Master's thesis

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

Salmon Yemane Ghebredngl

Developing a hybrid data-driven health-aware controller for optimizing production in a gas-lifted oil well network

Master's thesis in Chemical engineering
Supervisor: Johannes Jåschke
Co-supervisor: Jose Otavio Assumpcao Matias
June 2021

NTNU
Norwegian University of Science and Technology

Salmon Yemane Ghebredngl

Developing a hybrid data-driven health-aware controller for optimizing production in a gas-lifted oil well network

Master's thesis in Chemical engineering
Supervisor: Johannes Jåschke
Co-supervisor: Jose Otavio Assumpcao Matias
June 2021

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

NTNU
Norwegian University of
Science and Technology

Declaration of Authorship

I, SALMON YEMANE, declare that this thesis titled, " Developing a hybrid data-driven health-aware controller for optimizing production in a gas-lifted oil well network" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Salmon Yemane

Date: 10/06/2021

"The only impossible journey is the one you never begin"

Anthony Robbins

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Abstract

Faculty of Natural Sciences (NV)
Department of chemical engineering

Master in science

Developing a hybrid data-driven health-aware controller for optimizing production in a gas-lifted oil well network

by SALMON YEMANE

In subsea oil and gas production systems, unexpected breakdown and maintenance interventions are costly. Therefore, a maintenance strategy that can ensure reliable operation is required. However, there is an intuitive trade-off between optimizing production and minimizing equipment degradation. For example, in most oil wells, it is always desirable to extract as much oil as possible, which harms the remaining useful life of the equipment. To avoid equipment wear, engineers often adopt a conservative production strategy, leading to sub-optimal operation and potential profit loss. This thesis proposes a new approach based on forecasting system degradation through a predictive process model. Prognostics and health monitoring (PHM) are then integrated into the control structure to avoid conservative operation by actively steering plant degradation and preventing violation of health constraints.

This thesis develops, therefore, a novel method that aims at solving the combined problem. Both the data-driven degradation model and the process models are solved through hybrid data-driven model predictive control. In this control structure, the controller calculates the optimal inputs using the data-driven models, and system feedback in the form of diagnostics is added to cope with the uncertainties in the system. The proposed method is applied to a synthetic case study, in which the system of interest is an oil and gas well network with artificial gas- lifting.

The simulation results show that hybrid model predictive control is a possible alternative to solving the control problem. However, the plant-model mismatch was observed to have a detrimental effect on the performance of the HAC controllers. The HAC with NN model in both prognostics and diagnostics showed a more significant plant-model mismatch and had the shortest breakdown time[1]. In contrast, the HAC with no plant-model mismatch was the one with the longest breakdown time. We can say that the hybrid data-driven HAC controllers were prone to plant-model mismatch and managed to minimize conservativeness in production but at the cost of constraint violation which made them break down early and, therefore, produced less oil in total. The overall conclusion is that the performance of hybrid-data-driven HAC is dependent on the type of data-driven model and the quality of the feedback from diagnostics. In addition, the complexity of data-driven models doesn't necessarily give a better result.

[1]The time at which erosion of one of the wells exceeds the failure threshold

Sammendrag

Uventede sammenbrudd og vedlikehold innen offshore oljeproduksjonssystemer er kostbare. Derfor er det nødvendig å ha en vedlikeholdsstrategi som kan sikre pålitelig drift for å opprettholde produksjonskapasiteten. Det er imidlertid viktig å gjøre en avveing mellom det å optimere produksjonen, og det å minimere utstyr degradering. For eksempel er det alltid ønskelig i de fleste oljebrønner å utvinne så mye olje som mulig, noe som skader utstyrets gjenværende levetid. For å unngå slitasje på utstyr, tar ingeniører ofte en konservativ produksjonsstrategi, noe som fører til suboptimal drift og potensielt fortjenestetap. Målet med denne oppgaven er å undersøke muligheten til å automatisere denne prosessen ved å bruke modellbasert prediktiv regulering (MPC) med integrert helseovervåkingsystem for å redusere konservativ drift ved å aktivt styre degradering av strupeventilen (”choke valve”) og forhindre brudd på helsebegrensninger.

Denne oppgaven utvikler derfor en ny metodikk som tar sikte på å løse det kombinerte problemet. Både den databasert prognose- degraderingsmodellen og prosessmodellen løses gjennom en hybrid datadrevet MPC. I denne kontrollstrukturen bergener kontrolleren de optimale "inputs" ved hjelp av de datadrevne modellene, og "system feedback" i form av diagnostikk legges til for å takle usikkerheten i systemet. Den foreslåtte metoden ble anvendt på en syntetisk case-studie av subsea oljeproduksjonssystem.

Simuleringsresultatene viser at hybridmodell prediktiv kontroll er et mulig alternativ til å løse kontrollproblemet. Imidlertid ble "plant-model mismatch" observert til å ha en skadelig effekt på ytelsen til HAC-kontrollerne. HAC med NN-modell i både prognostikk og diagnostikk viste en mer signifikant "plant-model mismatch" og hadde kortest ytelsestid, mens HAC uten "plant-model mismatch" var den med lengst ytelsestid. Vi kan si at de hybride HAC-kontrollerne var utsatt for uoverensstemmelse mellom plantemodeller og klarte å minimere konservativitet i produksjonen, men på bekostning av brudd på helse begrensningen som fikk dem til å ha kort ytelsestid og derfor produserte mindre olje totalt. Den overordnede konklusjonen er at ytelsen til hybrid HAC er avhengig av typen datadrevet modell og kvaliteten på "feedback" fra diagnostikk. Videre, kompleksiteten til datadrevne modeller gir ikke nødvendigvis et bedre resultat.

Acknowledgements

It has been a very interesting experience to work with this nascent technology and follow the current researches and developments of control frameworks. Furthermore, I would like to express my greatest gratitude to my supervisor Associate Professor Johannes Jäschke and co-supervisor Jose Otavio Assumpcao Matias for their valuable guidance, assistance and support during the whole process of this thesis. Special thanks to Jose Otavio Assumpcao Matias for always being available for questions and taking the time for me in his hectic schedule.

Contents

List of Figures

List of Tables

List of Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
CBM	**C**onditioni **B**ased **M**aintenance
DAE	**D**ifferential **A**lgebraic **E**quations
DMC	**D**ynamic **M**atrix **C**ontrol
EMPC	**E**conomic **M**odel **P**redictive **C**ontrol
GOR	**G**as **O**il **R**atio
HAC	**H**ealth- **A**ware **C**ontrol
LQR	**L**inear **Q**uadratic **R**egulator
MIMO	**M**ulti- **I**nput **M**ulti- **O**utput
MPC	**M**odel **P**redictive **C**ontrol
NMPC	**N**onlinear **M**odel **P**redictive **C**ontrol
ODE	**O**rdinary **D**ifferential **E**quations
PHM	**P**rognostics and **H**ealth **M**onitoring
PDFs	**P**robability **D**ensity **F**unctions
RUL	**R**emaining **U**seful **L**ifetime

Physical Constants

Cross-sectional area of riser	$A_r = 0.0115\,\mathrm{m}^2$
Density of oil	$\rho_o = 800\,\mathrm{kg\,m}^{-3}$
Density of oil in riser	$\rho_{ro} = 800\,\mathrm{kg\,m}^{-3}$
Diameter of riser	$D_r = 0.121\,\mathrm{m}$
Dynamic viscosity of oil	$\mu_o = 0.001\,\mathrm{Pa\,s}$
Height of riser	$H_r = 500\,\mathrm{m}$
Length of riser	$L_r = 500\,\mathrm{m}$
Number of wells	$n_w = 3$
Riser temperature	$T_r = 303\,\mathrm{K}$
Sampling time	$T = 86\,400\,\mathrm{s}$
Separator pressure	$p_s = 20\,\mathrm{bar}$
Valve constant for riser valve	$C_{pr} = 0.01$

List of Symbols

A_t	Area exposed to erosion	m^2
A_p	Area of pipe	m^2
A_g	Area of the the annulus	m^2
α	Characteristic impact angle	rad
A_r	Cross-sectional area of piping over the injection point	m^2
A_w	Cross-sectional area of piping under the injection point	m^2
ρ	Density	kg m $^{-3}$
ρ_p	Density of sand particles	kg m $^{-3}$
ρ_a	Density of the gas in the annulus	kg m $^{-3}$
ρ_w	Density of the mixture in the tubing	kg m $^{-3}$
ρ_o	Density of the oil	kg m $^{-3}$
A	Dimensionless constant	-
μ s^{-1}	Dynamic viscosity	kg m^{-1}
A_g	Effective gallery area	m^2
E	Erosion	mm
ER	Erosion rate	mm yr^{-1}
g	Gravitational constant	m s^{-2}
d	Length from cage and choke body	m
Δu_{max}	Maximum change in input	kg s^{-1}
C_1	Model geometry factor	-
G	Particle size correction factor	-
γ	Relationship b/n particle diameter and diameter of choke	-
γ_c	Relative critical particle diameter	-
$\dot{m}_p$	Sand rate	kg s^{-1}
d_p	Sand particle diameter	m
C_{unit}	Unit conversion factor	mm m^{-1}
C_{iv}	Valve constant for the injection valve	-
C_{pc}	Valve constant for the production valve	-
ρ_s	Weighting for slack variable	-

Dedicated to my dear parents Yemane Ghebredngl and Ghenet Mekonen . . .

Chapter 1

Introduction

1.1 Motivation

The global energy consumption in the last half-century has been increasing at an unprecedented pace and is expected to continue to grow over the next 50 years (Goswami and Kreith, 2015). Even though the transition to renewable energy sources is also accelerating with emerging technologies, oil and gas remain relevant to the energy sector. However, as old oil fields mature and get depleted, the capacity to extract oil and gas in harsh offshore environments efficiently and safely will be essential in the coming years.

Operating a subsea production facility in remote areas of the sea is challenging due to the inaccessibility for maintenance and reliability checks. As a consequence, performing maintenance in such facilities is costly. It is therefore critical to minimize the downtime of the facility while maximizing both reliability and production. However, there is always a trade-off between optimizing production and maintaining equipment's health, which results in a conservative design and operation of equipment such that the risk of failure is minimized. Moreover, since many of the units in the subsea industry are remotely located, operational data can be scarce and inaccurate. Traditional maintenance strategies can therefore lead to a large profitability loss.

Over the recent years, condition-based maintenance (CBM) and prognostics and health management (PHM) have emerged as powerful technologies impacting maintenance practices. We are witnessing a true paradigm shift in how complex dynamic systems such as air-crafts, shipboard systems, and industrial and manufacturing processes are operated and maintained (Vachtsevanos and Vachtsevanos, 2006). The old approach was to perform maintenance when the equipment broke down, or performance goes down severely. The new policy is based on forecasting system degradation through a prognostic process on which the health of the equipment is continuously monitored for obtaining an early indication of failure. This new approach enables significantly better equipment maintenance and prognosis of the RUL[1].

This thesis addresses the questions raised above by developing a control structure that incorporates health monitoring, prognostics, and diagnostics of critical system components. As, a result we obtain the so-called Health-aware control structure, which achieves the control and production objectives without jeopardizing the equipment's health.

[1] Remaining Useful Lifetime

1.2 Objectives

The primary purpose of this thesis is to study the performance of the Health-aware controller applied to a gas-lifted subsea oil and gas production network. Diagnostics and prognostics models used in this thesis are based on (Jahren, 2020). Here, the author used several statistical methods and an artificial data set to obtain the prognostics and diagnostics of a choke valve in a gas-lifted subsea oil and gas production network. Those models are integrated into the production planning problem to avoid conservative operation by actively steering the choke degradation and preventing violation of health-critical constraints. Therefore, the health-aware MPC controller is implemented using both data-driven models in the controller and a phenomenological model in the plant. The hybrid nature of this control structure will be the center of this thesis's discussion. This leads to the main research question that is answered in this thesis:

How can we systematically integrate PHM into an existing control structure, and how will the plant-model mismatch in the prognostics and the imprecise system feedback in diagnostics affect the performance of the Health-aware controller that aims to maximize production without jeopardizing equipment RUL?

1.3 Thesis Structure

This thesis is structured as follows. Chapter 2 covers the literature review, highlighting important aspects that support this work. Chapter 3 describes the theory used to obtain the results presented and discussed in Chapter 5. Chapter 4 presents methodology and process description of the case study used in this thesis. This thesis is closed with the conclusion and final remarks in Chapter 6.

1.4 Thesis Contribution

In the Authors view , the main contributions of the thesis is

- **A new method of simultaneously optimizing production and maintaining the degradation of critical equipment's below acceptable levels. This is implemented by integrating equipment prognostics and diagnostics into the control structure**
- **Developing a hybrid data-driven MPC controller that uses both data-driven and first principal models in the control structure. The controller will use the data-driven models instead of the actual plant model to calculate the optimal inputs in this structure.**
- **The application of health-aware model predictive control into a gas-lifted subsea oil and gas production network**

Chapter 2

Literature review

In this chapter, a literature review on the topic of prognostics and control is revised. It is divided into three parts: (I) the first part covers condition-based maintenance (CBM) and prognostics and health monitoring (PHM); (II) the second part covers Model predictive control (MPC) and (III) the third part covers health-aware control (HAC).

Part I

2.1 Condition Based Maintenance (CBM)

Traditionally, maintenance has been performed after the failure of the equipment. This run-to-failure approach, known as reactive maintenance, has dominated the industry for decades (Vachtsevanos and Vachtsevanos, 2006). However, the focus has recently shifted to proactive maintenance, where operational data is used to perform maintenance at constant intervals (Vachtsevanos and Vachtsevanos, 2006). This approach is known as clock-based maintenance. Another more advanced approach is age-based maintenance, which is based on frequent monitoring of the remaining useful lifetime of the system through its age and available measurements. The practical drawback to deploying these approaches is that the time between maintenance is based on statistical information, which is insufficient to ensure a satisfactory level of operation until the next planned maintenance stop (Vachtsevanos and Vachtsevanos, 2006).

Currently, condition-based maintenance (CBM) is evolving rapidly and becoming the standard approach in the industry (Vachtsevanos and Vachtsevanos, 2006). Condition-based maintenance uses equipment run-time data to determine the equipment's failure condition, which is then used to plan repair and maintenance before breakdown. In contrast to planned maintenance, where maintenance is performed on predefined intervals, CBM is performed only after a decrease in the condition of the equipment has been detected. This means that CBM is performed while the equipment is operationally active, minimizing disruption and production stops.

The average operation maintenance prices of the different maintenance practices mentioned above are shown in Figure 2.1. Due to the varying degrees of availability for maintenance, the various maintenance practices have different operational costs. Given perfect information about the degradation state of a system, condition-based maintenance is shown to be the cheapest. Furthermore, we can also observe that corrective maintenance is associated with the highest cost compared with the other maintenance strategies. Clock-based and Age-based maintenance have relatively lower prices compared to corrective maintenance.

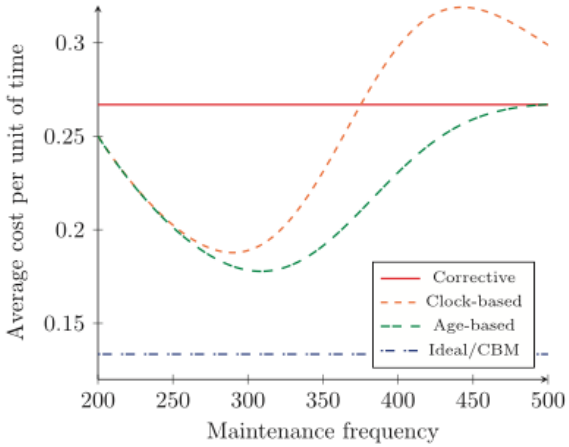

FIGURE 2.1: Cost comparison between different maintenance strategies (Verheyleweghen, 2020)

2.2 Prognostics, diagnostics and health monitoring (PHM)

Understanding the condition and health of the system equipment is fundamental in the field of maintenance optimization. However, the state and health of equipment are usually not measurable directly and are typically estimated using the measurements of other parameters in the process, such as temperature and pressure. In this manner, we can assess the current failure state (diagnosis) and the future failure states (prognosis). We can then use this information to detect failures and determine if the given equipment can perform at an acceptable level until the next scheduled maintenance stop.

Diagnostics and prognostics are often called prognosis and health management (PHM) (Vachtsevanos and Vachtsevanos, 2006). The sole purpose behind PHM is the prognosis, which is the ability to predict the remaining useful lifetime (RUL) of a failing component precisely. The RUL of equipment is defined as the time the equipment's health indicator exceeds its failure threshold. Prognosis has been the Achilles' heel of the CBM and PHM due to considerable model uncertainty (Vachtsevanos and Vachtsevanos, 2006). Prediction of a component's fault evolution requires methodologies that can represent and manage the inherent uncertainties in the model. Furthermore, good probabilistic models of fault growth and statistically sufficient samples of failure data are essential for an accurate and precise prognosis. Therefore, prognosis performance metrics, robust algorithms, and experimental platforms that can provide the needed data have been at the center of CBM/PHM research in the past years (Vachtsevanos and Vachtsevanos, 2006).

Equipment failure prognostics has been approached through various techniques ranging from Bayesian estimation and other statistical methods to artificial intelligence methodologies. Some of the techniques include parameter estimation methods (Ljung, 1999), multi-step adaptive Kalman filter (Lewis, 1986), stochastic autoregressive integrated-moving-average models (Jardim-Gonçalves et al., 1996), and Weibull models (Groer, 2000). While some other works are done using Physics-based models, which explain the degradation process using a phenomenological model. However, they are a minority in this field as the knowledge about degradation processes is currently small (Vachtsevanos and Vachtsevanos, 2006). We can generally categorize these prognosis schemes into three categories: model-based,

probability-based, and data-driven. A comparison of the applicability, cost, and accuracy of these schemes is summarized in Figure 2.2.

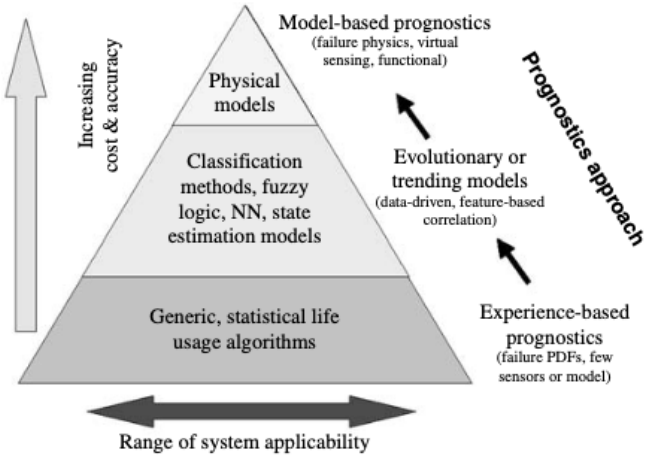

FIGURE 2.2: Prognosis technical approaches (Vachtsevanos and Vachtsevanos, 2006)

2.2.1 Model-based techniques

The model-based prognostic schemes are based on a mathematical first principle model representation of the system. Model-based approaches enable the user to calculate the degradation of critical components as a function of operating conditions. By using advanced stochastic modeling techniques, the model can be used to calculate the statistical distribution of RUL for a particular fault. The advantages of this approach are that they can be excellent if the model is accurate and that we can reuse the same model for different systems by re-parameterizing the model. The disadvantages are that deriving these models requires a deep understanding of the underlying degradation factors, which can be challenging. Moreover, model-based approaches often result in inaccurate prediction if some degradation factors are neglected while deriving the model. The complex nature of components failure can thus make the model too large and numerically expensive to solve.

2.2.2 Probability-based techniques

If a complete dynamic model of the system is impractical, probability-based prognostics can be utilized. Failure data of equipment usually takes a statistical form as failure occurs at different periods. We can therefore apply probabilistic methods to such systems. The advantage of these modes is that they require less information than the model-based techniques. The information needed is present in the probability density functions, not in the dynamic differential equations. Therefore, the prognosis can be easily implemented using the PDFs [1] of the observed data. Furthermore, confidence limits of the results can be used as a performance metric of the accuracy of the predictions.

[1] Probability Density Functions

2.2.3 Data-driven methods

Data-driven methods are machine learning methods that are based on pattern recognition. These techniques can be used when historical fault data leading up to failure is available. These techniques include Artificial Neural Networks (ANN) and fuzzy-logic systems with broad applications in various industries. Neural Network is an algorithm that is based on signal-processing methods present in the nervous system. In contrast, fuzzy-logic systems are based on a system that resembles human linguistic and reasoning abilities. These algorithms provide a structured nonlinear function mapping between the available data and the desired response variables. Some other commonly used methods include Bayesian Networks (BN) and hidden Markov models (HMMs).

In the field of prediction, those mentioned methods have been providing an alternative to both model-based and probability-based methods for years (Wilson and Sharda, 1994). ANN-trained models are known to consistently outperform traditional statistical methods such as regression (Werbos, 1988). The advantages of ANN are that, unlike the conventional model-based methods, ANN is both self-adaptive and data-driven with very few assumptions. Furthermore, ANN algorithms can learn from the data and capture the functional relationship in the data. Therefore, those methods are well suited for most practical problems, where we have access to data and not the complete knowledge of the underlying system. The drawback of using such algorithms is that they are a "black box" and have limited ability to identify possible relationships explicitly. They are also usually prone to overfitting (Tu, 1996).

Part II

2.3 Model predictive control (MPC)

Many strategies have been developed in the field of control engineering to control multiple-input multiple-output (MIMO) systems. One of the commonly used strategies is the optimal control with the Linear Quadratic Regulator (LQR) control. In those strategies, the optimal problems are solved offline with the assumption of a linear model representation of the system without system constraints (Skogestad and Postlethwaite, 2007). In the early years, optimal control techniques were not very popular in the industry due to their inability to deal with system constraints and were often regarded as impractical. As a result, the industrial community started developing a more robust control algorithm, the model predictive control (MPC).

First-generation MPC systems were first developed in 1970 by two industrial research groups. Dynamic Matrix Control (DMC) developed by Shell Oil, and ADERSA (Seborg et al., 2010). The main idea behind the MPC algorithm is to solve an online constrained optimization problem at each time interval, in which the objective function to be minimized measures the closeness of controlled variables to their reference trajectories (Qin and Badgwell, 1997). Since its inception, MPC has become the method of choice for difficult multivariable control problems with inequality constraints. One of its advantages is its ability to control large nonlinear multi-input multi-output (MIMO) systems effectively.

In spite, however, of these advantages, there are serious drawbacks related to its demanding computational capability. Recent technological advancements have substantially reduced the costs and increased the capability of computers to make

the benefits of using computationally intensive control systems larger. Simultaneously, advanced mathematical algorithms for optimization have also greatly improved the speed and reliability of the calculations required by MPC. Informative reviews of MPC are available in books ((Camacho, Bordons, and Normey-Rico, 2003; Maciejowski, 2002; Rossiter, 2003) and research papers ((Chai, Qin, and Wang, 2014; Darby and Nikolaou, 2012)).

Part III

2.4 Combining PHM and control in Health-aware control (HAC)

The main task of this thesis is to combine both PHM and control. Currently, control strategies that consider diagnostics and prognostics into the control structure remain little explored (Bernardino, 2019). Most of the control structures present today ignore the effects of degradation and damage in the control hierarchy. As shown in the block diagram of Figure 2.3, data from PHM is used for decision-making only by the operators. When faults are detected and the alarm goes off, the operators use their knowledge and experience to adjust the setpoints or schedule maintenance. In this setup, the operator interacts with the system by changing the setpoints, tuning the controller, or overriding the controller by directly defining the setpoints.

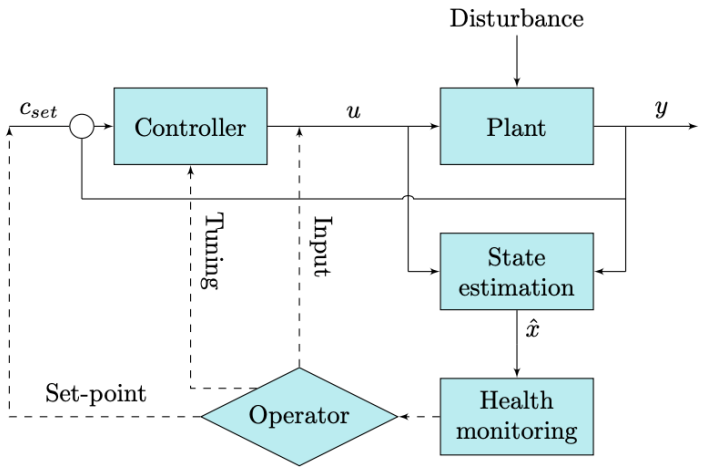

FIGURE 2.3: Block diagram of a system with health-monitoring where an operator is used to adjust the setpoints manually based on PHM data (Verheyleweghen, 2020)

The drawback with this type of control structure is that the decision is left to the operator. The operator's decisions are not always optimal, and there is always a delay due to the human reaction time. This problem can be solved by closing the loop and giving the controller the ability to make optimal decisions. This control structure could significantly improve the response time and performance of the system. However, this requires the integration of PHM into the control structure. The reliability of equipment is then taken into account by introducing new constraints in the control problem. The proposed control structure is shown in Figure 2.4.

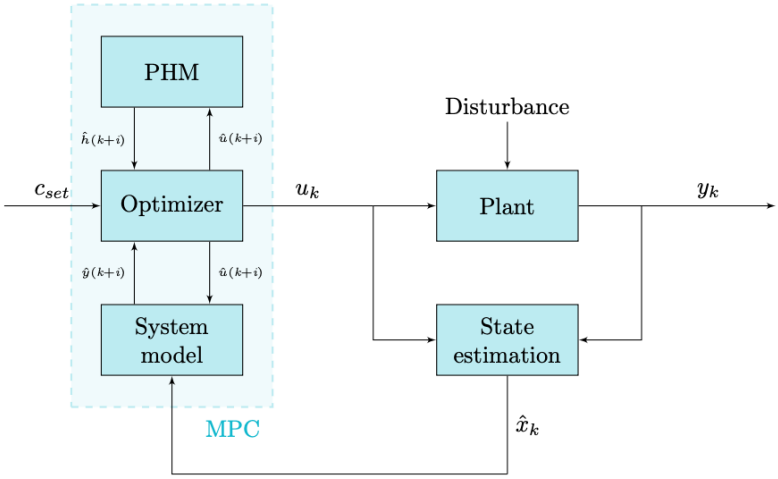

FIGURE 2.4: Block diagram of a system with health-monitoring integrated in the MPC control framework and PHM data is used by the controller to adjust the set-points automatically (Verheyleweghen, 2020)

During the late 70s and early 80s, engineers started discussing the idea of incorporating health prognostics in a control structure (Verheyleweghen, 2020). This was mostly related to the application of control structures in airplanes. The plan was to integrate a supervisory layer that adjusts the control structure based on fault detection and identification techniques. Thus, the control structure would still perform at a satisfactory level despite the biased sensors and fault actuators present in the system. (Chizeck, 1978) first used the term fault-tolerant control (FTC) to describe the control structure. Since its inception, FTC has been the focus of research by the aviation industry.

