Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Iver Osnes

Recurrent Neural Networks and Nonlinear Model-based Predictive Control of an Oil Well with ESP

Master's thesis in Cybernetics and Robotics
Supervisor: Lars Struen Imsland
July 2020

NTNU
Norwegian University of Science and Technology

Iver Osnes

Recurrent Neural Networks and Nonlinear Model-based Predictive Control of an Oil Well with ESP

Master's thesis in Cybernetics and Robotics
Supervisor: Lars Struen Imsland
July 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

Process modeling and simulation is a crucial tool to gain a better understanding of nonlinear systems. Accurate models can be used for control purposes, but are often hard to obtain. In the field of petroleum engineering, parameters tend to change over time and some components are unknown. This makes it hard to both model and make prediction algorithms in order to yield optimal control. A possibility can be to resort to data-driven models when accurate models are missing. Earlier studies have shown that Echo State Networks (ESNs) are suitable for model recognition of complex dynamical systems with a black-box modeling approach. Such models are preferable since they have a built-in error term and are able to adapt if a plant is changing its behavior as time goes on. This project aims to create a data-driven model of an Electric Submersible Pump (ESP) based on an ESN approach. Further, a Nonlinear Model Predictive Controller (NMPC) is implemented to yield optimal control on the ESP using the obtained data-driven model as a prediction model. Using an NMPC gives the ability to control future behavior while satisfying a set of constraints. Results from this study showed that an NMPC with an ESN as a prediction model can deliver a satisfactory operation on the ESP. It also showed that ESNs are well suited for model identification of an ESP.

Sammendrag

Modellering og simulering er et avgjørende verktøy for å forstå komplekse ulineære systemer. Nøyaktige modeller er ofte nyttige i reguleringssammenhenger, men kan være vanskelige å anskaffe. Innenfor oljeindustrien har parametre en tendens til å forandre seg over tid og noen komponenter blir sett på som ukjente parametre. Dette skaper utfordringer både med tanke på modellering og å prediktere fremtidig oppførsel. Det å kunne prediktere fremtidig oppførsel til en prosess kan brukes til å optimalisere reguleringen. Å ty til datadrevne modeller kan være et alternativ når nøyaktige modeller er vanskelige å anskaffe. Tidligere studier har vist at echo state nettverk er godt egnet til å gjenkjenne komplekse dynamiske systemer ved hjelp av store mengder data. Slike modeller er ofte fortrukne siden de kan ta hensyn til at en prosess endrer seg over tid. Målet for dette prosjektet er å lage en datadreven modell av en elektrisk nedsenkbar pumpe ved hjelp av et echo state nettverk. Modellen skal videre bli brukt til å prediktere fremtidig oppførsel i en ulineær modell-prediktiv regulator for å optimalisere reguleringen til pumpen. Fordelen med en modell-prediktiv regulator er at den kan predikter fremtidig oppførsel i tillegg til tilfredsstille et sett med begrensninger på systemet. Dette prosjektet har vist at en modell-prediktiv regulator som bruker et echo state nettverk som preiksjonsmodell kan levere tilfredsstillende regulering av en elektrisk nedsenkbar pumpe. Prosjektet har også vist at echo state nettverk er godt egnet til å lage datadrevne modeller av pumpesystemet.

Preface

This master's dissertation is submitted as the final part of the requirements for the master's degree at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). This work has partly been conducted at Universidade Federal de Santa Catarina (UFSC) through a bilateral agreement between NTNU and UFSC.

First of all, I want to thank prof. Eduardo Camponogara for his support, advising, writing hints and for believing in me. It has been a challenging semester with unpredicted circumstances, but his support has been extraordinary.

I would also like to thank prof. Eric Aislan Antonelo for his assistance, programming help, and for sharing his great deal of knowledge inside the field of artificial intelligence. Further, I am grateful for the help I received from Jean P. Jordanou and Marco Aruélio de Aguiar.

I would also like to thank my supervisor at NTNU, prof. Lars Imsland for his assistance and for making this project possible.

I want to thank Sondre Bø Hernes for all the great memories in Brazil. It was sad that we had to leave Brazil way too soon, but I appreciate the time we had down there. I would also like to thank him for our collaboration in parts of the project.

Lastly, I would like to thank my family and friends. Their support has been a motivation throughout my years of study.

Iver Osnes

Ulsteinvik, July 2020

Contents

Figures

Tables

Acronyms

ANN Artificial Neural Network.

DAE Differential Algebraic Equation.

ESN Echo State Network.

ESP Electric Submersible Pump.

LQR Linear Quadratic Regulator.

MBC Model-Based Control.

MPC Model Predictive Control.

NLP Nonlinear programming.

NMPC Nonlinear Model Predictive Control.

OCP Optimal Control Problem.

ODE Ordinary Differential Equation.

OLS Ordinary Least Square.

RNN Recurrent Neural Network.

SQP Sequential Quadratic Programming.

Chapter 1

Introduction

1.1 Motivation

In this day and age, the world is going through a digital transformation where both industries and processes tend to get smarter. Various industries are more likely to have custom-made technology and equipment. Modeling based on first principles is more demanding than ever before and controlling a plant using on-line data can be excessively hard due to high complexity.

Model-Based Control (MBC) has been a powerful tool in systems where on-line data is hard to obtain. A controller can use a model based on first principles and prior knowledge to control a plant without the use of on-line data. However, the use of MBC does not come without drawbacks. If an assumed model deviates from the plant's behavior it could lead to poor control. For this reason, it is necessary to look for better alternatives since using MBC with an inaccurate model could lead to either poor performance or an unstable closed-loop system [1].

The petroleum industry is one of many industries that has undergone a significant technological transformation in recent years. Traditional MBC theory has in many cases become impractical due to the level of complexity, besides the demanding process of obtaining a model based on first principles. Another problem in this industry is that plants tend to change their behavior over time. Despite all drawbacks, the digital transformation comes with a large set of benefits. Modern processes generate huge amounts of data during its processing time. Data can be stored and later used to design controllers where accurate models are missing. A controller designed based on input/output data without involving physical information serve as a data-driven controller.

When it comes down to obtaining a data-driven model based on large amounts of data, different approaches are available. The studies in [1] suggest that artificial intelligence and machine learning is a good approach when it comes to providing accurate models without prior knowledge following a black-box modeling

approach. It is also stated in [2] that data-driven control is related to machine learning since both utilize a form of black-box system identification.

The project preceding this dissertation, [3], conducted a case study on how MPC could yield optimal control for a data-driven tank study. Results from this study showed that the MPC had promising capabilities in yielding optimal control by utilizing a data-driven model as a prediction model in the MPC. However, it also showed that an Artificial Neural Network (ANN) fits simple systems such as a SISO system, but tend to become too computational expensive when the process complexity is increasing and/or several state variables are introduced.

A widely used tool regarding black-box model identification is the Recurrent Neural Network (RNN). An RNN is able to reproduce both linear and nonlinear behavior of a plant if a sufficient training set is provided. However, RNNs tend to be hard to train because of nonlinearities in the training process and no guarantees of finding a global optimum. This work will include a complex nonlinear plant. It is therefore necessary to look at other more convenient ways to obtain a data-driven model.

A more suitable approach for this problem is the Echo State Network (ESN). An ESN is in theory an RNN where only output weights are adapted. This will make the training process much faster than other neural networks since it can be trained by means of linear regression. ESNs are widely known for their ability to recognize complex dynamical systems with a black-box modeling approach. In [4] an ESN was utilized to learn dynamical nonlinear behaviors for a downhole pressure estimation. It was also successful to model increasingly complex behaviors by showing examples of behaviors in [5].

Model Predictive Control (MPC) showed great potential in regards of performing optimal control in [3], which used a data-driven model to predict and optimize future process behavior in addition to handle constraints on both input and controlled variables. It has proven to be convenient in systems where some of the sections are data-driven, such as an autonomous system with forecasts as constraints in [6]. It has also shown its reliability in [7] when it delivered optimal control for a real system with a partly inaccurate model. The proposition in this dissertation will be to combine ESNs with Nonlinear Model Predictive Control (NMPC) to perform optimal control of a plant.

The plant introduced in this work is the Electric Submersible Pump (ESP) which is used to provide artificial lifting in an oil well. As much as 23% of all ESP failures are due to operator mistakes [8]. Using different MPC strategies to avoid failures on ESPs has been a prevalent field of study in recent years. A linear MPC strategy was successfully applied to an ESP in [9]. Further, [10] managed to implement an MPC based on linearized models of the ESP on a PLC. Different MPC strategies showed sufficient robustness with respect to the ESP's nonlinearities in [11]. This work will look into how an ESN could be utilized to provide a data-driven model of the ESP and then control it using an NMPC.

1.2 Objective

This dissertation will explore how data-driven models obtained from Echo State Networks could work as prediction models in an NMPC to control an electric submersible pump system. The work is divided into three main parts:

- A nonlinear system of the ESP is going to be implemented and simulated from a set of differential-algebraic equations. A large set of data will then be obtained from the simulation and later used as a training set for the ESN.
- An echo state network will be built and trained by the obtained data set. The goal of the echo state network is to identify the nonlinear ESP model.
- The obtained data-driven ESP model will then be used as a prediction model for an NMPC. The NMPC aims to perform optimal control of the ESP plant by utilizing the data-driven ESP model to predict future behavior.

This work is carried out as a proof of concept. A conclusion would be whether or not this approach could yield promising control for the electric submersible pump system.

1.3 Contribution

This dissertation has led to the following contributions:

- A Python environment suitable for testing and simulating complex systems on DAE form.
- A nonlinear model predictive controller that supports echo state networks as prediction models.

1.4 Structure

This dissertation consists of 6 chapters.

- *Chapter 2* will give an introduction to the relevant theory for this project. This includes an introduction to the electric submersible pump, basic system knowledge, system identification and model predictive control.
- *Chapter 3* aims to give an overview of how every aspect is implemented in this work. The chapter is covering modeling and simulation of ESPs, model identification and model predictive control.
- *Chapter 4* will carry out experiments and provide results.
- *Chapter 5* provides a discussion of the obtained results.
- *Chapter 6* presents the conclusion of this work and will also present proposals for future work.

Chapter 2

Theory

This chapter will give an introduction to the relevant theory used in this work. Firstly, a brief introduction to artificial lifting and electric submersible pumps (ESPs) will be given. Then, the relevant theory for both model identification and control of the ESP will be provided. Section 2.1 is written together with Sondre Bø Hernes as a part of our collaboration.

2.1 Oil wells and artificial lifting

In order to bring oil from a reservoir to the surface, enough pressure is essential. If a well has enough pressure to push fluid to the surface, we call it a flowing well. Flowing wells have a natural lift, which means that the pressure at the bottom of the well is strong enough to overcome the pressure loss through the pipeline on its way to the surface. However, most oil wells do not have enough pressure in their reservoir to rely on natural lift alone. Some wells might have a natural lift in their early years of production, but the pressure will decrease over the lifetime. A well with insufficient pressure will leave valuable hydrocarbons deposited in the reservoir.

To overcome the problem of non-flowing wells, it is common practice to resort to artificial lifting. Artificial lifting is a method used to increase the pressure inside the well to boost oil production and to increase the lifetime of the well. There are mainly two different methods of artificial lifting: gas lift and pumping systems. The choice of method depends on multiple variables such as the volume of the well, depth, location (onshore or offshore), viscosity of the fluid, concentration of gas, and the condition of the well [12]. A commonly used method for artificial lifting is the Electric Submersible Pump (ESP), which will be presented in the next section.

2.1.1 ESPs

An Electric Submersible Pump (ESP) is a multistage centrifugal pump installed several hundred meters under the sea surface in an oil well [13]. ESPs will contribute to a boost in production and increased recovery for a well. The pump inside the ESP works on a dynamic principle. Firstly, the kinetic energy of the liquid is increased. Then, it is partly converted into pressure energy which will move the fluid through the pump [12].

It is primarily used in oil wells with high flow rates because of its high cost. They are therefore limited to high volume applications either offshore or onshore where the high cost can be justified.

ESPs have greater lifting abilities than most other artificial lifting methods. A set of advantages and disadvantages for the ESP are discussed in [12].

Advantages:

- Suited for lifting high liquid volumes from medium depths.
- Efficient as long as the production is higher than 1000 bpd.
- Well working in deviated wells.
- Potentially low maintenance if properly designed and operated.
- Suited for offshore installations because of its low space requirements.

Disadvantages:

- Demand high electric power with high voltage.
- Low flexibility if it is run on a constant electrical frequency.
- Free gas at suction can harm the efficiency of the pump and even stop liquid production.
- Abrasive material as e.g. sand will increase the equipment wear.
- Expensive to purchase, repair and operate.
- High velocity will increase power usage and reduce productivity.

Lifespan

The lifespan of ESPs depends on multiple factors. The length of service is an important factor, but would not cause a failure alone. ESPs do not normally wear out, it is often a sudden catastrophic event that causes the failure. Temperature, flow rate, vibration and power consumption can all affect the lifespan of an ESP. It is therefore normal to set constraints on these variables to increase the lifespan. A replacement of an ESP will cause a huge economic impact due to the cost of the replacement pump and the loss of production [10].

Available statistics says that 23% of all ESP failures are due to operator mistakes. When ESPs were first introduced, this number was as high as 80% [8]. A set of

constraints has been introduced in later years to decrease the number of failures.

2.2 System fundamentals

A system is in general defined as the relationship between an input (excitation) and an output (response), where different inputs will generate unique outputs [14]. If a system has one input terminal and one output terminal it is called a single-input single-output (SISO) system. In comparison, a multivariable system has multiple input terminals and multiple output terminals, being referred to as a multiple-input multiple-output (MIMO) system. A simple representation of a SISO system can be seen in Figure 2.1.

A system that accepts continuous-time signals as input and generates continuous-time signals as output is referred to as a continuous-time system. The input are denoted as $u(t)$ and the output as $y(t)$. The time t has a range of $-\infty$ to ∞. Continuous-time dynamical models should be used if the signal involves spikes, like in biological modeling [15].

In contrast, a discrete-time system is a system that accepts discrete-time signals as input and generates discrete-time signals as output. Discrete-time signals are assumed to have a sampling period T, and the input and output can be denoted as $u[k]$ and $y[k]$ respectively, where k represents a discrete-time instant and has a range of $-\infty$ to ∞.

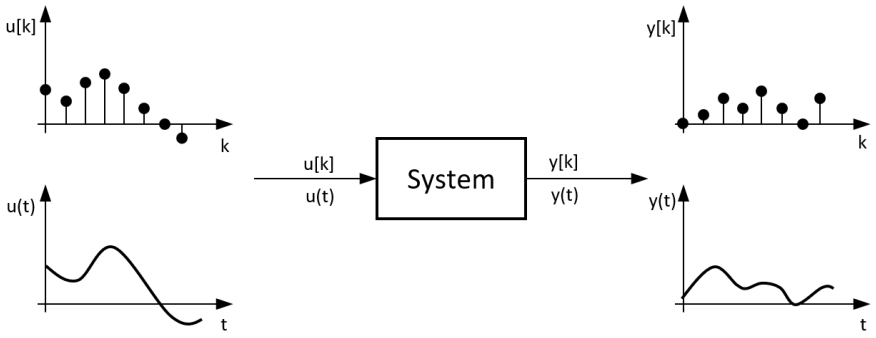


Figure 2.1: A simple SISO system, recreated from [14].

A system dependent on both current and previous inputs is referred to as a dynamical system [14]. In theory, a dynamical system is dependent on all previous inputs back to $t = -\infty$. This is in general both difficult and disadvantageous to trace. The idea of a state comes in handy regarding this problem. An initial state $x(t_0)$ of a system holds all information up till the time t_0. The state $x(t_0)$ combined with the input $u(t)$ for $t > t_0$ can generate the unique output $y(t)$ for all $t > t_0$ [14].

There are different ways to represent a dynamic system, one of the most common ways is the state-space representation. A state-space equation describes the internal variables in addition to the relationship between input and output. They are well suited for computer computation because it only consists of first-order derivatives in the discretization [14]. The state-space representation for a dynamical system can be expressed as:

$$\dot{x}(t) = f(x(t), u(t)) \tag{2.1a}$$
$$y(t) = h(x(t), u(t)) \tag{2.1b}$$

where Equation (2.1a) is the state equation and Equation (2.1b) is the output equation. A dynamic system has a kind of memory since the current state is dependent on previous states.

Similarly, if the continuous system is discretized with a discrete-time instant k, we can express the model as a system composed of first-order equations:

$$x[k+1] = f(x[k], u[k]) \tag{2.2a}$$
$$y[k] = h(x[k], u[k]) \tag{2.2b}$$

This work will use Differential Algebraic Equations (DAEs) to model dynamical systems. DAEs are Ordinary Differential Equations (ODEs) with additional algebraic constraints on the dynamic variables. The DAE standard form is given as [16]:

$$\dot{x}(t) = f(x, y, t) \tag{2.3a}$$
$$0 = g(x, y, t) \tag{2.3b}$$

2.2.1 Model identification

Models of real physical systems are essential in almost all disciplines. They can be useful for analyses and gaining a better understanding of a real system. Further, simulation can be run using a model that will be advantageous since simulating a real system can be both expensive and dangerous [17]. Such models can also be used to analyze the system over time and predict behavior with various inputs.

The process of building a mathematical model of dynamical systems from measured data is called model identification. It is common to divide different modeling approaches based on complexity and available data [2]. The three approaches can be seen in Figure 2.2 and are as follows:

- *White box models* are based on first principles such as physical, economical or chemical laws. Extrapolation is appropriate for these models and they are often both scalable and very reliable. A model is called a white box model if its parameters possess an interpretation in first principles.

