NTNU
Kunnskap for en bedre verden

TTK4550: Specialization Report

# Black-Box Modeling of an Electric Submersible Pump Lifted Well Using an Echo State Network

*Author:*
Ola Solli Grønningsæter

Fall 2022

# Abstract

This report documents the conducted work with my specialization project, which was completed as a part of the Master's program in cybernetics and robotics at NTNU. The project serves as preliminary work to my Master's dissertation which will be submitted in July, 2023. Today, optimization plays an important role to maximize profit and production in modern industries. Moreover, it is probably more important now than ever to at the same time minimize climate footprints as the world is working towards a goal of net zero emissions. However, most optimization methods require accurate models to provide satisfactory results. Unfortunately, many modern processes have become too complex for sufficiently accurate models to be derived using first principles like mass balance. New modeling methods like black-box modeling have therefore become a subject of interest. Another characteristic of modern industry is the large amount of data that is being monitored and collected. This data has proven valuable when modeling using artificial intelligence (AI) which has been proven as a powerful black-box modeling approach in recent research. In this specialization project, a third-order nonlinear dynamical system was modeled by utilizing a modern recurrent neural network (RNN) architecture called an echo state network (ESN). The system of interest was an electric submersible pump (ESP) lifted well which is of great interest in modern oil industry. More specifically, a simulator for this system and a framework for the ESN were implemented. Then, another RNN architecture, a long short-term memory network (LSTM) was implemented to compare with the ESN. Both networks were able to successfully recognize the nonlinear behavior of the ESP lifted well with a total average mean absolute percentage error (MAPE) below 1% when compared to the simulator. Although the project did not show any significant difference in average performance, it did show that the ESN required far less training time with 3 seconds while the LSTM required 17.5 minutes. Furthermore, it also showed that the hyperparameters in the ESN were easier to obtain. The project thus concludes that the ESN is better suited than the LSTM when it comes to black-box modeling of nonlinear dynamical systems if a sufficient amount of data is available.

# Sammendrag

Denne rapporten dokumenterer arbeidet med spesialiseringsprosjektet mitt som ble fullført som en del av masterprogrammet i kybernetikk og robotikk ved NTNU. Prosjektet er på mange måter et forarbeid til masteroppgaven jeg skal levere juli 2023. Optimering spiller i dag en viktig rolle for å maksimere profitt og produksjon i moderne industrier. Dessuten er det antakelig viktigere enn noen gang å samtidig minimere klimaavtrykket, da verden jobber mot et mål om netto nullutslipp. Mange optimeringsmetoder krever imidlertid nøyaktige modeller for å gi tilfredsstillende resultater. Dessverre har mange moderne prosesser blitt for komplekse for at tilstrekkelig nøyaktige modeller kan utvikles ved hjelp av første prinsipper som massebalanse. Derfor har nye modelleringsmetoder som *black-box* modellering blitt et interessant tema. Et annet kjennetegn ved moderne industri er den store mengden data som samles. Denne dataen har vist seg nyttig for modellering ved hjelp av kunstig intelligens som i nyere forskning også har vist seg å være et kraftig redskap for *black-box* modellering. I dette spesialiseringsprosjektet ble et tredjeordens ulineært dynamisk system modellert ved hjelp av en moderne rekurrent nevralt nettverk (RNN) arkitektur med navn *echo state network* (ESN). Systemet som ble benyttet var en modell av en *electric submersible pump* (ESP) hevet oljebrønn. Denne installasjonen er av stor interesse i moderne oljeindustri. I løpet av arbeidet ble det implementert en simulator for dette systemet, samt et rammeverk for implementasjon av ESN. Videre ble også en annen RNN-arkitektur kalt *longs short-term model* (LSTM) implementert for sammenlikning med ESN. Begge nettverkene klarte å modellere den ulineære oppførselen til den ESP-hevede brønnen med en total gjennomsnittlig absolutt prosentvis feil (MAPE) på under 1% når sammenliknet med simulatoren. Til tross for at prosjektet ikke viste noen signifikant forskjell i gjennomsnittlig ytelse, viste det at ESN krevde langt mindre treningstid med 3 sekunder, mens LSTM trengte 17.5 minutter. Prosjektet viste også at det var enklere å finne hyperparametere til ESN. Prosjektet konkluderer derfor med at ESN er et bedre verktøy enn LSTM for *black-box* modellering av ulineære dynamiske systemer så lenge nok data er tilgjengelig.

# Contents

# List of Figures

# List of Tables

# Nomenclature

## Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| ML | Machine Learning |
| NN | Neural Network |
| ANN | Artificial Neural Network |
| RNN | Recursive Neural Network |
| ESN | Echo State Network |
| LSM | Liquid State Machine |
| LSTM | Long Short-Term Memory |
| BPTT | Back-Propagation Through Time |
| SGD | Stochastic Gradient Descent |
| Adam | Adaptive Moments |
| MSE | Mean Square Error |
| (N)RMSE | (Normalized) Root Mean Square Error |
| MAPE | Mean Absolute Percentage Error |
| PRBS | Pseudo-Random Bit Stream |
| APRBS | Amplitude-modulated Pseudo-Random Bit Stream |
| ReLU | Rectified Linear Unit |
| MBC | Model-Based Control |
| MPC | Model Predictive Control |
| ESP | Electric Submersible Pump |
| VSD | Variable Speed Drivers |
| DAE | Differential Algebraic Equation |
| ODE | Ordinary Differential Equation |
| BHP | Blowout Horsepower |
| VCF | Viscosity Correcting Factor |

# 1 Introduction

## 1.1 Background and Motivation

In many real-world industries, it is both desired and necessary to not only control, but also optimize either parts of or the entire industrial process. Oil and gas production is one such industry that over the years has evolved to be one of the most important and economically profitable industries in the world (Stu, 2018). Because of huge economical interest and the global energy demand, oil production has become a subject of high interest. A lot of research in the field has provided advanced equipment and technology to maximize production and minimize costs. Another important aspect of optimization in the context of oil production is the carbon footprint that the industry leaves. It delivers one of the main sources of energy to the world, but the emission from burning fossil fuels is also one of the largest contributors to climate change (Perera and Nadeau, 2022). Even though the world is working towards phasing out fossil fuels, they will most likely remain important energy sources during this transition period and even after. Thus, it is desirable to optimize oil and gas production in a way that minimizes the environmental damage.

Model-Based Control (MBC) such as Model Predictive Control (MPC) is in general a powerful method for controlling a system optimally, but requires an accurate system model. Without this, MBC strategies may result in bad performance and perhaps instability in the closed-loop system (Hou and Wang, 2013). In the early days of MBC, these models were often derived using first principles such as mass balance and Newton's laws of motion. However, today, most processes have become too complex for being derived using such methods (Hou and Wang, 2013). The inability to derive models based on first principles also applies to processes in the oil industry. This is a consequence of many factors such as that plants tend to change over time due to wells being drained or that many production sites are located off-shore in harsh weather conditions. Models used for optimizing such processes are therefore often based on parameter estimation from system identification, or a combination of that and first principle methods (Egeland and Gravdahl, 2003). Over the last decade, there have been major developments in monitoring and storing of data produced in industrial processes (Zambrano et al., 2022). According to the same article, this is one of the characteristics of the fourth industrial revolution that many believe has begun. The large amount of available process data and several recent breakthroughs in deep learning (Grimstad, 2022) offer powerful approaches to obtain accurate models. This is often referred to as *black-box* modeling due to the ability of artificial intelligence (AI) to recognize high dimensional mappings between in- and outputs without any prior knowledge. These mappings would be impossible to derive using first principles like Newton's laws.

Recurrent neural networks (RNNs) are perhaps one of the most suited approaches for black-box modeling of dynamical systems using AI. This is because these networks are specialized in processing sequential data by being able to consider long-term dependencies (Goodfellow et al., 2016). Yet, they have proven particularly difficult to train because this produces a non-convex optimization problem which is NP-hard (Grimstad, 2022). These networks tend therefore to be computationally too expensive to train. An

alternative approach is the echo state network (ESN) which was the subject of interest in two previous master dissertations; (Osnes, 2020) and (Hernes, 2020). This network is an RNN architecture that only adapts the output weights by using linear regression. By doing so, it becomes significantly faster to train than other RNN architectures, making it more promising for deriving data-driven black-box models (Goodfellow et al., 2016).

## 1.2 Research Objectives and Research Questions

This specialization report documents the preliminary work of my master's dissertation which will be submitted in July, 2023. It will both serve as a preparation and an introduction to the concepts that will be encountered in the master's dissertation. The primary objective is thus to get a thorough understanding of the Electric Submersible Pump (ESP) and the ESN, and to successfully implement

- a simulator for an ESP lifted well.
- an ESN and a framework to support future implementations of the ESN.

Furthermore, the simulator will be used to generate data points which will be used to train an ESN model of the ESP lifted well. Then, a more traditional RNN architecture, the Long Short-Term Memory (LSTM) network, will be implemented and also used to model the ESP. Finally, the performance of the ESN and the LSTM models of the ESP lifted well will be compared. This leaves the two following secondary objectives:

- Implement an LSTM and use it to model the ESP lifted well.
- Compare the performance of the resulting ESP models.

It is also expected that these objectives will give important hands-on experience with popular frameworks for implementation of neural networks and simulation of dynamical systems such as *PyTorch*, *NumPy* and *CasADI*. The specialization project will also attempt to answer the following research questions:

- Can the nonlinear behavior of the ESP lifted well be successfully modeled using an ESN?
- Is there any significant difference in performance between a model obtained from the ESN and the LSTM?
- Which network is better suited for black-box modeling of nonlinear dynamical systems?

## 1.3 Structure of the Report

The report consists of six sections with the intention of documenting all aspects of the work conducted during the specialization project in *TTK4555*. This first introductory section is followed by:

***Section 2*** contains existing theory on the encountered fields in the project. The literature from which parts of the theory are found is cited using APA-in-text citations and all references are listed at the very end of the report.

***Section 3*** documents the methods and implementations used to obtain the final results. This section should enable the reader to reproduce the results that were obtained. Additional theoretical depth is given at some places to supplement the details of the presented methods.

***Section 4*** presents the conducted experiments and the obtained results of the project.

***Section 5*** discusses the experiments and the results from section 4.

***Section 6*** concludes upon the project using the discussion from section 5 as a basis. This section will at last propose some subjects of interest for future work.

# 2 Theory

## 2.1 Electric Submersible Pump

### 2.1.1 Introduction

Oil wells (reservoirs) are generally categorized as alive or dead based on how oil can be produced from them. An alive oil well has a sufficiently high pressure which ensures a natural and proper flow of oil, water and gas (production fluid) to the surface. These wells are often referred to as having "natural lift", and these wells are naturally suited for economical production. A dead oil well, on the other hand, is an oil well where the natural pressure is insufficient to sustain a proper flow of production fluid to the surface. These wells are most often unsuited for production economical at all. However, it is possible to turn these wells into alive wells by artificially increasing the pressure in the well. This is referred to as "artificial lifting" and can ensure economical production of oil and gas from dead wells. ESP installations are a technology used for artificial lifting (Pavlov et al., 2014), and this will be the production process studied in this specialization report. In this section, artificial lifting will first be presented in more detail, before a thorough introduction to the ESP components and their applications will be given. The following three subchapters are mainly based on (Takács, 2009).

### 2.1.2 Artificial Lifting

Most wells will in their early stages have natural lift and thus be "flowing wells" as fluid flows naturally to the surface. In these wells, the natural pressure at the well bottom is greater than the pressure loss over the flow path to the separator. However, when producing oil and gas, fluid is extracted from the well over time. This leads to the pressure decreasing resulting in the natural pressure eventually being lower than the pressure loss. At this point, the well dies as the natural lift becomes insufficient to sustain a proper flow of fluid to the surface. Alternatively, the well might die due to the pressure loss becoming greater than the bottom pressure even though the bottom pressure is theoretically sufficient. Increased pressure loss involves increased flow resistance in the well which is usually caused by increased density of the fluid because of decreased gas production or mechanical problems such as too small tubing size. Artificial lifting can be achieved using various methods with the main objective of producing fluids from already dead wells. It can also be used to increase the production rate from flowing wells. The two main methods are gas lifting and pump lifting.

Gas lifting can be divided into continuous and intermittent gas lift even though both techniques involve injecting high-pressure natural gas into the well stream. When applying a continuous gas lift, the main objective is to reduce the pressure loss occurring along the flow path by aerating the fluid. This will reduce the flowing mixture density and thus make the well pressure sufficient to revive the well flow. Continuous flow gas lifting is therefore often considered as a continuation of flow production. In the other method, intermittent gas lift, gas is periodically injected when a sufficient length of fluid has accumulated at the well bottom. The fluid is then pushed to the surface as a slug, making the production a batch production. This method can also be augmented to include a free plunger traveling in the well tubing to separate the upward-moving fluid

slug from the gas. Unlike continuous flow gas lifting, this method physically displaces the accumulated fluids from the well.

Pumping methods use pumps in the well bottom to increase the pressure in the reservoir to overcome the flowing pressure loss. In general, pumping methods are classified based on how the pump is driven and whether it uses rod or rodless pumping. There are many types of pumping methods involving rod pumps, but they all have in common that they make an oscillating or rotating movement to increase the pressure in the reservoir and drive fluid to the surface. Rodless pumping, on the other hand, uses electric, hydraulic and similar means to drive the downhole pump or employs high-pressure power fluid that is pumped down the hole. The ESP is a rodless pumping method that utilizes a submerged electrical motor that drives a multistage centrifugal pump. The ESP is ideally suited to produce high liquid volumes and is one of the most efficient pumps when considering all depths. Note that gas lifting is still more efficient.

### 2.1.3 History and Challenges

The ESP was invented by Armiais Arutunoff in the late 1910s, and the first ESP was successfully operated in the El Dorado field, Kansas in 1926. Since then, there have been multiple improvements to the technology, and it is approximated that around 10% of the world's oil supply is produced with ESP today. The ESP is therefore considered by many as one of the most successful techniques for artificial lifting both on- and off-shore with high liquid volumes from medium depths. This is a result of the ESPs being both energy efficient and easy to install with low demand for maintenance as long as it is properly installed and operated.

There are, however, some disadvantages with the ESP. One of these is the difficulty of repairing faulty equipment in the field. In most cases, faulty equipment must be sent back to the manufacturer for repair which means downtime in the production since the equipment cannot be replaced during this period. To make matters worse, the repair in itself is expensive and comes in addition to the multi-million loss due to the downtime. Hence, it is desirable to make the lifetime of the ESP as long as possible and the need for maintenance as small as possible. Available statistics show that 23% of all ESP failures are caused by operator mistakes (Sta, 2008). Although this is a large improvement from the early days when around 80% of all failures were due to human error, this is still causing enormous economical losses. The reason for these failures is the ESPs being run close to the operating limits because this is where optimal production points are located. There is in general little automation supporting the operation of the pump. Many ESPs are driven by variable speed drivers (VSD) enabling operators to run the ESP motor at different speeds. They are also able to adjust the opening of the production choke at the top of the well affecting both the well and the operation of the ESP. Hence, it is hard to find an optimal operation point where the production is optimized and the lifetime is not significantly reduced. There are usually large teams with competence on the ESP monitoring multiple variables and doing analyses to support the operators. However, this is prone to human error due to differences in experience, and the fact that most fields operate multiple ESPs making the task even more complex (Pavlov et al., 2014).

### 2.1.4 Components and Their Functionality

As mentioned in chapter 2.1.2, ESPs are installations utilizing a submerged electrical motor (powered by electricity from the surface) that drives a multistage centrifugal pump. The conventional ESP installation can be seen in figure 1. Generally, the ESP unit is submerged in well fluid and consists of a motor, a protector, a gas separator and a pump. On the surface, one can usually find a junction box where surface and submerged electric cables are joined, and a control unit labeled *switchboard* in the figure. This installation is still frequently used today as it has proven both effective and reliable under three assumptions:

1. Ideal pump conditions with only fluid entering the pump.
2. Produced fluid has a low viscosity (ideally close to the viscosity of water).
3. The motor is operating at a constant speed as it is powered with AC having a constant frequency. Also ensuring constant speed of the pump.

Even though these assumptions do not always hold, the conventional ESP installation has proven reliable in various conditions. In special cases, where for example the gas production is too large and breaks the first assumption, the ESP installation can be augmented with special equipment. Installations different from the conventional are, however, outside the scope of the specialization report. The next paragraphs will give a more thorough presentation of the submerged components of the conventional ESP installation.

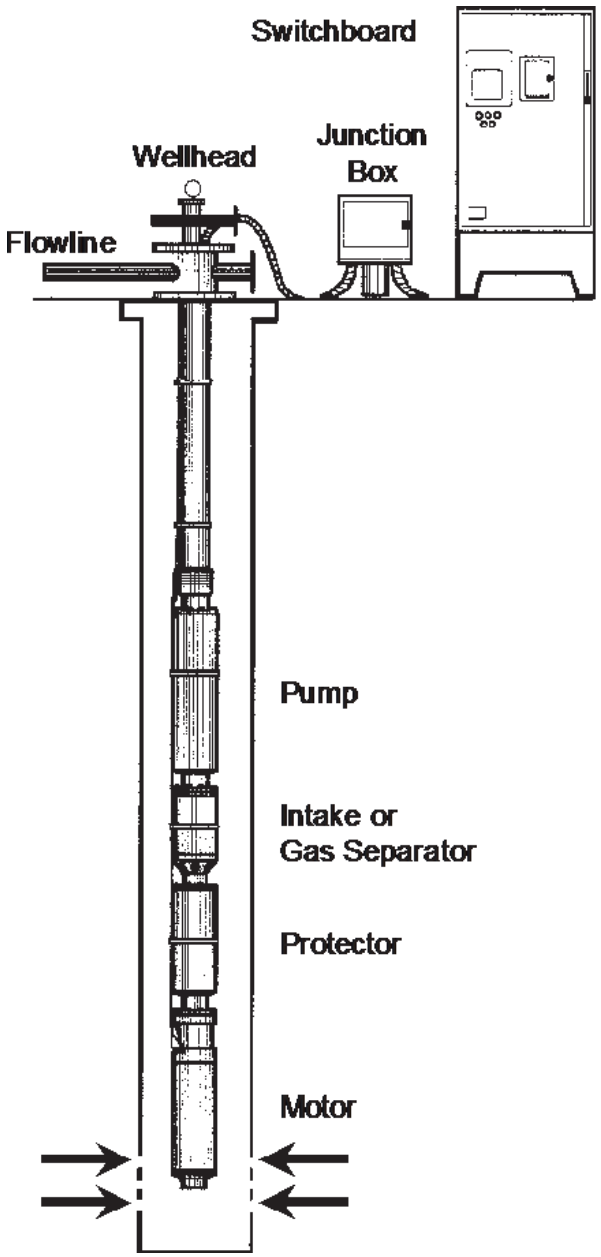



**Figure 1:** Conventional ESP installation borrowed from (Diaz and Nicolas, 2012).

The most important component of the ESP installation is the submersible pumps which for ESPs are multistage centrifugal pumps operated vertically. The basic principle of centrifugal pumps is that the produced fluids are transferred rotational energy from rotors (called impellers) increasing the fluid's velocity and pressure. This is the main operational mechanism of radial and mixed-flow pumps. It is the density of the fluid that decides the amount of rotational energy transferred. Since fluids have a much greater density than gas, they will also receive more rotational energy and thus obtain higher velocity and pressure. Reliable operation of the pump (and thereby the ESP) is therefore dependent on the gas separator in order to feed the pump with gas-free fluid. The main concerns of free gas in the fluid are decreased performance, mechanical damage due to cavitation at high flow rates and gas locking caused by the pump being filled with gas. Note that there exists equipment for operating ESPs with a large amount of

free gas, but this is outside the scope of this report and can be read in detail in (Takács, 2009). The conventional installation utilizes a reverse-flow gas separator assuming that gas rates are low. Note that this is one of the least efficient and simplest gas separators, but it is considered sufficient to illustrate the principles of the ESP installation. This gas separator works by directing the gas into the well's casing annulus given that the casing annulus is not sealed off by a packer. More specifically, it uses gravitational separation by changing the direction of the fluid flow to allow the gas to escape.