The first known usage of prognostics in a control structure is found in (Escobet, Puig, and Nejjari, 2012). This paper tries to integrate control and prognosis where a conveyor belt that uses an AC electric motor to move a cart from one end to the other end is used as the system. This new method based on both current and future health state estimates, provided by a prognosis module, takes into account the systems health information in the control objectives. The objective was to extend the useful lifetime of the conveyor belt by adjusting set-points to a simple PID controller.

More case studies that rely on advanced control techniques have also been explored. A research paper, (Sanchez et al., 2015) presented the use of MPC, integrated with a fatigue-based prognosis approach to minimize the damage of wind turbines. Another paper, (Verheyleweghen, Gjøby, and Jäschke, 2018) studied the use of a health-aware robust MPC for a subsea compression system subject to degradation. In this paper, a hierarchical approach was used for operating compressors subject to degradation. The degradation was estimated using Paris Law, one of the most used models to describe crack propagation in systems subject to stress, with a corrective online parameter estimation.

Chapter 3

Theory

This chapter introduces the theory used in the modeling and analysis of the data-driven diagnostic and prognostic models of the choke valves in the gas-lifted subsea oil and gas production network proposed by (Krishnamoorthy, Foss, and Skogestad, 2016). The theory behind the different statistical methods and analysis of the artificial data set used to obtain the prognostics and diagnostics models will be described. Initially, the artificial data set have to be preprocessed, and for this, we will discuss normalization. Afterward, the different regression methods will be introduced, including classical statistical learning methods such as linear regression and more sophisticated artificial neural networks (ANN). Then we will show an overview of the Economic Health- aware Model predictive controller.

3.1 Data Preprocessing

Data pre-processing is a widely used technique that involves transforming raw data into another format before analysis. Raw data sets are often influenced by factors such as noise, uncertainties, and large variance. Several methods exist, such as centering and scaling so that no single variable dominates the system due to its large scale and variance. Furthermore, normalization can also be used when dealing with different scales. Another method that can be applied is principal component analysis (PCA), which is used when the data dimension is large and reduction of the data without losing information is needed to manage the analysis better. In this project, normalization is chosen due to the varying orders of magnitude and units of measurements in the data. It should be noted that throughout this thesis, $\boldsymbol{X}$ is used to denote a n by p data matrix that contains p different variables and n different samples

3.1.1 Normalization

Normalization is used when the data consists of variables with different scales. This process compensates the variability in the orders of magnitude and units of measurements in the data by scaling all the data to be centered with unit variance and mean of zero. This is implemented using the standard score formula:

$$\boldsymbol{Z} = \frac{\boldsymbol{X} - \mu}{\sigma} \tag{3.1}$$

where $\boldsymbol{Z}$ is the standard score, $\boldsymbol{X}$ is the original data matrix, μ is the mean, and σ is the standard deviation.

3.2 Statistical data-driven methods

Data-driven methods are generally divided into prediction and description based on what they are used for as shown in Figure 3.1,. The prediction methods are further divided into two main groups: statistical methods and symbolic methods. The statistical methods are characterized by the representation of knowledge through mathematical models, while the symbolic methods are characterized by the representation of knowledge through the means of symbols and connectives. In this thesis, we are only interested in statistical methods. This section will therefore introduce the statistical methods that will be applied throughout this work including Regression models and Neural Networks. The methods in this section share the common features of using a data matrix $\boldsymbol{X}$ or equivalently a normalized data matrix $\boldsymbol{Z}$, to predict a matrix or vector of responses $\boldsymbol{Y}$. For simplicity in this chapter, $\boldsymbol{X}$ will be used even if most of the work is done on normalized data.

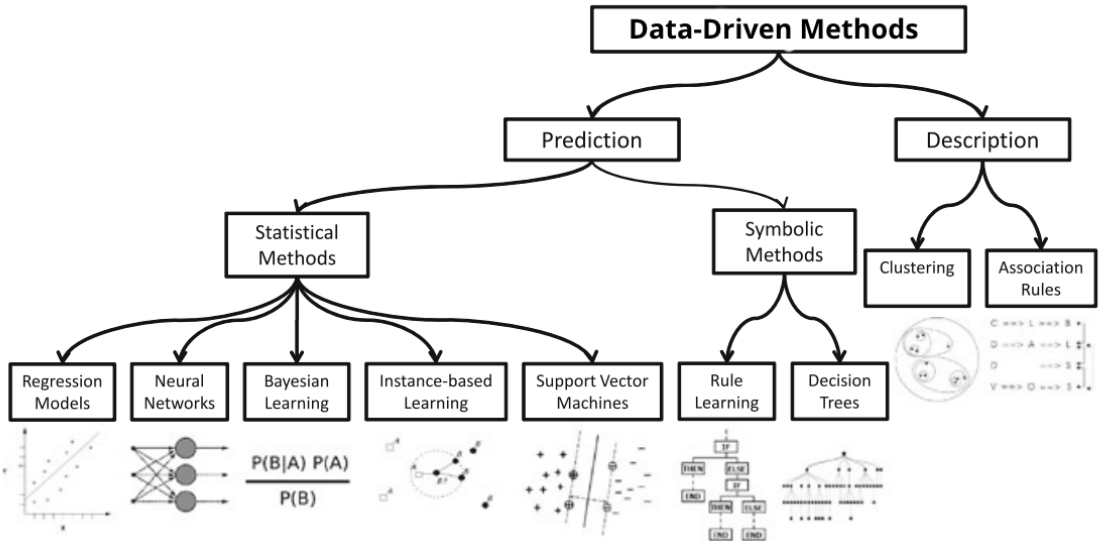

FIGURE 3.1: Data-Driven methods (García, Luengo, and Herrera, 2015)

3.2.1 Regression Models

Regression Models are one of the oldest models, used in estimation tasks (García, Luengo, and Herrera, 2015). Some of the most well known regression models are Linear, quadratic and logistic regression. The central idea of regression is to obtain a model for the functional relationship between a response variable and one or more predictor variables (Ott and Longnecker, 2015).

Linear regression

Linear regression assumes that there is a linear relationship between a response variable, $\mathbf{Y}$, and a set of predictor variables $\mathbf{X}$ along with some noise ϵ. In this model, the error term ϵ is assumed to be normally distributed, homoscedastic, meaning with the same variance at every $\mathbf{X}$ and has mean of zero.

$$\boldsymbol{Y} = \boldsymbol{B}\boldsymbol{X} + \boldsymbol{\epsilon} \tag{3.2}$$

Given a training data we can generate the estimate for $\boldsymbol{B}$, using the least squares method that minimizes the residual sum of squares. The estimate $\hat{\boldsymbol{B}}$ can be obtained with the analytic solution of the problem as follows:

$$\hat{\boldsymbol{B}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y} \tag{3.3}$$

where $\hat{\boldsymbol{Y}}$ and $\hat{\boldsymbol{B}}$ are the estimates of the true values.

From these estimated parameters, the functional relationship between $\mathbf{Y}$ and $\mathbf{X}$ can be found as follows:

$$\hat{\boldsymbol{Y}} = \hat{\boldsymbol{B}}\boldsymbol{X} \tag{3.4}$$

The model used in the thesis contains cross-terms of the predictor variables. The model is then modified and the interaction between the predictor variables are expressed as a product of the predictor variables as follows:

$$Y_i = \beta_0 + \beta_1 X_{i,1} + \beta_2 X_{i,2} + \beta_3 X_{i,1} X_{i,2} + \cdots + \epsilon_i \tag{3.5}$$

where $\beta_0, \beta_1, \beta_2$,and β_3 are the regression coefficient for the predictor variables, and Y_i is the value for the response variable for the ith case.

Subset Selection

The least-squares method can sometimes be further enhanced by setting some coefficients to zero. In the cases where the matrix $\boldsymbol{X}$ is not well condition(for example, strongly correlated predictors), the prediction accuracy of the linear model is negatively affected. One alternative to solve the problem is to set some of the coefficients in $\boldsymbol{B}$ to zero. In this case, the overall prediction accuracy is improved by adding bias to reduce the variance of the predicted values. Furthermore, in the case of problems with several predictors, subset selection is used to determine a smaller subset that shows only the most potent effects.

The main purpose of subset selection is to retain only a subset of the predictor variables and exclude the rest from the model. The least-squares method is then used to estimate the coefficients of the predictor variables that are retained. There are, however, several approaches to variable subset selection with linear regression, such as the best-Subsetselection, Forward- and Backward- Stepwise Selection, and Forward- stagewise regression. In this thesis, stepwise linear regression is used as it is faster than the other model-selection methods (Glen, September 24, 2015).

Stepwise linear regression

Stepwise regression is a statistical method used to build a model by choosing the best subset of models. This is done through a series of F-tests on the estimated coefficients aiming at adding or removing predictor variables of the original model. There are different approaches to this algorithm, such as forward-stepwise and backward-stepwise. Forward stepwise selection starts with the intercept and then sequentially adds into the model the predictor that improves the fit the most. This is repeated until further additions do not improve the fit significantly.

In contrast, the backward selection is initialized with a full model, including all terms and terms whose loss gives the most insignificant deterioration of the model fit are removed. It is also possible to combine both methods where terms are both removed and added at each step in search of the best model. Here, the metric used to

determine if the addition or subtraction of a predictor variable improves the model fit is F-test which is defined statistically as follows:

$$\boldsymbol{F} = \frac{\frac{RSS_1 - RSS_2}{p_2 - p_1}}{\frac{RSS_2}{n - p_2}} \tag{3.6}$$

where RSS_1 and RSS_2 are the residual sum of squares, p is the number of parameters in the models and n is the number of samples. This statistic follows an F distribution with $(p_2 - p_1, n - p_2)$ degrees of freedom.

Forward-stepwise selection is a greedy algorithm, producing a nested sequence of models. In this sense, it might seem sub-optimal compared to the best-subset selection, which finds the best subset that gives the smallest residual sum of squares. However, there are several reasons why it might be preferred. The first reason is that forward-stepwise selection is superior computationally for models with a large number of parameters. It is not always possible to compute the best subset sequence, but we can always compute the forward stepwise sequence. The second reason is that the size of variance in forward stepwise method is smaller than the best subset algorithm. The best subset algorithm pays the price in variance for selecting the best subset of each size (Hastie, Tibshirani, and Friedman, 2009).

3.2.2 Neural Network (NN)

Neural networks, also called Artificial Neural Networks is a set of algorithms that is used to discover underlying relationships in a data set (Chen, December 23, 2020). The NN is inspired by the biological nervous system translated to a computer. A single neuron is shown in Figure 3.2. The connections of thousands of neurons combined produces outstanding prediction results. The layers in NN are usually divided into input layer, hidden layer and output layer. The simplest NNs have only one hidden layer, see Figure 3.3. While more complex NNs have more than two hidden layers. A nonlinear activation function is placed at each neuron in both the hidden layer and the output layer. This activation function is responsible for the powerful prediction power of the NN.

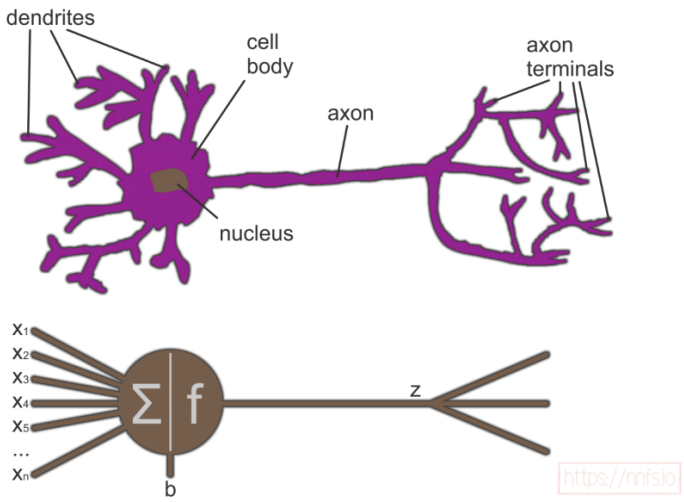

FIGURE 3.2: Diagram of a single neuron (Harrison Kinsley, 2020)

A diagram representation of a static feedforward NN is shown in Figure 3.3. This Neural Network model is described by the formula:

$$y_i = \sum_{j=1}^{L} w_{ij}\sigma(\sum_{k=1}^{n} v_{jk}x_k + \theta_{vj}) + \theta_{wi} \qquad i = 1,2,3,...,m \tag{3.7}$$

The values of x_k are the NN inputs (predictor variables), and y_i are the NN outputs (response). The function $\sigma(.)$ is a non-linear activation function found in the NN model's hidden layer. This Neural Network model has two layers of adjustable weights. The v_{jk} are the hidden layer weights, and the w_{ij} are the output-layer weights. Furthermore, θ_{vj} and θ_{wi} are the hidden-layer biases and the output-layer biases. L gives the number of hidden-layers,

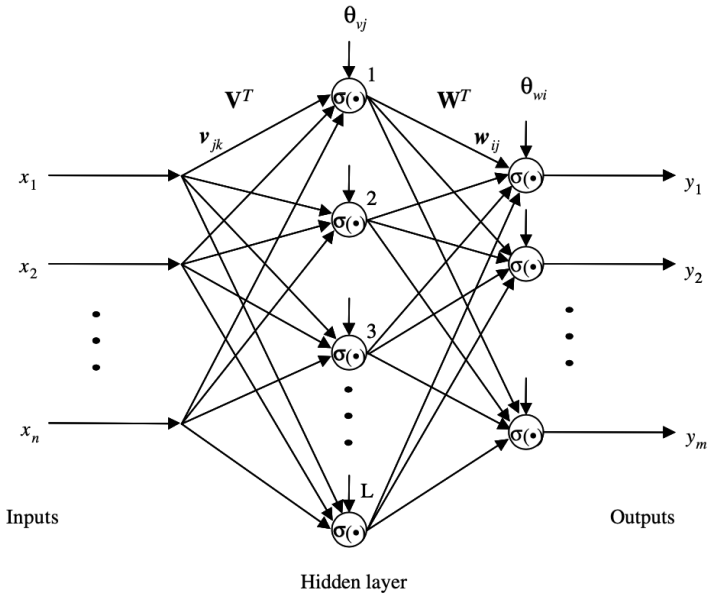

FIGURE 3.3: Diagram of a neural network (Vachtsevanos and Vachtsevanos, 2006)

This can be represented using the weight matrices that contains biases as the first column as follows;

$$\boldsymbol{V^T} = \begin{bmatrix} \theta_{v1} & v_{11} & \cdots & v_{1n} \\ \theta_{v2} & v_{21} & \cdots & v_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \theta_{vL} & v_{L1} & \cdots & v_{Ln} \end{bmatrix} \tag{3.8}$$

$$\boldsymbol{W^T} = \begin{bmatrix} \theta_{w1} & w_{11} & \cdots & w_{1L} \\ \theta_{w2} & w_{21} & \cdots & w_{2L} \\ \vdots & \vdots & \vdots & \vdots \\ \theta_{wm} & w_{m1} & \cdots & w_{mL} \end{bmatrix} \tag{3.9}$$

The Neural Network can then be written as

$$\boldsymbol{Y = W^T\sigma(V^T X)} \tag{3.10}$$

where $\boldsymbol{Y} = [y_1 \ y_2 \ \cdots \ y_m]^T$ is the output vector. Moreover, as the biases are in the first columns of the weight matrices, the input vector with predictor variables is augmented by 1 and defined as follows.

$$\boldsymbol{X} = \begin{bmatrix} \mathbf{1} & \boldsymbol{x_1} & \boldsymbol{x_2} & \cdots & \boldsymbol{x_n} \end{bmatrix}^T \tag{3.11}$$

Then one has the jth row of $\boldsymbol{V^T X}$ is given by:

$$\begin{bmatrix} \theta_{vj} & v_{j1} & v_{j2} & \cdots & v_{jn} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \theta_{vj} + \sum_{k=1}^{n} v_{jk} x_k \tag{3.12}$$

Similarly, the activation function $\sigma()$ used in the equation describing the NN is the augmented hidden-layer function vector, defined for a vector $\boldsymbol{w} = [w_1 \ w_2 \ \cdots \ w_L]^T$ as follows

$$\sigma(w) = \begin{bmatrix} \mathbf{1} & \sigma(w_1) & \sigma(w_2) & \cdots & \sigma(w_L) \end{bmatrix}^T \tag{3.13}$$

The fact that the activation functions $\sigma(.)$ in NN are nonlinear and the weights W and V can be tuned via learning procedures gives the NN a high computing power. More information on NN is found at (Vachtsevanos and Vachtsevanos, 2006). Common functions used as activation function are shown in Figure 3.4. In this thesis, the sigmoid activation function is chosen due to the simple form of its derivative.

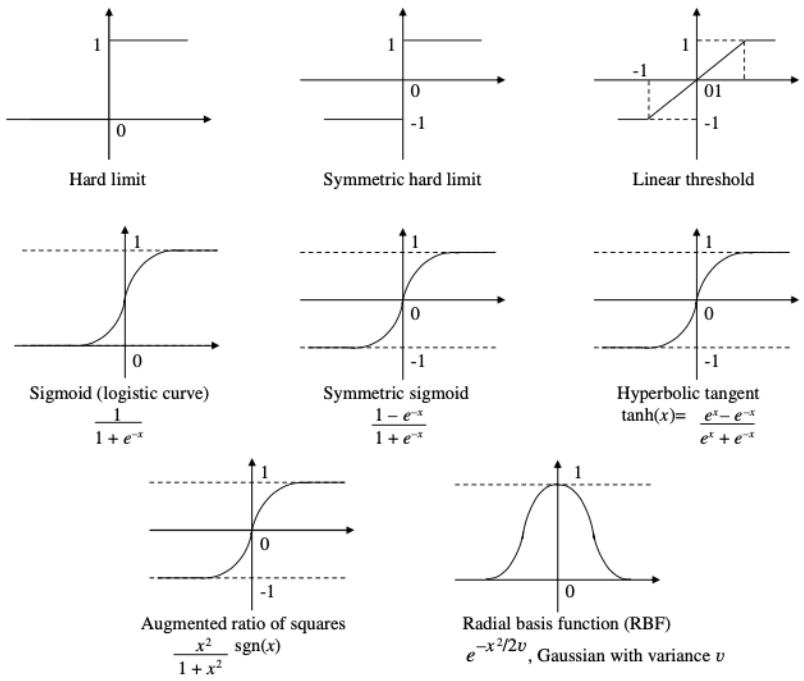

FIGURE 3.4: Common activation functions used in Neural Network (Vachtsevanos and Vachtsevanos, 2006)

The main advantage of NN is that it can be trained to capture the required knowledge, such as system modeling and prediction. The NN model "learns" by adjusting the weights. One of the most common training techniques is a gradient algorithm

based on back propagation error. The algorithm is as follows:

$$W_{t+1} = W_t + F\sigma(V_t^T x_d)E_t^T \tag{3.14}$$

$$V_{t+1} = V_t + Gx_d(\sigma_t'^T W_t E_t)^T \tag{3.15}$$

This algorithm is defined in discrete-time with time index t. The NN output in response to the reference input $x_d \in R^n$ is prescribed as $y_d \in R^m$. The output error,E_t at time t is then given by $E_t = y_d - y_t$, where y_t is the actual output at time t. Furthermore, F and G are weighting matrices chosen by the the user and determines how fast the algorithm converges. The term $\sigma_t'(.)$ is the hidden-layer output gradient and is defined as the derivative of the activation function $\sigma(.)$. The hidden-layer output gradient function for the sigmoid activation function is given as by:

$$\sigma_t' = diag\{\sigma(V^T x_d)\}[I - diag\{\sigma(V^T x_d)\}] \tag{3.16}$$

where $diag\{\}$ is a diagonal matrix with the indicated elements on the diagonal and I is the identity matrix.

3.3 Model predictive control (MPC)

The main idea behind MPC is to solve a constrained optimization problem at each time interval. The aim is to determine a sequence of input moves such that the predicted response tracks a given setpoint. In this method, Np control actions are calculated at each time step, and only the first control action is implemented. Once a new measurement is available, the initial condition of the model is updated and a new sequence of control action is calculated. This strategy is known as receding strategy and it enables online tracking of unmeasured disturbances. (Camacho and Alba, 1999)

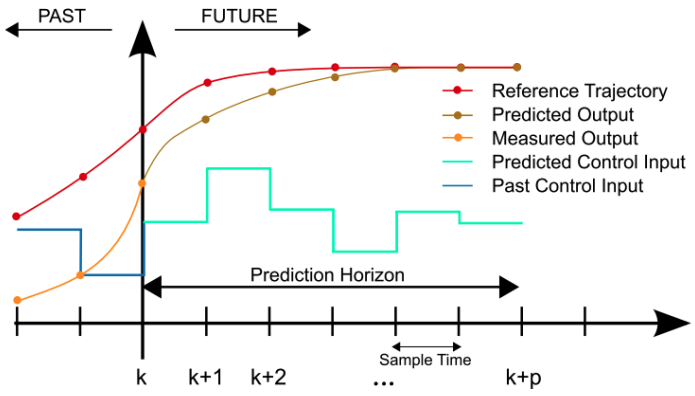

FIGURE 3.5: A sketch of the measured, predicted, and input variables in a model predictive control scheme (Commons, 2020)

A schematic representation of an MPC controller is shown in Figure 3.5. A moving prediction horizon as seen in the figure uses a window with a finite number of samples, from sample k to $k + Np$ (the prediction horizon, length of line in cyan(aqua)) to predict the future output of the plant (line in brown), and at each new time step this window is shifted forward. The longer the prediction horizon the better is the steady state performance of the controller. However, this comes with a

computational trade-off. The online computation will be impossible, if the sampling time of the controller is shorter than the computation time. Furthermore, the prediction horizon is divided into the input horizon and the output horizon. The input horizon is the time period where the controller can manipulate the inputs, while the time period between the input and out put horizon the input is kept constant. This is done to prevent too aggressive input moves.

The simplified structure of the MPC strategy is shown in a block diagram (Figure 3.6). As illustrated in the figure, the process model predicts the future outputs based on a sequence of inputs determined by the optimizer. Historical data in the form of past inputs and outputs, is returned to the MPC as feedback from the true system. Furthermore, the future errors are calculated such that the cost function is minimized and the system constraints not violated. (Camacho and Alba, 1999).

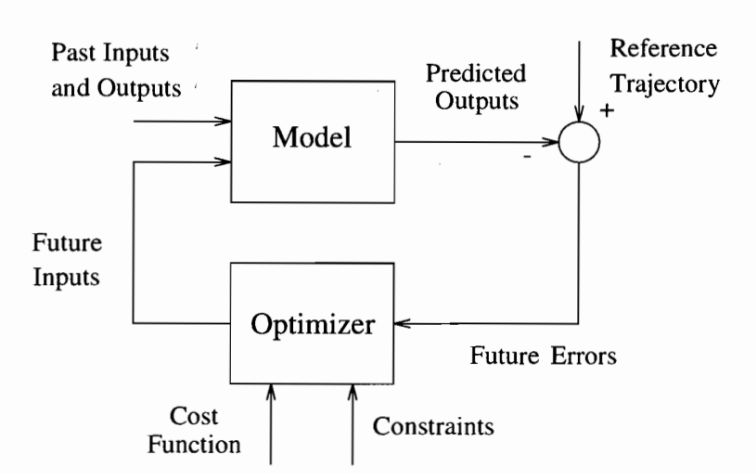

FIGURE 3.6: Basic structure of MPC (Camacho and Alba, 1999)

The control calculations in MPC are based on optimizing an objective function. There exists different types of objectives for the MPC such as economic control and setpoint control. In this thesis, an economic objective is considered, where the objective is to minimize an economic cost function. In the following section we will briefly explain economic model predictive control.

3.3.1 Economic model predictive control (EMPC)

In the Economic model predictive control (EMPC), the economic optimization and the control problems are solved simultaneously at each sampling time in one control layer. This control structure enables dynamic optimization over a moving horizon of process economic performance. In this control structure, process constraints are directly represented in the optimization problem. Furthermore, maximum freedom for optimization is achieved for better economic performance.

Even though EMPC is theoretically the optimal strategy, it has some practical drawbacks. Considering that EMPC must use a sufficiently large prediction horizon to account for a time-varying economic cost, the optimization problem may be challenging to solve fast enough to control the system in real-time. Additionally, compared to traditional hierarchical control strategies, EMPC requires more detailed and complex models to ensure that the constraints are satisfied, making the problem more difficult to solve efficiently. Finally, it can be challenging to balance the two different objectives of optimal economics and desired dynamic control performance in a single controller (Oliveira, 2016).

Formulation of Economic Model Predictive Control

In its basic formulation Economic Model Predictive Control looks at deterministic plants governed by finite-dimensional difference equations of the following type:

$$x(t+1) = f(x(t), u(t)) \tag{3.17}$$

where $x(t)$ is the state variable, and $u(t)$ is the control variable. The goal of the control design is to maximize profit or minimize costs, both during transient and steady-state operation. With $l(x, u)$ being the cost for operating the plant at state x, subject to input u, throughout a sampling interval. In more general scenarios, of course, both f and l might be time-dependent but, for the sake of simplicity, it is useful to consider situations in which costs and dynamics do not change significantly over the considered time window.

The stage cost is integrated over prediction horizon. Mathematically, the integrated cost is defined as follows:

$$J_l(\mathbf{x},\mathbf{u}) = \sum_{t=0}^{Np-1} l(x(t), u(t)) + \Psi_f(x(N)) \tag{3.18}$$

where Np is the prediction horizon. The finite sequences of indexed variables are denoted in bold fonts: $\mathbf{x}$ = [$x(0)$,$x(1)$, ...,$x(Np)$], $\mathbf{u}$ = [$u(0)$, $u(1)$, . . . , $u(Np-1)$]. Furthermore, the final weighting function $\Psi_f(.)$ is used to mitigate the effects of taking a short-sighted actions by providing some bound to the best achievable cost incurred over a very long horizon.

Nevertheless, the main difference between tracking MPC and Economic MPC, at the definition level, is in the stage cost, $l(x, u)$. Typically, this is taken to be a positive definite quadratic form of state and input, in the former, while it may be an arbitrary continuous function in the latter case. For instance, the following objective function is a typical choice in tracking MPC.

$$l(x,u) = (x - x_s)'Q * (x - x_s) + (u - u_s)' * R * (u - u_s) \tag{3.19}$$

where x_s and u_s are state-input pair for a different equilibrium state.