- *Black box models* are mainly based on experiments and data. In the building of a black box model, no prior knowledge is needed. This model is only for existing processes and the parameters have no relationship to first principles.
- *Gray box models* are a combination of white and black box models. Normally, the model structure relies on prior knowledge while the parameters are obtained through measured data.

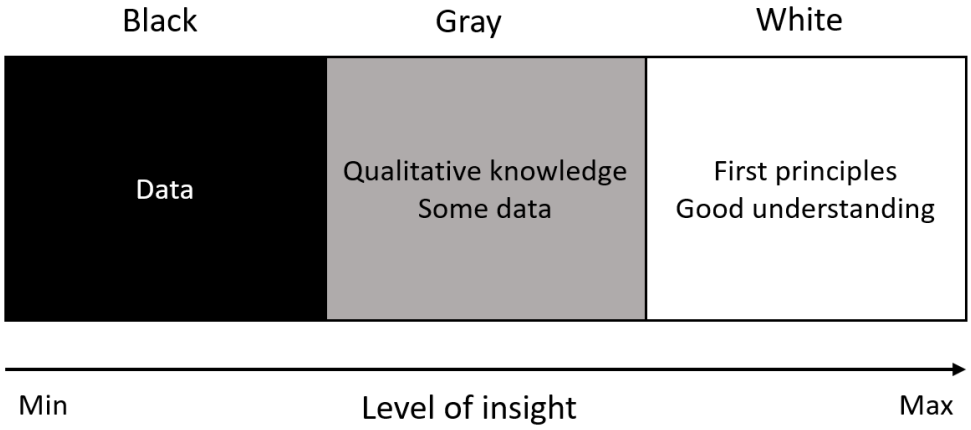


Figure 2.2: The different modeling approaches.

The quality of the model will usually determine the upper limit of the final solution and are often looked at as the bottleneck in the development of a whole system [2].

2.3 Artificial Neural Networks / deep learning

In the early days, artificial intelligence was a group of problems that was excessively hard to solve for human brains but easy to solve for computers. These problems were often described by a list of mathematical rules [18]. Later, it became clear that the real challenge was to make a computer perform tasks that were easy for humans, but hard to describe mathematically. The type of problems could be things that humans solved automatically or intuitively, like an image or speech recognition [18].

One of the simplest examples of a deep learning model is the feedforward neural network, also called a multilayer perceptron (MLP). It can be seen as a function that will map a set of inputs to a set of outputs. The mathematical function is composed of many simpler functions. A simple representation of a feedforward deep network can be seen in Figure 2.3.

An input $\mathbf{x}$ flows from the input layer, through the computation layer (hidden layer) defined by a function f, before reaching the output $\mathbf{y}$. The network aims to

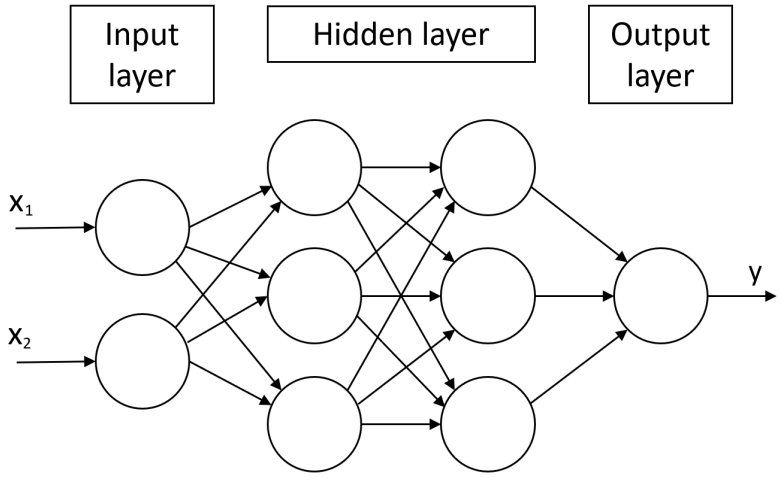


Figure 2.3: A simple feedforward deep network representation.

approximate a function f^* such that $y = f^*(x)$. To gain a better understanding of the architecture in the network, a simple example of a SISO neuron is shown in Figure 2.4.

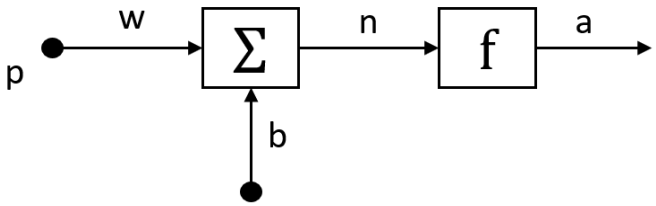


Figure 2.4: Network architecture and notation, extracted from [3].

In the figure above, p represents a scalar input. The input p is multiplied by a weight w before it enters a summation with a bias, b. Further, the net input n will proceed into an activation function f. The output from the activation function f is a scalar output a. This simple neuron can be described by:

$$a = f(n) = f(wp + b) \tag{2.4}$$

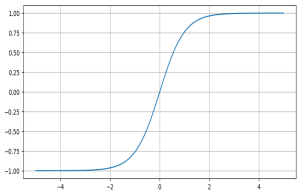
An activation function makes it possible to make sense of nonlinear complex mappings between input and output. The output would have been a simple linear function without the activation function. A linear function is of course easier to solve, but it discards the possibilities of learning complex mappings between input and output. A network without hidden layers and an activation function can be characterized as a linear regression model with limited power [19]. Three frequently used activation functions are expressed in Equation (2.5) with their following

plots in Figure 2.5.

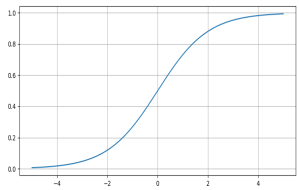
$$\tanh(x) = \frac{2}{1+e^{-2x}} - 1 \tag{2.5a}$$

$$\text{sig}(x) = \frac{1}{1+e^{-x}} \tag{2.5b}$$

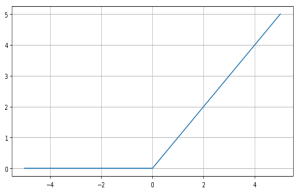
$$\text{ReLU}(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{2.5c}$$



(a) Tanh function.



(b) Sigmoid function.



(c) ReLU function.

Figure 2.5: Different types of activation functions.

2.3.1 RNN

If feedback is included in a neural network we call it a Recurrent Neural Network (RNN) [18]. All RNNs have at least one cyclic path between its neurons. RNNs are neural networks specialized in processing a sequence of values $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(\tau)}$. Adding feedback from the previous step will give the network a kind of memory. This will increase the network's ability to learn systems with dynamical characteristics [20]. An RNN on its general form can be written as:

$$\mathbf{x}[k+1] = f(\mathbf{W}_h x[k] + \mathbf{W}_i \mathbf{u}[k]), \tag{2.6}$$

$$\tilde{\mathbf{y}}[k+1] = \mathbf{W}_o \mathbf{x}[k+1], \tag{2.7}$$

where $\mathbf{x}[k]$ is the state of the hidden neurons, $\tilde{\mathbf{y}}[k]$ is the predicted output and $\mathbf{u}[k]$ is the input. Further, f is the activation function and $\mathbf{W}_h, \mathbf{W}_i$ and $\mathbf{W}_o$ are the weighting matrices in the hidden layer, input layer and output layer respectively. For traditional RNNs, during the training process, Back-Propagation Through Time (BPTT) is used to adapt the weights to minimize a quadratic error over time [18]. It is harder to train an RNN compared to a static network because RNNs are time-dependent. Besides, there is no guarantee for global convergence and it could have problems with a vanishing gradient [21].

2.3.2 Reservoir Computing

Despite the fact that RNNs had the ability to learn systems with dynamical characteristics, [22] showed that for the traditional training methods for RNNs, where all weights were adapted, significant weight adaptions were only made in the output layer. Reservoir Computing (RC) was introduced as an RNN-based framework where input data are transformed into spatiotemporal patterns by an RNN in a reservoir [23]. The main idea behind RC is that both input weights and the weights inside the reservoir are randomly generated and not trained, while the output weights are trained with a learning algorithm. This approach gives an advantage of a faster training process and less computational cost compared to traditional RNNs [15].

2.4 Echo State Network (ESN)

An Echo State Network (ESN) is a particular form of the RNN which goes under the definition of reservoir computing. It is well suited for model identification of complex dynamic systems. As previously mentioned, all RNNs have a kind of memory, and this is crucial for modelling history dependent systems. An ESN is built up by three components: an input layer, a reservoir (hidden layer) and a readout output layer. The ESN can be described by the following discrete-time dynamic equations:

$$x[k+1] = (1-\alpha)x[k] + \alpha f(\mathbf{W}_r^r x[k] + \mathbf{W}_i^r u[k] + \mathbf{W}_b^r + \mathbf{W}_o^r y[k]), \tag{2.8}$$

$$y[k+1] = \mathbf{W}_r^o x[k+1], \tag{2.9}$$

where $\mathbf{x}[k]$ is the state of the reservoir neurons at time instant k. Further, $\mathbf{u}[k]$ and $\mathbf{y}[k]$ are the current values of input and output neurons respectively. $\mathbf{W}$ represents weighting matrices, and are given on the form $\mathbf{W}_{from}^{to}$, where r is the reservoir, i is the input layer, o is the output and b is the bias. α is the leak rate and $f = \tanh(\cdot)$ is the nonlinear activation function [24].

Unlike the RNN, both input weights and reservoir weights are randomly initialized and will stay fixed throughout the whole process. The output weights are the only adaptable weights and can be trained with e.g. linear regression [25]. A representation of the ESN can be seen in Figure 2.6. Dotted arrows represent fixed weights while solid arrows represent adaptable weights. The grey circles represent neurons.

In order to understand how the ESN works, it is important to see its intuition. The overall goal of the ESN is to learn an inverse model based on a black-box modeling approach. A training sequence with input $u[k] \in \mathbb{R}^{N_u}$ and output $y^{\text{target}}[k] \in \mathbb{R}^{N_y}$ is given, where $k = 1, ..., T$ is the discrete-time instant. The aim of the ESN is to learn a model by predicting the output $y[k] \in \mathbb{R}^{N_y}$ such that the error between $y[k]$ and $y^{\text{target}}[k]$ is minimized.

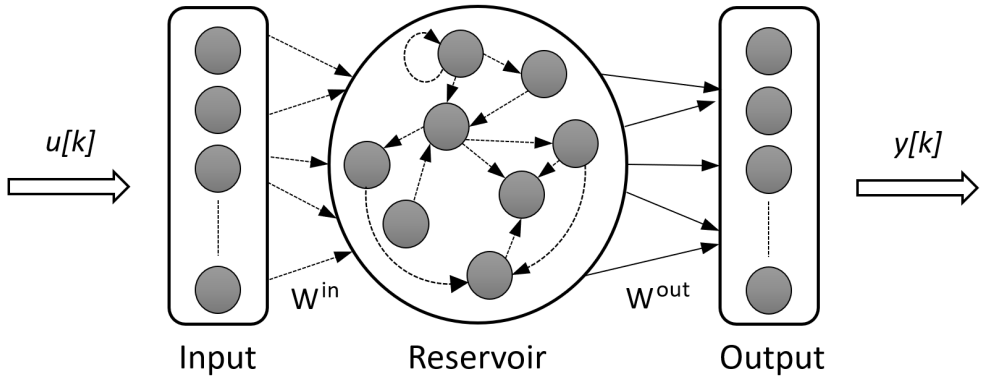


Figure 2.6: The structure of an echo state network, inspired by [15].

The process of building an ESN in its simplest form can be divided into the following steps [15]:

1. Generate the weights W_i^r and W_r^r to form a large random RNN which will function as a reservoir, and rescale its weights according to specific criteria;
2. Run the reservoir using a training input sequence $u[k]$ and collect the corresponding states $x[k]$;
3. Compute the output weights. This can be done using linear regression by minimizing the mean square error (MSE) between $y^{\text{target}}[k]$ and $y[k]$;
4. Test the network with a new set of input data $u[k]$ and generate the output $y[k]$ by utilizing the trained output weights.

2.4.1 Selection of global parameters

There are no particular recipes when it comes to selecting global parameters of the reservoir, a.k.a. hyperparameters, but there are a handful of precautions that must be considered in the process. Parameters are usually found by a trial and error approach. It also exists software packages with a grid search or other automated search methods that will ease the process. The following parameters should be considered when building an ESN.

Reservoir size

The first parameter is the reservoir size. The number of neurons N in the reservoir is equal to the dimension of $x[k]$. It should be higher than the number of network inputs and could be as big as computationally feasible. A big reservoir will have more memory, be more likely to generate rich enough signals and perform better as long as the learning is regularized. The drawback with a large reservoir is that the training time increases quadratically [15].

Sparsity

The sparsity in the ESN narrates the distribution of non-zero elements in the reservoir. Introducing sparsity in the connection between weights will reduce the computational cost. Sparsity has shown to be important regarding spiking neural networks such as Liquid State Machines but is in general not important for analog networks such as the ESN other than for computational reasons [15].

Leak rate

The leak rate is a percentage of how much of the current state that will be transmitted to the next state [15]. A low leak rate will slow down the reservoir dynamics and increase memory. Since ESNs are lacking a time constant, leaky integrator neurons are used to slow down dynamics in for instance a differential equation [15].

Spectral radius

Adapting the spectral radius is equivalent to scaling the reservoir weights. Scaling of the reservoir weights will influence the dynamical behavior. Setting the spectral radius to $\rho(\mathbf{W}) < 1$, with $\mathbf{W} = \mathbf{W}_r^r$, will guarantee the Echo State Property (ESP) in most cases, which is a stability property of the ESN to be satisfied. High values for $\rho(\mathbf{W})$ will increase the reservoir memory and induce more non-linearity, while low values will make the reservoir more responsive to recent inputs [15].

Input scaling

The input scaling of $\mathbf{W}^{in}$ is another parameter that should be optimized for better performance. A small input scaling should be used if the task is linear. This will allow the reservoir to work in the linear region of the activation function. A large input scaling will make the activation more nonlinear [15].

2.4.2 Training

In neural networks, training is referred to as the process of minimizing the error between a given output $y^{\text{target}}[k]$ and the output produced by the network $y[k]$. While all weights are trained using backpropagation through time (BPTT) in classical RNNs, the only weights adapted in ESNs are the output weights $\mathbf{W}_r^o$. This will result in a linear optimization problem since the loss function can be expressed as a sum of squares and the error between targeted output and model output is linear [2].

A training sequence with input $u[k] \in \mathbb{R}^{N_u}$ and output $y^{\text{target}}[k] \in \mathbb{R}^{N_y}$ for $k = 1, ..., T$ is fed into the network. The input and output can be formulated as $\mathbf{U} \in \mathbb{R}^{N_u \times T}$ and $\mathbf{Y}^{\text{target}} \in \mathbb{R}^{N_y \times T}$. The input matrix $\mathbf{U}$ is iterated through Equation (2.8) and the internal states $x[k]$ are harvested into a new state-collection matrix $\mathbf{X} \in$

$\mathbb{R}^{N_x \times T}$. The state-collection matrix and the targeted output matrix is formulated as:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & \cdots & x_{1N_x} \\ x_{21} & x_{22} & \cdots & \cdots & x_{2N_x} \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ x_{T1} & x_{T2} & \cdots & \cdots & x_{TN_x} \end{bmatrix}, \quad \mathbf{Y}^{\text{target}} = \begin{bmatrix} y_{11} & y_{12} & \cdots & \cdots & y_{1N_x} \\ y_{21} & y_{22} & \cdots & \cdots & y_{2N_x} \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ y_{T1} & y_{T2} & \cdots & \cdots & y_{TN_x} \end{bmatrix}$$

When the states from the training set are harvested into $\mathbf{X}$, it is time to train the readout. This is done by tuning the output weighting matrix $\mathbf{W}_r^o$. Having a linear optimization problem is highly desirable in a training process since it comes with a set of features such as an analytic one-shot solution, a unique optimum, and a recursive formulation [2]. Multiple powerful and efficient techniques are available, such as the Ordinary Least Square (OLS) regression method and the Ridge regression method [2].

OLS is a frequently used method when it comes to estimating parameters with linear regression. This method is used to minimize the cost function:

$$J = \sum_{k=0}^{N} \left\| y^{\text{target}}[k] - y[k] \right\|_2^2 = \left\| \mathbf{Y}^{\text{target}} - \mathbf{W}_r^o \mathbf{X} \right\|_2^2 \tag{2.10}$$

where N is the number of samples in the training data. It is clear that this problem would have a global minimum because of its linearity. An analytical solution to this problem can be formulated by isolating $\mathbf{W}_r^o$:

$$\mathbf{W}_r^o = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}^{\text{target}} \tag{2.11}$$

A weakness of the OLS method is that the problem of poorly conditioned Hessians becomes harsh if the number of regressors is large [2]. That is, a flexible model will have a severe variance of the worst estimated parameters. Neural networks are known for utilizing a large number of parameters, and this will introduce a serious variance problem. This problem can be solved using regularization techniques.

Ridge regression is one of the approaches that introduce a regularization parameter by multiplying a regularization parameter λ with the ℓ_2 norm of $\mathbf{W}_r^o$. The intuition behind this is simple, parameters that are negligible for solving the optimization problem are driven towards zero [2]. A $\lambda \to 0$ will result in a standard least square problem, while $\lambda \to \infty$ will force all parameters to zero [2]. Equation (2.11) with a regularization parameter leads to the following parameter estimate:

$$\mathbf{W}_r^o = (\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathbf{Y}^{\text{target}} \tag{2.12}$$

The training of the network is finished when the optimal $\mathbf{W}_r^o$ is found. The output weights should then be implemented in the network, which is now ready to use with Equation (2.8) and Equation (2.9).