In the conventional installation, three-phase, two-pole, squirrel cage induction motors are the most used motors. These motors are one of the simpler electric motors and are often durable as the electric supply is not directly connected to the rotor. They are also highly efficient and generally popular in oilfield installations. The construction of this motor can be read in detail in (Takács, 2009). Although the rotor of this motor can be made in one piece, it is made up of short segments with radial bearings in between when constructed for ESPs. This is to eliminate the radial instability due to the high rotational speed and the slenderness of rotors made of one piece. Moreover, because of the length of the motor shaft, radial vibrations must be sufficiently suppressed. This is done by using multiple radial bearings along the shaft length. Furthermore, the motor is filled with highly refined oil which circulates and provides the required dielectric strength to prevent short circuits between motor parts, proper lubrication for the bearings and sufficient thermal conductivity to carry heat out of the housing. This heat is transferred to the well stream and removed from the installation. The motor shaft also has a filter to remove solid particles from the oil. There are some significant differences between electrical motors used in ESPs and on land, and the most important ones were listed in the standard *API RP 11S4, 3rd Ed* given by the American Petroleum Institute in 2002 regarding recommended practice for sizing and selection of electric submersible pump installations as follows:

- Greater length-to-diameter ratio since they must be run inside the well's casing string.
- Motor power can only be increased by increasing the length of the unit.
- ESP motors are cooled by the convective heat transfer with the fluid flowing past the motor while surface motors are usually cooled by the surrounding air.
- More than ten times higher electric current densities can be used in ESP motors without overheating due to the significantly higher cooling effect of fluids (compared to air).
- ESP motors have exceptionally low inertia and accelerate to full speed in less than 0.2 seconds when starting.
- ESP motors are connected to their power source by long well cables and are prone to substantial voltage drop.

Note that this standard was revoked in 2013 and has also been substituted since then.

Between the motor and the gas separator in figure 1, it is also illustrated a protector. This component is used to protect the motor that would have been sealed if operated on the surface. ESP motors must be open to the surrounding in order to not blow up due to the expansion of dielectric oil at elevated temperatures. The main tasks of the protector are to

- Ensure that no axial thrust load develops in the ESP pump's stages.
- Isolate the clean dielectric oil from the well fluid.
- Allow expansion and contraction of the oil inside the motor.
- Equalize the pressure between the well fluids and the dielectric oil in the motor.
- Provide a mechanical connection between the pump and motor and transmit the torque produced by the motor to the pump shaft.

## 2.2 Modeling of Dynamical Systems Using Neural Networks

### 2.2.1 Introduction

Within deep learning, there are many different types of neural networks (NN), and RNNs are perhaps the most popular architecture for sequential data processing. The ESN is a subclass of this architecture and provides a supervised learning principle for the RNN. During the early 1990s, the general idea of the ESN was proposed within neuroscience and AI. Yet, the ESN was not adapted into machine learning (ML) before somewhere around 2000 by Herbert Jaeger (Jaeger, 2007). He is considered one of the pioneers within *reservoir computing* which mostly involves ESNs and liquid state machines (LSMs). The latter was developed at the same time by Wolfgang Maass and can be read more about in (Maass et al., 2002). Although it is based on the same principles as the ESN, it is outside the scope of the specialization report and will not be considered any further. Together with Harald Haas in 2004, Jaeger demonstrated that the ESN could learn chaotic time series prediction tasks with much greater precision than any previous method. In their paper (Jaeger and Haas, 2004), they outperformed previous methods with 2 orders of magnitude better signal-to-noise ratio performance on a nonlinear satellite communication benchmark task (Jaeger, 2007). Their demonstration yielded a black-box modeling tool for nonlinear dynamical systems. This is of great interest within engineering because most technical systems become nonlinear when operated close to saturation, and a system model is needed to simulate and control such systems (Jaeger and Haas, 2004). In the following subchapters, black-box modeling, RNNs, ESN and LSTM will be presented in more detail.

### 2.2.2 Differential Equations and Black-Box Modeling

Differential equations can be used to describe how states or variables vary over time (Gravdahl, 2019). These equations enable a dense description of system state-spaces, which also can be transformed into the complex domain resulting in transfer functions. Hence, differential equations are powerful mathematical tools when it comes to modeling physical systems. A state-space can be expressed using *ordinary* differential equations (ODE) which generally relate the output $y$ of a system to the input $u$ (Gros, 2021). These equations are typically given as:

$$\begin{aligned} \dot{x} &= f(x, u) \\ y &= h(x, u) \end{aligned} \tag{1}$$

where $f$ and $h$ can be either linear or nonlinear. Another benefit of using ODEs is that these systems are generally straightforward to express in discrete time enabling them to be solved numerically. There is, however, not guaranteed that there exist ODEs defining the entire state directly (Gros, 2021). Another type of differential equations omitting this problem is differential algebraic equations (DAEs). These are, simply put, systems of differential equations and algebraic equations and are typically given by:

$$\begin{aligned} \dot{x} &= f(x, y) \\ 0 &= g(x, y) \end{aligned} \tag{2}$$

DAEs will be used to implement the ESP simulator in this work.

Ultimately, when modeling a dynamical system, the main objective becomes to find the differential equation(s) describing the system at hand. This can, however, be very challenging and will in most cases require system data to estimate parameters. This data can be obtained through system identification which refers to repeatedly exciting a system with an input $u$, measuring the output $y$ and finding some function relating the in- and outputs (Nelles, 2013). Furthermore, the process of finding this function is divided into three categories in (Nelles, 2013). These are *white*, *gray* and *black* box modeling. The first category refers to models derived from using *first principles* which are utilizing theoretical laws such as Newton's laws of motion, mass balances, etc. The latter refers to modeling using only measured data. This is the reason why it is called a "black" box because very little prior knowledge can be exploited, and it is almost impossible to tell how the in- and outputs are related. An example of black-box modeling is using machine learning to learn the system model. Gray box modeling is modeling techniques using a combination of white- and black-box modeling.

### 2.2.3 Perceptrons and Artificial Neural Networks

The desire to build an intelligent machine (or "create artificial life") goes back to at least ancient Greece. There is, however, little doubt that it dates back even further. Although this was impossible to achieve due to that time's technological development, the idea still may have been an instigator for today's technological achievements. In the beginning of artificial intelligence (AI), as we know it today, AI proved very efficient to solve problems considered intellectually difficult for humans. These problems are typically easily formulated mathematically. The real challenge turned out to be using AIs to solve problems that are hard to formulate mathematically, but easy to solve for humans. That is for example face and speech recognition. Recent research on deep learning has shown that AIs can contribute a lot to these tasks, even though it is still unclear if a self-thinking machine as described in ancient times would ever be achievable (Goodfellow et al., 2016).

Artificial neural networks (ANNs) are inspired by the human brain. In the same way, as the brain consists of networks with neuron cells, all ANNs are composed of layers with *perceptrons* which are an oversimplified model of the neuron cells. Hence, why it is called an *artificial* neural network (Langseth, 2022). Figure 2 shows a schematic representation of the perceptron.

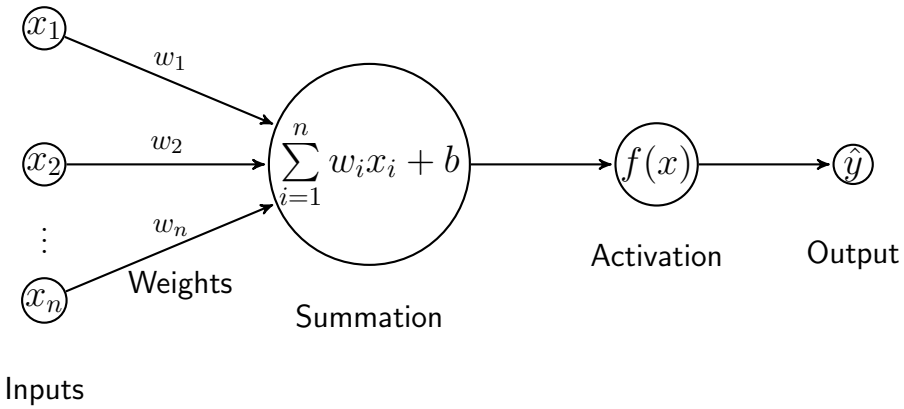



**Figure 2:** Schematic model of a perceptron inspired by a figure from a lecture by (Langseth, 2022).

Following the figure from left to right, every input is multiplied with a corresponding weight before they are summed. Then, the resulting sum is added a bias (denoted $b$ in the figure) before being evaluated in an activation function. There exist many activation functions that can be utilized, but they all define a threshold on whether the output is 0 or 1. One can think of a perceptron as a function mapping an input to an output described by (Langseth, 2022):

$$\hat{y} = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{3}$$

The activation function introduces parameter-free nonlinearity to the network, enabling it to learn complex mappings between the in- and outputs. Hence, without this function, the perceptron would only yield linear outputs. Three popular activation functions are the hyperbolic tangent, the sigmoid function and the rectified linear unit (ReLU) which are given in (4a)-(4c), respectively (Grimstad, 2022). These functions are also plotted in figure 3. Note that a step function could also have been used, but is less desirable because it does not work well with back-propagation which is an important part of the training of many NN architectures (Langseth, 2022).

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{4a}$$

$$sig(x) = \frac{1}{1 + e^{-x}} \tag{4b}$$

$$ReLU(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{4c}$$

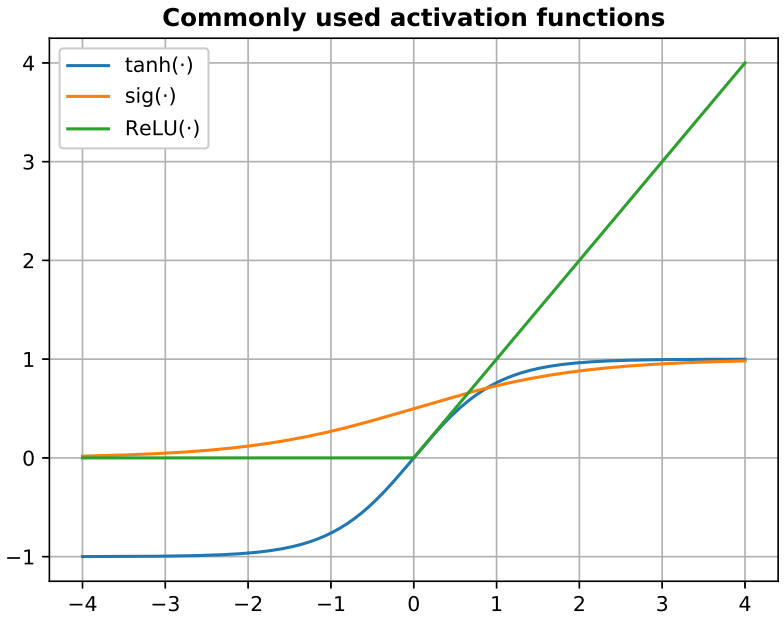



**Figure 3:** Plot of popular activation functions. These are the hyperbolic tangent, the sigmoid function and the ReLU function.

A single perceptron can be considered a single-layer ANN, only yielding a linear classifier. This means that it can only represent simple logical operators such as AND and OR. It is, however, insufficient to represent more sophisticated operators such as XOR (Goodfellow et al., 2016). However, by making networks of multiple layers with perceptrons, more advanced patterns can be represented. An ANN with a single hidden layer is often referred to as a NN because it does not have very much representative power. By increasing the number of hidden layers, deeper networks are obtained. Figure 4 shows a deep neural network with two hidden layers, often called a deep feedforward network.

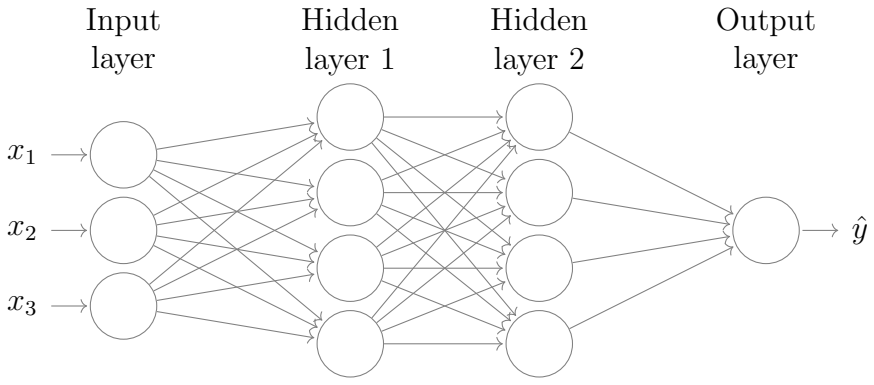



**Figure 4:** Schematic model of a general deep feedforward network.

The name comes from the nature of an input $\boldsymbol{x}$ flowing from the input layer, through the hidden layers which are defined by a function $f$ before it reaches the output layer

$\hat{\boldsymbol{y}}$. The overall objective of the network is to approximate the function $f^*$ such that $\hat{\boldsymbol{y}} = f^*(\boldsymbol{x})$. This is why the NN is said to represent a mapping $\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$.

### 2.2.4 Training of Artificial Neural Networks

The training of an NN representing a mapping $\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$ is equivalent to finding the values of $\boldsymbol{\theta}$ that results in the best approximation of $f$. Since most machine learning problems can be rewritten as a finite sum optimization, training can be conducted by minimizing this finite sum:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{5}$$

where $J(\boldsymbol{\theta})$ is a cost function (Grimstad, 2022). Unlike other ML problems, cost functions associated with deep NNs are non-convex and high-dimensional making this problem NP-hard. There is therefore no general method for finding the global minima for this problem. However, according to the authors of (Goodfellow et al., 2016), for sufficiently sufficiently deep NNs, most local minima will have a low-cost function value. This means that it will suffice to search for local minima in most cases which is achievable with methods such as gradient descent and Newton methods, with gradient descent methods being the most popular (Grimstad, 2022).

### 2.2.5 Gradient Decent and Back-Propagation

During the training of a deep NN, the gradient descent uses the cost function gradient to update $\boldsymbol{\theta}$. Calculating this gradient can be extremely expensive computationally, and it is usually calculated by using back-propagation algorithms instead. Given some training data and using figure 4 as an example, a cost $J(\boldsymbol{\theta})$ is first found through forward-propagation. This equivalent with information flowing from $\boldsymbol{x}$ to $\hat{\boldsymbol{y}}$. Then, the information flow is reversed using a back-propagation algorithm which allows efficient calculation of the cost gradient (Grimstad, 2022). Note that there is a lot more to this method that can be read in (Goodfellow et al., 2016).

### 2.2.6 Recursive Neural Networks

If feedback is included in the feedforward network described in chapter 2.2.3, it is called a *recursive* neural network. These networks have at least one cyclic path between their neurons giving them memory. This feature enables the network to model dynamical systems making it very suitable to process sequential data:

$$\begin{aligned} \boldsymbol{x}_{t+1} &= f(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\theta}) \\ \boldsymbol{y}_{t+1} &= g(\boldsymbol{x}_{t+1}) \end{aligned} \tag{6}$$

where $f(\cdot)$ is a function mapping the state at time $t$ to $t+1$ and $g(\cdot)$ is a function producing the output from the next state (Grimstad, 2022). The state equation in the RNN can be further written as

$$\boldsymbol{x}_{t+1} = h(\boldsymbol{x}_{t+1}, \boldsymbol{x}_t, \boldsymbol{x}_{t-1}, ..., \boldsymbol{x}_2, \boldsymbol{x}_1) = f(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\theta}) \tag{7}$$

where the $h(\cdot)$ takes all previous states as input and produces the next state. This represents the unfolded recurrent structure that enables $h(\cdot)$ to be factorized into repeated applications of $f(\cdot)$. The advantage of this structure is that the learned model always has the same input size, and it makes it possible to use the same transition function $f(\cdot)$ at every timestep. Furthermore, unfolding allows sharing parameters across the whole structure of a deep RNN (Goodfellow et al., 2016). Figure 5 shows how the RNN unfolding works schematically. Note that the black box illustrates one timestep with delay.

