Master's thesis

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

Siri Holde Hegsvold

Uncertainty estimation in Neural Networks

Master's thesis in Cybernetics & Robotics
Supervisor: Professor Tommy Gravdahl
Co-supervisor: Esten Ingar Grøtli, Mark Haring
August 2022

NTNU
Norwegian University of
Science and Technology

Siri Holde Hegsvold

Uncertainty estimation in Neural Networks

Master's thesis in Cybernetics & Robotics
Supervisor: Professor Tommy Gravdahl
Co-supervisor: Esten Ingar Grøtli, Mark Haring
August 2022

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

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Uncertainty Estimation in Neural Networks

Siri Holde Hegsvold

Master thesis
Main supervisor: Professor Tommy Gravdahl
Co-supervisor 1: Esten Ingar Grøtli
Co-supervisor 2: Mark Haring
Submission date: August 21, 2022

Department of Engineering Cybernetics
Norwegian University of Science and Technology

Abstract

This master thesis will look into methods for uncertainty estimation of neural networks. To narrow the scope, the application area for these neural networks is land-based process industries. The main focus will be the uncertainty of data as inputs to the neural networks (aleatoric), not the uncertainty of the model itself (epistemic). The purpose of this article is to provide a measure of predicted uncertainty that is necessary to integrate deep learning algorithms in safety-critical applications. This is important to move the process industry from the traditional way of running their process to a more efficient and forward-looking one. Uncertainty estimation is also important to investigate carefully before implementing such technology into extensive physical processes to ensure nothing goes wrong. However, these industries have highly complex dynamics, and the composition materials are not always known. This means that it is hard to get the information and knowledge needed to derive a model traditionally. To address this, various uncertainty estimation methods will be compared based on assumptions and the performance of static feedforward neural networks. To simplify this, a benchmark is made as ground truth, so that developed methods can be compared to this. I found that Monte Carlo sampling and Gaussian quadrature is good approaches for uncertainty estimation and performs well compared to the benchmark. However, the sampling method has a large computational cost compared to the Gaussian quadrature. Linearization performs quite poorly. This means that the Gaussian quadrature method can be used in future research to decide if it is a good approach in neural networks and for real-life applications. Future research should identify the challenges of applying this model to a dynamical state-space model based on neural networks. A method for overcoming the challenges should be implemented, and finally, the developed uncertainty estimation method for the dynamical state-space model should be compared.

One-liner: Using traditional control theory and statistical methods, this paper investigates methods for uncertainty estimation of neural networks to include more machine learning in the decision-making of land-based process industry, showing that Gaussian quadrature and Monte Carlo sampling can be advantageous in further research compared to the simpler methods, such as linearization.

Keywords: Artificial neural networks, uncertainty estimation, process industry, Monte

