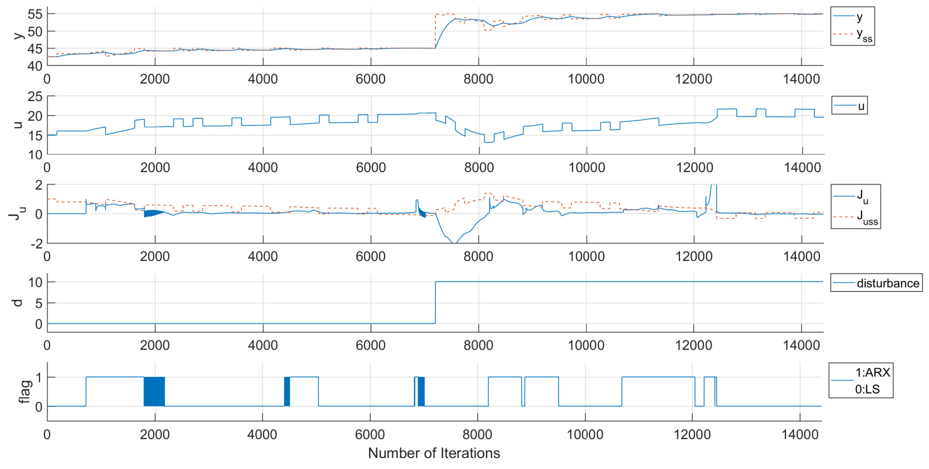

Figure 5.11: Simulation result for a disturbed single gas-liftd oil well, second-order with zero system using a modified dynamic ESC with a threshold of 0.1

Chapter 6

Case study 3 : Multiple gas lifted oil wells

6.1 Process description

In the offshore gas and oil field, the number of oil wells is decided based on the limitations of the drill rig and trade-offs as suggested in Gupta and Grossmann (2012). Thus, although only one gas-lifted oil well is considered in the previous case studies, it is a natural step to consider multiple gas-lifted oil wells altogether. Figure 6.1 is describing an oil field with 6 gas-lifted oil wells where the produced oil from each gas-produced oils from each well are assumed to be possible to measure.

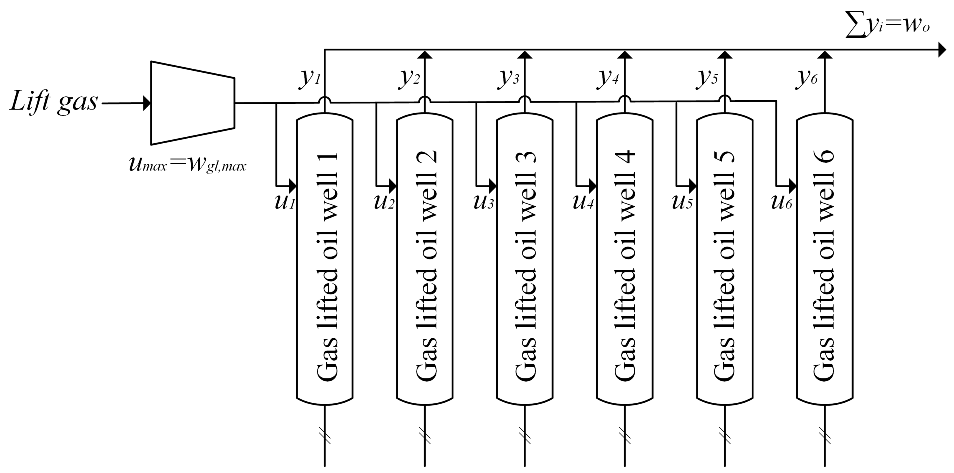

Figure 6.1: A schematic process diagram of multiple gas lifted oil wells

In this process, lift gas is compressed first and pumped into the annulus of gas lifted oil wells. More details of the gas injection process inside the wells are illustrated in the Figure 1.1 where the lift gas is injected by an injection valve in the deep position in the oil wells.

The greatest important difference from the Case study 1 and 2 is that this process not only considers multiple inputs but also constraints. The constraints stem from the limitations of a compressor or the amount of lift gas. This fact makes a maximum possible amount of compressed lift gas $w_{gl,max} = u_{max}$ so that the summation of the inputs $\sum u_i$ cannot exceed the maximum lift gas. Since the constraint should be tightly controlled to obtain maximum profit, $u_{max} = \sum u_i$. The amount of produced oil is notated as w_o which is a summation of the oil from each oil wells $\sum y_i$.

As the second case study, Hammerstein and Wiener's model is used to describe each gas-lifted oil wells. All of the static nonlinear parts is depicted by quadratic equations, and they have similar dynamics as stated in Table 6.1.

	Static non-linearity, $f(u_i)$	**Linear Dynamic,** $G(s)$
Gas lifted oil well 1	$f(u_1) = -0.1(u_1 - 20)^2 + 45 = -0.1{u_1}^2 + 4u_1 + 5$	$\frac{1}{174s+1}$
Gas lifted oil well 2	$f(u_2) = -0.5(u_2 - 10)^2 + 55 = -0.5{u_2}^2 + 10u_2 + 5$	$\frac{1}{180s+1}$
Gas lifted oil well 3	$f(u_3) = -0.2(u_3 - 15)^2 + 55 = -0.2{u_3}^2 + 6u_3 + 10$	$\frac{1}{170s+1}$
Gas lifted oil well 4	$f(u_4) = -0.05(u_4 - 30)^2 + 55 = -0.05{u_4}^2 + 3u_4 + 10$	$\frac{1}{176s+1}$
Gas lifted oil well 5	$f(u_5) = -0.04(u_5 - 25)^2 + 40 = -0.04{u_5}^2 + 2u_5 + 15$	$\frac{1}{180s+1}$
Gas lifted oil well 6	$f(u_6) = -0.1(u_6 - 10)^2 + 30 = -0.1{u_6}^2 + 2u_6 + 20$	$\frac{1}{177s+1}$

Table 6.1: Description of 6 gas lifted oil wells for Case study 3

The initial values of the variables are:

$$(u_1, y_1) = (3, 16.1) \qquad (u_2, y_2) = (14, 47) \qquad (u_3, y_3) = (4, 30.8)$$

$$(u_4, y_4) = (24, 53.2) \qquad (u_5, y_5) = (1, 16.96) \qquad (u_6, y_6) = (10, 30)$$

Each gas-lifted oil wells have a different extremum. The maximum amount of lift gas is set to be $u_{max} = 56$, consequently, each oil wells cannot be on their extremum and cannot be controlled by simply setting the gradient at zero as well. This leads to the need for a new control strategy in chapter 6.2

6.2 Control strategy

In plantwide control design, it is extremely important to select right control structure including the decision of controlled variable as suggested by Downs and Skogestad (2011). In the study, they provided and proved a control strategy for parallel units which corresponds to the plant model in this case study.

In specific, when the cost of each unit i equals to L_i and the feed rates of each unit i equals to u_i, the total cost and feed rate can be written as $L = \sum L_i$ and $u = \sum u_i$ respectively. The total feed rate u is assumed to be a constant. The optimal condition is minimizing the total cost L as $\frac{\delta L}{\delta u} = 0$ where u is the vector of feed rates of each units i, u_i. Another assumption is that each unit is independent. Therefore, the cost of each unit L_i depends only on the feed rate of each unit u_i. Additionally, when the total number of units is n, the degrees of freedom is $n-1$ so that the feed rate of the unit n is described as $u_n = u - \sum_{i=1}^{n-1} u_i$ and consequently $du_n = -du_i$. Then the optimal condition can be written as equation (6.1).

$$\frac{\delta L}{\delta u_i} = \frac{\delta(L_1 + L_2 + ... + L_i + L_n)}{\delta u_i} = \frac{\delta L_i}{\delta u_i} + \frac{\delta L_n}{\delta u_i} = \frac{\delta L_i}{\delta u_i} - \frac{\delta L_n}{\delta u_n} = 0 \tag{6.1}$$

or simply

$$\frac{\delta L_i}{\delta u_i} = \frac{\delta L_n}{\delta u_n} \tag{6.2}$$

Equation (6.2) means that the gradients of each unit are the same value. Thus, in the process with 6 gas-lifted oil wells, the controlled variables are $J_{u_1} = J_{u_2} = J_{u_3} = J_{u_4} = J_{u_5} = J_{u_6}$. Tuning parameters are suggested in Appendix.