Therefore, the stage-cost l is designed in order to penalize deviations from the assigned setpoint, rather than optimizing the plants profits. A non-quadratic can also be used for this purpose. Its particularly common to take $l(x, u)$ to be positive definite with respect to the point x_s, u_s (Hedengren et al., 2014). Mathematically, it is described as follows:

$$l(x_s, u_s) < l(x, u) = 0 \tag{3.20}$$

This inequality doesn't have to hold for l in EMPC setups, even if (x_s, u_s) is chosen to be the best feasible equilibrium. This is demonstrated in Figure 3.7.

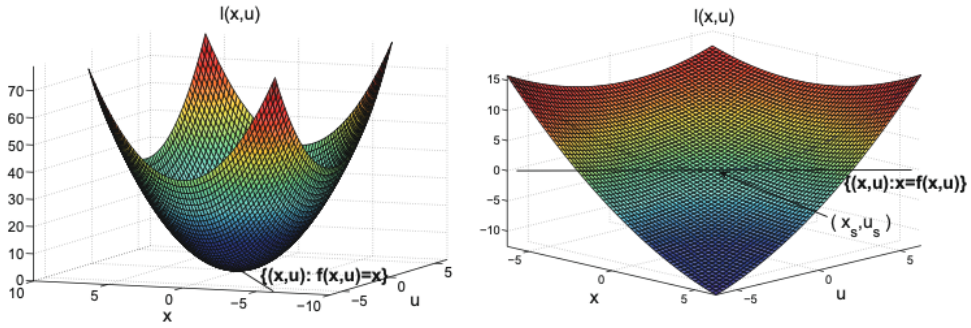

FIGURE 3.7: Stage cost $l(x, u)$ for Tracking vs Economic MPC (Levine et al., 2018)

The Economic model predictive control scheme in general form is given by Figure 3.8. As can be seen in both Figure 3.8 and 3.9, the tracking cost function is replaced by an economic objective function leaving the controller with no setpoint to track.

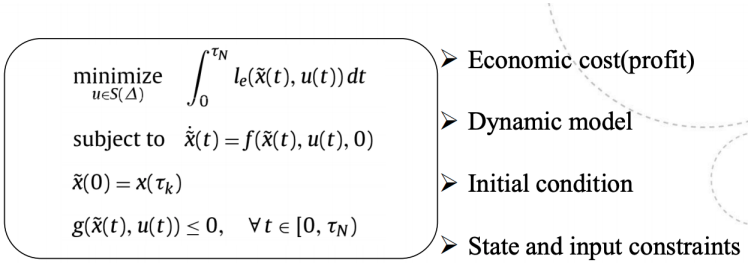

FIGURE 3.8: General form of an economic model predictive control

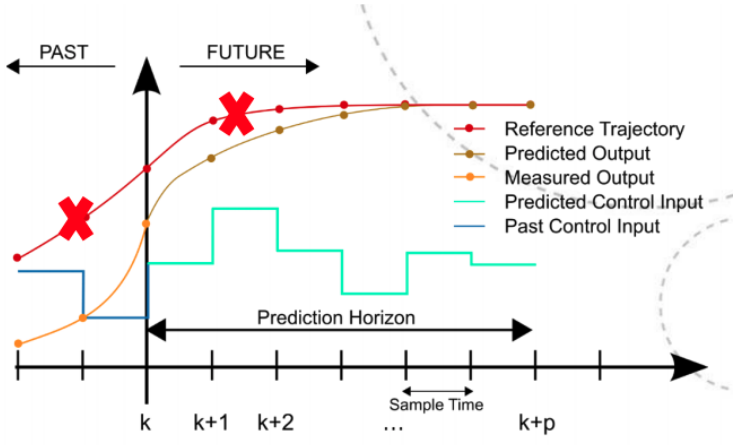

FIGURE 3.9: A sketch of the measured, predicted and input variables in an economic model predictive control

The type of systems we are dealing with in this thesis consists of both algebraic and differential equations, resulting in a differential algebraic equation system. These equations are used as constraints in the optimization problem. Furthermore, there are also inequality constraints which specify both the allowed values of the inputs and the changes in the inputs. The objective function and the model can

therefore be simplified as follows

$$min\Psi = \int_0^{Np} (-l(t) + \frac{1}{2}\Delta u(t)^T R_{\Delta u}\Delta u(t))\, dt \tag{3.21}$$

subject to

$$\dot{x}(t) = f(x(t), z(t), p(t), u(t)) \tag{3.22}$$
$$0 = g(x(t), z(t), p(t), u(t)) \tag{3.23}$$
$$0 \leq h(x(t), z(t), p(t), u(t)) \tag{3.24}$$
$$x(0) = \dot{x}_0 \tag{3.25}$$
$$z(0) = \dot{z}_0 \tag{3.26}$$

where Np is the prediction horizon, $l(t)$ is the cost, the second term describes a regularization term on the change in inputs, Δu. This forces the controller to minimize the change of inputs. R is a tuning parameter that weights the regularization term in the objective function. $\dot{x}$ describes the time derivative of differential states, x describes the differential states, z the algebraic states, p are the parameters of the system, u is the input of the system. The model differential equations and the algebraic equations are represented by f and g, respectively.

All MPC optimization problems have to always be transformed from a continous problem into a discrete problem, holding a finite number of variables. The discretized MPC can then be written in the form as a parameteric Nonlinear Program (pNLP). Once in this form, the problem can be passed to commercial solvers, such as IPOPT (Wächter and Biegler, 2006).

There exists several transcription algorithms that facilitates for efficient calculation of such problems. They are generally divided into two broad classes: shooting methods and simultaneous methods. They differ in how they enforce the constraint on the system's dynamics. Shooting methods use a simulation to explicitly enforce the system dynamics. Simultaneous methods enforce the dynamics at a series of points along a trajectory.

Shooting methods

Single-shooting

Single-shooting is the most straightforward method for transcribing an optimal control problem. Consider the problem of trying to score a goal in a basketball game. We have two decision variables (the firing angle and the amount of power we throw the ball with) and one constraint (trajectory passes through the target). This system's dynamics are simple projectile motions, and the cost function is the amount of power we exert. The single shooting method is similar to what a player might achieve with practice. The player guesses the angle and the amount of power and then throws the ball. If he throws the ball under the basketball hoop, he would perhaps increase the amount of energy he exerts on the next throw. By repeating this method, he would eventually score while using as little power as possible. Single shooting operates the same way if we replace the practices with simulations. In single-shooting, an arbitrary function such as piecewise linear is chosen to approximate the continuous control input. Generally, single shooting works well with simple problems. However, it usually fails when faced with complicated problems. The main reason for

this is the lack of a good approximation of the relationship between the decision variables and the objective and constraint function in the linear (or quadratic) model that the non-linear programming solvers use. (Kelly, 2015)

Multiple-shooting

We break up the trajectory into segments in multiple shooting method and solve each segment using the single shooting method. As a result, when the segments get shorter, the relationship between the decision variables and the objective function and constraints becomes more linear. Furthermore, the difference between the end of one segment and the start of the next is added to the problem as a constraint, increasing the number of decision variables and constraints in the optimization problem. Although it might seem that this would make the optimization problem harder, it turns out to make it easier. Even though, the Multiple shooting method results in a higher-dimensional non-linear problem, it is more sparse and linear than the problem that is solved in single shooting method.(Kelly, 2015)

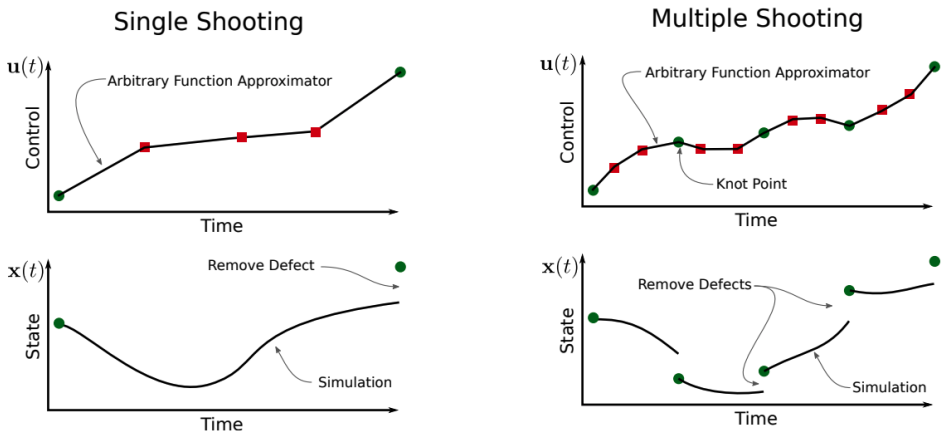

FIGURE 3.10: Single Shooting vs Multiple Shooting. In both methods the state trajectory is stored as the result of a simulation. Notice that multiple shooting is just like a series of single shooting methods, with additional constraint added to make the trajectory continuous. (Kelly, 2015)

Simultaneous methods

Orthogonal Collocation

Orthogonal collocation is a simultaneous method that uses orthogonal polynomials to approximate the state and control functions. Orthogonal polynomials have several useful properties. The key concept is that a polynomial can be represented over some finite domain by its value at a special set of grid points over that domain. When represented in this form, it is easy to do fast and accurate numerical interpolation, differentiation, and integration of the polynomial. (Kelly, 2015)

Orthogonal collocation on finite elements is based on dividing the prediction horizon in to finite elements. Each of these elements are then further divided into a given number of collocation points.

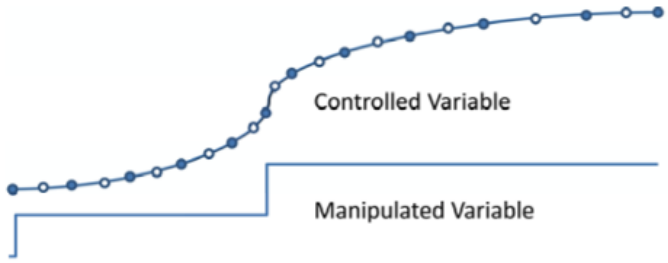

FIGURE 3.11: Dynamic equations are discretized over a time horizon and solved simultaneously. With one internal node for each segment, this example uses a 2nd order polynomial approximation for each step (Hedengren et al., 2014)

.

As shown in Figure 3.11 the dynamic equations are discretized over a time horizon and solved simultaneously. The solid nodes in the figure represents starting and ending point for local polynomial approximations that are stitched together over the time horizon (Hedengren et al., 2014). The main idea behind orthogonal collocation is to determine a weighting matrix M that relates the derivatives to non-derivative values over a time horizon at points 1,...,n as exhibited in equation 3.27. The initial value, x_0, is either a fixed initial condition or equal to the final point from the last interval.

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = M \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} - \begin{bmatrix} x_0 \\ x_0 \\ x_0 \end{bmatrix} \right) \tag{3.27}$$

The solution of the differential equations at discrete time points is approximated as a polynomial as follows:

$$x(t) = A + Bt + Ct^2 + Dt^3 \tag{3.28}$$

where t is the placement of the collocation points on the finite element. The derivative of x with respect to t is then given by:

$$\dot{x}(t) = Bt + 2Ct + 3Dt^2 \tag{3.29}$$

The collocation points used in this project is the Gauss-Radaue with numbers (0.1151, 0.6449, 1.0000). The time points are shifted to a reference time of zero and final time of 1. This enables the user to calculate the solutions without interpolation. For initial value problems, the coefficients A is equal to x_0, when the initial time is defined as zero. The coefficients B, C, and D are calculated by substituting equation 3.29 into 3.27.

$$M \begin{bmatrix} B + 2Ct_1 + 3Dt_1^2 \\ B + 2Ct_2 + 3Dt_2^2 \\ B + 2Ct_3 + 3Dt_3^2 \end{bmatrix} = \begin{bmatrix} A + Bt + Ct_1^2 + Dt_1^3 \\ A + Bt + Ct_2^2 + Dt_2^3 \\ A + Bt + Ct_3^2 + Dt_3^3 \end{bmatrix} - \begin{bmatrix} x_0 \\ x_0 \\ x_0 \end{bmatrix} \tag{3.30}$$

Rearraging and setting A = x_0 gives:

$$M \begin{bmatrix} 1 + 2t_1 + 3t_1^2 \\ 1 + 2t_2 + 3t_2^2 \\ 1 + 2t_3 + 3t_3^2 \end{bmatrix} \begin{bmatrix} B \\ C \\ D \end{bmatrix} = \begin{bmatrix} t_1 & t_1^2 & t_1^3 \\ t_2 & t_2^2 & t_2^3 \\ t_3 & t_3^2 & t_3^3 \end{bmatrix} \begin{bmatrix} B \\ C \\ D \end{bmatrix} \tag{3.31}$$

Finally, rearranging and solving for M gives the solution:

$$M = \begin{bmatrix} t_1 & t_1^2 & t_1^3 \\ t_2 & t_2^2 & t_2^3 \\ t_3 & t_3^2 & t_3^3 \end{bmatrix} \begin{bmatrix} 1 + 2t_1 + 3t_1^2 \\ 1 + 2t_2 + 3t_2^2 \\ 1 + 2t_3 + 3t_3^2 \end{bmatrix}^{-1} \tag{3.32}$$

For intervals that are not between 0 and 1, a scaling parameter h is introduced.

$$M \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_0 \\ x_0 \end{bmatrix} + hM \begin{bmatrix} f(x_1, z_1, p_1, u_1) \\ f(x_2, z_2, p_2, u_2) \\ f(x_3, z_3, p_3, u_3) \end{bmatrix} \tag{3.33}$$

The objective function and constraints given in equations 2.5 - 2.8, becomes a non-linear optimization problem which can be solved using orthogonal collocation. A constraint on the differential states are enforced within every collocation point to ensure that the trajectory for the differential states are continuous. The objective function is then evaluated at the end of every collocation point.

Even though, sequential methods are easier to implement, they may use unreasonable time to converge, especially problems with a large numbers of degrees of freedom. The simultaneous methods have generally computational advantage over sequential methods (Hedengren et al., 2014). Especially for control problems with many decision variables and a moderate number of state variables. In this paper, orthogonal collocation has been chosen as the preferred way of solving the dynamic algebraic equation system for its low computational cost and its accurate results.

Chapter 4

Methodology: Case study modeling

In this chapter, the system model, along with the choke valve degradation model is described. Furthermore, the sand production rate is assumed to be the main source of damage to the choke valves. Hence, sand production is also described at the end of this chapter.

4.1 Process Description

The system studied in this thesis is a subsea oil and gas production network containing three wells. An illustration of the gas-lifted subsea oil and gas production system is shown in Figure 4.1. The wells are connected to a common manifold. The combined flow from the three wells goes through a riser to a topside facility. When the reservoir pressure is not high enough to lift the fluids from the reservoir to the top facility, artificial methods are often needed. Among the artificial lifting methods, gas lift is a commonly used method in which a compressed gas is injected into fluid mix to reduce the mixture density (Krishnamoorthy, Foss, and Skogestad, 2016). As a consequence, the hydrostatic pressure drop in the well and the pressure at the well bottom decreases, increasing the flow from the reservoir.

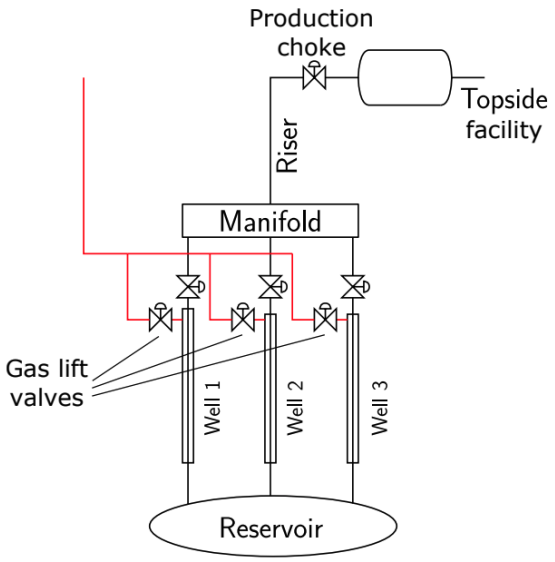

FIGURE 4.1: Illustration of the gas-lifted network (A. Verheyleweghen, 2018)

However, increased volume flow leads to increased equipment degradation, like the choke valves in the system. In particular, the erosion of choke valves used to shut down the wells when rerouting the production to different manifolds is severe. A well stream typically consists of a mix between oil, gas, water, sand, and other various particles. When all of these elements hit the internal surface of the choke valves for long periods, it causes erosion, shortening the useful life of the choke valves. Particularly, high production of sand has been known to cause considerable erosion damage in critical parts of the choke valves (A. Verheyleweghen, 2018). Additionally, if excessive gas is injected, the frictional pressure drop increases, which negatively affects the flow rate. At some point, the benefit of reduced hydrostatic pressure drop is overcome by the increase in the frictional drop (Krishnamoorthy, Foss, and Skogestad, 2016). Hence, oil wells have a desirable gas lift injection rate. Therefore, the objective is to find the optimal gas injection rate for each well so that the total oil production is maximized while satisfying the health constraints.

4.2 Gas-Lift Model

The model used to describe this gas-lifted well system is based on the work by (A. Verheyleweghen, 2018). A single gas-lift oil well is shown in Figure 4.2. It consists of an annulus volume into which lift gas is injected through the gas-lift choke. An injection valve located at the bottom of the annulus allows the gas to flow into the tubing.

The injected gas, as mentioned above, reduces the density of the produced fluid from the well, thereby reducing the pressure at the bottom of the well, often referred to as the downhole pressure (DHP). The lower the downhole pressure, the higher the rate of production from the reservoir.

A brief description of the production model for a gas lifted well model that is used in the optimization problem is given in four parts: (i) the mass balance of the different phases; (ii) the density models; (iii) the pressure models and (iv) the flow models are described under. It should be noted that all these models are for a single well, but they can be easily extended to the three wells.

The mass balance of each well is given by:

$$\dot{m}_{ga} = w_{gl} - w_{iv} \tag{4.1}$$

$$\dot{m}_{gt} = w_{iv} - w_{pg} + w_{rg} \tag{4.2}$$

$$\dot{m}_{ot} = w_{ro} - w_{po} \tag{4.3}$$

where the m_{ga} is the mass of the gas annulus, m_{gt} is the mass of the gas inside the well tubing, m_{ot} is the mass of oil in the well tubing, w_{gl} is the gas lift injection rate, w_{iv} is the gas flow from the annulus to the tubing, w_{pg} is the flow rate of produced gas, w_{rg} is the flow rate of gas from reservoir, w_{po} is the mass flow rate of produced oil, and w_{ro} is the flow rate of oil from the reservoir. Note that the dot notation indicates the derivative with respect to time.

These differential equations were set to be algebraic, due to the large time scale difference between the erosion rate and the differential equations, i.e, the equation over can be rewritten as:

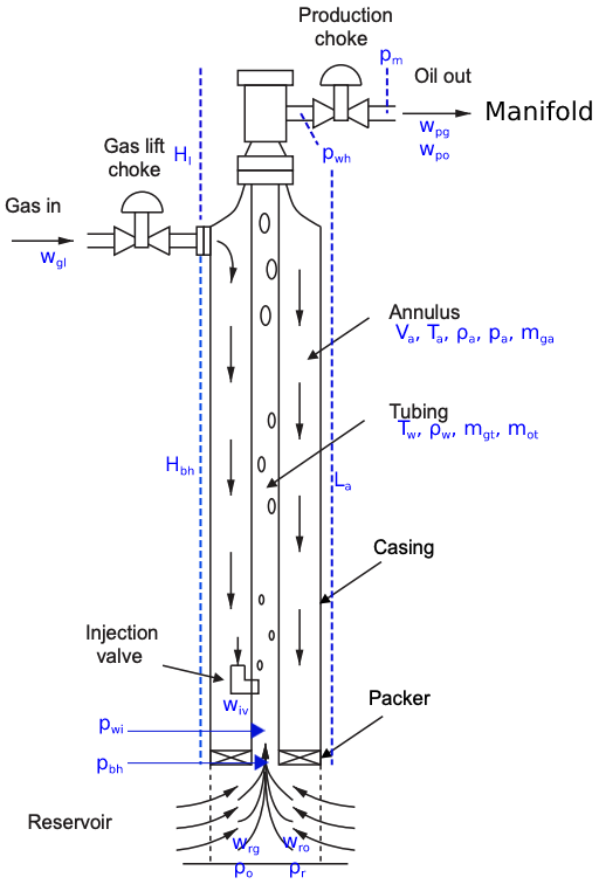

FIGURE 4.2: Illustration of a single gas-lift well adapted from (Eikrem, 2006)

$$0 = w_{gl} - w_{iv} \tag{4.4}$$

$$0 = w_{iv} - w_{pg} + w_{rg} \tag{4.5}$$

$$0 = w_{ro} - w_{po} \tag{4.6}$$

The model is based on a large time horizon to capture the erosion dynamics. In this thesis, the mass hold-ups are assumed to be constant and hence the dynamics are dictated by gradual choke erosion and slow decline in reservoir pressure. In such systems, the dynamics are usually dictated by the reservoir (Foss, Knudsen, and Grimstad, 2018). Therefore, this is not a very restrictive assumption.

The density model is given by:

$$\rho_a = \frac{M_w p_a}{T_a R} \tag{4.7}$$

$$\rho_w = \frac{m_{gt} + m_{ot} - \rho_0 L_r A_r}{L_w A_w} \tag{4.8}$$

where the ρ_a is the density of the gas in the annulus. ρ_w is the density if the fluid mixture in the tubing, M_w is the molecular weight of the gas, R is the gas constant, ρ_0 is the density of the oil in the reservoir, T_a is the annulus temperature, L_r is the length

of the well above the injection point and L_w is the length below the injection point, A_r is the cross-sectional area above the injection point and A_w is the cross-sectional area below the injection point.

The pressure model is given by:

$$p_a = (\frac{T_a R}{V_a M_w} + \frac{g L_a}{L_a A_a}) m_{ga} \tag{4.9}$$

$$p_{wh} = \frac{T_w R}{M_w} (\frac{m_{gt}}{L_w A_w + L_r A_r - \frac{m_{ot}}{\rho_0}}) \tag{4.10}$$

$$p_{wi} = p_{wh} + \frac{g}{A_w L_w} (m_{ot} + m_{gt} - \rho_0 L_r A_r) H_w \tag{4.11}$$

$$p_{bh} = p_{wi} + \rho_w g H_r \tag{4.12}$$

In this equations p_a is the annulus pressure, p_{wh} is wellhead pressure. p_{wi} is well injection point pressure, p_{bh} is the bottom hole pressure, L_a is the length of the annulus, A_a is the cross-sectional area of the annulus, T_w is the temperature of the well tubing,H_r is the vertical height of the well tubing below the injection point, H_w is the height of the well tubing above the injection point, g is the gravitational acceleration.

The flow model is given by:

$$w_{iv} = C_{iv} \sqrt{\rho_a max(0, p_{ai} - p_{wi})} \tag{4.13}$$

$$w_{iv} = C_{pc} \sqrt{\rho_w max(0. p_{wh} - p_m)} \tag{4.14}$$

$$w_{pg} = \frac{m_{ot}}{m_{gt} + m_{ot}} w_{pc} \tag{4.15}$$

$$w_{po} = \frac{m_{gt}}{m_{gt} + m_{ot}} w_{pc} \tag{4.16}$$

$$w_{ro} = PI(p_r - p_{bh}) \tag{4.17}$$

$$w_{rg} = GOR * w_{ro} \tag{4.18}$$

where w_{pc} is the total flow through the production choke, C_{iv} is the injection valve coefficient, C_{pc} is the production choke valve coefficient, PI is the reservoir productivity index, p_r is the reservoir pressure, p_m is the manifold pressure and GOR is the gas-oil ratio.

Gas to oil ratio (GOR) is among the essential parameters that is uncertain in the production network. In this thesis, the GOR is assumed to be known for the three wells for the nominal model.

4.3 Choke degradation model

Erosion of choke valves depends on a number of factors, such as physical properties of the fluid and how particles hit the internal surface of the choke valves. Furthermore, erosion rates are heavily dependent on the physical geometry of the choke valves in question, as this influences the flow patterns. Its therefore always a challenging task to develop a precise erosion model of a given choke valve, without

performing expensive computational fluid dynamics (CFD) simulations. The erosion model used in this project thesis is based on a choke erosion model in (DNV, 2015). The erosion model is given by Equation 4.19. For the rest of this thesis, this phenomenological model will be used for representing the actual erosion of the system. The data-driven models mentioned in Sections 3.2.1 and 3.2.2 will be used to approximate its behavior.

$$\dot{E} = \frac{KF(\alpha)U_p^n}{\rho_t A_t} * G * C_1 * GF * \dot{m}_{sand} * C_{unit} \tag{4.19}$$

where $\dot{E}$ is the erosion rate, while K, n, C_1, GF and C_{unit} are constants. $F(\alpha)$ is the ductility and is a function of particle impact angel α. U_p^n is the particle impact velocity. $\dot{m}_{sand}$ is the production rate and G is a parameter defined as

$$G = \frac{d_p * \beta * (1.88 * \log(A) - 6.04)}{D_{pipe}} \tag{4.20}$$

where d_p is the particle diameter, and D_p is the pipe diameter. β and A are dimensionless parameters

$$A = \frac{Re * \tan(\alpha)}{\beta} \tag{4.21}$$

$$\beta = \frac{\rho_p}{\rho_f} \tag{4.22}$$

where Re is the Reynolds numbers of the flow, ρ_p is the particle density and ρ_f is the fluid density.

Furthermore, the sand production, m_{sand} is assumed to have an exponential shape as follows:

$$m_{sand}(t) = m_{sand}(0) * e^{k*t} \tag{4.23}$$

where $m_{sand}(0)$ is the initial sand production rate and k is a parameter that affects the rate of exponential growth. In addition, $F(\alpha)$ is defined as:

$$F(\alpha) = 0.6 * [\sin(\alpha) + 7.2(\sin\alpha - \sin^2(\alpha))]^{0.6} * [1 - exp(-20\alpha)] \tag{4.24}$$

The particle impact angel α is the is given by:

$$\alpha = \arctan(\frac{1}{\sqrt{2r}}) \tag{4.25}$$

where r is the radius of the choke gallery. Lastly, The particle impact velocity U_p is calculated by:

$$U_p = \frac{3 * Q}{4 * A_g} = \frac{3 * Q}{8 * H * d} \tag{4.26}$$

where Q is the actual volumetric flow rate, A_g is the effective gallery area, H is the effective height of the gallery, and d is the space between the choke cage and the choke body, see Figure 4.3.