Carlo sampling, linearization, Gaussian quadrature

Sammendrag

Denne masterrapporten vil se på metoder for usikkerhetsestimering av nevrale nettverk. For å begrense omfanget er bruksområdet for disse nevrale nettverkene landbaserte prosessindustrier. Hovedfokuset vil være usikkerheten til data som inngang til de nevrale nettverkene (aleatorisk), ikke usikkerheten til selve modellen (epistemisk). Hensikten med denne artikkelen er å gi et mål på predikert usikkerhet som er nødvendig for å fullt ut integrere dyplæringsalgoritmer i sikkerhetskritiske applikasjoner. Dette er viktig for å flytte prosessindustrien fra den tradisjonelle måten å drive prosessen på til en mer effektiv og fremtidsrettet. Et usikkerhetsestimat er også viktig å ha før slik teknologi implementeres i omfattende fysiske prosesser for å sikre at ingenting går galt. Imidlertid har disse bransjene svært kompleks dynamikk og sammensetningsmaterialene som brukes er ikke alltid kjent. Dette betyr at det er vanskelig å få den informasjonen og kunnskapen som trengs for å utlede en modell tradisjonelt. For å løse dette vil ulike metoder for usikkerhetsestimering bli sammenlignet basert på forutsetninger og ytelsen til statiske feedforward nevrale nettverk. For å forenkle dette lages en modell som skal gjenspeile sannheten, slik at utviklede metoder kan sammenlignes med dette. Jeg fant at Monte Carlo-sampling og Gaussisk kvadratur er en god tilnærming for usikkerhetsestimering og gir gode resultater sammenlignet med benchmark. Samplingsmetoden har imidlertid en stor beregningskostnad i motsetning til den Gaussiske kvadraturen. Linearisering, fungerer ganske dårlig. Dette betyr at den Gaussiske kvadraturmetoden kan brukes i fremtidig forskning for å avgjøre om det er en god tilnærming i nevrale nettverk og for virkelige applikasjoner. Fremtidig forskning bør identifisere utfordringene ved å bruke denne modellen på en dynamisk stat-rom-modell basert på nevrale nettverk. En metode for å overvinne utfordringene bør implementeres, og til slutt bør den utviklede usikkerhetsestimeringsmetoden for den dynamiske tilstand-rom-modellen sammenlignes.

En-linje: Ved å bruke tradisjonell kontrollteori og statistiske metoder, undersøker denne artikkelen metoder for usikkerhetsestimering av nevrale nettverk med det formål å inkludere mer maskinlæring i beslutningstakingen av landbasert prosessindustri, og viser at Gaussisk kvadratur og Monte Carlo-prøvetaking kan være fordelaktige i videre forskning sammenlignet med de enklere metodene, som linearisering.

Nøkkelord: Kunstige nevrale nettverk, usikkerhetsestimering, prosessindustri, Monte Carlo sampling, linearisering, gaussisk kvadratur

Preface

This thesis is submitted for the degree of Master of Technology in Engineering Cybernetics, course code TTK4900, to the Department of Engineering Cybernetics at the Norwegian University of Science and Technology. The work was carried out during the spring semester of 2022. The work presented in this thesis has been carried out under the supervision of Professor Tommy Gravdahl at NTNU and Esten Ingar Grøtli and Mark Haring at SINTEF Digital.

The thesis is in collaboration with SINTEF Digital project "Towards autonomy in process industries: Combining data-driven and model-based methods for estimation and control (TAPI)". Beside SINTEF and NTNU, there are four industry partners, Hydro, Elkem, Yara and Borregaard.

This master's thesis is a continuation of a specialization project I conducted during the autumn of 2021. As is customary, the specialization project is not published. This means that important background theory, methods, and results from the project report will be restated in full throughout this report to provide the best reading experience. Below a complete list of the material included from the specialization project is listed.

- Chapter 1 (Some changes to section 1.1)
- Chapter 2 (Specifically sections 2.2, 2.2.1, 2.2.2, and 2.4, with some changes to section 2.1, 2.1.1, and 2.3.2,)
- Chapter 3 (Specifically sections 3.2 with some changes to section 3.1, 3.1.1, 3.3, 3.5.1, 3.5.2, and 3.5.3)
- Chapter 4 (Specifically sections 4.1, and 4.2)
- Chapter 5 (Specifically sections 5.1, and 5.2, with some changes to section 5.4)

Code for the experiments in this thesis was primarily written in the Python v3.9.7 [22] programming language. The most used packages are the *NumPy* library, which is used for scientific computing, the *Mathplotlib* library for plot creations, the *Math*

library for mathematical functions, and the *SymPy* for symbolic mathematics. The details about the neural network implemented in chapter 3.6 was provided to me by Mark Haring.

The software framework developed for the thesis will be shared upon request. Please contact the author.

Unless otherwise stated, all figures and illustrations have been created by the author.

I would like to thank my supervisors, Professor Tommy Gravdahl at NTNU and Esten Ingar Grøtli and Mark Haring at SINTEF Digital, for providing valuable feedback and discussions throughout the two semesters.

