Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Geoscience and Petroleum

Henrik Ågrav

Proxy Modeling for Waterflooding Optimization

Master's thesis in Reservoir Engineering
Supervisor: Ashkan Jahanbani Ghahfarokhi
Co-supervisor: Cuthbert Shang Wui Ng

June 2021

NTNU
Norwegian University of
Science and Technology

Henrik Ågrav

Proxy Modeling for Waterflooding Optimization

Master's thesis in Reservoir Engineering
Supervisor: Ashkan Jahanbani Ghahfarokhi
Co-supervisor: Cuthbert Shang Wui Ng
June 2021

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Geoscience and Petroleum

Abstract

Waterflooding is one of the most common methods of increasing the productivity of a hydrocarbon reservoir; it is cheap, effective, and has been performed for a long time. Optimization of a waterflooding process is typically done using numerical reservoir simulators, combined with either trial and error or an optimization algorithm. Due to the complexity of reservoir this means that this optimization process will be time consuming, as a more complex and accurate reservoir model will require the numerical simulator to run for a long time for each attempted simulation.

This thesis proposes an approach to use a proxy model for the reservoir, and use this model for optimization of the cumulative oil production of a synthetic reservoir. The model will be developed using machine learning to make an artificial neural network, which will be able to replicate the reservoir simulator responses. This proxy model will then be coupled together with a genetic algorithm, which is a meta-heuristic optimization algorithm. The optimization using the proxy model will then be compared with optimization using a numerical reservoir simulator.

The proxy model is able to accurately replicate the numerical reservoir simulator responses. The proxy model has a slight over-prediction of 0.75 %. Both methods achieve a similar result when optimized with the genetic algorithm. The optimization with the proxy model takes 71 seconds, while it takes 18 hours, 32 minutes, and 30 seconds with the numerical reservoir simulator. This means that the optimization algorithm goes 940 times faster when it is done with a proxy model as opposed to a conventional numerical reservoir simulator.

Sammendrag

Vannstrømning er en av de mest vanlige metodene for å øke produktiviteten til et olje & gass reservoar; det er billig, effektivt, og man har drevet med det veldig lenge. optimalisering av vannstrømningsprosesser gjøres typisk ved å bruke en nummerisk reservoar simulator, og enten teste ute forkjellige scenarioer eller ved å bruke en optimaliseringsalgoritme. Grunnet kompleksiteten til reservoarer vil denne optimaliseringen av produksjonen ta veldig lang tid, siden kompliserte reservoarer vil ta lengere tid på en nummerisk reservoar simulator.

Denne oppgaven foreslår en metode der man lager en proxymodell for reservoaret, og bruker denne modellen til å optimalisere den kumulative oljeproduksjonen til et syntetisk reservoar. Modellen blir utviklet ved å bruke et kunstig nevralt nettverk, som er sentralt innen konseptet maskinlæring. Dette netverket vil kunne klare å gjenskape de samme resultatene som en nummerisk reservoar simulator. Denne proxymodellen vil så bli brukt sammen med en genetisk algoritme, som er en meta-heuristisk optimaliserings algoritme. Optimaliseringen av proxymodellen vil så bli sammenliknet med en tilsvarende optimalisering ved bruk av numerisk reservoar simulator.

Proxymodellen var i stand til å gjenskape resultatene til den numeriske reservoarsimulatoren. Modellen hadde en tendens til å over-predikere med 0.75 %. Både proxymodellen og reservoarsimulatoren fikk tilsvarende resultat etter optimaliseringen med den genetiske algoritmen. Optimaliseringen med proxymodellen tok 71 sekunder, imens det tok 18 timer, 32 minutter, og 30 sekunder med den nummeriske reservoarsimulatoren. Dette betyr at optimaliseringsprosessen går 940 ganger raskere dersom man bruker en proxymodell kontra en mer standard nummerisk reservoarsimulator.

Preface

This thesis will conclude my work in the concept of the application of proxy model in the petroleum industry, and the application of proxy modeling for the process of waterflooding. It is a continuation of my specialization project, which was a literature study on the applications of proxy modelling. The thesis is the final piece of work over the course of my five years as a reservoir engineering student, and will result in a master's degree within reservoir engineering from the Norwegian University of Science and Technology (NTNU).

My supervisor for this thesis is Ashkan Jahanbani Ghahfarokhi, who is an associate professor at the Department of Geoscience and Petroleum at NTNU. My co-supervisor has been Cuthbert Shang Wui Ng, who is a Ph.D candidate at the Department of Geoscience and Petroleum.

This thesis is a continuation of my specialization project[1], and some parts of this project have been reused for this master thesis. Chapter 1 has been reused and slightly altered, as the objective of this thesis is not the same as the specialization project. Some parts remain the same, while others are re-written or expanded upon. Chapter 2 inherits a lot from chapters 2 & 3 from the specialization project. The new chapter 2 has been expanded a bit from the sections that are taken from the specialization project, and some parts have also been re-written. Chapters 3,4,5, and 6 are brand new for this master's thesis.

For this thesis, a lot of work has been done writing code in python. This code is available on my github.[1]

Trondheim, June 10, 2021
Henrik Ågrav

[1]https://github.com/haagrav/TPG4920—Master-thesis

Acknowledgements

I've received a lot of help during the process of making this thesis, both academically and mentally. I am very grateful for all the support I've gotten.

First I would like to thank my supervisor Ashkan Jahanbani Ghahfarokhi. He is the one who orchestrated this whole project and helped me through the rougher patches of this project.

I would also like to thank my co-supervisor Cuthbert Shang Wui Ng. He was always very helpful whenever I needed help with the implementations for this project. He made sure that if there was anything I was wondering I should just send him a message, and he'd always reply very quickly. If I was stuck, Cuthbert always had a good idea of how to move forward with the project.

It's been a different year, and I don't want to imagine what it would have been like without the support of my friends and family. I want to specifically thank my girlfriend Tilde Hjermann, who's been supporting me both in my highpoints and lowpoints. She's very good at keeping me relaxed when I need it the most, either by encouraging a lazy afternoon on the couch or by taking me out on ski trips. I also want to give a special thanks to my friend Lars Angelsen. From afar he's keenly listened to me ramble on about the specifics of my project without knowing a single thing about reservoir engineering.

I also want to thank everyone I've met during my 5 year stay in Trondheim. I'm very grateful for all the people that I've seen almost every day, and people I've just met here and there. You've all made my time in Trondheim very special, and because of you this city has a very special place in my heart.

Contents

List of Figures

List of Tables

List of Abbreviations

IOR	**I**mproved **O**il **R**ecovery
EOR	**E**nhanced **O**il **R**ecovery
RSM	**R**esponse **S**urface **M**odel
DOE	**D**esign **O**f **E**xperiment
ROM	**R**educed **O**rder **M**odel
POD	**P**roper **O**rthogonal **D**ecomposition
TPWL	**T**rajectory **P**iece**W**ise **L**inearization
DEIM	**D**iscrete **E**mpirical **I**nterpolation **M**ethod
SVD	**S**ingle **V**alue **D**ecomposition
AI	**A**rtificial **I**ntelligence
ANN	**A**rtificial **N**eural **N**etwork
LHS	**L**atin **H**ypercube **S**ampling
ReLu	**Re**ectified **L**inear **U**nit
ACO	**A**nt **C**olony **O**ptimization
PSO	**P**article **S**warm **O**ptimization
GA	**G**enetic **A**lgorithm
FOPR	**F**ield **O**il **P**roduction **R**ate
BHP	**B**ottom **H**ole **P**ressure

Chapter 1

Introduction

1.1 Background and motivation

Numerical reservoir simulation models have long been the industry standard for modeling fluid flow through porous media. Reservoir models are specially tailored for each specific reservoir, taking in geological field- measurements and interpretations such as seismic imaging, direct field measurements, and also incorporating laboratory measurements. By adding this data, the reservoir models become more and more akin to how the reservoir acts in real life. A more detailed and complex model will be a more accurate representation of the reservoir. It does, however, make the calculations for a numerical simulator more complex. Added complexity means that more calculations have to be conducted for each time-step. In addition to this the reservoir simulators will involve more assumptions regarding these new physical processes that will be modeled, which will add to the uncertainty of the model. Ultimately this added complexity will directly correlate to a higher computational time.

Having the ability to simulate the flow in a reservoir is an important step, as it is possible to optimize the productivity of the reservoir by testing scenarios on such a simulator before actually committing large economic investments in real life. As there is a clear demand for systems of higher complexity, so called Surrogate Reservoir Model become a common solution to reservoir modelling. Surrogate models are also known as proxy models [24], or meta models. A surrogate model is either an alternative or a subsidiary to the numerical reservoir model. Surrogate models have a substantially reduced complexity allowing for accurate models which require less computational time in comparison to traditional reservoir simulation.

One of the methods for developing these surrogate models is to use artificial neural networks. An artificial neural network is inspired by biological brains, and it uses a large amount of data to train a network of artificial neurons to complete specifically designated tasks. One of the most common methods is to supply the network of a dataset containing sample data, where the purpose of the network is to find an input-output relationship. Based on the input data the network will predict an output, which is then compared to a desired output which is also provided. The network learns through a method known as backpropagation, meaning that the error between the predicted output and the desired output is calculated, and the different connections between the neurons in the network is strengthened or weakened based on how the connection between neurons contributed to the error.

Applying machine learning algorithms to reservoir data will aid to reduce the computational complexity of the simulation. The trade of is that the development of these surrogate models requires quite a bit of computation time. These surrogate models have been developed successfully in the oil- and gas industry. This allows for reservoir models with lower computational times and with an acceptable accuracy as opposed to numerical models. But even if the concept is successful, there are also downsides with applying this method. This method requires a lot of data. This data can be sampled from real measurements in the field, resulting in a purely data-driven surrogate model. This requires a lot of data, which can be difficult and time consuming to collect. It is also important to collect good quality data for the network. Poor training data may result in unforeseen and unwanted biases which might prevent the network from being able to perform the intended task within the preferred accuracy.

As this purely data-driven method requires a lot of different measurements one can also supply the measured dataset with values gathered from reservoir simulation, resulting in a hybrid model. While seemingly appealing, this method is not as preferred as the data-driven one. Error and uncertainty in the data-driven model will occur as it is an approximation method, while in this hybrid equation this uncertainty from interpreting the data will also be coupled with error and uncertainty coming from several assumptions made when gathering data using a reservoir simulator. Neural networks have a big disadvantage in that they are difficult to interpret. Due to the internal complexity and the large amounts of data neural networks are able to process, most humans look at these neural network as more of a black box. In comparison, the conventional reservoir models are based on equations which are made from making assumptions in explainable physics.

The purpose of both the numerical models and the surrogate models is to accurately model how the fluids flow in the reservoir, such as to optimize the productivity of the reservoir. The most common method of increasing the productivity of a hydrocarbon reservoir is by injecting water into the reservoir. This thesis will be focused on the application of such a hybrid model for the purpose of optimizing the reservoir production in a water injection process.

1.2 Objectives

The objective for this thesis is to use artificial intelligence to develop a hybrid model for a reservoir. For this thesis the model will be designed specifically to model a waterflooding operation. The model will be estimating the field oil production rate (**FOPR**) of the synthetic reservoir. The objective of this thesis can be summarized as such

1. Develop a hybrid for a synthetic reservoir, using artificial intelligence
2. Apply a genetic algorithm to the proxy model to optimize the cumulative oil production of the reservoir
3. Apply a genetic algorithm to a numerical reservoir simulator to optimize the cumulative oil production of a reservoir
4. Compare results from the optimization process with a proxy and the numerical reservoir simulator

1.3 Thesis outline

This Thesis is split up into a total of 6 chapters. The first chapter will include the background and motivation for the project. It will also cover the project objective, and an overview of the structure of the project as an entirety. Chapter 2 will introduce the theoretical background of the concepts that are used for the work in this project. The topics mentioned in this chapter will mention the basics of hydrocarbon recovery from a reservoir, and the process of developing a proxy model for the reservoir. It will also cover machine learning, and the mechanics of a genetic algorithms, such as the operators and the overall workflow of such an optimization algorithm. Chapter 3 will include a rundown of how the work has been done for this project. It will introduce the synthetic reservoir used in this project, show the sampling process for the database, and briefly explain how the python code has been done for the various implementations.

Chapter 4 will show results from the procedures described in chapter 3. This will include an elaboration on the performance of the proxy model and the genetic algorithms. It will also talk about the advantages and shortcomings of the procedures used. Chapter 5 will consolidate the results into a more concrete conclusion. Chapter 6 will talk about how the work could have been improved, and suggest improvements that could be done to similar projects in the future.

Chapter 2
Theory

This chapter will first cover a couple of the traditional methods of stimulating a hydrocarbon reservoir. The methods discussed are fundamentally defined within the industry. Then the concept of waterflooding will be covered, as well as some fundamentals within reservoir simulation. The chapter will also cover the concept of proxy modeling, and discuss briefly different methods of developing such a model. Then the chapter will go into more detail on the development of a proxy model using artificial intelligence, the concept of artificial neural networks, how they work and how they learn based on the input data. The final topic that will be covered in the chapter is the broad concept of evolutionary computing, which will have a clear focus on elaborating how a genetic algorithm works.

2.1 Reservoir production

A reservoir is regarded as an underground, stratigraphic container that contains fluid resources. It is highly pressurized, and the fluids can be accessed and produced by drilling a well into the reservoir.

The life of a reservoir starts in a *source rock*. A source rock is a porous, sedimentary rock that contains sufficient amount of biological material that can be generated into a large enough cumulative amount of hydrocarbons to saturate a hydrocarbon reservoir. Biological material in the source rock will over time turn into oil and gas [2]. The hydrocarbons will then migrate through various passages to a more suitable *reservoir rock*. A requirement for the reservoir rock is that there is a cap rock above, which is an impermeable barrier that keeps the hydrocarbons in place. This cap rock could be a shale formation, or, as is common in offshore fields; salt domes. After migrating to the reservoir rock the hydrocarbon fluids will stay in place until it will later be produced.

2.1.1 Recovery stages

The production lifespan of an oil and gas reservoir is often separated into three different categories. These are

1. Primary recovery
2. Secondary recovery
3. Tertiary recovery

It is common to refer to the later stages of recovery as Improved Oil Recovery (**IOR**), while tertiary recovery could be referred to as Enhanced Oil Recovery(**EOR**). Mechanically some of the recovery categories are quite similar, as they are all based on producing hydrocarbons from an underground, porous medium. They do differ quite a bit in terms of human intervention, technical difficulty, how they affect the reservoir and reservoir fluids, and different economical motivations and restrictions.

Primary recovery

Primary recovery is also known as the natural production of a hydrocarbon reservoir. This requires no human interaction other than drilling a well into the reservoir, and perforating in order for the reservoir fluid to naturally flow through the reservoir. All hydrocarbon production relies on the natural energy stored inside the reservoir itself [38]. Hydrocarbon reservoirs are pressurized chambers, which are closed in due to the impermeable cap rock sealing the hydrocarbon fluids inside of the reservoir. This initial reservoir pressure will be the driving force for the production of the hydrocarbons.

As the reservoir is producing reservoir fluids, the pressure will steadily drop, causing a lower production of fluids over time. Depending on the geological features of the reservoir there are a couple of factors that may affect how quick this decay of reservoir pressure will be. Water influx from a nearby water aquifer could be one such factor [44], where water from a connected water aquifer may flow into the reservoir due to pressure equilibrium. This influx of water slows the average pressure decline during the natural depletion of the hydrocarbon reservoir, which causes a prolonged period of production. Most reservoirs will at least experience some production due to water influx [9]. This will have two effects on the reservoir: the pressure will increase due to more fluid being present in the reservoir, and the water will also push the oil out of the reservoir.

Secondary recovery

Secondary recovery is the second phase of reservoir production. As the natural energy of the reservoir is depleted, or at least too low for the reservoir to remain economically reliable, energy has to be added to the reservoir to allow for additional oil recovery [38]. To counteract this decrease in production there are several ways to interact with the reservoir in order to further stimulate the production. The goal of this production phase is to prolong the total life of the field by keeping it pressurized such as to maintain a viable level of production.

Secondary recovery involves interaction with the reservoir, most commonly this will mean to inject water or reservoir gas into the reservoir. Both of these injecting fluids will mimic the natural responses from the reservoir that happen during natural production: water injection will act similarly to a natural water influx from a nearby aquifer, and gas injection will be similar to gas cap expansion. One could say that in the secondary recovery phase, the injected fluids are fluids that are already naturally found in the reservoir [20]. The injection of water is known as waterflooding. In principle, the injection of water aids at increasing the pressure of the reservoir, and it also helps by pushing the oil from the injector well towards the producing well. The water could be injected straight into the reservoir, but if the reservoir has a connected aquifer the water could be injected into the aquifer as well.

Tertiary recovery

After a while, secondary recovery will stop being economically viable. When this occurs, additional supplementary energy is required for the reservoir to remain in production. This is known as the tertiary recovery period, however it is often referred to as enhanced oil recovery (**EOR**). EOR is the process of extracting more oil from the reservoir by injecting substances that are not naturally present in the reservoir [20]. A key distinction from secondary recovery is that with EOR the added energy will be due to the absence of the natural displacement methods caused by primary- or secondary production. The mechanics of EOR are more based on exploiting chemical interactions between fluids in the reservoir, by introducing heat, altering the miscibility of in-situ fluids, or by altering the fluid properties of either injected water or the oil in the reservoir. Thus there is a clear distinction between secondary- and tertiary production.

In order to talk about the mobility of oil it is important to determine how this mobility is defined. Standing [37] defined the mobility of a fluid as the ratio between the fluids permeability and viscosity, as such

$$\lambda = \frac{k}{\mu} \tag{2.1}$$

where λ is the mobility of the fluid. When talking about fluid displacement it is common to refer to the fluid whose saturation decreases as the *displaced* fluid, and the fluid whose saturation is increasing as the *displacing* fluid. When describing a fluid flow system, the ratio between the mobility of the displacing fluid over the displaced fluid is known as the mobility ratio of the fluids. Standing[37] has described the mobility ratio as such

$$M = \frac{\lambda_{displacing}}{\lambda_{displaced}} \tag{2.2}$$

This ratio varies from a value of 0.1 and up to about 10 [37]. If the mobility ratio is less or equal to 1 the system is seen as quite stable, as the fluid front is quite predictable and smooth. If the ratio is above 1 the features may vary a bit more, and it will give us a displacement which is more ragged and causes "viscous fingering". In terms of an oil reservoir, water is generally the displacing fluid while oil is being displaced. For a higher mobility ratio, water moves faster though the reservoir system compared to oil. This means that after a while the well will start to produce water as well as hydrocarbons, because the water has made a "highway" through the reservoir. The main target of EOR is the immobile oil that has been left in the reservoir after conducting secondary recovery. This means oil that is unable to be efficiently produced by conventional waterflooding procedures. One method to produce this oil is to manipulate the injected fluid in order to get a more favorable mobility ratio between the injected fluid and the oil.

The mobility ratio can become better by heating up the oil, causing the viscosity of the oil to decrease [8], which will decreases the mobility ratio. This can be done by injecting steam into the reservoir, as the steam will transfer heat to the oil and then condense into water. This will also have similar effects as with waterflooding, as steam is just water with added energy. It is also possible to add polymers to the injected water, in order to make the water thicker and increase its viscosity, as this will also decrease the mobility ratio [23]. Water infused with polymers will have increased viscosity, due to the generally high molecular weight of the added polymers [20]. This cause the water to more efficiently push the oil towards the producing well, and the waterflooding process will have a greater sweep efficiency. Polymer infused water could help to prevent viscous fingering to appear in the reservoir. Depending on the type of polymer injected the effective permeability of the reservoir may decrease. A reservoir with a decreased effective permeability may experience a less steep decrease in pressure drop during production.

In terms of EOR, miscible gas injection is one of the more effective approaches [44]. This means that the injected fluid will mix with the oil, which is a process that alters some of the properties of the reservoir oil. This new fluid will have some altered properties in comparison with the original reservoir oil, including that it will generally have a lower viscosity, which as mentioned before will provide the reservoir system with a more beneficial mobility ratio. This kind of miscibility usually has to be induced onto the reservoir fluids, as miscibility usually requires quite large pressures. The lowest pressure at which miscibility will occur is known as the minimum miscibility pressure.

2.1.2 Waterflooding

Waterflooding is a part of secondary recovery. It is the process ff injecting water into the reservoir. It has been considered common practice in the oil- and gas industry since the 1930s, when oil production companies started to re-inject the water that was produced from the reservoir [15]. When water starts being produced from the producing wells, it has do be dealt with. On off-shore rigs this produced water can usually be processed such that there's little to no trace of hydrocarbons in it, and then it can be cleaned for particles and released back into the ocean. For on-shore rigs however it's not that simple. This produced water has to be disposed some way or another.