6.3 Dynamic ESC using ARX model

The simulation result of the 6 gas-lifted oil wells using the dynamic ESC is stated in Figure 6.2. Based on the control strategy, the integral controllers have a setpoint where all of the estimated gradients becomes the same. The controller gain is set to be the same in each oil well, please see the Appendix for more information on tuning parameters. In terms of the PRBS wave perturbation, the gas-lifted oil well 1&2, 3&4, and 5&6 are assumed to affect each other. Thus, $PRBS_1 = -PRBS_2$, $PRBS_3 = -PRBS_4$, and $PRBS_5 = -PRBS_6$.

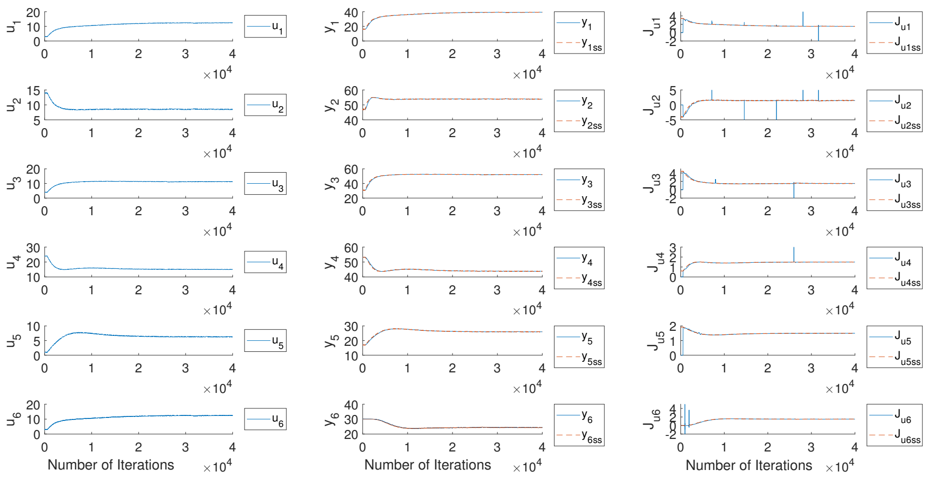

Figure 6.2: Simulation result for multiple gas lifted oil wells using a dynamic ESC

In the result, all of the systems converge to a point where the estimated steady-state gradient becomes 1.5. This fact means that the control strategy is implemented well with the dynamic. Even though there are some spikes in the gradient, they are not serious and can be ignored since the estimated gradient is not converging to zero.

The result comparison with the classic ESC is not conducted in this case study. The reason is that as pointed out in the section 2.1, the parameter tuning in the classic is difficult and expected to be time-consuming. In detail, in this kind of multiple parallel systems, the high pass filter frequency and the low pass filter frequency should be decided in each oil well. Since only the system is very sensitive to its tuning, one wrong-decided tuning parameter has a probability to make the whole system chaotic and a large number of simulations will be required.

Chapter 7

Discussion

With the motivation of figuring out a dynamic ESC algorithm which is expected to achieve a faster convergence to a new extremum, the ARX model is introduced to estimate the steady-state gradient which is a local linear dynamic model.

In the design of the dynamic ESC, the second case study provided some ideas. Specifically, a simple single input-output system is described by Hammerstein and Wiener models and three different transfer functions are tested. The developed dynamic ESC algorithm works perfectly in the first-order system that the estimated gradient follows the true gradient with a setpoint of zero. However, in the second-order without zero system, a problem is caused that the estimated steady-state gradient could not follow the true gradient.

The approach to explain this behavior comes from the step responses of different transfer functions since there was no such a problem in the first-order system. In detail, the step response of the second-order system has a secondary time lag, and the sinusoidal wave is not enough to obtain the full information of the output. With this approach, the PRBS wave is introduced instead of a simple sinusoidal perturbation to provide multiple frequencies, and it could solve the problem. Due to a better performance of the ARX model for the gradient estimation, the PRBS perturbation is used in all of the case studies.

The second problem is detected in the second order system as well. In detail, numerical spikes happen and sometimes they make deviations from the extremum. The ARX model accounts for this phenomenon with the observation that the spikes are prone to firstly happen right after the estimated steady-state gradient becomes close to zero. In specific, when there is a change in the input near the extremum, the output almost does not change making the estimation of only one parameter possible, while the ARX model still tries to estimate two parameters in the first-order system or four parameters in the second-order system. This is the main drawback of the ARX model that the estimated gradient near zero is not reliable and sometimes it makes serious problems.

To fix this limitation, the combination of the ARX model with the LS method is introduced with the idea that the ARX model estimates gradient faster in the early stages, and it is switched to the LS method when the estimated gradient is close enough to zero. Although the LS method is a kind of a local linear static model, it could not affect the iteration numbers to converge since most of the important part is estimated by the ARX model. This implementation is concluded to work efficiently making the dynamic ESC more robust.

With the progress in the development of the dynamic ESC algorithm, there is an attempt to apply the dynamic ESC in a realistic plant system in the first case study with two parallel heat exchangers with a split. Since the control objective is maximizing the outlet temperature depending on the split ration, this process could be a perfect candidate for the ESC.

In the second case study, a gas-lifted oil well is described with the Hammerstein and Wiener's model and three different transfer functions are applied.

In the first and second case studies, the dynamic ESC is applied first and the modified dynamic ESC is introduced When a deviation happens by numerical spikes. The results are compared with the results obtained with the classic ESC, and it is revealed that the dynamic ESC requires only less than 10% of the number of iterations compared to the classic ESC.

Finally, the third case study applied multiple inputs system which is multiple gas-lifted oil wells. There is the constraint in the amount of the lift gas and it is not possible to reach the true extremum. Therefore, a new control strategy is introduced which is making the gradient of each oil well the same.

The dynamic ESC could find the optimal point with the ARX model as well in the multiple inputs system where all the gradients become 1.5. Since the setpoint of the gradient is not zero, the numerical spikes problem is not serious in this case. Additionally, each oil wells have different dynamics and gains for the changes in the inputs, so that applying different controller gains could improve the convergence.

In all case studies, it is clear the dynamic ESC achieve a much faster convergence to a new optimum. The instabilities in the ARX estimation near zero-gradient could be successfully diminished by combining the ARX model with the LS method.

Chapter 8

Conclusion

In this project, the slow control performance of the classic ESC provided the biggest motivation to develop the dynamic ESC. The main idea of this algorithm is introducing a local linear dynamic model for the gradient estimation instead of a local linear static model. One of the simple time-variant models is the ARX model, and it is implemented in this project.

Overall, it is concluded that the dynamic ESC is successfully implemented in the three case studies. The estimated steady-state gradient follows the true gradient, and the system converged into the true extremum. Based on this study, several future works can be suggested.

In the comparison with the simulation results of the classic ESC, it was shown that the convergence time was hugely reduced in the dynamic ESC, especially only $3.6 - 10\%$ of convergence time were needed for the dynamic ESC. It verifies that the dynamic ESC is more efficient and faster control strategy than the classic ESC. This is very valuable result for processs control applications, where the system dynamics in general are very slow.

The first future work focuses on the ARX model. As explained in the section 2.2.1, the polynomial parameter θ is estimated by a least square method. Because the estimated steady-state gradient becomes closer to its setpoint even in one window, implementing weighted least square method inside the ARX algorithm is expected to be efficient on the gradient estimation. In this case, the arx function in Matlab cannot be used and a better understanding of the ARX model itself will be necessary.

Secondly, the model predictive controller can be applied replacing a simple integral controller when the process is a multi-variable system. Because a better controller gives a better control, it is expected to improve the dynamic ESC algorithm especially in the presence of constraints.

Bibliography

Abd-Elrady, E., Gan, L., 2008. Identification of hammerstein and wiener models using spectral magnitude matching. IFAC Proceedings Volumes 41 (2), 6440–6445.

Aliev, F. A., Jamalbayov, M., et al., 2015. Theoretical basics of mathematical modeling of the gas lift process in the well-reservoir system (russian). In: SPE Russian Petroleum Technology Conference. Society of Petroleum Engineers.

Downs, J. J., Skogestad, S., 2011. An industrial and academic perspective on plantwide control. Annual Reviews in Control 35 (1), 99–110.

Eikrem, G. O., Aamo, O. M., Foss, B. A., et al., 2008. On instability in gas lift wells and schemes for stabilization by automatic control. SPE Production & Operations 23 (02), 268–279.

Gupta, V., Grossmann, I. E., 2012. Optimal development planning of offshore oil and gas field infrastructure under complex fiscal rules.