2.5 Model Predictive Control (MPC)

This section will introduce the MPC and discuss its intuition. Firstly, a brief introduction to optimization will be given. Some theory in this section is extracted from the project preceding this dissertation [3].

2.5.1 Optimization

In control theory, optimization is the procedure of minimizing or maximizing an objective function on a finite set of feasible solutions. In order to take advantage of optimization, an objective must be defined. The objective should be a single number which works as a quantitative number of the performance of the system [26]. An objective should also be dependent on the system variables. Optimal values of the variables must be found in order to optimize the objective. A general notation of an optimization problem can be expressed as [26]:

- x is the variable vector;
- f is the objective function which is the function we want to maximize or minimize;
- c_i are functions containing constraints. Constraints are functions of x that define equalities and inequalities which the vector x must satisfy.

By using the notation above, the optimization problem can be written as:

$$\min f(x) \quad \text{subject to} \begin{cases} c_i(x) = 0, i \in \mathscr{E} \\ c_i(x) \geq 0, i \in \mathscr{I} \end{cases} \tag{2.13}$$

where $\mathscr{E}$ is a set of equality constraints and $\mathscr{I}$ is a set of inequality constraints. The solving algorithm of an optimization problem depends on its objective function and constraints. Optimization problems are divided into three general classes. The classes with their respective solving algorithm are:

- *Linear Programming* (LP): Both constraints and the objective function are linear. The linearity will result in a convex problem. The Simplex method is a widely used algorithm for LP problems.
- *Quadratic Programming* (QP): This problem has a quadratic objective function while the constraints are linear. An active set method is suitable for solving QP problems. This problem is also convex.
- *Nonlinear Programming* (NLP): NLP problems are known by their nonlinear equality constraint functions. A Sequential Quadratic Programming (SQP) algorithm is suitable for these problems. NLP problems are non-convex.

This work will focus on NLP problems since the ESP system is nonlinear.

2.5.2 Forming an NLP problem

There exist multiple methods in how NMPC optimization problems are solved. Most of them are, however, based on Sequential Quadratic Programming (SQP) methods as aforementioned. The SQP methods are iterative, which means that it makes a quadratic approximation to the objective function and a linear approximation to the constraints. Then it solves a QP problem at each iteration in order to find the search direction. A line search is then used to solve the QP problem and find the next iterate [27].

How the QP problem is formulated can play a big role regarding calculation cost. The three primarily used methods are [27]:

- *Single shooting:* This method is also known as a *sequential approach*. Firstly, the states are removed by forward simulation such that the optimization variables only consists of discretized controls $u_0, u_1, ..., u_L$. Then a reduced QP problem is solved. The advantages of this method are that there are few optimization variables and that the solution will be feasible with respect to the model at each SQP iteration. It is also the most intuitive method.
- *Simultaneous approach:* This approach implements the model as explicit equality constraints. It will result in a large number of optimization variables since both controls and states are optimized. This approach demand high computational cost.
- *Multiple shooting:* This method is a composite of the two other approaches. What makes this method special is that the control horizon is divided into sub-horizons. Significantly faster than single shooting if the prediction horizon is long.

This work will focus on single shooting because of its intuitiveness. Despite the fact that multiple shooting is a faster method, only small prediction horizons will be necessary in this project.

2.5.3 Building an MPC

Model Predictive Control (MPC) refers to a class of control algorithms which by utilizing an explicitly formulated process model can predict and optimize future behaviors of a process [28]. MPC controllers can handle constraints on both manipulated variables (input) and states/controlled variables. In general, an MPC can be described as a controller which [27]:

- utilize a process model to predict future behavior;
- optimize future behavior by using a class of control algorithms;
- can handle constraints on both controlled and manipulated variables.

There are several variants of the MPC. The functionality depends on how the objective function is defined and what kind of prediction model is used. This work

will look at how a data-driven model can be used as a prediction model.

Objective function

A general way to formulate an optimal control problem is by making a minimization problem that punishes deviation in both states and controls. An objective function can be formulated as:

$$J = \sum_{k=0}^{\infty} \left(\left\| x[k] - x[k]^{ref} \right\|_Q^2 + \left\| u[k] - u[k]^{ref} \right\|_R^2 \right) \tag{2.14}$$

where x is a vector of future states and u is a vector of future inputs. x^{ref} and u^{ref} is the desired values. Q and R are weighting matrices on states and control respectively, which are typically symmetric and positive definite.

Equation (2.14) is an infinite horizon objective function since it takes all future steps into account. This is a typical approach in controllers without constraints such as the Linear Quadratic Regulator (LQR). An MPC utilizes constraints in the optimization problem which means that a finite horizon formulation must be used. This can be expressed as:

$$J = \sum_{k=0}^{N-1} \left(\left\| x[k] - x[k]^{ref} \right\|_Q^2 + \left\| u[k] - u[k]^{ref} \right\|_R^2 \right) \tag{2.15}$$

where N is the prediction horizon. The parameters Q, R and N are the main parameters for tuning. An increase of Q will lead to more aggressive closed-loop behavior, and an increase in R will lead to less aggressive behavior. The prediction horizon N is related to both performance and computational complexity. A short N will give a low complexity optimization problem with lacking performance, while a large N will give better performance but also a higher computational cost. N is normally selected as large as computational limitations permit, as this will make the closed-loop behave close to an infinite horizon controller [27]. All tuning parameters should be tuned by a trial-and-error approach.

This work will utilize a data-driven model as a prediction model. This can lead to a model-plant mismatch (due to accuracy) and it is therefore important to not choose a too long prediction horizon. That is because uncertainties tend to be amplified as one predicts far into the future with an imperfect model.

Constraints

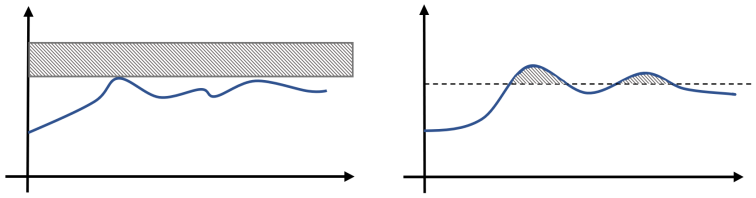
In practice, all processes are bounded by some kind of limits. In control theory, control valves have a range of action, pressure in a tank is constrained for safety and operational constraints can be introduced for economic or environmental reasons.

Output constraints must be considered in advance since the output variables are affected by the process dynamics. Input constraints are often kept inside a window limited by the manipulated variables properties (e.g., a valve or an actuator). The constraints are formulated as functions of the control inputs (manipulated variables) and the states (controlled variables):

$$u_{min} \leq u[k+i] \leq u_{max}, \quad i = 0, 1, ..., N-1 \tag{2.16}$$
$$x_{min} \leq x[k+i] \leq x_{max}, \quad i = 0, 1, ..., N \tag{2.17}$$
$$\Delta u_{min} \leq \Delta u[k+i] \leq \Delta u_{max}, \quad i = 1, 2, ..., N-1 \tag{2.18}$$

where $\Delta u = u[k+1] - u[k]$. That is, punishing the change of control.

Constraints are divided into two categories, soft and hard constraints. The difference between the two types is how they are treated by the MPC. An illustration of the two constraints can be seen in Figure 2.7. The left figure represents a hard constraint where a violation will cause infeasibility. On the left, a soft constraint (dotted lines) is represented. A violation will not cause infeasibility but is penalized in the cost function.



(a) Hard constraint. **(b)** Soft constraint.

Figure 2.7: Difference between hard and soft constraints.

Optimal control problem (OCP)

An Optimal Control Problem (OCP) can be obtained by combining the objective function with the constraints. An OCP on its general form can be written as:

$$\min_{u} \quad \mathrm{J}(x_0, u) \quad \text{subject to:} \begin{cases} x[k+1] = f(x[k], u[k]) \\ x[0] = x_0 \\ u[k] \in \mathrm{U}, \forall k \in \{0, ..., N-1\} \\ x[k] \in \mathrm{X}, \forall k \in \{0, ..., N\}, \end{cases} \tag{2.19}$$

where J is the objective function. There is not much of a difference between a linear MPC and a nonlinear MPC (NMPC) other than the process model being a nonlinear model. The process model is implemented as a constraint in the MPC and can be seen at the top of Equation (2.19). As aforementioned, having a nonlinear equality constraint will result in a Nonlinear programming (NLP) problem.

The MPC principle

Future outputs are predicted at each discrete-time instant k for a horizon N using a process model. An optimal future control sequence is obtained by solving an OCP at every discrete-time instant k. The first control instant in the sequence ($u[k]$, $u[k+1]$, ... ,$u[k+N]$) is then applied to the plant and the prediction model, while the rest of the sequence is rejected. New future outputs are then predicted at $k+1$ where a new optimal future control sequence is obtained. The MPC principle is illustrated in Figure 2.8.

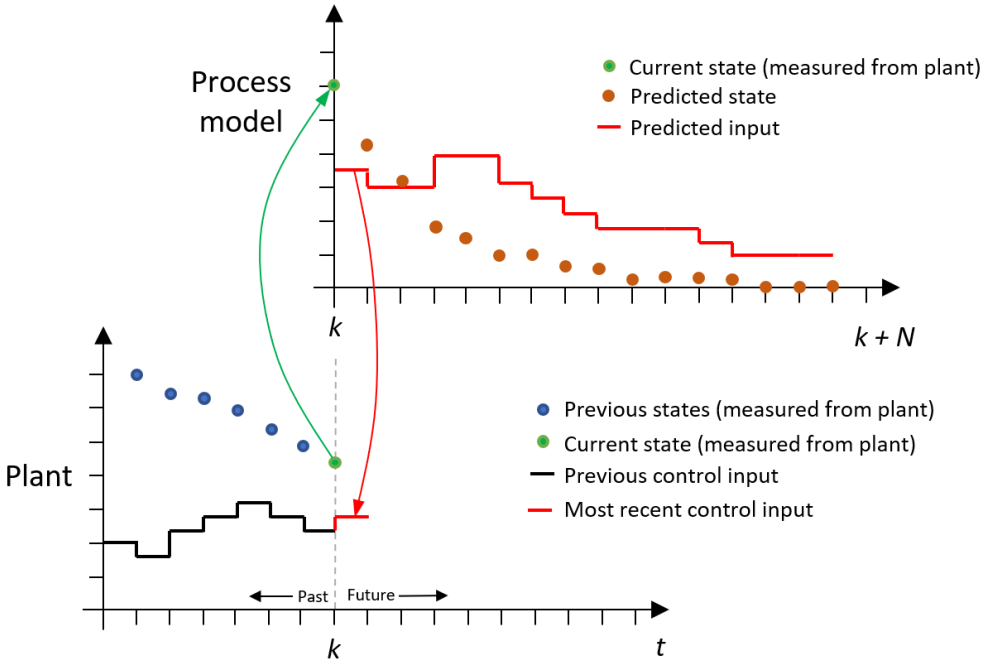


Figure 2.8: MPC principle, recreated from [29].

An MPC is often a safe controller choice when future references are given. It is an intuitive concept with a simple tuning process. For this reason, the MPC is an attractive choice for staff with limited control understanding.

Chapter 3

Implementation

This chapter aims to give an overview over the implementation in this work. It is divided into three parts. Firstly, a model of the ESP is implemented and tested. Section 3.2 is a cooperation together with Sondre Bø Hernes. The next part will discuss how the data-driven model is obtained from an ESN. At last, an NMPC is implemented to control the ESP model by utilizing the data-driven ESN model as a predictor.

3.1 Software

This work is primarily built in Python3. Python is great for prototyping and has one of the best programming communities in the world. It is also an obvious choice for machine learning tasks because of its well-developed libraries and simple approach.

Packages such as *numpy* and *scipy* were fundamental in all parts of the project. All results are displayed with *matplotlib*. Further, the library *esn_pnmpc* developed in [30] was used in the process of training and building the ESN. Lastly, the library *Oger* [31] was applied for grid searching global parameters in the ESN.

For the NMPC, the open-source tool *CasADi* [32] was used. CasADi is a tool for nonlinear optimization and algorithmic differentiation [32]. It is based on a symbolic framework that gives the flexibility of a programming language and the performance of a modeling language.

3.2 DAE modeling of wells with ESPs

The mathematical model of the system is based on a model developed by Statoil (Equinor) in [13], and additional equations from [33] are added to include viscosity. The system model consists of an Electric Submersible Pump (ESP) and a production choke valve. Perfect system knowledge is assumed for the model. The

simulator is implemented in Python with CasADi as a tool for solving DAEs. This dynamic model would work as a foundation for the development of further control and optimization strategies. A schematic picture of the model can be seen in Figure 3.1 and a description of the associated variables in Table 3.1.

The principles in this system is fairly simple. A mixture of liquid (oil, water and possibly gas) is flowing into the well from the reservoir (q_r). It will reach the ESP pump which will generate additional pressure and then raise to the production choke at the top of the well. An operator can control the ESP speed and production choke opening to reach a desired production or optimization target. The model assumes constant fluid properties to avoid an overly complex controller. Additional constraints are added to increase the lifespan of the ESP [13].

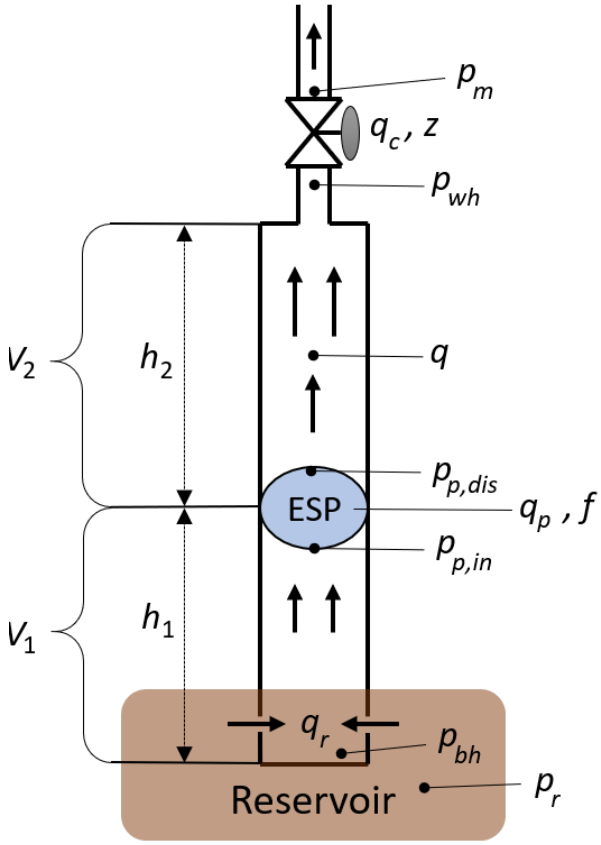


Figure 3.1: ESP lifted well, recreated from [33].

3.2.1 Model equations and parameters

The model of the ESP is divided into reservoir inflow, production pipe volumes, ESP and production choke. Despite excluding complexity such as effects due to gas and change of viscosity, the model will still represent the well dynamics quite

Table 3.1: Model variables

Control inputs	
f	ESP frequency
z	Choke valve opening

ESP data	
p_m	Production manifold pressure
p_{wh}	Wellhead pressure
p_{bh}	Bottomhole pressure
$p_{p,in}$	ESP intakepressure
$p_{p,dis}$	ESP discharge pressure
p_r	Reservoir pressure

Parameters from fluid analysis and well tests	
q	Average liquid flow rate
q_r	Flow rate from reservoir into the well
q_c	Flow rate through production choke

accurately [13]. The system has three states: bottomhole pressure p_{bh}, wellhead pressure p_{wh} and average flow rate q. Their differential equations are as follows:

$$\dot{p}_{bh} = \frac{\beta_1}{V_1}(q_r - q) \tag{3.1a}$$

$$\dot{p}_{wh} = \frac{\beta_2}{V_2}(q - q_c) \tag{3.1b}$$

$$\dot{q} = \frac{1}{M}(p_{bh} - p_{wh} - \rho g h_w - \Delta p_f + \Delta P_p) \tag{3.1c}$$

where Δp_f are the pressure loss due to friction and ΔP_p are pressure loss due to the ESP dynamics. The differential equations comes with a set of constraints, which can be described as the following algebraic equations:
Flow:

$$q_r = PI(p_r - p_{bh}) \tag{3.2a}$$

$$q_c = C_c \sqrt{p_{wh} - p_m}\, z \tag{3.2b}$$

Friction:

$$\Delta p_f = F_1 + F_2 \tag{3.3a}$$

$$F_i = 0.158 \frac{\rho L_i q^2}{D_i A_i^2} \left(\frac{\mu}{\rho D_i q} \right)^{\frac{1}{4}} \tag{3.3b}$$

ESP:

$$\Delta p_p = \rho g H \tag{3.4a}$$

$$H = C_H(\mu)\left(c_0 + c_1\left(\frac{q}{C_Q(\mu)}\,\frac{f_0}{f}\right) - c_2\left(\frac{q}{C_Q(\mu)}\,\frac{f_0}{f}\right)^2\left(\frac{f}{f_0}\right)^2\right) \tag{3.4b}$$

$$c_0 = 9.5970 \cdot 10^2 \tag{3.4c}$$

$$c_1 = 7.4959 \cdot 10^3 \tag{3.4d}$$

$$c_2 = 1.2454 \cdot 10^6 \tag{3.4e}$$

The parameters used in this model are based on the parameters from [33]. Parameters are given in Table 3.2 and consist of fixed parameters such as well dimensions and ESP parameters, and parameters found from analysis of fluid such as bulk modulus β_i and density ρ [33]. Parameters such as the well productivity index PI, viscosity μ and manifold pressure p_m are assumed constant in this project.