Waterflooding was a natural step forward for the industry as a whole. It is the most widely used process for increasing the productivity of a reservoir due to the availability of water, it has a low cost relative to other injectants, the ease of injecting water into a formation, and water is quite effective at displacing oil in a reservoir [36]. There are two main effects that are achieved when injecting water into a hydrocarbon reservoir

- Voidage replacement
- Oil displacement

Voidage replacement refers to the influx of water into a reservoir as oil is being produced out of the reservoir. As oil is produced form the reservoir, the average reservoir pressure will decrease proportional to the rate of oil production. By injecting water into the reservoir, this pressure decline will become slower or sometimes remedied entirely, causing the reservoir pressure to creep up towards the initial average reservoir pressure [44].

Oil displacement refers to another key attribute of water injection, which is proficient at pushing oil towards the producer well. The oil that has been displaced by water in a waterfloding process can be defined by the following equation

$$N_D = N * E_A * E_V * E_D \tag{2.3}$$

where N_D is the amount of displaced oil, N is the original oil in place, E_A is the areal sweep efficiency, E_V is the vertical sweep efficiency, and E_D is the displacement efficiency.

The areal sweep efficiency refers to the fraction of the reservoir that has been in contact with the injected water in a waterflooding process. Similarly, vertical sweep efficiency is a similar concept just for the vertical plane of a reservoir. Conversely it is not uncommon to refer to the effect of both of these concept as volumetric sweep efficiency, or conformance [44]. The volumetric sweep efficiency is a measure of how much of the reservoir that has been in contact with the injected fluid. There are a lot of variables which determine this parameter, such as the relative flow parameters of

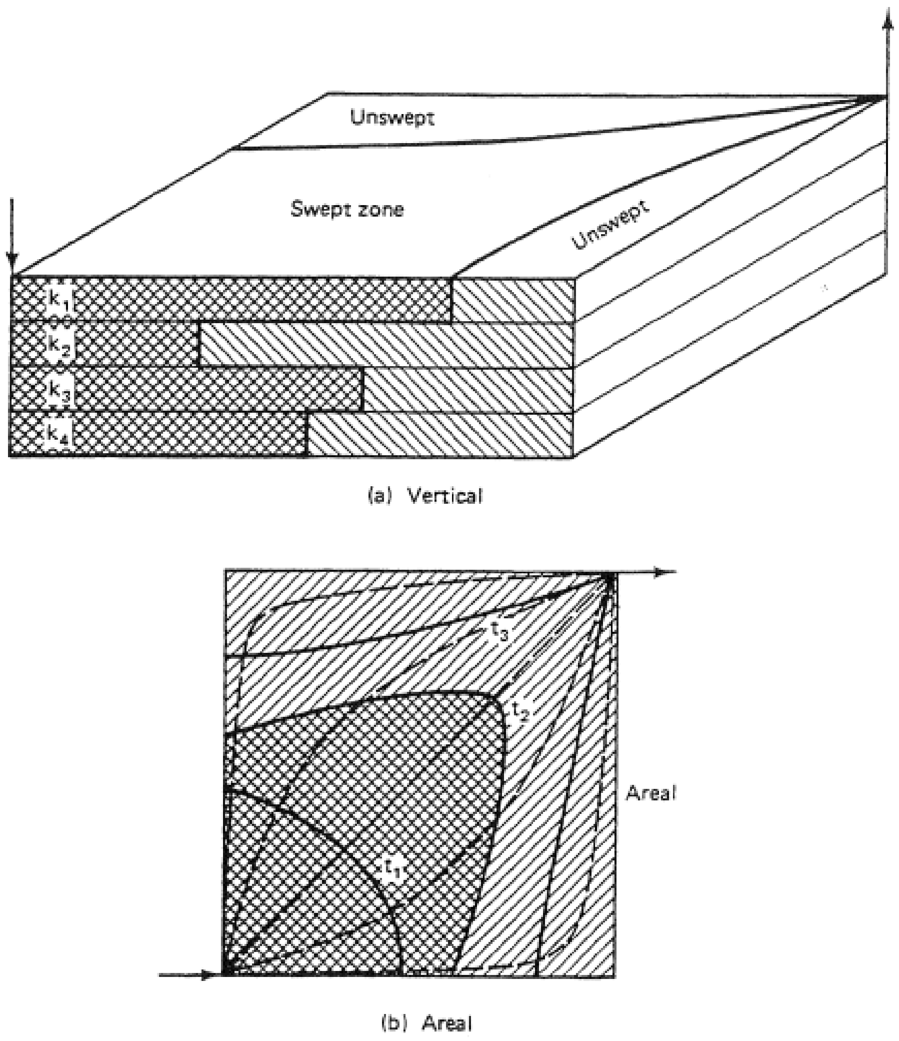

FIGURE 2.1: Schematic of the different sweep efficiencies. The top image shows the vertical sweep efficiency, marked by the crossed out areas. The bottom shows the areal sweep efficiency as the scribbled area. [20]

the fluids, the degree of tilt on the reservoir, the well pattern, and also directional permeability in the reservoir. The displacement efficiency correspond to how effective the waterflooding process will be.

2.2 Reservoir simulation

Models and simulations are often used to get a better grasp of how real-life environments look and feel. The benefit of using such tools is to simulate how real life works without the costs of real life trials and errors. The same goes for reservoir simulation, where it is possible to estimate how different production scenarios will perform on a certain reservoir. The benefit of doing this in a simulator before applying to an actual field is huge, as it is possible to test out these scenarios before actually applying them to a real life reservoir [29]. This drastically increases the odds of achieving a more optimal production from a reservoir.

Numerical reservoir simulators have gone from being a subject for research to being the industry standard tool within reservoir engineering [22]. A simulator models the reservoir as a connected set of blocks, which constitutes the reservoir. Each block is assigned reservoir properties, such as porosity, permeability, and transmisibility. In the simulator, fluid flow is simulated on a grid block level. The procedure of using such simulators has become a regular practice for production optimization and history matching.

One of the common reservoir simulators is ECLISE, which is developed by Sclumberger [32]. The ECLIPSE simulation suite comes with two simulators: ECLIPSE 100 which is a black oil simulator, and ECLIPSE 300 which is a compositional simulator.

2.3 Proxy modelling

George Box was a statistician famous for being one of the great statistical minds of the 20th century. Box [7] has stated that

".. all models are wrong, but some models are useful. "

This quote highlights that models are not some analytical truth in no way, shape, or form. Naturally it is an overly dramatic oversimplification of the concept of models. Models are designed to illustrate how the real world works, and they can paint a good image of how the world seems to function and the relations between all the different parts of the world. This concept of models can also be applied in reservoir engineering.

The most commonly applied modelling practice within reservoir engineering is numerical reservoir simulator models [22]. These are complex models, capable of simulating a wide range of physical and chemical processes that happen in a hydrocarbon reservoir. A numerical simulator calculates the effects caused by these processes for each grid block at each time step. Since a numerical model could include for example more than one million grid blocks the process of simulation is a time consuming matter, as well as being computationally intensive. Recent progressions within computer hardware have provided a lot of help and opportunities to the field of numerical simulation, but still the process is considered time consuming. Thus, there is a lot of incentive to discover an alternative to this conventional reservoir simulation, and one such alternative is known as proxy modelling.

A proxy model is a mathematical or statistically defined function which aims at replicating the input and output response of a real life system. This system could be based on real data gathered from the field or based on a dataset that is generated by a conventional reservoir simulator. Proxy models have a wide range of applicability, where most of these applicable tasks are considered either computationally intensive or labour intensive. The methods involved in making these proxy models could be based on supplying large amounts of data to find a correlation in the data, or they can be developed by making simplifications or assumptions to pre-existing models. Data gathered from a hydrocarbon reservoir can be quite comprehensive and non-linear, making it difficult for an engineer to intuitively make sense of it. A computer on the other hand will analyse the data unbiased. So by making a computer analysis of this data it will often find value in the otherwise un-interpretable data.

There are various methods of developing a proxy model. The method of developing such a proxy model is picked based on the problem that the model is supposed to be solving, as some methods are better applied to some specific types of reservoirs. The main methods of developing such a proxy model [3] are

- Reduced physics method
- Statistical method
- Reduced order method
- Artificial intelligence method

In this section, the basis of the different methods of developing proxy models will be covered. All the different methods will be covered, while there will be a more specific focus on the method of applying artificial intelligence methods to developing proxy models.

2.3.1 Reduced physics models

The numerical simulator models in use today are practical as they take a lot of the real physics into account, giving them a lot of analytical credibility. Numerical reservoir simulators use complicated formulas that are developed specifically to mimic the physics happening in the reservoir to calculate how these processes affect the fluid flow system incrementally. As these calculations are being done at each timestep and for each grid cell, of which there could be millions, the process of doing just one run of a numerical simulation may take a long time. One approach to try to shorten this time is by developing what is known as a *reduced physics model*, which is a simpler model that is taking fewer physical interaction and grid cells into account compared to a full-physics model [45]. A reduced physics model will still be investigated using a numerical setup.

Some reservoirs will be more complicated than others. A reservoir may have fractures which must be taken into consideration when modelling the flow of fluids through the system, the oil may be extremely viscous causing conventional methods to be less effective, or the reservoir rock may be an impermeable rock such as shale. A lot of the available gas reservoirs in the US are shale gas reservoirs, which often require the operating company to perform hydraulic fracturing to stimulate flow, and this has to be modelled. Modelling all of these additional complexities may give accurate results when performing the numerical simulation, however it may become quite time consuming to run the simulations. Thus these kinds of reservoirs are often considered quite suitable for a reduced physics proxy-model. The extra accuracy from performing a full physics simulation is redundant when compared to the increase in the amount of time the simulation will spend on numerical calculations compared to a proxy modelling approach.

K.Wilson & L-Durlofsky (2012)[45] wrote a paper regarding the application of a reduced physics model for the purpose of doing a computational optimization study of shale gas production in a field in the US. Gas shale reservoirs in North America are often considered complex. The best method of producing this gas is by conducting a series of hydraulic fracturing procedures on the shaly formation due to the inherent low permeability of shale. A numerical simulation model of a shale gas reservoir has to be able to handle simulation of complex matrix flow and flow through fractures, gas desorption, non-Darcy flow effects, and stress dependant permeability zones. Gas desorption refers to gas that is absorbed by the rock surface that may be released during interaction with the reservoir.

The full physics model had a grid size of 106x53x1 grid cells, and included high resolution fracture networks, dual porosity, dual permeability, and gas desorption. A local grid refinement was set up around the stimulated zone, as well as around the primary and secondary fractures. Thus the total amount of grid cells will vary based on how the experiment is set up, but generally staying within 50,000 - 250,000 grid cells. Simplifications are made to the full-physics model to develop a reduced physics proxy model, such that the proxy model will approximate the reservoir responses with a similar accuracy as the numerical simulator in a fraction of the time. For this study it was found that a single-porosity system without desorption or any local grid refinement provided a good approximation of the full-physics model, and instead of explicitly modelling fractures each fracture was modelled as an additional perforation along the wellbore in the stimulated region of the reduced physics model. The simplification resulted in a proxy model consisting of 5,618 grid cells, which could be more than 10 times less than the full-physics model. For long wells the full physics model took about 2100 seconds to run one simulation, and for short wells it took 650 seconds. The reduced physics model took 4 seconds for both scenarios, with an error of less than 5%.

Reducing the complexity of a reservoir model will result in a faster simulation compared to one that has all the physics included in the simulation. It is applicable to systems that may include a complicated geological situation such as fractures and impermeable zones, and also in fields where unconventional methods are applied to extract the oil. The method may not be regarded as a universal method, as there seems to be a reservoir-specific proxy development method. Overall, the method of reduced physics is a good approach to a gas shale situation considering the complexity of the physics with regard to induced hydraulic fracturing.

2.3.2 Statistical method

Statistical proxy models are often referred to as Response Surface Models (RSM) [3]. These proxy models are based on statistical or mathematical functions which are able to replicate a desired input-output relationship. These methods generally perform quite well at modelling tasks and sensitivity analysis of complex system which has a lot of parameters.The technique of using a statistical approach to model the flow in a subsurface reservoir has been proven to be quite successful, as this is a complex system with a lot of interaction.

A RSM requires a lot of data in order to be properly set up, thus it is important to perform proper *design of experiments* (**DoE**). DoE is a method of planning how to perform the sampling of the data, by assessing which parameters are essential in order for the model to perform as expected. B.Yeten et al. [46] defined a workflow for the development of a RSM using statistical methods, which they used as a basis for a comparative study on various approaches. The approach is as follows:

1. Define a set of key parameters, and their probability distributions.
2. Perform a low level experimental design study
3. Run simulations corresponding to the experiments in the previous point.
4. Fit the estimated results to a simple response surface
5. Use the parameter probability distributions and run Monte Carlo simulations in the response surfaces
6. Generate a plot or diagram of the sensitivity of each of the parameters
7. Screen the "heavy hitters". Heavy hitters is referring to the parameter combinations that have a larger impact on the model responses.
8. Perform a more detailed DoE on these parameters
9. Repeat steps 3 & 4
10. Run the Monte Carlo simulation again on these new response surfaces

The workflow mentions a method known as a Monte Carlo simulation. Monte Carlo simulation is a term describing stochastic simulations that incorporate random variability into a model [6]. The term "Monte Carlo" refers to the famous casino in the sovereign state of Monaco, which was arbitrarily picked as a name by Ullam and Von Neumann [6] when they applied this method while working on the Manhattan project. This method of probability modelling is a popular technique which is used commonly within almost every sector of industry, health care, and science. To conduct a Monte Carlo simulation one must first define all input parameters and determine a probability distribution for each of them, and this must be done for any parameter which may carry uncertainty to the model. Then the probability simulation is run on the parameters, and the process is able to calculate a possibility distribution of the different possible outcomes. This method differs from deterministic simulations in that the variability of parameters is taken into consideration.

Kriging, also known as Gaussian process modelling[27], is a method of developing SRMs which is also based on least squares mean regression which can easily be adapted to handle multiple dimensions and some arbitrarily based points in space. Kriging assumes that all spacial points are correlated to each other through a co-variance function, or a variogram model, γ. This co-variance function can be given as

$$\gamma(h) = Ch^{\omega} \tag{2.4}$$

, where h is the distance between two sampled points, C is a positive slope, and ω is a power which is between 0 and 2. This function will be able to correlate the spacial variation between sampled and unsampled points, and provide some weight to each sampled point. The more spatially correlated a point is to a target location the bigger the weight is. Kriging is a data-exact modelling technique. One issue with kriging is that it is not that suitable for using when there are a lot of observations that have to be implemented in the model , as if there are N different observations the kriging based SRM requires an inversion matrix of (N+1) x (N+1).

Artificial neural networks is also a method of developing a response surface model, as the neural network proxy will generate some model which approximates input-output relationships based on a large amount of input data for several parameters. The implementation of artificial neural networks is still different, and thus this topic will be covered in more detail later.

The application of a statistically based proxy model is relatively new within the petroleum industry, but the method of setting up some probability distribution for a physical phenomenon and then performing a Monte Carlo simulation has been used for decades in other sectors of global industry. Application within the oil and gas industry also shows the promise of performing such models to the simulation of multiphase flow in a subsurface flow system.

2.3.3 Reduced order method

A reduced order model is a proxy model which approximates the responses of a modelled system by reducing the high dimensionality of the model. A model is regarded as high-dimensional depending on the amount of parameters that go into producing the model responses. As a model has more parameters involved there are more correlations that have to be developed as well, as many of these parameters interact with one another. An oversimplified way of looking at a reduced order model is that a sensitivity analysis is conducted on the involved parameters in order to identify the parameters that has the most significant impact on the model responses, and then try to perform a simulation by only involving the calculations based on these significant parameters. This new parameter space that is spanning over these significant parameters will have a lower dimension than the full parameters space, and thus all the involved calculations will be faster in the reduced order parameter space as opposed to the original high dimensional space.

Reduced order modelling has been performed on a lot of engineering issues often related to simulation, classification, visualization, and the compression of data [3]. Its use in the petroleum industry is relatively new however, and in these applications the reduced order models are often based on Krylov subspace, balanced truncation, and proper orthogonal decomposition (POD) [21]. A common denominator between these methods is the transformation of a higher dimensional parameter space into a lower dimensional subspace. The more common method of these is POD, as this is generally quite consistent for solving problems which include highly non-linear system such as subsurface multi-phase flow. It is also common to add some modifications to the traditional POD scheme to get a better performance, and these modifications are the POD-TPWIL (Trajectory PieceWise Linearization), and POD-DIEM (Discrete Empirical Interpolation Method).

2.3.4 Artificial Intelligence method

Workflow related to reservoir engineering tend to be based around data management and analysis. Real data is collected from the field for the purpose of analysing the state of the hydrocarbon reservoir and to realize how the reservoir is responding to both the production and to a variety of induced tests. Using this knowledge, the operating company will be able to decide what production strategy to perform moving forwards in order to produce the largest amount of hydrocarbons. This data can be collected from pressure measurements in the wellbore, fluid samples from the formation, core samples from the formation, and more. All of this data is then interpreted by geologists or engineers in order to produce more data. As effective as these methods are in terms of determining the best way to move forward, the process is quite labour intensive and thus also time consuming.

The idea of implementing artificial intelligence to solve some of these issues has been talked about in the industry since the early 1990s [24], and has in recent years seen a lot more use. Artificial intelligence(AI) can be used to set up a proxy model by utilizing the concept within AI known as artificial neural networks, or ANNs. These are computer based entities which are designed to solve specified tasks based on a methodology that is inspired by how a biological brain works, by sending data through a highly convoluted network of nodes, known as neurons. As this network is supplied with data, the data will be sent through the nodes in the network and ultimately produce some output. The concept will be described in more detail in the next section. The core concept is that this method is able to handle a lot of data and can be used to perform highly time consuming tasks in a shorter amount of time compared to the traditional methods.

The ANN method is regarded as data driven, meaning that the neural networks performance is strongly dependent on the amount and quality of data that will be provided to it [3]. The network will learn how to deal with the specific task based on the provided data, and thus it is important that the supplied data is of a high quality. When supplied with data, such a neural network will look at all the input data and then produce some output data, and based on how good this prediction is the internal structure of the networks alters slightly so that the next time it is provided data the prediction will be of a higher quality. Proxy models that are developed using ANNs will not be *data-exact*, which means that an ANN will use the supplied data to form correlations rather than mimic the provided data.

Using artificial intelligence to set up a proxy model has a lot benefits, such as a relatively short development time, fairly low computational time, it doesn't require a powerful computer to efficiently generate the proxy model, and the model stays true to the natural data that is fed into the model [10]. The use of artificial intelligence in the petroleum industry can be used to make efficient proxy models [3], estimating the permeability of a reservoir based on resistivity measurements [19], and determine the optimal well placements [4]. The field of AI is relatively well established, but it is by no means considered an industry standard approach in the oil and gas industry.

2.4 Artificial intelligence and machine learning

Artificial intelligence (**AI**) is a broad umbrella term, generally referring to technology within computers that is able to mimic the human mind in terms of decision making and problem solving. An important part of AI is machine learning, which is the automated process of computers recognising patterns in data. The process of developing these machine learning algorithms differs quite a lot from traditional programming, as the functions themselves are not explicitly created by the programmer, but are instead generated as the machine learning software is supplied with large amounts of data and then they create correlation within the data dynamically.

Applied artificial intelligence has been shown to be capable of completing a wide variety of tasks ranging from image recognition, fraud prevention in banks, and tasks related to data processing[24]. One common factor in a lot of these applications is that these tasks are often viewed as human tasks, meaning that they require some human trait such as reason, experience, or intuition in order to solve the problem. One example of computers outperforming humans is chess, where the AlphaZero algorithm is able to perform at a chess rating(ELO) of well over 3000 with only 4 hours of training solely by reinforced learning from playing against itself [35]. ELO is a rating system which provides a relative measure of how good a player is relative to other players. A ELO rating of 3000 means that this machine will in theory outperform most humans as the top humans in the world sit at around 2800-2900 ELO. This example is of course trivial in terms of engineering, but in terms of artificial intelligence it is a good example in terms of the promise of machine learning algorithms.

There are a couple of ways to set up a machine learning algorithm [26] . The first method is known as *supervised learning*. This is a method in which the machine learning algorithm is supplied with a dataset containing both input data and the desired output data. The algorithm will then take in this input data, run it through some internal processes, and then produce output data. After the algorithms makes a prediction this predicted output will then be compared with the supplied desired output. If the prediction is wrong in comparison to the desired output, the machine learning algorithm makes corrections to the internal processes before running it again. This learning mechanism is known as backpropagation, and it is based on minimizing a cost function,C_0. This cost function is often proportional to the error between the output produced from the machine learning algorithm and the desired output. Figure 2.2 shows a schematic of how the process works.