Lastly, I am very grateful for all the moral support from my family during this period.

Siri Holde Hegsvold August 21, 2022

Contents

List of Figures

Acronyms

AI Artificial Intelligence. 1, 2, 9, 10

DL Deep Learning. 1, 2, 5, 7, 9, 10, 77

DNN Deep Neural Network. 1, 2, 12, 17, 76

ML Machine learning. 1, 2, 9

XAI Explainable Artificial Intelligence. 1, 10

MC Monte Carlo. 3, 4, 13, 14, 21, 27, 32–34, 36–38, 46, 55, 66, 72–76

ANN Artificial neural networks. 5

ReLU Rectified Linear Unit. 8, 35

ELU Exponential Linear Unit. 8, 35

EKF Extended Kalman Filter. 12, 13

PDF Probability Density Function. 17, 18, 24

ADF Assumed Density Filtering. 76

Chapter 1

Introduction

Artificial Intelligence (AI) has left its mark in developing new technology for the last few years [3]. Neural networks and the field of Deep Learning (DL) are an essential part of the development, and new models are constantly emerging that can be used in new areas. In 2006 the "third wave" of neural networks research began, and this wave is the reason why training of neural networks is known as DL. The researchers were now able to train deeper neural networks than they had been able to do earlier, which gave focus to the theoretical importance of depth. The depth of the model and increase in the size of the dataset are essential factors that have led to Deep Neural Network (DNN) outperforming AI systems and more traditional Machine learning (ML) models.

1.1 Background and motivation

This section is based on section 1.1 in [26]. A crucial feature in today's practical deployment of AI models is the barrier of explainability. The paradigms underlying this barrier fall within the field of Explainable Artificial Intelligence (XAI), a field working toward responsible AI for large-scale implementation of AI methods with fairness, model explainability, and accountability at its core. To achieve a level of human understanding of AI, a collective term called XAI is defined for the learning techniques where the goal is to improve trustworthiness, transferability, confidence, and uncertainty. This project focuses on the latter, namely uncertainty [3].

There will always be uncertainty, small or large, associated with the decision-making process, and there are no exceptions for uncertainty related to scientific decision-making. Decision-making associated with DL is defined as an automated decision-making process. Since this process is automated, there is no guarantee that wrong choices get caught before they affect a larger system. The focus in most ML and DL applications is the optimization of performance, which is rarely a good indicator

of trust [19]. However, a good performance can be misleading and give too much confidence in the model. The increased available data and model complexity can provide a very accurate model. However, they can still be prone to rely on spurious correlations, magnify bias, and draw conclusions that do not consider the dynamics in a system [19].

To illustrate the importance of uncertainty estimation in AI, it can be relevant to move outside the scope of safety-critical application. Autonomous driving or the field of medicine is often used to demonstrate the importance of uncertainty estimation. However, it can also be crucial in application areas that are not always expected. An example is when Google Photos labeled black people as gorillas in their image recognition algorithm, leading to accusations of racial discrimination [30]. Suppose the algorithm used in this example had considered the prediction's uncertainty, and the classification was assigned a high level of uncertainty to the wrong prediction. In that case, the system may have avoided the disaster.

Problem formulation

The lack of knowledge about the uncertainty of the predictions made from DNN limits the areas for which it may be applied. The land-based process industry is an example of an application limited by this lack of knowledge. This has led to a collaboration between the Norwegian University of Science and Technology (NTNU) and SINTEF Digital, including four industry partners, Hydro, Elkem, Yara, and Borregaard. The goal is to investigate how to safely combine traditional control theory and DL algorithms in the process industry. This project goes by the name: "Towards autonomy in process industries: Combining data-driven and model-based methods for estimation and control (TAPI)". The goal of this project is to increase productivity and competitiveness. Many ML algorithms consisting of data-driven models have shown sufficient performance in many sectors. However, that does not mean they can be adapted directly to the process industry. This industry comes with many challenges and requirements, such as stability and sparse manual measurements. Another problem is that ML is not based on the laws of physics as traditional control methods, and it lacks mathematical analysis [27].