Table 3.2: Model parameters

Well dimensions and other known constants			
g	Gravitational acceleration constant	9.81	m/s^2
C_c	Choke valve constant	$2 \cdot 10^{-5}$	*
A_1	Cross-section area of pipe below ESP	0.008107	m^2
A_2	Cross-section area of pipe above ESP	0.008107	m^2
D_1	Pipe diameter below ESP	0.1016	m
D_2	Pipe diameter above ESP	0.1016	m
h_1	Height from reservoir to ESP	200	m
h_w	Total vertical distance in well	1000	m
L_1	Length from reservoir to ESP	500	m
L_2	Length from ESP to choke	1200	m
V_1	Pipe volume below ESP	4.054	m^3
V_2	Pipe volume above ESP	9.729	m^3

ESP data			
f_0	ESP characteristics reference freq.	60	Hz
I_{np}	ESP motor nameplate current	65	A
P_{np}	ESP motor nameplate power	$1.625 \cdot 10^5$	W

Parameters from fluid analysis and well tests			
β_1	Bulk modulus below ESP	$1.5 \cdot 10^9$	Pa
β_2	Bulk modulus below ESP	$1.5 \cdot 10^9$	Pa
M	Fluid inertia parameter	$1.992 \cdot 10^8$	kg/m^4
ρ	Density of produced fluid	950	kg/m^3
P_r	Reservoir pressure	$1.26 \cdot 10^7$	Pa

Parameters assumed to be constant			
PI	Well productivity index	$2.32 \cdot 10^{-9}$	m^3/s/Pa
μ	Viscosity of produced fluid	0.025	$Pa \cdot s$
P_m	Manifold pressure	20	Pa

3.2.2 Simulation and validation

In order to use the obtained model from Section 3.2.1 for further development, it has to be verified. The verification process is based on comparing the model's implementation and its associated data with the description and specification given by the developer. In this particular case, verifying the model means to carry out simulation studies in order to see if it responds as intended.

Multiple tests with different input values are simulated in order to see the response from the system. All simulations are using the same initial values:

$$P_{bh0} = 70 \text{ bar}$$
$$P_{wh0} = 30 \text{ bar}$$
$$q_0 = 36 \ m^3/h$$

Constant valve opening and pump frequency

The first simulation will look at how the ESP model responds to a constant valve opening z and pump frequency f. A desirable behavior for this simulation is that the system will reach a steady-state and remains stable. The input values are given as:

$$z = 100 \ \%$$
$$f = 53 \text{ Hz}$$

Figure 3.2 shows how the system responded to the constant input. Notice how the system reaches a steady-state in about two minutes. This is a desirable behavior when the system is kept with constant inputs. The displayed simulation was also used to find suitable initial values for the states, which further provided the basis for further simulation.

Step response on valve opening

The second simulation disclosed how the system would respond to a step response on the valve opening while the pump frequency was held constant. The initial values were given as:

$$z = 50 \ \%$$
$$f = 50 \text{ Hz}$$

The valve opening z was then increased after 5 minutes, resulting in the following input values:

$$z = 100 \ \%$$
$$f = 50 \text{ Hz}$$

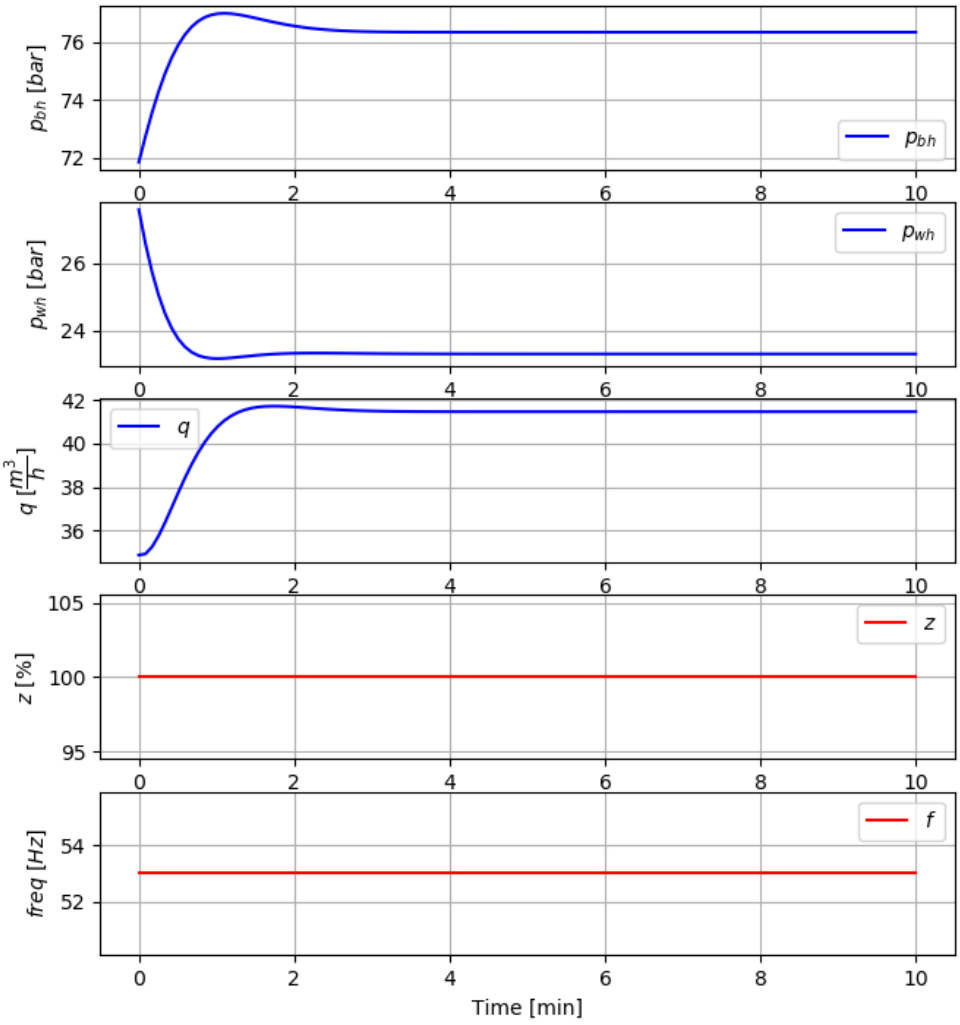


Figure 3.2: System response with constant input.

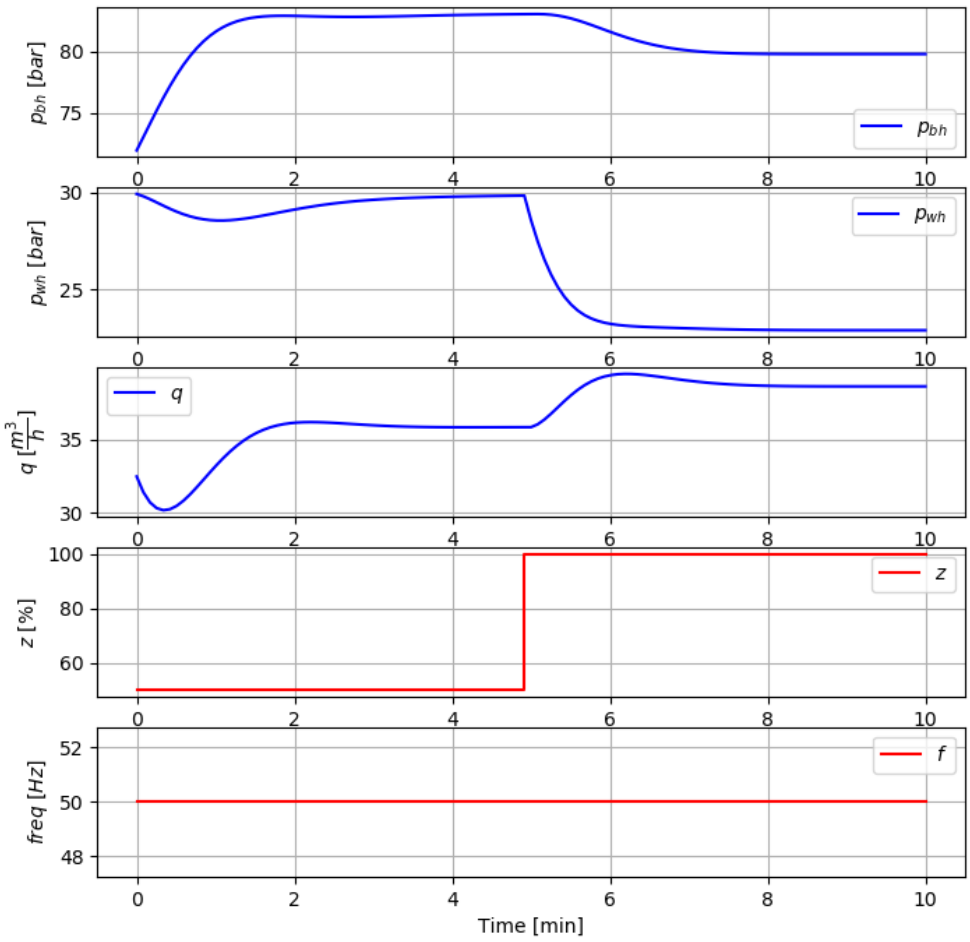


Figure 3.3: System response to a step response on the valve, starting at 50% and increasing to 100%.

The resulting plot can be seen in Figure 3.3. By the intuition of the system, it is natural that the liquid flow q will increase when the opening of the production choke valve is increased. Further, since more flow will lead to less pressure, it is natural that both the bottomhole pressure p_{bh} and the wellhead pressure p_{wh} will drop. The wellhead pressure will experience the highest drop rate since it is coupled with the production choke valve.

Increasing pump frequency

The last simulation looked into how the system responded to a change in the pump frequency. It is done by keeping the valve opening constant while the pump frequency f is increasing with 2 Hz every 10 minutes. In other words, the input values are:

$$z = 60\ \%$$
$$f = 50 \text{ Hz, increased by 2 Hz every 10 minutes.}$$

This simulation looked at how the system would respond to a pump frequency change. It is expected that a change of frequency will cause an increase in the liquid flow q. Additionally, the wellhead pressure p_{wh} will increase because it is connected to the output of the pump. At last, the bottomhole pressure p_{bh} will decrease because it is connected to the input of the pump. As seen in Figure 3.4, the system responded as expected to the increase of frequency.

3.2.3 Discussion

The aim of this section was first to give an introduction to the modeling process of the electric submersible pump, then to verify the system model. Different scenarios were simulated to verify the DAE model since small errors could cause deviations. DAE models are often too complex to comprehend by equations alone, but simulation gives the opportunity to isolate variables in order to get a better system understanding. Since the model responded as expected for all simulations, we consider the model implementation as validated. This model will be used for further control and optimization in this dissertation.

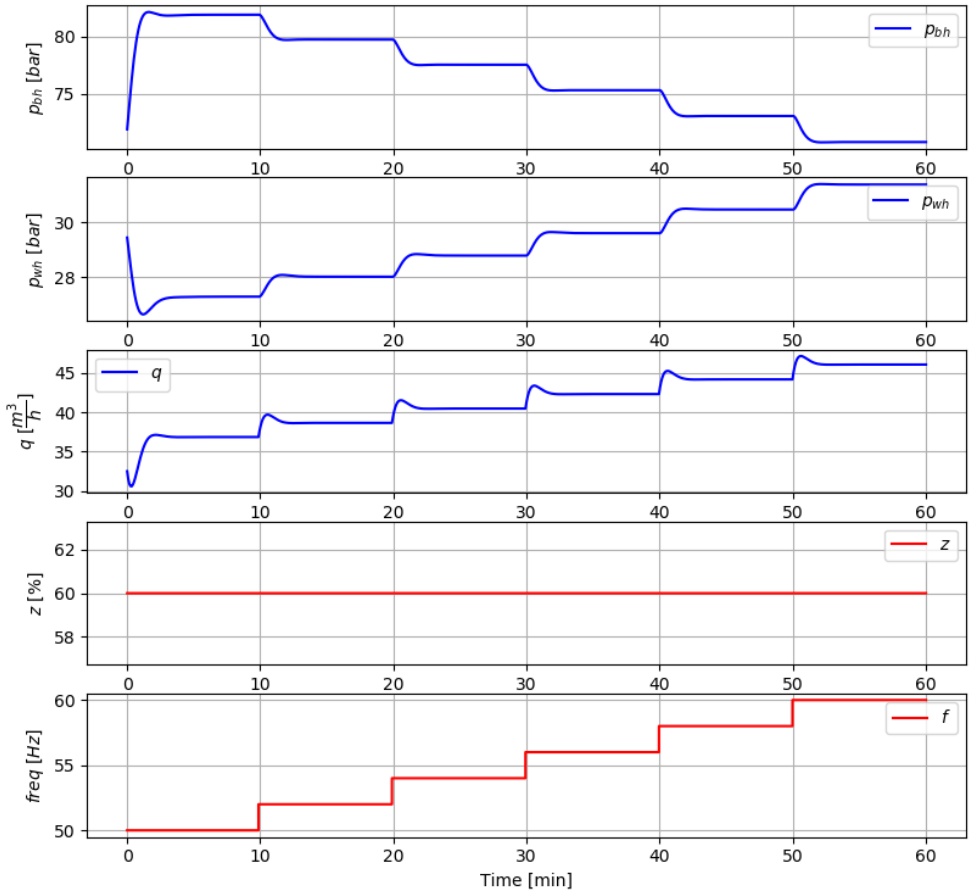


Figure 3.4: System response with an increasing frequency

3.3 Data set

In order to build a data-driven model with an ESN, large amounts of data are necessary. It is important to notice the fact that ESN is a black box modeling technique. This implies that the network would not give good results if the network is operating in a range that was never visited during training [15]. How this data set is created is crucial for both performance and accuracy.

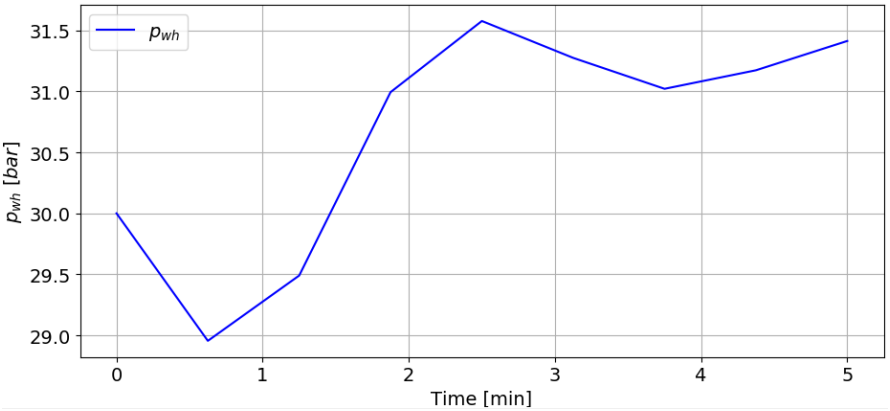
3.3.1 Sampling time

Finding a suitable sampling time is the first step in the process of creating a data set. The ESN accepts input/output mappings in discrete-time. Having a too large sampling time (low sampling rate) can cause aliasing which would lead to loss of information. On the other hand, a too small sampling time (high sampling rate) would lead to increased computational cost since the number of data points is higher than necessary. A balance between computational cost and performance must be considered when selecting a sampling time.

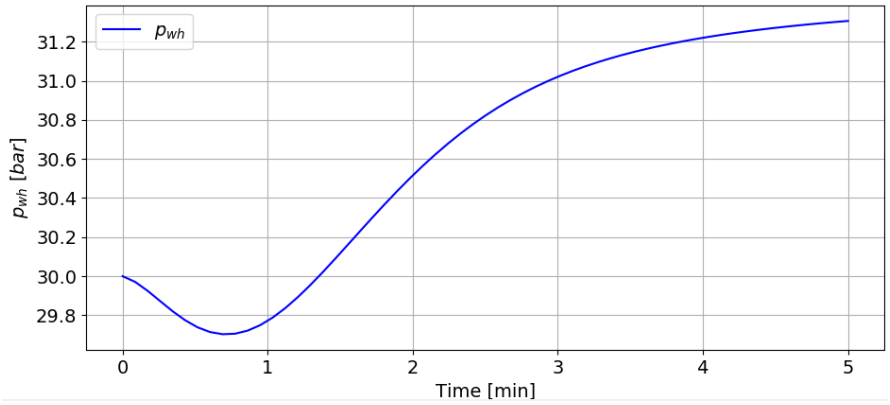
A suitable sampling rate could be found by utilizing one of the simulations from Section 3.2.2. By looking at how the wellhead pressure p_{wh} is responding to different sampling rates could give valuable feedback. The system is simulated with the following constant inputs:

$$z = 50\ \%$$
$$f = 53\ \text{Hz}$$

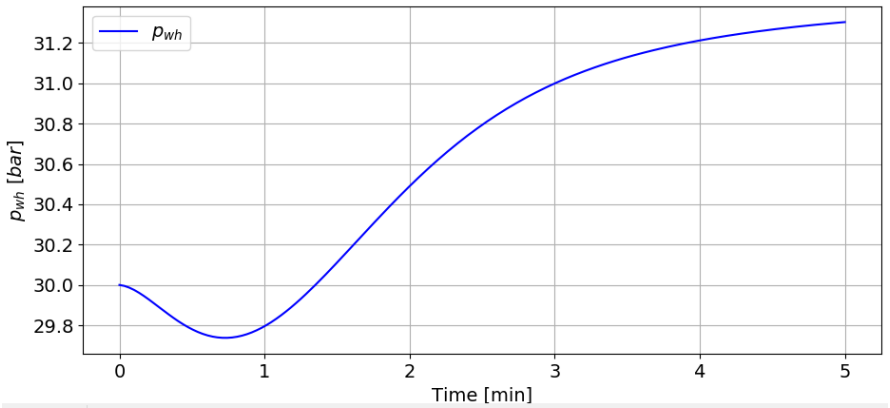
From Figure 3.5 it is obvious that a sampling rate of 2 samples/minute will lead to information loss, while both 12 samples/minute and 30 samples/minute are able to represent the dynamics of the model well. It is decided to use a sampling rate of 12 samples/minute since it represents the system dynamics well while being computationally cheaper than 30 samples/minute.