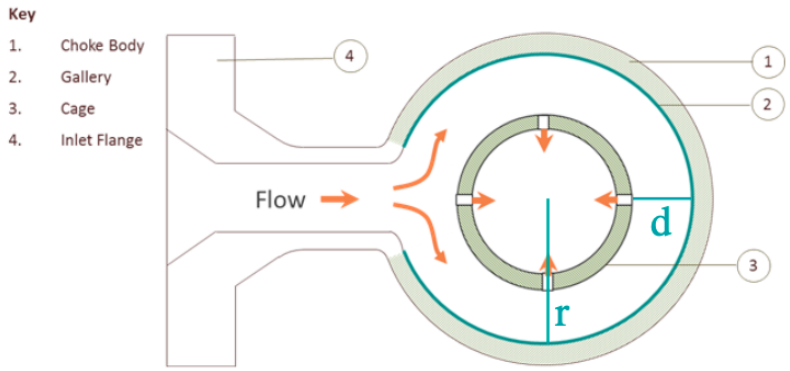

FIGURE 4.3: Choke gallery (DNV, 2015)

4.4 Sand production rate

The sand production rate m_{sand} in the actual oil wells is not constant. It changes as the field matures. Generally, the sand production rate increases significantly over the lifetime of the production well (Hettema et al., 2006). Much work has already been done on modeling the sand production rate (Pham, 2017). Nevertheless, this thesis aims at estimating the sand production profile as an exponential function, as seen in Equation 4.23. This sand profile is chosen because the sand rate is the key driving force for the erosion in the choke-valves, and we wanted to see how the health-aware controller would react to this extreme condition.

4.5 Simulation Data

This section presents a brief description of the simulated data sets used to provide the training data for the data-driven models used in the hybrid MPC. Simulations were carried out using the model described in Section 4.2. Note that the data generated in the following sections have already been trained and tested in (Jahren, 2020).

The data used of the data-driven model inside the MPC was generated with an exponential growing sand production rate. The code that was used for this purpose is shown in Appendix D. This simulation creates data for the three wells. In the actual operation, engineers adjust the gas injection rate according to the production plan. Thus, in the simulations we change the gas injection rate every 50 days. Furthermore, to generate a data set that is informative enough to describe the process, the input (gas lift rate) was randomly changed using a uniform distribution between $U_{min} + 1 \cdot (U_{max} - U_{min})$ to $U_{min} + 3 \cdot (U_{max} - U_{min})$. Additionally, a noise was also added in addition to random variation to incorporate the uncertainty in measurements. The erosion profile is shown in Figure 4.4.

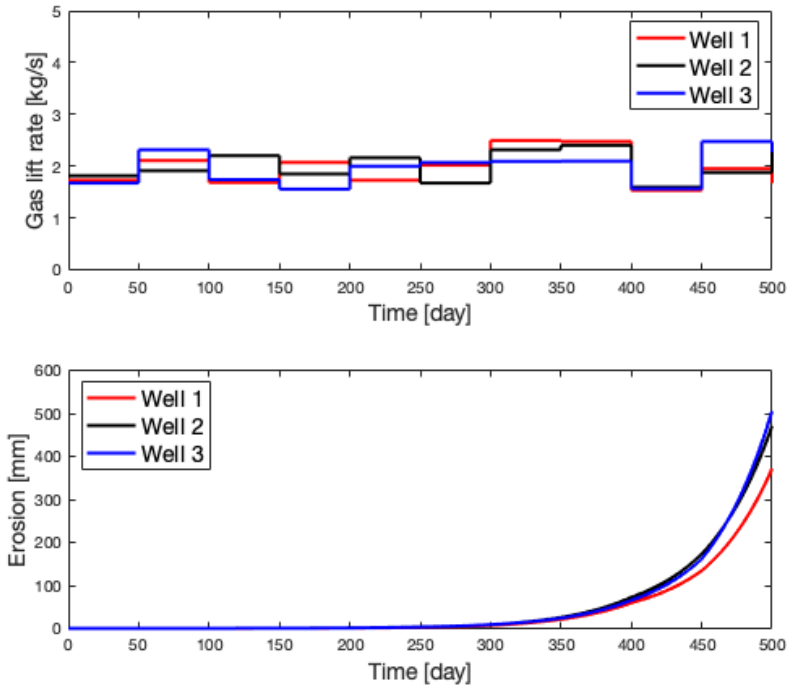

FIGURE 4.4: Gas-lift rate and erosion in mm plotted against time in days for 3 wells with exponential sand production profile

As can be seen in Figure 4.4, the erosion is increasing with an exponential shape as expected. The simulations are run over 500 days iteration. This simulation generated data for the control input (gas-lift rate), measurement values of parameters (predictors) and the response (erosion) of each well for 500 days. Since, the models used are trained on the normalized form of this data, the statistical data-driven models were trained using this training data.

Chapter 5

Results and discussion

This chapter presents and discusses the results of the case study, where the data-driven statistical methods discussed in chapter 3 are applied to the data simulated with the model presented in chapter 4. This chapter is organized as follows. First, we present the preliminary study related to the prediction capacity of the data-driven models, where we compare their accuracy with the phenomenological model. Note that this study is performed in an open-loop (i.e., the data-driven models are not embedded in the controller). The second part of this chapter is concerned with the performance of the data-driven models when integrated into the MPC (Health-aware controller). We use the perfect controller (complete feedback plus perfect model predictions, i.e., using the phenomenological model inside the HAC) as the baseline for the comparison. The controller's performance is compared in terms of financial results and conservativeness. The latter represents the loss in performance due to the plant-model mismatch added by the data-driven approximation of the equipment degradation.

5.1 Modelling erosion using Stepwise linear regression model

The models used in this project have already been trained and tested previously. Thus, we only include a brief description of these steps. A more detailed description can be found at (Jahren, 2020).

Prior to modelling, the simulated data consisting of 500 measured data points was pre-processed. This data points were chosen in order to represent a significant range of different operating conditions. The predictor variables shown in table 5.1 were subject to normalization as described in section 3.1.1 such that all the variables have a standard deviation and mean of 1 and 0 respectively. This was done due to the differing units of measurements in the data. Furthermore, the models were trained using the gradient of erosion measurements instead of direct erosion measurements, as such, erosion rate was used to predict the erosion, before it was transformed to cumulative erosion.

TABLE 5.1: The parameters used for the regression, with var 1-9 as predictor variables and var 10 as response variable

Predictor variable	Symbol	Description
var1	p_{ai}	Annulus pressure
var2	p_{wh}	Well head pressure
var3	w_{ro}	Well head oil production rate
var4	w_{rg}	Well head gas production rate
var5	p_{rh}	Riser head pressure
var6	p_m	Manifold pressure
var7	w_{to}	Riser head total oil production rate
var8	w_{tg}	Riser head total gas production rate
var9	w_{gl}	Gas lift rate
var10	ER	Erosion rate (Response)

The results of the stepwise linear regression model are compared to the phenomenological model in Figure 5.1. The comparison of the models is made with a constant sand production rate of 0.1. Furthermore, to generate a simulation that is informative enough to describe the comparison, the input (gas lift rate) was randomly changed every 10 days using a uniform distribution between $U_{min} + 1 \cdot (U_{max} - U_{min})$ to $U_{min} + 3 \cdot (U_{max} - U_{min})$. With U_{min} and U_{max} set to be 1.5 and 3.5 respectively. From the three wells, well two is observed to be the one with the highest plant-model mismatch. There seems to be a correlation between GOR and the plant-model mismatch with an error percentage of below 2.5%.

On the other hand, well 2 has the largest GOR and presents the most significant mismatch (error percentage of 4.7%). As mentioned in Section 4.2, GOR, which is among the critical parameters in the production network, is in reality uncertain. Therefore, such behavior indicates that these models should be used with caution in practical applications.

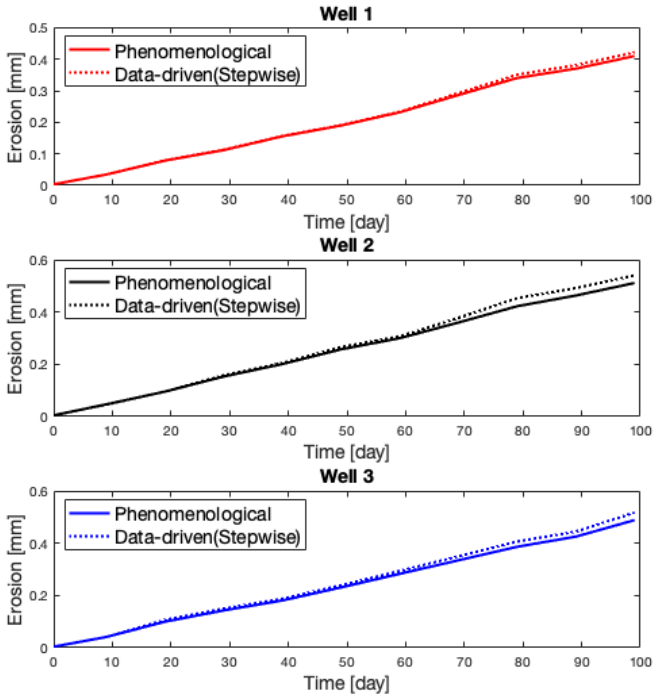

FIGURE 5.1: The phenomenological model vs the data-driven stepwise linear prognostic model

5.2 Modeling erosion using Neural network

The data sets used for the NN modeling are exactly the same as the one in the stepwise linear regression, see Table 5.2 . The trained neural network used in this thesis has two hidden layers with 20 nodes each. The sigmoid activation function is introduced to the model to facilitate flexibility in fitting non-linear functions. As seen in Figure 5.2, the NN regression performed well. Differently from the stepwise model, well one is observed to be the one with the highest plant-model mismatch. The prediction was more accurate in comparison to the stepwise linear model, with an error percentage of below 3% for the three wells. Despite the excellent prediction capacity, a lower percentage of errors can indicate that the NN is over-fitting the data, mainly because the data used in this preliminary analysis was inside the training data range. NN models are usually prone to over-fitting training data due to a large number of parameters present in the model (Tu, 1996).

We should note that those comparisons of accuracy with the phenomenological model are made for constant sand production. However, in the controllers, a more realistic exponential sand production profile is implemented. We can find the performance of those models for this case study at (Jahren, 2020). The author's data sets for this case study are almost identical to the one we did with a constant sand production rate. The only difference is adding one predictor, the sampled sand production rate, sampled at a set interval. The sampling rate of the sand production rate was found to have a significant impact on the model performances. This is reasonable as the sand production rate is an essential predictor.

TABLE 5.2: The parameters used for the regression, with var 1-9 as predictor variables and var 10 as response variable

Predictor variable	Symbol	Description
var1	p_{ai}	Annulus pressure
var2	p_{wh}	Well head pressure
var3	w_{ro}	Well head oil production rate
var4	w_{rg}	Well head gas production rate
var5	p_{rh}	Riser head pressure
var6	p_m	Manifold pressure
var7	w_{to}	Riser head total oil production rate
var8	w_{tg}	Riser head total gas production rate
var9	w_{gl}	Gas lift rate
var10	ER	Erosion rate (Response)

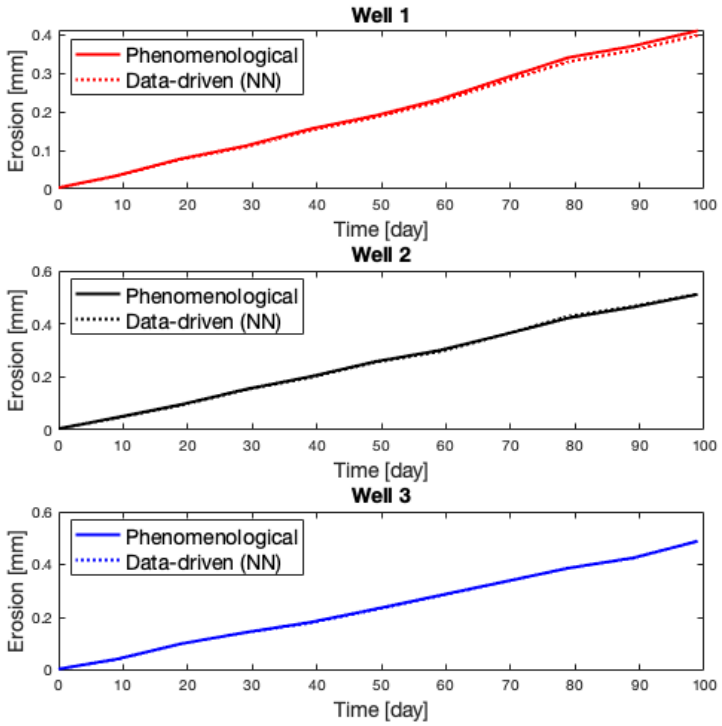

FIGURE 5.2: Plant-model mismatch (Comparison of the real true erosion vs the NN predicted erosion)

5.3 Performance of the Hybrid data-driven HAC controllers

The data-driven models developed in sections 5.1 and 5.2 were used for modeling the erosion rate in the controller, while the phenomenological model was used for the plant. Our approach is well presented in Figure 5.3. HAC with no plant-model mismatch (i.e. perfect erosion rate model) is also included for comparison. We tested the case studies with only prognostics first, and diagnostics was added afterward for the hybrid MPC controllers. This is done to include system feedback in the form of diagnostics in addition to prognostics to cope with the uncertainties in the system. In

other words, we will be testing the influence of having an approximation of the erosion model inside the controller (prognostics), and the influence of a not so accurate feedback (diagnostics) on the overall performance of the HAC.

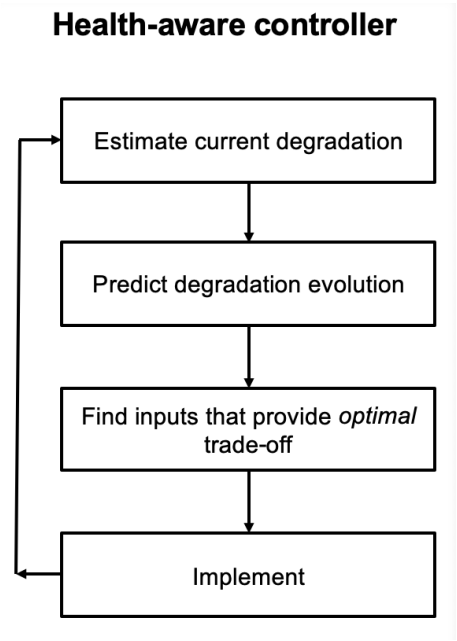

FIGURE 5.3: Block diagram of our approach **(Matias, 2021)**

The main objective of the MPC, as mentioned previously, is to maximize the total production of oil while keeping the erosion for each well below a threshold value of $2mm$ which was chosen arbitrarily . The differential equations shown in Section 4.3 together with the algebraic equations shown in Section 4.2 represent the system behavior. They will therefore be the constraints for the optimization problem. The problem for the MPC can then be written as follows:

$$min\Psi = \int_0^{Np} (\sum_{i=1}^{3} -w_{i,po}(t) + \frac{1}{2}\Delta u(t)^T R_{\Delta u}\Delta u(t) + \sum_{i=1}^{3} \rho_{i,s} s_i(t)\, dt \tag{5.1}$$

subject to:

$$\dot{E}(t) = f(E(t), z(t), u(t)) \qquad t \in [0, Np] \tag{5.2}$$
$$g(z(t), u(t)) = 0 \qquad t \in [0, Np] \tag{5.3}$$
$$-\Delta u_{max} \leq \Delta u(t) \leq \Delta u_{max} \qquad t \in [0, M] \tag{5.4}$$
$$u_{min} \leq u(t) \leq u_{max} \qquad t \in [0, Np] \tag{5.5}$$
$$\Delta u(t) = 0 \qquad t \in [M, Np] \tag{5.6}$$
$$0 \leq E(t) + s(t) \leq E_{max} \qquad t \in [0, Np] \tag{5.7}$$

In this optimization problem, $g(.)$ are the algebraic equation mentioned in section 4.3. u is the gas flow rate, w_{po} is the wellhead oil production rate, and z are the algebraic variables. Furthermore, the term $s_i(t)$ is introduced to the problem as a slack variable to give the controller flexibility to violate the constraint at a high cost for the objective function. The penalty is given by the weighting parameters $\rho_{i,s}$, which was arbitrarily set to be 1000000 for the three wells. The main objective of the slack variable is to ensure that the controller does not enter an infeasible region when the controller can no longer satisfy the constraint on the maximum allowed erosion.

In addition, M and Np represents the control and prediction horizon, respectively. Δu is a regularization term introduced to the problem if it is ill-conditioned and therefore might make the controller unstable. The weighting of the regularization, $R\Delta u$, is set to be:

$$R\Delta u = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.8}$$

Orthogonal collocation was used to solve the algebraic equations and was executed at every collocation point. A lower input bound of 0.4kg/s and an upper limit of $u_{max} = 2kg/s^{-1}$ for the gas lift rate were used. Furthermore, a conservative value of $\Delta u_{max} = 0.01kg/s^{-1}$ was used in the simulation. The controller was run over a prediction horizon of 100 days and control horizon of 70 day, where each time step is a day. We simulate the system until its breakdown i.e, until one of the wells reaches the maximum failure threshold value of $2mm$.

Additionally, an exponentially varying sand production rate is used to simulate the HAC controller. This sand profile is chosen because the sand rate is assumed to be the key driving force for the erosion in the choke-valves, and we wanted to see how the health-aware controller would react to this extreme condition. The flow rates and the pressures from the plant, see Table 5.1 are measured with an added noise to emulate a more realistic situation. Those values are then used to predict the current value of erosion (diagnostics) is then fed back to the HAC.

It is important to note that the simulations are run to failure, i.e. until one of the wells fails or crosses over the set failure threshold. Therefore, the HAC controllers in the case studies are expected to have different breakdown times and total oil production. We will, therefore, run the controllers to failure first to determine the breakdown time. After that, they will all be simulated for the same time period which is greater than their the breakdown time for illustration purposes only to make it easier for the reader to compare the performance of the controllers.

The case studies investigated in this thesis are summarized as follows:

- In the first case study, we will investigate HAC with no plant-model mismatch (i.e. perfect erosion rate model). Also, we assume that the erosion E(t) is measured at every sampling time

- In the second case study, we will investigate a hybrid data-driven HAC controller. The controller calculates the optimal inputs with the data-driven stepwise linear model developed, i.e., replace f in Equation 5.2 with the stepwise model. First, we assume that the hybrid HAC has perfect feedback, in which the erosion is measured. Then, to represent a more realistic situation, we study the inclusion of a diagnostics step in the HAC loop. The need for diagnostics steps comes from the fact that the current erosion is not measured in actual applications, and we need to estimate it based on the current plant measurements.

- In the third case study, we will investigate a hybrid data-driven HAC controller. The controller calculates the optimal inputs with the data-driven Neural network model developed, i.e., replace f in Equation 5.2 with the NN model. Similar to the previous case study, we tested the hybrid NN HAC with perfect information first and then with the diagnostics step.

TABLE 5.3: The MPC controllers studied for comparison in the three case studies.

Case study	Type of model in the controller	Type of model in the plant
1	Phenomenological	Phenomenological
2	Data-driven Stepwise linear model	Phenomenological
3	Data-driven Neural Network model	Phenomenological

5.3.1 Case study 1: HAC with no plant-model mismatch

We can see the simulation results for this case study in Figure 5.4. We start the simulation with an arbitrarily chosen input value of $[0.5, 0.5, 0.5]\frac{kg}{min}$. At the beginning of the simulation, the controller tries to maximize the total oil production by increasing the gas lift rate as fast as possible. However, the controller is constrained by the maximum allowed change of input, Δu. Around 150 days, it reaches u_{max}, where the gas lift can no longer be increased and it is kept until day 173 in the maximum value. This input sequence is determined by the controller because it predicts that the erosion will not reach the maximum threshold within the prediction horizon M, which can be clearly seen in the middle plot of Figure 5.4. At around day 173 (line in green), the controller again adjusts the gas lift rate of well two by decreasing, as the systems start to violate the constraint on the erosion.

In addition, the result showed that the erosion rate of the three wells within the prediction horizon was different, with well 2 and 3 being the wells exposed to the highest erosion. The main reason is the value of the chosen reservoir parameters, especially GOR. These values were specified such that well 2 and 3 have higher productivity indexes. With higher reservoir outflows, the choke valves of these wells tend to erode faster. This behavior was purposely chosen for testing the HAC performance. Due to different erosion rates, it needs to determine different input profiles for fast degrading wells. We see in the top plot of Figure 5.4 that indeed HAC starts decreasing the inputs from wells 2 and 3 before well 1, which was expected.

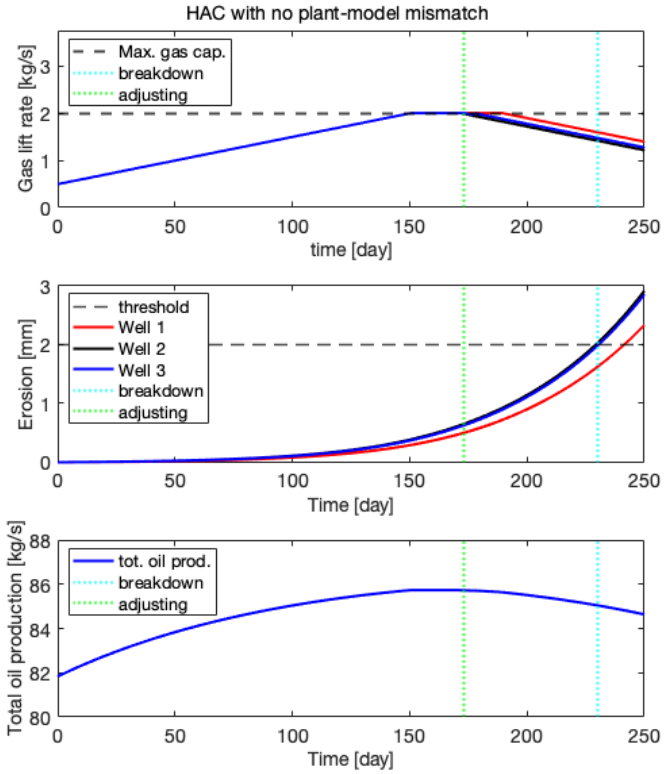

FIGURE 5.4: Result of the Health-aware controller with no plant model mismatch (Case study 1)

5.3.2 Case study 2: Hybrid HAC with stepwise linear model

The goal here is to analyze how the controller behaves when the plant and the controller model are different, i.e., with a plant-model mismatch. The simulation results of this case study are shown in Figure 5.5 and 5.6 with prognostics only and with prognostics and diagnostics, respectively.

Stepwise linear prognostic model and perfect erosion feedback

We observe a similar behavior as the HAC with no plant- model mismatch at the beginning of the simulation. The controller tries to maximize the total oil production by increasing the gas lift rate as fast as possible—followed by a constant gas lift injection period. However, in this case study, the controller decreases the gas lift rate after 180 days (line in green) which is seven days after the original MPC. This is due to the prediction error of the stepwise model. This controller does not predict the true system is about to break, so it keeps on producing more. This increase in production comes, therefore, at the cost of constraint violation. The system breaks after 229 days (line in cyan(aqua)) which is one day before the HAC with no plant-model mismatch and perfect feedback broke. As expected, the erosion rate of the three wells was also different here, subject to the same arguments in case 1.

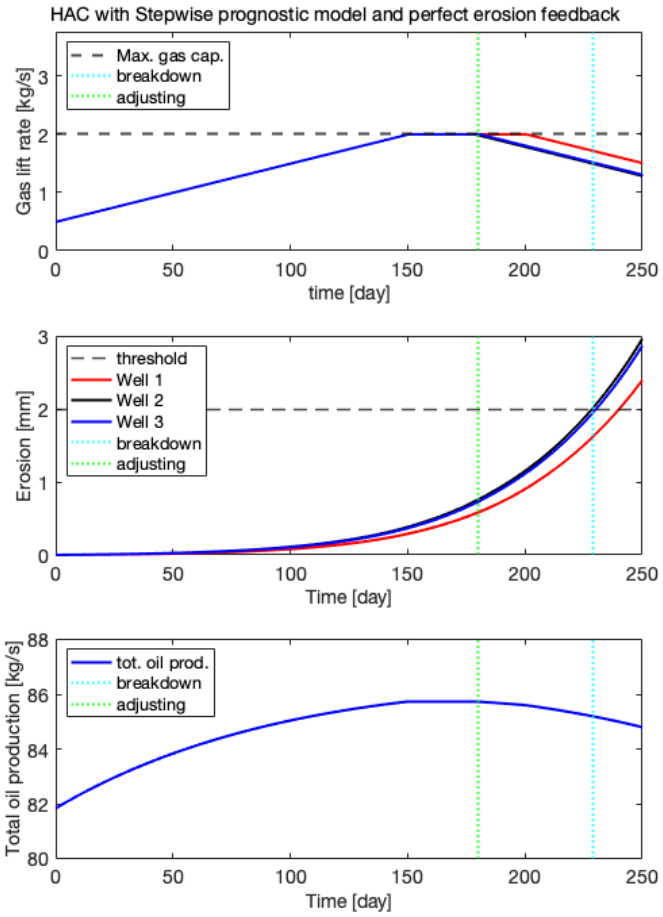

FIGURE 5.5: Result of the Health-aware controller with Data-Driven Stepwise linear prognostic model (Case study 2)

Stepwise linear prognostic model with diagnostics

In this case study, we added diagnostics to the control structure. A stepwise data-driven model was used to estimate the current state of the erosion and inside the controller for erosion prognostics. This controller showed similar results to the one with prognostic model only, see Figure 5.6. The breakdown time (the time at which the erosion of the wells exceeds the threshold) of this controller was the same as the one without diagnostics, 229 days (line in cyan (aqua)). Therefore, we conclude that the diagnostics step is accurate enough and does not influence the HAC overall behavior. This controller decreases the gas-lift rate after 173 days (line in green) like the HAC with no plant-model mismatch. It seems that the plant-model mismatch that is shown in Figure 5.7 is not significant enough to deteriorate the performance.