Hamedi, H., Rashidi, F., Khamehchi, E., 2011. A novel approach to the gas-lift allocation optimization problem. Petroleum Science and Technology 29 (4), 418–427.

Hunnekens, B., Haring, M., van de Wouw, N., Nijmeijer, H., 2014. A dither-free extremum-seeking control approach using 1st-order least-squares fits for gradient estimation. In: Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on. IEEE, pp. 2679–2684.

Jäschke, J., Skogestad, S., 2014. Optimal operation of heat exchanger networks with stream split: Only temperature measurements are required. Computers & Chemical Engineering 70, 35–49.

Krishnamoorthy, D., Pavlov, A., Li, Q., 2016. Robust extremum seeking control with application to gas lifted oil wells. IFAC-PapersOnLine 49 (13), 205–210.

Krstić, M., Wang, H.-H., 2000. Stability of extremum seeking feedback for general nonlinear dynamic systems. Automatica 36 (4), 595–601.

Leblanc, M., 1922. Sur l'electrification des chemins de fer au moyen de courants alternatifs de frequence elevee. Revue générale de l'électricité 12 (8), 275–277.

Liu, S.-J., Krstic, M., 2012. Stochastic averaging and stochastic extremum seeking. Springer Science & Business Media.

Ljung, L., 1998. System identification. In: Signal analysis and prediction. Springer, pp. 163–173.

Nešić, D., 2009. Extremum seeking control: Convergence analysis. In: Control Conference (ECC), 2009 European. IEEE, pp. 1702–1715.

Peixoto, A. J., Pereira-Dias, D., Xaud, A. F., Secchi, A. R., 2015. Modelling and extremum seeking control of gas lifted oil wells. IFAC-PapersOnLine 48 (6), 21–26.

Plucenio, A., Pagano, D. J., Camponogara, E., Traple, A., Teixeira, A., 2009. Gas-lift optimization and control with nonlinear mpc. IFAC Proceedings Volumes 42 (11), 904–909.

Tan, Y., Li, Y., Mareels, I. M., 2013. Extremum seeking for constrained inputs. IEEE Transactions on Automatic Control 58 (9), 2405–2410.

Appendix

In this part, the tuning parameters are listed firstly in section 8.1, both for the dynamic ESC and the classic ESC. The parameters are decided by a trial and error method, and the best tuning parameters are only stated in tables.

In the case of the modified dynamic ESC, the simulation parameters are the same as the dynamic ESC. One important value if the threshold to switch the ARX model with the LS method which is stated in each section in the main part.

Secondly, the Matlab scripts for the simulations are attached for each case study.

8.1 List of tuning parameters

Tuning parameters	**Notations**	**Case study 1**
Sampling time	T_s	1s
Window size	l	$4 \cdot 600$
Controller gain	K_i	$-1 \cdot 10^{-7}$
PRBS calculation steps		600
PRBS amplitude		0.01
ARX order		arx(data,[1,1,0])

Table 8.1: Tuning parameters for Case study 1:dynamic ESC

Tuning parameters	**Notations**	**Case study 1**
Sampling time	T_s	1s
Controller gain	K_i	$1 \cdot 10^{-8}$
Low pass filter frequency	ω_l	$0.1 \cdot \frac{1}{6000}$
High pass filter frequency	ω_h	$5 \cdot \frac{1}{6000}$
Sine perturbation frequency	ω	$\frac{1}{6000}$
Sine perturbation amplitude	α	0.01

Table 8.2: Tuning parameters for Case study 1:classic ESC

Tuning parameters	Notations	Case study 2		
		G1	G2	G3
Sampling time	T_s	1s	1s	1s
Window size	l	$4 \cdot 180$	$4 \cdot 180$	$4 \cdot 180$
Controller gain	K_i	0.005	0.005	0.005
PRBS cal. steps		30	30	30
PRBS amplitude		1	1	1
ARX order		arx(data,[1,1,0])	arx(data,[2,2,0])	arx(data,[2,2,0])

Table 8.3: Tuning parameters for Case study 2:dynamic ESC

Tuning parameters	Notations	Case study 2		
		G1	G2	G3
Sampling time	T_s	1s	1s	1s
Controller gain	K_i	0.005	0.005	0.005
Low pass filter frequency	ω_l	$0.1 \cdot \frac{1}{800}$	$0.1 \cdot \frac{1}{1800}$	$0.1 \cdot \frac{1}{1800}$
High pass filter frequency	ω_h	$5 \cdot \frac{1}{800}$	$5 \cdot \frac{1}{1800}$	$5 \cdot \frac{1}{1800}$
Sine perturbation frequency	ω	$\frac{1}{800}$	$\frac{1}{1800}$	$\frac{1}{1800}$
Sine perturbation amplitude	α	1	1	1

Table 8.4: Tuning parameters for Case study 2:classic ESC

Tuning parameters	**Notations**	**Case study 3**
Sampling time	T_s	1s
Window size	l	$3 \cdot 180$
Controller gain	K_i	Gas lifted oil well $1 : 0.0005$ Gas lifted oil well $2 : 0.0004$ Gas lifted oil well $3 : 0.0010$ Gas lifted oil well $4 : 0.0050$ Gas lifted oil well $5 : 0.0010$ Gas lifted oil well 6 : -
PRBS calculation steps		60
PRBS amplitude		0.1
ARX order		arx(data,[1,1,0])

Table 8.5: Tuning parameters for Case study 3:dynamic ESC

8.2 Matlab Script for Case study 1

8.2.1 Dynamic ESC

```
clear
clc

addpath ('D:\matlab\casadi-matlabR2014b-v3.2.3')
addpath('D:\matlab')
import casadi.*

warning('off','all')

u  = MX.sym('u',1);  % split ratio - control input
T  = MX.sym('T',1);  % temperature - measured cost
T1 = MX.sym('T1',1); % temperature in channel 1
T2 = MX.sym('T2',1); % temperature in channel 2

w  = MX.sym('w',1);  % total flow rate
Th = MX.sym('Th',2); % temperature coming into each heaters
cp = MX.sym('cp',2); % specific heat capacity for 2 channels
UA = MX.sym('UA',2); % heat transfer coeffieicnt*area

tau1 = 600; tau2 = 600; tau3 = 600;
par.UA = [50;80]+273;
par.cp = [50;30];
par.Th = [120;220]+273;
par.w  = 12;

Ts = 1;

k1 = w*cp(1)/UA(1);
```

```
k2 = w*cp(2)/UA(2);
T1_ss = Th(1)/(u*k1 + 0.5);
T2_ss = Th(2)/((1-u)*k2 + 0.5);
T_ss = u*T1 + (1-u)*T2;

% ODE model
dT1 = (T1_ss - T1)/tau1;
dT2 = (T2_ss - T2)/tau2;
dT  = (T_ss - T)/tau3;

diff = vertcat(dT1,dT2,dT);
diff = substitute(diff,cp,par.cp);
diff = substitute(diff,UA,par.UA);
diff = substitute(diff,Th,par.Th);
diff = substitute(diff,w,par.w);

x_var = vertcat(T1,T2,T);
p_var = vertcat(u);

L = -T; % cost function (econimic objective)

ode = struct('x',x_var,'p',p_var,'ode',diff,'quad',L);
opts = struct('tf',Ts);

% create IDAS integrator
F = integrator('F','cvodes',ode,opts);

%%
% initialization
xf = [180.9681,818.8306,244.8138];
u_in0 = 0.9;
u_in  = u_in0;

Ki = -1e-07;
```

```
i = 4*600;
nIter = 2*7*10^4;
prbs=0;
h = waitbar(0,'Simulation in Progress...');

for sim_k = 1:nIter
 waitbar(sim_k /nIter,h,sprintf('Time: %0.0f min',sim_k*Ts/60))

      Fk = F('x0',xf,'p',u_in);

        xf = full(Fk.xf);
        x_real(:,sim_k) = xf;
        J_real(sim_k) = full(Fk.qf);

        Ju_SS(sim_k) = ...
            -0.5*par.Th(1)/(u_in*par.w*par.cp(1)/par.UA(1) + ...
            0.5).^2 + ...
            0.5*par.Th(2)/((1-u_in)*par.w*par.cp(2)/par.UA(2) + ...
                0.5).^2;

        sim.y(sim_k) = full(Fk.qf) + 0.000*randn(1);;
        sim.u(sim_k) = u_in + 0.000*randn(1);

        if sim_k>1
        y_SS(sim_k) = sim.y(sim_k) + ...
            (sim.y(sim_k)-sim.y(sim_k-1))*600; % inverse: G(s)^-1
        end
        if sim_k > i

            ymeas = sim.y(sim_k-i:sim_k)';
            umeas = sim.u(sim_k-i:sim_k)';

            Y0 = ymeas - mean(ymeas);
            U0 = umeas - mean(umeas);

            data = iddata(Y0,U0,Ts);
            sysARX = arx(data,[1,1,0]);
            sysD = idss(sysARX);
            sys = d2c(sysD);
            Ju_hat = (-sys.C*(sys.A\sys.B) + sys.D);
            Ju(sim_k) = Ju_hat;

            if sim_k > i   % I-controller
            u_in0 =   u_in0  +( 1*Ki*Ju_hat );
            else
                u_in0 = u_in0;
            end
        end

         if rem(sim_k,600)==0     % calculate next prbs wave in ...
             every 600 steps
           prbs = 1*idinput(1);

         end
        u_in = max(0.001,min(0.999,u_in0+0.01*prbs));
end
close(h);

%% Plotting
figure(1)
set(0,'DefaultAxesFontSize', 18)

subplot(3,1,1)
hold all
plot(sim.u)
ylabel 'u'
legend('u','location','eastoutside')