(a) p_{wh} with 2 samples/minute.



(b) p_{wh} with 12 samples/minute.



(c) p_{wh} with 30 samples/minute.

Figure 3.5: Comparing p_{wh} with different sampling rates.

3.3.2 Excitation signals

The input (excitation) signal u plays an important role in system identification as this is the only way to influence the process in order to gather information about its behavior [2]. It is important to visit all working conditions that will later be met when the trained model is used in a controller. This is easier said than done with nonlinear systems.

A well-motivated approach for linear systems is to use white noise as input as this will reveal most of the system's behavior. However, this does not apply to nonlinear systems. Typically, a white noise input will only visit a small region of the nonlinear system's working range [15]. This will cause problems when the trained model is working with a slow changing input for instance. The model will then be driven to a different working range which will deviate from the range used in the training.

It is also important not to cover a working range wider than what will be met in the testing or exploitation, as this will occupy the modeling capacity with irrelevant information. Poor model accuracy in the actual working region will be a result of a larger training region than necessary.

A ground rule is that the training input should reflect the inputs that would be used in testing and exploitation, but it should be a bit more varied than the expected input in the exploitation phase [15].

PRBS (Pseudo-Random Binary Signal) is a prevalent choice when it comes to excitation signals. It imitates white noise in discrete-time and excites all frequencies equally [2]. A PRBS can be seen in Figure 3.6.

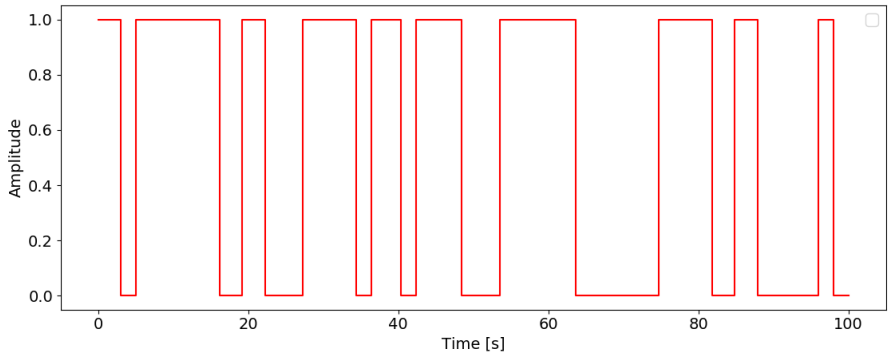


Figure 3.6: Pseudo-Random Binary Signal (PRBS)

Although PRBS is suitable for linear systems, no information about the system behavior is given for amplitudes other than 0 and 1. There are, however, a simple solution to this problem. Giving the PRBS signal a random amplitude will result in an APRBS (Amplitude-modulated Pseudo-Random Binary Signal). The APRBS will give a rich input data distribution which can be seen in Figure 3.7.

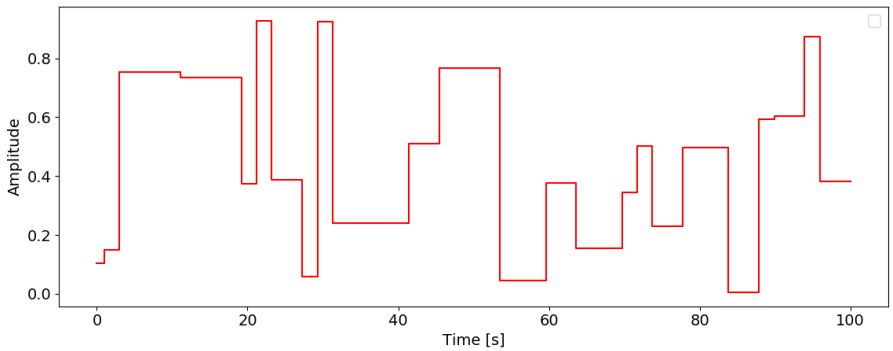


Figure 3.7: Amplitude-modulated Pseudo-Random Binary Signal (APRBS)

APRBS was selected as the excitation signal for this work. The minimum and maximum values for the excitation signal can be found in Table 3.3.

Table 3.3: Input signal limits

	min	max
z	0 %	100 %
f	35 Hz	65 Hz

3.3.3 Training set

When the MPC will be used for control later, it will send a new input to the ESN every sampling instant k. In other words, the ESN will receive a new input every 5 seconds. The ESN must also be able to respond to receiving the same input for a larger period of time, making the ESN converge to a steady-state. It is therefore smart to involve both fast and slow input changes in the data set. In this way, the ESN would be accurate for both fast and slow dynamics. An example of the training set can be seen in Figure 3.8. The red graphs z and f are labeled as input and the blue graphs p_{bh}, p_{wh} and q are labeled output. This set is simulated for 1000 minutes which gives a total of 12000 data points. It is evident to see that holding an input for a longer period of time will visit more working conditions of the outputs p_{bh}, p_{wh} and q than fast input changes alone.

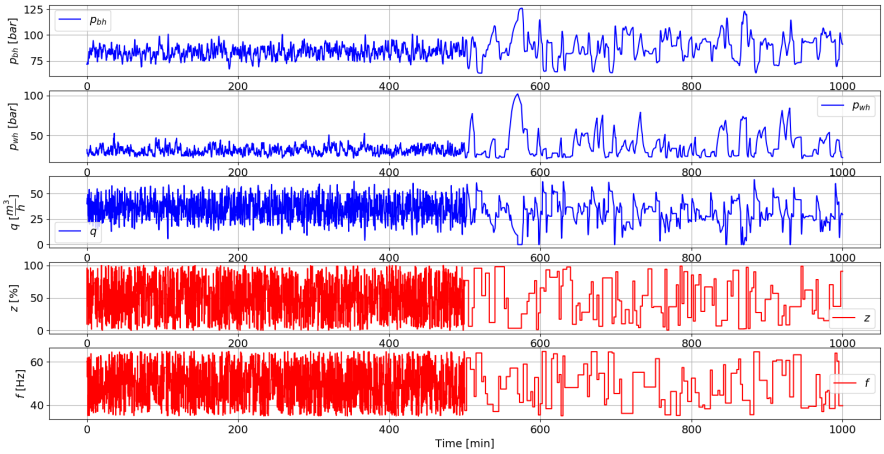


Figure 3.8: Training set example

3.4 Building an Echo State Network (ESN)

Building an ESN is referred to as selecting parameters described in Section 2.4. Before starting the selection process, it is important to have a measure on performance. This is normally done by looking at the error between the ESN output and the real system output. Since the ESN is working with normalized values, it is natural to also use normalized values in the error calculation such that each state is equally weighted in the error function. The normalized values can be extracted from Equation (3.11).

$$\hat{y} = \frac{y - y_{min}}{y_{max} - y_{min}}, \tag{3.11}$$

where $\hat{y}$ is the normalized value. The normalized value $\hat{y}$ will have a range between 0 and 1. An error function can then be defined as:

$$\text{Error} = \sum_{k=0}^{M} |\hat{p}_{bh} - \hat{p}_{bh}^{ESN}| + |\hat{p}_{wh} - \hat{p}_{wh}^{ESN}| + |\hat{q} - \hat{q}^{ESN}|, \tag{3.12}$$

where M is the number of data points.

The network was firstly initialized with a set of nominal values. These values were selected with a quick trial-and-error approach where gaining an understanding of how each variable influenced the network was essential. A more in-depth search was later done when nominal values were obtained. The in-depth search consisted of a search in one parameter while the other parameters were kept fixed.

A data set consisting of 12000 data points were used for each test. The data set was divided into 10 folds and a 10-fold cross-validation method was performed. The average of five different reservoirs was used in order to exclude any deviations. Using multiple reservoirs is important since the ESN is randomly initialized and performance can vary for each reservoir.

Ridge regression was used to train the ESN. The *regularization parameter* λ was found by performing 10-fold cross-validation with a trial and error approach. Values on a logarithmic scale were tested, and $\lambda = 1 \times 10^{-8}$ yielded the best performance.

For the *spectral radius* ρ a value close to one is desirable in order to have a large range of dynamics in the reservoir. It is also important to keep it inside the unit circle of the complex plane in order to ensure stability. Hence, $\rho = 0.99$ was selected as spectral radius.

A *reservoir size* of $N = 200$ was chosen based on a line search method and visual inspection. An error plot from the search can be seen in Figure 3.9. The training time was fast for all reservoir sizes and was neglected in the decision. It was later revealed that the reservoir size played a huge role in the calculation time for the NMPC.

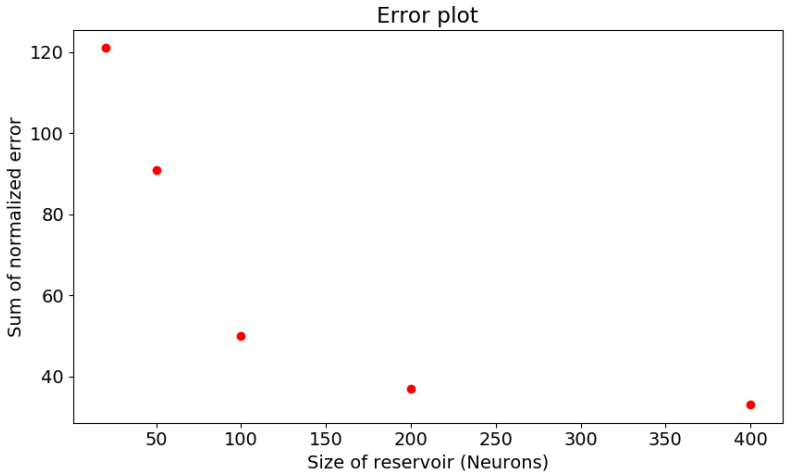


Figure 3.9: Sum of normalized error compared to size of reservoir.

As for the reservoir size, the same search method was utilized to find an optimal *leak rate* α. Firstly, a search for $0.1 \leq \alpha \leq 1$ with a step interval of 0.1 was conducted. The results from this search can be seen in Figure 3.10. A leak rate of $0.1 \leq \alpha \leq 0.3$ gives the best error. This implies that the ESN will have a focus on slowing down the reservoir dynamics and increase the memory. A new line search for $0.1 \leq \alpha \leq 0.3$ with a step interval of 0.02 was then performed. The results can be seen in Figure 3.11 and $\alpha = 0.12$ seems to give the best error. This value was therefore selected as the leak rate.

For the *input scaling* f_i^r a small value is desirable since the influence of the input on the network becomes small. This will make a controller work less aggressively. A search for $0 \leq f_i^r \leq 0.2$ with a step interval of 0.02 resulted in $f_i^r = 0.1$.

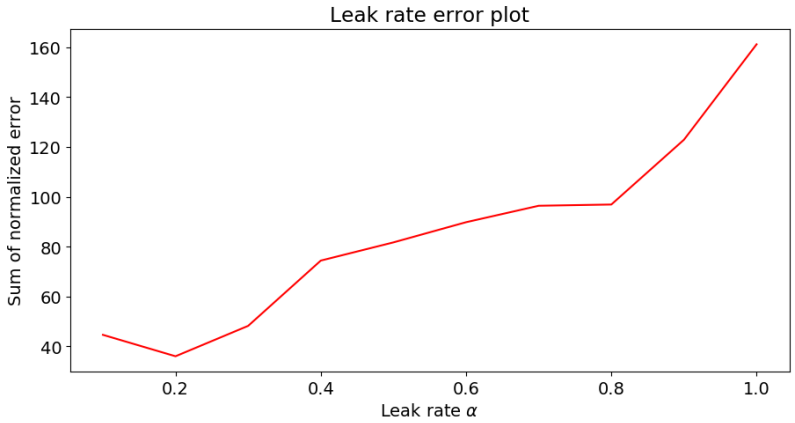


Figure 3.10: Sum of normalized error relative to leak rate α.

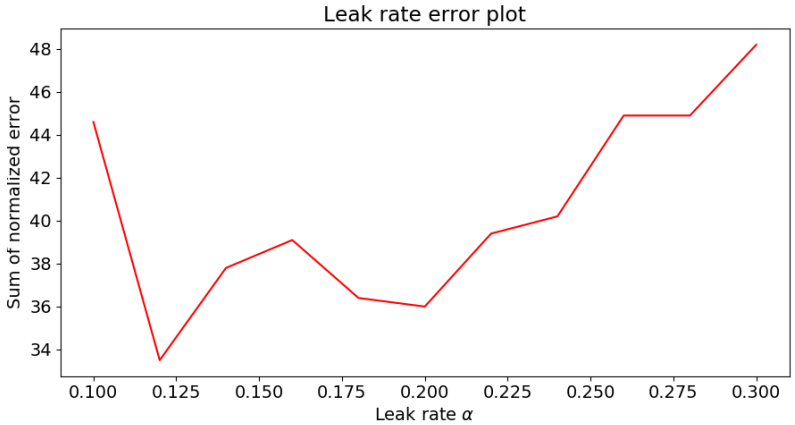


Figure 3.11: Sum of normalized error relative to leak rate α with smaller step interval.

Sparsity was neglected from this work. A strong sparsity was needed to see a major difference in computational cost, but this led to poorer performance.

To sum up, all selected parameters can be seen in Table 3.4.

Table 3.4: Global parameters for ESN

Global Parameters		
N	Reservoir Size	200
α	Leak Rate	0.12
ρ	Spectral Radius	0.99
ψ	Sparsity	0
T_s	Sampling Time	5 s
f_i^r	Input Scaling	0.1

An important note for the ESN is to run a warm-up phase before the states are harvested during training and before the ESN is used in a controller. Since dynamical systems are history-dependent, it is important for the ESN and the system to have an equivalent origin. The problem can be seen in Figure 3.12. Having these deviations in addition to negative values will cause issues on the control part. This is simply solved by sending the same input to both the ESN and the system for e.g. 200 samples before it is used for control purposes.

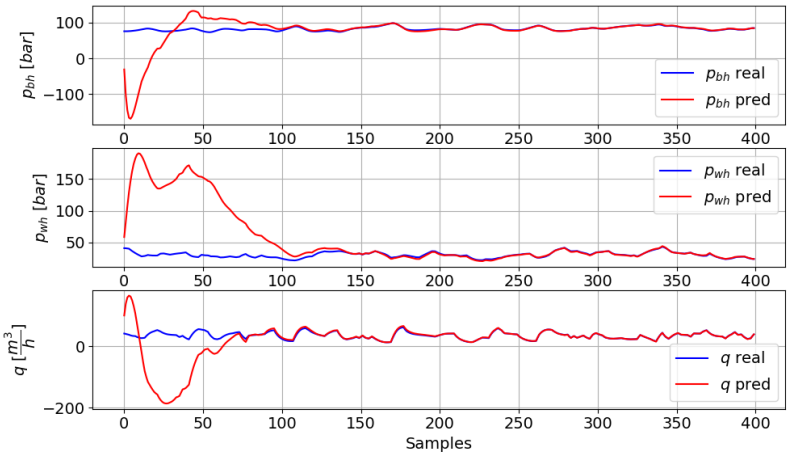


Figure 3.12: Model identification without a warm-up phase.

3.5 NMPC implementation

The final step in the implementation is to build an NMPC. The NMPC is aiming to perform optimal control on the plant by utilizing the ESN as a prediction model.

3.5.1 Optimal Control Problem

Finding a suitable Optimal Control Problem (OCP) is the initial step in implementing an NMPC. That is, finding a control law for the ESP system in order to obtain optimal control. The optimal control target is typically specified for each output variable. In this case, the output variables are bottomhole pressure p_{bh}, wellhead pressure p_{wh} and liquid flow q. The controller made in this work can handle both lower and higher limits in addition to a setpoint at each output variable. For the input variables z and f (manipulated variables), hard constraints are implemented both for high and low limits.

The constraints were considered according to [10]:

$$0 \leq z \leq 100 \ [\%] \tag{3.13}$$

$$35 \leq f \leq 45 \ [Hz] \tag{3.14}$$

$$1 \leq p_{wh} \leq 60 \ [\text{bar}] \tag{3.15}$$

$$0 \leq q \ [m^3/s] \tag{3.16}$$

The constraints will stay fixed throughout this work. A general OCP with constraints can therefore be defined as:

$$\min \quad \text{J}(x_o, u) \quad \textit{subject to:} \begin{cases} x[k+1] = f(x[k], u[k]) \\ x(0) = x_0 \\ \underline{u}[k] \leq u[k] \leq \overline{u}[k] \\ \underline{x}[k] \leq x[k] \leq \overline{x}[k] \end{cases} \tag{3.17}$$

where

$$x = \begin{bmatrix} p_{bh} \\ p_{wh} \\ q \end{bmatrix} \text{ and } \quad u = \begin{bmatrix} z \\ f \end{bmatrix}$$

Further, the upper and lower limits are defined as:

$$\underline{x} = \begin{bmatrix} 0 \text{ bar} \\ 1 \text{ bar} \\ 30 \ m^3/h \end{bmatrix}, \quad \overline{x} = \begin{bmatrix} \infty \\ 60 \text{ bar} \\ 80 \ m^3/h \end{bmatrix}, \quad \underline{u} = \begin{bmatrix} 0 \ \% \\ 35 \text{ Hz} \end{bmatrix} \quad \text{and} \quad \overline{u} = \begin{bmatrix} 100 \ \% \\ 65 \text{ Hz} \end{bmatrix}$$

The lower bound on the production choke valve z was later set to 10% because the model was imprecise at low values of z.