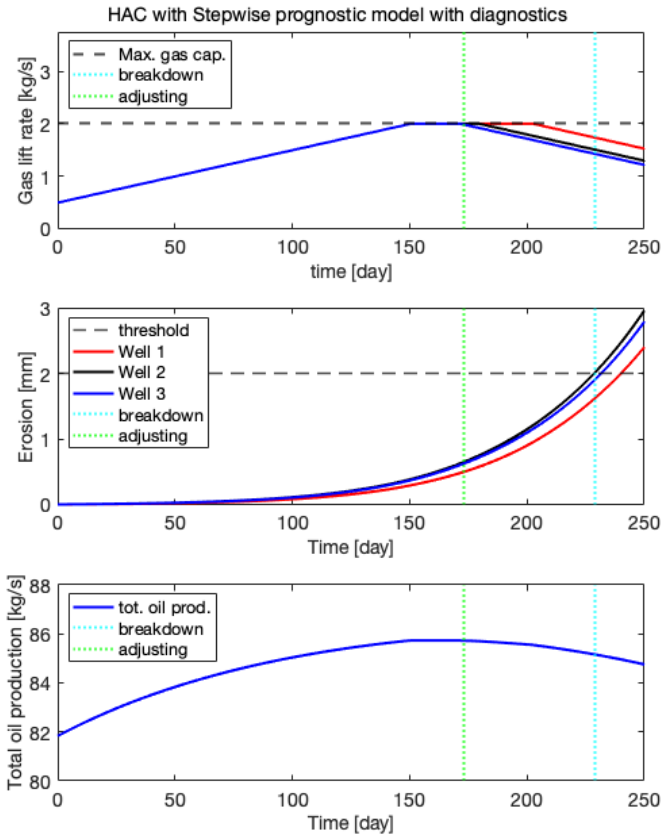

FIGURE 5.6: Result of the Health-aware controller with Data-Driven Stepwise linear prognostic and diagnostic model (Case study 2)

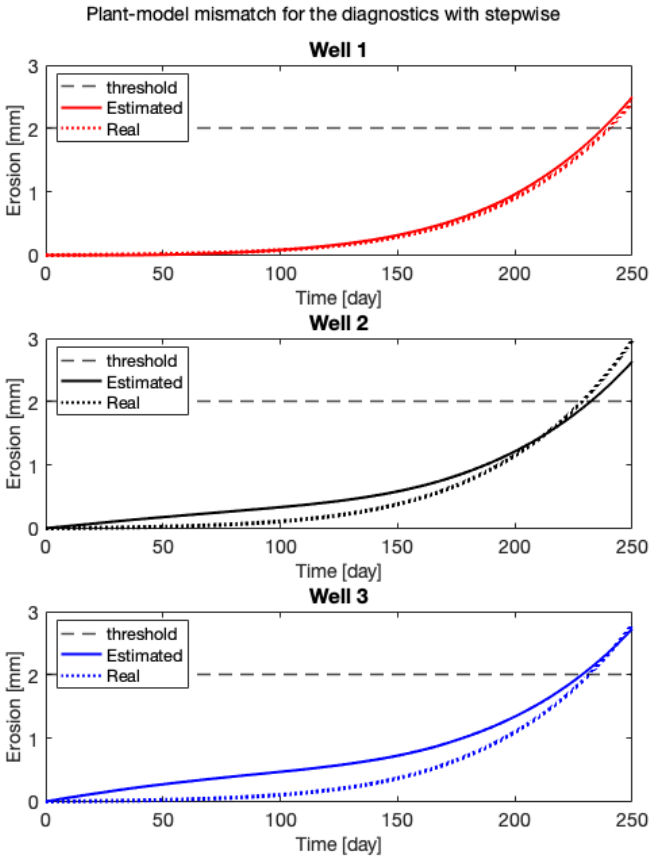

FIGURE 5.7: The plant-model mismatch of the stepwise linear diagnostic model in closed loop (real erosion vs the predicted erosion)

5.3.3 Case study 3: Hybrid MPC with Data-driven Neural Network

The NN is implemented as explained in Section 3.2.2. The simulation results of this case study are shown in Figure 5.8 and 5.9 with prognostics only and with prognostics and diagnostics, respectively.

Neural network prognostic model and perfect erosion feedback

The result for this case study is shown in Figure 5.8. Like all the other case studies, the controller is observed to increase the total oil production by increasing the gas lift rate as quickly as possible at the start of the simulation. This is followed by a constant gas lift period where the controller is constrained by the input upper bound. The controller then starts to decrease the production by reducing the gas lift rate after 213 days (line in green). This is a late reaction, and as a result, the system breaks at day 227 (line in cyan(aqua)), which is which is sooner than the other controllers. The main reason for this is the NN model's imperfect prediction, which leads to significant plant-model mismatch. Although the NN had the best performance in the case with a constant sand production rate, its performance degrades considerably as the sand production rate increases exponentially. As mentioned in Section 5.2, the NN models are prone to over-fitting training data and have poor extrapolating ability compared to stepwise linear regression models. They also have worse gradient prediction abilities in comparison to stepwise linear model. The controller therefore does not predict that the system is about to break. As a result, it keeps producing more, leading to a shorter breakdown time.

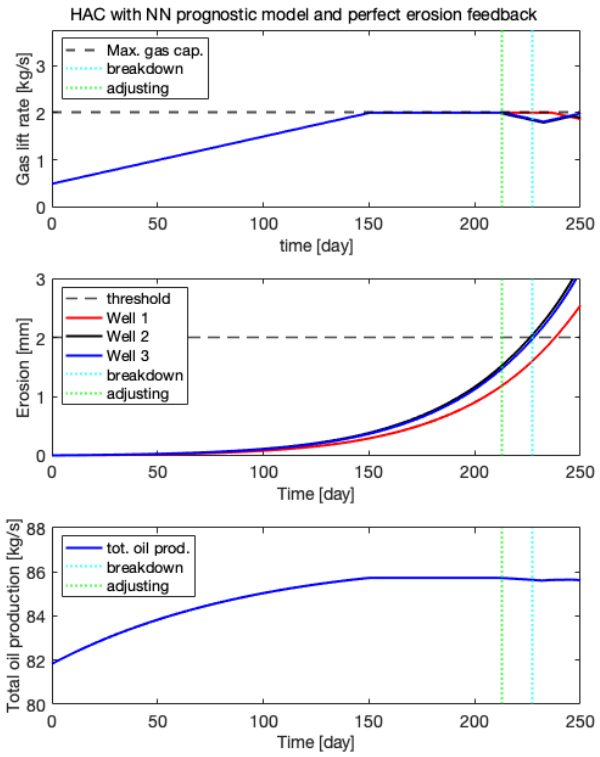

FIGURE 5.8: Result of the Health-aware controller with Data-Driven Neural-Network prognostic model (Case study 2)

Neural network prognostic model with diagnostics

The simulation results for this case study is shown in Figure 5.9. At the beginning of the simulation, the controller is observed to increase the total oil production by increasing the gas lift rate. However, after the upper limit on the gas lift injection is reached, the controller keeps the gas lift rate at this level until the breakdown time have been reached. This is due to the large plant-model mismatch in the NN model and the large deviation of the system feedback from the diagnostics, see Figure 5.10. In this particular case, it seems that the plant the NN models poor extrapolating ability and gradient prediction ability has made the controller to behave in this manner as its unable to accurately predict the erosion and see the system is about to break. And as a result the controller keeps increasing the production and ends up with the shortest breakdown time of 226 days (line in cyan(aqua)).

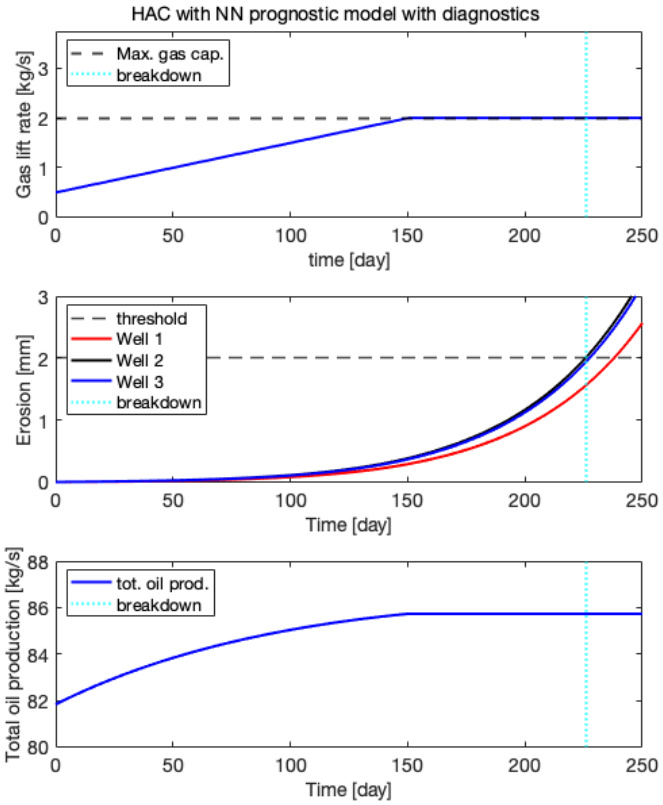

FIGURE 5.9: Result of the Health-aware controller with Data-Driven Neural-Network prognostic and diagnostic model (Case study 2)

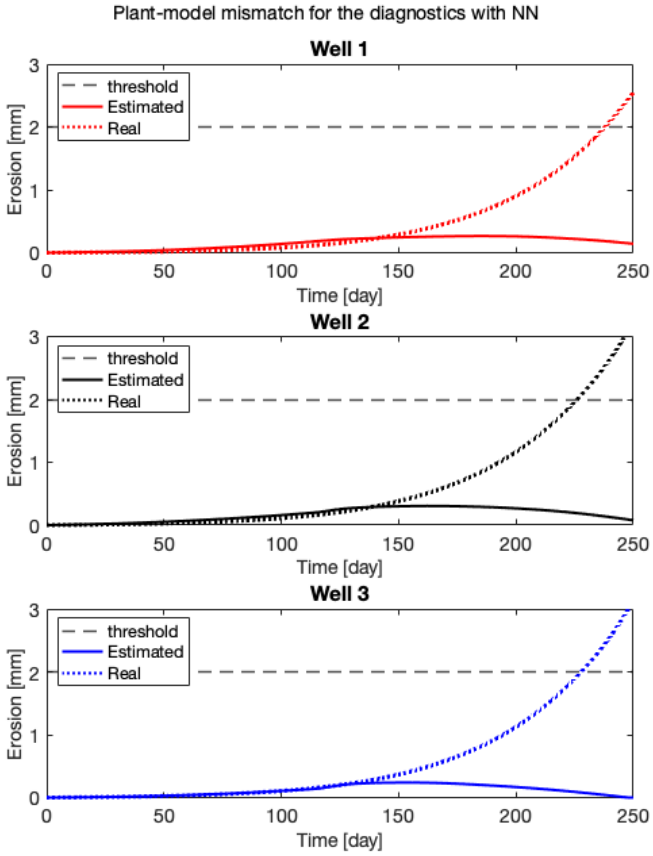

FIGURE 5.10: The plant-model mismatch of the NN diagnostics in closed loop (real erosion vs the predicted erosion)

5.4 Comparison of the performance of the five controllers

Generally, an increase in erosion was observed with an increase in the gas-lift rate in all the case studies. The greater the mass flow rate through the production valve, the greater the impact velocity of the sand particles hitting the surfaces of the choke valves, causing more erosion on the valves. Moreover, the effect of the well-specific reservoir parameter GOR, on the erosion of the three wells was observed. We note that erosion increases with a larger GOR value. As a result, well two was eroded faster during the simulation.

Furthermore, a trade-off between total oil production and constraint violation was observed in the simulation results. The increase in production during the simulation horizon in the hybrid data-driven HAC controllers comes at the cost of constraint violation. In order to analyze this trade-off we compared the five controllers in terms of the total oil we managed to produce, which is calculate by integrating the area under the total production curve (bottom plot of Figures 5.4, 5.5, 5.6, 5.8, and 5.9). Furthermore, the system breakdown time (the time at which the erosion rate of the wells exceeds the threshold value), the time the controller starts to adjust (decrease) the production, and the percentage of production from the perfect controller are also compared in Table 5.4. Comparison of the total oil production before breakdown for each case study is also shown in Figure 5.4.

TABLE 5.4: Metrics used for comparison of the Health-aware controllers studied in the three case studies

Case study	Breakdown time	Tot prod before breakdown	Adjusting prod. time	Relative production
1(HAC no-mismatch)	230 days	1.6834e+09 kg	173 days	
2 (SW prognostics)	229 days	1.6764e+09 kg	180 days	- 0.42 %
2 (SW prognostic + diagnostic)	229 days	1.6763e+09 kg	173 days	- 0.42 %
3 (NN prognostic)	227 days	1.6625e+09 kg	213 days	- 1.24 %
3 (NN Prognostic + diagnostic)	226 days	1.6551e+09 kg		- 1.68 %

The results clearly show that the inclusion of the data-driven models had a detrimental effect on production before system breakdown. The stepwise linear models resulted in a total production that is 0.42% lower than the perfect controller. The system was observed to break down one day before the perfect controller. Furthermore, surprisingly the inclusion of diagnostics seemed to have a negligible effect on the performance of the HAC with the perfect erosion feedback. This has led us to believe that the diagnostics step is accurate enough and does not influence the HAC overall behavior. This also makes sense as stepwise linear models have good gradient prediction abilities in comparison to NN models. It could also be because stepwise linear models have a better extrapolating ability due to the few system parameters that need to be tuned.

In contrast, the inclusion of the NN models resulted in a total production that is 1.24% lower than the perfect controller. The system was observed to break down after only 227 days, which is three days before the perfect controller breaks down. This could be due to the proneness of the NN- models to training data over-fitting. Furthermore, the inclusion of diagnostics as expected lowered the performance of the HAC controller. The controller with diagnostics broke down after only 226 days, which is one day before the HAC with perfect erosion feedback and four days before the perfect controller with no plant-model mismatch. Therefore, we can conclude that the diagnostic model with NN was not accurate enough to cope with the uncertainties in the system. We believe that these results could be due to the poor gradient prediction ability and extrapolating ability of the NN model compared to the stepwise linear model.

Moreover, the comparison of the total oil production for HAC controllers is shown in Table 5.4. As mentioned before, the controllers are run to failure, and as a result, they had different breakdown times. We can observe that the HAC controller with no plant-model mismatch had the highest total oil production as it run longer without breaking. In contrast, the hybrid NN data-driven HAC with diagnostics had the lowest total oil production as it had the shortest run time before breakdown. The poor extrapolating ability of the NN model made the prediction way off from the actual value. This led to constraint violation and high production of oil at the start of the simulation. Hence, the constraints were violated early, causing the controller to break down. While the HAC with no plant-model mismatch had the actual erosion value and was, therefore, the controller that respected the constraints the most, and as a result, the total production of oil was the highest as it was able to run for longer days.

All the parameter values and erosion threshold were chosen arbitrarily and not validated with actual information in this simulation. Therefore, the values here should not be considered for quantitative analysis. They qualitatively indicate how the performance of the HAC is tightly connected to the data-driven model used and with the quality of the plant feedback, as expected. And, the most important result,

that the complexity of the data-driven model is not directly associated with performance. A simpler model can yield better results.

Chapter 6

Conclusion

This paper studied the incorporation of data-driven diagnostics and prognostics into the health-aware controller loop. The controllers were tested in simulations of a gas-lifted subsea oil/gas production network subject to sand-caused choke erosion. We show that we can integrate the choke valve's data-driven prognostic and diagnostic models into the control framework to optimize production while keeping erosion levels of choke valves below critical levels. In this manner, health monitoring and prognostics are included in the production planning to find the optimal operational strategy.

The MPC framework studied in this thesis is implemented using stepwise linear and NN data-driven models. These models were used in diagnostics (erosion measurement - feedback to the control) and prognostics (erosion model - erosion evolution prediction). This is done to study the influence of having an approximation of the erosion model inside the controller. After assessing the impact of the model inside the controller, we included the diagnostics to study how much non-accurate feedback will affect the overall performance. For comparison, we also implemented a baseline controller, in which the erosion was measured, and its evolution model inside the controller was perfect. The hybrid nature of the MPC controllers was therefore at the center of the discussion.

The simulation results show that the hybrid Health aware controller is a possible alternative to solving the problem. However, the plant-model mismatch was found to have a detrimental effect on the performance of the HAC controllers. The NN prognostic model with diagnostic gave the most significant plant-model mismatch and constraint violation, while the stepwise linear prognostic model gave the least significant mismatch. The plant-model mismatch present due to the limitation of the data-driven method's prediction ability was the main reason for the constraint violation observed in the results. Furthermore, the imperfect feedback information obtained by the diagnostics was also another reason for the constraint violation observed in the data-driven HAC controllers.

In conclusion, it can be said that in this thesis, a qualitative analysis of the performance of a hybrid data-driven health-aware controller (HAC) was performed. The results show that the type of data-driven model used together with the quality of the plant feedback was detrimental to the the performance of the HAC. In addition, the results also showed that the complexity of data-driven models is not directly related to the performance of the HAC. Simple models can give better results.

6.1 Future work

For the future work, the robustness of the controllers should be analyzed using Monte-Carlo simulation in order to ensure that the controllers have good performance to adjust setpoints and decrease the effect of disturbances. Furthermore, to account for the plant-model mismatch in the controllers, some of the parameter in the systems can be considered to be stochastic. The objectives of this thesis was to showcase the possibility of Hybrid Health-aware controller that incorporates both data-driven and first principle models in the control framework, rather than providing results that correspond with real field data. We used therefore a simple choke erosion model. Its however recommended to develop accurate models for the particular equipment of interest before real-world implementation.

Bibliography

A. Verheyleweghen, J. Jäschke (2018). "Oil production optimization of several wells subject to choke degradation". In: *IFAC-PapersOnLine* 51, pp. 1–6.

Bernardino, Lucas Ferreira (2019). "HEALTH-AWARE CONTROL AND MODEL-BASED PROGNOSTICS OF A SUBSEA OIL AND GAS SEPARATION SYSTEM". In:

Camacho, Eduardo F and Carlos Bordons Alba (1999). *Model predictive control*. Springer-Verlag London Limited 1999.

Camacho, Eduardo F, Carlos Bordons, and Julio E Normey-Rico (2003). *Model predictive control springer, Berlin, 1999, ISBN 3540762418, 280 pages*.

Chai, Tianyou, S Joe Qin, and Hong Wang (2014). "Optimal operational control for complex industrial processes". In: *Annual Reviews in Control* 38.1, pp. 81–92.

Chen, James (December 23, 2020). *"Neural Network"*. https://www.investopedia.com/terms/n/neuralnetwork.asp#:~:text=A%20neural%20network%20is%20a,organic%20or%20artificial%20in%20nature.. [Online; accessed 1-May-2021].

Chizeck HJ, Willsky (1978). "Towards fault-tolerant optimal control". In:

Commons, Wikimedia (2020). *File:MPC scheme basic.svg — Wikimedia Commons, the free media repository*. [Online; accessed 27-April-2021]. URL: \url{https://commons.wikimedia.org/w/index.php?title=File:MPC_scheme_basic.svg&oldid=465567294}.

Darby, Mark L and Michael Nikolaou (2012). "MPC: Current practice and challenges". In: *Control Engineering Practice* 20.4, pp. 328–342.

DNV, GL (2015). "Managing sand production and erosion". In: *Recommended Practice DNVGL-RP-O501. DNV GL Company, Oslo, Norway*.

Eikrem, Gisle Otto (2006). "Stabilization of gas-lift wells by feedback control". PhD thesis. Citeseer.

Escobet, T, V Puig, and F Nejjari (2012). "Health aware control and model-based prognosis". In: *2012 20th Mediterranean Conference on Control & Automation (MED)*. IEEE, pp. 691–696.

Foss, Bjarne, Brage Rugstad Knudsen, and Bjarne Grimstad (2018). "Petroleum production optimization–a static or dynamic problem?" In: *Computers & Chemical Engineering* 114, pp. 245–253.

García, Salvador, Julián Luengo, and Francisco Herrera (2015). *Data preprocessing in data mining*. Vol. 72. Springer.

Glen, Stephanie (September 24, 2015). *"Stepwise Regression"*. [Online; accessed 23-December-2020]. URL: \url{https://www.statisticshowto.com/stepwise-regression/}.

Goswami, D Yogi and Frank Kreith (2015). *Energy efficiency and renewable energy handbook*. CRC Press.

Groer, Peter G (2000). "Analysis of time-to-failure with a Weibull model". In: *Proceedings of the maintenance and reliability conference*, pp. 59–01.

Harrison Kinsley, Daniel Kukieła (2020). *Neural Networks from scratch in Python*. Vol. 1. Kinsley Enterprizes.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

Hedengren, John D et al. (2014). "Nonlinear modeling, estimation and predictive control in APMonitor". In: *Computers & Chemical Engineering* 70, pp. 133–148.

Hettema, Marc H et al. (2006). "The relative importance of drawdown and depletion in sanding wells: Predictive models compared with data from the Statfjord Field". In: *SPE International Symposium and Exhibition on Formation Damage Control*. Society of Petroleum Engineers.

Jahren, Jan Henrik (2020). "Data-driven modelling of subsea equipment degradation using simulated and experimental case studies". In:

Jardim-Gonçalves, R. et al. (1996). "Application of stochastic modelling to support predictive maintenance for industrial environments". In: *1996 IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems (Cat. No.96CH35929)* 1, 117–122 vol.1.

Kelly, Matthew P (2015). "Transcription methods for trajectory optimization". In: *Tutorial, Cornell University, Feb*.

Krishnamoorthy, Dinesh, Bjarne Foss, and Sigurd Skogestad (2016). "Real-time optimization under uncertainty applied to a gas lifted well network". In: *Processes* 4.4, p. 52.

Levine, William S et al. (2018). "Handbook of model predictive control". In:

Lewis, FL (1986). *Optimal Estimation with an Introduction to Stochastic Control Theory, John Wiely & Sons*.

Ljung, Lennart (1999). "System identification". In: *Wiley encyclopedia of electrical and electronics engineering*, pp. 1–19.

Maciejowski, Jan Marian (2002). *Predictive control: with constraints*. Pearson education.

Matias, Jose (2021). "Sroject 3.8b - Systems Control: Validation of Methods for Optimizing Remaining Useful Life presentaiton". In: p. 5.

Oliveira, Vinicius de (2016). "Optimal operation strategies for dynamic processes under uncertainty". In:

Ott, R Lyman and Micheal T Longnecker (2015). *An introduction to statistical methods and data analysis*. Nelson Education.

Pham, Son Tung (2017). "Estimation of Sand Production Rate Using Geomechanical and Hydromechanical Models". In: *Advances in Materials Science and Engineering* 2017.

Qin, S Joe and Thomas A Badgwell (1997). "An overview of industrial model predictive control technology". In: *AIche symposium series*. Vol. 93. 316. New York, NY: American Institute of Chemical Engineers, 1971-c2002., pp. 232–256.

Rossiter, J Anthony (2003). *Model-based predictive control: a practical approach*. CRC press.

Sanchez, Hector et al. (2015). "Health-aware model predictive control of wind turbines using fatigue prognosis". In: *IFAC-PapersOnLine* 48.21, pp. 1363–1368.

Seborg, Dale E et al. (2010). *Process dynamics and control*. John Wiley & Sons.

Skogestad, Sigurd and Ian Postlethwaite (2007). *Multivariable feedback control: analysis and design*. Vol. 2. Citeseer.

Tu, Jack V (1996). "Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes". In: *Journal of clinical epidemiology* 49.11, pp. 1225–1231.

Vachtsevanos, George J and George J Vachtsevanos (2006). *Intelligent fault diagnosis and prognosis for engineering systems*. Vol. 456. Wiley Hoboken.

Verheyleweghen, Adriaen (2020). "Control Degrees of Freedom for Optimal Operation and Extending Remaining Useful Life". In:

Verheyleweghen, Adriaen, Julie Marie Gjøby, and Johannes Jäschke (2018). "Health-Aware Operation of a Subsea Compression System Subject to Degradation". In: *Computer Aided Chemical Engineering*. Vol. 43. Elsevier, pp. 1021–1026.

Wächter, Andreas and Lorenz T Biegler (2006). "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical programming* 106.1, pp. 25–57.

Werbos, Paul J (1988). "Generalization of backpropagation with application to a recurrent gas market model". In: *Neural networks* 1.4, pp. 339–356.

Wilson, Rick L and Ramesh Sharda (1994). "Bankruptcy prediction using neural networks". In: *Decision support systems* 11.5, pp. 545–557.

Appendix A

Calculation of dynamic viscosity of mixture

The dynamic viscosity of the mixture of interest is calculated under the assumption of ideal gas. The density of the gas, ρ_g is calculated as a function of the molar mass, pressure, and temperature as follows:

$$\rho_g = \frac{p_m * Mm_g}{R * T_w} \tag{A.1}$$

where p_m is the manifold pressure and T_r is riser temperature.

The mixed volumetric flow is then the sum of the liquid volumetric flow and the gas volumetric flow.

$$Q_m = Q_{pg} + Q_{po} = \frac{w_{pg}}{\rho_g} + \frac{w_{po}}{\rho_o} \tag{A.2}$$

where Q_m,Q_{pg}, and Q_{po} are the mixed, gas, and oil volumetric flows. The mixed volumetric flow can then be calculated by substituting equation A.1 in equation A.2 as follows:

$$Q_m = \frac{w_{po}}{\rho_o} + \frac{R * T_w * w_{pg}}{p_{wh} * Mm_g} \tag{A.3}$$

The dynamic viscosity, μ_m for the mixture is then given by:

$$\mu_m = \frac{\mu_o * V_o + \mu_g * V_g}{V_o + V_g} \tag{A.4}$$

where μ_o and μ_g are the dynamic viscosity of oil and gas respectively. V_o and V_g are the velocity of oil and gas and given as follows:

$$V_o = \frac{Q_{po}}{A_p} \tag{A.5}$$

$$V_g = \frac{Q_{pg}}{A_p} \tag{A.6}$$

where A_p is the piupe diameter. The dynamic viscosity for the mixture can then be rewritten as follow:

$$\mu_m = \frac{\mu_o * Q_{po}}{Q_{po} + Q_{pg}} + \frac{\mu_g * Q_{pg}}{Q_{po} + Q_{pg}} \tag{A.7}$$

Under the assumption of that the viscosity of the gas, μ_g is much lower than the viscosity of the liquid, μ_o and substituting equation A.3 in to equation A.7, the final expression for the dynamic viscosity of the mixture is simplified as follows:

$$\mu_m = \mu_o * \frac{\frac{w_{po}}{\rho_o}}{\frac{w_{po}}{\rho_o} + \frac{R*T_w*w_{pg}}{p_{wh}*Mm_g}} \tag{A.8}$$

Appendix B

Least Squares Estimator

The least squares estimation is a method in regression analysis that is used to minimize the sum of squared error of the residuals, defined as:

$$\boldsymbol{\epsilon} = \boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{B}} \tag{B.1}$$

where $\hat{\boldsymbol{B}}$ is a vector of estimate of $\boldsymbol{B}$, and $\boldsymbol{\epsilon}$ denotes vector of residuals.