When conducting supervised learning it is often a good idea to partition the dataset into three different sets of data. These sets are:

1. Training dataset
2. Validation dataset
3. Testing dataset

The purpose of having multiple separate datasets is to improve the learning efficiency of the machine. Computers are quite capable at memorizing data perfectly, and to prevent the algorithm from simply memorizing all the outputs from the learning data it is important to introduce it to some fresh data. With this split the machine learning algorithms are first shown the learning data, which may tend to be up to

90% of the total data. After running through the learning data, the algorithm is exposed to the validation dataset in order to check whether it has just memorized the learning dataset or not. Every iteration where the algorithm sees the learning data and validation data is known as an *epoch*, and generally the learning process is performed based on a pre-determined number of epochs or until an acceptable accuracy on the validation data is reached. Lastly, the algorithm is exposed to the test set, where it is common practice to calculate the accuracy of the predicted data output from the machine learning algorithm. This sort of learning method tends to be quite suitable to problems such as image recognition, categorizations, and general classification problems. More specifically in the oil- and gas industry this sort of learning algorithms have been used in classifying lithologies given well log data and conducting history matching [19], [13].

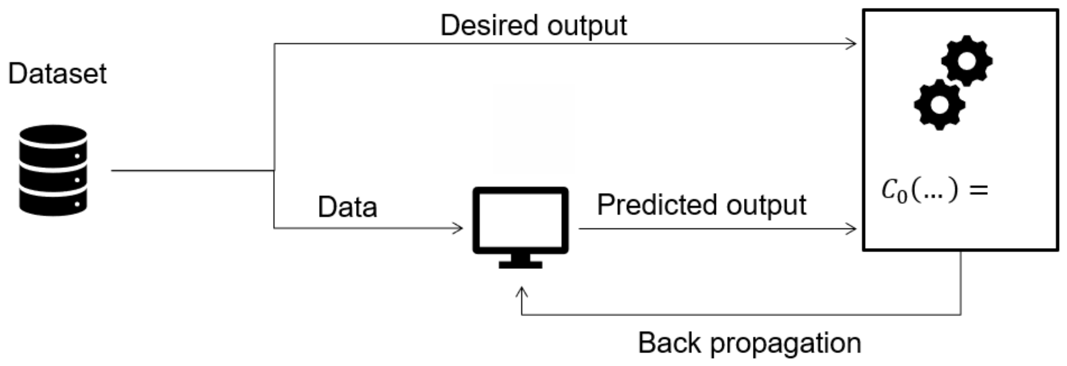

FIGURE 2.2: General schematic illustrating the workflow in a supervised learning setup

Another type of machine learning is *unsupervised learning*(figure 2.3). In this method of machine learning the algorithm is only provided with input data. As with the supervised learning the machine learning algorithm will use this input data to produce an output, and this output will then be inserted into a function that calculates the cost of the output. The cost of the output determines how good the output was. The cost function is chosen based on what task the machine learning algorithm is meant to be learning. If the algorithm has a better cost it will be used to develop newer algorithms, which hopefully will provide even better cost scores. In theory the cost score will only get better and better as this sort of learning process goes on, and thus this type of learning is quite suitable to optimization tasks.

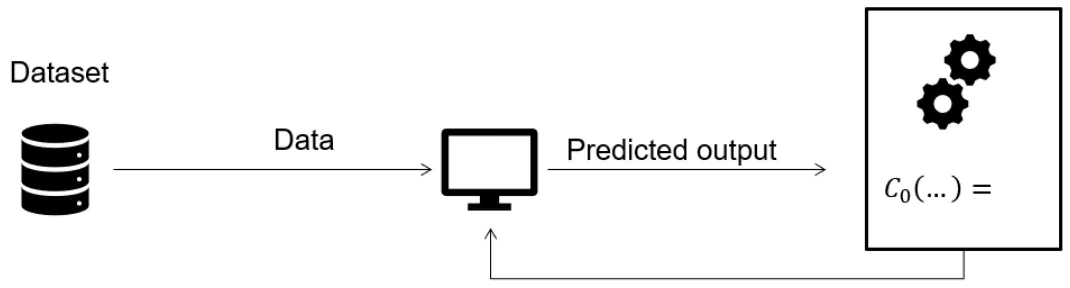

FIGURE 2.3: The general workflow of unsupervised learning

Reinforced learning(figure 2.4) is the third of the fundamental machine learning algorithm methods. It saw an increase in popularity in the 1990s, which is some time after machine learning as a whole started its meteoric rise in demand, which was during the 1940s and 50s [18]. Reinforced learning is appealing as the algorithms are learning without the programmer needing to specify how to perform better at the task. This means that this method of learning doesn't require a dataset to learn, which is desirable since creating a dataset is one of the most time consuming tasks when conducting machine learning. The method is based on placing an agent in a dynamic environment where the agent must learn behaviour through trial and error. The programmer specifies a reward function that will give the agent positive reinforcements if this reward function is activated, and negative reinforcement if not. As the agent is exposed to this environment it learns through genetic algorithms and statistical techniques to estimate the value gained. Over time the agent tries to optimize its behaviour so that it will receive the most amount of reward. One downside of this sort of machine learning is that it generally takes a long time for the machine learning model to become efficient as dynamic environments can be complex.

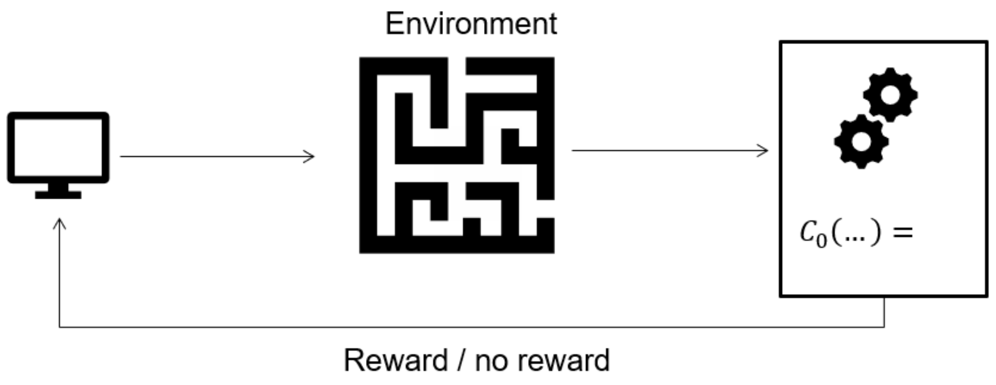

FIGURE 2.4: The workflow of a reinforced learning setup

2.5 Latin hypercube sampling

Latin hypercube sampling (**LHS**) is a popular and powerful sampling scheme that is used in a multitude of different disciplines such as science, engineering, and mathematics. [34]

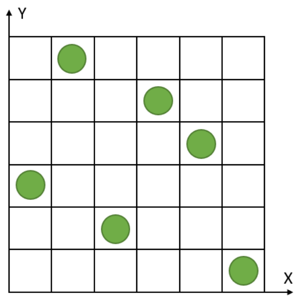

FIGURE 2.5: A 6 point latin square

A latin hypercube is an extension of the combinatoric term *latin square*. A latin square of an arbitrary order t is an arrangement of t values in a t x t sized array in such a way that each value occurs only once in each row and once in every column [5]. A *latin hypercube* is an expansion of this concept, where a latin square setup is applied to an arbitrary amount of dimensions. Figure 2.5 shows a latin square, where it is clear that each point is the only point occupying that space in both the x- and y-direction.

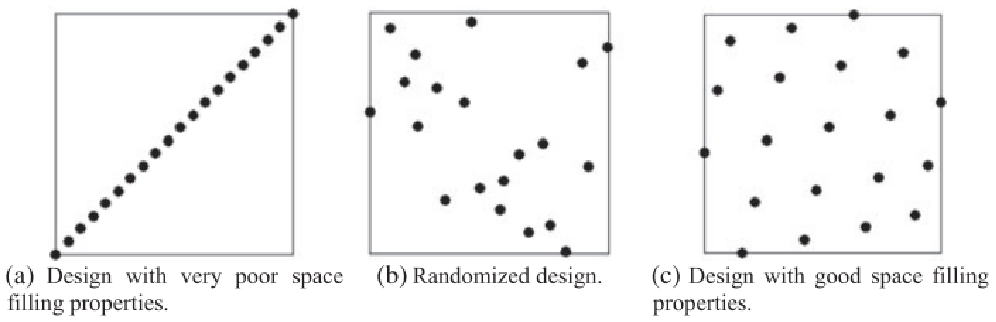

FIGURE 2.6: The space filling of linear, random, and LHS sampling, [43]

LHS is a form of stratified sampling, which is a larger concept within data sampling. It can be viewed as a compromise between a random sampling and stratified sampling, which incorporates some of the features from both methods [16]. It is important that the data sampling is done in such a way that as much of the parameter space as possible is covered. This can be achieved by random sampling, as this tends to have a decent spread across the parameter space. Figure 2.6 shows an example of linear- , randomized-, and LHS sampling design, and how these sampling methods affect the sampled space. The linear approach has a poor space filling, while the random has the potential to be a good space filling. The LHS-design is overall the most balanced approach. LHS can be viewed as a random sampling with structured rules, stating that the parameter space is divided into equal parts and a random sample is taken from each of the equal parts. This yields an overall better coverage of the parameter space compared to a linear design or a random design [43].

2.6 Artificial neural networks

Artificial neural networks is a biologically inspired program paradigm, which is focused on providing an artificial network of neurons with large amounts of data in order for the network to learn how to perform specified tasks. It is based on how the biological brain works, where the activation of one neuron will trigger the activation of other neurons[24]. The network will complete a series of mathematical operations which generates some correlation between the input variables and output variables.

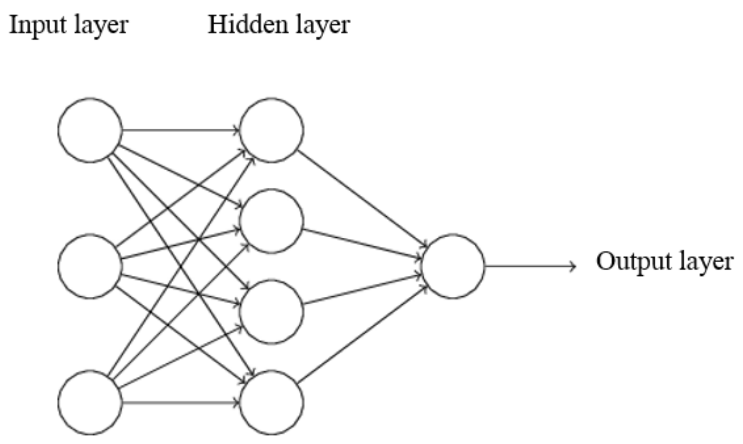

FIGURE 2.7: The input-, hidden-, and output layers of a neural network. [26]

In the sense of ANN a neuron can be viewed as a node which is holding a value, usually between zero and one. Figure 2.7 illustrates the setup of a neural network. A neural network is set up as a series of such neurons. Each column of neurons is referred to as a layer. As seen in the figure, every neuron is connected to the neurons in the next layer by a connection. This connection decides the contribution from one neuron to the connected neurons activation [26].

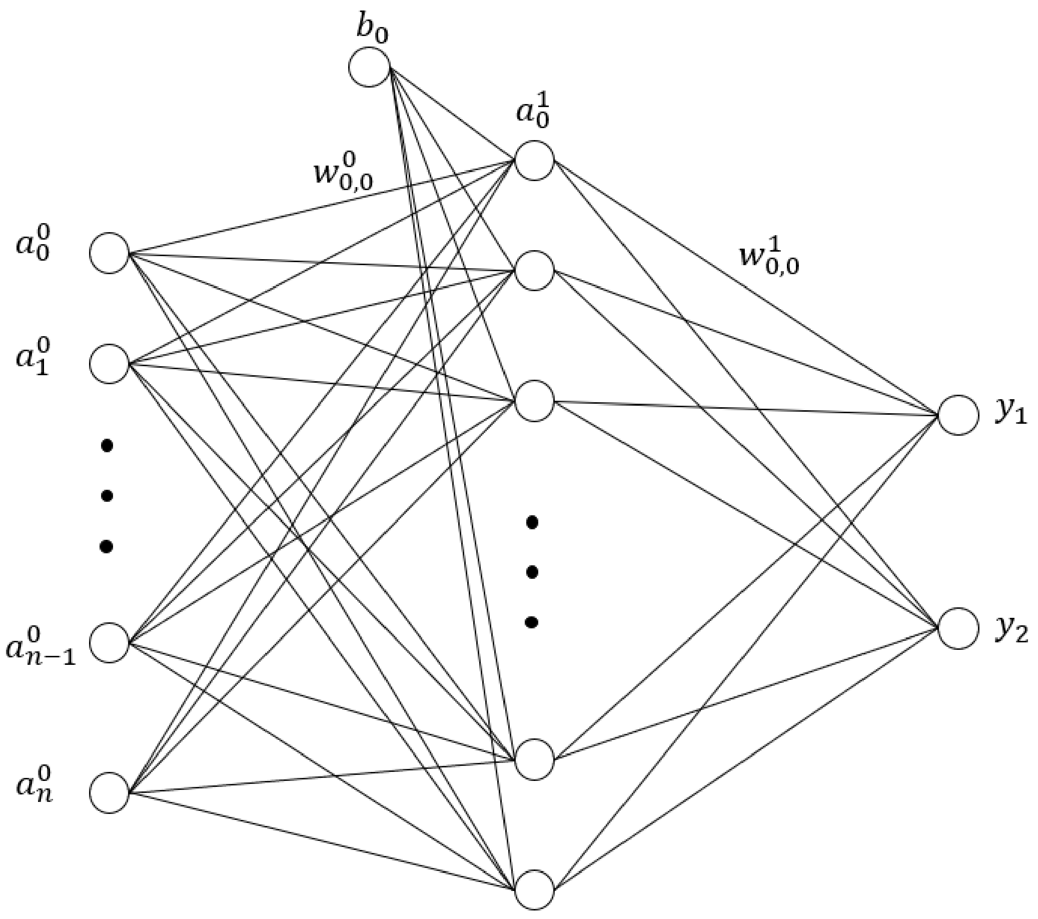

FIGURE 2.8: A neural network with labeled nodes

For the purpose of explanation, the following equations will refer to nodes and layers as seen in figure 2.8. The activation value of one neuron is computed as a weighted sum of all the activations from the previous layers.

$$a^L = h(\sum_{i=0}^{N} x_i^{L-1} w_{i,j}^{L-1} + b) \tag{2.5}$$

where a_n^L refers to the n-th node in layer L, x_i^L refers to the value of the i-th node in layer L, $w_{i,j}$ refers to the weight from the i-th node in the next layer to the j-th node in layer L-1, and b^L refers to a bias added to the calculations in layer L. h(.) is known as the activation function of the node. The activation a node in layer L, a_0^L, will be calculated as such

$$a_0^L = w_{0,0} a_0^{L-1} + w_{0,1} a_1^{L-1} + ... + w_{0,n} a_n^{L-1} + b_0 \tag{2.6}$$

One method of simplifying the calculation is to imagine that all of the activations in layer is a part of one vector of activations, the weights between two layers is a matrix of operator, the activations in a layer is a vector of activations, and so on.

This provides an initial activation vector of

$$a^L = \begin{bmatrix} a_0^L \\ a_1^L \\ \vdots \\ a_n^L \end{bmatrix} \tag{2.7}$$

and a bias vector, which will look similar. Subsequently, the weights can be set up in a matrix structure as such

$$W = \begin{bmatrix} w_{0,0} & w_{0,1} & ... & w_{0,n} \\ w_{1,0} & w_{1,1} & ... & w_{1,n} \\ ... & ... & \ddots & \vdots \\ w_{k,0} & w_{k,1} & ... & w_{k,n} \end{bmatrix} \tag{2.8}$$

Now that the activation vector and the weight matrix are defined, the calculations for all the activations of layer L can be written as a linear algebraic equation, as such

$$a^L = Wa^{L-1} + b \tag{2.9}$$

$$\begin{bmatrix} a_0^L \\ a_1^L \\ \vdots \\ a_n^L \end{bmatrix} = \begin{bmatrix} w_{0,0} & w_{0,1} & ... & w_{0,n} \\ w_{1,0} & w_{1,1} & ... & w_{1,n} \\ ... & ... & \ddots & \vdots \\ w_{k,0} & w_{k,1} & ... & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{L-1} \\ a_1^{L-1} \\ \vdots \\ a_n^{L-1} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \tag{2.10}$$

This activation value is then passed through a nonlinear activation function, providing the following equation

$$a^L = h(Wa^{L-1} + b) \tag{2.11}$$

To simplify this notation, Z^L can be defined as

$$Z^L = Wa^{L-1} + b \tag{2.12}$$

such that the activation for layer L can be simplified as

$$a^L = h(Z^L) \tag{2.13}$$

By passing the weighted sum through a nonlinear activation function h(), the system will be able to exhibit nonlinear behaviour. An activation function seeks to normalize the output of the node, causing the node to replicate the functions of a biological neuron in that the neuron is either activated or deactivated. The non-linearity of the activation function is essential. This non-linearity makes it possible to use the learning process of backpropagation.

One of the most popular activation functions is known as the sigmoid function:

$$h(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.14}$$

The sigmoid function will take the value of the weighted sum, and squish it between a value of 0 and 1. This is going back to the analogy of trying to imitate how a biological brain works, by having the neurons either active, inactive, or somewhere in between. In more recent times the sigmoid functions have become less used within neural networks as it comes with a couple of issues. One issue is that for large positive or negative values of z it will grant a small gradient, as the sigmoid function curve becomes parallel with the x-axis at large values. The gradient of the activation function will be used in the learning process of the network, and thus if the gradient is too small the network might struggle to learn, or not be able to learn at all.

The hyperbolic tangent function is another function which is commonly used.

$$h(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.15}$$

This function is quite similar to the sigmoid function, with the distinction that the *tanh*-function squishes the output into a range between -1 and 1. Other than this, the tanh-function has the same advantages and disadvantages of the sigmoid.

One of the more computationally efficient functions in use today is the Rectified Linear unit function, otherwise known as ReLu, which can be expressed as

$$h(z) = max\{0, z\} \tag{2.16}$$

This function, as well as the many variations of this function, is understandably desired as the efficiency in the calculation of the functions causes a neural network to converge quicker than with other less efficient functions. Even though the function looks linear, it has a derivative that suits backpropagation quite well. As the input values gets closer to zero or are negative, the gradient of the ReLu-functions becomes zero which may cause the network not to learn at all. To address, this one could alternatively use the so called leaky-ReLu, which is a modification which causes the gradient to never be zero. Compared to the regular ReLu-function this modified version is not as consistent. One quite potent alternative to the ReLu-function is the Swish-function [31], which acts a lot like the ReLu-function does while it doesn't have a gradient equals to zero.

The swish-function is defined as

$$h(z) = z\sigma(z) \tag{2.17}$$

where $\sigma(z)$ is the sigmoid function. Like the ReLu-funciton this swish function is unbounded for high values, and bounded close to zero for low values. Figure 2.9 shows how the mentioned activation functions looks like. The previously mentioned attributes are clearly shown when the functions are plottet in such a way, like the sigmoid is clearly between 0 and 1 and the gradient will clearly be larger for values that fall closer to 0, and the tanh acts similarly to the sigmoid but for values between -1 and 1. The swish and ReLu will remain low or zero for values less than 0, and then monotonically increase for larger values.