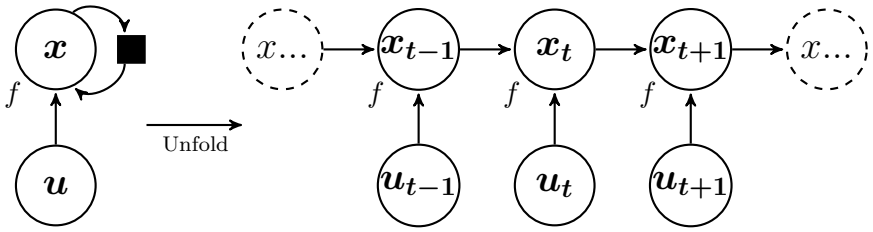



**Figure 5:** Schematic illustration of unfolding in RNN inspired by a figure in (Goodfellow et al., 2016).

The general RNN, however, is given by the following equations:

$$\begin{aligned} \boldsymbol{x}_{t+1} &= f(\boldsymbol{W}_h \boldsymbol{x}_t + \boldsymbol{W}_i \boldsymbol{u}_t) \\ \hat{\boldsymbol{y}}_{t+1} &= \boldsymbol{W}_o \boldsymbol{x}_{t+1} \end{aligned} \tag{8}$$

where $f(\cdot)$ is an activation function and $\boldsymbol{W}_i$, $\boldsymbol{W}_h$ and $\boldsymbol{W}_o$ are the weights of the input layer, hidden layers and the output layer, respectively. A schematic illustration of the general RNN can be obtained by introducing equation (8) to figure 5. This is shown in figure 6. Note that unfolding is not shown in this figure and that $\boldsymbol{L}$ is a loss function measuring the difference between the RNN output $\hat{\boldsymbol{y}}$ and the actual output $\boldsymbol{y}$. This measurement is used when training the network which corresponds to finding the weights that minimize the difference between these values. Traditionally, this is done by using back-propagation, but is in practice harder compared to static ANNs due to the time dependency between the states.

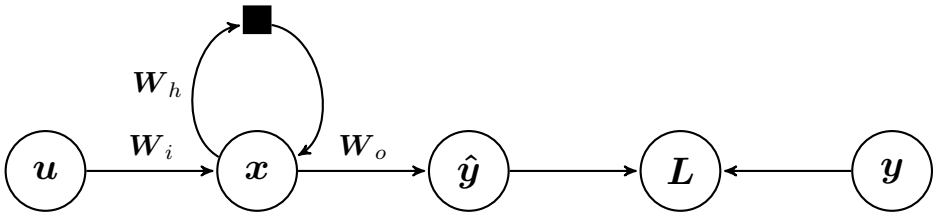



**Figure 6:** Schematic illustration of the general RNN inspired by a figure in (Goodfellow et al., 2016).

## 2.3 Echo State Networks

### 2.3.1 Introduction

The ESN is, as previously mentioned, an RNN using reservoir computation instead of multiple hidden layers. In reservoir computing, input and reservoir weights are randomly initialized entailing faster training and less computational cost compared to the RNNs where all weights are trained. This approach was proven eligible in (Schiller and Steil, 2005) where the authors showed that nearly all significant weight adaption happens in the output layer during the standard training of RNNs. In terms of structure, an ESN consists of a random input layer, the reservoir which acts as a hidden layer and an output layer that is trained. This is illustrated in figure 7. Note that ESNs can also be augmented to have feedback to the reservoir. This is not shown in the figure and is outside the scope of this specialization report.

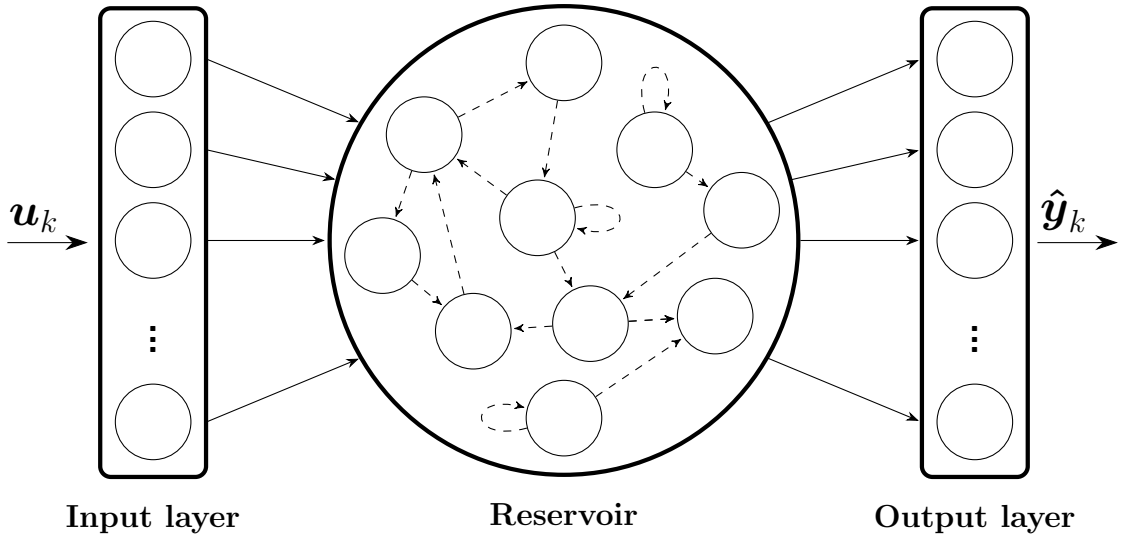



**Figure 7:** Schematic illustration of the general ESN inspired by a figure in (Jaeger, 2007).

### 2.3.2 Structure

ESNs can to some extent be considered specialized networks for fast learning of complex dynamical systems with low computational cost. This is not only due to the exploitation of reservoir computation, but also the use of the supervised learning principle. The formal task of an ESN was formulated in (Jaeger, 2001):

*Given a teacher I/O time series ($\boldsymbol{u}_{teach,k}$, $\boldsymbol{y}_{teach,k}$) for k=0,...,T, where the inputs come from a compact set $\boldsymbol{U}_{in}$ and the desired outputs $\boldsymbol{y}_{teach,k}$ from a compact set $\boldsymbol{U}_{out}$. A RNN whose output $\hat{\boldsymbol{y}}_k$ approximates $\boldsymbol{y}_{teach,k}$ is wanted.*

The ESN can in the same way as RNNs be described by a dynamical system. This system without feedback is given by the following equations:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= (1-\alpha)\boldsymbol{x}_k + \alpha f(\boldsymbol{W}_r \boldsymbol{x}_k + \boldsymbol{W}_i \boldsymbol{u}_{k+1} + \boldsymbol{W}_b) \\ \boldsymbol{y}_{k+1} &= \boldsymbol{W}_o \boldsymbol{x}_{k+1} \end{aligned} \tag{9}$$

where $\boldsymbol{x}_k$, $\boldsymbol{u}_k$, $\boldsymbol{y}_k$ are the state of the reservoir neurons, the values of the input neurons and the values of the output neurons, respectively. $\boldsymbol{W}$ represents the weight matrix with the subscripts $i$, $r$, $o$ and $b$ being input, reservoir, output and bias. The remaining two properties, $\alpha$ and $f(\cdot)$ correspond to the leak rate (which will be covered later) and the nonlinear activation function (Lukoševičius, 2012). Furthermore, according to (Lukoševičius, 2012), the general method of reservoir computing introduced with ESNs is:

1. Define $\boldsymbol{W}_i$, $\boldsymbol{W}_r$ and $\alpha$, and generate a random reservoir RNN.
2. Run the reservoir using the training input $\boldsymbol{u}_k$ and collect the corresponding reservoir activation states $\boldsymbol{x}_k$;
3. Compute $\boldsymbol{W}_o$ by minimizing the mean square error (MSE) between $\hat{\boldsymbol{y}}_{k+1}$ and $\boldsymbol{y}_k^{target}$.
4. Use the trained network on new input data $\boldsymbol{u}_k$ by computing $\hat{\boldsymbol{y}}_{k+1}$ using the computed $\boldsymbol{W}_o$.

When building an ESN, there are five hyperparameters that must be considered. These are reservoir size, sparsity, leak rate, spectral radius and input scaling. Unfortunately, there is no analytic way of finding these parameters, and it is therefore often done by trial and error (Lukoševičius, 2012). Note that automated methods such as grid search can also be applied (Grimstad, 2022). The following five subsections are based on (Lukoševičius, 2012) and will describe all hyperparameters in more detail.

### 2.3.3 Reservoir Size

The reservoir size refers to the number of nodes (often denoted $N_x$) in the reservoir. In general, one should use as many nodes as computationally affordable which makes it more likely to find a linear combination of reservoir signals $\boldsymbol{x}_k$ approximating $\boldsymbol{y}_k^{target}$. Yet, it can be cumbersome to determine the remaining hyperparameters if starting with a large reservoir. It is therefore recommended to start with a relatively small reservoir and determine the other hyperparameters before scaling up. Note that large reservoirs require appropriate regularization measures to avoid overfitting, and it is not always necessary to scale up the reservoir if a smaller reservoir yields satisfactory results.

### 2.3.4 Sparsity

Sparsity is relevant for both the input layer and the reservoir. In the input layer, it is recommended to make most of the values in $\boldsymbol{W}_i$ equal or as close to zero as possible. The performance of the ESN is generally not affected by sparsity in the reservoir. Hence, this hyperparameter is mostly used to speed up computations since it reduces the number of connections in the reservoir making the state matrix more sparse.

### 2.3.5 Spectral Radius

The spectral radius $\rho(\boldsymbol{W}_r)$ is perhaps the most important hyperparameter because it scales the maximum absolute eigenvalue of the reservoir weight matrix. This is equivalent to scaling the width of the distribution of its nonzero elements. After $\boldsymbol{W}_r$ is generated randomly, it should be divided by $max(|eig(\boldsymbol{W}_r)|)$ ensuring unit spectral radius before it is scaled with the spectral radius. This is to ensure that the ESN satisfies the echo state property being that the reservoir is uniquely defined by the fading history of the input. Usually, the echo state property is ensured by choosing $\rho(\boldsymbol{W}_r) < 1$. For practical purposes, the spectral radius is selected to maximize the performance, and it determines how fast the influence of an input dies and the stability of the reservoir activation. Generally, the spectral radius should be greater in tasks requiring longer memory of the input.

### 2.3.6 Leak Rate

The leak rate can be considered as the speed of the reservoir update dynamics. Note that the reservoir has no time constant and instead uses leaky nodes to slow down the dynamics in the system at hand. In (Osnes, 2020), the author describes the leak rate as *how much of the current state that will be "leaked" to the next*. This seems like a good interpretation considering its role in (9) and that its value ranges from zero to one. By further inspecting (9) it is clear that a low leak rate slows down the dynamics and increases the memory as more of the current state is leaked to the next. Tuning this parameter is mostly done by trial and error.

### 2.3.7 Input Scaling

Input scaling determines the scaling of the input weight $\boldsymbol{W}_i$ affecting how nonlinear the reservoir responses are. Recalling that this is also affected by the spectral radius, these two parameters must be considered together. These two parameters regulate the amount of nonlinearity in $\boldsymbol{x}_k$ and the relative effect of the current input $\boldsymbol{u}_k$ on $\boldsymbol{x}_k$ opposed to history. In order to have few hyperparameters in the ESN, $\boldsymbol{W}_i$ should be scaled uniformly. However, it is common to include the bias weight $\boldsymbol{W}_b$ as the first column in $\boldsymbol{W}_i$, and if the goal is to increase the performance these should be scaled separately. It is also possible to scale the columns of $\boldsymbol{W}_i$ separately if the input sequence contributes differently to the task. Note that each scaling introduces another hyperparameter that must be tuned.

### 2.3.8 Training of ESNs

In this final paragraph, the training method of ESNs will be presented. Given a training set $\boldsymbol{u}_k$ and a target set $\boldsymbol{y}_{target,k}$ with $k = 1, ..., T$, and by including the bias weight in $\boldsymbol{W}_i$, (9) can, by using matrix notation, be rewritten as:

$$\boldsymbol{Y} = \boldsymbol{W}_o \boldsymbol{X}, \tag{10}$$

where $\boldsymbol{Y} \in \mathbb{R}^{N_y \times T}$ and $\boldsymbol{X} = [1 \quad \boldsymbol{u}_k \quad \boldsymbol{x}_k]^T \in \mathbb{R}^{(1+N_u \times N_x) \times T}$ are all outputs and states produced by the reservoir when presented all inputs $\boldsymbol{U} \in \mathbb{R}^{N_u \times T}$ (Lukoševičius, 2012).

Finding the optimal $\boldsymbol{W}_o$ is equivalent to minimizing the MSE between the ESN output $\hat{\boldsymbol{y}}_k$ and the given target $\boldsymbol{y}_{target,k}$. Since $T >> 1 + N_u + N_x$ in most cases, this is an overdetermined linear regression problem (Lukoševičius, 2012). One of the most common remedies when dealing with an overdetermined system is to use the ridge regression (also known as Tikhonov regularization):

$$\boldsymbol{W}_o = \boldsymbol{Y}_{target}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T + \beta\boldsymbol{I})^{-1} \tag{11}$$

where $\boldsymbol{Y}_{target} \in \mathbb{R}^{N_y \times T}$ is a matrix with every given $\boldsymbol{y}_{target,k}$ from $k = 1, ..., T$, $\beta$ is the regularization coefficient and $\boldsymbol{I}$ is the identity matrix. This will avoid both overfitting and feedback instability (Lukoševičius, 2012) and solves the following optimization problem:

$$\boldsymbol{W}_o = \arg\min_{\boldsymbol{W}_o} = \frac{1}{N_y}\sum_{i=1}^{N_y}\left(\sum_{k=1}^{T}(\hat{y}_{i,k} - y_{target,i,k})^2 + \beta||\boldsymbol{w}_i^o||_2^2\right) \tag{12}$$

compared to the ordinary MSE, this object function also includes $\beta||\boldsymbol{w}_i^o||_2^2$ which is a regularization term that penalizes large $\boldsymbol{W}_o$ making a compromise between small training error and small output weights. $\beta$ determines the relative importance and is a hyperparameter that must be tuned (Lukoševičius, 2012). There are multiple ways of determining $\beta$ with grid search being one of them (Grimstad, 2022). Note that (11) was transposed in the implementation in order to get it on the form $\boldsymbol{A}x = \boldsymbol{B}$ to exploit the *pytorch* linalg solver. This yielded:

$$\boldsymbol{W}_o^T = (\boldsymbol{X}\boldsymbol{X}^T + \beta\boldsymbol{I})^{-1}\boldsymbol{X}\boldsymbol{Y}_{target}^T \tag{13}$$

From this, it is clear that training the ESN is different from training traditional ANNs as described in chapter 2.2.4. Instead of running through the training data multiple times minimizing a cost function, all data points are presented to the network once. This is often referred to as *one-shot* training (Lukoševičius, 2012), and is, in most cases, faster than traditional training of NNs. When using ridge regression there are no limits to the amount of training data, and the time of the training procedure will be independent of the number of training points (Lukoševičius, 2012).

## 2.4 Long Short-Term Memory Networks

### 2.4.1 Introduction

The long short-term memory (LSTM) network is another RNN architecture more similar to the feedforward network than the ESN. It was introduced in the late '90s in (Hochreiter and Schmidhuber, 1997) as a remedy to the vanishing and exploding gradient problem associated with back-propagation through time (BPTT) in RNNs (Smagulova and James, 2020). These problems were a result of RNNs using the same weight matrix $\boldsymbol{W}$ at each timestep which most often would lead to the gradient vanishing because of finite-precision numbers in computers (Sussillo and Abbott, 2015). Traditional RNNs are consequently only capable of utilizing short-term dependencies. The LSTM architecture was more or less designed to learn long-term dependencies (Hoyer et al., 2022). Hence the name, since it allows the RNN to process sequences of arbitrary length. Today, it is still one of the most popular architectures for modeling sequential data such as speech recognition and machine translations (Goodfellow et al., 2016).

### 2.4.2 The LSTM Cell

In order to cope with the vanishing (and exploding) gradient problem, the LSTM creates paths where the gradients can flow unchanged through time when back-propagating (Goodfellow et al., 2016). This is made possible by extending the original RNN units with both memory and gates (Smagulova and James, 2020). These extended RNN units are usually referred to as *LSTM cells*, and they have the same in- and outputs as general RNNs. The gates within the cell enable control of the weight on the self-loops in the RNN making these weights context dependent rather than fixed. Furthermore, this provides the network with internal recurrence in addition to the general recurrence of the RNN described in chapter 2.2.6. Using gates is also why LSTMs are considered as a type of *gated* RNNs which are mentioned as the most effective sequence models used in practical applications in (Goodfellow et al., 2016). Note that there has been proposed many different cell types, but only the traditional cell will be covered here. A schematic illustration of this cell is shown in figure 8.
From inspection of this figure, it is evident that the cell has three gates; the forget, input and output gate. The forget gate $\boldsymbol{f}_t$ is an extension to the originally proposed architecture which prevents the state value from growing to infinity (Smagulova and James, 2020). This gate's output is given by:

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{h}_{t-1} \boldsymbol{b}_f) \tag{14}$$

where $\sigma$ denotes the sigmoid function which ensures the value of $f_t$ between 0 and 1. This corresponds to the amount of the previous state(s) that should be "remembered". Furthermore, $\boldsymbol{x}$ and $\boldsymbol{h}$ are the current input and the previous cell output, and $\boldsymbol{b}_f$, $\boldsymbol{U}_f$ and $\boldsymbol{W}_f$ denote the bias vector, hidden unit weights and the gate weights, respectively (Smagulova and James, 2020).