This thesis will therefore look into some methods for uncertainty estimation of neural networks. The methods considered are based on estimating aleatoric uncertainty of a single input - single output perspective, where the input is a stochastic nonlinear function. The land-based process industry will be considered as the application together with general use when analyzing the advantages and disadvantages of the different methods.

1.2 Objectives

The objectives for this project are summarized in the following research questions.

1. Develop various uncertainty estimation methods for nonlinear functions
2. Compare the developed methods based on made assumptions and performance
3. Test each developed method on a static feed-forward neural network and compare the methods
4. Identify areas for future research on uncertainty estimation for neural networks

1.3 Approach

The approach for the first objective is to develop benchmark models that analytically compute the output mean and variance. The benchmark for mean and variance will be tested and compared to the methods developed for calculating the output variance of a nonlinear function. The models for uncertainty estimation investigated in this thesis are linearization, Monte Carlo (MC) sampling, and Gaussian quadrature.

The second objective, which is to compare the developed methods based on made assumptions and performance, will be to evaluate the results presented together with the theory about the methods. The methods will be compared to a benchmark model representing the ground truth.

The third objective is to test the methods developed in the first objective on a static feed-forward neural network. To do this, a new benchmark for mean and variance is made, so new experiments have some form of validation. This also makes it possible to compare and discuss the results. Since the calculations of the neural networks' output variance are often computationally heavy, a goal is to find methods with a low computational cost that also have an accurate approximation. This approach makes it possible to use the techniques for real-time applications in estimation and control.

The approach for the fourth objective is to discuss recommendations for further work based on the results and the following discussion of the results.

1.4 Contributions

The first part of this report focuses on giving the relevant background information. This includes an overview of existing literature and tools to understand the uncertainty estimation of neural networks. In the second part, the main contribution is

investigating and developing methods to estimate nonlinear systems' output mean and variance. These methods will again be used as inputs to a neural network. The result will be presented and discussed before a conclusion is made.

1.5 Outline

This thesis is divided into six main chapters.

It starts in Chapter 2 by introducing the theoretical framework, which begins with a brief explanation of the building blocks of a neural network. Then the theoretical framework for uncertainty in neural networks is presented. This starts with information about why it is necessary to estimate uncertainty and different methods to perform uncertainty estimation. Then some fundamental statistic is given to lay the foundation for upcoming chapters.

Chapter 3 presents the methodology. The methods implemented are described. These methods are linearization, MC sampling, and, Gaussian quadrature. A benchmark is also presented in this chapter. This is made as ground truth and is used to study the performance of the methods. Some plots are generated to view the comparison between the benchmark and developed methods. There will also be a description of how this is done in this section. Then the neural network implementations are presented. After details about the network are given, a deeper explanation of how each of the three methods implemented can be used together with this neural network is explained.

Chapter 4 presents the results. The methods implemented are applied to different nonlinear functions to see when and where it has good performance and when the method is insufficient. After these results for each method alone are presented, the method in combination with the neural network is studied.

Chapter 5 discuss the result presented in chapter 4.

Finally, chapter 6 concludes the work and presents recommendations for further work.

Chapter 2

Background

This chapter will provide the theoretical framework necessary for the upcoming chapters. The first sections give information about neural networks. Then, some underlying theory about this field is necessary for understanding upcoming chapters about applying uncertainty estimation of a neural network. The following three sections present theory about uncertainty estimation. Both methods and necessary statistics.