```

```
subplot(3,1,2)
hold all
plot(x_real')
ylabel 'y'
legend('T_1','T_2','T_{out}','location','eastoutside')

subplot(3,1,3)
hold all
plot(1.*Ju');
plot(Ju_SS,'--');
legend('J_u','J_{uss}','location','eastoutside');
ylabel 'Ju'
```

8.2.2 Modified Dynamic ESC

```
clear
clc

addpath ('D:\matlab\casadi-matlabR2014b-v3.2.3')
addpath('D:\matlab')
import casadi.*

warning('off','all')

u  = MX.sym('u',1);  % split ratio - control input
T  = MX.sym('T',1);  % temperature - measured cost
T1 = MX.sym('T1',1); % temperature in channel 1
T2 = MX.sym('T2',1); % temperature in channel 2

w  = MX.sym('w',1);  % total flow rate
Th = MX.sym('Th',2); % temperature coming into each heaters
cp = MX.sym('cp',2); % specific heat capacity for 2 channels
```

```matlab
UA = MX.sym('UA',2); % heat transfer coeffieicnt*area

tau1 = 600; tau2 = 600; tau3 = 600;
par.UA = [50;80]+273;
par.cp = [50;30];
par.Th = [120;220]+273;
par.w  = 12;

Ts = 1; % Sampling time

k1 = w*cp(1)/UA(1);
k2 = w*cp(2)/UA(2);
T1_ss = Th(1)/(u*k1 + 0.5);
T2_ss = Th(2)/((1-u)*k2 + 0.5);
T_ss = u*T1 + (1-u)*T2;

% ODE model
dT1 = (T1_ss - T1)/tau1;
dT2 = (T2_ss - T2)/tau2;
dT  = (T_ss - T)/tau3;

diff = vertcat(dT1,dT2,dT);
diff = substitute(diff,cp,par.cp);
diff = substitute(diff,UA,par.UA);
diff = substitute(diff,Th,par.Th);
diff = substitute(diff,w,par.w);

x_var = vertcat(T1,T2,T);
p_var = vertcat(u);

L = -T; % cost function (econimic objective)

ode = struct('x',x_var,'p',p_var,'ode',diff,'quad',L);
opts = struct('tf',Ts);
```

```

% create IDAS integrator
F = integrator('F','cvodes',ode,opts);

%%
% initialization
xf = [180.9681,818.8306,244.8138];
u_in0 = 0.9;
u_in  = u_in0;

Ki = -1e-07;
l = 4*600;
nIter = 2*7*10^4;
prbs=0;

h = waitbar(0,'Simulation in Progress...');
for sim_k = 1:nIter
 waitbar(sim_k /nIter,h,sprintf('Time: %0.0f min',sim_k*Ts/60))

      Fk = F('x0',xf,'p',u_in);

        xf = full(Fk.xf);
        x_real(:,sim_k) = xf;
        J_real(sim_k) = full(Fk.qf);

        Ju_SS(sim_k) = ...
            -0.5*par.Th(1)/(u_in*par.w*par.cp(1)/par.UA(1) + ...
            0.5).^2 + ...
            0.5*par.Th(2)/((1-u_in)*par.w*par.cp(2)/par.UA(2) + ...
                0.5).^2;

        sim.y(sim_k) = full(Fk.qf);
        sim.u(sim_k) = u_in + 0.000*randn(1);

        if sim_k>1
        y_SS(sim_k) =  sim.y(sim_k) + ...
            (sim.y(sim_k)-sim.y(sim_k-1))*600; % inverse: G(s)^-1
        end
        if sim_k > l

            ymeas = sim.y(sim_k - l:sim_k)';
            umeas = sim.u(sim_k - l:sim_k)';

            Ju(l) = 1;     % initialization
            Threshold = 50;     % a threshold to switch the ARX ...
                model to the LS method

        if abs(Ju(sim_k-1)) > Threshold     % ARX estimation

            Y0 = ymeas - mean(ymeas);
            U0 = umeas - mean(umeas);

            data = iddata(Y0,U0,Ts);
            sysARX = arx(data,[1,1,0]);
            sysD = idss(sysARX);
            sys = d2c(sysD);
            Ju_hat = (-sys.C*(sys.A\sys.B) + sys.D);
            Ju(sim_k) = Ju_hat;
            flag(sim_k)=1;

        else                                % LS estimation
            Y = ymeas;
            U = umeas;
            X = [U ones(size(U))];
            b = (inv(X'*X))*(X'*Y);
            Ju_hat = b(1,1);
            Ju(sim_k) = Ju_hat;
            flag(sim_k)=0;
        end

            if sim_k > 1    % I-controller
            u_in0 =   u_in0  +( 1*Ki*Ju_hat );
            else
                u_in0 = u_in0;
            end
        end

         if rem(sim_k,600)==0     % calculate next prbs wave in ...
             every 600 steps
           prbs = 1*idinput(1);

         end
        u_in = max(0.001,min(0.999,u_in0+0.01*prbs));
end
close(h);

%%
figure(1)
set(0,'DefaultAxesFontSize', 18)

subplot(4,1,1)
hold all
plot(sim.u)
ylabel 'u'
legend('u','location','eastoutside')

subplot(4,1,2)
hold all
plot(x_real')
ylabel 'y'
legend('T_1','T_2','T_{out}','location','eastoutside')

```

```
subplot(4,1,3)
hold all
plot(1.*Ju');
plot(Ju_SS,'--');
plot(k,'k:');
legend('J_u','J_{uss}','location','eastoutside');
ylabel 'Ju'

subplot(4,1,4)
hold all
plot(flag)
ylabel 'flag'
legend('1:ARX','0:LS','location','eastoutside')
```