3.5.2 Objective function

The control objective in this work is to maintain the bottomhole pressure p_{bh} at a desirable setpoint. Further, the liquid flow q should be maximized inside an operating window. Different control approaches can be provided by selecting various objective functions.

Firstly, a standard approach where only soft constraints on the controlled variables are included in the objective function. The objective function will punish deviation between a setpoint p_{bh}^{ref} and p_{bh}. Maximizing the liquid flow q can be done by adding a high unreachable setpoint as q^{ref} and punish the controller for deviation. This is often referred to as setpoint chasing in industrial applications. The objective function can be written as:

$$J(x_0, u) = \sum_{k=0}^{N-1} \left\| x[k] - x[k]^{ref} \right\|_Q^2 \tag{3.18}$$

where the weighting matrix Q will be used for tuning.

The second approach that will be tested is an objective function that introduce change of control in the calculation. This will prevent the controller from making big leaps on the manipulated variables. Big leaps in manipulated variables can often lead to more wear and tear on equipment. This approach can be described with the following cost function:

$$J(x_0, u) = \sum_{k=0}^{N-1} \left(\left\| x[k] - x[k]^{ref} \right\|_Q^2 + \|\Delta u[k]\|_R^2 \right) \tag{3.19}$$

where Q would be used to tune the controlled variables and R would be used to tune the manipulated variables.

3.5.3 Solving the OCP

The OCP must be transformed into a nonlinear programming (NLP) problem in order to be solved numerically. *Single shooting* is selected as the shooting method because of its intuitiveness. The computational cost between single shooting and multiple shooting showed to be roughly the same for the experiments in this work because of the short prediction horizon. Anyhow, single shooting is unfavorable if the prediction horizon is large since the function tends to become highly nonlinear.

CasADi [32] is used to formulate the NLP problems. The problems are then solved using an Interior Point Optimizer (IPOPT). IPOPT is an open-source software package widely used for nonlinear optimization. Further, the sampling time for the MPC is equal to the sampling time used in the ESN, 5 seconds.

3.5.4 The MPC cycle

The ESN obtained from Section 3.4 is predicting future states for a prediction horizon N. A rule of thumb is to select a horizon as long as it takes the system to reach a steady-state without control. However, as mentioned earlier, a system-model mismatch will be amplified when the prediction horizon grows larger. For this reason, a trial-and-error approach will be used to find an optimal prediction horizon. The prediction horizon plays an important role regarding the computational cost as well.

The NMPC is made inside a for loop where IPOPT is solving a new NLP for each iteration. The solution of the NLP gives an optimal input u which is applied to both the plant and the ESN. A new NLP problem is then defined at the next iteration. A figure of the NMPC cycle can be seen in Figure 3.13.

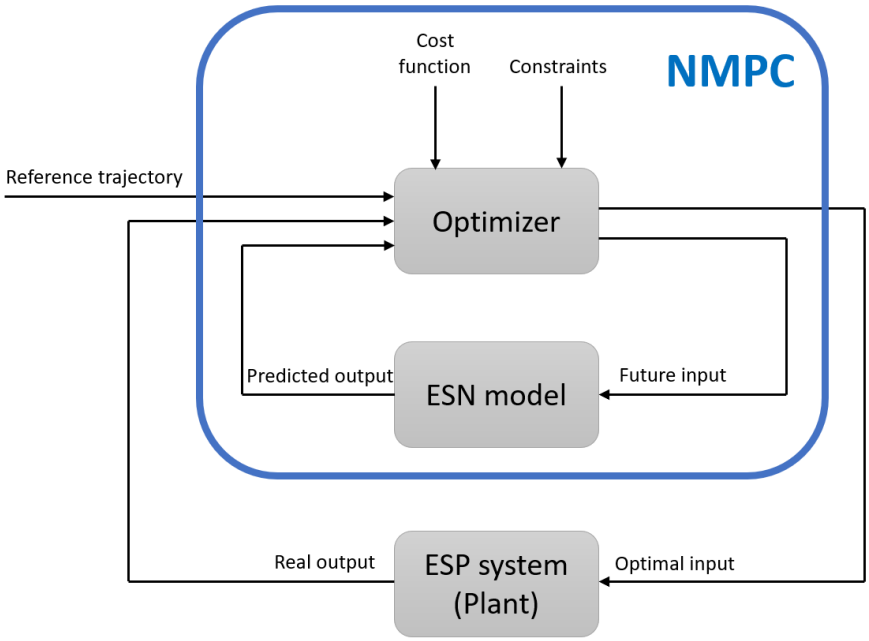


Figure 3.13: NMPC block diagram.

Chapter 4

Experiments and results

This chapter explains how simulations were conducted to obtain the results, which are then presented and discussed. It is divided into two parts; one where the ESN is used for model identification and another where the ESN based NMPC is tested.

4.1 ESN experiments

This section will initially look at how the ESN with parameters from Table 3.4 will identify the plant. After the ESN is trained and the warm-up phase is executed, the obtained data-driven model is verified with 400 random input values and compared to the real system. A warm-up phase consisting of a constant input where $z = 50\%$ and $f = 40$ Hz for 200 samples will be used in order to reach an equivalent starting condition for the plant and the ESN model.

A prediction error from the ESN is measured on test data with respect to the desired output. This is done by calculating the *Relative Error*:

$$e_{pre,\%}[k] = 100 \times \left| \frac{y_m[k] - y_{esn}[k]}{y_m[k]} \right| \tag{4.1}$$

where y_{esn} is the predicted output and y_m is the measured output from the plant.

Multiple experiments are attempted in order to ascertain that the ESN is suitable for the NMPC. Two different data sets will be utilized, one with only fast dynamics and one with a mix of fast and slow dynamics as in Figure 3.8. All data sets consist of 24000 data points, which is equal to the plant running for 2000 minutes.

4.1.1 Steady-state test

The first experiment will look at the trained ESN's ability to reach a steady-state. This is an important trait for the network to have as the NMPC will try to reach

several steady-states. The input values are given as:

$$z = 71\ \%$$
$$f = 63\ \text{Hz}$$

Firstly, the ESN is trained with only fast dynamics. Its results can be seen in Figure 4.1. The relative error is shown in Figure 4.2. It is clear to see that the ESN can handle the first quick movements, but the ESN is struggling with some deviation when it is supposed to reach a steady-state reference.

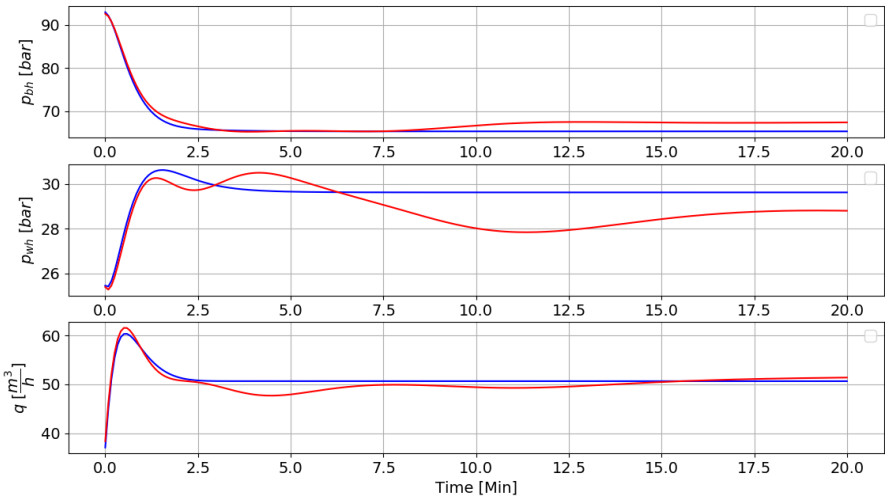


Figure 4.1: ESN trained with fast dynamics trying to reach a steady-state. Blue is the true system while red is the ESN prediction.

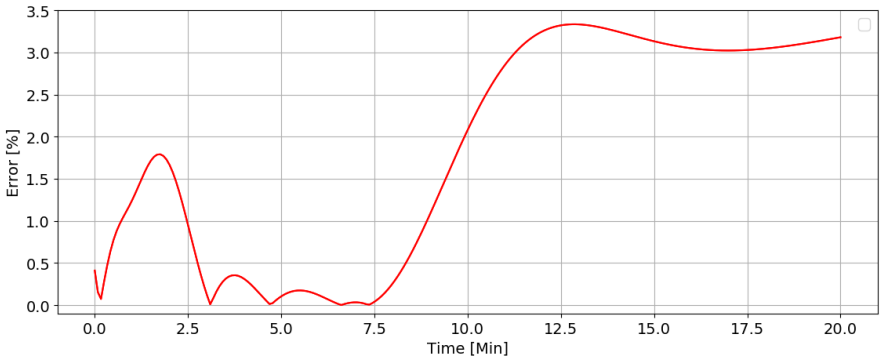


Figure 4.2: Relative error between the true system and the ESN prediction in Figure 4.1.

Secondly, an ESN trained with a mix of fast and slow dynamics is tested. The results can be seen in Figure 4.3 and the following error can be seen in Figure 4.4. It is obvious that the ESN trained with mixed dynamics is able to reach the steady-state in a better manner than the one trained purely on fast dynamics. It is also

worth noticing how p_{wh} is deviating after 5 minutes but is able to catch up over time.

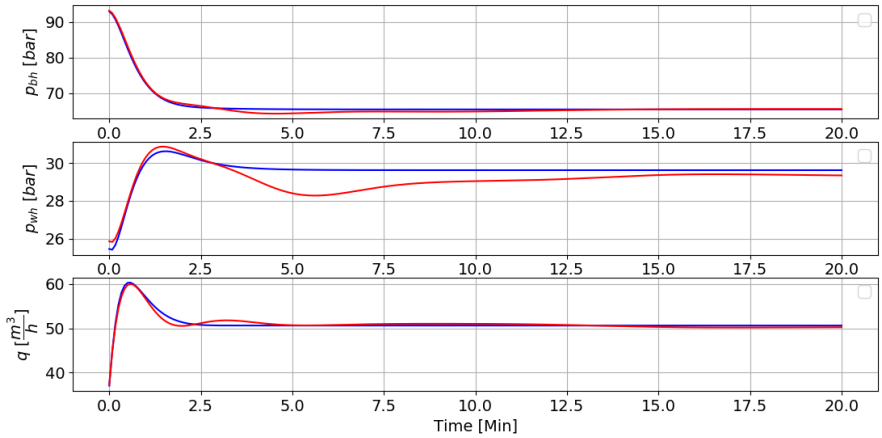


Figure 4.3: ESN trained with mixed dynamics trying to reach a steady-state. Blue is the true system while red is the ESN prediction.

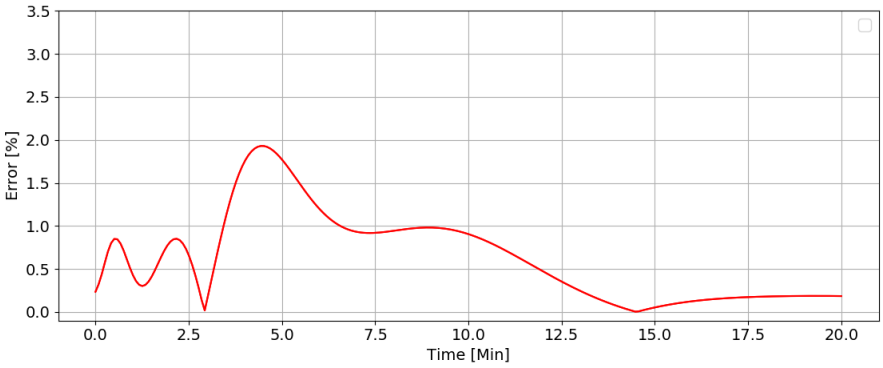


Figure 4.4: Relative error between the true system and the ESN prediction in Figure 4.3.

4.1.2 Fast dynamics

The ESN should react to a new input from the NMPC every 5 seconds. It is therefore crucial that the network is able to respond correctly to fast dynamics. This experiment will also give an indication of how much the mixed ESN will suffer from having both fast and slow dynamics in the data set.

Firstly, the ESN trained purely on fast dynamics is tested. The result can be seen in Figure 4.5 and its corresponding error in Figure 4.6. There is no doubt that the ESN catches the plant's dynamics well and manages to imitate its dynamics properly. The ESN trained solely on fast dynamics will work as a reference for the ESN trained with mixed dynamics in this experiment.

It is made clear in Figure 4.7 that using a mixed training set would not cause a big impact on how the ESN responds to fast dynamics. This is also confirmed in Figure 4.8 where the error plot is shown.

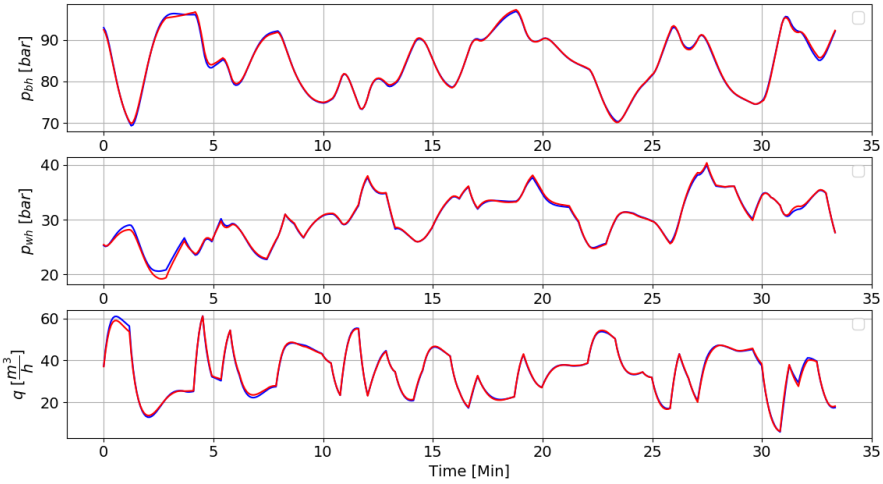


Figure 4.5: ESN trained with fast dynamics trying to imitate fast dynamics. Blue is the true system while red is the ESN prediction.

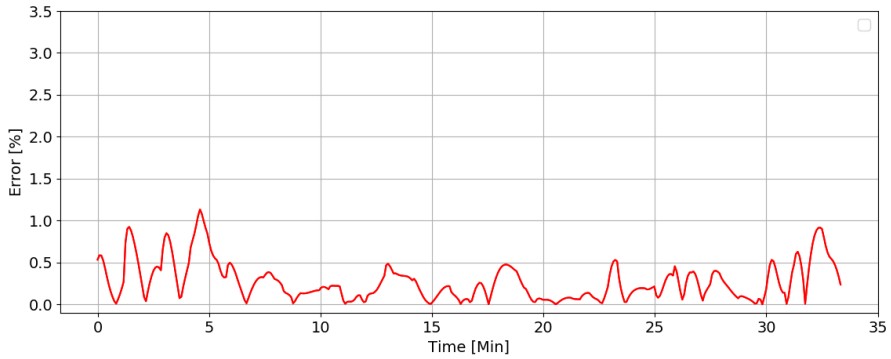


Figure 4.6: Relative error between the true system and the ESN prediction in Figure 4.5.

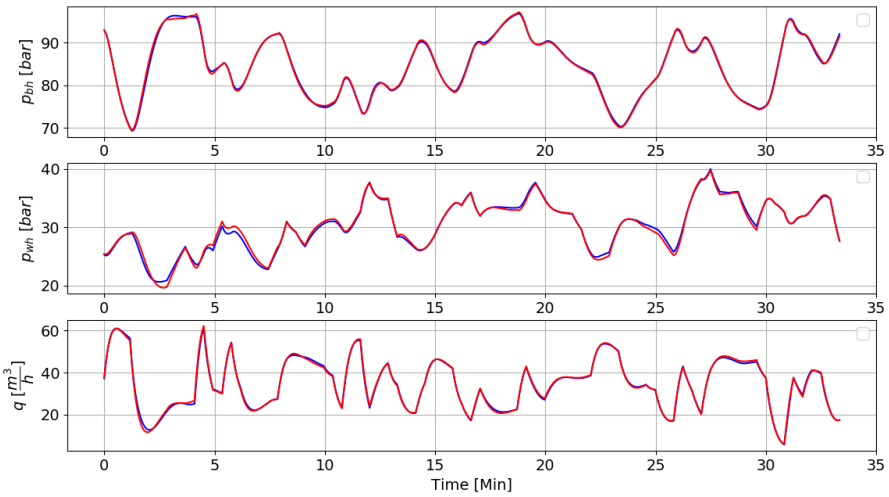


Figure 4.7: ESN trained with mixed dynamics trying to imitate fast dynamics. Blue is the true system while red is the ESN prediction.