2.1 Artificial neural networks

This section is based on section 2.1 in [26]. Artificial neural networks (ANN) is the definition of some of the first learning algorithms, which are intended to be computational models of the brain, or what's called biological learning. Taking inspiration from the brain was mainly motivated by two reasons. The first reason was the interest in understanding the human brain and the principles of human intelligence. Secondly, the brain was watched as proof that intelligent behavior is possible. By reverse-engineering the principle behind the brain, scientists could duplicate its computational functionality into engineered systems [10]. However, these ANN can only be viewed as very simplified computational models of biological networks, because it is nearly impossible to simulate the number of neurons and their interconnections, and our knowledge of their operations is not fully understood [4].

A network composed of artificial neurons organized into multiple layers is called DL. DL solves a central problem in representation learning. Representational learning, also known as feature learning, is learning features of input data done either by transforming the data or extracting features from the data. This approach enables the computer to build complex concepts out of simpler concepts [10].

2.1.1 Artificial neurons - Perceptrons

This section is based on section 2.1.1 in [26]. An important key idea about neural networks is the components they consist of. As mentioned in the brief introduction above, a network is composed of artificial neurons. Artificial neurons come in different types based on different mathematical models. In the 1950s an artificial neuron called a perceptron was developed and has since then gone by the name "the simplest learning machines" [17]. Perceptrons are not very common to use in modern networks, however, their mathematical model has been important for the development of modern methods and is therefore important to understand [18].

A perceptron is neuron that takes in one or several inputs, $x_1, x_2, ..., x_n$ and outputs one binary value. Each of the inputs to the network is weighted by a value, $w_1, w_2, ..., w_n$, that says something about the importance of the corresponding input. Based on the combination of the input and the weight, the output of the perceptron is either 0 or 1. Whether or not it is a 0 or a 1 is decided by a chosen real-valued threshold. The mathematical description of the following is presented below:

$$\begin{cases} 0 & \sum_j w_j x_j \leq threshold \\ 1 & \sum_j w_j x_j > threshold \end{cases}$$

The binary property of the perceptron makes it a decision-making model, and as mentioned earlier, it can be compared to human decision-making on some levels. Different decision-making models can be made by varying the weights and thresholds. In Figure 2.1 a simple network with one hidden layer is illustrated. The hidden layer consists of five perceptrons each making a decision, by weighting the input. The output is then the sum of each of the decisions made in the hidden layer. By adding more layers between the input and output, making a many-layer network, an even more complex decision-making process is made.

However, the perceptrons do not extend to modern network models. As mentioned, adjusting the weights and thresholds can change the decision-making model. When working with highly complex models a small adjustment in weights or threshold for just a single perceptron in the network can cause the output of a neuron to flip, which again can give a propagating error through all layers in the network, resulting in a wrong output of the network. Correcting this error will be very complicated. It will therefore be necessary to use other artificial neurons than the perceptron to obtain the desired behavior when dealing with complicated network structures. To handle the flip in the neurons a nonlinear activation function is introduced [18]. It is this introduction of nonlinearities that contributes to the increase of complexity.

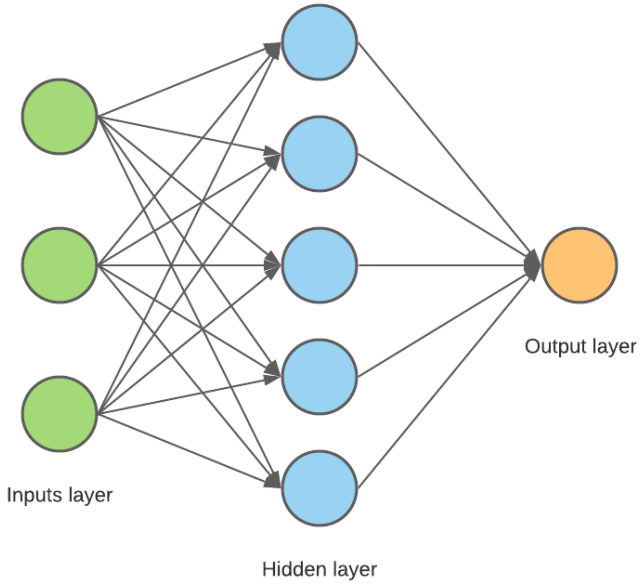


Figure 2.1: Simple illustration of a neural network. The hidden layer consists of five perceptrons, each with three inputs and one output.