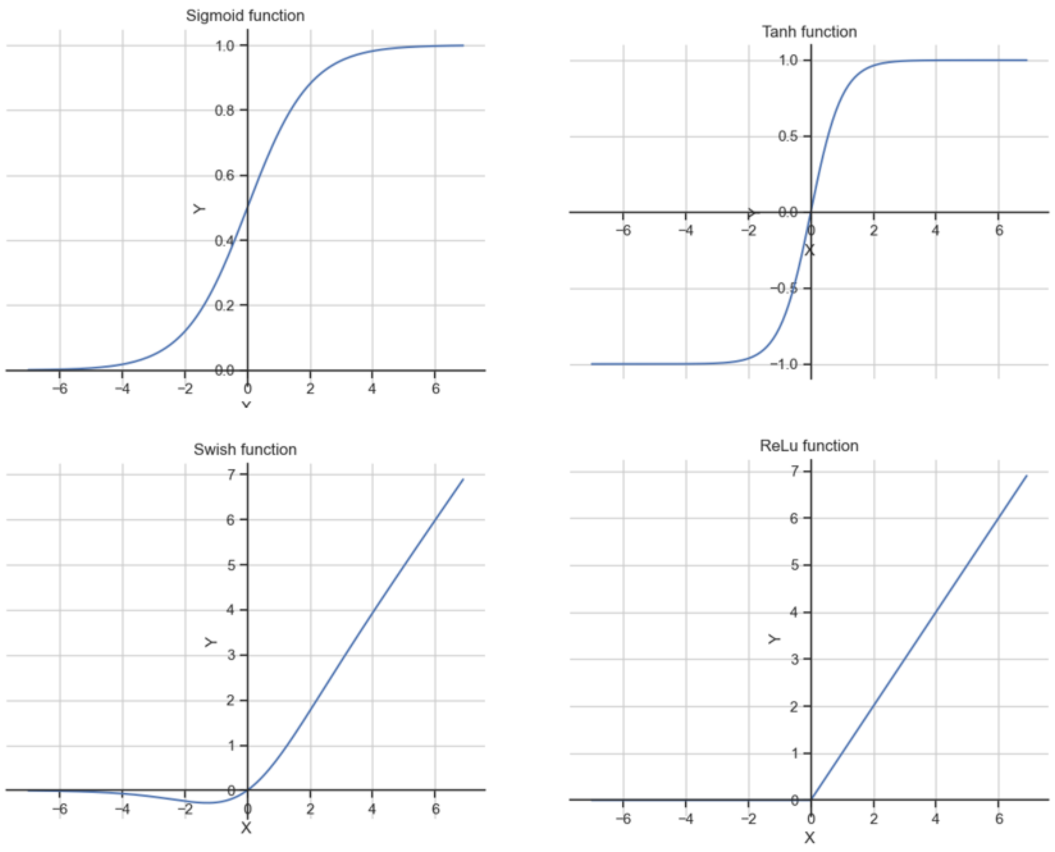

FIGURE 2.9: The plots in this figure show how some of the most common activation functions look like

It is important to assess how well the network performs. Every neuron is connected to every previous neuron and the weights connecting the neurons decides the activation of the active neuron, which ultimately affects how the network will perform at the desired task. To properly train the function it is an effective approach to determine a *cost function*. A cost function is simply a function used to calculate how wrong the prediction from the neural network is in terms of the desired output. One typical method of calculating this error is by using a sum of squares, where the error between the desired output and the predicted output is known as the cost of the training example. The cost function can be defined as

$$C_j = \sum_{j=0}^{n_L-1} (a_j^L - y_j)^2 \tag{2.18}$$

where C_j is the cost function for sample j, a_j^L is the predicted output for sample j, and y_j is the desired output for sample j. n_L is the total number of layers. The cost of the training examples will likely be higher at the beginning of the iterative process of leaning, and the goal of the machine learning process is to get this cost function to provide as low values as possible. The better the prediction, the lower the cost function will be. The cost function is often minimized by finding the steepest descent of the cost function. This is done by calculating the gradient of the cost function using the whole set of training examples, then taking a small step in the direction of the gradient, and then repeat this over and over again until the cost function gradient becomes lower and lower. The gradient of the cost function will decide how much to change each of the weights and biases connecting the neural network.

Backpropagation is a common method used to train a neural network. It is a reliable method within the sector of neural networks. The gradient of the cost function will be a function of all the different weights and biases, calculated by averaging the cost of every training example provided. The goal of backpropagation is to figure out how sensitive the cost function is to changes between the various variables involved in calculating the prediction for each training example.

By defining the derivative of the cost function with respects to the changeable parameters, which are the weights and biases, it is possible to find out how much these parameters affect the cost function gradient. This determines how much to change these parameters for the given iteration. The derivative of the cost function can be written as $\frac{\partial C_o}{w_{jk}^L}$. By applying the chain rule this derivative can be expressed as such

$$\frac{\partial C_0}{\partial w_{jk}^L} = \frac{\partial Z_j^L}{\partial w_{jk}^L} \frac{\partial a_j^L}{\partial Z_j^L} \frac{\partial C_0}{\partial a_j^L} \tag{2.19}$$

As seen in this formula, due to the introduction of Z^L as a representation of the calculation of the next activation layer this chain rule derivative proves quite possible to solve. The derivative of this Z^L term over the weight is

$$\frac{\partial Z_j^L}{\partial w_{jk}^L} = a_j^{L-1} \tag{2.20}$$

The derivative of the activation layer over the Z^L term is

$$\frac{\partial a_j^L}{\partial Z_j^L} = h'(Z_j^L) \tag{2.21}$$

And lastly, the derivative of the cost function over the current activation layer is

$$\frac{\partial C_0}{\partial a_j^L} = 2(a_j^L - y) \tag{2.22}$$

Thus the derivative of the cost function over the weights for layer L is given as

$$\frac{\partial C_0}{\partial w_{jk}^L} = a_j^{L-1} h'(Z_j^L) 2(a_j^L - y) \tag{2.23}$$

This derivative is an indication of how sensitive the cost function is to changes made to the weights of the system. Note that this derivative is only for one training example. In order to incorporate the influence from all of the training examples in the provided dataset one could take the average of all the different cost function sensitivities as such

$$\frac{\partial C}{\partial w^L} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{w^L} \tag{2.24}$$

The expression sensitivity in terms of small changes to the bias is fairly similar to this one, and by using the chain rule it can be defined as such

$$\frac{\partial C_0}{\partial b_j^L} = \frac{\partial Z_j^L}{\partial b_j^L} \frac{\partial a_j^L}{\partial Z_j^L} \frac{\partial C_0}{\partial a_j^L} = h'(Z_j^L) 2(a_j^L - y) \tag{2.25}$$

These equations are used to calculate how sensitive the cost function is to changes to the weights and biases in the current layer, and the next step is to move, or propagate, backwards in the system in order to make sure that all of the weights and biases in the network are taken into consideration when calculating the gradient of the cost function. This is the reason why the method is known as backpropagation. In order to do this there need to be an expression for the sensitivity of the cost function with respect to changes in the activation function for the previous layer, which can be done by using the following expression

$$\frac{\partial C_0}{\partial a_j^{L-1}} = \sum_{j=0}^{n_L - 1} \frac{\partial Z_j^L}{\partial a_j^{L-1}} \frac{\partial a_j^L}{\partial Z_j^L} \frac{\partial C_0}{\partial a_j^L} = w_{jk}^L h'(Z_j^L) 2(a_j^L - y) \tag{2.26}$$

Even though this activation for layer $L - 1$ cannot be directly influenced by the gradient of the cost function, this derivative can be used to calculate the sensitivities of all the weights and biases in all the previous layers of the neural network.

2.7 Evolutionary computing

Artificial neural networks is just one method where a biological process is being replicated in the area of computer science. Another concept is known as *evolutionary computing*. The *evolution* in this term comes from the attempt to mimic biological evolutionary processes in a computer algorithm. The overall concept of evolutionary computing follows a workflow that may look like this [14]

1. Initialize a population of solution
2. Calculate the fitness of each individual of the population

3. Select specific well performing individuals from the population as parents for the next generation
4. Generate a new population based on the previously selected individuals
5. Perform other evolutionary processes on the new population, such as mutation
6. Return to point 2 until a stopping criteria has been met

In the context of biological creatures the concept of fitness will be related to how well adapted the creatures are to their environment. Within computer science it is rather a question of how well the population score on an objective function, which is set based on the optimization problem in question.

One can view the natural evolution as an optimization problem [24]. In nature, a creature is dependant on food to survive, it needs to reproduce, and it has to overcome both the harshness of the nature in it's environment, and also need to be able to evade any potential predators. As the generations go on these creatures will eventually become better adapted to their environment. As a generation is exposed to the environment, only the individuals that are able to fulfil the previous challenges are able to pass on their genes to the next generation of creatures. Because this new generation of creatures inherit their genes from only the most capable from the previous generation, this new generation will on average be more capable of finding food, finding a mate, and evading all potential dangers.

The challenges mentioned in the previous example could be defined as a problem. The problem is that the creature needs to survive, and it survives by getting food and evading danger. The ultimate goal of the survival is to reproduce, such as to keep the species of the creature from going extinct. The factors that go into this survival problem is a creature's ability to find the food, a creature's desirability towards other creatures of that species, and the creature's abilities to evade potential dangers. To put it differently; there is an optimization problem of survival, and the input variables are the individual creatures abilities to deal with issues surrounding the problem. As the population of creatures go from one generation to the next, the overall abilities included as input parameters will be better, which in theory will lead to more creatures surviving each generation. This is the core concept of evolutionary computing.

One such nature inspired optimization algorithms is known as *ant colony optimization*(ACO)[14] . As the name suggests this is a method that is inspired from ants. This is a metaheuristic optimization algorithm. The original idea comes from the observation of ants exploitation of resources and food. Even with the ants limited cognitive ability they are able to communicate effectively which path to take in order to find the shortest path towards a food source. This is all done without any high level form of communication. Initially, ants will explore the nearby area seemingly random to search for food. If an ant finds food, it picks up some and brings it back to the hive. On the way back the ant will leave behind a trail of pheromones, which will indicate to other ants that this path will lead to food. The amount of pheromones that are deposited depends on the quality of food and the distance towards the source. Still, the next time an ant leaves in search for food they are more likely to follow a pheromone path than to simply walk a random path. Over time, the shortest path towards the food source should be the path with the strongest concentration of pheromones, as more and more ants have taken this path in search of food. This concept can be seen in figure 2.10.

ACO is a digital approximation of this core concept [14]. It has been conducted for digital domains on both discrete scenarios and also for continuous domains. ACO has a wide range of uses, and has been used for the optimization of water-flooding processes [30]. As multi-phase flow through a real, heterogeneous reservoir is a complex process it is a good match with ACO, which provides a good solution to an otherwise difficult optimization venture.

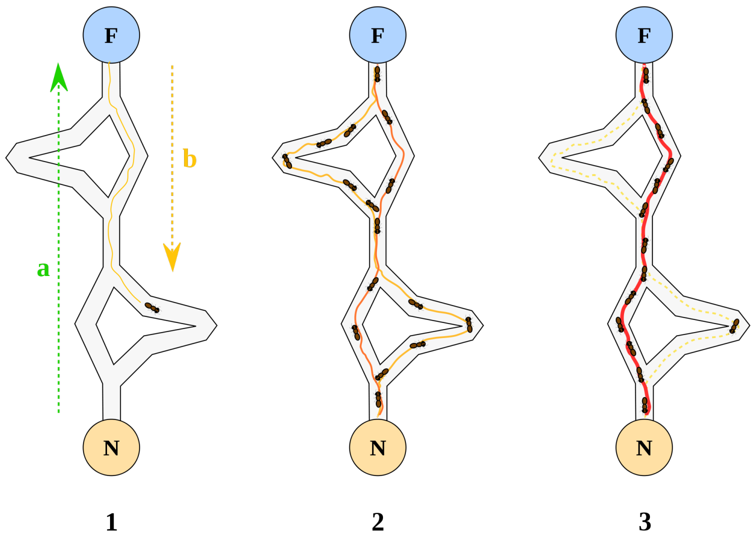

FIGURE 2.10: Illustration of Ant Colony Optimization. The bottom node is the ant nest, and the food source is the top node labeled F[40].

Another common method is known as *particle swarm optimization*(PSO) [14] . This algorithm is inspired by how several species of animals will act as a seemingly disorganized collection of individuals that tend to move in large clusters, such as fish, insects, or birds. The algorithm attempts to mimic the movements and foraging patterns that are used by these clusters of individuality. For example: a bird doesn't know which way is the best to go in search of food, but if it sees a flock of other bird flying towards a tree it will be inclined to follow the rest of the flock. Or if the bird knows that it will usually get some food by approaching the benches in the park, it will most likely fly towards the park. So an individual in a swarm will make a choice based on their own experiences, as well as based on what the swarm as a collective is doing.

When moving it is assumed that the direction that an individual will move will be based by factor of three components:

- A physical component, which means that this individual tends to follow the current direction that it is moving
- A social component, which means that the individual will move towards the best location visited by any other individuals in the swarm.
- A cognitive component, which means that the individual will move towards the best location that it has visited itself

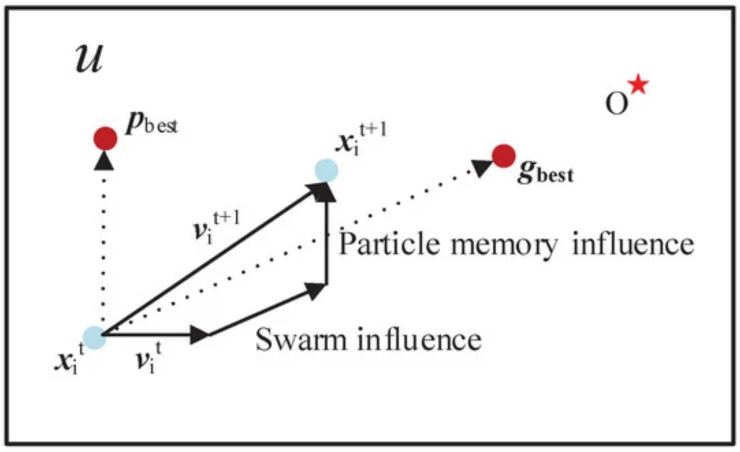

FIGURE 2.11: Illustration showing how the three factors correlate to the overall movement of a particle in PSO. [47]

These directions can be represented as a vector, and then a new vector is being made based on these three other vectors. How the different factors affect the overall movement vector can be seen in figure 2.11. Note that the overall movement vector v_i^{t+1} is a combination of the initial path the individual was taking, as well as the individuals best location p_{best} and the swarms best location g_{best}. In a mathematical sense the best locations will be the locations where the best solution to an objective function has been reached. As the algorithm moves on, it will eventually converge towards a minimum value for the objective function. As the method is using a swarm of different initial solutions, the odds of the algorithms being able to find the absolute minimum is quite good. Even if a couple of individuals in the swarm find a local minimum they will be more inclined to move towards the global minimum, assuming that another particle has reached it already.

2.7.1 Genetic algorithm

Genetic algorithm(GA) is a meta-heuristic optimization procedure that is heavily inspired by natural selection. In nature the animals that are most able to adapt to changes in the environment will survive, while the animals that aren't able to cope will falter. The survivors are then able to pass on their genes to the next generation of animals, and the characteristics that made the animals survive will be passed on to the new generation [25]. This concept is a generalization of the natural evolution happening in nature, and the concept that is emulated in a genetic algorithm. Genetic algorithms use this adaptation of natural selection to achieve an optimized solution, and thus GA is a derivative free optimization algorithm. Over time, a GA is capable of maximizing or minimizing a given function. The plot in figure 2.12 shows a minimization of a function, and clearly shows how the function evaluation is minimized over time.

The optimization in a genetic algorithm setup is based on having an initial population of solutions, and then gradually evolve the population. The population will consist of an arbitrary amount of individuals, which are characterized by their chromosome. This chromosome will be practically a set of values which directly influences the *fitness* of each individual, where *fitness* is a term that describes how well an individual is able to handle the given problem. The structure of the chromosomes

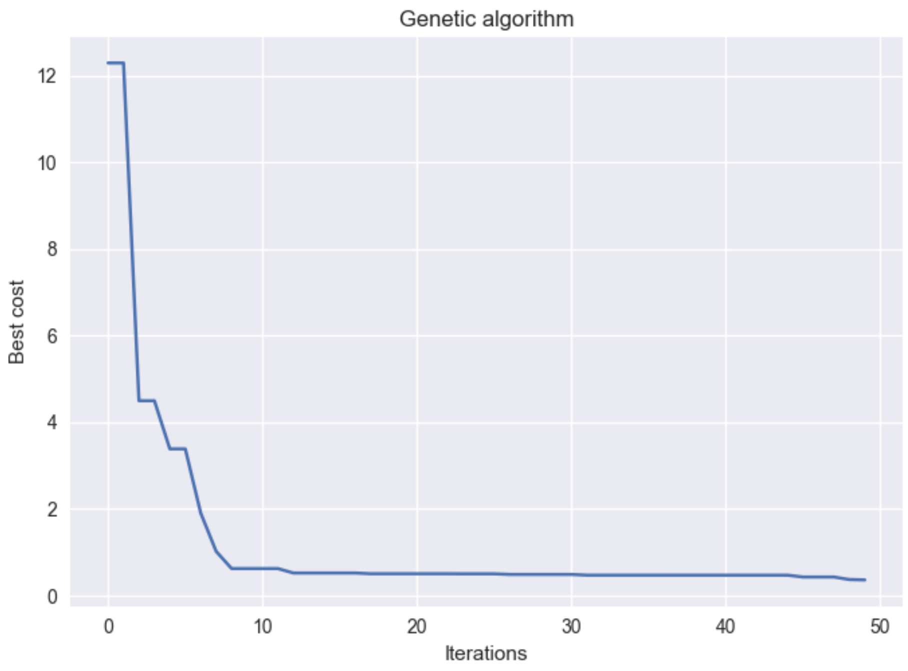

FIGURE 2.12: The convergence of a genetic algorithm. In this plot, the cost of the objective function is plotted along the iterations of the GA. The objective function for this plot is $\sum x^2$.

and the problem that determines the fitness of the population depends on the problem that is being solved by the GA.

For a given problem, the fitness score of different individuals of the population will determine how good each individual is performing relative to the rest of the population. As GA is an optimization algorithm, the objective function used to calculate the fitness score will be some function where the aim is to either maximize or minimize the value of the function [11].

The population of solutions will evolve from one generation to the next. This evolution is done by a set of evolutionary operator, that will decide how the next generation will differ from the previous. These operation govern which individuals that will join the next generation, which individuals will be chosen to create "offspring", which of these offspring will have an added mutation in their chromosome, how this mutation will be performed, and all of these will be performed for how ever many such generation that will be simulated.

The term *offspring* in the context of genetic algorithm means that certain crossover operators will be used on a selected few individuals that will be parents for the next generation. The cross-over operator decides how the two parent chromosomes will be used to create offspring, which will be a combination of the chromosome from both parents. One crossover method is known as the *roulette wheel selection*. This is a weighted random selection function. As the name suggests this method is based around a roulette wheel, where each sector on the wheel represents one individual. As the wheel stops spinning, an arrow lands on an individual that is then selected as a parent. The size of each sector is proportional to the individuals fitness, such that the individuals with a higher fitness will have a higher probability of being chosen as a parent. So for individual i, with a fitness score of f_i, the probability of any individual of being chosen as a parent is given by the following equation:

$$P_i = \frac{f_i}{\sum_{i=1}^{n} f_i} \tag{2.27}$$

Another common method of parent selection is known as *tournament selection*. In this method, k individuals are chosen out of the population, and they are then ranked based on their fitness scores. The highest fitness scores get chosen as parents for the next generation. It is called tournament selection due to the similarity with a tournament bracket, where the individual with the highest fitness will be the winner of the tournament. The analogies of the parent selection schemes can be seen illustrated in figure 2.13. Tournament selection is based on a tournament bracket, while roulette wheel is set up with every section representing each individual where the size of the area is proportional to the fitness of that individual.

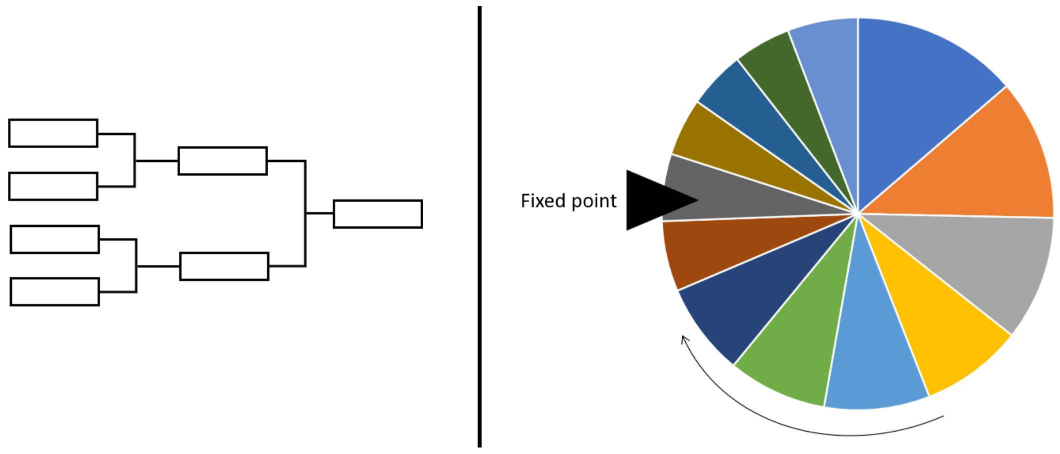

FIGURE 2.13: Tournament selection (left), and roulette wheel selection (right).

Another key aspect of a genetic algorithm is the choice of crossover operator. This operator will decide how the next generation will perform comparatively to the previous generation. The crossover operator decides how much of the chromosome of the new generation will be inherited from the parents.

The simplest crossover operator is the *single point crossover*. This operator decides one random point on the chromosome, which will be the same for both parent chromosomes. Two offspring chromosomes are then created by combining one part of parent 1 with the other part of parent 2, and another offspring will be created by the parts not used for the first offspring. The new offspring consists of one part of each parent. Figure 2.14 shows an illustration of how this crossover operator works.