The least squares estimator is then determined, using the sum of squares of the residuals as follows.

$$\begin{aligned} S(\hat{B}) = \sum \epsilon_i^2 = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon} = (\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{B}})^T(\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{B}}) \\ = \boldsymbol{Y}^T\boldsymbol{Y} - \boldsymbol{Y}^T\boldsymbol{X}\hat{\boldsymbol{B}} - \hat{\boldsymbol{B}}^T\boldsymbol{X}^T\boldsymbol{Y} + \hat{\boldsymbol{B}}^T\boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{B}} \end{aligned} \tag{B.2}$$

The minimum of $\boldsymbol{S}(\hat{\boldsymbol{B}})$ is determined by setting its derivatives equal to zero. With further simplifications the equation is simplified as follows:

$$\frac{\partial \boldsymbol{S}}{\partial \hat{\boldsymbol{B}}} = -\mathbf{2}\boldsymbol{X}^T\boldsymbol{Y} + \mathbf{2}\boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{B}} \tag{B.3}$$

The least squares estimator is then obtained by minimizing $\boldsymbol{S}(\hat{\boldsymbol{B}})$. Setting the derivatives equal to zero gives the normal equations

$$\boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{B}} = \boldsymbol{X}^T\boldsymbol{Y} \tag{B.4}$$

Rearenging and solving for $\hat{\boldsymbol{B}}$, we obtain

$$\hat{\boldsymbol{B}} = (\boldsymbol{X}^T\boldsymbol{X})^{-\mathbf{1}}\boldsymbol{X}^T\boldsymbol{Y} \tag{B.5}$$

Appendix C

Parameters used in the simulation

C.1 Parameters for choke degradation model

TABLE C.1: Parameters used for erosion calculation

Parameter	Unit	Description	value
ρ_p	kg m^3	Density of sand particles	$2.5 \cdot 10^{-3}$
C_1	-	Model geometry factor	1.25
C_{unit}	mm m^{-1}	Unit conversion factor	1000
D	m	Length from cage and choke body	0.1
d_p	m	Sand particle diameter	$2.5 \cdot 10^{-4}$
GF	-	Geometry factor	2.0
H	m	Height of Gallery	0.3
K	-	Material erosion constant	$2 \cdot 10^{-9}$
Mm_g	$gmol^{-1}$	Molar mass of gas	20
$\dot{m}_p$	kgs^{-1}	Sand rate	$50 \cdot 10^{-2}$
n	-	Velocity exponent	2.6
r	m	Radius of curvature	0.2

C.2 Parameters for gas-lift model

TABLE C.2: Parameters used in the gas lift model

Parameter	Unit	Description	value
μ_o	Pa s	Dynamic viscosity of oil	0.001
ρ_o	kgm^{-3}	Density of oil	$8 \cdot 10^2$
ρ_{ro}	kg m^{-3}	Density of oil in riser	$8 \cdot 10^2$
A_r	m^2	Cross-sectional area of riser	0.0115
C_{pr}	-	Valve constant for riser valve	0.01
D_r	m	Diameter of riser	0.121
H_r	m	Height of riser	500
L_r	m	Length of riser	500
n_w	-	Number of wells	3
p_s	bar	separator pressure	20
T	s	Sampling time	86400
T_r	K	Riser temperature	303

Appendix D

Matlab codes used for calculations

LISTING D.1: Code for calculating the parameters and the erosion using the well plant model

```
function [xk,zk] = WellPlantModel(dx0,z0,u0,m0,par)
    % based on a script by: Jose Otavio Matias

     addpath('/Users/SALI/Desktop/MATLAB/casadi-osx-matlabR2015a-v3.5.5')
     import casadi.*

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

    %properties
    %density of oil - dim:  nwells x 1
    rho_o = par.rho_o; %[kg/m3]
    %riser oil density
    rho_ro = par.rho_ro;%[kg/m3]
    %1cP oil viscosity
    mu_oil = par.mu_oil;% [Pa s]

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

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

    %annulus - [m]
    H_a = par.H_a;
    V_a = par.V_a; %[m3]

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

```
%injection valve characteristics  - dim:  nwells x 1
C_iv = par.C_iv;%[m2]
%production valve characteristics  - dim:  nwells x 1
C_pc = par.C_pc;%[m2]
%riser valve characteristics
C_pr = par.C_pr;%[m2]
% account for differences in the vapor and oil velocity
slip = par.slip_real;


%% For erosion model
% Sand
d_p = par.d_p; %[m] particle diameter
rho_p = par.rho_p; %[kg/m3] particle density
mdot_p = par.mdot_p; %[kg/s] sand rate

% Choke
K = par.K; %[-] material erosion constant
rho_t = par.rho_t; %[kg/m3] sensity CS
r = par.r; %[m] radius of curvature
D = par.D; %[m] Gap between body and cage
H = par.H; %[m] Height of gallery

% Constants
C_unit = par.C_unit; % Unit conversion factor: now in mm/s
C_1 = par.C_1; %[-] Model/geometry factor
n = par.n; %[-] Velocity coefficient
GF = par.GF; %[-] Geometry factor

% Precalculations of erosion in choke:
alpha = par.alpha;
F = par.F;
A_g = par.A_g; %[m2] Effective gallery area
G = 1; % THIS MUST BE CHANGED
ER_constant = par.ER_constant;

gma = d_p./D;


%% Differential states
%symbolic declaration
%erosion rate
ER = MX.sym('ER',n_w); % 1-3[s]

%% Algebraic states
%pressure - annulus
p_ai = MX.sym('p_ai',n_w);      % 1-3 [bar] (bar to Pa  = x10^5)
%pressure - well head
p_wh = MX.sym('p_wh',n_w);      % 4-6 [bar]
%pressure - injection point
p_wi = MX.sym('p_wi',n_w);      % 7-9 [bar]
%pressure - below injection point (bottom hole)
p_bh = MX.sym('p_bh',n_w);      % 10-12 [bar]
%density - annulus
rho_ai = MX.sym('rho_ai',n_w);  % 13-15 [100 kg/m3]
%mixture density in tubing
rho_m = MX.sym('rho_m',n_w);    % 16-18 [100 kg/m3]
%well injection flow rate
w_iv = MX.sym('w_iv',n_w);      % 19-21 [kg/s]
%wellhead total production rate
w_pc = MX.sym('w_pc',n_w);      % 22-24 [kg/s]
%wellhead gas production rate
w_pg = MX.sym('w_pg',n_w);      % 25-27 [kg/s]
```

```
%wellhead oil production rate
w_po = MX.sym('w_po',n_w);      % 28-30 [kg/s]
%oil rate from reservoir
w_ro = MX.sym('w_ro',n_w);      % 31-33 [kg/s]
%gas rate from reservoir
w_rg = MX.sym('w_rg',n_w);      %34 -36  [0.1 kg/s]
%riser head pressure
p_rh = MX.sym('p_rh',1);        % 37 [bar]
%mixture density in riser
rho_r =  MX.sym('rho_r',1);     % 38 [100 kg/s]
%manifold pressure
p_m = MX.sym('p_m',1);          % 39 [bar]
%riser head total production rate
w_pr = MX.sym('w_pr',1);        % 30 [kg/s]
%riser head total oil production rate
w_to = MX.sym('w_to',1);        % 41 [kg/s]
%riser head total gas production rate
w_tg = MX.sym('w_tg',1);        % 42 [kg/s]
%gas holdup @ annulus
m_ga = MX.sym('m_ga',n_w); % 43-45 [ton]
%gas holdup @ well
m_gt = MX.sym('m_gt',n_w); % 46-48 [ton]
%oil holdup @ well
m_ot = MX.sym('m_ot',n_w); % 49-51 [ton]
%gas holdup @ riser
m_gr = MX.sym('m_gr',1);   % 52 [ton]
%oil holdup @ riser
m_or = MX.sym('m_or',1);   % 53 [ton]
%particle impact velocity
V_p = MX.sym('V_p',n_w);   % 54-56
%mixed dynamic viscosity
mu_f = MX.sym('mu_f',n_w); % 57-59
%g1
g1 = MX.sym('g1',n_w);     % 60-62

%control input
%gas lift rate
w_gl = MX.sym('w_gl',n_w); %[kg/s]

%parameters
p_res = MX.sym('p_res',n_w);
%productivity index
PI = MX.sym('PI',n_w); %[kg s^-1 bar-1]
%GasOil ratio
GOR = MX.sym('GOR',n_w); %[kg/kg]
%Annulus temperature
T_a = MX.sym('T_a',n_w); %[oC]
%well temperature
T_w = MX.sym('T_w',n_w); %[oC]
%riser temperature
T_r = MX.sym('T_r',1); %[oC]
%separator pressure
p_s = MX.sym('p_s',1); %[bar]
%time transformation: CASADI integrates always from 0 to 1 and
%the USER does the time scaling with T.
T = MX.sym('T',1); %[s]

%sandrate rate
mdot_p = MX.sym('mdot_p',1); %

%% Modeling
%gas fraction (mass) of the well holdup - avoiding zero division
xGwH = (m_gt.*1e3./max(1e-3,(m_gt.*1e3+m_ot.*1e3)));
```

```
%gas fraction (mass) of the riser holdup
xGrH = (m_gr.*1e3./(m_gr.*1e3+m_or.*1e3));
xGw = slip.*xGwH./(1 + (slip-1).*xGwH);
xOw = 1 - xGw;
xGr = slip.*xGrH./(1 + (slip-1).*xGrH);
xOr = 1 - xGr;

% ==================================
%     Well model with/withou pressure loss
% ==================================
% algebraic equations (all symbolic) ...
%annulus pressure - %g = 9.81
f1 = -p_ai.*1e5 + ((R.*T_a./(V_a.*Mw) + 9.81.*H_a./V_a).*m_ga.*1e3)...
    + (Mw./(R.*T_a).*((R.*T_a./(V_a.*Mw) + 9.81.*H_a./V_a).*m_ga.*1e3)...
    ).*9.81.*H_a;
%well head pressure
f2 = -p_wh.*1e5 + ((R.*T_w./Mw).*(m_gt.*1e3./(L_w.*A_w + L_bh.*A_bh -...
m_ot.*1e3./rho_o))) - ((m_gt.*1e3+m_ot.*1e3 )./(L_w.*A_w)).*9.81.*H_w/2;
%well injection point pressure
f3 = -p_wi.*1e5 + (p_wh.*1e5 + 9.81./(A_w.*L_w).*max(0,(m_ot.*1e3+...
    m_gt.*1e3-rho_o.*L_bh.*A_bh)).*H_w) + (128.*mu_oil.*L_w.*w_pc./...
    (3.14.*D_w.^4.*((m_gt.*1e3 + m_ot.*1e3).*p_wh.*1e5.*Mw.*rho_o)./...
    (m_ot.*1e3.*p_wh.*1e5.*Mw + rho_o.*R.*T_w.*m_gt.*1e3)));
%bottom hole pressure
f4 = -p_bh.*1e5 + (p_wi.*1e5 + rho_o.*9.81.*H_bh + 128.*mu_oil.*...
    L_bh.*w_ro./(3.14.*D_bh.^4.*rho_o));
%gas density in annulus
f5 = -rho_ai.*1e2 +(Mw./(R.*T_a).*p_ai.*1e5);
%fluid mixture density in well
f6 = -rho_m.*1e2 + ((m_gt.*1e3 + m_ot.*1e3).*p_wh.*1e5.*Mw.*rho_o)./...
    (m_ot.*1e3.*p_wh.*1e5.*Mw + rho_o.*R.*T_w.*m_gt.*1e3);
%well injection flow rate
f7 = -w_iv + C_iv.*sqrt(rho_ai.*1e2.*max(0,(p_ai.*1e5 - p_wi.*1e5)));
%wellhead prodution rate
f8 = -w_pc + 1.*C_pc.*sqrt(rho_m.*1e2.*max(0,(p_wh.*1e5 - p_m.*1e5)));
%wellhead gas production rate
f9 = -w_pg + xGw.*w_pc;
%wellhead oil prodution rate
f10 = -w_po + xOw.*w_pc;
%oil from reservoir flowrate
f11 = -w_ro + PI.*1e-6.*(p_res.*1e5 - p_bh.*1e5);
%gas from reservoir production rate
f12 = -w_rg.*1e-1 + GOR.*w_ro;
%riser head pressure
f13 = -p_rh.*1e5 + ((R.*T_r./Mw).*(m_gr.*1e3./(L_r.*A_r))) - ...
    ((m_gr.*1e3+m_or.*1e3 )./(L_r.*A_r)).*9.81.*H_r/2;
%riser density
f14 = -rho_r.*1e2 + ((m_gr.*1e3 + m_or.*1e3).*p_rh.*1e5.*Mw.*rho_ro)./...
    (m_or.*1e3.*p_rh.*1e5.*Mw + rho_ro.*R.*T_r.*m_gr.*1e3);
%manifold pressure
f15 = -p_m.*1e5 + (p_rh.*1e5 + 9.81./(A_r.*L_r).*(m_or.*1e3+m_gr.*1e3)...
    .*H_r) + (128.*mu_oil.*L_r.*w_pr./(3.14.*D_r.^4.*((m_gr.*1e3 +...
    m_or.*1e3).*p_rh.*1e5.*Mw.*rho_ro)./(m_or.*1e3.*p_rh.*1e5.*Mw + ...
    rho_ro.*R.*T_r.*m_gr.*1e3)));
%total production rate of well
f16 = -w_pr + 1.*C_pr.*sqrt(rho_r.*1e2.*(p_rh.*1e5 - p_s.*1e5));
%oil total production rate
f17 = -w_to + xOr.*w_pr;
%gas total production rate
f18 = -w_tg + xGr.*w_pr;
% setting differential equations as algebraic equations since the
% dynamics of ER is on a much larger time scale
f19 = (w_gl - w_iv).*1e-3;
```

```
    f20 = (w_iv + w_rg.*1e-1 - w_pg).*1e-3;
    f21 = (w_ro - w_po).*1e-3;
    f22 = (sum(w_pg) - w_tg).*1e-3 ;
    f23 = (sum(w_po) - w_to).*1e-3 ;
    f24 = - V_p + 3/(4*A_g)*(w_po/rho_o + R*T_w.*w_pg./(p_wh.*10^5*Mw));
    f25 = - mu_f + mu_oil.*(w_po/rho_o)./(w_po./rho_o + R.*T_w.*w_pg./...
        (p_wh.*10^5*Mw));
    % Assuming that gamma < 0 (checked in main)
    f26 = -g1 + gma/0.1;

    % differential equations - (all symbolic) - [ton]
    % Erosion rate
    ER_constant = par.K*par.F*par.C_1*par.GF*mdot_p*par.C_unit/(par.rho_t...
        *par.A_t);
    df1 =  ER_constant.*g1.*(V_p).^n;

    % Form the DAE system
    diff = vertcat(df1);
    alg = vertcat(f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15,f16,...
        f17,f18,f19,f20,f21,f22,f23,f24,f25,f26);

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

    diff = substitute(diff,p_res,par.p_res);
    diff = substitute(diff,T_w,par.T_w);

    % concatenate the differential and algebraic states
    x_var = vertcat(ER);
    z_var = vertcat(p_ai,p_wh,p_wi,p_bh,rho_ai,rho_m,w_iv,w_pc,w_pg,w_po,...
        w_ro,w_rg,p_rh,rho_r,p_m,...
        w_pr,w_to,w_tg,m_ga,m_gt,m_ot,m_gr,m_or,V_p, mu_f,g1);
    p_var = vertcat(w_gl,mdot_p,GOR,PI,T);

    %end modeling
    %% Casadi commands
    %declaring function in standard DAE form (scaled time)
    dae = struct('x',x_var,'z',z_var,'p',p_var,'ode',T*diff,'alg',alg);

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

    %assuming inputs as symbolic in order to obtain
    %the gradients symbolically
    theta = MX.sym('theta',3);

    %integration results
    Fend = F('x0',dx0,'z0',z0,'p',[u0;m0;par.GOR;par.PI;par.T]);
    %extracting the results (from symbolic to numerical)
    xk = full(Fend.xf);
    zk = full(Fend.zf);
    xf = Fend.xf;
end
```

LISTING D.2: Code for generating the artificial training data

```
clear
close all
clc

%% noise --> For reproducibility
% This is choosing a seed for generating random numbers
rng(1)

%%

%% Initializing the table to store multiple time series:

nTimeseries = 1;
dataSz = [nTimeseries 10];
varNames = {'Num','uArray', 'yMeas','erosionArray', 'H', 'x0', 'xk',...
    'yk', 'z0', 'zk'};
varTypes = {'double', 'cell', 'cell', 'cell', 'cell', 'cell', 'cell',...
    'cell', 'cell', 'cell'};
Data = table('Size', dataSz, 'VariableNames', varNames, 'VariableTypes',...
    varTypes);


%% Model initialization
sandArray = sandproductionrate(0.01,500,'log',0.02);
%% [sandArray,sandArrayNoise,stepSandArray] =
%% sandproductionrate(0.01,500,'log',0.015);
for i_timeseries = 1:nTimeseries



% initial condition (pre-computed)
[x0,z0,u0] = InitialConditionGasLift_P;

%% model parameters

%par = ParametersGasLift(1);
par = ParametersGasLift(1,sandArray); % For varying sand production rate

%states to measurement mapping function
H = zeros(16,length(z0));
%pai - annulus pressure, well 1-3
H(1,1) = 1;
H(2,2) = 1;
H(3,3) = 1;
%pwh - well head pressure, well+ 1-3
H(4,4) = 1;
H(5,5) = 1;
H(6,6) = 1;
%wro - wellhead gas production rate, well 1-3
H(7,25) = 1;
H(8,26) = 1;
H(9,27) = 1;
%wrg - wellhead oil production rate, well 1-3
H(10,28) = 1;
H(11,29) = 1;
H(12,30) = 1;
%prh - riser head pressure
H(13,37) = 1;
%pm - manifold pressure
H(14,39) = 1;
%wto - riser head total oil production rate
H(15,41) = 1;
```

```
%wtg - riser head total gas production rate
H(16,42) = 1;




%% Simulation parameters

% Number of simulation steps
nSim = 500;  % time_total = 3600*24*500; %[s]



[dx0,z0,u0] = InitialConditionGasLift_P;

%sampling time /control interval /1 simulation iteration time
par.T = 3600*24; % [s]

%% Simulation initialization
xk = x0;
zk = z0;
uk = u0;
yk = H*z0;


%%
fprintf('Time series number: >>> %0.0f \n',i_timeseries)
%creating random array for the inputs
uArray = [];
%bounds on the inputs
uMin = 1.5;
uMax = 2.5;

for t = 0:nSim
    if rem(t,50) == 0 %every 50 days we change the inputs
        uk = (uMax - uMin).*rand(3,1) + uMin;
    end
    uArray = [uArray, uk];
end

% measurements (for plotting)
yMeas = yk;
erosionArray = xk;


for u = 1:nSim
    sarray = sandArray(u);
    %fprintf('     iteration >>> %0.0f \n',t)
    % integrating the system
    [xk,zk] = WellPlantModel(xk,zk,uArray(:,u),par);
    %[xk,zk] = WellplantModelNN(xk,zk,uArray(:,u),par,sarray);
    par = ParametersGasLift(u,sandArray);  % Varying sand production
    par.T = 3600*24;
    % Adding noise to the measurements
    yMeas = [yMeas, H*zk + par.scale.*randn(length(yk),1)];
    erosionArray = [erosionArray, xk];
end
Data(i_timeseries, :) = {i_timeseries, uArray, yMeas, erosionArray, H,...
    x0, xk, yk, z0, zk};


end
```

```
%% saving the data in a mat file
filename = 'datamatrix_' + string(nTimeseries)+'.mat';
save(filename, 'Data')
%% saving the matrix in a mat file
%filename1 = 'hmatrix' + '.mat';
%save(filename1,'H')

%% Plotting
figure(1)


time = 0:1:nSim; %[days]


% System inputs
subplot(2,1,1)
    stairs(time,uArray(1,:),'LineWidth',2);
    hold on
    stairs(time,uArray(2,:),'LineWidth',2);
    stairs(time,uArray(3,:),'LineWidth',2);
    legend('Well 1','Well 2','Well 3');

    ylim([0,5]);
    xlabel('Time [day]');
    ylabel('Gas lift rate [kg/s]');

% erosion
subplot(2,1,2);
    plot(time,transpose(Data.erosionArray{1,1}(1,:)),'LineWidth',2);
    hold on
    plot(time,transpose(Data.erosionArray{1,1}(2,:)),'LineWidth',2);
    plot(time,transpose(Data.erosionArray{1,1}(3,:)),'LineWidth',2);
    %ylim([0,5.2]);
    legend('Well 1','Well 2','Well 3');
    legend('Location','northwest');
    xlabel('Time [day]');
    ylabel('Erosion [mm]');

figure(2)

% Pressure
subplot(2,1,1)
    plot(time,yMeas(4,:),'LineWidth',1.5);
    hold on
    plot(time,yMeas(5,:),'LineWidth',1.5);
    plot(time,yMeas(6,:),'LineWidth',1.5);
    axis([0 500 45 60]);
    legend('Well 1','Well 2','Well 3');

    %ylim([0,2.2]);
    xlabel('Time [day]');
    ylabel('Well head pressure [bar]');

% erosion
subplot(2,1,2);
    plot(time,yMeas(15,:),'LineWidth',2); %oil
    %plot(time,yMeas(14,:)); %gas
    axis([0 500 50 100]);

    xlabel('Time [day]');
    ylabel('Flowrate [kg/s]');
```

LISTING D.3: ModelSandArray: Code for generating the sand production rate

```
function [sandArray,sandArrayNoise,stepSandArray] = sandproductionrate....
    (initial_mdot,days,type,k)

if (type == 'exp')
sandArray = [initial_mdot];

    for n = 1:days
     %sandArray = [sandArray initial_mdot*(0.5+exp(k*n))];
     sandArray = [sandArray initial_mdot*(exp(k*n))];
    end

    stepSandArray = [];
    step = 1;

    for n = 1:days
       stepSandArray = [stepSandArray sandArray(step)];
       if mod(n,30) == 0
           step = n;

       end

    end
end

if (type == 'log')
sandArray = (10*initial_mdot)/(1+exp(-k*(0-days/2)));

    for n = 1:days
     newVal = (10*initial_mdot)/(1+exp(-k*(n-days/2)));
     sandArray = [sandArray newVal];
    end

    stepSandArray = [];
    step = 1;

    for n = 1:days
       stepSandArray = [stepSandArray sandArray(step)];
       if mod(n,30) == 0
           step = n;

       end

    end
end

if (type == 'cst')
    sandArray = initial_mdot*ones(1,days+1);
end

sandArrayNoise = sandArray + 0.1 * rand(1, length(sandArray));
end
```

LISTING D.4: Code used for building the dynamic model

```
function [diff,alg,x_var,z_var,p_var] = BuildingDynModel(par,modelFlag)

    % Importing the CASADI library
    addpath('/Users/SALI/Desktop/MATLAB/casadi-osx-matlabR2015a-v3.5.5')
    import casadi.*
```

```
%% Normalization (for stepwise and NN)
norm = load('normalizationValues');

%% Parameters
% number of wells
n_w = 3; %[]
% gas constant
R = par.R; %[m3 Pa /K /mol]
% molecular weigth
Mw = par.Mw; %[kg/mol]

% properties
% density of oil - dim:  nwells x 1
rho_o = par.rho_o; %[kg/m3]
%riser oil density
rho_ro = par.rho_ro;%[kg/m3]
%1cP oil viscosity
mu_oil = par.mu_oil;% [Pa s]

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

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

% annulus - [m]
H_a = par.H_a;
V_a = par.V_a; %[m3]

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

% injection valve characteristics  - dim:  nwells x 1
C_iv = par.C_iv;%[m2]
% production valve characteristics  - dim:  nwells x 1
C_pc = par.C_pc;%[m2]
% riser valve characteristics
C_pr = par.C_pr;%[m2]
% account for differences in the vapor and oil velocity
slip = par.slip_real;


%% For erosion model
% Sand
d_p = par.d_p; %[m] particle diameter
% Choke
D = par.D; %[m] Gap between body and cage
% Constants
n = par.n; %[-] Velocity coefficient
% Precalculations of erosion in choke:
A_g = par.A_g; %[m2] Effective gallery area
gma = d_p./D;
```

```
%% Algebraic states
%pressure - annulus
p_ai = MX.sym('p_ai',n_w);      % 1-3 [bar] (bar to Pa  = x10^5)
%pressure - well head
p_wh = MX.sym('p_wh',n_w);      % 4-6 [bar]
%pressure - injection point
p_wi = MX.sym('p_wi',n_w);      % 7-9 [bar]
%pressure - below injection point (bottom hole)
p_bh = MX.sym('p_bh',n_w);      % 10-12 [bar]
%density - annulus
rho_ai = MX.sym('rho_ai',n_w);  % 13-15 [100 kg/m3]
%mixture density in tubing
rho_m = MX.sym('rho_m',n_w);    % 16-18 [100 kg/m3]
%well injection flow rate
w_iv = MX.sym('w_iv',n_w);      % 19-21 [kg/s]
%wellhead total production rate
w_pc = MX.sym('w_pc',n_w);      % 22-24 [kg/s]
%wellhead gas production rate
w_pg = MX.sym('w_pg',n_w);      % 25-27 [kg/s]
%wellhead oil production rate
w_po = MX.sym('w_po',n_w);      % 28-30 [kg/s]
%oil rate from reservoir
w_ro = MX.sym('w_ro',n_w);      % 31-33 [kg/s]
%gas rate from reservoir
w_rg = MX.sym('w_rg',n_w);      %34 -36 [0.1 kg/s]
%riser head pressure
p_rh = MX.sym('p_rh',1);        % 37 [bar]
%mixture density in riser
rho_r =  MX.sym('rho_r',1);     % 38 [100 kg/s]
%manifold pressure
p_m = MX.sym('p_m',1);          % 39 [bar]
%riser head total production rate
w_pr = MX.sym('w_pr',1);        % 40 [kg/s]
%riser head total oil production rate
w_to = MX.sym('w_to',1);        % 41 [kg/s]
%riser head total gas production rate
w_tg = MX.sym('w_tg',1);        % 42 [kg/s]
%gas holdup @ annulus
m_ga = MX.sym('m_ga',n_w); % 43-45 [ton]
%gas holdup @ well
m_gt = MX.sym('m_gt',n_w); % 46-48 [ton]
%oil holdup @ well
m_ot = MX.sym('m_ot',n_w); % 49-51 [ton]
%gas holdup @ riser
m_gr = MX.sym('m_gr',1);   % 52 [ton]
%oil holdup @ riser
m_or = MX.sym('m_or',1);   % 53 [ton]
%particle impact velocity
V_p = MX.sym('V_p',n_w); % 54-56
%dynamic viscosity of mixture
mu_f = MX.sym('mu_f',n_w); % 57-59
g1 = MX.sym('g1',n_w); % 60-62