2.1.2 Modern deep feedforward neural networks

Modern DL is a powerful framework that has the capability to handle a higher number of layers and more units within each layer. Deep networks can therefore represent functions of increasing complexity. These deep network models are called feedforward because information flows from the input function evaluated $\boldsymbol{x}$, through the computations used to approximate the function f, and ends up at the output $\boldsymbol{y}$. The goal of this model is then to approximate some function f^* by learning the mapping $\boldsymbol{y} = f(x; \boldsymbol{\theta})$. $\boldsymbol{\theta}$ is the parameterization of the network [10].

This mapping is often composed of many different maps/functions

$$f(x) = f^{(L)}(f^{(L-1)}(...(f^{(1)}x; \theta)))) \tag{2.1}$$

where $f^l, l \in 1, ..., L$ is a layer in the network. The number of maps in the composition gives the depth of the network. The feedforward network's final layer, L-th, is called the output layer, while the first layer is called the input layer. The layers between the input and output layers are called hidden layers. The reason for naming these layers "hidden", is because the usage of the layers is not given by the input data, but is decided by a learning algorithm to approximate f^*. The hidden layers are described by

$$z^{(l)} = f^{(l)} = \phi(W^{(l)} z^{(l-1)} + b^{(l)}) \tag{2.2}$$

where $W^{(l)}$ is the weight matrix and $b^{(l)}$ the bias vector. Both $W^{(l)}$ and $b^{(l)}$ is trainable parameters. ϕ is an activation function.

Each hidden layer consists of a number of units. The number of units says something about the width of the layer. Each unit i in a given layer l is described as

$$z_i^{(l)} = \phi(w_i^{(l)} z^{(l-1)} + b_i^{(l)}) \tag{2.3}$$

where $w_i^{(l)}$ are weights connection output of the previous layer with the unit in the current layer and b_i is the bias corresponding to the unit. ϕ still symbolizes the activation function [10].

Different types of activation functions exist and choosing the correct one is important to achieve the simplest derivative and a problem that is possible to compute analytically [1]. Examples of well-known activation functions are the Rectified Linear Unit (ReLU), which is defined as

$$ReLU(z) = max\{0, z\} \tag{2.4}$$

the sigmoid function is defined as

$$S(z) = \frac{1}{1 + e^{-z}} \tag{2.5}$$

and the hyperbolic tangent function is given by

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{2.6}$$

Another not so well known activation function is the Exponential Linear Unit (ELU) function

$$R(z) = \begin{cases} z \text{ for } z \geq 0 \\ \alpha(e^z - 1) \text{ for } z < 0 \end{cases} \tag{2.7}$$

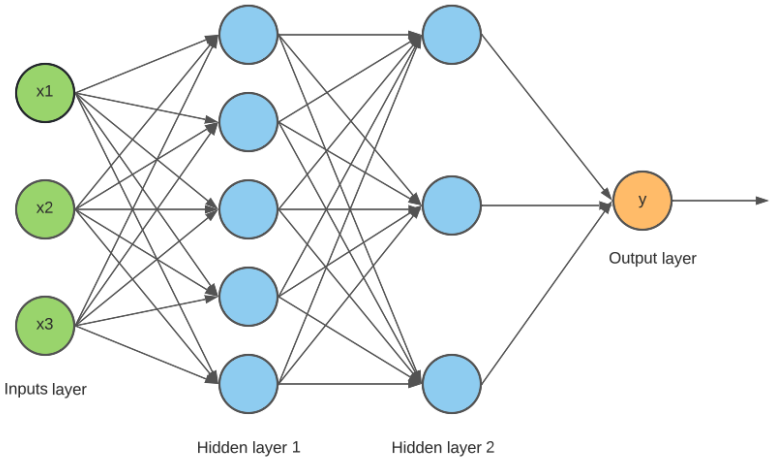


Figure 2.2

2.2 Uncertainty estimation

This hole chapter is a direct copy of chapter 2.2 in [26]. A crucial part of every decision-making process is uncertainty and there are no exceptions for uncertainty related to scientific decision-making. Computer scientists and software engineers typically work in somewhat clean and certain environments and therefore it can be surprising that the field of AI and ML is closely related to probability theory [10].