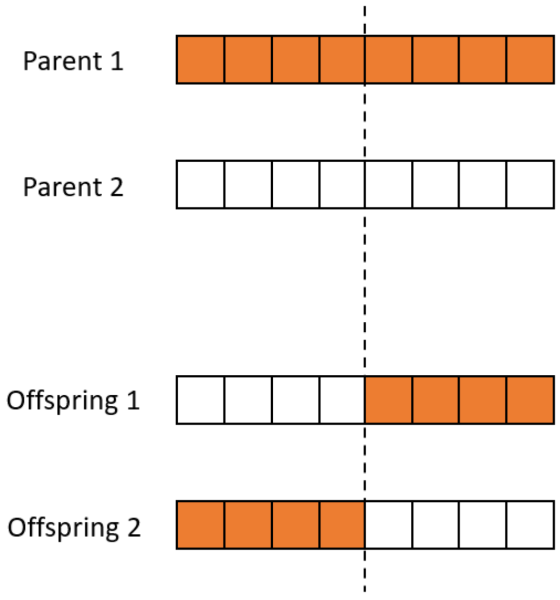

FIGURE 2.14: Illustration showing the single point crossover operation

Other crossover operations include the *multi-point crossover*, or a *k-point crossover*[42], as seen in figure 2.15. Where the single point crossover only selects one point on the chromosome, k-point crossover will select k random points on the chromosomes which will represent crossover lines. The offspring will then be created by combining every other part of each parent, such that the offspring will consist of some parts of each of the parent chromosomes.

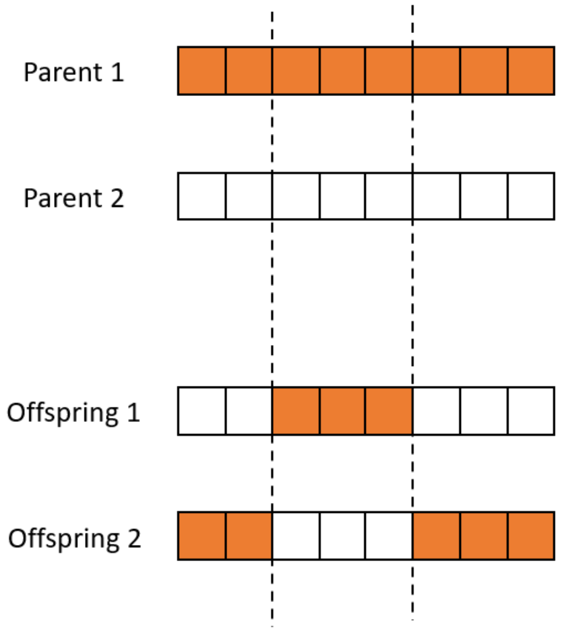

FIGURE 2.15: Illustration showing how k-point crossover works, in this case with a two-point-crossover

Uniform crossover(figure 2.16) is a method of performing crossover which does not involve some crossover point on the chromosomes. In uniform crossover each of the genes of the parents will be randomly assigned to one of the offspring. A uniformly random vector α will be generated, where each index in the vector is referring to whether a specific gene will be used for one offspring or the other. Let $P1$ and $P2$ be the parents that are picked to produce offspring, as such:

$$P_1 = [P_{1,1}, P_{1,2}, P_{1,3}, ..., P_{1,N}] \tag{2.28}$$

$$P_2 = [P_{2,1}, P_{2,2}, P_{2,3}, ..., P_{2,N}] \tag{2.29}$$

The uniformly random vector α is given as

$$\alpha = [\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n], \alpha_i \in (0, 1) \tag{2.30}$$

Then the offspring, C_i , will be given as such

$$C_i = \alpha_i P_{1,i} + (1 - \alpha_i) P_{2,i} \tag{2.31}$$

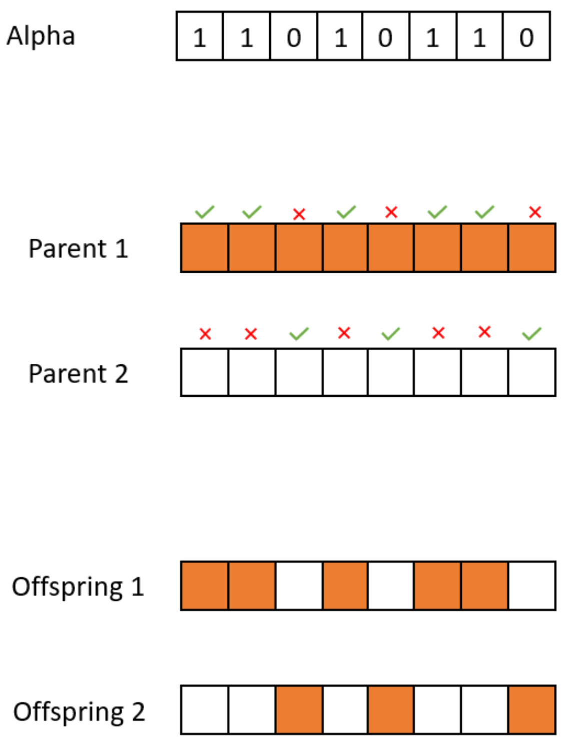

FIGURE 2.16: Illustration of a uniform crossover operator

After the parents have been selected, and the crossover has been completed, the offspring may be subjected to *mutation*. Mutation means that some of the chromosomes in the offspring will be changed at random, which is analogous to the mutation of genes in real animals from one generation to the next. Mutation is dictated by a mutation rate, which operates as a probability of each gene in a chromosome being a subject to mutation. If the offspring simply mimics the parent generation, or a combination of the best parents, it means that the only genes left in the gene pool after a set amount of generation will be a combination of the best genes in the initial population. By including a mutation operation, there will always be new genes added to the total gene pool. This means that a wider range of genes will be used to search for the optimum combination of individual genes. Without the mutation operator, the GA algorithm is extremely dependant on having a "good" initial generation, meaning an initial population with a high average fitness score, as the genes in this generation are the only genes that will be used for optimization of the fitness score. The mutation operator reduces this dependency on the initial population, which ultimately improves the applicability and utilization of the genetic algorithm. The mutation operation can bee seen in figure 2.17 for two different mutation rates.

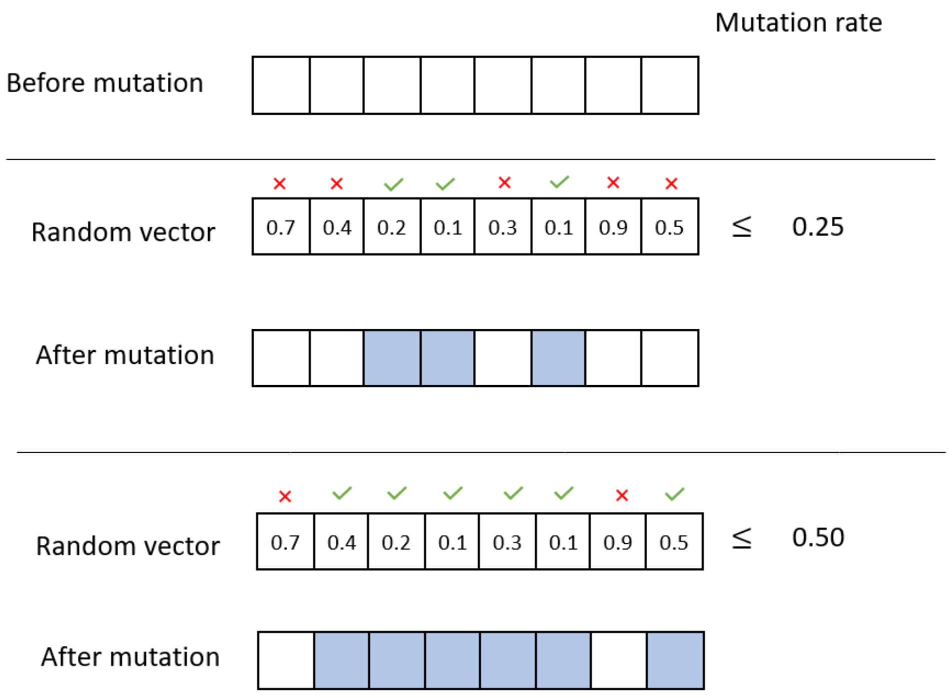

FIGURE 2.17: The mutation operation shown with two examples of mutation rate

First a vector is generated, containing random values between 0 and 1. Then the script generates a new vector, containing boolean values of whether the random values are higher or lower than the mutation rate. The indices where the random value is lower than the mutation rate gets mutated. This concept is illustrated in figure 2.17.

After initializing a population, performing crossover using a selected pair of parent chromosomes, and performing mutation on a selection of the offspring, the next generation will be initialized. The population size of each generation is an arbitrary parameter in genetic algorithms. Every generation after the first one will consist of a combination of the parents, the offspring, the mutated offspring, and also some of the best performing individuals of the previous generation. When the best individuals of the previous generation are brought into the next generation it is referred to as *elitism*. It is a convenient way of populating the next generation and ensure that the gene pool of the next generation consists of more than just the genes of the individuals that produced offspring. the concept is illustrated in figure 2.18. First, a set of parents is determined from a population of 6. Then, a set of 6 new offspring are made by applying a crossover operation on the chosen parents. These new offspring are then subjected to mutation, where some of them will change their attributes. Then the initial population and the offspring population are consolidated into one population, which is sorted by the fitness of each individual. Then the 6 best individuals are picked for the next generation.

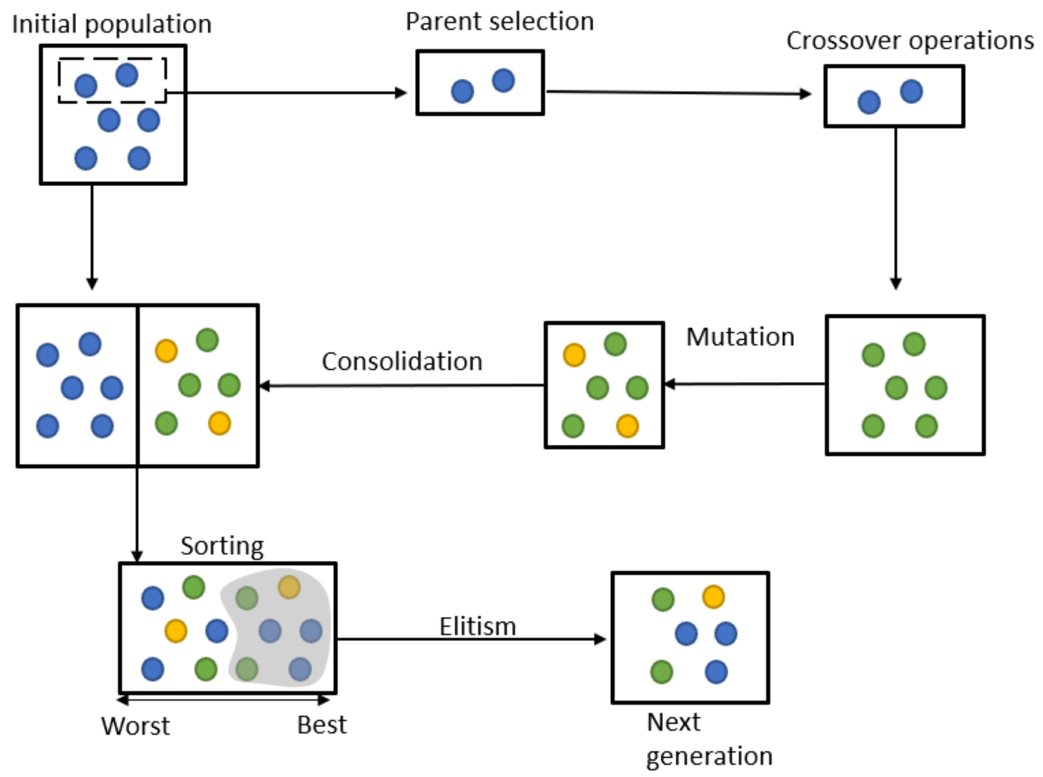

FIGURE 2.18: Illustration of the concept of elitism

As the genetic algorithm moves from one generation to the next, the overall population will be able to perform better at the optimization problem at hand. This is similar to the concept of evolution of species of animals, which was mentioned earlier. In the animal kingdom, the nature that surrounds species of animals determine how the species will interact with the surroundings, as the animals that are able to survive and procreate will be the ones passing on their genes to the next generation of animals. The same principle is what works as the main mechanic in a GA. For each new generation, the individuals in the population will over time be able to better solve the optimization problem in question, whether it is to minimize or maximize the objective function. A general trend in GA is that over the iterations, which represent the generations, the overall cost function will be reduced over time, where the cost of an individual has a negative correlation with the fitness.

A common benchmark for GA is the traveling salesman problem[41]. In this problem, a salesperson is supposed to travel to n specified cities to sell wares. The distance between each city will vary a lot, and the salesperson wants to take the most optimal route such that they spend the least amount of time traveling between the cities. To solve this problem one could implement a brute force approach, which is done by finding all the possible routes one could take, and then compare the times. Mathematically this is an inefficient approach with a big O notation of $O(n!)$, which means that this approach will use drastically longer time for each city added to the problem description. If a GA, or another meta heuristic approach, is applied to this issue one can generally find an optimal solution to this problem in a much shorter amount of time. Often this solution may not be the absolute optimal solution, but generally within a slight range of the optimum[41].

Chapter 3 Methodology

This chapter will be presenting how the work was done for this thesis, in terms of which software was used and also how they were used. The methods used will be described in detail, and some pseudo-code will be used to illustrate how the concepts have been adapted in python.

3.1 Workflow

The overall goal of the project work is to create a smart proxy model for a synthetic reservoir model, so the first step is to have a synthetic reservoir to work with. The artificial neural network is supposed to predict field oil rate for the reservoir. For the development of a proxy model, the workflow can be summarized as such:

1. Investigate the reservoir
2. Determine reservoir constraints
3. Collect samples from ECLIPSE using a LHS design.
4. Format the samples into a database
5. Import the database into python
6. Pre-process the data, normalization
7. Separate the data into training-, testing-, and validation data
8. Train the ANN to predict FOPR using the dataset
9. Validate ANN; tune hyperparameters or alter preprocessing on the data until the model reaches acceptable levels of accuracy using a blind test
10. Perform a GA on the proxy model
11. Perform a GA using ECLIPSE
12. Compare results

The synthetic reservoir that was used is known as the Egg Model[17], which will be explained more in depth in a later section. This reservoir is then coupled together with a synthetic fluid model such that fluid flow can be modelled accurately. For consistency's sake the fluid model is the one that is mentioned in the paper[17]. An artificial neural network requires a proper sampling scheme in order to counteract over-fitting, so data sampling was performed on the synthetic reservoir model. Data from these samples were then formatted into a .csv file using a spreadsheet software, such that they can be transported into python. After importing the data into python, it is used to train the ANN. The hyperparameters of the ANN are tweaked until it reaches an acceptable accuracy. After the proxy model has been trained, it will be subjected to a genetic algorithm which will also be done in python. This GA will use the cumulative oil production as an objective function, and it will use the proxy model to do evaluations of the reservoir production. The proxy model is estimating the FOPR of the reservoir, and this oil production rate can easily be turned into cumulative oil production by simply taking the sum and multiplying with the sampled timestep. This same GA will be used on the ECLIPSE reservoir simulator. Then, the results of both of the optimization methods, as well as the run time, will be compared and discussed.

3.2 Software

Reservoir simulation

ECLIPSE reservoir simulator, from Schlumberger[32], is a well known and well established reservoir simulator. For the work in this thesis ECLIPSE has been used for reservoir simulation. This has been used both when conducting limit testing on the synthetic model, sampling on the synthetic model, and coupled together with a GA for optimization of oil production.

Python

Python is a high level programming language, which is open source and thus has become an industry standard in a modern society. It can be used for a wide range of applications, and has a large variety of different libraries that can be applied to almost any setting.

For the work in this thesis python has been used for preprocessing of data, data visualization, artificial neural network training, genetic algorithm, and data sampling. The work has been done within an anaconda environment, which have been managed within the pyCharm software. Anaconda is an open source package management system which is convenient to use, especially when working with artificial intelligence as there are a lot of packages that are convenient to use for this.

For easy data management the package *pandas*[28] has been used. This package makes it quick and easy to import data straight from a file into python, and have it organized in a fast and easily manageable dataframe. For the purposes of this thesis it was to import from a ".csv" file, which is a file with comma separated values.

When working with ANNs it is important to be consistent with how the data is being presented to the neural network. To ease this process the package *scikit-learn*[33] has been used to simplify the processing of data. This package makes it simple to normalize data and separate the data. The package includes a lot of tools, which can be used as they are or they can be modified with little effort.

Tensorflow [39] is an open ended package which aids at simplifying the applications of machine learning. It has been used in this thesis for a lot of the work that has been done with the ANN. Specifically the package *keras* within tensorflow can be used to set up a neural network, where it is simple to specify the amount of hidden layers, nodes per layers, the optimizer for the network, the loss function, the activation function for the layers, and saving/loading of the ANNs.

3.3 Reservoir model

The reservoir model used for this project is the Egg Model [17](figure 3.1). This is a synthetic reservoir that was initially created for a PhD thesis published by Marteen Zandvilet and Gjis van Essen [12]. Since then is has been used for a large variety of studies which aim at applying some computer-assisted optimization algorithm or history matching. Jansen et al.[17], have put together a standardized version of the Egg model such that comparisons between different publications are comparable.

One could refer to the egg model as a sort of benchmark model for a lot of publications on computer applications to reservoir engineering.

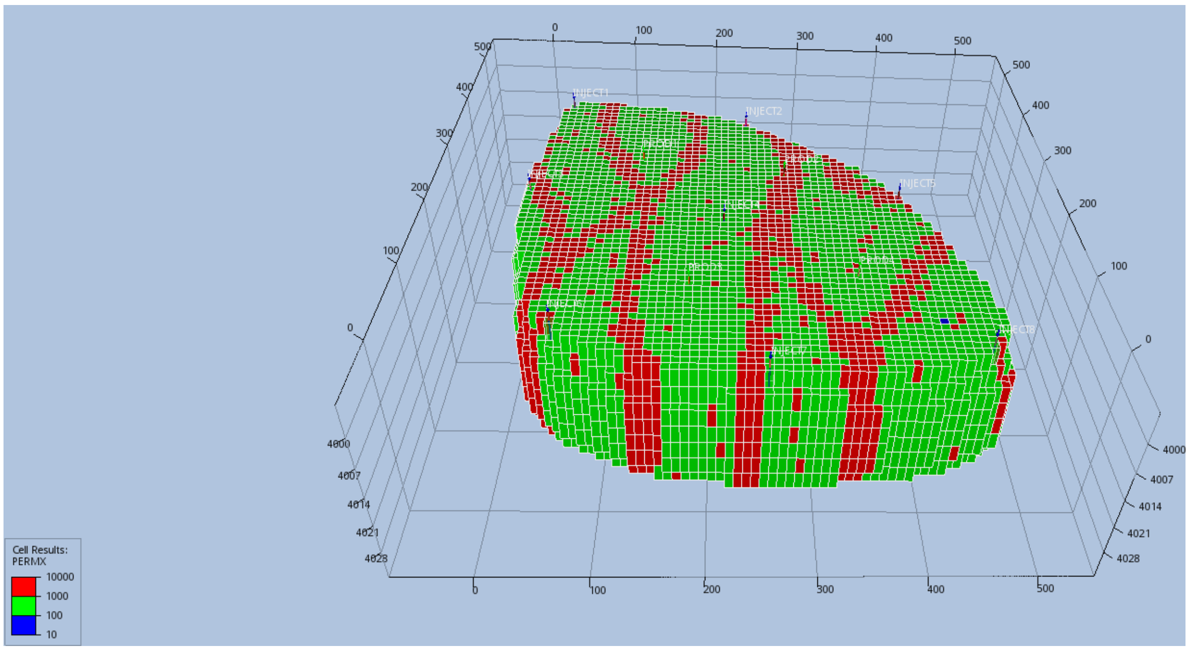

FIGURE 3.1: The egg model, with a clearly visible channelized permeability through the entire model. [17]

The egg model is a homogeneous reservoir with an assumed constant porosity of 20% , with a channelized permeability. This means that there are several veins moving through the reservoir with a good flowing permeability, while the rest of the reservoir is fairly low in permeability. The reservoir consists of 60 x 60 x 7 grid cells, which makes a total of 25.200 cells. Not all of these are active, only 18.553 cells, and the active cells make up a shape which is similar to an egg, hence the name. The rock parameters of the synthetic reservoir is seen in table 3.1, and the fluid parameters in table 3.2. The reservoir has 8 injector wells and 4 producers. The injectors are located mostly around the outer perimeter of the active cells and one injector in the center of the reservoir. The four producers are surrounding the middle injection well. The simulation has been done over a timeline of 4000 days($\sim$11 years) , starting on the 15th of June 2011 and ending on the 28th of May 2022.