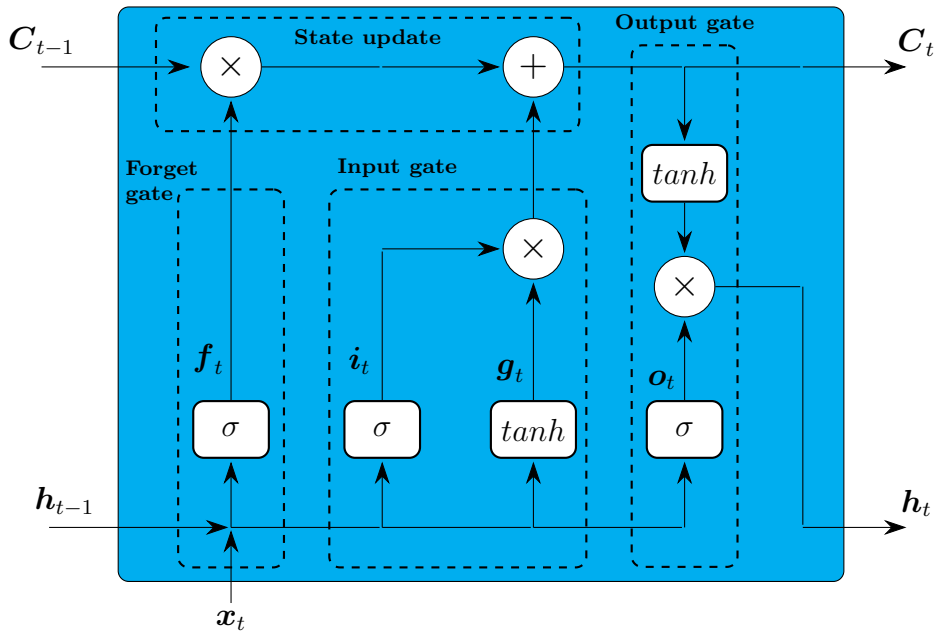



**Figure 8:** Schematic illustration of the traditional LSTM cell inspired by a figure from (Smagulova and James, 2020).

The remaining gates are given by:

$$\begin{aligned} \boldsymbol{i}_t &= \sigma(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{h}_{t-1} \boldsymbol{b}_i) \\ \boldsymbol{g}_t &= tanh(\boldsymbol{W}_g \boldsymbol{x}_t + \boldsymbol{U}_g \boldsymbol{h}_{t-1} \boldsymbol{b}_g) \\ \boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{h}_{t-1} \boldsymbol{b}_o) \end{aligned} \tag{15}$$

Finally, the new memory state and cell output; $\boldsymbol{C}_t$ and $\boldsymbol{h}_t$ are given by:

$$\begin{aligned} \boldsymbol{C}_t &= \boldsymbol{f}_t \cdot \boldsymbol{C}_{t-1} + \boldsymbol{g}_t \cdot \boldsymbol{i}_t \\ \boldsymbol{h}_t &= \boldsymbol{o}_t \cdot tanh(\boldsymbol{C}_t) \end{aligned} \tag{16}$$

### 2.4.3 Training of LSTMs

LSTMs are, fundamentally, regular RNNs and can be implemented using multiple layers. Training of LSTMs are therefore usually done the same way as for RNNs with gradient descent using BPTT to compute the cost gradient (Smagulova and James, 2020). The MSE is a much used cost function in training of LSTMs since this network is based on regression analysis (Goodfellow et al., 2016). The MSE is given by:

$$\frac{1}{N} \sum_{i=1}^{N} (f^*(\boldsymbol{x}) - f(\boldsymbol{x}, \boldsymbol{\theta}))^2 \tag{17}$$

# 3 Method

This section covers the details of the methods used in this project. The first two subsections will cover the mathematical model and the implementation of the ESP simulator. This is followed by two subsections covering the details of the system identification and data generation, respectively. At last, will the implementation of the ESN and the LSTM be detailed in the final two subsections.

## 3.1 Mathematical Model of an ESP Lifted Well

The ESP is, as described in chapter 2.1.4, a complex system with multiple mechanical components. In addition, the well must also be considered in order to derive a mathematical model for optimizing operation and ultimately oil production. One such model is presented in (Pavlov et al., 2014) where it was used in an MPC and tested extensively at Equinor's R&D center in Porsgrunn, Norway. This model was originally developed in 2010 by the same authors in cooperation with Equinor (previously Statoil). In the same year, (Binder et al., 2015) enhanced this model to also include estimates of flow rates and viscosity in a similar well. This model, including the reported parameters and assumptions, is the model that will be used to implement the simulator in this project. An illustration of the model is presented in figure 9. From inspection, it is evident that there are many variables and parameters that must be considered. All the reported numerical values are given in table 1.

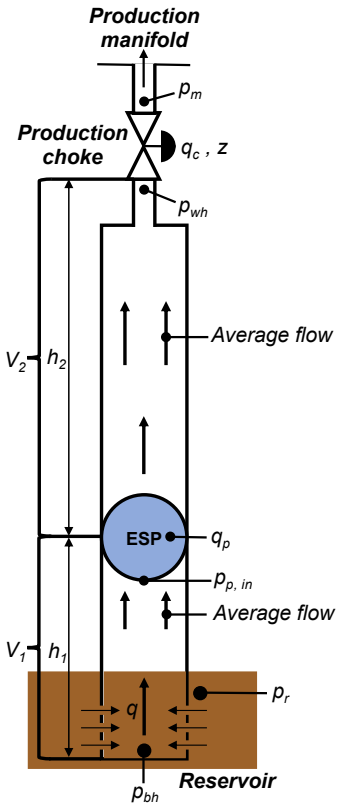



**Figure 9:** Mathematical ESP model inspired by a figure from (Binder et al., 2015).

The resulting model in (Binder et al., 2015) is a third-order nonlinear model given by the following differential equations:

$$\dot{p}_{bh} = \frac{V_1}{\beta_1}(q_r - \hat{q}) \tag{18}$$

$$\dot{p}_{wh} = \frac{V_2}{\beta_2}(\hat{q} - q_c) \tag{19}$$

$$\dot{\hat{q}} = \frac{1}{M}(p_{bh} - p_{wh} - \rho g h_w - \Delta p_f + \Delta p_p) \tag{20}$$

where $q_c$, $q_r$, $\Delta p_f$ and $\Delta p_p$ represent the choke model, the well inflow model, the ESP model and friction, respectively. More specifically, the choke model represents the flow $q_c$ through the production choke valve and is given by

$$q_c = C_c\sqrt{p_{wh} - p_m} \cdot z \tag{21}$$

where $C_c$, $p_{wh}$, $p_m$ and $z$ are the choke valve constant, the wellhead pressure, the production manifold pressure and the production choke opening, respectively. Similarly, the inflow model is given by

$$q_r = PI(p_r - p_{bh}) \tag{22}$$

with $p_r$ being the reservoir pressure (natural pressure in the well) and $PI$ being the well productivity index. The ESP model is given by

$$\Delta p_p = \rho g H \tag{23a}$$

$$H = C_H(\mu)H_0(q_0)\left(\frac{f}{f_0}\right)^2 \tag{23b}$$

$$q_0 = \frac{q}{C_q(\mu)}\left(\frac{f_0}{f}\right) \tag{23c}$$

This model is modified from the original model in (Pavlov et al., 2014) to also include viscosity correcting factors (VCFs) for the flow rate ($C_q(\mu)$), the head ($C_H(\mu)$) and break horsepower (BHP) ($C_P(\mu)$). The latter is an imperial measurement for the electric power consumed by the ESP motor. These values, in addition to the ESP head and BHP characteristics ($H_0(q)$ and ($P_0(q)$), are assumed to be known for the particular ESP in the model. Note that VCFs are usually obtained from published sources or empirically obtained, whereas the characteristics for the ESP are provided by the pump vendor (Binder et al., 2015). The VFCs and ESP characteristics are in (Binder et al., 2015) given by polynomials on the following form

$$P(x) = \sum_{i=0}^{4} c_i x^i \tag{24}$$

where the coefficients ($c_i, i = 1, ..., 4$) are given in table 2. These are the same as in the publication.

Finally, $\Delta p_p$ models, as mentioned, friction. This entity depends on the fluid density and the physical properties of the mechanical components. It also depends on the fluid viscosity and the average flow which both are varying properties. The full expression is given by

$$\Delta p_f = 0.158 \cdot \sum_{i=1}^{2} \frac{\rho L_i q^2}{D_i A_i^2} \left( \frac{\mu}{\rho D_i q} \right)^{\frac{1}{4}} \tag{25}$$

| Variable | Description | Value | unit |
|---|---|---|---|
| **Known constants** | | | |
| $g$ | Gravitational acceleration constant | 9.81 | $m/s^2$ |
| $C_c$ | Choke valve constant | $2e^{-5}$ | $*$ |
| $A_1$ | 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 frequency | 60 | $Hz$ |
| $I_{np}$ | ESP motor nameplate current | 65 | $A$ |
| $P_{np}$ | ESP motor nameplate power | $1.625e^5$ | $W$ |
| **Parameters from fluid analysis and well tests** | | | |
| $\beta_1$ | Bulk modulus below ESP | $1.5e^9$ | $Pa$ |
| $\beta_2$ | Bulk modulus above ESP | $1.5e^9$ | $Pa$ |
| $M$ | Fluid inertia parameter | $1.992e^8$ | $kg/m^4$ |
| $\rho$ | Density of produced fluid | 950 | $kg/m^3$ |
| $p_r$ | Reservoir pressure | $1.26e^7$ | $Pa$ |
| **Unknown parameters** | | | |
| $PI$ | Well productivity index | $2.32e^{-9}$ | $m^3/s/Pa$ |
| $\mu$ | Viscosity of produced fluid | Varying | $Pa \cdot s$ |
| $H_0$ | ESP head characteristics | Varying | $m$ |
| $P_0$ | ESP BHP characteristics | Varying | $W$ |
| $q_0$ | Theoretical flow rate at reference frequency | Varying | $m^3/s$ |
| $C_H$ | VCF for head | Varying | – |
| $C_P$ | VCF for brake horsepower of the ESP | Varying | – |
| $C_q$ | VCF for ESP flow rate | Varying | – |

**Table 1:** Model parameters used in the simulator. These values are obtained from (Binder et al., 2015).

| | $c_4$ | $c_3$ | $c_2$ | $c_1$ | $c_0$ |
|---|---|---|---|---|---|
| $H_0$ | 0 | 0 | $-1.2454e^6$ | $7.4959e^3$ | $9.5970e^2$ |
| $P_0$ | 0 | $-2.3599e^9$ | $-1.8082e^7$ | $4.3346e^6$ | $9.4355e^4$ |
| $C_q$ | 2.7944 | -6.8104 | 6.0032 | -2.6266 | 1 |
| $C_H$ | 0 | 0 | 0 | -0.03 | 1 |
| $C_P$ | -4.4376 | 11.091 | -9.9306 | 3.9042 | 1 |

**Table 2:** Coefficients used for VCFs and ESP characteristics. These values are obtained from (Binder et al., 2015).

## 3.2 Implementation of the ESP Lifted Well Simulator

The simulator implementation is inspired by (Osnes, 2020) and implemented using Python 3.10, *NumPy* and *CasADI*. *NumPy* is an open-source framework for numerical computation in Python. This was mainly used for handling tensors inside the simulator. *CasADI* is also an open-source tool, but for nonlinear optimization and algorithmic differentiation (Andersson et al., 2019). In the implementation, *CasADI* was utilized for solving the differential equations.

To simulate the ESP lifted well, the simulator takes the current state $x$ and an input $u$ and solves the initial value problem one time step ahead. The simulator thus requires an initial state $x_0$ which in this work was given by

$$x_0 = \begin{bmatrix} p_{bh,0} \\ p_{wh,0} \\ q_0 \end{bmatrix} = \begin{bmatrix} 75e^5 \\ 30e^5 \\ 0.01 \end{bmatrix} \tag{26}$$

where the numerical values were inspired by (Osnes, 2020), but slightly changed after conducting some experiments. Note that both the initial pressures and the initial flow are given in *pascal* and $m^3/s$, respectively, whereas the later reported results are given in *bar* and $m^3/h$. The input $u$ is a vector with the choke opening $z$ in percent and the ESP motor frequency $f$ given in $Hz$:

$$u = \begin{bmatrix} z \\ f \end{bmatrix} \tag{27}$$

During the initial testing of the simulator, there were some issues with the *CasADI* integrator not converging due to encountering negative flow values. This is equivalent to the flow suddenly changing direction and is not consistent with the expected behavior of the ESP because it would require the motor or the head pressure to change direction. Further investigation revealed that when solving (20), the flow value $q$ became infinitesimally small leading to numerical instability. Hence, a minimum $q$ value of $1e^{-9}$ was enforced in the simulator removing the instability and thus the convergence issues.

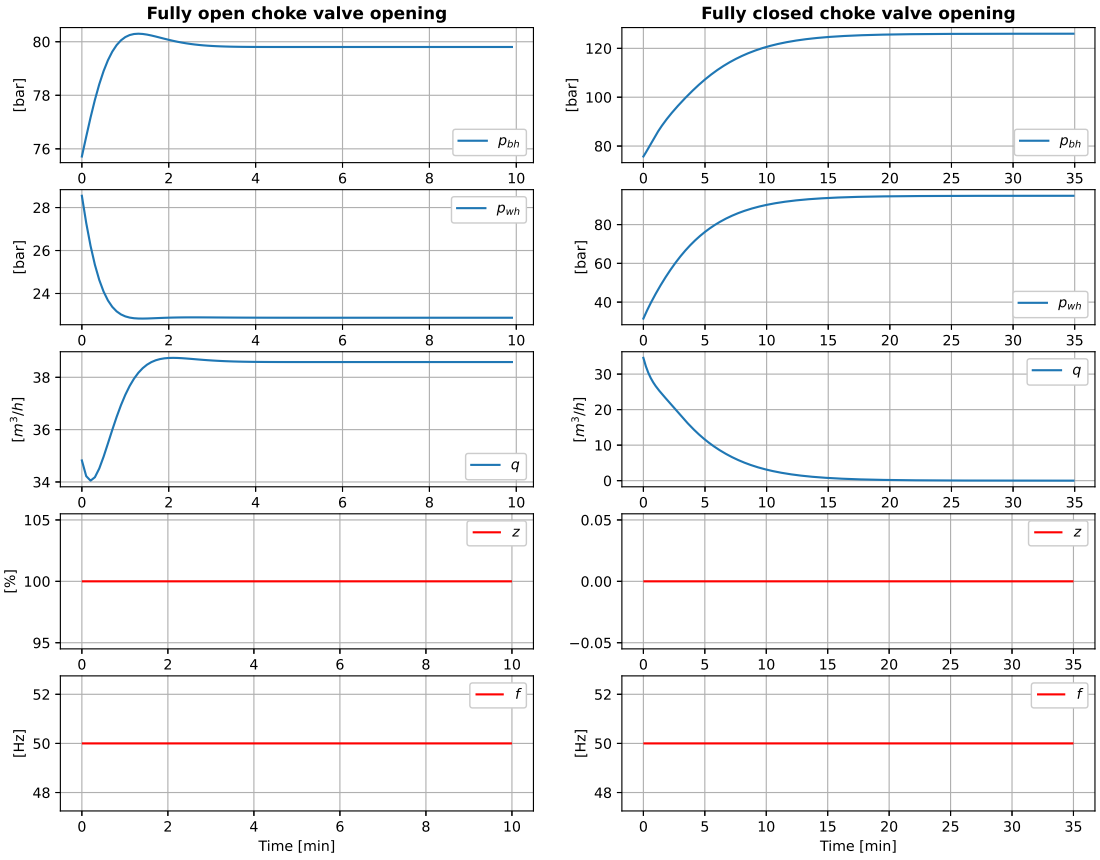
## 3.3 Confirmation of the Simulator Implementation

In the following subchapters, three experiments will be conducted to confirm if the simulator behaves as expected. The initial value $x_0$ is the same as in chapter 3.2.

Explanations of the conduction, input values $u$ and results from each experiment will be presented in the respective subchapter.

### 3.3.1 Fully Open and Fully Closed Choke Valves

In these experiments, the simulator was first run with a fully open choke valve ($z = 100\%$). Then, the same experiment was conducted again with the choke valve being fully closed ($z = 0\%$). The motor had a constant frequency $f$ of $50Hz$ in both cases (this is within normal operation rates according to (Pavlov et al., 2014)). In the first experiment, it was expected that the simulation would reach a steady state since it more or less simulates normal production. Similar results were expected in the second experiment as well, but since the choke is closed, the steady state should correspond to the maximum pressure that the motor is capable of building. Despite that the ESP should never be run this way, it is still interesting to see how the model behaves in such an extreme case. Results from the simulations are shown in figure 10(a) and 10(b).




(a) Fully open choke valve.

(b) Fully closed choke valve.

**Figure 10:** Results from running simulator with fully open and closed choke valve and constant motor frequency.

From inspection of figure 10(a) it is evident that the bottomhole pressure $p_{bh}$ and the average flow $q$ increases until both reach a steady state after about three minutes. Furthermore, it can be seen that the wellhead pressure $p_{wh}$ decreases until it also reaches

a steady state around the same time. This is consistent with the expectations since the inputs are different from those producing the initial state, meaning that the system should reach a new operating point where the states and the inputs are in equilibrium. Looking at each state individually, it is expected that the wellhead pressure $p_{wh}$ is relieved when the choke valve is suddenly fully opened. Hence, why it decreases in the simulation. The new steady state is reached when the valve reaches its maximum throughput. The change will further propagate in the system leading to an increase in the average flow $q$ which directly increases the wellhead pressure $p_{wh}$. This explains why both states reach a steady state at the same time. How the bottomhole pressure $p_{bh}$ is affected by the choke valve can not be interpreted from this experiment alone. Yet, one can expect that it should decrease when the valve is open and the motor runs on a constant frequency since this allows a greater flow through the well. Anyway, the result from this experiment is satisfactory because it demonstrate that the simulator accounts for both mechanical and physical capacities when given a new operating point.

Figure 10(b) shows as expected similar results as the first experiment but in the reverse case. It is evident that both the bottomhole pressure $p_{bh}$ and the wellhead pressure $p_{wh}$ increase until they reach a steady state after about 20 minutes. During the same period, the average flow $q$ decreases until reaching a steady state at $0m^3/h$ meaning that it stops. At this point, the ESP is filled up with fluid since no produced fluid is released through the production choke valve. This is further motivated by observing that the buildup of pressure slows down taking almost 20 minutes before reaching an equilibrium. The main actuator for building the pressure is the motor which has finite power. Looking at the exponential decrease in the average flow $q$, it is apparent that the buildup of pressure is inversely proportional to the total pressure. This is an interesting result because it means that the simulator also accounts for the limitations of the motor. However, operating the ESP this way will eventually lead to motor failure due to overheating. It can also lead to other mechanical failures or in worst case an explosion if the pressure becomes too large for the components to withstand. Neither of these consequences are accounted for in the simulator nor are they expected since it is assumed that the ESP is not operated this way.

### 3.3.2 Step Response of the Choke Valve

Even though this experiment is similar to the first experiment in chapter 3.3.1, it is expected to shed light upon how the bottomhole pressure $p_{bh}$ is affected by changes in the choke valve. In this experiment, the choke valve $z$ will be held at 60% for half of the simulation before it is opened to 100% with the motor frequency $f$ still held constant at $50Hz$. The resulting simulation is shown in figure 11.