% control input
% gas lift rate
w_gl = MX.sym('w_gl',n_w); %[kg/s]

% parameters
p_res = MX.sym('p_res',n_w);
%productivity index
PI = MX.sym('PI',n_w); %[kg s^-1 bar-1]
%GasOil ratio
```

```
GOR = MX.sym('GOR',n_w); %[kg/kg]
%Annulus temperature
T_a = MX.sym('T_a',n_w); %[oC]
%well temperature
T_w = MX.sym('T_w',n_w); %[oC]
%riser temperature
T_r = MX.sym('T_r',1); %[oC]
%separator pressure
p_s = MX.sym('p_s',1); %[bar]
%time transformation: CASADI integrates always from 0 to 1 and
%the USER does the time scaling with T.
T = MX.sym('T',1); %[s]

%sandrate rate
mdot_p = MX.sym('mdot_p',1); %

%erosion rate
ER = MX.sym('ER',n_w); %

%% Modeling
%gas fraction (mass) of the well holdup - avoiding zero division
xGwH = (m_gt.*1e3./max(1e-3,(m_gt.*1e3+m_ot.*1e3)));
%gas fraction (mass) of the riser holdup
xGrH = (m_gr.*1e3./(m_gr.*1e3+m_or.*1e3));

xGw = slip.*xGwH./(1 + (slip-1).*xGwH);
xOw = 1 - xGw;
xGr = slip.*xGrH./(1 + (slip-1).*xGrH);
xOr = 1 - xGr;

% ==================================
%     Well model with/withou pressure loss
% ==================================
% algebraic equations (all symbolic)
%annulus pressure - %g = 9.81
f1 = -p_ai.*1e5 + ((R.*T_a./(V_a.*Mw) + 9.81.*H_a./V_a).*m_ga.*1e3)...
+ (Mw./(R.*T_a).*((R.*T_a./(V_a.*Mw) + 9.81.*H_a./V_a).*m_ga.*1e3))...
.*9.81.*H_a;
%well head pressure
f2 = -p_wh.*1e5 + ((R.*T_w./Mw).*(m_gt.*1e3./(L_w.*A_w + L_bh.*A_bh...
- m_ot.*1e3./rho_o))) - ((m_gt.*1e3+m_ot.*1e3 )./(L_w.*A_w))...
.*9.81.*H_w/2;
%well injection point pressure
f3 = -p_wi.*1e5 + (p_wh.*1e5 + 9.81./(A_w.*L_w).*max(0,(m_ot.*1e3+...
m_gt.*1e3-rho_o.*L_bh.*A_bh)).*H_w) + (128.*mu_oil.*L_w.*w_pc./...
(3.14.*D_w.^4.*((m_gt.*1e3 + m_ot.*1e3).*p_wh.*1e5.*Mw.*rho_o)./...
(m_ot.*1e3.*p_wh.*1e5.*Mw + rho_o.*R.*T_w.*m_gt.*1e3)));
%bottom hole pressure
f4 = -p_bh.*1e5 + (p_wi.*1e5 + rho_o.*9.81.*H_bh + 128.*mu_oil.*...
    L_bh.*w_ro./(3.14.*D_bh.^4.*rho_o));
%gas density in annulus
f5 = -rho_ai.*1e2 +(Mw./(R.*T_a).*p_ai.*1e5);
%fluid mixture density in well
f6 = -rho_m.*1e2 + ((m_gt.*1e3 + m_ot.*1e3).*p_wh.*1e5.*Mw.*rho_o)...
    ./(m_ot.*1e3.*p_wh.*1e5.*Mw + rho_o.*R.*T_w.*m_gt.*1e3);
%well injection flow rate
f7 = -w_iv + C_iv.*sqrt(rho_ai.*1e2.*max(0,(p_ai.*1e5 - p_wi.*1e5)));
%wellhead prodution rate
f8 = -w_pc + 1.*C_pc.*sqrt(rho_m.*1e2.*max(0,(p_wh.*1e5 - p_m.*1e5)));
%wellhead gas production rate
f9 = -w_pg + xGw.*w_pc;
%wellhead oil prodution rate
f10 = -w_po + xOw.*w_pc;
```

```
%oil from reservoir flowrate
f11 = -w_ro + PI.*1e-6.*(p_res.*1e5 - p_bh.*1e5);
%gas from reservoir production rate
f12 = -w_rg.*1e-1 + GOR.*w_ro;
%riser head pressure
f13 = -p_rh.*1e5 + ((R.*T_r./Mw).*(m_gr.*1e3./(L_r.*A_r))) - ...
    ((m_gr.*1e3+m_or.*1e3 )./(L_r.*A_r)).*9.81.*H_r/2;
%riser density
f14 = -rho_r.*1e2 + ((m_gr.*1e3 + m_or.*1e3).*p_rh.*1e5.*Mw.*rho_ro)...
    ./(m_or.*1e3.*p_rh.*1e5.*Mw + rho_ro.*R.*T_r.*m_gr.*1e3);
%manifold pressure
f15 = -p_m.*1e5 + (p_rh.*1e5 + 9.81./(A_r.*L_r).*(m_or.*1e3+m_gr...
    .*1e3).*H_r) + (128.*mu_oil.*L_r.*w_pr./(3.14.*D_r.^4.*((m_gr.*...
    1e3 + m_or.*1e3).*p_rh.*1e5.*Mw.*rho_ro)./(m_or.*1e3.*p_rh.*...
    1e5.*Mw + rho_ro.*R.*T_r.*m_gr.*1e3)));
%total production rate of well
f16 = -w_pr + 1.*C_pr.*sqrt(rho_r.*1e2.*(p_rh.*1e5 - p_s.*1e5));
%oil total production rate
f17 = -w_to + xOr.*w_pr;
%gas total production rate
f18 = -w_tg + xGr.*w_pr;
% setting differential equations as algebraic equations since the
% dynamics of ER is on a much larger time scale
f19 = (w_gl - w_iv).*1e-3;
f20 = (w_iv + w_rg.*1e-1 - w_pg).*1e-3;
f21 = (w_ro - w_po).*1e-3;
f22 = (sum(w_pg) - w_tg).*1e-3 ;
f23 = (sum(w_po) - w_to).*1e-3 ;
f24 = - V_p + 3/(4*A_g).*(w_po./rho_o + R*T_w.*w_pg./(p_wh.*10^5*Mw));
f25 = - mu_f + mu_oil.*(w_po./rho_o)./(w_po./rho_o + R.*T_w.*w_pg./...
    (p_wh.*10^5*Mw));
f26 = -g1 + gma/0.1;
% differential equations - (all symbolic) - [ton]
% Erosion rate

if modelFlag(1) == 1
    % Use phenomenological model
    ER_constant = par.K*par.F*par.C_1*par.GF*mdot_p*par.C_unit/...
        (par.rho_t*par.A_t);
    df1 =  ER_constant.*g1.*(V_p).^n;
else
% computing the regressors
%regr = [mdot_p; p_ai; p_wh; w_ro; w_rg; p_rh; p_m; w_to; w_tg; w_gl];
% separating the regressor for the three wells. Some regressors are
% shared by the three models
    regr = [[mdot_p,mdot_p,mdot_p];
            p_ai';
            p_wh';
            w_ro';
            w_rg';
            [p_rh,p_rh,p_rh];
            [p_m,p_m,p_m];
            [w_to,w_to,w_to];
            [w_tg,w_tg,w_tg];
            w_gl'];

    % normalizing regressors with data from (ExpDataAnalysis.m)
    regrN = (regr - norm.regrCenter)./norm.regrScale;

    if modelFlag(2) == 1
        % number of regressors
        nReg = length(regr);
```

```
        % modeling stepwise
        % putting the weights in the right place
        SW_M1 = zeros(1,nReg);
        SW_M1(1) =  0.936202622379369;
        SW_M1(2) = -0.00210657217345035;
        SW_M1(3) =  0.00340292568131542;
        SW_M1(4) = -0.00201748541907079;
        SW_M1(5) =  0.00805275248024692;
        SW_M1(8) =  0.00991932368893031;
        SW_M1(9) =  0.00896809727051577;
        SW_M1(10) = 0.102758189995450;

        SW_M2 = zeros(nReg,nReg);
        SW_M2(1,2) =  0.0324197329838858;
        SW_M2(1,10) = 0.0780912090793351;
        SW_M2(2,4) = -0.00874738623206122;
        SW_M2(2,5) = -0.0258263927628834;
        SW_M2(2,8) =  0.160177833500558;
        SW_M2(2,9) =  0.00893152554186846;
        SW_M2(2,10) = 0.0117711262910581;
        SW_M2(5,8) = -0.126740929538979;
        SW_M2(5,10) = 0.0281549336959051;
        SW_M2(8,9) = -0.0606654291484604;

        intercept = -0.00741542407032436;

        %building model
        df1N = [];
        for well = 1:3
            df1N = [df1N, intercept + SW_M1*regrN(:,well) + ...
                regrN(:,well)'*SW_M2*regrN(:,well)];
        end

    elseif modelFlag(3) == 1
        % Use neural net model
        % Input 1
        x1_step1.xoffset = [-0.960381876486174;-4.46248802722717;...
            -5.17852887458485;-4.48073715332605;-3.39270547151972;...
            -4.89034267011056;-4.09638956423933;-22.1011864634595;...
            -12.6725101583375;-1.68741224378933];
        x1_step1.gain = [0.744587117901481;0.305647520874011;...
            0.217139586607698;0.278754220747781;0.313094454989053;...
            0.224952360464966;0.2398799497799;0.0883066374206199;...
            0.130806471865733;0.582950679298246];
        x1_step1.ymin = -1;

        % Layer 1
        b1 = [0.14367720318601545637;2.6585932597481978235;...
            0.21614754193338026056;-0.77521005365754414029;...
            -0.15016048001209872376;-0.18895893794020338086;...
            0.33334021415001674482;1.0495651391350007131;...
            0.096084458314441720 25;0.091400705452594849243;...
            0.9808633719414890928;-2.3640155618252562952;...
            -3.9683753071066560913;2.4716459238704158174;...
            .91033887225303244 95;-1.6445206688436171394;...
            1.7270617641242649309;1.3710565804733929607;...
            -0.83839148097354254663;2.8957307008678130344];
        IW1_1 = [-2.4154308174831933265 1.1316734083225514773...
            -0.18798508993018392399 0.50752499754821289724 ...
            0.26643122967183774374 0.18496867771410097081 ...
            -0.084381354525935925448 0.07085802248326253383...
            -0.16205693380690422423 -0.77140106419434217866;...
            -2.5214783316455919859 -0.61767172907865119935 ...
```

```
-0.010085000823519651991 -0.17096230134548739965...
-0.045858540570502259737 -0.075738671538680010786...
0.011697215615493047544 0.89427052368563375584...
0.017841955990787663339 0.36230068772579970826;...
-0.25273389398206069778 0.19984515133597760055...
-0.012532406301889509326 -0.38710321317608464842 ...
0.31744353291743737655 0.30392705893160959496 ...
-0.48629036603932135341 -0.38527217222465176549...
0.45637964870801228656 -0.003963315323805872113 2;...
-0.14137397926041037066 0.22839219307467162334 ...
0.69305240971952908335 -0.84866491694379209143...
1.1527220931617192523 0.94439458980977164515 ...
-0.30465769707891104945 0.37955421319014276405...
0.40549188230617233542 -0.45339085921888461206;...
0.11430828401814195627 -0.11175304651558616575...
-0.47704525296752076091 0.64561656700497560557 ...
-0.916648058594788262 -0.7749657348705597526 ...
0.31298468285909419873 0.54810711313383120302 ...
-0.44114123533557358936 0.31640372701778679554;...
0.068730081107593654632 0.15546512469078524465...
0.050894605481055565921 -0.18914112137188648921...
0.018993525551347312241 0.19626799805238009933...
-0.035325882080909642 99 0.055668577385928771917...
0.084229796875306933712 -0.26300932511266794656;...
2.6071271914854570184 1.4513736888071724351 ...
0.007458460740365098 3112 0.033216876158219783843...
0.42244033655690699236 0.38798092469935352433 ...
-0.15221425801286470048 0.38258859113757298642...
0.73193461514120727873 -1.3507173453150382869;...
-0.028825152288339869061 -2.2081808752373786042...
-0.061859892727623107256 -0.24630047120646228476...
-0.14981246649827839601 -0.034852738621890928805...
-0.030574791510504781278 0.037204871248042073462...
-0.16536073768499864878 1.4525224141871446726;...
1.9848749283793558629 -0.65087570179592324493...
-0.015401431195047120964 0.033833379986935939454 ...
0.24766683536891354045 0.23619359560874905735...
-0.061828332005150851702 1.3513681000858552839...
0.474773459888372662 -0.10609490256059561641;...
-0.036487529426405596045 -0.97950882073050538068...
0.13802254156185905787 0.51835177883852656677 ...
0.5464992554107237499 -0.40436909977663293425 ...
0.054607279876782710559 0.27824744732054024 21...
-0.15734213693163773273 0.19478347868992934577;...
1.0475016718182594833 2.6268066719483176286...
0.10555368421258967682 0.39987028553426201549...
0.18887696676464152401 0.070115438039725697106 ...
0.059914126014487971428 -0.45423202911649523 95...
0.32303468224857806446 -1.6871729733077229785;...
2.7529274108906229834 -0.765014788435594561 56...
0.001816806835687384 1133 -0.187973170630295671 75...
-0.056699649112672743934 -0.029790717506350518351...
0.009821245676382175 2437  -1.080001716418058643 8 ...
0.12325875968482226386 0.513407727577221639 77;...
-4.5690876368247836936 0.29730258463782233136 ...
0.021112838346440350457 -0.10558917971050130191...
-0.12168972717260564953 0.005885748073428376 7684...
-0.081390986053868324968 -2.0464715164136473291 ...
0.10785277117965144655 0.131086146364862016 21;...
-4.8994890602218816866 0.35234830166711450516 ...
-0.010013370059842452778 0.070144745130894969876 ...
0.046229090500515740425 0.004113416526185162229 4...
-0.024505471792468629805 2.3295584707548151471 ...
```

```
    -0.18538945206563731127 -0.24224570259646180381;...
    1.2615286870191546598 -0.11816400022351983279...
    -0.078271926206767192258 0.00040698798715488505471...
    0.12261647857164348352 -0.025018061970269584587...
    0.062754883480531528 65 1.1711207094070865686...
    -0.15128987683747979753 0.011625717221515827204;...
    0.25536351914056132362 1.5069046552312748144...
    0.040419707542304748882 -0.017552326786328720926...
    0.2518564757364556983 0.15937380698629941 ...
    -0.023214434568652780183 0.67048552164710273349...
    0.19329951559387478777 -1.2091523245907629391;...
    1.4177390842659385317 -0.05695648275691699014...
    -0.10986006754139095165 -0.019963781991928088166 ...
    -0.13614685136084950234 -0.35015709512873327558...
    0.13999639673564417963 0.4251125746769744973 ...
    -0.8814843316401170803 0.1666711241041228253;...
    0.41968287139809840047 -2.6683330419759339058 ...
    -0.098395196139792875933 -0.41495981268258286256...
    -0.11880388438818172137 0.011362744772546578068 ...
    -0.03973014734543630494 -0.50920599962215373768...
    -0.053892242227375410091 1.6413468033934477397;...
    -1.3231026141444686139 -1.8493906165810243269...
    -0.056111174375088730681 -0.14865803431058338679 ...
    -0.14887176618909070402 -0.071243715652375777525 ...
    -0.020647144186653382941 0.35039232222030086694...
    -0.26197868120861045327 1.2119089090990931012;...
    -4.9902268121632555875 -0.41721491113174447 03...
    -0.0015895483878233280118 -0.15307047671214385476 ...
    -0.02357680456433861732 -0.066690710366615932325 ...
    -0.010463477723396974808 2.27970201198322453...
    -0.10341726132984332964 0.25867357616550795685];

% Layer 2
b2 = -1.3850185588017143168;
LW2_1 = [-0.17239018609501124968 -2.4686570038789317216...
    -0.28132495741565127778 -0.65952833029270596654 ...
    -1.1372584731091543997 -1.9460586966551658428...
    -0.41574234276122307152 1.7184263126322130155 ...
    0.77527660655890684449 -0.38169495762906624492 ...
    -1.1742597402214667301 -2.7883027491604210901 ...
    -1.9903053748364492037 -2.069048211694330508 ...
    -2.037627117462171622 0.86637931156103764607...
    0.53376422641646292799 -0.94540883414461107659...
    -1.9486075433738543339 2.0027264449775534771];

% Output 1
y1_step1.ymin = -1;
y1_step1.gain = 0.677942953516802;
y1_step1.xoffset = -1.24456183373413;

% ===== COMPUTING NN OUTPUTS ========
xp1 = (regrN - x1_step1.xoffset).*x1_step1.gain + ...
    x1_step1.ymin;

% Layer 1
temp1 = b1 + IW1_1*xp1;
a1 = 2 ./ (1 + exp(-2*temp1)) - 1;

% Layer 2
a2 = b2 + LW2_1*a1;

% Output 1
df1N = (a2 - y1_step1.ymin)/y1_step1.gain + y1_step1.xoffset;
```

```
        end

        % de-normalizing response (and also changing time units)
        df1 = (df1N'*norm.predScale + norm.predCenter)/(par.T);

    end

    % Form the DAE system
    diff = vertcat(df1);
    alg = vertcat(f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15,...
        f16,f17,f18,f19,f20,f21,f22,f23,f24,f25,f26);

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

    diff = substitute(diff,p_res,par.p_res);
    diff = substitute(diff,T_w,par.T_w);

    % concatenate the differential and algebraic states
    x_var = vertcat(ER);
    z_var = vertcat(p_ai,p_wh,p_wi,p_bh,rho_ai,rho_m,w_iv,w_pc,w_pg,...
        w_po,w_ro,w_rg,p_rh,rho_r,p_m,...
        w_pr,w_to,w_tg,m_ga,m_gt,m_ot,m_gr,m_or,V_p, mu_f,g1);
    p_var = vertcat(w_gl,mdot_p,GOR,PI,T);

end
```

LISTING D.5: Code for building the controller (Optimizer)

```
function solver = BuildingNMPC(diff,alg,x_var,z_var,p_var,par,nmpcPar)


addpath('/Users/SALI/Desktop/MATLAB/casadi-osx-matlabR2015a-v3.5.5')
import casadi.*

%% Using 3 collocation points:
h = par.T;
% Radau
t = [collocation_points(3, 'radau')];
% Finding M
M = [t',0.5*t'.^2,1/3*t'.^3]*inv([[1;1;1],t',t'.^2]);

%% Defining system OF
% for computing Du
U_1 = MX.sym('U_1',nmpcPar.nu);
U1 = MX.sym('U1',nmpcPar.nu);

% slack variable
s = MX.sym('s',nmpcPar.nx);

% objective function
w_po = z_var(28:30);
L =  -sum(w_po)+ nmpcPar.rho*sum(s)  + 1/2 * ((U1 - U_1)'*nmpcPar.R*...
    (U1 - U_1));

% creating system function (LHS of the dynamic equations)
```

```
f = Function('f',{x_var,z_var,p_var,U1,U_1,s},{diff,alg,L});

%% Defining empty nlp-problem
% objective function
J = 0;

% declare variables (bounds and initial guess)
w = {};
% w0 = [];
% lbw =[];
% ubw = [];

% declare constraints and its bounds
g = {};
% lbg = [];
% ubg = [];

%% declaring parameters
xk_meas = MX.sym('xk_meas',nmpcPar.nx);
zk_meas = MX.sym('zk_meas',nmpcPar.nz);
uk_meas = MX.sym('uk_meas',nmpcPar.nu);

% mdot_p,GOR,PI,T - which are fixed
p = MX.sym('p',8);

%% Lifting initial conditions

% initial state
x_prev = MX.sym('X0',nmpcPar.nx);
w = {w{:},x_prev}; % 1-3
% lbw = [lbw,xk_meas];
% ubw = [ubw,xk_meas];
% w0 = [w0;xk_meas];

% initial input
uk = MX.sym('uk_init',nmpcPar.nu);
w = {w{:}, uk}; % 4-6
% w0 = [w0;uk_meas];
% lbw = [lbw;uk_meas];
% ubw = [ubw;uk_meas];



%% Looping through until timeend
for k = 1:nmpcPar.np

    % storing the previous input
    uprev = uk;

    % creating current input
    uk = MX.sym(['uk_' num2str(k)],nmpcPar.nu);
    w = {w{:}, uk}; % 7-9
%     w0 = [w0;  uk_meas];
%     lbw = [lbw;nmpcPar.umin*ones(nmpcPar.nu,1)];
%     ubw = [ubw;nmpcPar.umax*ones(nmpcPar.nu,1)];

    % creating current slack variables
    s = MX.sym(['s_',num2str(k)],nmpcPar.nx);
    w = {w{:},s}; % 10-12
%     w0 = [w0;0*ones(nmpcPar.nu,1)];
%     lbw = [lbw;0*ones(nmpcPar.nu,1)];
%     ubw = [ubw;1*ones(nmpcPar.nu,1)];
```

```
    % Adding constraint for delta_u
    duk = uk - uprev;
    g = {g{:},duk};

%     if k > nmpcPar.nm
%         lbg = [lbg;zeros(nmpcPar.nu,1)];
%         ubg = [ubg; zeros(nmpcPar.nu,1)];
%     else
%         lbg = [lbg;-nmpcPar.dumax*ones(nmpcPar.nu,1)];
%         ubg = [ubg;nmpcPar.dumax*ones(nmpcPar.nu,1)];
%     end

    % Collocation points
    fk = [];
    Xk1 = [];
    gk = [];
    L1 = [];

    for d = 1:3
        % creating states at collocation points
        Xk = MX.sym(['Xk_' num2str(k),'_',num2str(d)],nmpcPar.nx);
        Zk = MX.sym(['Zk_' num2str(k),'_',num2str(d)],nmpcPar.nz);
        w = {w{:}, Xk, Zk}; % 13-15 | 16 - 77
%         w0 = [w0;xk_meas;zk_meas];
%         lbw = [lbw;zeros(nmpcPar.nx,1);zeros(nmpcPar.nz,1)];
%         ubw = [ubw;inf*ones(nmpcPar.nx,1);inf*ones(nmpcPar.nz,1)];

        % for continuinity
        Xk1 = [Xk1,Xk];

        % Calculating xdot and objective function
        [fk1,gk1,L] = f(Xk,Zk,vertcat(uk,p),uk,uprev,s);

        L1 = [L1;L];
%         if k > nmpcPar.nm
%             L1(d) = L - 1/2 * ((uk-uprev)'*nmpcPar.R*(uk-uprev));
%         end

        fk = [fk, fk1];
        gk = [gk, gk1];

    end

    % integrating the system
    x_next1 = [];
    for d = 1:3
        % Calculating M*xdot for each collocation point
        Mfk = M(d,1)*fk(:,1) + M(d,2)*fk(:,2) + M(d,3)*fk(:,3);

        % Calculating x
        x_next = x_prev+h*Mfk;
        x_next1 = [x_next1,x_next];

        % Adding xk and Xk1 as constrains as they must be equal - in
        % collocation intervals
        % algebraic constraints are set to zero in the collocation point
        g = {g{:},x_next-Xk1(:,d),gk(:,d)};
%         lbg = [lbg;zeros(nmpcPar.nx,1);zeros(nmpcPar.nz,1)];
%         ubg = [ubg;zeros(nmpcPar.nx,1);zeros(nmpcPar.nz,1)];
    end


    % updating objective function
```

```
    ML = M*L1;
    J = J + ML(3);

    % New NLP variable for state at end
    x_prev = MX.sym(['x_init_' num2str(k)],nmpcPar.nx);
    w = {w{:}, x_prev}; % 78 - 80
%     w0 = [w0;xk_meas];
%     lbw = [lbw;zeros(nmpcPar.nx,1)];
%     ubw = [ubw;inf*ones(nmpcPar.nx,1)];
%
    % Gap
    g = {g{:},x_next-x_prev};
%     lbg = [lbg;zeros(nmpcPar.nx,1)];
%     ubg = [ubg;zeros(nmpcPar.nx,1)];

    % Constraint on erosion
    g = {g{:},x_prev-s};
%     lbg = [lbg;zeros(nmpcPar.nx,1)];
%     ubg = [ubg;nmpcPar.x_threshold*ones(nmpcPar.nx,1)];

end

% Formalizing problem
nlp = struct('x',vertcat(w{:}),'g',vertcat(g{:}),'f',J,'p',vertcat...
    (xk_meas,zk_meas,uk_meas,p));

% Assigning solver (IPOPT)
solver = nlpsol('solver','ipopt',nlp);


end
```

LISTING D.6: Codes used for calculating in the controller (optimizer)

```
function [u_,sArray,ehatArray,inputArray,ofValue,solFlag] = SolvingNMPC...
    (solver,x_next,z_next,u_k,mdot_p,GOR,PI,T,nmpcPar)

% declare variables (bounds and initial guess)
w0 = [];
lbw =[];
ubw = [];

% declare constraints and its bounds
lbg = [];
ubg = [];

% initial state
lbw = [lbw,x_next];
ubw = [ubw,x_next];
w0 = [w0;x_next];

% initial input
w0 = [w0;u_k];
lbw = [lbw;u_k];
ubw = [ubw;u_k];

%% Looping through until timeend
for k = 1:nmpcPar.np
    w0 = [w0; u_k];
    lbw = [lbw;nmpcPar.umin*ones(nmpcPar.nu,1)];
    ubw = [ubw;nmpcPar.umax*ones(nmpcPar.nu,1)];
```

```
    % creating current slack variables
    w0 = [w0;0*ones(nmpcPar.nu,1)];
    lbw = [lbw;0*ones(nmpcPar.nu,1)];
    ubw = [ubw;10*ones(nmpcPar.nu,1)];

    if k > nmpcPar.nm
        lbg = [lbg;zeros(nmpcPar.nu,1)];
        ubg = [ubg; zeros(nmpcPar.nu,1)];
    else
        lbg = [lbg;-nmpcPar.dumax*ones(nmpcPar.nu,1)];
        ubg = [ubg;nmpcPar.dumax*ones(nmpcPar.nu,1)];
    end

    for d = 1:3
        % creating states at collocation points
        w0 = [w0;x_next;z_next];
        lbw = [lbw;zeros(nmpcPar.nx,1);zeros(nmpcPar.nz,1)];
        ubw = [ubw;inf*ones(nmpcPar.nx,1);inf*ones(nmpcPar.nz,1)];

    end

    % integrating the system
    for d = 1:3

        % Adding xk and Xk1 as constrains as they must be equal - in
        % collocation intervals
        % algebraic constraints are set to zero in the collocation point
        lbg = [lbg;zeros(nmpcPar.nx,1);zeros(nmpcPar.nz,1)];
        ubg = [ubg;zeros(nmpcPar.nx,1);zeros(nmpcPar.nz,1)];
    end