Symbol	**Parameter**	**Value**	**Unit**
ϕ	Porosity	0.2	-
h	Reservoir thickness	28	m
C_o	Oil compressibility	$1.0x10^{-10}$	Pa^{-1}
C_r	Rock compressibility	0	Pa^{-1}
C_w	Water compressibility	$1.0x10^{-10}$	Pa^{-1}
p_{Ri}	Initial reservoir pressure	400	Bara
S_{wi}	Initial water saturation	0.2	-

TABLE 3.1: Reservoir parameters used for numerical reservoir simulation on the Egg Model

Symbol	Parameter	Value	Unit
ρ_o^o	Oil density at surface conditions	900	kg/m^3
ρ_w^o	Water density at surface conditions	1000	kg/m^3
μ_o	Oil dynamic viscosity	$5.0x10^{-3}$	Pa s
μ_w	Water dynamic viscosity	$1.0x10^{-3}$	Pa s

TABLE 3.2: Fluid parameters used for numerical reservoir simulation on the Egg Model

3.4 Artificial neural network

For this thesis, one of the goals was to develop a smart proxy model for a synthetic reservoir that is able to accurately replicate the field oil production rate (FOPR) of the synthetic reservoir. The smart proxy model is being developed using a machine learning, which is a data driven method, meaning that the ANN will learn how the reservoir operates based on the data provided. It is therefore important that the data is of high quality, and that it covers the parameter space such that the machine learning model actually learns the intricate relationships within the reservoir and doesn't just memorize how it works in a couple of examples. For this reason it is of utmost importance to the performance of the proxy model that the model is provided with good data, which can be achieved by performing a proper sampling technique.

For a simple approach the proxy model doesn't include that much data. The input data for the model is the injection rates, as well as the timestep. The ANN will use this data to predict the current field oil production rate of the synthetic model.

3.4.1 Data sampling

Choosing a good data sampling scheme is important in order to prevent the machine learning model of over-fitting. The sampling should have adequate space filling. The sampling in this thesis has been done using the latin hypercube sampling method, which has been described further in chapter 2.

For LHS the specific number of samples can vary. Some cases require more samples in order to achieve proper space filling. For this project an initial sampling of 20 samples was done before realizing that this did not provide the machine learning model with enough data to achieve a desired degree of accuracy. Thus, a 30 sample LHS design was done.

The data samples were gathered from the ECLIPSE reservoir simulator. When performing data sampling it is important to have a good understanding of what data you need to collect, as this data will be what is determining how well the final proxy model will perform. The goal is to make a proxy model that will accurately predict the FOPR of the synthetic reservoir.

The samples consist of output data from several simulation runs performed in ECLIPSE. There are several types of proxy models for a hydrocarbon reservoir. These are grid based-, well based-, and field based proxy models. A well based proxy model will be based on data related to all the wells in the field. Typically this will mean that the input data is data related to the wells, which can typically be the bottom hole pressure for all the injector wells. A field based proxy uses data related to the reservoir on the field scale, which typically is injection rates into the reservoir and field oil production rate. In this thesis the focus was to make a field based proxy. The input data for this proxy was decided to be injection rates and time. One could possibly add porosity and permeability as well, but seeing as the porosity and average permeability of the reservoir would be constant for any samples they could just as well be left out of the training of the proxy model.

Another option for a field based proxy is to also include information from the well bottom hole pressure (BHP) of the injector wells. An issue with doing this is that the bottom hole pressure is a dynamic data, meaning that it changes over time as a function of reservoir production. The proxy model for this synthetic reservoir is supposed to be able to only predict the FOPR. If the BHP was to be used as input data this would require to have one proxy model for predicting the BHP for each of the injector wells, and then supplying this data to the original proxy model for the FOPR. This kind of design is called a cascading design . For the work in this thesis it was decided that it is overall beneficial for the proxy model if such a design was not performed.

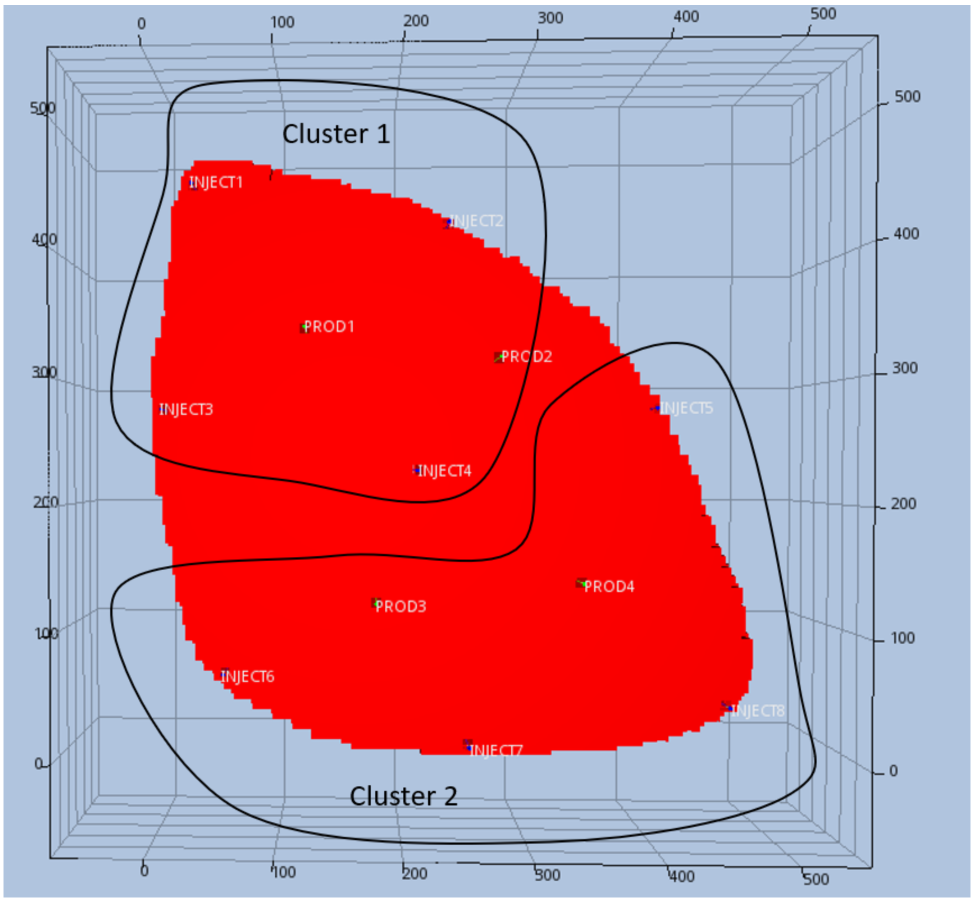

FIGURE 3.2: Separation of the 8 injector wells into 2 clusters of wells

Initially the egg model has 8 injector wells and 4 producer wells. Tho simplify the sampling process the injector wells were split up into two clusters of wells, which will further be referred to as "Injection cluster 1" and "Injection cluster 2". The separation of the wells can be seen in figure 3.2 and table 3.3, where injector wells 1-4

Cluster 1	Cluster 2
INJECT1	INJECT5
INJECT2	INJECT6
INJECT3	INJECT7
INJECT4	INJECT 8

TABLE 3.3: The separation of the injector wells

have been placed in cluster 1 and wells 5-8 have been placed in cluster 2. It was decided that the injection rates were kept constant for each simulation duration. After performing some test simulations on the synthetic reservoir it seemed like the reservoir responded quite well to injection rates between 50 $\frac{Sm^3}{d}$ and 150 $\frac{Sm^3}{d}$, so this was picked as the parameter space to collect samples. In total 30 samples were collected from this interval, as well as 4 extreme points which were taken from the extreme values of the interval (table 3.4). The samples can bee seen in figure 3.3.

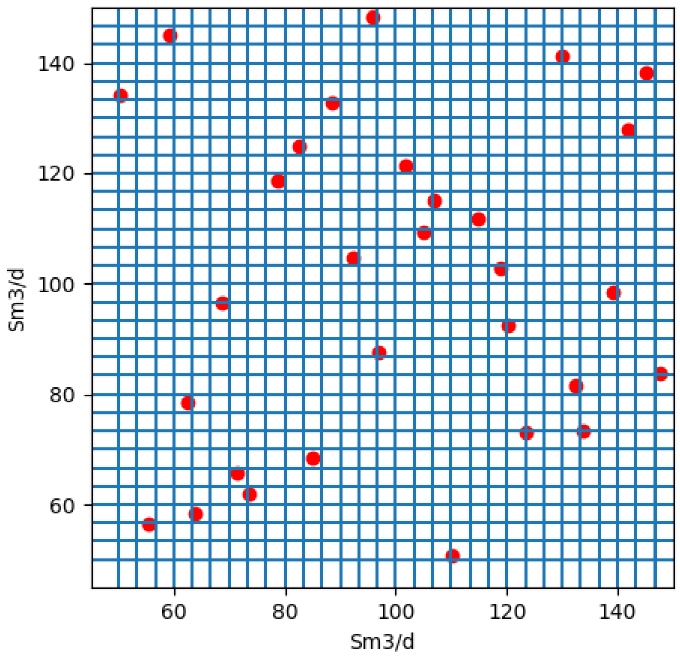

FIGURE 3.3: 30 LHS samples

These samples will be consolidated into a dataset that will be used for training, testing, and validation of the ANN. In addition to this it is important to also have a blind test, which consists of data that the model has never seen before. A blind test aids at assessing how well the ANN has learned what is happening in the reservoir, and ensuring that it isn't just memorizing results from the train-test-dataset. The blind test has been taken at 100 $\frac{Sm^3}{d}$ for both of the injection clusters, which represents the middle of the parameter space. Assessing the performance of an ANN can be done with the training- and testing dataset as well as the blind test.

Sample	Cluster 1	Cluster 2
#	$[\frac{Sm^3}{d}]$	$[\frac{Sm^3}{d}]$
1	50.15	134.29
2	55.35	56.53
3	59.16	145.00
4	62.59	78.42
5	63.77	58.44
6	68.74	96.41
7	71.74	65.73
8	73.08	61.91
9	78.80	118.60
10	82.42	124.78
11	84.87	68.37
12	88.43	132.94
13	92.19	104.71
14	95.82	148.38
15	96.76	87.64
16	101.70	121.24
17	105.07	109.42
18	106.19	115.0
19	110.08	50.75
20	114.74	111.91
21	118.90	102.85
22	120.07	92.40
23	123.51	73.07
24	129.91	141.35
25	132.38	81.58
26	133.64	73.49
27	139.11	98.47
28	141.82	127.92
29	145.11	138.28
30	147.61	83.72
31	50.00	50.00
32	50.00	150.00
33	150.00	50.00
34	150.00	150.00

TABLE 3.4: The 30 LHS samples and the 4 extreme points that constitute the dataset that will be used to train the proxy model.

3.4.2 Artificial neural network setup

Keras has been used to set up the ANN. Optimization of the hyperparameters of a neural network is important in order to get the network to perform as good as possible.

It is important to normalize the data before supplying it to the ANN. This has been done by normalizing the data within the range of $[-1, 1]$. After normalization, the dataset is separated into distinct datasets containing training-, testing-, and validation data. The data is split into an 80%-20% split of respectively training data and testing data. The blind test is used for validation.

It is simple to specify the amount of hidden layers in keras, as well as nodes per layer. The exact correct amount of layers and nodes often requires some tuning in order to find the best combination of hyperparameters. Even with a perfect set of hyperparameters there will always be some randomness in terms of how well a neural network will learn, so you could supply the same neural network with the same data and not get exactly the same results. Still, by doing tuning of these hyperparameters the impact of this randomness will be reduced. ReLu was used as the activation function of each neuron. In order to minimize the impact of over-fitting one could include dropout layers between the regular layers of the neural network. These dropout layers will make sure that some of the outputs from the neural network is ignored or skipped. This adds an effect of random noise to the learning of the network, which aids at avoiding the issue of over-fitting the model.

The ANN uses *Adam* as an optimization function. The optimization function of the neural network determines how the neural network is learning. Adam is an alternative optimization method which is an improvement on a more traditional gradient descent search function. One parameter that goes into Adam is the *learning rate*, which is the stepsize used in the gradient search algorithm.

The loss function used for training the model is mean squared error(MSE) from the package *scikit-learn*:

$$MSE(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n} (y_i - \hat{y}_i)^2 \tag{3.1}$$

where y represents the real value, and $\hat{y}$ represents the predicted value by the ANN.

MSE has also been used as a validation metric for the accuracy of the ANN. Another measure of the accuracy is the coefficient of determination , also known as the R^2-score. The R^2 score provides a measure of how good of a fit the ANN is compared to the real data. It has also been calculated using the *scikit-learn* package as such

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \overline{y})^2} \tag{3.2}$$

where $\overline{y}$ is the average across the measurements. The coefficient determines how well of a fit the model is compared to real data. A good fit is regarded for model that achieves R^2-scores that are close to 1.0, as 1.0 is the maximum score possible. High values indicates that the model is more likely to predict pretty good if it's supplied with blind data, and a low score corresponds to a lower probability of getting an accurate prediction.

3.5 Genetic algorithm implementation

The genetic algorithms have been coded in python. The overall workflow has been summarized in the following pseudocode:

```
def geneticAlgorithm():
    Initialize population of chromosomes
    for max_generations:
        for individuals in population:
            evaluate individuals
            Select parents
                Roulette wheel or Tournament selection
            Perform crossover operation
            Mutate offspring
            Add children to the population
        Sort the population based on fitness
        Select n individuals to be next generation
```

The initial population is decided by a predetermined value, n_{pop}. Each individual of the population consists of a chromosome. The chromosome consists of 8 genes, where the value of the genes decides the value for both of the injector clusters. For the initial population the values of each of the genes are randomized, and the injection rate for each cluster is calculated by taking the sum of four of the genes for each cluster. The fitness of each chromosome is calculated using a script in python that does the following:

```
def evaluatePopulation(chromosome):

    InjectionCluster 1 = f(chromosome)
    InjectionCluster 2 = f(chromosome)
    gaData = f(InjectionCluster 1, InjectionCluster 2)
    modelEvaluation = proxyModel.evaluate(gaData)
    FOPT = sum(modelEvaluation)

    return FOPT
```

The input parameters for this function is the problem description and a single chromosome. The function calculates the value of both injection clusters based on the individual chromosome, and uses these values to generate a dataframe. This dataframe is represented in the same way as the training, testing, and validation data that is used on the proxy model. This data is then provided to the proxy model to generate the FOPR of the reservoir, which is summed together and multiplied with the sampled timestep to get the cumulative production of the reservoir. This objective function is used to evaluate every individual in the population. It is quite fast, with each evaluation with the proxy model not taking more than a couple of seconds.

Parent selection is done based on the fitness scores of the population. A common implementation is roulette wheel selection. In this case it is probably not the optimal choice. In a hydrocarbon reservoir, production will occur even without the injection of water. This means that even for lower injection rates the reservoir will still be producing oil, which will give even individuals with low injection rates quite decent production. Thus a roulette wheel selection will be a bit too random to prove useful. When the roulette wheel selection was in charge of selecting the parents, the GA used many generations with a random correlation in terms of which injection rate provides the best FOPT each iteration. Just because it is random does not mean that it is objectively bad, but by implementing a tournament selection instead the algorithm seemed to converge quicker so in this case it is preferable to avoid the initial randomness.

How many children that are generated each iteration is decided by a factor which is called the *proportion of children*. This number dictates how many times the parent selection is performed, and it runs until the proportion is met. This number can be any value between 0 and 1, where 1 means that there will be generated as many children as there are individuals in the population. This implementation used the uniform crossover operator. After choosing parents and performing crossover operations, each offspring has a chance of being mutated. This is decided by the *mutation rate* of the GA.

After generating all the offspring for each generation, these get added to the total population. Then the population gets sorted, and n_{pop} individuals are selected to move on to the next generation. This means that if some of the new offspring have a lower fitness score than some non-parent individuals of the previous generation, the offspring does not join the next generation.

3.6 Coupling genetic algorithm with ECLIPSE

The technicalities of the GA implementation is similar for both the proxy model and for using ECLIPSE. Both use the same mechanics for population generation, parent selection, mutation, and crossover. The difference lies in how to evaluate the fitness of each of the individuals in the population. When performing evaluations using the proxy model, the proxy model is supplied with a dataset of time and the injection rates, and then the model generates a vector of FOPR for the time vector, which is used to calculate the FOPT. For ECLIPSE, a new set of input files are generated in python, then ECLIPSE is being run from python, and then the value for FOPT has to be imported back into python. The following pseudo-code illustrates how the objective function is solved for the GA running ECLIPSE:

```
def evaluatePopulation(chromosome):

    InjectionCluster 1 = f(chromosome)
    InjectionCluster 2 = f(chromosome)
    makeInputFile = f(InjectionCluster 1, InjectionCluster 2)
    runEclipse(inputFiles)
    FOPT = getFOPTfromRSM(RSM_file)

    return FOPT
```

The input for this objective function is the same as for the proxy model. In this case the injection rates of the clusters are used to generate an input file for ECLIPSE rather than a dataframe. Input files for ECLIPSE can be handled similarly to .txt files it is simple to create these files in python. After generating an input file, ECLIPSE is run from python by using a macro. One possible output file for ECLIPSE is a .RSM file, which can be opened in both Microsoft Excel and python. This file is opened from another function within the python script, which finds the final value of FOPT after running a simulation. This value is then returned.

Chapter 4

Results & Discussion

The following chapter will show the results from the implementations that were described in the previous chapter. It will also include a discussion of what the results mean, or how they could be interpreted. The chapter will also include some shortcoming of the implementation at the end of the chapter.

4.1 Proxy model

The proxy model has been trained on the training and testing data, and then a blind test has been used for validation of the accuracy of the model. The hyperparameters used were determined by trial and error, and the optimal values can be seen in table 4.1.

Hyperparameter	Value
Epochs	1000
Learning rate	0.001
# of hidden layers	5
# of nodes per layer	50
Activation function	ReLu
Dropout	None

TABLE 4.1: Hyperparameters used to generate the ANN

The plots in figures 4.1 and 4.2 shows the proxy model used to evaluate for 2 random samples from the test dataset. The proxy model prediction is a good fit compared to the real data gathered from ECLIPSE. The plots are near an exact match. Table 4.2 and 4.3 also show that the proxy model is performing well on the test database. The R2 score is close to 1 and the MSE is low.

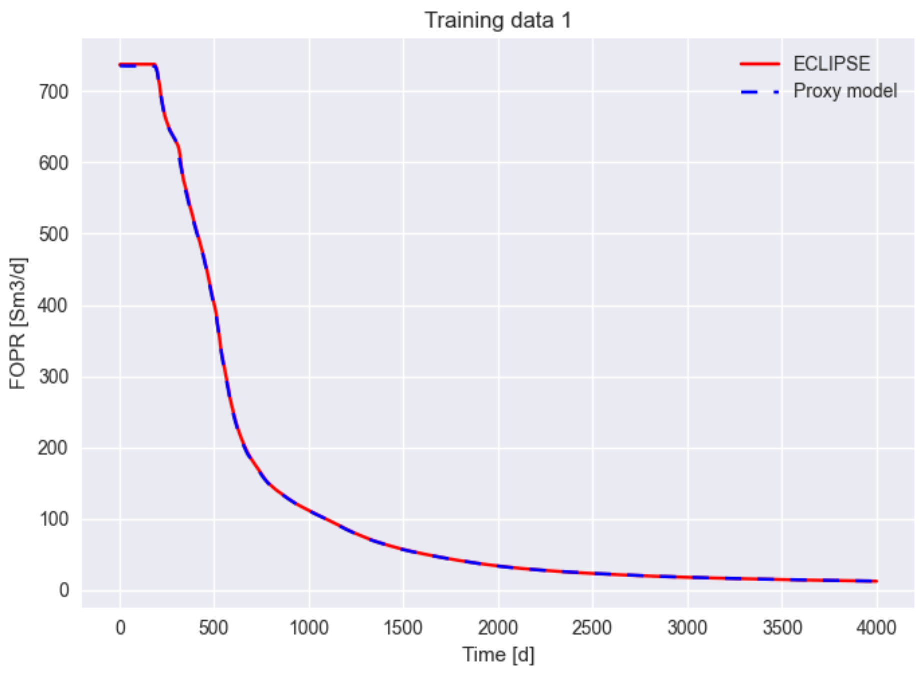

FIGURE 4.1: Proxy model prediction plotted versus real data from the training dataset

The R2 score is slightly worse for the test dataset, and the MSE for the test dataset is significantly higher than the other datasets. This might not be too significant. Figure 4.3 shows the error between the test data from ECLIPSE and the prediction from the proxy model. This plot shows that the proxy model is generally doing a good job

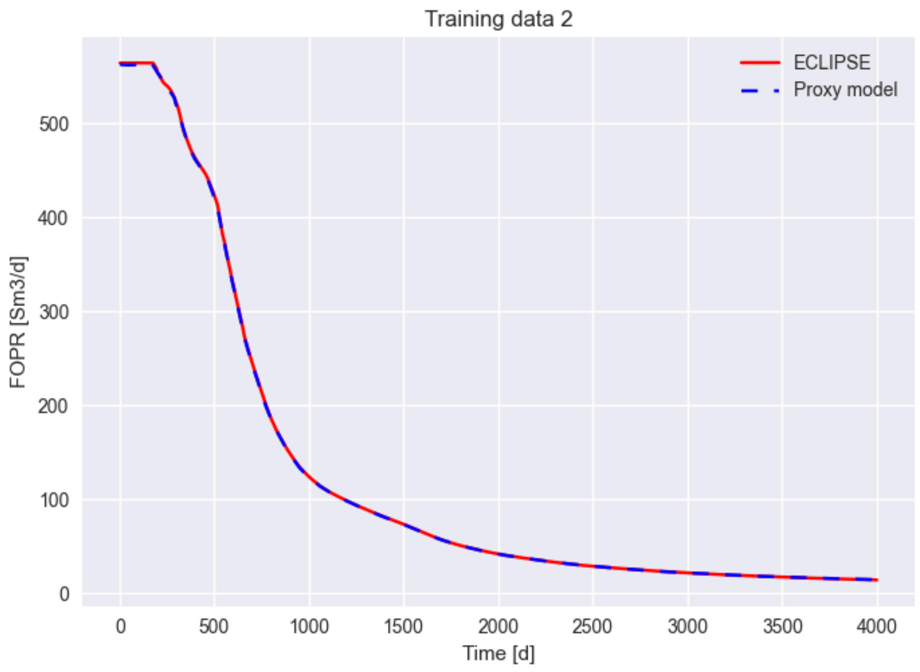

FIGURE 4.2: Proxy model prediction plotted versus real data from the training dataset

at predicting the FOPR, while there are several points where the error is significant. As the MSE of the test data is 162.6 $(\frac{Sm^3}{d})^2$ it could be that this high number is caused by outliers, as the overall error seen in figure 4.3 shows that there is an overall good correlation between the proxy model and the test data. The R2 score of the test data is still quite good and does not match the high value of the MSE.