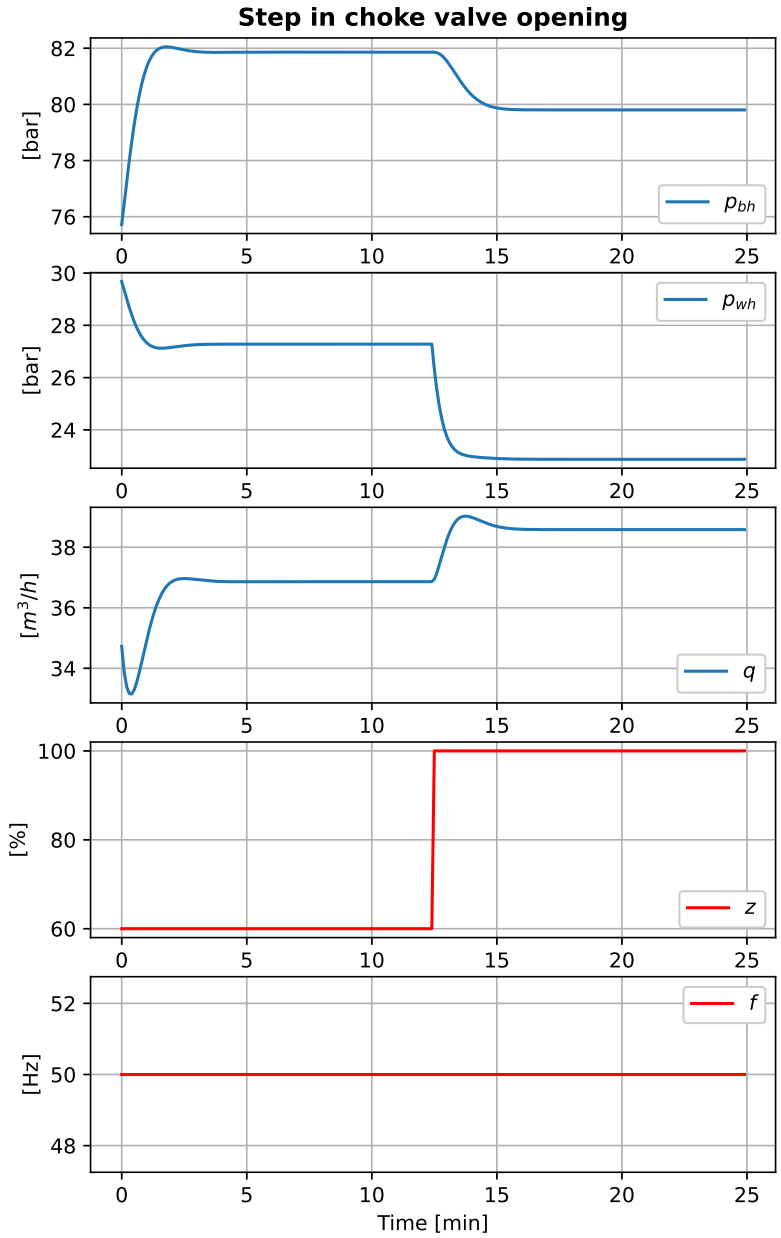



**Figure 11:** Results from running simulator with a step in the choke valve opening and constant motor frequency.

Inspection of figure 11 shows that all states reach and maintain a steady state before the choke valve is further opened. After the choke valve is opened, both the bottomhole pressure $p_{bh}$ and the wellhead pressure $p_{wh}$ decrease and maintain new steady states. The first observation confirms the hypothesis from chapter 3.3.1 of a decrease in the bottomhole pressure $p_{bh}$ when the choke valve is opened, and the latter is consistent with previous observations. Furthermore, it can be seen that the average flow increases and decreases slightly before reaching a new steady state. This indicates that the model also accounts for a sudden increase in the flow which will be a little higher at the beginning to replace the fluid that was suddenly released. Results from a similar experiment with a sudden decrease in the valve opening are shown in appendix A.1. This experiment showed the same results but reversed.

### 3.3.3 Gradually Increased Motor Frequency

The effect of changes in the motor frequency $f$ is investigated in this final experiment. It was conducted holding the choke valve $z$ constant at 50% (this is within normal operation rates according to (Pavlov et al., 2014)). The motor frequency $f$ was then increased with $5Hz$ every 10th minute for 60 minutes starting at $45Hz$. The resulting simulation is shown in figure 12.

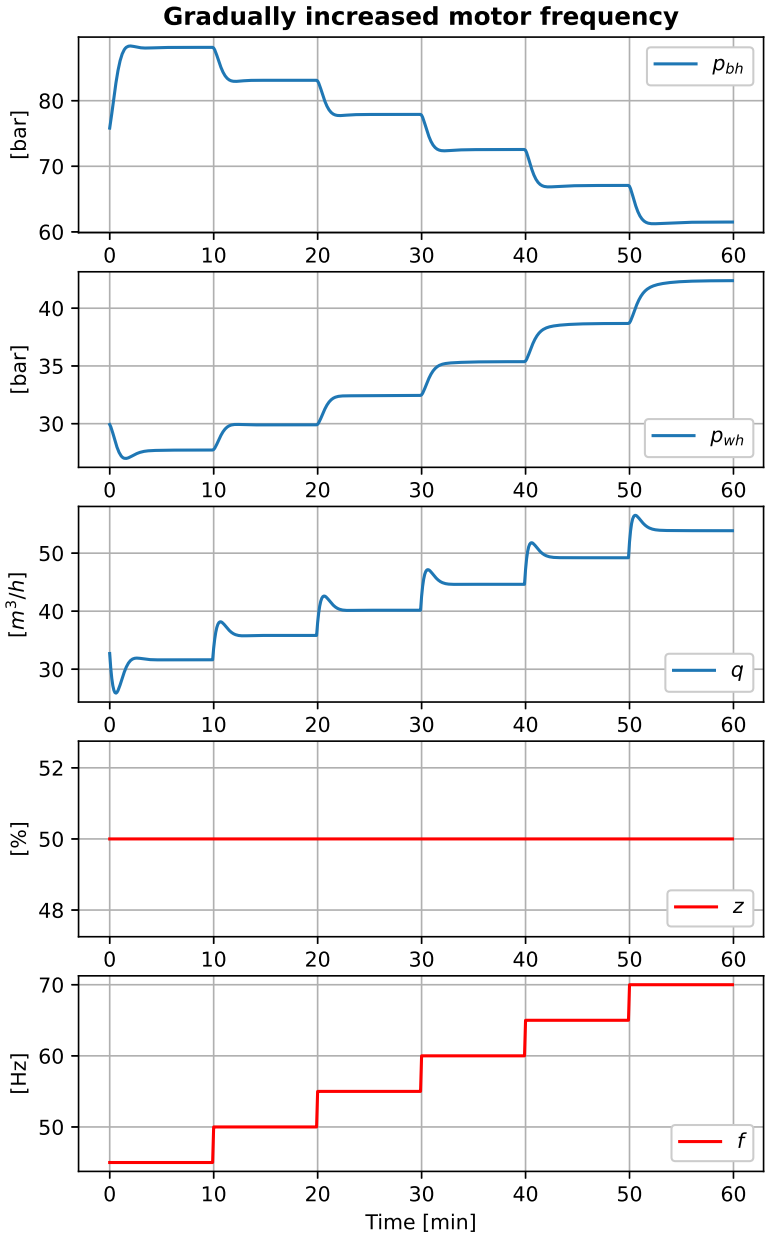



**Figure 12:** Results from running the simulator with constant choke valve opening and gradually increasing motor frequency.

Inspection of figure 12 shows that the bottomhole pressure $p_{bh}$ decreases and maintains a new steady state for every increase in the motor frequency. This is expected since a greater amount of the fluid in the well is pumped through the ESP. Consequently,

this should also increase the average flow $q$ at each step which the simulation confirms. Furthermore, it can also be seen that the wellhead pressure $p_{wh}$ increases at each step. This is a result of the increasing average flow and the constant choke valve opening which leads to pressure building up at the valve inlet. From the experiment, it is therefore clear that the simulator gives satisfactory and expected results. An experiment where the motor frequency was gradually decreased was also conducted, and it can be seen in appendix A.2. This experiment gave the same results but reversed.

## 3.4 System Excitation Using APRBS

When gathering information about a system, it is important to excite it within its working conditions. This is a crucial part of identifying a system model (Nelles, 2013). Linear systems (including nonlinear systems linearized around some working point) are often identified by exciting the system symmetrically around some working area using white noise. This way, the identification experiment provides a linear model that is not polluted by nonlinear effects close to saturation points or in unreachable areas of the model. The pseudo random bit stream (PRBS), shown in figure 13, is a popular choice of excitation signal because it behaves as discrete white noise (Nelles, 2013).

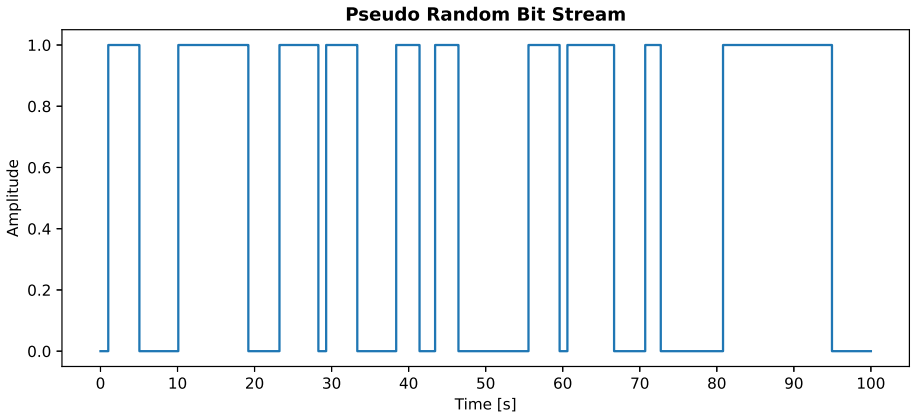



**Figure 13:** Example of a PRBS excitation signal.

In the case of nonlinear systems, white noise will only visit a small region of the system's working range (Osnes, 2020). This can be seen in figure 13 as the signal only steps between 0 and 1 with random interval lengths. However, by giving the PRBS a random amplitude (which results in an amplitude-modulated PRBS (APRRS)) more system regions can be visited. This signal is thus well suited for nonlinear system identification (Nelles, 2013). APRBS will, therefore, be used as an excitation signal in this specialization project. An example of an APRBS excitation signal is shown in figure 14. Note that the holding time for the signal is constant in this figure. This enables control over the speed of the system dynamics during identification and testing.

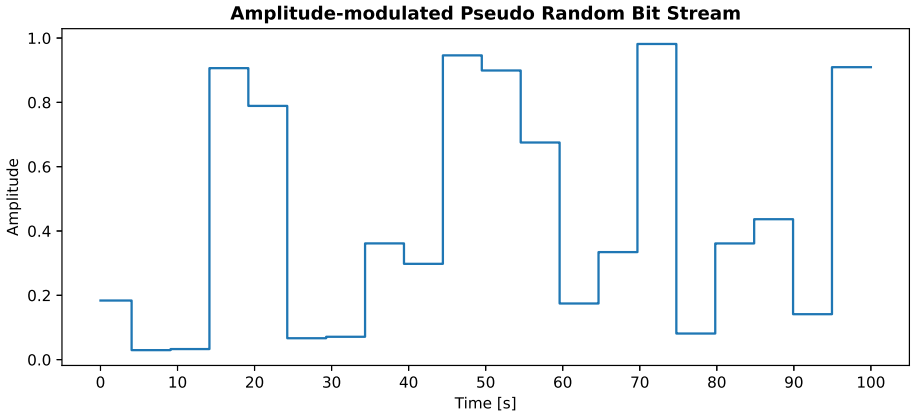



**Figure 14:** Example of an APRBS excitation signal.

## 3.5 Generating the Training, Validation and Test Set

The training set was generated by using a 2 000 minutes long APRBS as input to the ESP simulator. Using a sampling time of 12 samples per minute produced a total of 24 000 data points. This sampling time was reported sufficient to capture all of the ESP dynamics in (Osnes, 2020). Furthermore, to capture both faster and slower dynamics, the first 1 000 minutes use a holding time of 0.8 seconds (10 samples), whereas the final 1 000 minutes use a holding time of approximately 4.2 seconds (50 samples). Training with different dynamics is necessary for the network to learn as much of the system behavior as possible (Osnes, 2020). The amplitude bounds for this APRBS used in this project are given in table 3. These were chosen based on the normal operation rates reported in (Pavlov et al., 2014). The resulting training set is shown in figure 15.

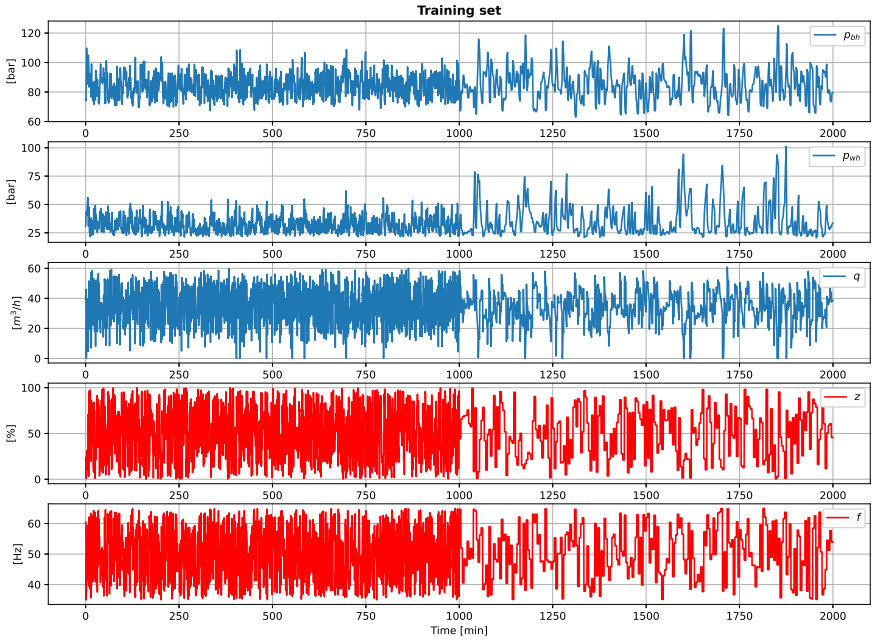



**Figure 15:** Training set used for training the ESN and the LSTM.

| Input | Max | Min |
|---|---|---|
| $f$ | 65 | 35 |
| $z$ | 100 | 0 |

**Table 3:** Amplitude bounds for the APRBS.

An unique validation set was generated to tune the hyperparameters in both networks. Using a validation set different from the test set for tuning is necessary to prevent data leakage (Grimstad, 2022). Since the ESP simulator can provide an unlimited amount of data points, the validation set was generated the same way as the training set. Being able to generate the validation set avoids the use methods like k-folding (Grimstad, 2022). The validation set contained 1 200 samples which proved adequate to validate the network performance. Using the same time constant as for the training set, this corresponded to 100 minutes of simulation. The resulting validation set is shown in figure 16, and it can be seen that the input stays constant for 200 samples (approx. 16.5 minutes) in the beginning. This is called a *warm-up stage* and ensures that the network and the simulator have the same initial value. Note that these 200 first samples are discarded when evaluating the performance, and thus only 1 000 samples (approx. 83.5 minutes) are used for validation. From these 1 000 samples, the first 500 contain faster dynamics with a hold time of 5 samples (approx. 0.4 seconds) and the last 500 contain slower dynamics with a hold time of 20 samples (approx. 1.7 seconds). Although this is somewhat faster than in the training set, it ensures that the smaller validation set contains enough changes in the input to evaluate the network performance.

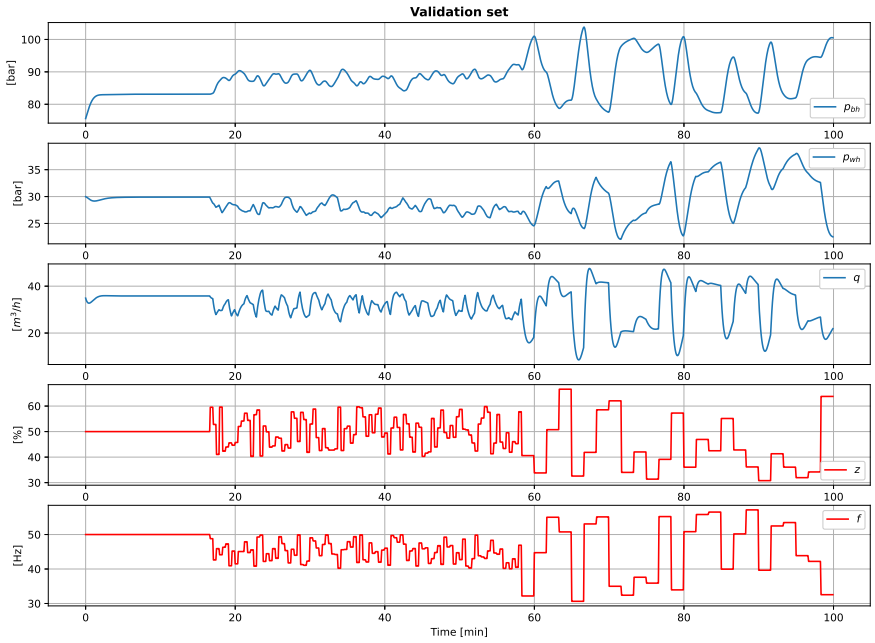



**Figure 16:** The generated validation set.

The test data set was generated the same way as the validation set. Note that every generated APRBS is random. This ensured that the validation and test sets are different.

## 3.6 Error Metrics

The overall objective of the implemented networks is to learn the dynamical model of the ESP lifted well, meaning that the performance can be calculated in terms of how close the network output is to the corresponding simulated data point in a validation or test set. This type of error is typically calculated using an MSE function (Lukoševičius, 2012) such as the root mean square error (RMSE) given by:

$$\text{RMSE}(\boldsymbol{y}, \boldsymbol{y}_{target}) = \frac{1}{N_y} \sum_{i=1}^{N_y} \sqrt{\frac{1}{T} \sum_{k=1}^{T} (\hat{y}_{i,k} - y_{target,i,k})^2} \tag{28}$$

where the state errors are averaged over the output dimensions to give an overall measure of the model error. Furthermore, the RMSE can also be normalized by dividing it by the variance of the target data set. This error function is called normalized RMSE (NRMSE) and is independent of any particular scaling (Lukoševičius, 2012). However, error functions based on MSE can to some extent be hard to interpret for humans. Another error function is therefore the mean absolute percentage error (MAPE) (Grimstad, 2022) given by:

$$\text{MAPE}(\boldsymbol{y}, \boldsymbol{y}_{target}) = \frac{1}{T} \sum_{k=1}^{T} \left| \frac{\hat{y}_k - y_{target,k}}{y_{target,k}} \right| \cdot 100\% \tag{29}$$

This error function indicates on average how much each model state deviates from the target in percent. It is also independent of data-scaling (Chen et al., 2017). Moreover, if averaged over the dimension of the output such as (28), it indicates on average how much the entire model deviates from the target. This is particularly easy to interpret for humans. For the purpose of gaining experience with different error metrics, the NRMSE will be used to find hyperparameters for the networks, whereas the MSE and MAPE will be used to compare the resulting models. Note that either of these could have been used for both tasks.