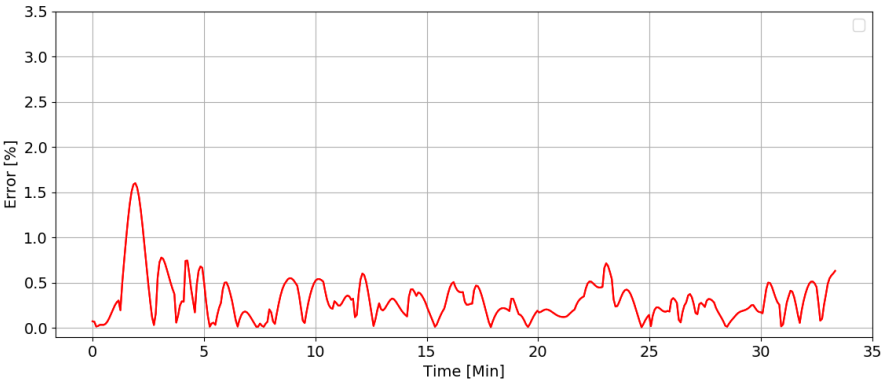


Figure 4.8: Relative error between the true system and the ESN prediction in Figure 4.7.

4.2 NMPC simulation

The obtained ESN trained with mixed dynamics yielded the most promising performance. It will therefore be implemented as a prediction model in the NMPC. That is, the ESN is going to predict future behavior of the plant in order to yield optimal control.

The experiments in this section will be assessed based on visual inspection and error metrics. Regarding the error metrics, a *mean trajectory error* e_T is viable for these experiments. It is defined as:

$$e_T = \frac{1}{N}\sum_{k=0}^{N}\left\|p_{bh}^{\text{ref}} - p_{bh}\right\|_1, \tag{4.3}$$

where N is the number of steps in the simulation. An *Integral Absolute Error* (IAE) will also be presented, since it is a widely applied metric in similar studies. The IAE is directly related to the e_T and is obtained by multiplying the e_T with the number of steps in the simulation N:

$$IAE = N \times e_T. \tag{4.4}$$

Neither e_T nor IAE will capture any behavior of the controller, such as oscillatory behavior. A metric that looks at control action is therefore presented as an alternative metric. The metric Δz and Δf is implemented to present the total control variation of the production choke valve z and ESP frequency f. The total control variations is defined as:

$$\Delta z = \sum_{k=0}^{N-1}\|z[k+1] - z[k]\|_1, \tag{4.5}$$

$$\Delta f = \sum_{k=0}^{N-1}\|f[k+1] - f[k]\|_1. \tag{4.6}$$

This metric will increase if the control variation changes between consecutive steps in the simulation. It is desirable to keep Δz and Δf as low as possible.

4.2.1 Reaching a bottomhole pressure reference

The first experiment will look at the NMPC's ability to reach a reference point for the bottomhole pressure p_{bh} while maximizing the liquid flow q. A suitable objective function must be defined in order to reach a desirable control target. Since this experiment is going to both reach p_{bh}^{ref} in addition to maximizing q, both these targets must be included in the objective function that will be minimized by the optimizer. Reaching the bottomhole pressure reference is done by punishing deviation between p_{bh} and p_{bh}^{ref}. Further, maximizing liquid flow q can be done in several ways. This experiment will introduce a soft constraint on the liquid flow

q. That is, implementing an unreachable high value as the reference for q. This approach is known as a setpoint chasing and is common to use in many industrial applications.

The objective function J for this experiment is defined as:

$$\mathrm{J} = \sum_{k=0}^{N-1} \left\| x[k] - x[k]^{ref} \right\|_Q^2 , \quad \text{where } Q = \begin{bmatrix} q_1 & 0 & 0 \\ 0 & q_2 & 0 \\ 0 & 0 & q_3 \end{bmatrix} \tag{4.7}$$

Tuning Q is done with a trial-and-error approach where a balance between accuracy on p_{bh} and flow maximization must be considered. The obtained tuning parameters can be seen in Table 4.1.

Table 4.1: Weighting values with their respective parameters for the first experiment.

	Variable	Weight
q_1	p_{bh}	1×10^{-8}
q_2	p_{wh}	0
q_3	q	100

The high unreachable setpoint for q is chosen equal to the upper limit of q defined in the OCP in Equation (3.17):

$$q^{ref} = 80 \ m^3/h$$

A fairly short prediction horizon is tested in this experiment. One motivation for selecting a short horizon is that the calculation time is rather short. $N = 3$ means that the NMPC will predict 15 seconds forward in time since the sampling time is 5 seconds.

The plant's response and the optimal control can be seen in Figure 4.9. The reference value for p_{bh} is marked in yellow, which is performing a step from 75 bar down to 68 bar. By inspecting the bottomhole pressure at the top of the figure it is clear that the NMPC is able to reach the reference fairly well. However, the control is changing excessively and would not be convenient for either the production choke valve or the pump. It is also worth noticing how excessive control will make p_{bh} deviate from its reference right after the start.

This experiment was also conducted with a prediction horizon of $N = 10$. This performed significantly worse than $N = 3$ with big oscillations on both the pump frequency and control valve. Large oscillations on the manipulated variables are most likely due to the optimizer focusing too hard on maximizing the liquid flow q.

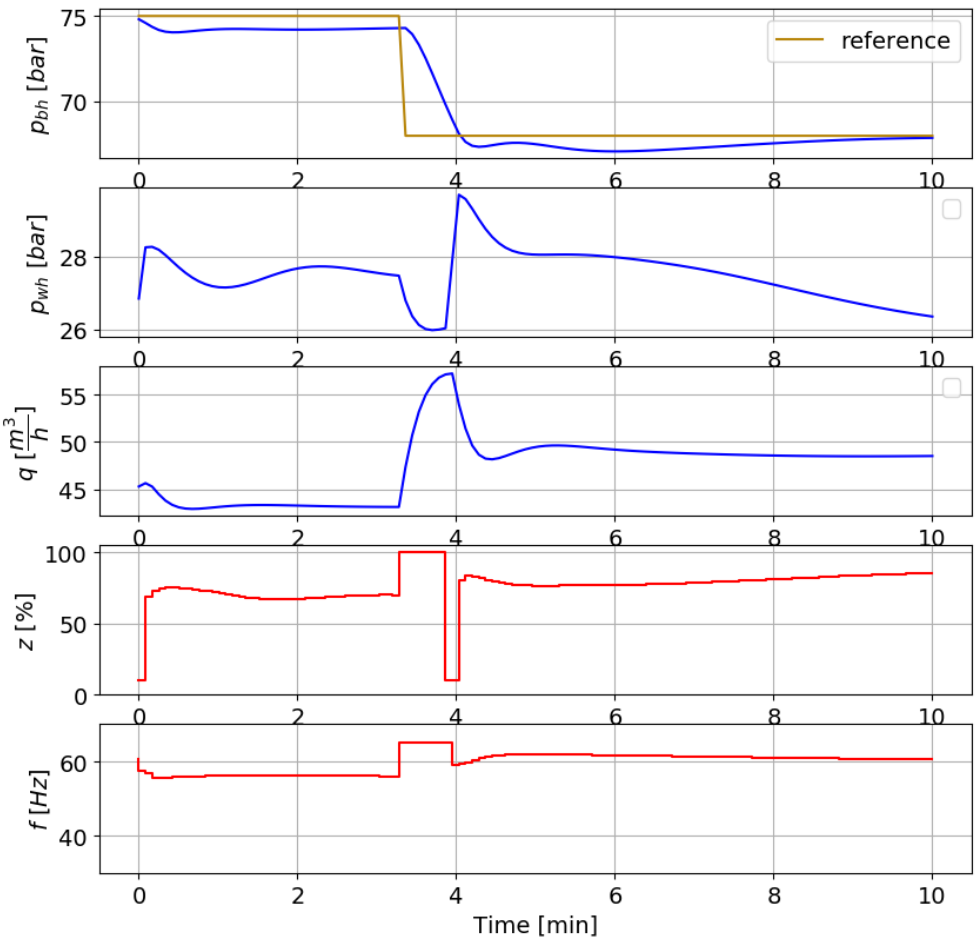


Figure 4.9: NMPC reaching a setpoint without punishing change of control.

4.2.2 Punishing change of control

The experiment conducted in Section 4.2.1 showed that the manipulated variables had big leaps when they ran freely. It is therefore necessary to implement some form of constraint where control variation is punished in the objective function. This is done by including both the production choke valve z and the ESP frequency f with a weighting matrix R. The objective function is therefore defined as:

$$\mathrm{J} = \sum_{\mathrm{k}=0}^{\mathrm{N}-1} \left\| \mathrm{x[k]} - \mathrm{x[k]^{ref}} \right\|_{\mathrm{Q}}^2 + \left\| \Delta \mathrm{u[k]} \right\|_{\mathrm{R}}^2 \tag{4.8}$$

where the weighting matrices Q and R are given as:

$$\mathrm{Q} = \begin{bmatrix} q_1 & 0 & 0 \\ 0 & q_2 & 0 \\ 0 & 0 & q_3 \end{bmatrix} \quad \text{and} \quad \mathrm{R} = \begin{bmatrix} r_1 & 0 \\ 0 & r_2 \end{bmatrix}$$

This experiment will focus on keeping the production choke valve as steady as possible and let the ESP frequency run more freely. The tuning parameters obtained from testing are given in Table 4.2. As for the previous experiment, this

Table 4.2: Weighting values with their respective parameters for the second experiment.

	Variable	Weight
q_1	p_{bh}	1×10^{-8}
q_2	p_{wh}	0
q_3	q	1×10^{3}
r_1	z	1×10^{5}
r_2	f	1

experiment will also utilize a prediction horizon of $N = 3$. It becomes clear in Figure 4.10 that implementing soft constraints on the manipulated variables will lead to a smoother control. This controller will focus on keeping the manipulated variables constant and give maximization of the liquid flow less priority. It is also worth noticing how p_{bh} reaches its reference at the same pace as in Figure 4.9 with far less control variation. This also comes clear in Table 4.3 where the error metrics for Figure 4.9 and Figure 4.10 is displayed.

Table 4.3: Mean trajectory error, integral absolute error and control variation metrics for Figure 4.9 and Figure 4.10.

	Figure 4.9	Figure 4.10
e_T	0.873	0.528
IAE	104.7	63.3
Δz	286.0	20.2
Δf	25.0	22.8

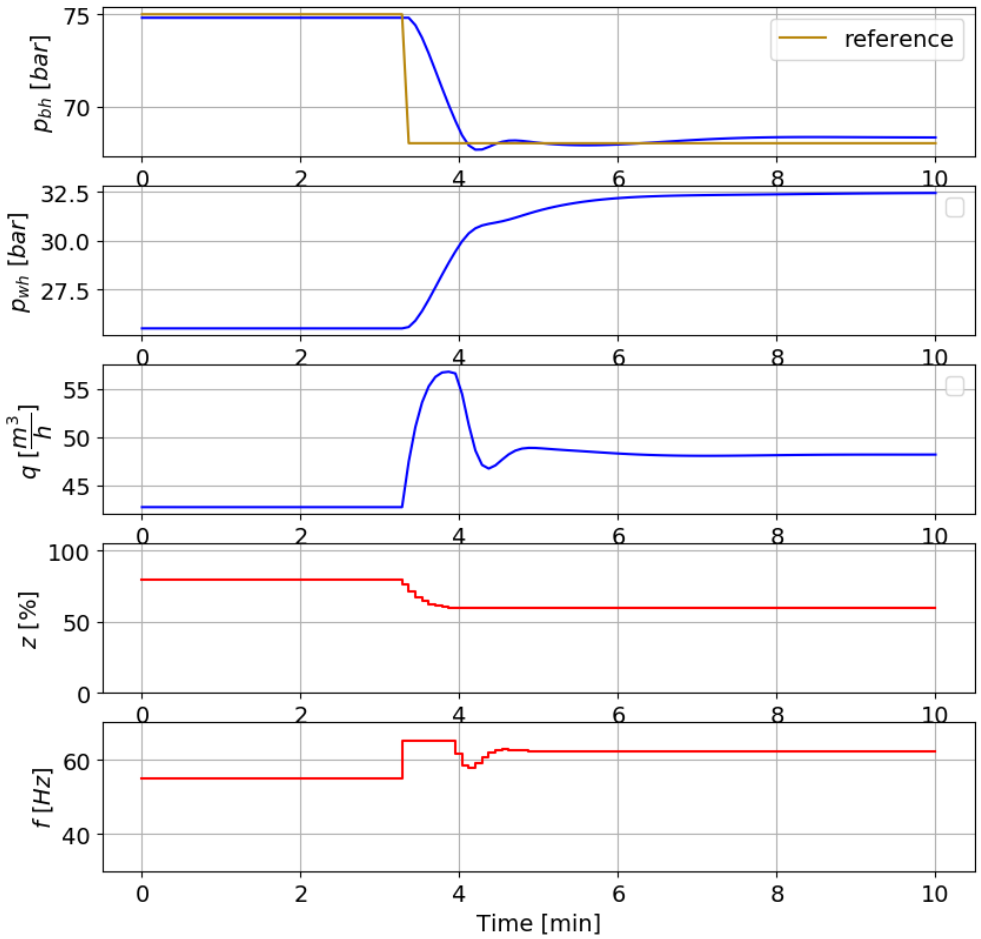


Figure 4.10: NMPC reaching a setpoint with punishing change of control.

4.2.3 Control with multiple steps

The last experiment will look at how the controller obtained in Section 4.2.2 will respond to a bottomhole reference with multiple steps. There is no doubt that including a constraint on control variation will improve the overall performance of the controller. This experiment will have the same objective function as in Section 4.2.2, but two different prediction horizons will be examined. The weighting values for this experiment can be seen in Table 4.4.

Table 4.4: Weighting values with their respective parameters for the third experiment.

	Variable	Weight
q_1	p_{bh}	1×10^{-8}
q_2	p_{wh}	0
q_3	q	1
r_1	z	1×10^{4}
r_2	f	1

Using multiple steps could give important information on how the controller would react to different reference values. This experiment will simulate a drop from 82 bar down to 68 bar divided into 4 steps. Firstly, a prediction horizon of $N = 3$ is tested. The results of this experiment are shown in Figure 4.11.

Lastly, a prediction horizon of $N = 10$, which will predict 50 seconds of the future behavior before it calculates an optimal control sequence. Increasing the prediction horizon can in many cases lead to more soft control because the controller is able to predict further into the future. Figure 4.12 shows how the controller responds to a larger prediction horizon.

It becomes clear that the two different horizons will produce quite similar optimal control. However, by inspecting Δf in Table 4.5 it is clear that using a larger prediction horizon will get rid of some excessive frequency change. The increased prediction horizon manages to lower the IAE in addition to reducing the change of control.

Table 4.5: Mean trajectory error, integral absolute error and control variation metrics for Figure 4.11 and Figure 4.12

	Figure 4.11	Figure 4.12
e_T	0.423	0.410
IAE	121.9	118.1
Δz	7.40	6.12
Δf	142.36	105.52

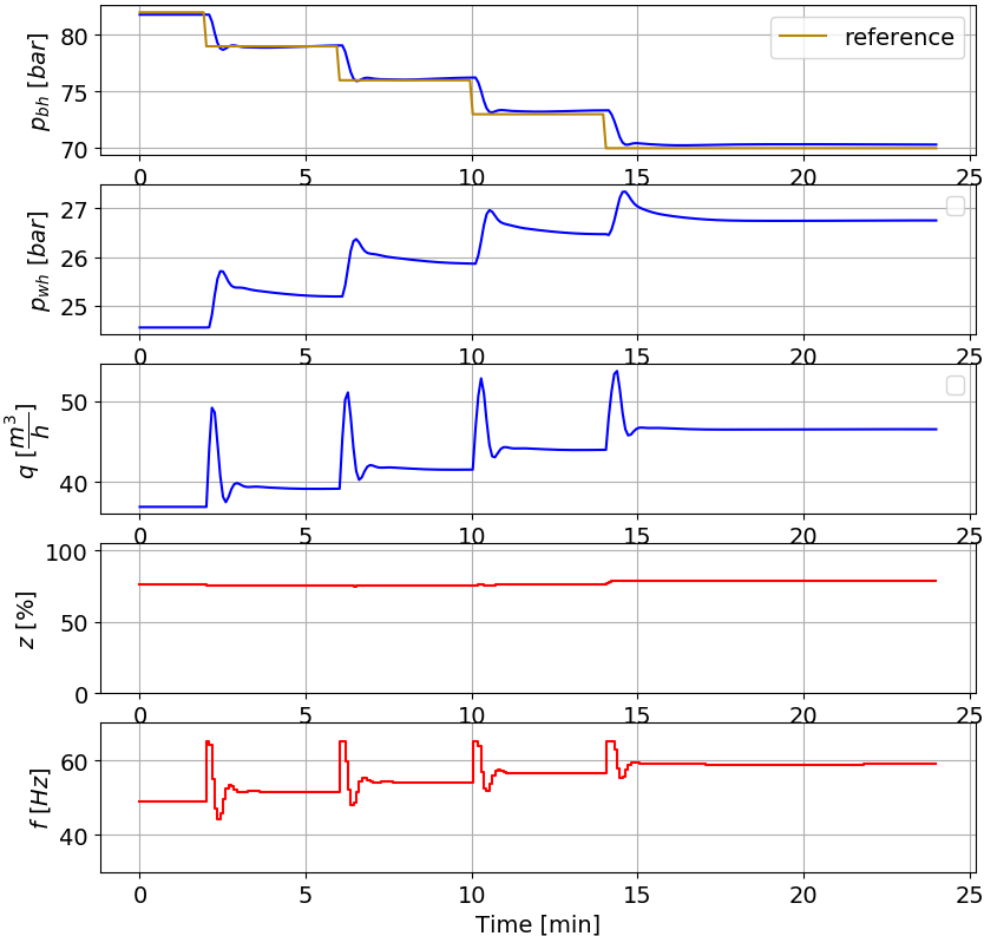


Figure 4.11: NMPC reaching multiple setpoints with punishing change of control with a prediction horizon of $N = 3$.