    % New NLP variable for state at end
    w0 = [w0;x_next];
    lbw = [lbw;zeros(nmpcPar.nx,1)];
    ubw = [ubw;inf*ones(nmpcPar.nx,1)];

    % Gap
    lbg = [lbg;zeros(nmpcPar.nx,1)];
    ubg = [ubg;zeros(nmpcPar.nx,1)];

    % Constraint on erosion
    lbg = [lbg;zeros(nmpcPar.nx,1)];
    ubg = [ubg;nmpcPar.x_threshold*ones(nmpcPar.nx,1)];

end


% Solving the problem
sol = solver('x0',w0,'lbx',lbw,'ubx',ubw,'lbg',lbg,'ubg',ubg,'p',...
    [x_next;z_next;u_k;mdot_p;GOR;PI;T]);

%% Extracting solution
w_opt = full(sol.x);

% catch error
if solver.stats.success ~=1
    % solution failed
    solFlag = 0;

    u_ = u_k; % fix inputs
```

```
    %dummy
    sArray = zeros(nmpcPar.nx,1);
    ehatArray = zeros(nmpcConfig.nx,nmpcConfig.np + 1);

else
    % solution succeeded
    solFlag = 1;

    % variable order
    % 1-3: x0
    % 4-6: u0
    % 7-9: u1
    % 10-12: s1
    % 13-15 | 78-80 | 143 - 145: x11, x12, x13
    % 16-77 | 81-142| 146 - 207: z11, z12, z13
    % 208-210: xprev1

    % 211-213: u2
    % 214-216: s2
    % 217-219 | 282-284 | 347 - 349: x21, x22, x23
    % 220-281 | 285-346 | 350 - 411: z21, z22, z23
    % 412-414: xprev2

    % 415-417: u3
    % 418-420: s3
    % 421-423 | 486-488 | 551 - 553: x31, x32, x33
    % 424-485 | 489-550 | 554 - 615: z31, z32, z33
    % 616-618: xprev3

    % total variables in one iteration = 3(u) + 3(s) + 9(xkd) +
    % 186(zkd) + 3(xprev) = 204

    u_ = w_opt(7:9);

    ofValue = full(sol.f);

    ehatArray = [w_opt(1:3), w_opt(208:210)];
    inputArray = w_opt(7:9);
    sArray = w_opt(10:12); %s1

    for ii = 1:nmpcPar.np - 1
        temp = 412 + (ii - 1)*204;
        ehatArray = [ehatArray, w_opt(temp:temp + 2)];

        temp2 = 214 + (ii - 1)*204;
        sArray = [sArray, w_opt(temp2:temp2 + 2)];

        temp3 = 211 + (ii - 1)*204;
        inputArray = [inputArray, w_opt(temp3:temp3 + 2)];
    end

end

end
```

LISTING D.7: Code used for initial conditions used in the calculation

```
function [dx0,z0,u0] = InitialConditionGasLift_5

%% Differential states
```

```
%well erosion rate
ER0 = [1,1,1]'/(365*24*3600); %[mm/s] 9-11

%% Algebraic states
%pressure - annulus
p_ai0 = [61.9230, 62.1454, 62]';%[bar] 1-3  (bar to Pa  = x10^5)
%pressure - well head
p_wh0 = [42.5851, 45.3082, 44]';%[bar] 4-6
%pressure - injection point
p_wi0 = [56.8713, 57.1119, 57]';%[bar] 7-9
%pressure - below injection point (bottom hole)
p_bh0 = [96.1433, 98.3867, 96.25]';%[bar] 10-12
%density - annulus
rho_ai0 = [0.4949, 0.4967, 0.4955]';%[100 kg/m3] 13-15
%mixture density in tubing
rho_m0 = [2.3400, 2.2359, 2.3]';%[100 kg/m3] 16-18
%well injection flow rate
w_iv0 = [0.5000, 0.5000,0.5000]';%[kg/s] 19-21
%wellhead total production rate
w_pc0 = [30.1212, 33.3235,32]';%[kg/s] 22-24
%wellhead gas production rate
w_pg0 = [3.1928, 4.0168, 4]';%[kg/s] 25-27
%wellhead oil production rate
w_po0 = [26.9283, 29.3067, 28]';%[kg/s] 28-30
%oil rate from reservoir
w_ro0 = [26.9283, 29.3067, 28]';%[kg/s] 31-33
%gas rate from reservoir
w_rg0 = [26.9283, 35.1680, 28]';%[0.1 kg/s] 34-36
%riser head pressure
p_rh0 = 22.9558;%[bar] 37
%mixture density in riser
rho_r0 = 1.3618;%[100 kg/m3] 38
%manifold pressure
p_m0 = 32.8920;%[bar] 39
%riser head total production rate
w_pr0 = 63.4446;%[kg/s] 40
%riser head total oil production rate
w_to0 = 56.2350;%[kg/s] 41
%riser head total gas production rate
w_tg0 = 7.2096;%[kg/s] 42

%%setting diff states as algebraic
%gas holdup @ annulus
m_ga0 = [1.0568, 1.0606, 1.0644]';%[ton] 43-45 mga(2) +(mga(2)-mga(1))
%gas holdup @ well
m_gt0 = [0.7470, 0.7956, 0.8442]';%[ton] 46-48 mgt(2) + (mgt(2)-mgt(1))
%oil holdup @ well
m_ot0 = [6.3000, 5.8047, 5.3094]';%[ton] 49-51
%gas holdup @ riser
m_gr0 = 0.1265;%[ton] 52
%oil holdup @ riser
m_or0 = 0.9863;%[ton] 53
%particle impact velocity
V_p0 = [2,2,2]'; % 54-56
%mixed dynamic viscosity
mu_f0 = [0.001,0.001,0.001]'; % 57-59
%g1
g10 = [0.2,0.2,0.2]'; % 60-62

%% Inputs
%gas lift rate
w_gl0 = [0.5,0.5,0.5]'; %[kg/s]
```

```
dx0 = vertcat(ER0);
z0 = vertcat(p_ai0,p_wh0,p_wi0,p_bh0,rho_ai0,rho_m0,w_iv0,w_pc0,w_pg0,...
    w_po0,...
    w_ro0,w_rg0,p_rh0,rho_r0,p_m0,w_pr0,w_to0,w_tg0,m_ga0,m_gt0,m_ot0,...
    m_gr0,m_or0,V_p0,mu_f0,g10);
u0 = w_gl0;
```

LISTING D.8: Code for the gas lift parameters

```
function  par = ParametersGasLift

%number of wells
par.n_w = 3;
%gas constant
par.R = 8.314; %[m3 Pa/(K mol)]
%molecular weigth
par.Mw = 20e-3; %[kg/mol] -- Attention: this unit is not usual

%% Properties
%density of oil - dim:  nwells x 1
par.rho_o = 8*1e2;%[8;8].*1e2; %[kg/m3]
%riser oil density
par.rho_ro = par.rho_o; %[kg/m3]
%1cP oil viscosity
par.mu_oil = 1*0.001; %[Pa s or kg/(m s)]

%% Project
%well parameters - dim:  nwells x 1
%length
par.L_w = [1500;1500;1500]; %[m]
%height
par.H_w  = [1000;1000;1000]; %[m]
%diameter
par.D_w = [0.121;0.121;0.121]; %[m]
%well transversal area
par.A_w = pi.*(par.D_w/2).^2;%[m2]

%well below injection - [m]
par.L_bh = [500;500;500];
par.H_bh = [500;500;500];
par.D_bh = [0.121;0.121;0.121];
par.A_bh = pi.*(par.D_bh/2).^2;%[m2]

%annulus - [m]
par.L_a = par.L_w;
par.H_a = par.H_w;
par.D_a = [0.189;0.189;0.189];
%volume of the annulus
par.V_a = par.L_a.*(pi.*(par.D_a/2).^2 - pi.*(par.D_w/2).^2); %[m3]

%riser - [m]
par.L_r = 500;
par.H_r = 500;
par.D_r = 0.121;
%riser areas
par.A_r = pi.*(par.D_r/2).^2;%[m2]

%injection valve characteristics  - dim:  nwells x 1
par.C_iv = [0.1e-3;0.1e-3;0.1e-3];%[m2]
%production valve characteristics  - dim:  nwells x 1
par.C_pc = [2e-3;2e-3;2e-3];%[m2]
```

```
%riser valve characteristics
par.C_pr = [10e-3];%[m2]
%parameter to account for differences in gas and liquid pressures
par.slip_real = 1;

%parameters
%reservoir pressure
par.p_res = [150;155;160]; % [bar]
%Annulus temperature
par.T_a = [28+273;28+273;28+273]; %[K]
%well temperature
par.T_w = [32+273;32+273;32+273]; %[K]
%riser temperature
par.T_r = 30+273; %[K]
%separator pressure
par.p_s = 20; %[bar]

%sampling time /control interval /1 simulation iteration time
par.T = 3600*24; % [s]

%% For erosion model
% Sand
par.d_p = 2.5*10^(-4); %[m] particle diameter
par.rho_p = 2.5*10^3; %[kg/m3] particle density
par.mdot_p = 100*10^(-3); %[kg/s] sand rate

% Choke
par.K = 2*10^(-9); %[-] material erosion constant
par.rho_t = 7800; %[kg/m3] sensity CS
par.r = 0.2; %[m] radius of curvature
par.D = 0.05; %[m] Gap between body and cage
par.H = 0.15; %[m] Height of gallery

% Constants
par.C_unit = 1000; % Unit conversion factor: now in mm/s
par.C_1 = 1.25; %[-] Model/geometry factor
par.n = 2.6; %[-] Velocity coefficient
par.GF = 2; %[-] Geometry factor

% Precalculations of erosion in choke:
par.alpha = atan(1/sqrt(2*par.r));
par.F = 0.6*(sin(par.alpha) + 7.2*(sin(par.alpha) - sin(par.alpha)^2))...
    ^0.6 * (1-exp(-20*par.alpha));
par.A_g = 2*par.H*par.D; %[m2] Effective gallery area
par.A_t = par.D_w(1)^2*pi/(4*sin(par.alpha)); % Area exposed to erosion
par.ER_constant = par.K*par.F*par.C_1*par.GF*par.mdot_p *par.C_unit/...
    (par.rho_t*par.A_t);


%System parameters for nominal model
par.GOR = [0.10;0.12;0.11];
par.PI = [5;5;5];

%% For scaling the noise
% pressure meters = 1
% flow meters = 0.1
par.scale = [1,1,1,1,1,1,0.1,0.1,0.1,0.1,0.1,0.1,1,1,0.1,0.1]';
```

LISTING D.9: Code for simulation the hybrid Health-aware controllers

```
% Implementation of Health Aware Controller based on Joachim's thesis
clear
close all
clc

addpath('/Users/SALI/Desktop/MATLAB/casadi-osx-matlabR2015a-v3.5.5')
import casadi.*

% for reproduciability
rng('default');

%% Configuration
% Flag to choose the model (choose only one)
phenomeno = 1;
stepwise = 0;
neuralnet = 0;
flagModel = [phenomeno,stepwise,neuralnet];

% show the plot while simulating
showPlot = true; % true | false

% setting simulation length
simLength = 231; % 450 | 500 | 550

% Initial condition
[x0,z0,u0] = InitialConditionGasLift_5;

% System parameters
par = ParametersGasLift;

%states to measurement mapping function
H = zeros(16,length(z0));
%pai - annulus pressure, well 1-3
H(1,1) = 1;
H(2,2) = 1;
H(3,3) = 1;
%pwh - well head pressure, well+ 1-3
H(4,4) = 1;
H(5,5) = 1;
H(6,6) = 1;
%wro - wellhead gas production rate, well 1-3
H(7,25) = 1;
H(8,26) = 1;
H(9,27) = 1;
%wrg - wellhead oil production rate, well 1-3
H(10,28) = 1;
H(11,29) = 1;
H(12,30) = 1;
%prh - riser head pressure
H(13,37) = 1;
%pm - manifold pressure
H(14,39) = 1;
%wto - riser head total oil production rate
H(15,41) = 1;
%wtg - riser head total gas production rate
H(16,42) = 1;
par.H = H;

% generating sand profile
sandArray = sandproductionrate(0.01,simLength,'exp',0.02);

% Building Dynamic Model
[diff,alg,x_var,z_var,p_var] = BuildingDynModel(par,flagModel);
```

```
% control tuning
nmpcConfig.umax = 2;    %  3 | 2
nmpcConfig.umin = 0.4;
nmpcConfig.dumax = 0.01;
nmpcConfig.x_threshold = 2;  % 0.8 | 1 | 2
nmpcConfig.nx = size(x0,1);
nmpcConfig.nz = size(z0,1);
nmpcConfig.nu = size(u0,1);

nmpcConfig.nm = 70;
nmpcConfig.np = 100;
%nmpcConfig.rho = 1e3; % 1e3 | 999999
nmpcConfig.rho = 1000000; % 1e3 | 999999

nmpcConfig.R = 0.01*eye(nmpcConfig.nu); % 0 | 0.01 | 1

% Building NMPC
solverNMPC = BuildingNMPC(diff,alg,x_var,z_var,p_var,par,nmpcConfig);

%% Initializing the simulation
xk = x0;
xHat = x0; % initial value is assumed known
zk = z0;
uk = u0;
yk = par.H*z0;

%  Initializing  slacks (not needed  plotting if not using the controller)
s = zeros(nmpcConfig.nx,nmpcConfig.np);
% erosion predict "inside the controller" --> prognostics
erosionHat = zeros(nmpcConfig.nx,nmpcConfig.np + 1);
% input sequence prediction
inputSeq = zeros(nmpcConfig.nx,nmpcConfig.np);

OF = 0;

solFlag = 0;
controlTime = 0;

% creating variable to save breakdown time (if applicable)
tBreak = [];

% preparing for plotting
%colors associated with each well
cc = {'r','k','b'};

% plant info
xPlant = [];
zPlant = [];
yPlant = [];
totalProductionPlant = [];

%diagnositics info
erosionHatDiag = [];

%control info
inputControl = [];
flagControl = [];
ofControl = [];
CPUtimeControl = [];

for ii = 0:simLength
    erosionPrediction{ii + 1} = [];
```

```
    slackControl{ii + 1} = [];
    inputPrediction{ii + 1} = [];
end

%% Simulating

for tt = 0:simLength
    fprintf('     iteration >>> %0.0f [day]\n',tt)

    if tt ~=0
        % estimating the current erosion
        xHat = Diagnostics(xk,xHat,yk,uk,sandArray(tt + 1),par,flagModel);

        tic
        % Calculating input with NMPC
        % perfect information
        [uk,s,erosionHat,inputSeq,OF,solFlag] = SolvingNMPC...
            (solverNMPC,xk,zk,uk,sandArray(tt + 1)...
            ,par.GOR,par.PI,par.T,nmpcConfig);

        % with diagnostics
        % [uk,s,erosionHat,inputSeq,OF,solFlag] = ...
        %SolvingNMPC(solverNMPC,xHat,zk,uk,sandArray(tt + 1)...
          %  ,par.GOR,par.PI,par.T,nmpcConfig);

          % computing execution time
          controlTime = toc;
    end

    % Finding the state after applying input u
    [xk,zk] = WellPlantModel(xk,zk,uk,sandArray(tt + 1),par);

        %%%%%%%%%%%%%%%%
        % Saving Data %
        %%%%%%%%%%%%%%%%
        erosionHatDiag = [erosionHatDiag, xHat];

        inputControl = [inputControl, uk];
        flagControl =  [flagControl, solFlag];
        erosionPrediction{tt + 1} = erosionHat;
        slackControl{tt + 1} = s;
        inputPrediction{tt + 1} = inputSeq;
        ofControl = [ofControl, OF];
        CPUtimeControl = [CPUtimeControl, controlTime];

        xPlant = [xPlant, xk];
        zPlant = [zPlant, zk];

        yk = par.H*zk + par.scale.*randn(length(yk),1);
        yPlant = [yPlant, yk];
        totalProductionPlant = [totalProductionPlant, sum(zk(28:30))];



        if tt > 0 && showPlot %rem(t,50) == 0 % checking results sporadiacally
            time = 0:1:tt;

            %%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Plotting --- controller %
            %%%%%%%%%%%%%%%%%%%%%%%%%%%
            f1 = figure(1);
            clf
```

```
            subplot(3,1,1);
                hold on
                for well = 1:3
                    stairs(time,inputControl(well,:),cc{well},...
                        'LineWidth',2,'HandleVisibility','off'); %% 1.5 2
                end

                xline(simLength,'r:','HandleVisibility','off');
                yline(nmpcConfig.umax,'k--','LineWidth',2);   %%

                ylim([0,3.75]);
                xlim([0,simLength + 100]);
                set(gca,'FontSize',14)
                legend('Max. gas cap.','Position',[0.69 0.90 0.20 0.041])

                xlabel('time [day]','fontsize',14);
                ylabel('Gas lift rate [kg/s]','fontsize',14);

            subplot(3,1,2);

                yline(nmpcConfig.x_threshold,'k--','LineWidth',2);  %% 1, 2
                hold on
                for well = 1:3
                    plot(time,xPlant(well,:),cc{well},'LineWidth',2);  % 1.5 , 2
                    plot(tt:tt + nmpcConfig.np,erosionHat(well,:)...
                        ,cc{well},'linestyle',':','LineWidth',1);
                    plot(tt,xHat(well),'MarkerFaceColor',cc{well},...
                        'marker','o','HandleVisibility','off');
                end

                xline(simLength,'r:');

                text(simLength + 50,0.7,'\leftarrow Maintenance',...
                    'HorizontalAlignment','center','FontSize',7);
                text(simLength + 50,0.5,'Stop','HorizontalAlignment',...
                    'center','FontSize',7);
                set(gca,'FontSize',14)
                legend({'threshold','Real','Predicted'},'Position',...
                    [0.14 0.59 0.17 0.11])

                box on

                ylim([0,nmpcConfig.x_threshold*1.5]);
                xlim([0,simLength + 100]);
                xlabel('Time [day]','fontsize',14);
                ylabel('Erosion [mm]','fontsize',14);

            subplot(3,1,3);
                plot(time,totalProductionPlant,'b','LineWidth',2); %% 1.5 2
                hold on
                xline(simLength,'r:');
                set(gca,'FontSize',14)
                ylim([80,88]);
                xlim([0,simLength + 100]);
                xlabel('Time [day]','fontsize',14);
                ylabel('Oil production [kg/s]','fontsize',14);

        end

%         % emulating system breaking if threshold is achieved
%         if ~all(xk < nmpcConfig.x_threshold)
%
%             % saving break time
```

```
%             tBreak = tt;
%
%             % breaking loop
%             break
%         end


end

%% saving results
if flagModel(1) == 1
    name = 'HAC_pheno';
elseif flagModel(2) == 1
    name = 'HAC_step';
elseif flagModel(3) == 1
    name = 'HAC_NN';
end

save(name,'simLength','tBreak','xPlant','zPlant','yPlant',...
    'totalProductionPlant','slackControl','inputControl',...
    'flagControl','erosionPrediction','CPUtimeControl',...
    'ofControl','erosionHatDiag','inputPrediction')

%% Plotting the final profiles
f2 = figure(2);

if ~isempty(tBreak)
   simLength = tBreak;
end
time = 0:1:simLength;

    subplot(3,1,1);
        stairs(time,inputControl(1,:),cc{1},'LineWidth',2,...
            'HandleVisibility','off');  %% 1.5
        hold on
        stairs(time,inputControl(2,:),cc{2},'LineWidth',2,...
            'HandleVisibility','off');   %% 1.5
        stairs(time,inputControl(3,:),cc{3},'LineWidth',2,...
            'HandleVisibility','off');   %% 1.5

        yline(nmpcConfig.umax,'k--','LineWidth',2);

        ylim([0,3.75]);
        xlim([0,simLength]);
        set(gca,'FontSize',14)
        legend('Max. gas cap.','Position',[0.69 0.90 0.20 0.041])

        xlabel('time [day]','fontsize',14);
        ylabel('Gas lift rate [kg/s]','fontsize',14);

    subplot(3,1,2);

        yline(nmpcConfig.x_threshold,'k--','LineWidth',1.5);
        hold on
        plot(time,xPlant(1,:),cc{1},'LineWidth',2);
        plot(time,xPlant(2,:),cc{2},'LineWidth',2);
        plot(time,xPlant(3,:),cc{3},'LineWidth',2);

        box on
        set(gca,'FontSize',14)
        legend({'threshold','Well 1','Well 2','Well 3'},...
            'Location','northwest');
```

```
        ylim([0,3]);
        xlim([0,simLength]);
        xlabel('Time [day]','fontsize',14);
        ylabel('Erosion [mm]','fontsize',14);

    subplot(3,1,3);
        plot(time,totalProductionPlant,'b','LineWidth',2);
        hold on
        set(gca,'FontSize',14)
        ylim([80,88])
        xlim([0,simLength]);
        xlabel('Time [day]','fontsize',14);
        ylabel('Total oil production [kg/s]','fontsize',14);

    set(gcf,'color','w');

    namePlot = [name,'_results.pdf'];
    print(f2,'-r300','-dpdf',namePlot);


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Plotting --- diagnosis  %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    f3 = figure(3);

    for well = 1:3
        subplot(3,1,well);
        yline(nmpcConfig.x_threshold,'k--','LineWidth',1.5);
        hold on
        plot(time,erosionHatDiag(well,:),cc{well},'LineWidth',2);
        plot(time,xPlant(well,:),cc{well},'linestyle',':','LineWidth',2);

        box on
        set(gca,'FontSize',14)
        legend({'threshold','Estimated','Real'},'Location','northwest');

        ylim([0,nmpcConfig.x_threshold*1.5]);
        xlim([0,simLength]);
        xlabel('Time [day]','fontsize',14);
        ylabel('Erosion [mm]','fontsize',14);

        set(gcf,'color','w');

    end

    namePlot = [name,'_diag_results.pdf'];
    print(f3,'-r300','-dpdf',namePlot);


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Plotting --- statistics %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    f4 = figure(4);

  subplot(2,1,1)
        plot(time,flagControl,'marker','x','linestyle',':','markersize',5);

        ylim([-0.1,1.1]);
        yticks(0:1);
        yticklabels({'no','yes'});
        xlim([0,simLength]);
        xlabel('iteration [-]','fontsize',14);
        title('Control converged?','fontsize',14);
```

```
  subplot(2,1,2)
        plot(time,CPUtimeControl,'marker','x','linestyle',':','markersize',5);

        xlim([0,simLength]);
        xlabel('Iteration [-]');
        ylabel('exec. time [s]');
        title('Control time');

set(gcf,'color','w');

    namePlot = [name,'_stats.pdf'];
    print(f4,'-r300','-dpdf',namePlot);

    %%%%%%%%%%%%%%%%%%%%%%%%
    % Plotting --- slacks %
    %%%%%%%%%%%%%%%%%%%%%%%%
    f5 = figure(5);

    cmap = flip(gray(simLength));

    for well = 1:3
        subplot(3,1,well)
        hold on
        for ii = 1:simLength
            plot(ii:(ii + nmpcConfig.np - 1),slackControl{1,ii}(well,:),...
                'Color',cmap(ii,:),'LineWidth',0.75);
        end

        box on

        ylim([0,1]);
        xlim([0,simLength + nmpcConfig.np]);
        xticks(0:50:500);
        xlabel('Iteration [-]');
        ylabel('slack [mm]');
        title(['Well ',num2str(well)]);

    end


set(gcf,'color','w');

    namePlot = [name,'_slack.pdf'];
    print(f5,'-bestfit','-r300','-dpdf',namePlot);

    %%%%%%%%%%%%%%%%%%%%%%%%
    % Plotting --- inputs %
    %%%%%%%%%%%%%%%%%%%%%%%%
    f6 = figure(6);

    cmap = flip(gray(simLength));

    for well = 1:3
        subplot(3,1,well)
        hold on
        for ii = 1:simLength
            plot(ii:(ii + nmpcConfig.np - 1),inputPrediction{1,ii}...
                (well,:),'Color',cmap(ii,:),'LineWidth',0.75);
        end

        yline(nmpcConfig.umax,'k--','LineWidth',1);
```

```
        box on

        ylim([0,2.75]);
        xlim([0,simLength]);

        xlabel('Iteration [-]');
        ylabel('Gas lift rate [kg/s]');
        title(['Well ',num2str(well)]);

    end


set(gcf,'color','w');

    namePlot = [name,'_input_seq.pdf'];
    print(f6,'-bestfit','-r300','-dpdf',namePlot);

   %%
    %%%%%%%%%%%%%%%%%%%%%%%
    % Plotting --- OF     %
    %%%%%%%%%%%%%%%%%%%%%%%
    f7 = figure(7);

    % computing terms
    RtermArray = [];
    StermArray = [];
    for ii = 1:simLength
    % Slack
     temp1 = slackControl{1,ii};
    % Regularization
     temp2 = inputPrediction{1,ii} - [u0, inputPrediction{1,ii}(:,1:end - 1)];

        Rterm = 0;
        Sterm = 0;
        for kk = 1:nmpcConfig.np
            Rterm = Rterm + temp2(:,kk)'*nmpcConfig.R*temp2(:,kk);
            Sterm = Sterm + nmpcConfig.rho*sum(temp1(:,kk));
        end

         RtermArray = [RtermArray, Rterm];
         StermArray = [StermArray, Sterm];
    end

        plot(time,ofControl,'k','marker','d','linestyle','-','markersize',3);
        hold on
        plot(time(2:end),RtermArray,'r','marker','x','linestyle',':',...
            'markersize',3);
        plot(time(2:end),StermArray,'b','marker','o','linestyle','--',...
            'markersize',3);

        legend({'OF value','Reg. Term','Slack Term'})

        xlim([0,simLength]);
        xlabel('iteration [-]');
        ylabel('OF value [-]');


set(gcf,'color','w');

    namePlot = [name,'_obj_fun.pdf'];
    print(f7,'-bestfit','-r300','-dpdf',namePlot);
```

Salmon Yemane Ghebredngl

NTNU
Norwegian University of
Science and Technology