## 3.7 Implementation of the Echo State Network

The ESN was implemented in Python 3.10 using mainly *PyTorch* which is an open-source machine learning framework. Since neither this framework nor any other well-known commercial frameworks offer any modules for reservoir computing, a framework for ESN had to be implemented from scratch. *PyTorch* did, however, play a crucial role in handling tensors inside the network. Furthermore, *NumPy* and *SciPy* were also used to support *PyTorch* to simplify the implementation as it does not offer all the required functions. The final model was trained using the training set normalized between 0 and 1.

### 3.7.1 Searching for Hyperparameters

The hyperparameters were found by first finding some suboptimal hyperparameters from a trial and error approach using the reported values in (Osnes, 2020) as a starting point. Then, a grid search was conducted for each hyperparameter. Grid search involves searching for the optimal hyperparameter one at a time (Goodfellow et al., 2016). In this context, optimal refers to the value that results in the lowest NRMSE on the validation set. Multiple ESN networks were thus initialized and trained with different values for one of the hyperparameters at a time while the others were kept constant. Grid search is generally effective when having a low number of hyperparameters to tune (Goodfellow et al., 2016). Since the initialization of the reservoir is random, the same seed was used in all searches enabling comparison of the resulting NRMSEs. Note that this method does not guarantee the optimal *combination* of parameters since some particular combinations may yield better performance than those obtained. However, it allows one to find satisfactory values relatively fast based on the suboptimal parameters used as a starting point.

#### Regularization coefficient

The regularization coefficient $\beta$ was, in contrast to the other hyperparameters, found by initializing only a single reservoir. This is because it only affects the output weight $\boldsymbol{W}_o$. Figure 17 shows the resulting NRMSEs for different coefficients. Note that the grid search was conducted on a logarithmic scale and thus yielded $10^{-1}$ as the optimal value with an NRMSE of 1.471.

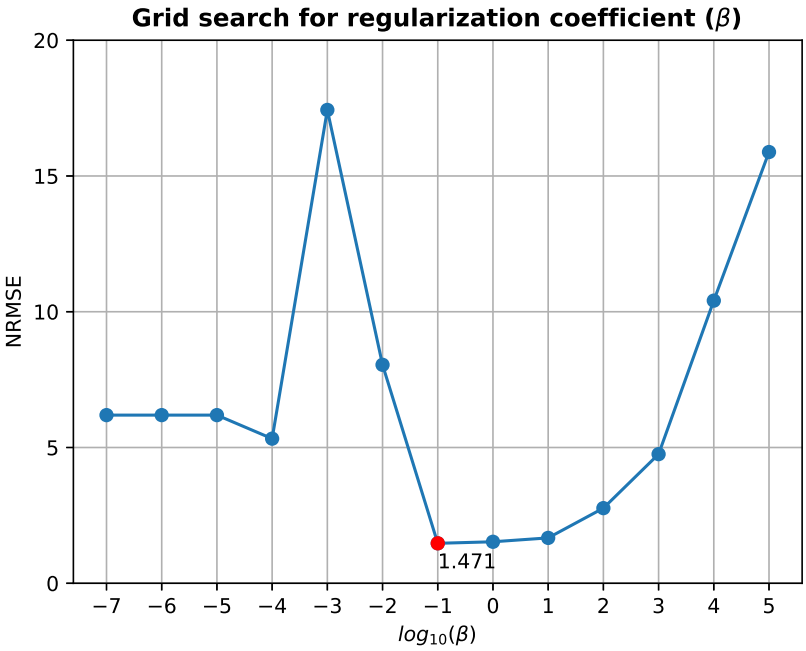



**Figure 17:** Grid search for the regularization coefficient.

### Reservoir size

The reservoir size $\boldsymbol{N_x}$ should, as described in chapter 2.3.3, be chosen as large as computationally possible. This is also confirmed by the grid search shown in figure 18 where the lowest NRMSE is obtained for $\boldsymbol{N_x} = 500$. However, it is hard to tell how much the reservoir size affects the learning ability of the network (Lukoševičius, 2012). Still, it is proven that the size of the reservoir directly affects the computational time of the network. For this reason, a smaller reservoir size of 250 was chosen. This should not affect the learning ability of the network significantly because the NRMSE barely decreases from 250 to 500.

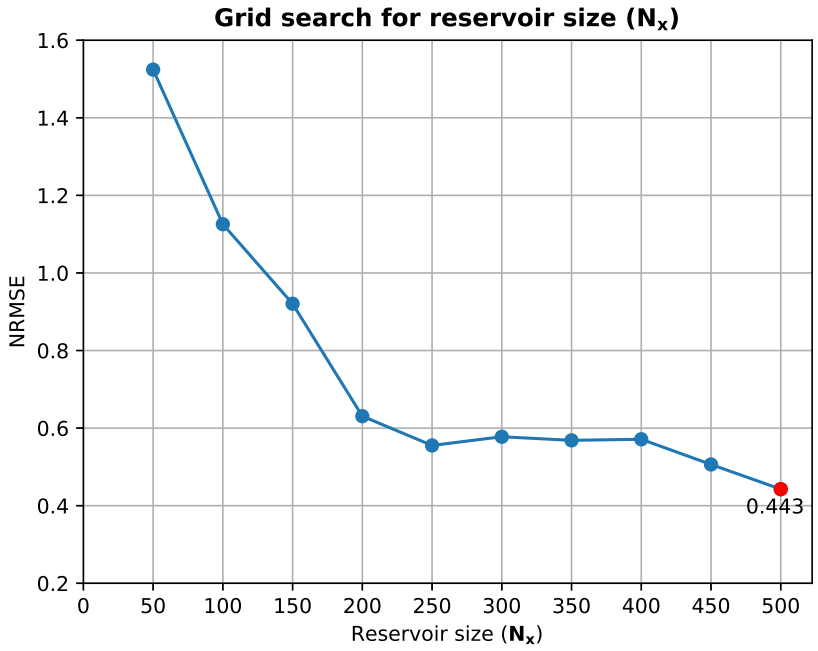
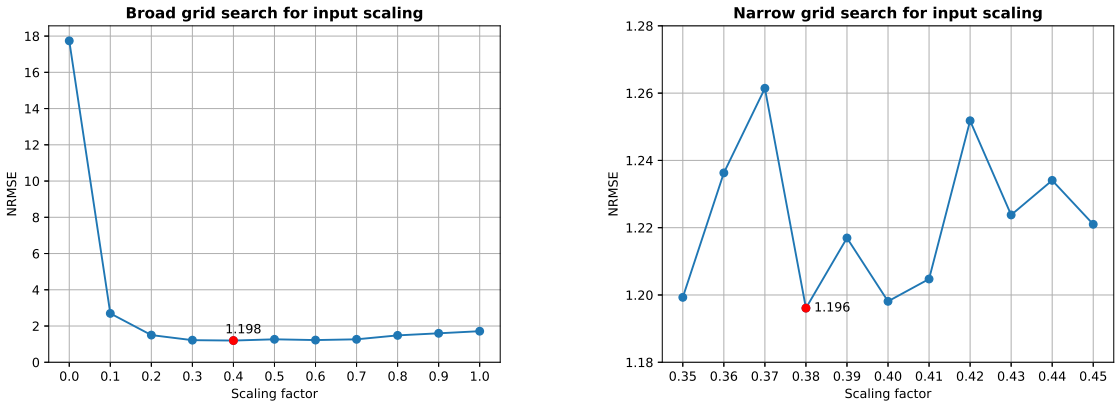



**Figure 18:** Grid search for the reservoir size.

### Input scaling

Since the input weights are initialized randomly, it is desirable that these weights do not affect the input data too much (Lukoševičius, 2012). All data is normalized meaning that a large input scaling will produce values within the saturation area of the $tanh(\cdot)$ activation function. Therefore, a broad grid search was conducted from 0 to 1 with a step length of 0.1 first. Then, a narrow grid search was conducted from the optimal value from the broad search $\pm 0.5$ with a step length of 0.01. This yielded an optimal input scaling of 0.38 with an NRMSE of 1.196. Both searches are shown in figure 19.



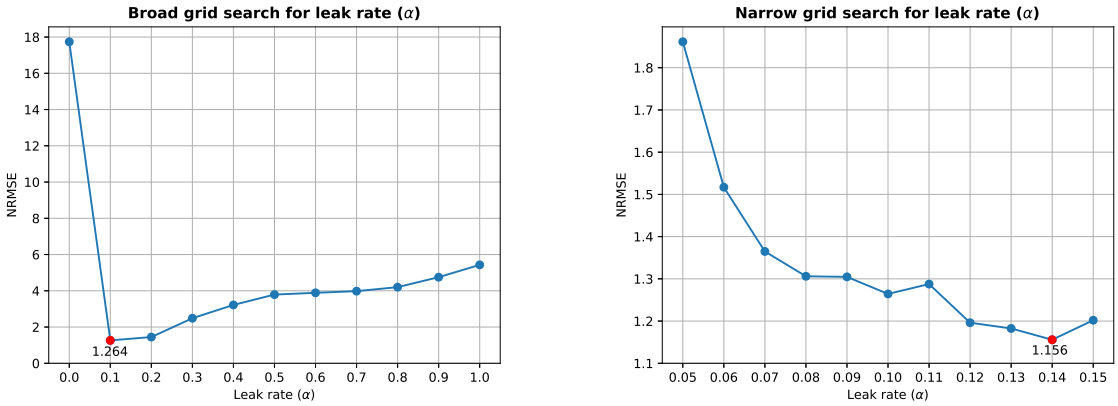

(a) Broad search.

(b) Narrow search.

**Figure 19:** Broad (a) and narrow (b) grid search for the input scaling.

### Leakage rate

The leakage rate $\alpha$ affects how much of the previous state that will influence the next state. Hence, it is given in percent and a broad and narrow grid search was conducted the same way as for the input scaling. This yielded an optimal leak rate of 0.14 with an NRMSE of 1.156. Both searches are shown in figure 20.




(a) Broad search.

(b) Narrow search.

**Figure 20:** Broad (a) and narrow (b) grid search for the leak rate.

### Spectral radius

The spectral radius must, as mentioned in chapter 2.3.5, be a value between 0 and 1 in order to ensure the echo state property. Hence, a broad grid search with a step length of 0.1 was conducted between these values. This is shown in figure 21. From inspection it is clear that the performance increases with increasing spectral radius. A spectral radius of 0.99 with an NRMSE of 0.707 was therefore chosen.

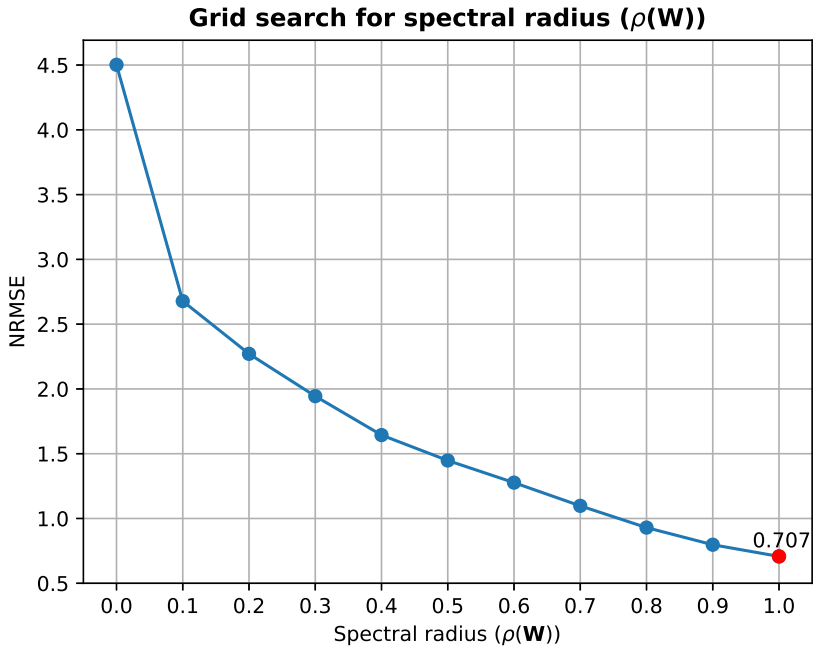



**Figure 21:** Grid search for spectral radius.

The resulting hyperparameters are given in table 4. Note that sparsity is not considered in this project and therefore set to 1. This is equivalent to having full connection between the nodes in the reservoir.

| **Parameter** | **Value** |
|---|---|
| Regularization coefficient ($\beta$) | $10^{-1}$ |
| Reservoir size ($N_x$) | 250 |
| Input scaling | 0.38 |
| Leakage rate ($\alpha$) | 0.14 |
| Spectral radius ($\rho(\mathbf{W})$) | 0.99 |
| (Sparsity) | 1 |

**Table 4:** Resulting hyperparameters used in the ESN implementation.

## 3.8 Implementation of the Long Short-Term Memory Network

The LSTM network was implemented in Python 3.10 using the LSTM class offered in the neural network module in *PyTorch*. The resulting network was implemented with a single (hidden) LSTM layer and had a linear output layer. This is illustrated in figure 22. Furthermore, the network was trained using stochastic gradient descent (SGD) with the MSE cost function described in chapter 2.4.3. After calculating the gradient, the parameters were updated using the adaptive moments (Adam) optimizer offered in the *PyTorch* optimize package. Adam was chosen based on its popularity and robustness when doing stochastic optimization (Kingma and Ba, 2014). The final model was trained using the training set normalized between $-1$ and 1.

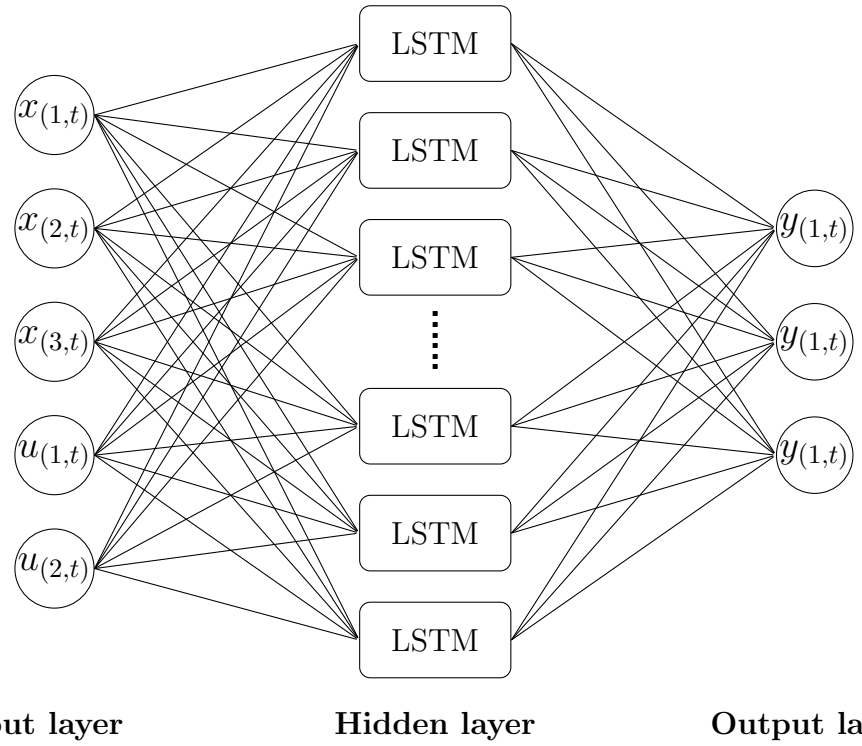



**Figure 22:** Schematic illustration of the implemented LSTM network.

### 3.8.1 Selecting Hyperparameters

Even though there exist many LSTM architectures, the most basic implementation was chosen in this project. This implementation uses the LSTM cell described in chapter 2.4.2 and requires only four hyperparameters; learning rate, hidden layer size, batch size and number of epochs. These were found mostly through trial and error due to the significantly longer training time required for the LSTM compared to the ESN.

#### Learning Rate and Size of the Hidden Layer

The learning rate and hidden layer size were chosen to be $10^{-3}$ and 200, respectively. This learning rate is the default value of the Adam implementation in *PyTorch*. The hidden layer size was chosen arbitrarily but proved to work well. It was, generally, spent little time tuning these parameters.

### Batching of Training Data

When training an LSTM network, the weights are typically updated after a batch of training data. It becomes thus relevant how the training data is batched as it directly affects the efficiency of the optimization. According to (Goodfellow et al., 2016), some hardware achieve better runtime if the batch size is a power of 2. Hence, the batch size was set to 64 because this is both a power of 2 and gives exactly 375 batches (24000 is divisible by 64). Furthermore, these were shuffled randomly before each epoch using the *DataLoader* offered in *PyTorch*. This was to make the model more robust and to increase the effectiveness of the optimization algorithm (Goodfellow et al., 2016).

### Number of Epochs

Training of NN involves typically looping over the training data multiple times. Each full loop is called an *epoch* and the number of epochs is a hyperparameter that decides the duration of the training. Yet, this hyperparameter was not decided directly. In a preliminary run, the network was trained using 100 epochs while its performance on the validation set after each epoch was monitored. This run took about 17 minutes and is shown in figure 23.

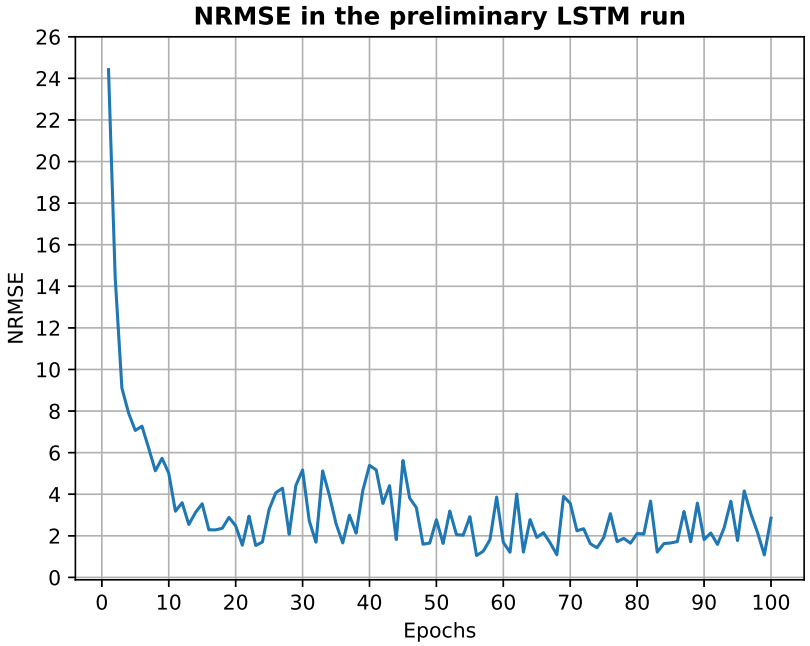



**Figure 23:** Performance of the LSTM on the validation set after each epoch in the preliminary LSTM run.

From inspection of this figure, it is evident that the NRMSE oscillates between 1 and 4 after about 50 epochs. This suggests that the model should be trained for at least that many epochs. Note that these oscillations are caused by the gradient being calculated from random samples from each batch (Goodfellow et al., 2016).