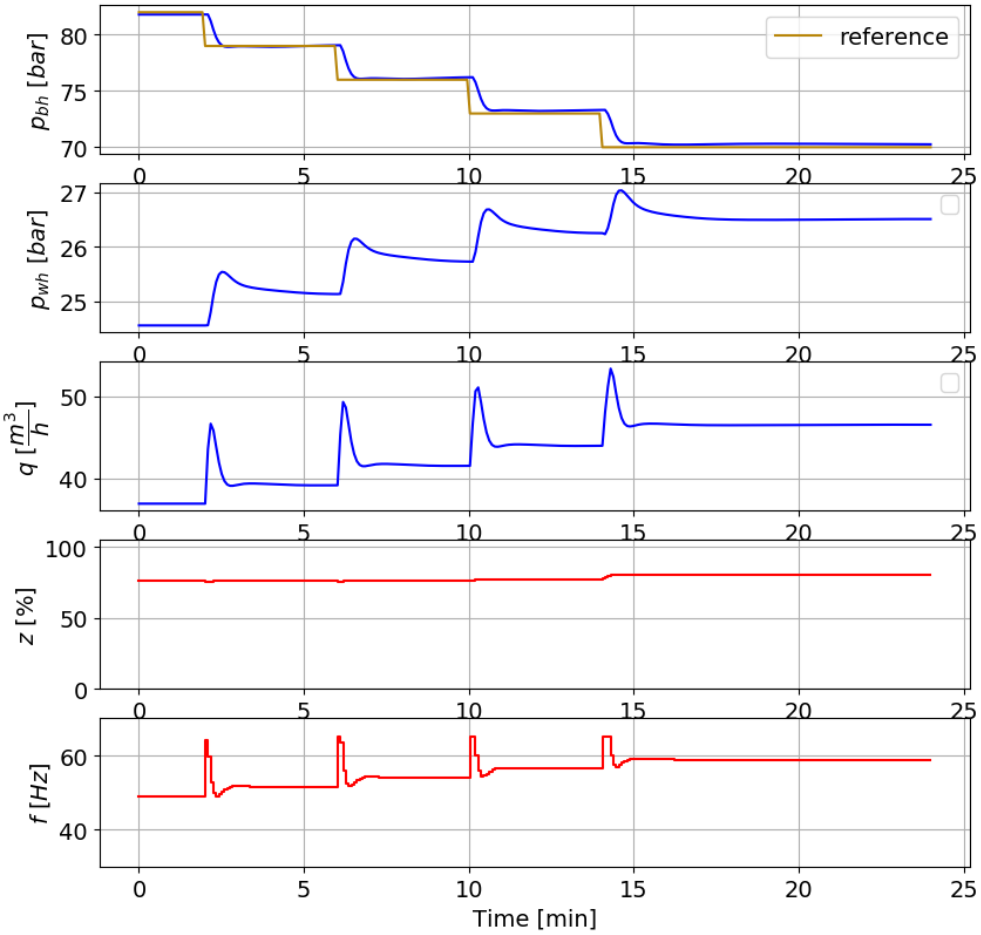


Figure 4.12: NMPC reaching multiple setpoints with punishing change of control with a prediction horizon of $N = 10$.

Chapter 5

Discussion

This work started with an implementation of the nonlinear Electric Submersible Pump (ESP) system in Python. The ESP system was initially given as a set of Differential Algebraic Equations (DAEs). Simulations of the system made it easier to gain a better understanding of the dependent variables and the system as a whole.

Python was an ideal choice of programming language for this work. Further, the CasADi framework made it easy to implement and simulate the ESP system. However, a drawback with CasADi is that troubleshooting is complicated because of the symbolic framework. Accessing future states in the ESN was a complicated operation and hard to troubleshoot.

Regarding the data set, a mixture of fast and slow dynamics yielded best performance. This mixture of dynamics covered a wider working range and performed significantly better in cases where the ESN received a constant input over a large period of time. However, the steady-state test for the ESN could have yielded even better performance. Increasing the length of the data set for the slow dynamics could have solved this problem. Increasing the data set will increase the computational cost, but would hardly affect the computational cost in the long run. One motivation for keeping the data set small is that real plants could have limited amounts of data available.

Using 200 nodes in the reservoir had a significantly better performance than using 100 nodes. It does, however, increase the computational cost in the NMPC considerably. The computational cost in the NMPC increased quadratically with respect to the number of nodes in the ESN. Using 100 nodes with a larger data set could have made a significant improvement regarding the computational cost in the NMPC.

For the NMPC, *single shooting* was utilized to form a Nonlinear Programming (NLP) problem for the Optimal Control Problem (OCP). This method was intuitive

to implement and troubleshoot, but will be more computationally expensive than *multiple shooting* if the prediction horizon is long. For this work, a short prediction horizon yielded promising control. For a prediction horizon $N = 3$ the algorithm took around 1 second to solve the NLP problem in each iteration, while for $N = 10$ it needed around 20 seconds to reach a solution in each iteration.

With regards to objective functions in the NMPC, using only state deviation yielded poor control. Big leaps in the manipulated variables will increase wear on equipment and could cause a shorter lifespan. To maximize the liquid flow q is an important control target but would cause oscillatory control if it is highly prioritized.

Including a punishment for control change in the objective function resulted in a huge control improvement. For this objective, the production choke valve z was heavily punished with regards to changes, while the ESP frequency f ran more freely. Using this objective yielded more steady control with less excessive changes. In this objective function, only a soft constraint was implemented for the change of control. Normally, production choke valves have limitations on how quickly they can open and close. An interesting approach would have been to use hard constraints on the change of control to get a more realistic simulation. It would also be natural to minimize the power consumption for the pump, however, that will most likely force the production choke valve to be fully open at all times for this system.

Chapter 6

Conclusion

This work investigated how an Echo State Network (ESN) could work as a prediction model for an NMPC. The motivation was to utilize data-driven control in complex real-world problems where physics-based models are excessively hard to obtain. That is, building a model for a system without prior knowledge with a black-box modeling approach. A key point in this work was to look into how this approach could deliver a satisfactory operation for the Electric Submersible Pump (ESP) system.

The ESN managed to identify the ESP system satisfactorily. High accuracy is needed since the network lacks feedback from the plant. Using an ESN lowered the computational cost significantly compared to other Recurrent Neural Networks (RNNs) since ESNs have a linear training method.

The NMPC with the data-driven ESN model as a predictor managed to yield optimal control for the experiments conducted in this work. Punishing change of control in the objective function was a key factor in order to avoid excessive control usage.

This work was carried out as a proof of concept. The approach delivered a satisfactory operation and is promising for further development. The next section will discuss recommendations for future work.

6.1 Future work

This section will present proposals and recommendations for future work in order to increase the quality and achieve more realistic results. This work has been established on assumptions and simplifications in order to narrow the scope of the task.

6.1.1 Echo State Network

The ESN used in this work managed to identify the ESP satisfactorily, but it is necessary to reduce the size of the network to acquire a faster controller. It should also be developed a better method to access future states in the ESN, a task which was performed in a manual manner for this project.

6.1.2 Nonlinear Model Predictive Control

There are multiple areas that can be improved in the NMPC. Firstly, *multiple shooting* should be implemented for more speed and the opportunity to look further into future behavior.

A more suitable objective function is necessary in order to achieve more realistic results. Controlling the ESP intake pressure to a setpoint would be a better control target since the intake pressure directly affects the flow-rate from the reservoir. This target would allow the operator to control the production flow-rate. It is also natural to minimize the ESP frequency to minimize the power usage in the ESP. The change of ESP frequency is usually constrained in order to avoid fast changes in suction, while the change of the production choke valve opening is constrained by limits from the choke characteristics. These changes should be implemented as hard constraints established by the operators.

Lastly, there is no feedback implemented in the ESN. This will not cause any huge problems for a noiseless simulation, but once noise is introduced in the plant, a correction filter must be implemented. A correction filter is a low-pass filter that corrects the error between the predicted output and the current measured output. The filter will also play a part in correcting modeling errors.

Bibliography

[1] Z.-S. Hou and Z. Wang, "From model-based control to data-driven control: Survey, classification and perspective," *Information Sciences*, vol. 235, pp. 3–35, 2013.

[2] O. Nelles, *Nonlinear System Identification From Classical Approaches to Neural Networks and Fuzzy Models*. Springer, 2001.

[3] I. Osnes, "Model Predictive Control using data-driven models obtained from Artificial Neural Networks," Department of Engineering Cybernetics, NTNU – Norwegian University of Science and Technology, Specialization Project in TTK4551, 2019.

[4] E. Antonelo, E. Camponogara, and B. Foss, "Echo State Networks for data-driven downhole pressure estimation in gas-lift oil wells," *Neural Networks*, vol. 85, 2017.

[5] E. A. Antonelo and B. Schrauwen, "On Learning Navigation Behaviors for Small Mobile Robots With Reservoir Computing Architectures," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 4, pp. 763–780, 2015.

[6] U. Rosolia, X. Zhang, and F. Borrelli, "Data-Driven Predictive Control for Autonomous Systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 259–286, 2018.

[7] S. Gros and M. Zanon, "Data-driven Economic NMPC using Reinforcement Learning," *IEEE Transactions on Automatic Control*, 2019. DOI: 10.1109/tac.2019.2913768.

[8] *Centrilift Europe and Africa ESP Failures 1999-2008*, Centrilift, 2008.

[9] P. Delou, J. Azevedo, D. Krishnamoorthy, M. de Souza Jr, and A. Secchi, "Model predictive control with adaptive strategy applied to an electric submersible pump in a subsea environment," *IFAC-PapersOnLine*, vol. 52, pp. 784–789, 2019.

[10] B. J. Binder, K. M. Kufoalor, A. Pavlov, and T. A. Johansen, "Embedded Model Predictive Control for an Electric Submersible Pump on a Programmable Logic Controller," in *2014 IEEE Conference on Control Applications (CCA)*, IEEE, 2014, pp. 579–585.

[11] D. Krishnamoorthy, E. Bergheim, A. Pavlov, M. Fredriksen, and K. Fjalestad, "Modelling and robustness analysis of model predictive control for electrical submersible pump lifted heavy oil wells," *IFAC-PapersOnLine*, vol. 49, pp. 544–549, 2016.

[12] G. Takacs, *Electrical Submersible Pumps Manual: Design, Operations, and Maintenance.* San Diego: Elsevier Science, 2017.

[13] A. Pavlov, D. Krishnamoorthy, K. Fjalestad, E. Aske, and M. Fredriksen, "Modelling and Model Predictive Control of Oil Wells with Electric Submersible Pumps," in *2014 IEEE Conference on Control Applications (CCA)*, 2014, pp. 586–592.

[14] C. Chen, *Linear System Theory and Design*. Oxford University Press, 1999.

[15] H. Jaeger, "Tutorial on training Recurrent Neural Networks, covering BPPT, RTRL, EKF and the Echo State Network approach," *GMD-Forschungszentrum Informationstechnik*, vol. 5, 2002.

[16] P. Kunkel and V. Mehrmann, *Differential-Algebraic Equations, Analysis and Numerical Solution*. European Mathematical Society, 2006.

[17] I. Matei and C. Bock, "Modeling methodologies and simulation for dynamical systems," 2012.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[19] A. S. Walia, *Activation functions and it's types-which is better?* 2017. [Online]. Available: `https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8`.

[20] Jing Dai, Pinjia Zhang, J. Mazumdar, R. G. Harley, and G. K. Venayagamoorthy, "A comparison of MLP, RNN and ESN in determining harmonic contributions from nonlinear loads," in *2008 34th Annual Conference of IEEE Industrial Electronics*, 2008, pp. 3025–3032.

[21] S. Hochreiter, "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, pp. 107–116, 1998.

[22] U. Schiller and J. Steil, "Analyzing the weight dynamics of recurrent learning algorithms," *Neurocomputing*, vol. 63, pp. 5–23, 2005.

[23] G. Tanaka, T. Yamane, J. Heroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," 2018.

[24] M. Lukosevicius, "A Practical Guide to Applying Echo State Networks," in *Neural Networks: Tricks of the Trade*, 2012.

[25] Q. Ma, L. Shen, E. Chen, S. Tian, J. Wang, and G. W. Cottrell, "WALKING WALKing walking: Action recognition from action echoes," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 2457–2463.

[26] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 1999.

[27] L. Imsland, *Introduction to Model Predictive Control*, 2007.

[28] K. R. Muske and J. B. Rawlings, "Model Predictive Control with Linear Models," 1993.

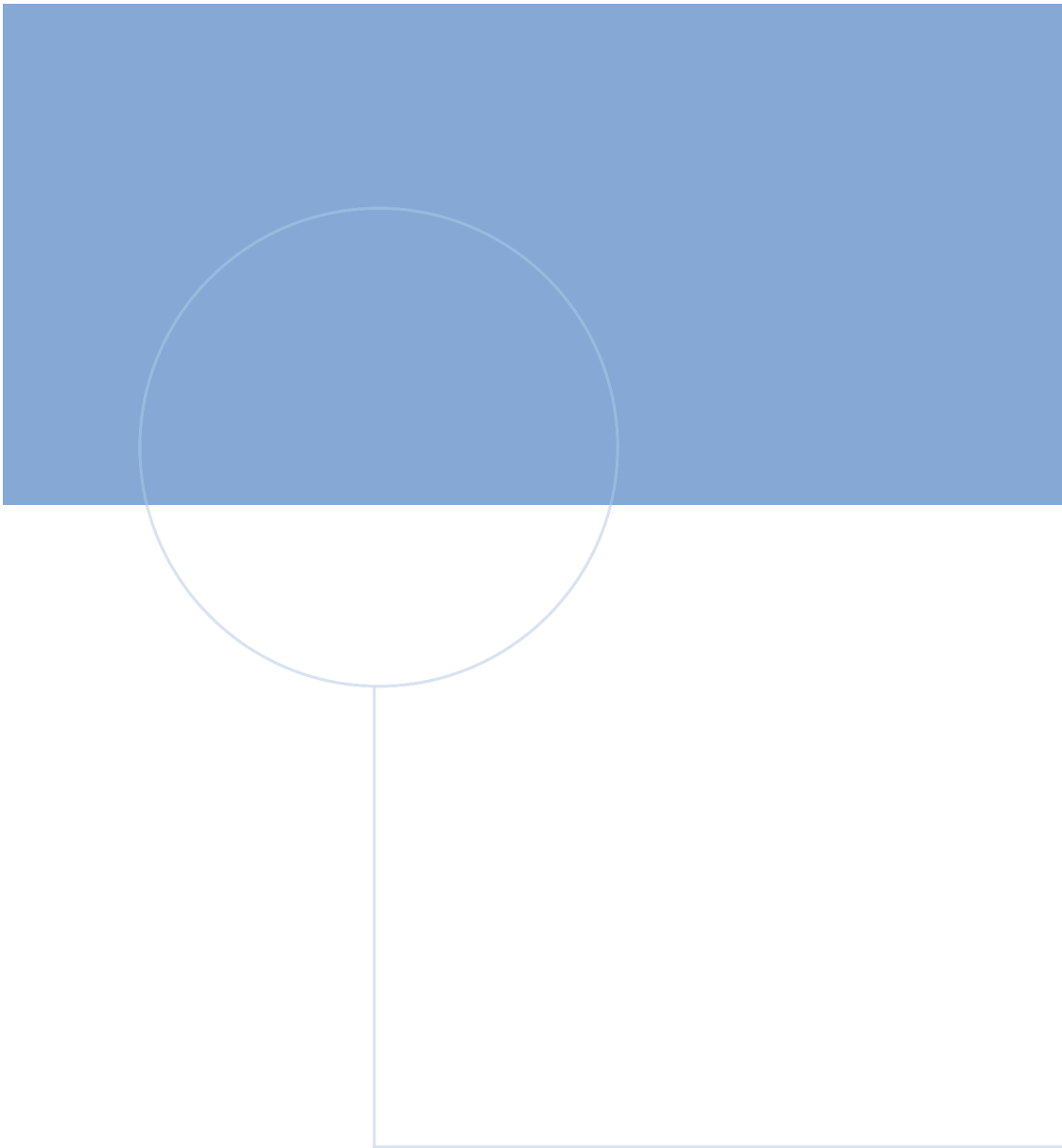
[29] B. Foss and T. A. N. Heirung, "Merging Optimization and Control," 2016.

[30] J. Jordanou, E. Camponogara, E. Antonelo, and M. A. Aguiar, "Nonlinear Model Predictive Control of an Oil Well with Echo State Networks," vol. 51, pp. 13–18, 2018.

[31] D. Verstraeten, B. Schrauwen, S. Dieleman, P. Brakel, P. Buteneers, and D. Pecevski, "Oger: Modular Learning Architectures For Large-Scale Sequential Processing," *The Journal of Machine Learning Research*, vol. 13, pp. 2995–2998, 2012.

[32] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, pp. 1–36, 2019.

[33] B. J. Binder, A. Pavlov, and T. A. Johansen, "Estimation of Flow Rate and Viscosity in a Well with an Electric Submersible Pump using Moving Horizon Estimation," IFAC-PapersOnLine, 2015, pp. 140–146.



NTNU

Norwegian University of
Science and Technology