2.2.1 Overview: Uncertainty in neural network

Uncertainty estimation of neural networks is crucial for high-risk decision-making. The reason for this is because data going into the network can, for example, be corrupted by noise, or the input sample can be out of the training distribution. These weaknesses make the neural network predictions uncertain, which again can say something about possible failures that can happen at the output of the model [15]. When integrating AI algorithms into control theory or robotic applications, it is essential to know the uncertainty of harms divided from decisions made on the output of a AI model [15]. Uncertainty is an essential factor for deciding when to abstain from predictions. It is necessary to know how to deal with outliers, and anomalies, delegate high-risk predictions to humans, and so on. This is especially important for what's defined as safety-critical applications, which are applications where failure can lead to hazardous events. A hazardous event is classified as a special kind of environmental event that poses threats to humans and to human value [5].

A common problem in DL when it comes to safety-critical applications is that a lot of existing models are built on a concept that it is more practical to use a simple but uncertain rule rather than a complex but certain one [10]. The same mindset is

seen in empirical studies in the field of DL, where the main focus is to win in terms of the accuracy of the model, instead of bringing out insight and understanding of the model [24].

The second problem when it comes to DL and uncertainty for safety-critical applications is that it is hard to collect enough or any data from critical events.

Explainable Artificial Intelligence (XAI) and uncertainty

The new field of XAI has now come as a counterweight to the development happening in AI where performance is set higher than interpretability. Interpretability says something about the degree to which a human can understand the cause of the decision. Up to now, most AI algorithms are considered black boxes. This means that it is not possible for humans to understand the decisions being made in the algorithm [3]. Interpretability is especially important for three reasons:

1. Ensure impartiality in decision-making
2. Highlighting potential adversarial perturbations that could change the prediction, i.e. increase robustness
3. Only meaningful variables infer the output

However, it is important to mention that there is a trade-off between accuracy and interpretability [3]. This is probably also part of the reason why interpretability is down-prioritized in many models.

Two types of uncertainty in neural networks

The prediction uncertainty of neural networks is generally divided into two types:

1. **Aleatoric uncertainty**, also called data uncertainty arises from the natural stochasticity of observations. This is because of the unpredictable, random nature of the physical system under study.
2. **Epistemic uncertainty**, also called model uncertainty arises from a lack of knowledge of the system when it comes to quantities and processes within the system.

An important difference between aleatoric and epistemic uncertainty is that aleatoric uncertainty cannot be reduced. This means that supplying more data to the algorithm will not improve the aleatoric uncertainty, in contrast to epistemic uncertainty which will have the benefit of a decrease in uncertainty for a higher amount of data [2]. The

difference between aleatoric and epistemic uncertainty is expressed mathematically in equation 2.8. $E(Y - \hat{Y})^2$ represents the expected value of the squared difference between the predicted and actual value of Y, epistemic uncertainty, and $Var(\epsilon)$ represents the variance of the error term ϵ, aleatoric uncertainty.

$$E(Y - \hat{Y})^2 = E[f(X) + \epsilon - f(\hat{X})]^2 = \underbrace{[f(X) - f(\hat{X})]^2}_{\text{Reducible}} + \underbrace{Var(\epsilon)}_{\text{Irreducible}} \tag{2.8}$$

There will be a more in-depth description of expected value and variance in section 2.4.1.

Aleatoric and epistemic uncertainty are not mutually exclusive, however, in this thesis, the main focus is aleatoric uncertainty[2].