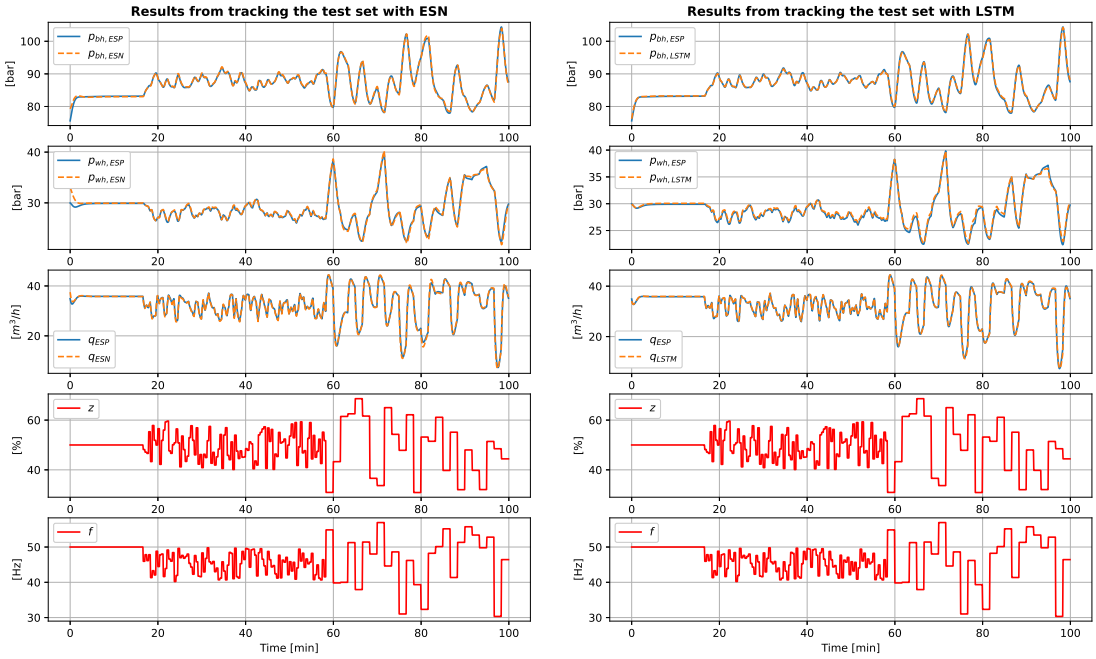
The final model was found by training the model for 100 epochs several times collecting the model yielding the lowest NRMSE on the validation set. The resulting best model had an NRMSE of 0.62. Note that this is somewhat luck-based due to the gradient being calculated using random data points. It should be expected to conduct multiple runs if an equivalent model is desirable. However, there exist methods to achieve even better models faster, but these were not used here as they are outside the scope of the project.

# 4 Results

In the following subchapters, three experiments will be conducted on the final ESN LSTM implementations. Each experiment and the results will be presented in the respective subchapter. Note that the 200 first samples are excluded from every error calculation due to being a warm-up stage ensuring same starting point for the networks and the simulator. The tested ESN and LSTM models was trained for 3 seconds and 17.5 minutes, respectively.

## 4.1 Tracking of the Test Set

In this experiment, both networks were tested on how well they could track the test set. This set consisted of 50% slower and 50% faster randomly generated dynamics in order test how the models behaved when given random input. The resulting network responses are shown in figure 24(a) and 24(b) together with the actual simulator outputs, and the respective calculated errors are presented in table 5 and 6.




(a) ESN (b) LSTM

**Figure 24:** Results from the ESN (a) and LSTM (b) models tracking the test set. The network outputs are plotted in orange, and the simulator output in blue.

| State | MSE | MAPE [%] |
|---|---|---|
| $p_{bh}$ | 0.0000189 | 0.23611 |
| $p_{wh}$ | 0.0000093 | 0.64431 |
| $q$ | 0.0000604 | 1.20837 |
| Averaged total | 0.0000295 | 0.69626 |

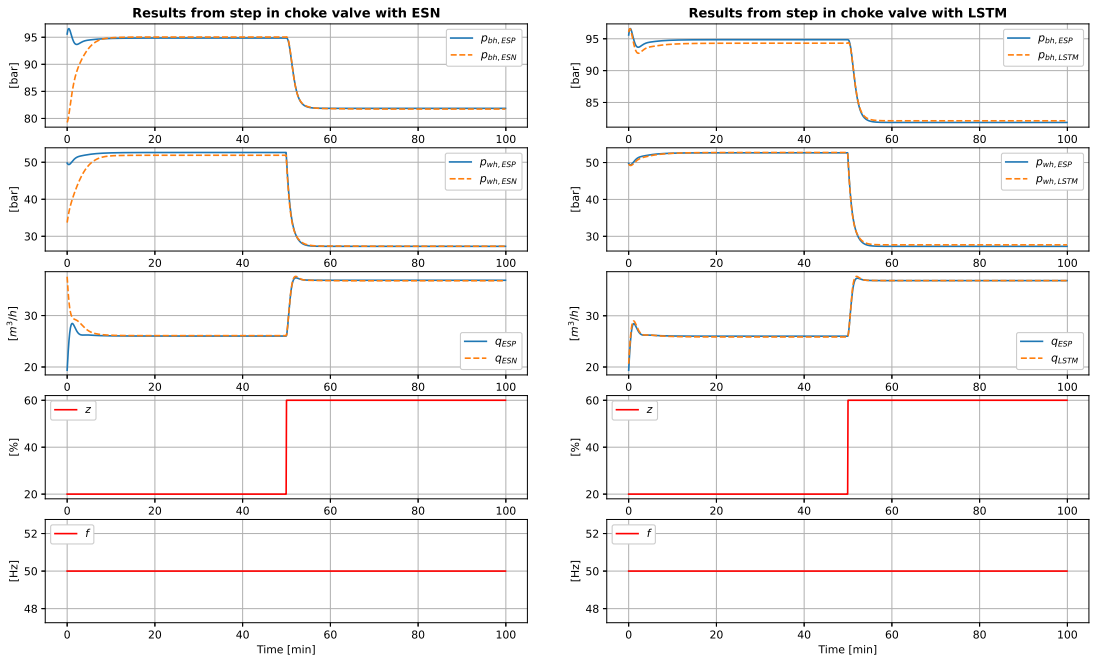
**Table 5:** ESN: calculated errors from tracking the test set.

| State | MSE | MAPE [%] |
|---|---|---|
| $p_{bh}$ | 0.0000894 | 0.25028 |
| $p_{wh}$ | 0.0000717 | 1.03272 |
| $q$ | 0.0001231 | 0.83335 |
| Averaged total | 0.0000947 | 0.70545 |

**Table 6:** LSTM: calculated errors from tracking the test set.

## 4.2 Step Response of the Choke Valve

This experiment is the same as the one conducted in chapter 3.3.2. It was conducted to see how the models behave when there is a step in the choke valve. The resulting network responses are shown in figure 25(a) and 25(b) together with the actual simulator outputs, and the respective calculated errors are presented in table 7 and 8. An experiment with a reverse step was also conducted, and the results are available in appendix B.1.




(a) ESN (b) LSTM

**Figure 25:** Responses from the ESN (a) and LSTM (b) after a step in the choke valve. The network outputs are plotted in orange, and the simulator output in blue.

| State | MSE | MAPE [%] |
|---|---|---|
| $p_{bh}$ | 0.0000054 | 0.15532 |
| $p_{wh}$ | 0.0000361 | 0.69701 |
| $q$ | 0.0000024 | 0.28065 |
| Averaged total | 0.0000147 | 0.37766 |

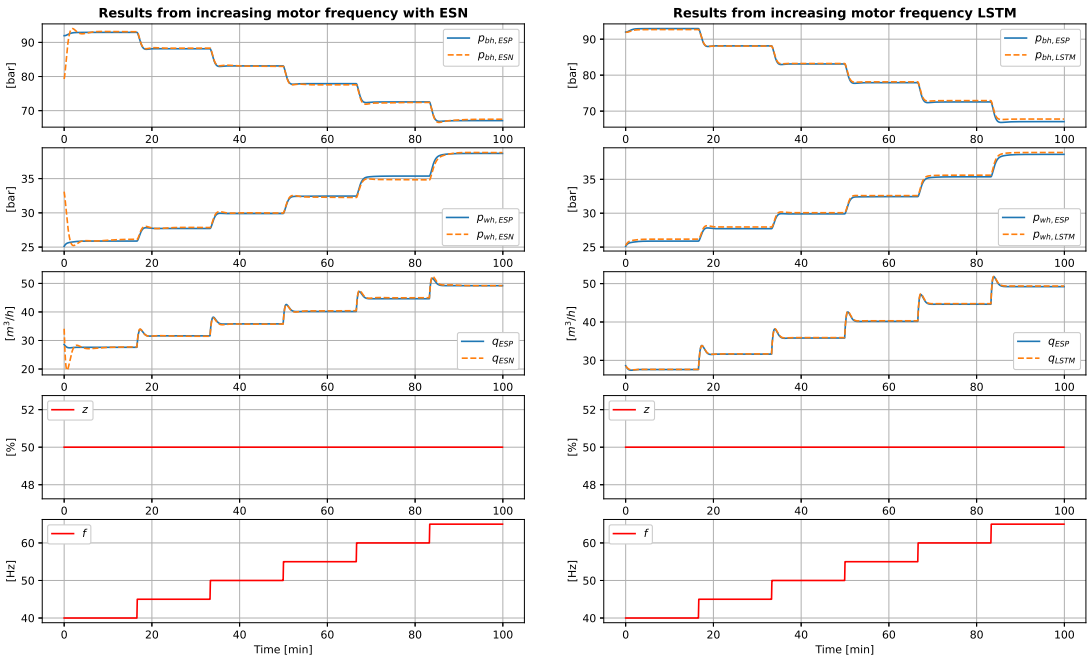
**Table 7:** ESN: calculated errors from the choke valve step test.

| State | MSE | MAPE [%] |
|---|---|---|
| $p_{bh}$ | 0.0001656 | 0.41465 |
| $p_{wh}$ | 0.0000591 | 0.91752 |
| $q$ | 0.0000161 | 0.27391 |
| Averaged total | 0.0000803 | 0.53536 |

**Table 8:** LSTM: calculated errors from the choke valve step test.

## 4.3 Gradually Increased Motor Frequency

This experiment is the same as the one conducted in chapter 3.3.3. It was conducted to see how the models behave when the motor frequency gradually increases. The resulting network responses are shown in figure 26(a) and 26(b) together with the actual simulator outputs, and the respective calculated errors are presented in table 9 and 10. An experiment with decreasing motor frequency was also conducted, and the results are available in appendix B.2.




(a) ESN (b) LSTM

**Figure 26:** Responses from the ESN (a) and LSTM (b) networks when gradually increasing motor frequency. The network outputs are plotted in orange, and the simulator output in blue.

| State | MSE | MAPE [%] |
|---|---|---|
| $p_{bh}$ | 0.0000159 | 0.29415 |
| $p_{wh}$ | 0.0000104 | 0.61208 |
| $q$ | 0.0000139 | 0.42845 |
| Averaged total | 0.0000134 | 0.44489 |

**Table 9:** ESN: calculated errors from gradually increasing motor frequency.

| State | MSE | MAPE [%] |
|---|---|---|
| $p_{bh}$ | 0.0001553 | 0.42378 |
| $p_{wh}$ | 0.0000350 | 0.71018 |
| $q$ | 0.0000190 | 0.30518 |
| Averaged total | 0.0000698 | 0.47971 |

**Table 10:** LSTM: calculated errors from gradually increasing motor frequency.

# 5 Discussion

In this section, the reported results in chapter 4 will be discussed. Although both the MSE and the MAPE are reported, only MAPE can be used to compare the resulting models. This is because MSE is scale dependent (Chen et al., 2017), and the models use different normalizations.

Investigation of the results from the first experiment in chapter 4.1 shows that the ESN and LSTM on average perform equally well when tracking the test set with an averaged total MAPE of $\approx 0.70\%$ and $\approx 0.71\%$, respectively. This is also evident from qualitatively inspecting figure 24(a) and 24(b). However, some smaller differences in performance can be seen from reviewing the individual states. Meanwhile, the two networks perform equally well when tracking $p_{bh}$, the ESN is $\approx 0.4\%$ better on $p_{wh}$, whereas the LSTM is $\approx 0.4\%$ better on $q$. Though, these differences in performance are too small to be seen from only inspecting the figures. Furthermore, MSE is in general known for being particularly sensitive to outlying values (Chen et al., 2017). This suggests that both models have few outliers since all reported MSE values are close to zero. Note that MSE still can give an indication of individual performance even though it can not be used to compare the networks directly. It is also expected that the reported MSEs for the LSTM are somewhat larger than those from ESN due to the LSTM data being normalized on a wider interval.

In the second experiment, in chapter 4.2, ESN performs slightly better than the LSTM with a total averaged MAPE of $\approx 0.38\%$ and $\approx 0.54\%$, respectively. From visual inspection of figure 25(b) it is evident that the LSTM struggles to reach the starting point of $p_{bh}$ leaving a $0.5 - 1.0$ bar offset. This is further supported by the MAPEs of the state which indicate that the ESN performs almost 0.25% better on average than the LSTM. Similarly, it can be seen that the ESN struggles to reach the starting point of $p_{wh}$ leaving a somewhat smaller offset. Though, the error metrics reveal that the ESN still performs about 0.20% better on average than the LSTM. For the last state, $q$, it is clear from both the figure and the error metrics that both models perform equally well with a MAPE of $\approx 0.28\%$ and $\approx 0.27\%$, respectively. Another interesting observation is that the ESN performs on average $\approx 0.12\%$ worse in the reverse step test reported in appendix B.1. The main contributors to this result seem to be $p_{wh}$ and $q$. In both experiments, it seems like the ESN model struggles to reach the larger steady state. The LSTM, on the other hand, performs overall equally well in this experiment, but with an increase in the MAPE of $p_{bh}$ and $q$ and a decrease in $p_{wh}$. Hence, it seems like the models perform better if the simulation starts with the parts where the models struggle the most. One reasonable explanation is therefore that the results are affected by the exclusion of the 200 first data points suggesting that the step should have appeared later. However, this had only a small effect on the results and should not be enough to invalidate them. In both experiments, the MSE is close to zero suggesting also here that there are few outliers in the network responses.

The results from the third experiment, described in chapter 4.3, indicate that both networks performed equally well overall. The ESN performed $\approx 0.1\%$ better on $p_{bh}$ and $p_{wh}$, whereas the LSTM performed $\approx 0.1\%$ better on $q$. Inspection of the reversed ex-

periment with decreasing motor frequency, reported in appendix B.2, shows much of the same tendencies as in the experiment with the increasing motor frequency. However, the LSTM performs $\approx 0.07\%$ better on average. This is due to the LSTM performing $\approx 0.2\%$ better on $q$. Further comparison of these two experiments suggests that they are also slightly affected by excluding 200 data points in the beginning. Note that this was far less evident than discussed in the second experiment.

From these results, it is clear that the ESN and LSTM perform equally well with an average MAPE of all experiments below 1%. This is to some extent expected as ESN (Jaeger, 2007) and LSTM (Goodfellow et al., 2016) are two architectures of the RNN specialized in modeling (dynamical) systems where long-term dependencies play a crucial role. However, finding hyperparameters for the LSTM was relatively time-consuming due to a training time of about 17.5 minutes, compared to the 3 seconds consumed when training the ESN. This suggests that the ESN might be easier to implement because many more configurations can be tested within a given timeframe. On the other hand, far less time was spent on the implementation of the LSTM in this project. It may therefore exist better implementations requiring considerably less training, and also other methods for tuning the LSTM which could yield an even better model. Yet, it is unlikely that any LSTM implementation could be guaranteed a training time below 3 seconds due to the non-convex optimization problem that appears during training. Moreover, the main idea with the ESN was to make it computationally more efficient by only adapting the output layers (Jaeger and Haas, 2004), outperforming the LSTM (and other RNN architectures) in terms of the training time as it utilizes linear regression which in comparison is guaranteed to be fast.

Since the recurrent weights in the reservoir are initialized randomly (Jaeger and Haas, 2004), not all the dynamics are necessarily useful for the particular task at hand. In the LSTM, on the other hand, these weights are instead adapted through training. This should, given the right parameters, facilitate better learning of a model and thus better performance. Although the simulator in this work provided unlimited amounts of data, other applications might only offer a fraction in comparison. The ESN must at least have as much data as the sum of the bias, input and reservoir sizes in order to provide a determined regression problem (Lukoševičius, 2012). Sufficient model generalization might thus be hard to obtain if the amount of data demands a reservoir that is too small to cover all the dynamics in the system. In this case, an LSTM might do better as the low amount of data still can provide a usable model if trained properly. Another important difference is that the LSTM is an architecture specialized in processing almost any kind of sequential data because it overcomes the vanishing gradient problem associated with the general RNN (Smagulova and James, 2020). In comparison, the ESN does not use BPTT and is therefore not affected by the vanishing gradient problem. It also overcome the large training time and the difficulty with finding hyperparameters making it particularly suited for recognizing complex dynamical systems (Osnes, 2020). These traits are a result of the randomly initialized recurrent weights, and thus the reason why ESNs can not be used for applications like machine language translation like the LSTM.

# 6 Conclusion and Further Work

## 6.1 Conclusion

The work with this specialization project has successfully fulfilled the primary and secondary objectives stated in chapter 1.2. Through the process of writing the theory chapter, a thorough understanding of the ESP and the ESN was obtained. This work has also resulted in a successful implementation of a simulator for the ESP lifted well in addition to a framework for future implementations of ESNs. Furthermore, the LSTM was also implemented and the performance of the ESN and the LSTM modeling the ESP lifted well was compared. This has made it possible to conclude upon the research questions stated at the end of the aforementioned chapter:

- With an average MAPE below 1%, the ESN was able to successfully model the nonlinear behavior of the ESP lifted well.
- The results and discussion presented in chapter 4 and 5, respectively, suggests there are no significant difference in performance between a model obtained from using an ESN and an LSTM.
- In the discussion in chapter 5, it is argued that the ESN requires significantly lower training time and that the hyperparameters are easier to obtain compared to the LSTM. The results in this work therefore suggests that the ESN is better suited for black-box modeling of nonlinear dynamical systems as long as enough data is available.