8.2.3 Classic ESC

```
clear
clc

addpath ('D:\matlab\casadi-matlabR2014b-v3.2.3')
addpath('D:\matlab')
import casadi.*

warning('off','all')

u  = MX.sym('u',1);  % split ratio - control input
T  = MX.sym('T',1);  % temperature - measured cost
T1 = MX.sym('T1',1); % temperature in channel 1
T2 = MX.sym('T2',1); % temperature in channel 2

w  = MX.sym('w',1);  % total flow rate
Th = MX.sym('Th',2); % temperature coming into each heaters
```

```
cp = MX.sym('cp',2); % specific heat capacity for 2 channels
UA = MX.sym('UA',2); % heat transfer coeffieicnt*area

tau1 = 600; tau2 = 600; tau3 = 600;
par.UA = [50;80]+273;
par.cp = [50;30];
par.Th = [120;220]+273;
par.w  = 12;

Ts = 1;

k1 = w*cp(1)/UA(1);
k2 = w*cp(2)/UA(2);
T1_ss = Th(1)/(u*k1 + 0.5);
T2_ss = Th(2)/((1-u)*k2 + 0.5);
T_ss = u*T1 + (1-u)*T2;

% ODE model
dT1 = (T1_ss - T1)/tau1;
dT2 = (T2_ss - T2)/tau2;
dT  = (T_ss - T)/tau3;

diff = vertcat(dT1,dT2,dT);
diff = substitute(diff,cp,par.cp);
diff = substitute(diff,UA,par.UA);
diff = substitute(diff,Th,par.Th);
diff = substitute(diff,w,par.w);

x_var = vertcat(T1,T2,T);
p_var = vertcat(u);

L = -T; % cost function (econimic objective)

ode = struct('x',x_var,'p',p_var,'ode',diff,'quad',L);
```

```
opts = struct('tf',Ts);

% create IDAS integrator
F = integrator('F','cvodes',ode,opts);

%%
% initialization
xf = [180.9681,818.8306,244.8138];
u_in0 = 0.9;
u_in  = u_in0;

T = 6000;                % sine perturbation time constant
Th = (0.2)*T;            % High pass filter time constant
Tl = 10*T;               % Low pass filter time constant
amplitude = 0.01;        % sine perturbation amplitude

Ki = 1e-08;              % intergtal controller gain
nIter = 1e6;
ARX = 1;
prbs = 0;

%intitalization
z(1) = 0;
x(1) = 0;
u_esc(1) = u_in0;
y_esc(1) = xf(3);
sim.u(1) = u_in0;
sim.y(1) = xf(3);
J(1) = y_esc(1);
u_hat(1) = u_esc(1);
f = 1/T;        % sine perturbation frequency

% filter coefficients
al = Ts/(Ts + Tl);
```

```
ah = Th/(Ts + Th);

h = waitbar(0,'Simulation in Progress...');
for sim_k  = 2:nIter
    waitbar(sim_k /nIter,h,sprintf('Time: %0.0f min',sim_k*Ts/60))

    Fk = F('x0',xf,'p',u_in);
    xf = full(Fk.xf);

    sim.y(sim_k) = xf(3) + 0.00*randn(1);
    sim.u(sim_k) = u_in + 0.00*randn(1);

    y_esc(sim_k) = sim.y(sim_k);
    J(sim_k) = sim.y(sim_k);
    u_hat(sim_k) = sim.u(sim_k);


    z(sim_k) = ah*z(sim_k-1) + ah*J(sim_k) - ah*J(sim_k-1); ...
                        % High pass filter
    x(sim_k) = ((1-al)*x(sim_k-1) + ...
        al*z(sim_k)*sin(2*pi*f*Ts*(sim_k)));  % correlation and ...
        Low pass filter
    % Ju = x.*2/amplitude;
    u_hat(sim_k) = u_hat(sim_k-1) + Ts*Ki*x(sim_k).*2/amplitude; ...
                   % integration
    u_esc(sim_k) = u_hat(sim_k) + amplitude*sin(2*pi*f*Ts*(sim_k)); ...
                % estimated optimal input + dither

    u_in = max(0.001,min(0.999,u_esc(sim_k)));
end
close(h);

%% Plotting
figure(1)
```

```
subplot(311)
hold all
plot(sim.y)
ylabel 'y'

subplot(312)
hold all
plot(sim.u)
ylabel 'u'

subplot(313)
hold all
plot(x.*2/amplitude)
ylabel 'J_u'
```

8.3 Matlab Script for Case study 2

8.3.1 Plant process with First order transfer function

Dynamic ESC

```
clear
clc
warning('off','all')

addpath ('D:\matlab\casadi-matlabR2014b-v3.2.3')
addpath('D:\matlab')
import casadi.*

Ts = 1; % Sampling time
y = MX.sym('y');
u = MX.sym('u');
tau = (174/Ts);

% ODE model
dx1 = ((-0.1*u^2+4*u+5) - y)/tau;

ode = struct('x',y,'p',u,'ode',dx1,'quad',y);
opts = struct('tf',Ts);

% create IDAS integrator
F = integrator('F','cvodes',ode,opts);

%%
% initilization
xf = 42.5;
u_in0 = 15;
```

```
u_in = u_in0;

Ki = 0.005;                     % Controller gain
l = 4*180;                      % Window size
nIter = 2*3600;
ARX = 1;
prbs = 0;

h = waitbar(0,'Simulation in Progress...');
for sim_k  = 1:nIter
    waitbar(sim_k /nIter,h,sprintf('Time: %0.0f min',sim_k*Ts/60))

    Fk = F('x0',xf,'p',u_in);
    xf = full(Fk.xf);

    sim.y(sim_k) = xf + 0.00*randn(1);
    sim.u(sim_k) = u_in + 0.00*randn(1);
    sim.JuSS(sim_k) = -0.2*u_in+4;

    if sim_k>1
        sim.ySS(sim_k) =  sim.y(sim_k) + ...
            (sim.y(sim_k)-sim.y(sim_k-1))*tau; % inverse: G(s)^-1
    end

    if sim_k > l
        ymeas = sim.y(sim_k-l:sim_k)';
        umeas = sim.u(sim_k-l:sim_k)';
        if ARX
            Y0 = ymeas - mean(ymeas);
            U0 = umeas - mean(umeas);

            data = iddata(Y0,U0,Ts);
            sysARX = arx(data,[1,1,0]);
            sysD = idss(sysARX);
```

```
                sys = d2c(sysD);
                Ju_hat = (-sys.C*(sys.A\sys.B) + sys.D);
                Ju(sim_k) = Ju_hat;
            end
            if sim_k > 1  % I-controller
                u_in0 =   u_in0  + Ki*Ju_hat ;
            else
                u_in0 = u_in0 ;
            end
        end

    if rem(sim_k,30)==0     % calculate next prbs wave in every 30 steps
      prbs = 1*idinput(1);
    end
  u_in = u_in0+ prbs;     % add perturbation
  end
  close(h);

  %% Plotting

  figure(1)
  subplot(311)
  hold all
  plot(sim.y)
  plot(sim.ySS,'--')
  grid on
  ylabel 'y'
  legend('y','y_{ss}','location','Northeastoutside')

  subplot(312)
  hold all
  plot(sim.u)
  grid on
  ylabel 'u'
```

```
legend('u','location','Northeastoutside')

subplot(313)
hold all
plot(Ju)
plot(sim.JuSS,'--')
grid on
ylabel 'J_u'
xlabel 'Number of iterations'
legend('J_u','J_{uss}','location','Northeastoutside')
```

Classic ESC

```
clear
clc

addpath ('D:\matlab\casadi-matlabR2014b-v3.2.3')
addpath('D:\matlab')
import casadi.*

Ts = 1;
y = MX.sym('y');
u = MX.sym('u');
tau = (174/Ts);

% ODE model
dx1 = ((-0.1*u^2+4*u+5) - y)/tau;

ode = struct('x',y,'p',u,'ode',dx1,'quad',y);
opts = struct('tf',Ts);

% create IDAS integrator
```

```
F = integrator('F','cvodes',ode,opts);

%%
% initialization
xf = 42.5;
u_in0 = 15;
u_in = u_in0;

T = 800;                % sine perturbation time constant
Th = (0.2)*T;           % High Pass Filter
Tl = 10*T;              % Low Pass filter
amplitude = 1;          % sine amplitude

Ki = 0.005;              % intergtal controller gain
nIter = 4*1e5;
ARX = 1;

%intitalization
z(1) = 0;
x(1) = 0;
u_hat(1) = 15;
y_esc(1) = 42.5;
sim.u(1) = 15;
sim.y(1) = 42.5;
J(1) = y_esc(1);
u_hat(1) = u_hat(1);
f = 1/T;        % sine perturbation frequency

% filter coefficients
al = Ts/(Ts + Tl);
ah = Th/(Ts + Th);

h = waitbar(0,'Simulation in Progress...');
for sim_k  = 2:nIter
```

```
    waitbar(sim_k /nIter,h,sprintf('Time: %0.0f min',sim_k*Ts/60))

    Fk = F('x0',xf,'p',u_in);
    xf = full(Fk.xf);

    sim.y(sim_k) = xf + 0.005*randn(1);
    sim.u(sim_k) = u_in + 0.005*randn(1);

    y_esc(sim_k) = sim.y(sim_k);
    J(sim_k) = sim.y(sim_k);
    u_hat(sim_k) = sim.u(sim_k);


    z(sim_k) = ah*z(sim_k-1) + ah*J(sim_k) - ah*J(sim_k-1); ...
                          % High pass filter
    x(sim_k) = ((1-al)*x(sim_k-1) + ...
        al*z(sim_k)*sin(2*pi*f*Ts*(sim_k)));  % correlation and ...
        Low pass filter
    % Ju = x.*2/amplitude;
    u_hat(sim_k) = u_hat(sim_k-1) + Ts*Ki*x(sim_k).*2/amplitude; ...
                    % integration
    u_esc(sim_k) = u_hat(sim_k) + amplitude*sin(2*pi*f*Ts*(sim_k)); ...
                 % estimated optimal input + dither

    u_in = u_esc(sim_k);
end
close(h);
%% Plotting
figure(1)
subplot(311)
hold all
plot(sim.y)
ylabel 'y'