R^2**-score**	**Value**
Training data	0.99998
Testing data	0.99634
Validation data	0.99996

TABLE 4.2: R^2-scored for the proxy model on the various datasets

MSE	**Value**$[\frac{Sm^3}{d}]^2$
Training data	0.78561
Testing data	162.6443
Validation data	1.3545

TABLE 4.3: MSE for the proxy model on the various datasets

The blind test was done with both of the injection clusters having a rate of 100 $\frac{Sm^3}{d}$. Figure 4.4 shows a plot of the real data from the blind test as well as the proxy model prediction given just the injection rates and the time.

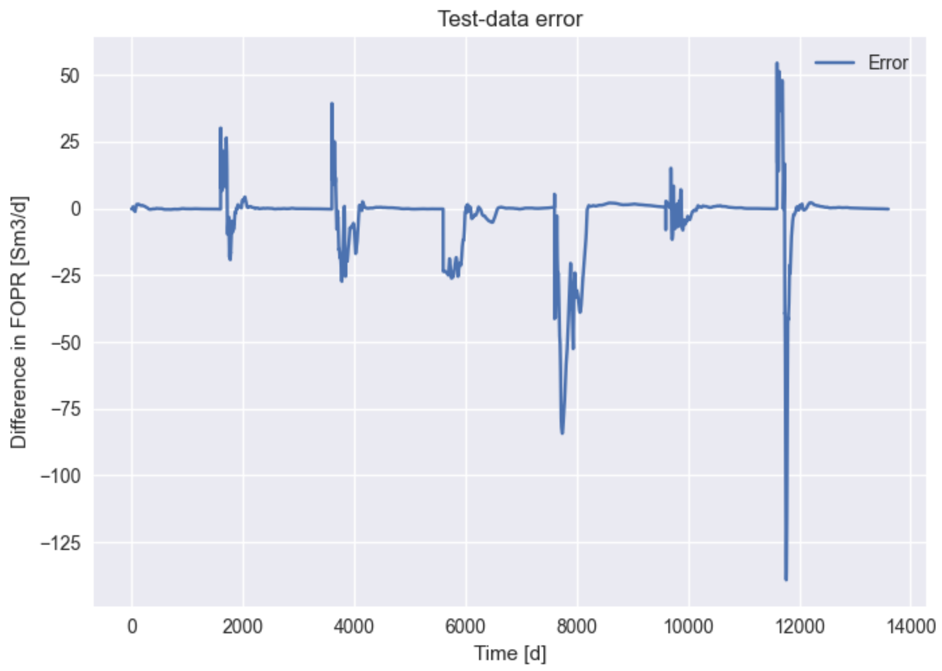

FIGURE 4.3: Error from the real test data from ECLIPSE minus the evaluation from the proxy model

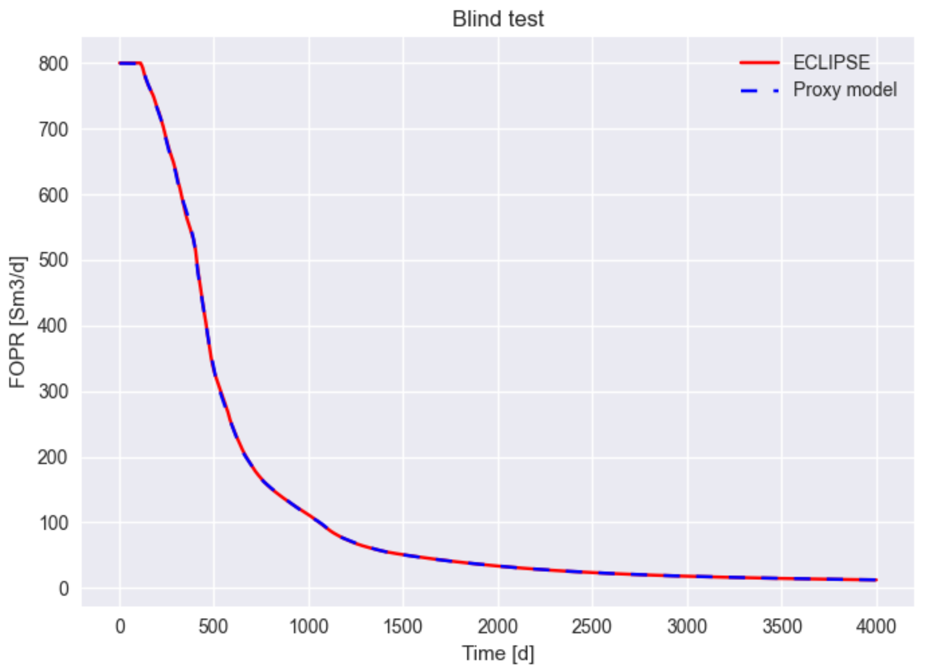

FIGURE 4.4: A plot showing the real blind test data along with the proxy model prediction.

Overall the proxy model is doing accurate prediction of the FOPR of the synthetic reservoir. As seen in figure 4.4, the proxy model prediction using the input data for the blind test is almost an exact match of the real data generated using ECLIPSE. The proxy model predicts a cumulative oil production of 5.13287 x 10^5 Sm^3, and according to the data from ECLIPSE the real value is 5.13922 x 10^5 Sm^3, which means that the proxy model is able to predict the oil production of the blind test with an error of less than 1 %. The R^2-score has been calculated using the training/testing dataset as well as the blind test, and the scores are quite high. A R^2-score of 0.99996 for the blind test indicated that the proxy model has learned how the injection rates correlate with the FOPR well, as the blind test includes input data that the proxy model has never seen before. Having a decent match with the blind test indicates that the proxy model has not been over-fitted to the train/test dataset. Due to the good fit of the proxy model, it is justifiable to use the proxy model for an optimization study on the synthetic reservoir.

It is curious considering that the model was set up without dropout layers in between the hidden layers of the neural network, and it still showed little signs of overfitting. This may be because the sampling is thorough enough such that the model is able to not become overfit even without dropout layers. Initially the sampling was done as a 20 sample LHS-design. With these samples the model was set up with 500 epochs and dropout layers between every hidden layer in the network, with a dropout rate of 5%. This proxy model performed quite well on the blind test. When it was used for the GA however, it predicted a higher cumulative oil production for lower injection rates. This is not expected behaviour, and the model was deemed unfit to be used for optimization purposes, as it was not representative of the reservoir behaviour. Thus, a 30 sample LHS was performed instead. This new model uses 1000 epochs, which could possibly positively correlate with more samples in terms of becoming un-biased towards data. The 20 samples might not have been enough to thoroughly teach the proxy model how the parameters in the reservoir correlate. On the other hand, 30 samples together with more epochs gives enough samples to avoid overfitting to some of the values, and thus the model is able to perform well even without dropout layers.

4.2 Genetic algorithm

The genetic algorithm is dependent on many variables. A couple of these variables are able to drastically determine the effectiveness of the optimization. These variables are the initial population(n_{pop}), proportion of children (p_C), and mutation rate (μ). To better determine the values for these critical parameters one could to several optimizations using a varied range of these parameters in order to find the combination that fits best. The GA will be coupled together with the proxy model. As the previous section shows the proxy model is capable of replicating results from a numerical reservoir simulator. The ranges that have been tested can be seen in table 4.4.

Parameter	Values
n_{pop}	16, 32, 40, 64
p_C	0.25, 0.50, 0.75, 1.00
μ	0.01, 0.10, 0.25, 0.5, 0.75

TABLE 4.4: Test ranges for the key parameters of the genetic algorithm optimization

Population size

Each individual is represented by a chromosome, which is a vector containing 8 variables. Thus the population sizes that have been tested are all some integer multiplied with 8. The values that are to be tested are n_{pop} = [16, 32, 40, 64] = [2*n_{var}, 4*n_{var}, 5*n_{var}, 8*n_{var}] . It is worth noting that even though it is probable that the larger population to perform better, this will also lead to more evaluations that are needed. The total run time will be a function of the population size and the proportion of children, as both of these parameters are closely correlated to the function evaluation, which is the operation that is the most time consuming in the whole optimization process. Thus the total time spent will be taken into consideration as well as the optimized value.

While altering the population size, the other parameters have been constant for all the optimizations. These parameters have been set to

Parameter	Values
p_C	0.75
μ	0.25

TABLE 4.5: The values set for p_C and μ while changing the population size

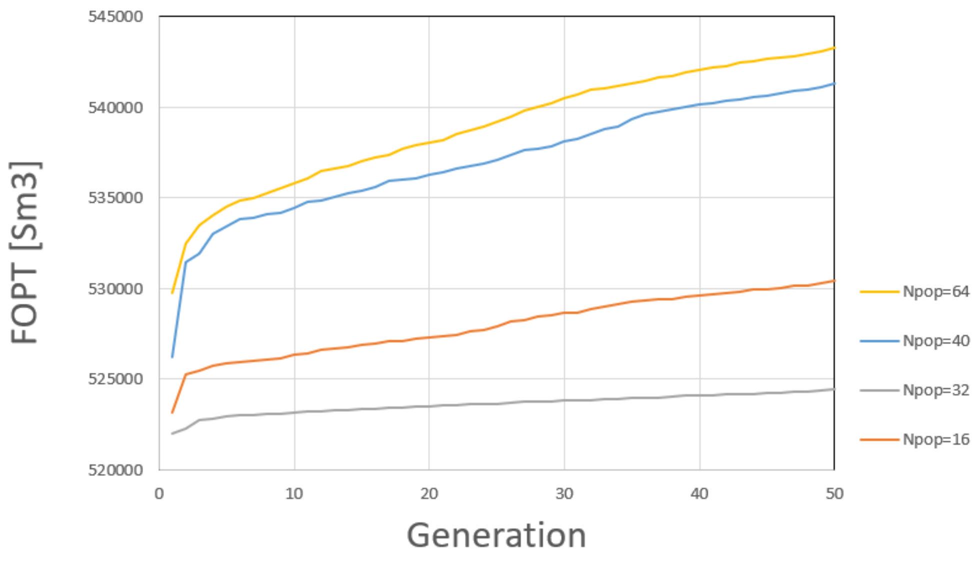

FIGURE 4.5: Results from performing optimization of FOPT using the proxy model with a varying n_{pop}.

Figure 4.5 shows the resulting optimized values for the FOPT estimated by the proxy model. As suspected the highest population is able to predict the highest value for the FOPT, which presumably will be closer to the absolute optimal value of the reservoir. As each individual chromosome is randomized, having a higher population will increase the probability of getting a good initial population. This is to some extent what is happening here. Curiously, the optimization that had an initial population of 32 has a lower FOPT than the optimization with an initial population of 16. This may be purely incidental, or it may because a population of 32 is still too low of a population to prevent the initial population of having "bad genes". From the figure it is a clear difference from n_{pop} = 16 and n_{pop} = 32 and up to the larger initial populations, which may indicate that a larger initial population leads to an overall better performing optimization algorithm.

Having a higher population is also causing the total run time to become longer(table 4.6). This is especially highlighted in table 4.7, which contains the time spent with p_C=1. As the overall optimal FOPT from the optimizations using n_{pop}=40 and n_{pop}=64 are quite close, it might be more optimal to use the a population of 40 due to the time saved by having a more moderate population.

n_{pop}	**Run time [s]**	**Run time**	**FOPT**[Sm^3]
16	68	1 minute and 8 seconds	5.30 x 10^5
32	130	2 minutes and 10 seconds	5.24 x 10^5
40	200	3 minutes and 20 seconds	5.41 x 10^5
64	315	5 minutes and 15 seconds	5.43 x 10^5

TABLE 4.6: Run times for varying populations. p_c = 0.75, μ = 0.25

n_{pop}	**Run time [s]**	**Run time**
16	104	1 minute and 44 seconds
32	198	3 minutes and 18 seconds
40	244	4 minutes and 4 seconds
64	387	6 minutes and 27 seconds

TABLE 4.7: Run times for varying populations. p_c = 1.0, μ = 0.25

Proportion of children

For this testing the proportion of children had the values of p_C = [0.25, 0.50, 0.75, 1.00], while the other parameters stayed the same for each of the different values of p_C (table 4.8) . The results can be seen on figure 4.6.

Parameter	**Values**
n_{pop}	40
μ	0.25

TABLE 4.8: The values for n_{pop} and μ during the testing using different values for p_C.

p_C	**FOPT**[Sm^3]
0.25	5.24 x 10^5
0.50	5.34 x 10^5
0.75	5.41 x 10^5
1.00	5.38 x 10^5

TABLE 4.9: Cumulative oil production of the various values of p_C

From the plot in figure 4.6 and values from table 4.9 it is clear that the final result is more optimized for a higher p_C, while it seems like p_C=0.75 is performing better

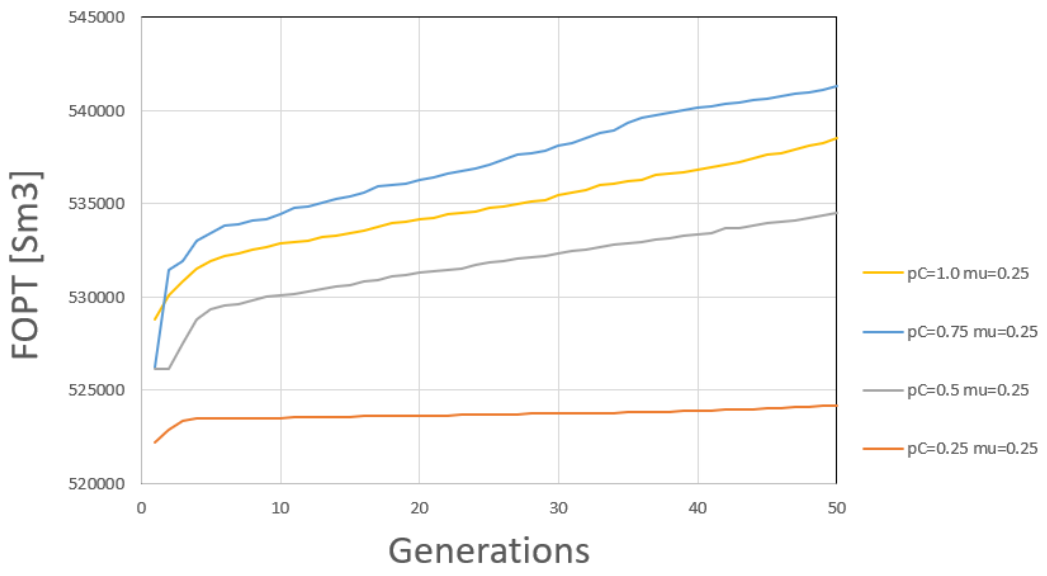

FIGURE 4.6: Results from performing optimization of FOPT using the proxy model with a varying p_C.

as compared to p_C=1.0. It is often intuitive to assume that larger values of the parameters will correspond with a better performance, however the data for this test says otherwise. The GA has elitism, which means that it is not guaranteed that the offspring created each iterations will take part in the next generation. As we move on to the later generation the general population is assumed to have a quite good average fitness. Thus, newly added offspring with potentially random genes are more likely to be outperformed by the overall population with a high average fitness. Thus having too high of a proportion of children seems to lead to diminishing returns. This doesn't necessarily mean that p_C =0.75 is objectively better than p_C=1, but as a higher p_C also makes the algorithm run slower it will definitely not optimal to use p_C=1 for optimizations in this specific case.

Mutation rate

The different mutation rates that have been tested are μ = [0.01, 0.10, 0.25, 0.5, 0.75]. In some cases mutation rate may cause a worse convergence towards the optimal value, as sometime a straight up crossover operation will provide better offspring as compared to a mutated offspring. The mutation rate could be viewed as a sort of step size for the GA, and sometime a smaller stepsize is preferred and sometimes a larger stepsize is preferred. The other parameters were kept constant (see table 4.10).

Parameter	**Values**
n_{pop}	40
p_C	0.75

TABLE 4.10: The values for n_{pop} and p_C during the testing using different values for μ.

The results from running the GA with a variation of mutation rates show an interesting result, with the mutation rates of μ=0.75 and μ=0.25 being the prominent

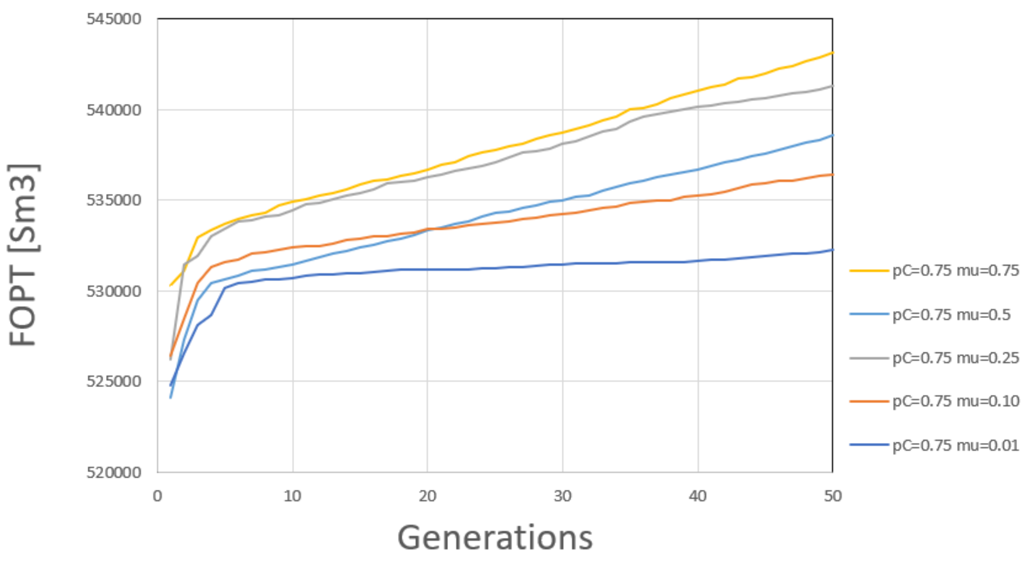

FIGURE 4.7: Results from performing optimization of FOPT using the proxy model with a varying μ

μ	**FOPT**$[Sm^3]$
0.01	5.32 x 10^5
0.10	5.36 x 10^5
0.25	5.41 x 10^5
0.50	5.38 x 10^5
0.75	5.43 x 10^5

TABLE 4.11: Cumulative oil production of the various values of μ

mutation rates. Even though the optimization using μ = 0.50 performs worse compared to the two mutation rates mentioned it has a consistent improvement for each generation. Deciding which mutation rate is the best could have more to do about the improvement of FOPT per generation rather than the overall optimized value, and thus any of the three highest scoring mutations rate will probably perform well in an optimization. The lower values for mutation rate clearly have quite a bit lower growth per generation.

4.2.1 Comparison

So far the GA parameters have been tested with the proxy model. This subsection will outline a comparison between the proxy model and ECLIPSE using the parameters specified. The parameters used are the ones in table 4.12. Even though a higher n_{pop} was found to be better, it was decided that a population of 16 will do an adequate job in this case as the simulation on ECLIPSE takes so much time anyways.

The plot in figure 4.8 shows the FOPT optimized by both the ECLIPSE and the proxy model for the mentioned parameters, and figure 4.9 shows the progression of the injection rates for each iteration of the GA. The most notable difference is the time spent on the optimization algorithm(table 4.13). With the proxy model it took only 71 seconds, while with ECLIPSE the algorithm took over 18.5 hours. This means that the GA took about 940 times longer with ECLIPSE compared to with the proxy model, or alternatively the proxy model provided a time save of 99.8%. This

Parameter	Value
Generations	50
n_{pop}	16
p_C	0.75
μ	0.75

TABLE 4.12: The parameters used for both ECLIPSE and the proxy model for comparison

is substantial, and it shows that using a proxy model to perform the optimization of injection rates is far superior compared to using an industry standard numerical reservoir simulator.