## 6.2 Further Work

During the work with this report some subjects of interest for future work became apparent. These are listed below together with a brief explanation of their relevance.

- Compare the ESN and LSTM in presence of disturbance. This is relevant because it would indicate the robustness of the networks.
- Compare the ESN with a more advanced implementation of the LSTM. This would indicate if the LSTM is able to perform even better and if an LSTM model could be obtained faster than reported here.
- Compare the LSTM with an ESN with implemented feedback. This is somewhat related to the previous bullet point, but could also be conducted on its own. In either case it would indicate if having feedback to the reservoir would have a large impact on the training time and performance of the ESN.

# Appendix A Resulting Plots From the Reversed ESP Experiments

## A.1 Reverse Step Response of the Choke Valve

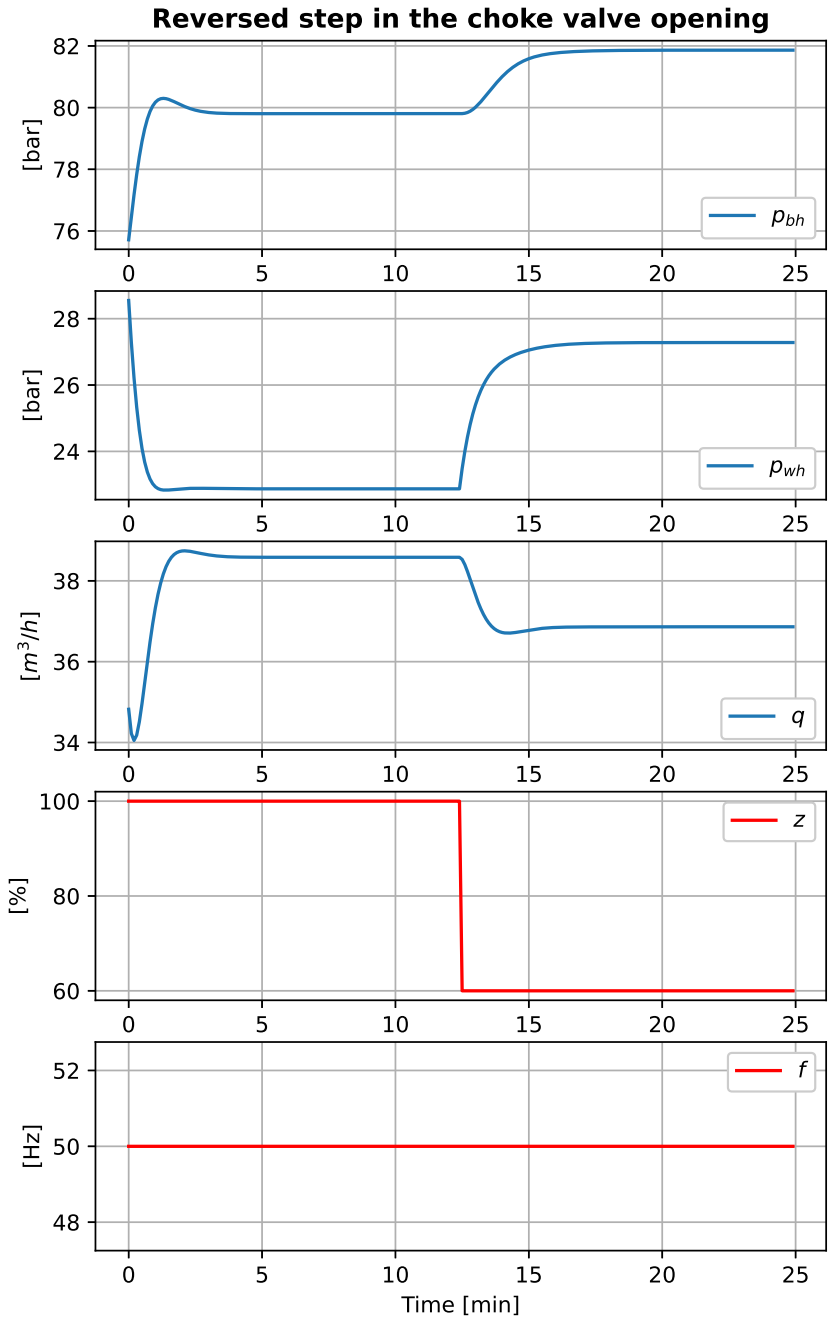



**Figure 27:** Results from running the simulator with a reversed step in the choke valve opening and constant motor frequency.

## A.2 Gradually Decreased Motor Frequency

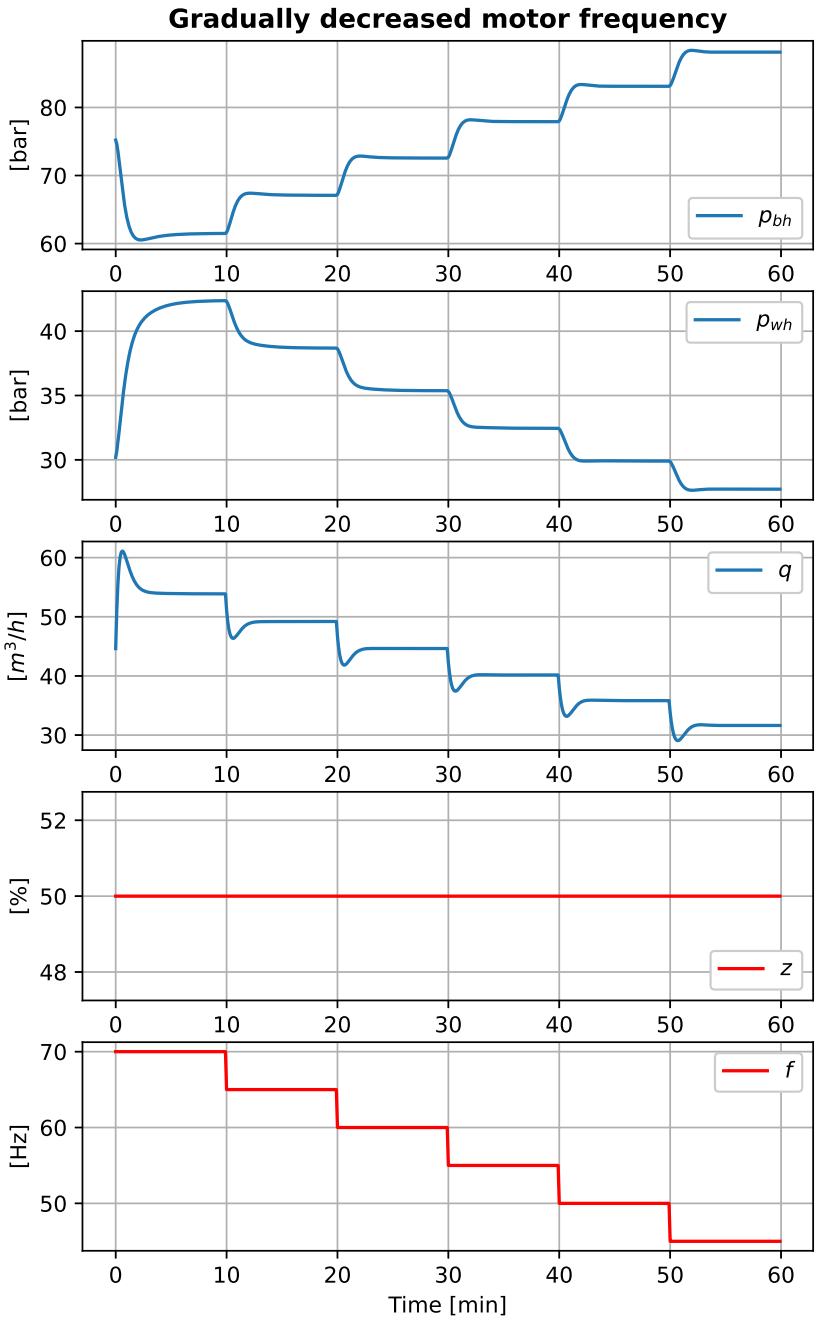
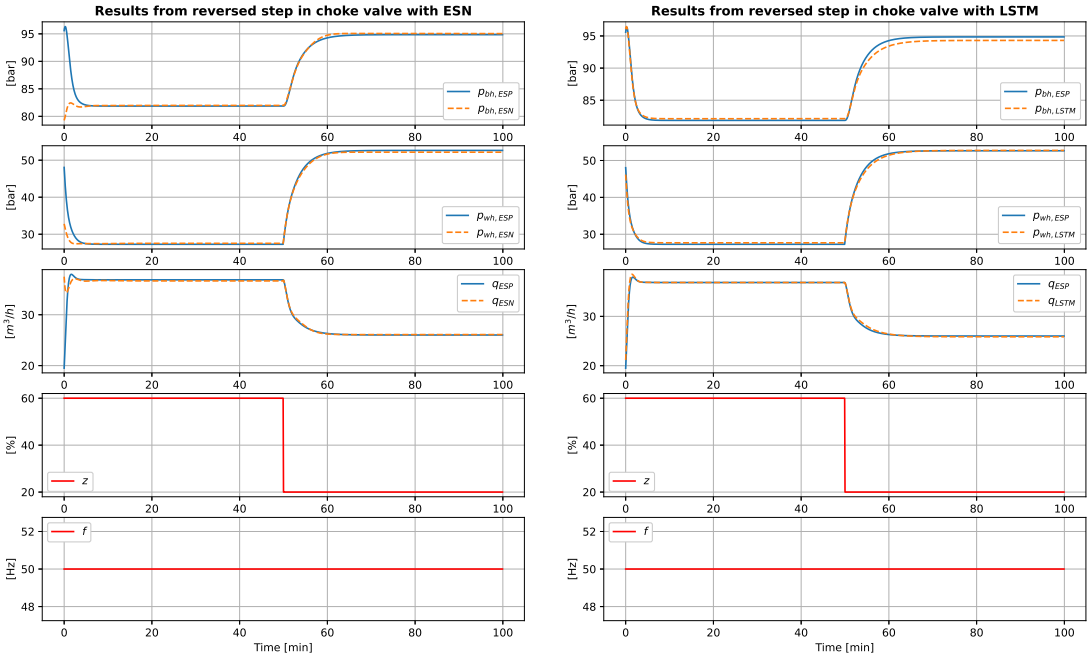



**Figure 28:** Results from running the simulator with constant choke valve opening and gradually decreasing the motor frequency.

# Appendix B Results From the Reversed ESN and LSTM Experiments

## B.1 Revers Step Response of the Choke Valve




(a) ESN (b) LSTM

**Figure 29:** Responses from the ESN (a) and LSTM (b) after a reverse step in the choke valve. The network outputs are plotted in orange, and the simulator output in blue.

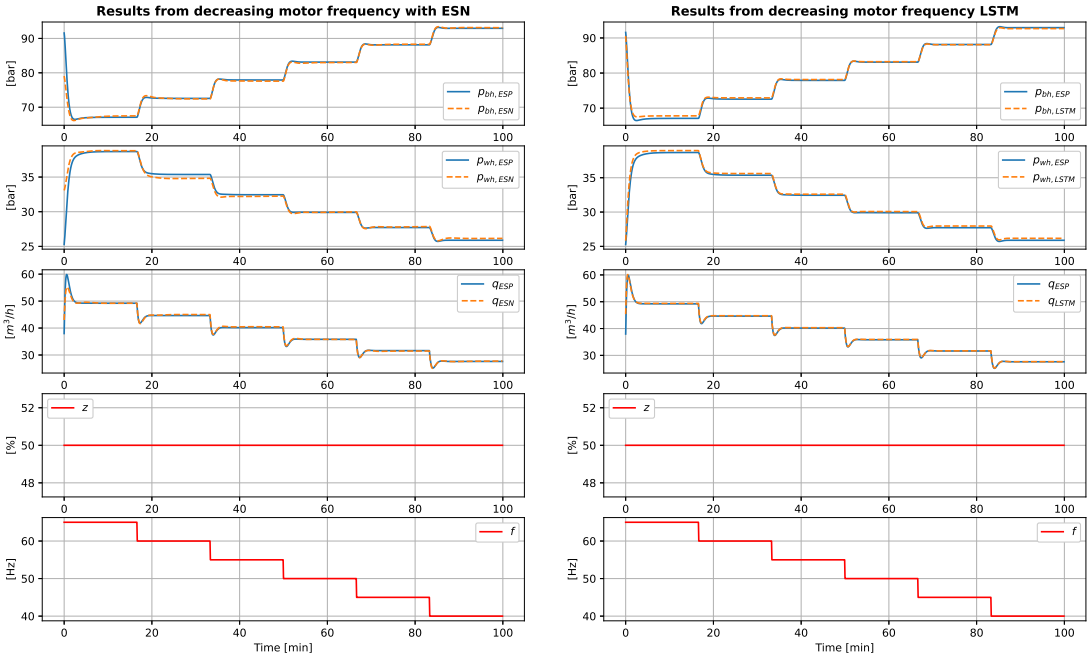
| State | MSE | MAPE [%] |
|---|---|---|
| $p_{bh}$ | 0.0000110 | 0.20475 |
| $p_{wh}$ | 0.0000209 | 0.85707 |
| $q$ | 0.0000067 | 0.43569 |
| Averaged total | 0.0000129 | 0.49917 |

**Table 11:** ESN: calculated errors from the reverse choke valve step test.

| State | MSE | MAPE [%] |
|---|---|---|
| $p_{bh}$ | 0.0002768 | 0.51600 |
| $p_{wh}$ | 0.0000706 | 0.81728 |
| $q$ | 0.0000240 | 0.37406 |
| Averaged total | 0.0001238 | 0.56911 |

**Table 12:** LSTM: calculated errors from the reverse choke valve step test.

## B.2 Gradually Decreased Motor Frequency




(a) ESN (b) LSTM

**Figure 30:** Responses from the ESN (a) and LSTM (b) networks when gradually decreasing motor frequency. The network outputs are plotted in orange, and the simulator output in blue.

| State | MSE | MAPE [%] |
|---|---|---|
| $p_{bh}$ | 0.0000113 | 0.22796 |
| $p_{wh}$ | 0.0000132 | 0.73788 |
| $q$ | 0.0000132 | 0.48964 |
| Averaged total | 0.0000126 | 0.48516 |

**Table 13:** ESN: calculated errors from gradually decreasing motor frequency.

| State | MSE | MAPE [%] |
|---|---|---|
| $p_{bh}$ | 0.0000592 | 0.25838 |
| $p_{wh}$ | 0.0000285 | 0.70295 |
| $q$ | 0.0000146 | 0.27489 |
| Averaged total | 0.0000341 | 0.41207 |

**Table 14:** LSTM: calculated errors from gradually decreasing motor frequency.

# References


(2008). Centrilift europe and africa esp failures 1999-2008.

(2018). Extractive industries : the management of resources as a driver of sustainable development.

Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M. (2019). CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36.

Binder, B. J. T., Pavlov, A., and Johansen, T. A. (2015). Estimation of flow rate and viscosity in a well with an electric submersible pump using moving horizon estimation. In *IFAC-PapersOnLine*, volume 28, pages 140–146.

Chen, C., Twycross, J., and Garibaldi, J. M. (2017). A new accuracy measure based on bounded relative error for time series forecasting. *PloS one*, 12(3):e0174202–e0174202.

Diaz, C. and Nicolas (2012). Effects of sand on the components and performance of electric submersible pumps.

Egeland, O. and Gravdahl, J. T. (2003). Modeling and simulation for automatic control.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Gravdahl, J. T. (2019). Kybernetikk introduksjon: Innføring i dynamikk og reguleringsteknikk.

Grimstad, B. (2022). Ttk28 modeling with neural networks - deep learning: Deep feed-forward networks.

Gros, S. (2021). Modelling and simulation - lecture notes for the ntnu/itk course ttk4130.

Hernes, S. B. (2020). Practical nmpc of electrical submersible pumps based on echo state networks.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hou, Z.-S. and Wang, Z. (2013). From model-based control to data-driven control: Survey, classification and perspective. *Information sciences*, 235:3–35.

Hoyer, M., Eivazi, S., and Otte, S. (2022). Efficient lstm training with eligibility traces. In *Artificial Neural Networks and Machine Learning – ICANN 2022*, Lecture Notes in Computer Science, pages 334–346. Springer Nature Switzerland, Cham.

Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, GMD - German National Research Institute for Computer Science.

Jaeger, H. (2007). Echo state network. *Scholarpedia*, 2(9):2330. revision #196567.

Jaeger, H. and Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Langseth, H. (2022). Tdt4171 artificial intelligence methods lecture 8 – basic anns.

Lukoševičius, M. (2012). *A Practical Guide to Applying Echo State Networks*, pages 659–686. Springer Berlin Heidelberg, Berlin, Heidelberg.

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Comput.*, 14(11):2531–2560.

Nelles, O. (2013). *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*. Springer.

Osnes, I. (2020). Recurrent neural networks and nonlinear model-based predictive control of an oil well with esp.

Pavlov, A. K., Krishnamoorthy, D., Fjalestad, K., Aske, E. M. B., and Fredriksen, M. (2014). Modelling and model predictive control of oil wells with electric submersible pumps. *2014 IEEE Conference on Control Applications (CCA)*, pages 586–592.

Perera, F. and Nadeau, K. (2022). Climate change, fossil-fuel pollution, and children's health. *The New England journal of medicine*, 386(24):2303–2314.

Schiller, U. D. and Steil, J. J. (2005). Analyzing the weight dynamics of recurrent learning algorithms. 63:5–23.

Smagulova, K. and James, A. P. (2020). *Overview of Long Short-Term Memory Neural Networks*, pages 139–153. Springer International Publishing, Cham.

Sussillo, D. and Abbott, L. F. (2015). Random walk initialization for training very deep feedforward networks. *arXiv.org*.

Takács, G. (2009). Electrical submersible pumps manual : design, operations, and maintenance.

Zambrano, V., Mueller-Roemer, J., Sandberg, M., Talasila, P., Zanin, D., Larsen, P. G., Loeschner, E., Thronicke, W., Pietraroia, D., Landolfi, G., Fontana, A., Laspalas, M., Antony, J., Poser, V., Kiss, T., Bergweiler, S., Pena Serna, S., Izquierdo, S., Viejo, I., Juan, A., Serrano, F., and Stork, A. (2022). Industrial digitalization in the industry 4.0 era: Classification, reuse and authoring of digital models on digital twin platforms. 14:100176.