```

```
subplot(312)
hold all
plot(sim.u)
ylabel 'u'

subplot(313)
hold all
plot(x.*2/amplitude)
ylabel 'J_u'
```

8.3.2 Plant process with second order transfer function

Dynamic ESC

```
clear
clc

addpath ('D:\matlab\casadi-matlabR2014b-v3.2.3')
addpath('D:\matlab')
import casadi.*

Ts = 1; % Sampling time
y = MX.sym('y');
u = MX.sym('u');
x1 = MX.sym('x1');
x2 = MX.sym('x2');
tau = (174/Ts);

tau_1 = 174;
tau_2 = 60;
tau_a = 40;                    % tau_a = 0 and 40 are studied
```

```

[a, b, c, d] = tf2ss([0 tau_a 1],[tau_1*tau_2 tau_1+tau_2 1]);

% ODE model
% dx/dt=Ax+Bu, y=Cx+Du, x=[x1 x2]', A=2X2 matrix
dx1 = a(1,1)*x1+a(1,2)*x2+b(1,1)*(-0.1*u^2+4*u+5);
dx2 = a(2,1)*x1+a(2,2)*x2+b(2,1)*(-0.1*u^2+4*u+5);
y = c(1,1)*x1+c(1,2)*x2+d*(-0.1*u^2+4*u+5);

ode = struct('x',vertcat(x1,x2),'p',u,'ode',vertcat(dx1,dx2),'quad',y);
opts = struct('tf',Ts);

% create IDAS integrator
F = integrator('F','cvodes',ode,opts);

%%
% initialization
xf = [0; 42.5/c(1,2)];
u_in0 = 15;
u_in = u_in0;

Ki = 0.005;                    % Controller gain
l = 4*180;                     % Window size
nIter = 4*3600;
ARX = 1;
pbrs = 0;

h = waitbar(0,'Simulation in Progress...');
for sim_k  = 1:nIter
    waitbar(sim_k /nIter,h,sprintf('Time: %0.0f min',sim_k*Ts/60))

    Fk = F('x0',xf,'p',u_in);
    xf = full(Fk.xf);

```

```
        sim.y(sim_k) = c*xf + 0.00*randn(1);
        sim.u(sim_k) = u_in + 0.00*randn(1);
        sim.ySS(sim_k) = -0.1*u_in^2+4*u_in+5;
        sim.JuSS(sim_k) = -0.2*u_in+4;

        if sim_k > 1
            ymeas = sim.y(sim_k-1:sim_k)';
            umeas = sim.u(sim_k-1:sim_k)';
            if ARX
                Y0 = ymeas - mean(ymeas);
                U0 = umeas - mean(umeas);

                data = iddata(Y0,U0,Ts);
                sysARX = arx(data,[2,2,0]);
                sysD = idss(sysARX);
                sys = d2c(sysD);
                Ju_hat = (-sys.C*(sys.A\sys.B) + sys.D);
                Ju(sim_k) = Ju_hat;
            end

            if sim_k > 1     % I-controller
                u_in0 =   u_in0  +Ki*Ju_hat ;
            else
                u_in0 = u_in0 ;
            end
        end

    if rem(sim_k,30)==0
        pbrs = 1*idinput(1);
    end
    u_in = u_in0+ pbrs;      % add perturbation
    end
    close(h);
    %% Plotting
```

```
figure(1)
subplot(311)
hold all
plot(sim.y)
plot( sim.ySS,'--')
grid on
ylabel 'y'
legend('y','y_{ss}','location','Northeastoutside')

subplot(312)
hold all
plot(sim.u)
set(gca,'FontSize',20);
grid on
ylabel 'u'
legend('u','location','Northeastoutside')

subplot(313)
hold all
plot(Ju)
plot(sim.JuSS,'--')
grid on
ylabel 'J_u'
legend('J_u','J_{uss}','location','Northeastoutside')
```

Modified Dynamic ESC

```
clear
clc

addpath ('D:\matlab\casadi-matlabR2014b-v3.2.3')
addpath('D:\matlab')
```

```
import casadi.*

Ts = 1; % Sampling time
y = MX.sym('y');
u = MX.sym('u');
x1 = MX.sym('x1');
x2 = MX.sym('x2');
tau = (174/Ts);

tau_1 = 174;
tau_2 = 60;
tau_a = 0;                  % tau_a = 0 and 40 are studied

[a, b, c, d] = tf2ss([0 tau_a 1],[tau_1*tau_2 tau_1+tau_2 1]);

% ODE model
% dx/dt=Ax+Bu, y=Cx+Du, x=[x1 x2]', A=2X2 matrix
dx1 = a(1,1)*x1+a(1,2)*x2+b(1,1)*(-0.1*u^2+4*u+5);
dx2 = a(2,1)*x1+a(2,2)*x2+b(2,1)*(-0.1*u^2+4*u+5);
y = c(1,1)*x1+c(1,2)*x2+d*(-0.1*u^2+4*u+5);

ode = struct('x',vertcat(x1,x2),'p',u,'ode',vertcat(dx1,dx2),'quad',y);
opts = struct('tf',Ts);

% create IDAS integrator
F = integrator('F','cvodes',ode,opts);

%%
% initialization
xf = [0; 42.5/c(1,2)];
u_in0 = 15;
u_in = u_in0;

Ki = 0.005;                  % Controller gain
```

```
l = 4*180;                   % Window size
nIter = 4*3600;
pbrs = 0;

h = waitbar(0,'Simulation in Progress...');
for sim_k  = 1:nIter
    waitbar(sim_k /nIter,h,sprintf('Time: %0.0f min',sim_k*Ts/60))

    Fk = F('x0',xf,'p',u_in);
    xf = full(Fk.xf);

    sim.y(sim_k) = c*xf + 0.00*randn(1);
    sim.u(sim_k) = u_in + 0.00*randn(1);
    sim.ySS(sim_k) = -0.1*u_in^2+4*u_in+5;
    sim.JuSS(sim_k) = -0.2*u_in+4;

    if sim_k > l
        ymeas = sim.y(sim_k-l:sim_k)';
        umeas = sim.u(sim_k-l:sim_k)';

        Ju(l) = 1;           % initialization
        Threshold = 0.1;     % a threshold to switch the ARX model ...
            to the LS method
                             % 0.1 for tau_a=0, 0.05 for tau_a=40

        if Ju(sim_k-1) > Threshold      % ARX estimation

            Y0 = ymeas - mean(ymeas);
            U0 = umeas - mean(umeas);

            data = iddata(Y0,U0,Ts);
            sysARX = arx(data,[2,2,0]);
            sysD = idss(sysARX);
            sys = d2c(sysD);
            Ju_hat = (-sys.C*(sys.A\sys.B) + sys.D);
            Ju(sim_k) = Ju_hat;
            flag(sim_k) = 1;
        else                               % LS estimation
            Y = ymeas;
            U = umeas;
            X = [U ones(size(U))];
            b = (inv(X'*X))*(X'*Y);
            Ju_hat = b(1,1);
            Ju(sim_k) = Ju_hat;
            flag(sim_k) = 0;
        end

        if sim_k > 1                       % I-controller
            u_in0 =   u_in0  +Ki*Ju_hat ;
        else
            u_in0 = u_in0 ;
        end
    end

if rem(sim_k,30)==0
    pbrs = 1*idinput(1);
end
u_in = u_in0+ pbrs;  % add perturbation
end
close(h);

%%

figure(1)
subplot(411)
hold all
plot(sim.y)
plot(sim.ySS,'--')
```

```
grid on
ylabel 'y'
legend('y','y_{ss}','location','Northeastoutside')

subplot(412)
hold all
plot(sim.u)
grid on
ylabel 'u'
legend('u','location','Northeastoutside')

subplot(413)
hold all
plot(Ju)
plot(sim.JuSS,'--')
grid on
ylabel 'J_u'
legend('J_u','J_{uss}','location','Northeastoutside')

subplot(414)
hold all
plot(flag)
grid on
ylabel 'flag'
```