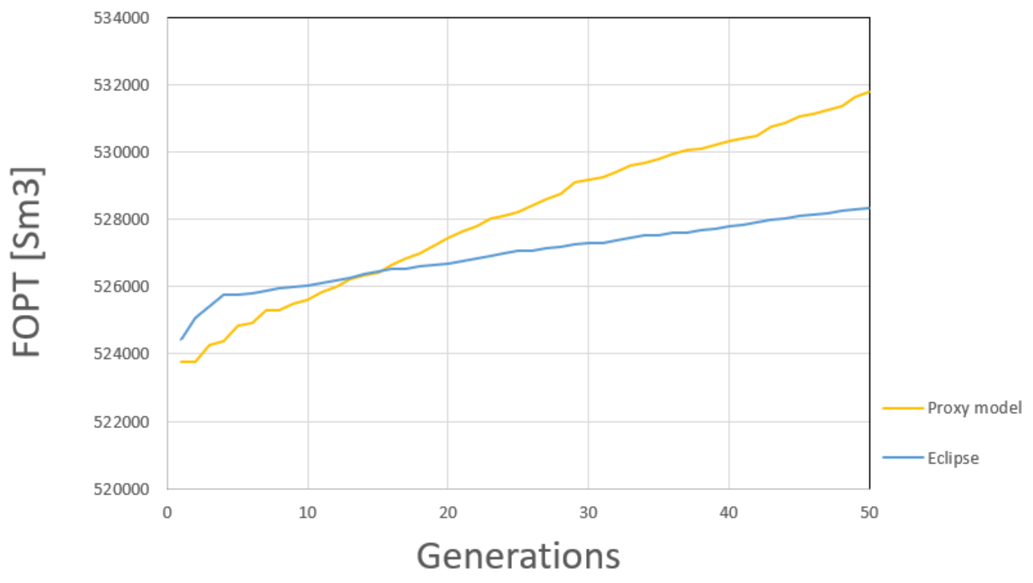

FIGURE 4.8: Evolution of the cumulative oil production for the proxy model and ECLIPSE over the GA optimization

Model	Time [s]	Time	FOPT[Sm^3]
Proxy model	71	1 minute and 11 seconds	5.31 x 10^5
ECLIPSE	66750	18 hours, 32 minutes, and 30 seconds	5.28 x 10^5

TABLE 4.13: Values from the GA with the proxy model and ECLIPSE

Also it is clear that the proxy model achieved a higher optimized value for the FOPT compared to ECLIPSE, both from looking at figure 4.8 and at the values from table 4.12. This is due to the proxy model over-predicting slightly for higher values of the injection rates. For validation propose the injection rates for the optimized value of the proxy model was used as input for the ECLIPSE reservoir simulator. The result of this simulation was a FOPT of **5.27 x** 10^5 Sm^3. From this evaluation it can be concluded that the optimized value from the GA using the proxy model had an over-prediction of 0.75%, and the real value of FOPT for the optimal injection rates is actually lower than the optimized value that the GA achieved when it used the ECLIPSE.

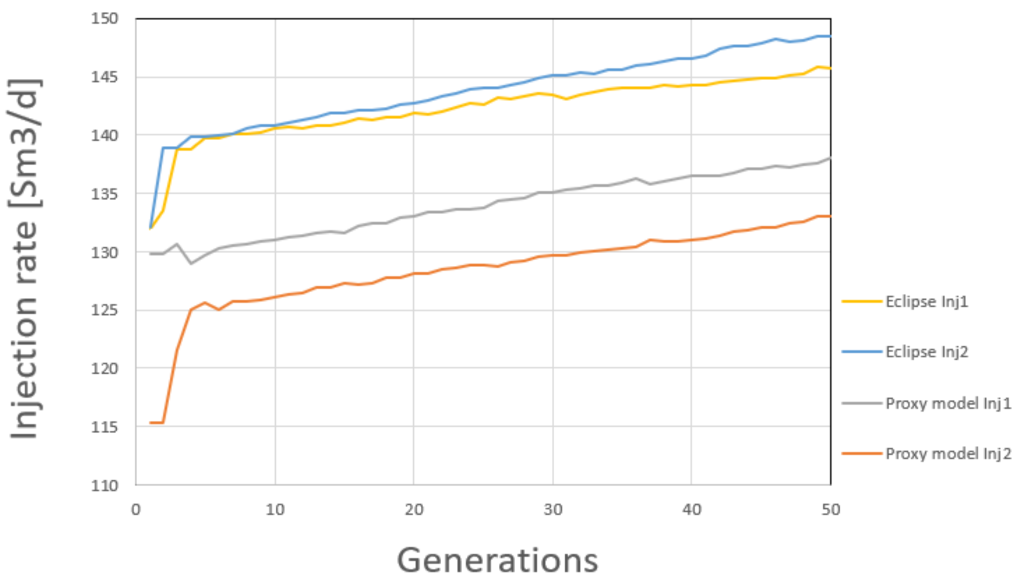

FIGURE 4.9: Evolution of the injection rates for the proxy model and ECLIPSE during the GA optimization

4.3 Shortcomings

Overall both the proxy model and the GA achieved what was expected. The proxy model is a good alternative or at least an assisting tool to numerical reservoir simulation, and the GA proved good at searching for an optimal value for the injection rates. It is worth noting the things that could have been better.

The proxy model developed in this project was over-predicting for higher values of the injection rates for the two injector well clusters. This may be due to a couple of reasons, on of them because the model is not adept enough for such high values. One method of mediating this may be to collect more samples for higher values of injection rates, even for values that may possibly exceed the interval that was initially specified. Using the GA to optimize the injection rates, the rates seemed to generally move towards the upper section of the samples interval. It is naturally excepted that higher values for the injection rate of water should result in more oil produced in total. This could be compensated for, either by enlarging the samples interval such that higher injection rates had better coverage or by simply moving the sampled interval higher in general. The proxy model might have had a better performance on the higher values of the injection rates if this was taken into consideration

For this project the ANN that constitutes the proxy model consists of so called *Dense*-layers in the keras model, which is the standard hidden layer in the context of neural networks. A dense layer simply takes input data in and returns a single value. But since the proxy model was supposed to predict the oil production rate for each time step it could have been beneficial to use another type of network such that the output of one evaluation can be used as another input for the next one. This could have been done by developing a recurrent neural network, where the evaluation at one timestep gets used as an input parameter for the next evaluation. Having a model such as this may provide better results for a regressional neural network such as this proxy model is.

To ease the process of developing a proxy model all the injection wells for the synthetic reservoir in this project were separated into two clusters of wells, where the injection rate of each such cluster was assumed to be constant for the entire production period. This is a simplification that limits the applicability of the proxy model. The model is also less versatile. With each well ingrained in a cluster it becomes difficult to retire one of the wells before the entire cluster, which may be an area that is worth exploring.

Chapter 5 Conclusions

The goal of this thesis was to examine the development of a proxy model for a synthetic reservoir using artificial intelligence, and then use this model for optimization of the oil production. This chapter aims at providing a conclusion based on the study to answer the questions raised in the dissertation.

Artificial intelligence based proxy models

From the work done in this thesis it is apparent that a single artificial neural network developed with 30 samples through a LHS-design is capable of representing the field oil production rate of a simple, synthetic reservoir. The proxy model in this thesis was able to accurately mimic the responses of a numerical reservoir simulator on the synthetic benchmark reservoir model known as the egg model [17]. The mean squared error of the proxy model compared to a blind test was 0.785 $\frac{Sm^3}{d}$. The model had a slight overprediction by 0.75%. The performance of the proxy model is highly dependent on the quality of the sampling, as a proxy model with artificial neural networks is a data based model.

The proxy model was able to perform evaluations of the cumulative oil production of the synthetic reservoir This time save is one of the key attributes of using a proxy model in stead of, or together with, a numerical reservoir simulator.

Waterflooding

Waterflooding has a major impact on the production of a hydrocarbon reservoir. This is apparent in the study of the genetic algorithm parameters in section 4.2, where the value of the cumulative oil production has a large variation when the proxy model is doing model evaluation for injection rates within the specified range of injection rates.

Using a genetic algorithm for optimization

A genetic algorithm is a robust and reliable optimization algorithm. Mathematically it is almost impossible to achieve the absolute maximum value for a given problem, and that's why there's a need for optimization algorithms such as a genetic algorithm. For each generation, the GA is steadily increasing the fitness of the population, resulting in an increase in the optimal solution. As seen in section 4.2, even in scenarios where the GA was running with sub-optimal GA-parameters the value was increasing.

GA is a versatile optimization algorithm. For the work in this thesis a GA was used to optimize the cumulative oil production using both a proxy model and a numerical reservoir simulator. The code that was implemented in python was the same for both of the reservoir evaluation methods, with the only difference being how the algorithm calculated the fitness of the individuals in the population. All of the GA operations are also quite fast, as it was the reservoir evaluations that was mostly responsible for the time spent on the optimization. It is apparent why GA has become one of the most popular optimization algorithms, as it can be used for almost any optimization scenario.

A genetic algorithm is quite a fast optimization algorithm. The largest time sink is the evaluation of the objective function, and if the GA takes longer than desired it is simple to simplify the optimization, or execute an early shut down of the algorithm.

When implementing a genetic algorithm this is often done in a higher level programming language such as python or C++. The numerical reservoir simulator ECLIPSE is simple to work with using any of these programming languages, as both the input- and output files for ECLIPSE are easily manipulated with simple file management using any programming language.

Chapter 6

Future work

Much of the work done for this theses was done manually, while it could have been done more efficiently by automating the process. For the sampling process, the latin square governing the values of the injection rates for the samples was calculated using a python script. Then 30 individual input files for ECLIPSE was made, then the simulation was ran for each of the set of input files. It is possible to specify to ECLIPSE that an .RSM file should be generated, which is an output file containing the values for all output parameters specified in the input files for each simulation run. These output files for all the samples were collected and were manually compiled into a large database in a spreadsheet software. And all of these things were done manually, when they potentially could be automated. When the GA is running with ECLIPSE, there are functions in place that automatically make input files for the ECLIPSE simulator, and that extracts the data from these .RSM files. Data sampling is one of the most time consuming parts of making a proxy with ANNs, so automating this process will save a lot of valuable time. Even if the code for automation has to be developed there is definitely a potential of saving time. ECLIPSE is a versatile reservoir simulator where it is not too difficult to manipulate both the input- and output-files with python, so this versatility should be exploited to a larger extent.

This thesis highlights a concept within reservoir engineering using a simple model, and the concept can scale up. In this project there was a fairly simple reservoir, where the position of the wells were un-altered from the one provided as a benchmark model in the paper [17]. The proxy model had the purpose of predicting field oil production rate during a waterflooding process, but potentially it could have done a lot more. There is the possibility that there could be a proxy model estimating the water production as well, or it could be used to model some other EOR process as well. Adding to this, the proxy model could have been coupled together with another model in order to optimize the well placements, as with this thesis all the wells remained static throughout the entire project. ANN use data in order to produce more data, so the possibility of connecting several ANNs together could prove to solve optimization problem of a much higher degree of complexity than in this project.

The work for this thesis was done on a homogeneous, synthetic reservoir. As well as expanding the work laterally there is also the possibility of applying the concepts of this thesis to a real reservoir. Either by using a numerical reservoir model for a real reservoir and following the procedure of this thesis, or by using real data from a real field. The purpose of this thesis is to prove that this sort of method works for examining the optimization of a hydrocarbon reservoir, and thus there is much room for expansion. There is a major difference between these two methods of developing a proxy model. A proxy model developed using data gathered from a reservoir simulator will still be a result of simulated data, while a model developed using real data will provide a more realistic model. The proxy model made by synthetic data will incorporate both the errors that may occur during the model development together with errors that stem from the numerical reservoir simulator itself, which may be approximation to physical processes, or simply data lost due to computer rounding of data values.

The GA could be sued to optimize the NPV of the field instead of the cumulative oil production of the reservoir. NPV optimization has more of a tangible value as it encompasses the value of the hydrocarbons produced, as well as the costs of the injected/produced fluid. For a waterflooding process the injected water may not tip the scales as much, as water is quite inexpensive. For a more complex operation with a CO_2 injection or a surfactant/polymer injection it may be more costly to inject fluids into the reservoir, and thus the NPV optimization provides a better image of an optimized process.

Bibliography

[1] Henrik Ågrav. *Applications of proxy models in the petroleum industry - A literary review*. 2020.

[2] Abbas Alkhudafi. *Fundamentals of Petroleum Engineering Part 1 Exploration and Reservoir Engineering*. Dec. 2018.

[3] Shohreh Amini. “Developing a Grid-Based Surrogate Reservoir Model Using Artificial Intelligence”. In: *Graduate Theses, Dissertations, and Problem Reports. 5096* (2015).

[4] O Badru, CS Kabir, et al. “Well placement optimization in field development”. In: *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers. 2003.

[5] Rosemary A Bailey. *Design of comparative experiments*. Vol. 25. Cambridge University Press, 2008.

[6] Peter L Bonate. “A brief introduction to Monte Carlo simulation”. In: *Clinical pharmacokinetics* 40.1 (2001), pp. 15–22.

[7] George EP Box. “Science and statistics”. In: *Journal of the American Statistical Association* 71.356 (1976), pp. 791–799.

[8] Rafael E Campos and Jose A Hernandez. *In-situ reduction of oil viscosity during steam injection process in EOR*. US Patent 5,209,295. May 1993.

[9] B.H. Caudle and I.H. Silberberg. “Laboratory Models of Oil Reservoirs Produced By Natural Water Drive”. In: *Society of Petroleum Engineers Journal* 5.01 (Mar. 1965), pp. 25–36. ISSN: 0197-7520. DOI: 10.2118/984-PA. eprint: https://onepetro.org/spejournal/article-pdf/5/01/25/2159725/spe-984-pa.pdf. URL: https://doi.org/10.2118/984-PA.

[10] Shabab D.Mohaghegh. “Petroleum Data Analytics”. Texas A& M SPE Chapter. 2020.

[11] Jason G Digalakis and Konstantinos G Margaritis. “On benchmarking functions for genetic algorithms”. In: *International journal of computer mathematics* 77.4 (2001), pp. 481–506.

[12] Gijs van Essen et al. “Robust waterflooding optimization of multiple geological scenarios”. In: *Spe Journal* 14.01 (2009), pp. 202–210.

[13] Aarushi Gupta, Utkarsh Soumya, et al. “Well Log Interpretation Using Deep Learning Neural Networks”. In: *International Petroleum Technology Conference*. International Petroleum Technology Conference. 2020.

[14] Abdolhossein Hemmat-Sarapardeh et al. *Applications of artificial intelligence techniques in the petroleum industry*. Gulf Professional Publishing, 2020.

[15] TA Hjelmas et al. "Produced water reinjection: experiences from performance measurements on Ula in the North Sea". In: *SPE Health, Safety and Environment in Oil and Gas Exploration and Production Conference*. Society of Petroleum Engineers. 1996.

[16] F Owen Hoffman and Jana S Hammonds. "Propagation of uncertainty in risk assessments: the need to distinguish between uncertainty due to lack of knowledge and uncertainty due to variability". In: *Risk analysis* 14.5 (1994), pp. 707–712.

[17] Jan-Dirk Jansen et al. "The egg model–a geological ensemble for reservoir simulation". In: *Geoscience Data Journal* 1.2 (2014), pp. 192–195.

[18] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[19] A Kohli and P Arora. "Application of artificial neural networks for well logs". In: *IPTC 2014: International Petroleum Technology Conference*. European Association of Geoscientists & Engineers. 2014, cp–395.

[20] Larry W Lake. "Enhanced oil recovery". In: *Old Tappan, NJ; Prentice Hall Inc.* (1989).

[21] R Markovinović and JD Jansen. "Accelerating iterative solution methods using reduced-order models as solution predictors". In: *International journal for numerical methods in engineering* 68.5 (2006), pp. 525–541.

[22] Calvin C. Mattax and Robert L. Dalton. "Reservoir Simulation (includes associated papers 21606 and 21620)". In: *Journal of Petroleum Technology* 42.06 (June 1990), pp. 692–695. ISSN: 0149-2136. DOI: `10.2118/20399-PA`. eprint: `https://onepetro.org/JPT/article-pdf/42/06/692/2222831/spe-20399-pa.pdf`. URL: `https://doi.org/10.2118/20399-PA`.

[23] Maria Aparecida de Melo et al. "Evaluation of polymer-injection projects in Brazil". In: *SPE Latin American and Caribbean Petroleum Engineering Conference*. Society of Petroleum Engineers. 2005.

[24] Shahab Mohaghegh et al. "Virtual-intelligence applications in petroleum engineering: Part 1—Artificial neural networks". In: *Journal of Petroleum Technology* 52.09 (2000), pp. 64–73.

[25] Shahab Mohaghegh et al. "Virtual-intelligence applications in petroleum engineering: Part 2—evolutionary computing". In: *Journal of Petroleum Technology* 52.10 (2000), pp. 40–46.

[26] Michael Nielsen. *Neural Networks and Deep Learning*. URL: `https://www.http://neuralnetworksanddeeplearning.com` (visited on 10/17/2020).

[27] W Terry Osterloh et al. "Use of multiple-response optimization to assist reservoir simulation probabilistic forecasting and history matching". In: *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers. 2008.

[28] *Pandas*. URL: `https://pandas.pydata.org/`.

[29] Øystein Pettersen. *Basics of Reservoir Simulation With the Eclipse Reservoir Simulator*. Dept. of Mathematics, University of Bergen, 2006.

[30] Andrei Sergiu Popa, Kailash Sivakumar, Stephen David Cassidy, et al. "Associative data modeling and ant colony optimizaton approach for waterflood analysis". In: *SPE Western Regional Meeting*. Society of Petroleum Engineers. 2012.

[31] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Swish: a self-gated activation function". In: *arXiv preprint arXiv:1710.05941* 7 (2017).

[32] Schlumberger. *ECLIPSE 2020.4*.

[33] *Scikit-learn*. URL: `https://scikit-learn.org/stable`.

[34] Michael D Shields and Jiaxin Zhang. "The generalization of Latin hypercube sampling". In: *Reliability Engineering & System Safety* 148 (2016), pp. 96–108.

[35] David Silver et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv preprint arXiv:1712.01815* (2017).

[36] J.T. Smith, W.M. Cobb, and Petroleum Technology Transfer Council (U.S.). Midwest Office. *Waterflooding*. Midwest Office of the Petroleum Technology Transfer COuncil, 1997. URL: `https://books.google.no/books?id=cqx5HAAACAAJ`.

[37] M.B Standing. "Notes on relative permeability relationships". In: *Division of Petroleum Engineering and Applied Geophysics. The Norwegian Institute of Technology* (August 1974).

[38] *The Alphabet Soup of IOR, EOR and AOR: Effective Communication Requires a Definition of Terms*. Vol. All Days. SPE International Improved Oil Recovery Conference in Asia Pacific. SPE-84908-MS. Oct. 2003. DOI: `10.2118/84908-MS`. eprint: `https://onepetro.org/SPEAPIOR/proceedings-pdf/03IIORC/All-03IIORC/SPE-84908-MS/1867477/spe-84908-ms.pdf`. URL: `https://doi.org/10.2118/84908-MS`.

[39] *TensorFlow*. URL: `https://www.tensorflow.org/`.

[40] M. Duran Toksari. "A hybrid algorithm of Ant Colony Optimization (ACO) and Iterated Local Search (ILS) for estimating electricity domestic consumption: Case of Turkey". In: *International Journal of Electrical Power Energy Systems* 78 (2016), pp. 776–782. ISSN: 0142-0615. DOI: `https://doi.org/10.1016/j.ijepes.2015.12.032`. URL: `https://www.sciencedirect.com/science/article/pii/S0142061515005840`.

[41] Chun-Wei Tsai et al. "A high-performance genetic algorithm: using traveling salesman problem as a case". In: *The Scientific World Journal* 2014 (2014).

[42] Anant J Umbarkar and Pranali D Sheth. "Crossover operators in genetic algorithms: a review." In: *ICTACT journal on soft computing* 6.1 (2015).

[43] Felipe AC Viana. "A tutorial on Latin hypercube design of experiments". In: *Quality and reliability engineering international* 32.5 (2016), pp. 1975–1985.

[44] Curtis Whitson and M. Brule. "Phase Behavior Monograph". In: *Soc. Petrol. Eng.* 20 (Jan. 2000).

[45] Kurt Wilson, Louis J Durlofsky, et al. "Computational optimization of shale resource development using reduced-physics surrogate models". In: *SPE Western Regional Meeting*. Society of Petroleum Engineers. 2012.

[46] Burak Yeten et al. "A comparison study on experimental design and response surface methodologies". In: *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers. 2005.

[47] Xin Zhang, Dexuan Zou, and Xin Shen. “A Novel Simple Particle Swarm Optimization Algorithm for Global Optimization”. In: *Mathematics* 6.12 (2018). ISSN: 2227-7390. DOI: 10.3390/math6120287. URL: https://www.mdpi.com/2227-7390/6/12/287.