Classic ESC

```
clear
clc

addpath ('D:\matlab\casadi-matlabR2014b-v3.2.3')
addpath('D:\matlab')
```

```
import casadi.*

Ts = 1; % Sample time

y = MX.sym('y');
u = MX.sym('u');
x1 = MX.sym('x1');
x2 = MX.sym('x2');
tau = (174/Ts);

tau_1 = 174;
tau_2 = 60;
tau_a = 40;     % tau_a = 0 and 40 are studied

[amplitude,b,c,d] = tf2ss([0 tau_a 1] , [tau_1*tau_2 tau_1+tau_2 1]);

% ODE model
% dx/dt=Ax+Bu, y=Cx+Du, x=[x1 x2]', A=2X2 matrix
dx1 = amplitude(1,1)*x1+amplitude(1,2)*x2+b(1,1)*(-0.1*u^2+4*u+5);
dx2 = amplitude(2,1)*x1+amplitude(2,2)*x2+b(2,1)*(-0.1*u^2+4*u+5);
y = c(1,1)*x1+c(1,2)*x2+d*(-0.1*u^2+4*u+5);

ode = struct('x',vertcat(x1,x2),'p',u,'ode',vertcat(dx1,dx2),'quad',y);
opts = struct('tf',Ts);

% create IDAS integrator
F = integrator('F','cvodes',ode,opts);

%%
% initialization
xf = [0; 42.5/c(1,2)];
u_in0 = 15;
u_in = u_in0;

```

```
T = 1800;                       % sine perturbation time constant
Th = (0.2)*T;                   % High Pass Filter time constant
Tl = 10*T;                      % Low Pass Filter time constant
amplitude = 1;                   % sine amplitude

Ki = 0.001;                      % Controller gain
nIter = 4e5;
ARX = 1;

%intitalisation
z(1) = 0;
x(1) = 0;
u_hat(1) = 15;
y_esc(1) = 42.5;
sim.u(1) = 15;
sim.y(1) = 42.5;
J(1) = y_esc(1);
u_hat(1) = u_hat(1);
f = 1/T;

% filter coefficients
al = Ts/(Ts + Tl);
ah = Th/(Ts + Th);

h = waitbar(0,'Simulation in Progress...');
for sim_k  = 2:nIter
    waitbar(sim_k /nIter,h,sprintf('Time: %0.0f min',sim_k*Ts/60))

    Fk = F('x0',xf,'p',u_in);
    xf = full(Fk.xf);

    sim.y(sim_k) = c*xf + 0.005*randn(1);
    sim.u(sim_k) = u_in + 0.005*randn(1);

    y_esc(sim_k) = sim.y(sim_k);
    J(sim_k) = sim.y(sim_k);
    u_hat(sim_k) = sim.u(sim_k);

    z(sim_k) = ah*z(sim_k-1) + ah*J(sim_k) - ah*J(sim_k-1); ...
                      % High pass filter
    x(sim_k) = ((1-al)*x(sim_k-1) + ...
        al*z(sim_k)*sin(2*pi*f*Ts*(sim_k))); % correlation and ...
        Low pass filter
    % Ju = x.*2/amplitude;
    u_hat(sim_k) = u_hat(sim_k-1) + Ts*Ki*x(sim_k).*2/amplitude; ...
                 % I-controller
    u_esc(sim_k) = u_hat(sim_k) + amplitude*sin(2*pi*f*Ts*(sim_k)); ...
              % estimated optimal input + dither

    u_in = u_esc(sim_k);
 end
close(h);

%% Plotting
figure(1)
subplot(311)
hold all
plot(sim.y)
ylabel 'y'

subplot(312)
hold all
plot(sim.u)
ylabel 'u'

subplot(313)
hold all
plot(x.*2/amplitude)
```

```
ylabel 'J_u'
```

8.4 Matlab Script for Case study 3

8.4.1 Dynamic ESC

```
clear
clc

addpath ('D:\matlab\casadi-matlabR2014b-v3.2.3')
addpath('D:\matlab')
import casadi.*

Ts = 1;

y1 = MX.sym('y1');
u1 = MX.sym('u1');
tau1 = (174/Ts);

y2 = MX.sym('y2');
u2 = MX.sym('u2');
tau2 = (180/Ts);

y3 = MX.sym('y3');
u3 = MX.sym('u3');
tau3 = (170/Ts);

y4 = MX.sym('y4');
u4 = MX.sym('u4');
tau4 = (176/Ts);

y5 = MX.sym('y5');
u5 = MX.sym('u5');
tau5 = (180/Ts);
```

```

y6 = MX.sym('y6');
u6 = MX.sym('u6');
tau6 = (177/Ts);

% ODE model
dx1 = ((-0.1*u1^2+4*u1+5) - y1)/tau1;
dx2 = ((-0.5*u2^2+10*u2+5) - y2)/tau2;
dx3 = ((-0.2*u3^2+6*u3+10) - y3)/tau3;
dx4 = ((-0.05*u4^2+3*u4+10) - y4)/tau4;
dx5 = ((-0.04*u5^2+2*u5+15) - y5)/tau5;
dx6 = ((-0.1*u6^2+2*u6+20) - y6)/tau6;

% cost functions
L1 = -y1;
L2 = -y2;
L3 = -y3;
L4 = -y4;
L5 = -y5;
L6 = -y6;

ode1 = struct('x',y1,'p',u1,'ode',dx1,'quad',L1);
ode2 = struct('x',y2,'p',u2,'ode',dx2,'quad',L2);
ode3 = struct('x',y3,'p',u3,'ode',dx3,'quad',L3);
ode4 = struct('x',y4,'p',u4,'ode',dx4,'quad',L4);
ode5 = struct('x',y5,'p',u5,'ode',dx5,'quad',L5);
ode6 = struct('x',y6,'p',u6,'ode',dx6,'quad',L6);

opts = struct('tf',Ts);

% create IDAS integrator
F1 = integrator('F','cvodes',ode1,opts);
F2 = integrator('F','cvodes',ode2,opts);
F3 = integrator('F','cvodes',ode3,opts);
```

```
F4 = integrator('F','cvodes',ode4,opts);
F5 = integrator('F','cvodes',ode5,opts);
F6 = integrator('F','cvodes',ode6,opts);

%%
% initialization
xf1 = 16.1;
u_in01 = 3;
u_in1 = u_in01;

xf2 =47;
u_in02 = 14;
u_in2 = u_in02;

xf3 = 30.8;
u_in03 = 4;
u_in3 = u_in03;

xf4 = 53.2;
u_in04 = 24;
u_in4 = u_in04;

xf5 = 16.96;
u_in05 = 1;
u_in5 = u_in05;

xf6 = 30;
u_in06 = 10;
u_in6 = u_in06;

ARX = 1;
prbs1 = 0;
prbs2 = 0;
prbs3 = 0;
```

```
prbs4 = 0;
prbs5 = 0;
prbs6 = 0;
y_SS1 = xf1;
y_SS2 = xf2;
y_SS3 = xf3;
y_SS4 = xf4;
y_SS5 = xf5;
y_SS6 = xf6;
umax = 56;            % constraint

Ki = 0.0005;          % intergtal controller gain
l = 3*180;       % window size of data
nIter = 40000;

h = waitbar(0,'Simulation in Progress...');
for sim_k  = 1:nIter
    waitbar(sim_k /nIter,h,sprintf('Time: %0.0f min',sim_k*Ts/60))
    warning('off','all')

    Fk1 = F1('x0',xf1,'p',u_in1);
    Fk2 = F2('x0',xf2,'p',u_in2);
    Fk3 = F3('x0',xf3,'p',u_in3);
    Fk4 = F4('x0',xf4,'p',u_in4);
    Fk5 = F5('x0',xf5,'p',u_in5);
    Fk6 = F6('x0',xf6,'p',u_in6);

    xf1 = full(Fk1.xf);
    xf2 = full(Fk2.xf);
    xf3 = full(Fk3.xf);
    xf4 = full(Fk4.xf);
    xf5 = full(Fk5.xf);
    xf6 = full(Fk6.xf);

    Ju_SS1(sim_k) = -0.2*u_in1+4;   % steady state gradient
    Ju_SS2(sim_k) = -1*u_in2+10;
    Ju_SS3(sim_k) = -0.4*u_in3+6;
    Ju_SS4(sim_k) = -.1*u_in4+3;
    Ju_SS5(sim_k) = -0.08*u_in5+2;
    Ju_SS6(sim_k) = -.2*u_in6+2;

    sim.y1(sim_k) = xf1 + 0.00*randn(1);
    sim.y2(sim_k) = xf2 + 0.00*randn(1);
    sim.y3(sim_k) = xf3 + 0.00*randn(1);
    sim.y4(sim_k) = xf4 + 0.00*randn(1);
    sim.y5(sim_k) = xf5 + 0.00*randn(1);
    sim.y6(sim_k) = xf6 + 0.00*randn(1);

    sim.u1(sim_k) = u_in1 + 0.00*randn(1);
    sim.u2(sim_k) = u_in2 + 0.00*randn(1);
    sim.u3(sim_k) = u_in3 + 0.00*randn(1);
    sim.u4(sim_k) = u_in4 + 0.00*randn(1);
    sim.u5(sim_k) = u_in5 + 0.00*randn(1);
    sim.u6(sim_k) = u_in6 + 0.00*randn(1);

    if sim_k>1
        y_SS1(sim_k) =  sim.y1(sim_k) + ...
             (sim.y1(sim_k)-sim.y1(sim_k-1))*tau1; % inverse: G(s)^-1
        y_SS2(sim_k) =  sim.y2(sim_k) + ...
             (sim.y2(sim_k)-sim.y2(sim_k-1))*tau2;
        y_SS3(sim_k) =  sim.y3(sim_k) + ...
             (sim.y3(sim_k)-sim.y3(sim_k-1))*tau3;
        y_SS4(sim_k) =  sim.y4(sim_k) + ...
             (sim.y4(sim_k)-sim.y4(sim_k-1))*tau4;
        y_SS5(sim_k) =  sim.y5(sim_k) + ...
             (sim.y5(sim_k)-sim.y5(sim_k-1))*tau5;
        y_SS6(sim_k) =  sim.y6(sim_k) + ...
             (sim.y6(sim_k)-sim.y6(sim_k-1))*tau6;
    end

    if sim_k > l        % ARX estimation part
            ymeas1 = sim.y1(sim_k-l:sim_k)';
            ymeas2 = sim.y2(sim_k-l:sim_k)';
            ymeas3 = sim.y3(sim_k-l:sim_k)';
            ymeas4 = sim.y4(sim_k-l:sim_k)';
            ymeas5 = sim.y5(sim_k-l:sim_k)';
            ymeas6 = sim.y6(sim_k-l:sim_k)';

            umeas1 = sim.u1(sim_k-l:sim_k)';
            umeas2 = sim.u2(sim_k-l:sim_k)';
            umeas3 = sim.u3(sim_k-l:sim_k)';
            umeas4 = sim.u4(sim_k-l:sim_k)';
            umeas5 = sim.u5(sim_k-l:sim_k)';
            umeas6 = sim.u6(sim_k-l:sim_k)';

        if ARX
            Y10 = ymeas1 - mean(ymeas1);
            Y20 = ymeas2 - mean(ymeas2);
            Y30 = ymeas3 - mean(ymeas3);
            Y40 = ymeas4 - mean(ymeas4);
            Y50 = ymeas5 - mean(ymeas5);
            Y60 = ymeas6 - mean(ymeas6);

            U10 = umeas1 - mean(umeas1);
            U20 = umeas2 - mean(umeas2);
            U30 = umeas3 - mean(umeas3);
            U40 = umeas4 - mean(umeas4);
            U50 = umeas5 - mean(umeas5);
            U60 = umeas6 - mean(umeas6);

            data1 = iddata(Y10,U10,Ts);
            data2 = iddata(Y20,U20,Ts);
            data3 = iddata(Y30,U30,Ts);
            data4 = iddata(Y40,U40,Ts);
            data5 = iddata(Y50,U50,Ts);
            data6 = iddata(Y60,U60,Ts);

            sysARX1 = arx(data1,[1,1,0]);
            sysD1 = idss(sysARX1);
            sys1 = d2c(sysD1);
            Ju_hat1 = -sys1.C*(sys1.A\sys1.B) + sys1.D;
            Ju1(sim_k) = Ju_hat1;

            sysARX2 = arx(data2,[1,1,0]);
            sysD2 = idss(sysARX2);
            sys2 = d2c(sysD2);
            Ju_hat2 = -sys2.C*(sys2.A\sys2.B) + sys2.D;
            Ju2(sim_k) = Ju_hat2;

            sysARX3 = arx(data3,[1,1,0]);
            sysD3 = idss(sysARX3);
            sys3 = d2c(sysD3);
            Ju_hat3 = -sys3.C*(sys3.A\sys3.B) + sys3.D;
            Ju3(sim_k) = Ju_hat3;

            sysARX4 = arx(data4,[1,1,0]);
            sysD4 = idss(sysARX4);
            sys4 = d2c(sysD4);
            Ju_hat4 = -sys4.C*(sys4.A\sys4.B) + sys4.D;
            Ju4(sim_k) = Ju_hat4;

            sysARX5 = arx(data5,[1,1,0]);
            sysD5 = idss(sysARX5);
            sys5 = d2c(sysD5);
            Ju_hat5 = -sys5.C*(sys5.A\sys5.B) + sys5.D;
            Ju5(sim_k) = Ju_hat5;

            sysARX6 = arx(data6,[1,1,0]);
            sysD6 = idss(sysARX6);
            sys6 = d2c(sysD6);
            Ju_hat6 = -sys6.C*(sys6.A\sys6.B) + sys6.D;
            Ju6(sim_k) = Ju_hat6;
        end

        if sim_k > 1    % I-controller
            u_in01 =   u_in01  +0.0005*(Ju_hat1-Ju_hat2);
            u_in02 =   u_in02  +0.0004*(Ju_hat2-Ju_hat3);
            u_in03 =   u_in03  +0.001*(Ju_hat3-Ju_hat4);
            u_in04 =   u_in04  +0.005*(Ju_hat4-Ju_hat5);
            u_in05 =   u_in05  +0.001*(Ju_hat5-Ju_hat6);
            u_in06 =   umax - (u_in01+u_in02+u_in03+u_in04+u_in05);
        end
        end

        if rem(sim_k,60)==0     % calculate next prbs wave in every ...
            60 steps
           prbs1 = 1*idinput(1);
           prbs2 = -prbs1;
           prbs3 = 1*idinput(1);
           prbs4 = -prbs3;
           prbs5 = 1*idinput(1);
           prbs6 = -prbs5;
        end

        u_in1 = u_in01+0.1*prbs1;
        u_in2 = u_in02+0.1*prbs2;
        u_in3 = u_in03+0.1*prbs3;
        u_in4 = u_in04+0.1*prbs4;
        u_in5 = u_in05+0.1*prbs5;
        u_in6 = u_in06+0.1*prbs6;
```

```
end
close(h);

%% Plotting
figure(1)
set(0,'DefaultAxesFontSize', 18)

subplot(6,3,1)
hold all
plot(sim.u1)
ylabel 'u_1'
subplot(6,3,4)
hold all
plot(sim.u2)
ylabel 'u_2'
subplot(6,3,7)
hold all
plot(sim.u3)
ylabel 'u_3'
subplot(6,3,10)
hold all
plot(sim.u4)
ylabel 'u_4'
subplot(6,3,13)
hold all
plot(sim.u5)
ylabel 'u_5'
subplot(6,3,16)
hold all
plot(sim.u6)
ylabel 'u_6'

subplot(6,3,2)
hold all
```

```
plot(sim.y1)
plot(y_SS1, '--')
ylabel 'y_1'
subplot(6,3,5)
hold all
plot(sim.y2)
plot(y_SS2, '--')
ylabel 'y_2'
subplot(6,3,8)
hold all
plot(sim.y3)
plot(y_SS3, '--')
ylabel 'y_3'
subplot(6,3,11)
hold all
plot(sim.y4)
plot(y_SS4, '--')
ylabel 'y_4'
subplot(6,3,14)
hold all
plot(sim.y5)
plot(y_SS5, '--')
ylabel 'y_5'
subplot(6,3,17)
hold all
plot(sim.y6)
plot(y_SS6, '--')
ylabel 'y_6'

subplot(6,3,3)
hold all
plot(1.*Ju1')
plot(Ju_SS1,'--')
ylabel 'J_u1'
```

```
subplot(6,3,6)
hold all
plot(1.*Ju2')
plot(Ju_SS2,'--')
ylabel 'J_u2'
subplot(6,3,9)
hold all
plot(1.*Ju3')
plot(Ju_SS3,'--')
ylabel 'J_u3'
subplot(6,3,12)
hold all
plot(1.*Ju4')
plot(Ju_SS4,'--')
ylabel 'J_u4'
subplot(6,3,15)
hold all
plot(1.*Ju5')
plot(Ju_SS5,'--')
ylabel 'J_u5'
subplot(6,3,18)
hold all
plot(1.*Ju6')
plot(Ju_SS6,'--')
ylabel 'J_u6'
```