2.2.2 Aleatoric uncertainty

Aleatoric uncertainty is also divided into two types [13]:

1. **Homoscedastic aleatoric uncertainty** is uncertainty that stays constant for different inputs.

2. **Heteroscedastic aleatoric uncertainty** do not stay constant but depend on the inputs to the model. Some inputs to the model will potentially have more noisy outputs than others.

In this thesis, heteroscedastic aleatoric uncertainty is the relevant type that will be investigated further.

2.3 Methods for evaluating aleatoric uncertainty

This section is a direct copy of section 2.3 in [26]. It is possible to distinguish between two approaches for propagating the uncertainty through the neural network. The first propagating method is layer-wise uncertainty propagation. In this approach, the first and second moment of the output of a single layer is expressed as the uncertainty of the inputs to the layer under some assumptions. Then the uncertainty of the input is propagated through the final output layer-by-layer [12]. This thesis will not consider any methods based on this approach. The thesis will, however, consider methods based on entire-network uncertainty propagation, which is also called an input-to-output approach. This means that each layer in the network is not assessed separately. Linearization and sampling are two methods based on input-to-output uncertainty propagation.

2.3.1 Linearization

The last paragraph of this chapter is basedon section 2.3.1 in [26].

A system is linear if it satisfies the following conditions. Assume two random input u_1 and u_2, which leads to the two actions x_1 and x_2. Then a new input signal is given by

$$u = \phi u_1 + \rho u_2 \tag{2.9}$$

where ϕ and ρ is random constant coefficients. Then if the resulting response is

$$x = \phi x_1 + \rho x_2 \tag{2.10}$$

then it is possible to say that the system is linear.

However, a physical process can never be fully linear, but it is possible to perform a linearization for small fluctuations, also called perturbations around a working point. Linearization involves creating a linear differential equation, which is a good approximation of the nonlinear system [11].

Consider the following nonlinear differential equation model with input u and output y

$$\frac{dy}{dt} = f(y, u) \tag{2.11}$$

the right hand side of the equation is linearized by a Taylor series expansion

$$\frac{dy}{dt} = f(y, u) \approx f(\overline{y}, \overline{u}) + \frac{\partial f}{\partial}\bigg|_{\overline{y},\overline{u}} (y - \overline{y}) + \frac{\partial f}{\partial u}\bigg|_{\overline{y},\overline{u}} (u - \overline{u}) \tag{2.12}$$

where $\overline{y}$ and $\overline{u}$ is the chosen working point.

The working point is chosen based on the purpose of the linearization. In this thesis the purpose of linearization is to approximate mean and variance. It is therefore relevant to use a input mean as the working point. The relevant calculation to achieve these approximations are found in section 3.2.

The Extended Kalman Filter (EKF) is a method based on linearization which can be used for uncertainty propagation. It examines a nonlinear system in the discretized time domain. At each time step, it makes a prediction based on the noise, input to the system, and the previous state. The result of this prediction combined with the observation noise is the current state of the system together with the accuracy of the estimation. This is applied to DNN by treating the layers as discrete time steps and

their value as states. Using EKF for uncertainty propagation has the advantage of low computational overhead and it allows model error to be naturally incorporated into the output uncertainty [28].

2.3.2 Sampling

This section is based on section 2.3.3 in [26]. Another typical approach is estimation of uncertainty through sampling. It exists two main types of sampling in the field of statistics. The first type is survey sampling. This is based on drawing samples from a set or population. The second type and the type relevant for this thesis is sampling from a probability distribution. If the distribution is simple it is easy to draw samples from it. However, if the distribution is complicated, it is not possible to simply and directly draw samples from it. It is then possible to use the MC method. There exist different types of this method and some examples is the simple MC methods, Markov Chain Monte Carlo (MCMC), and efficient MC methods [9].

In this thesis, the main focus is on simple MC methods. These methods draw samples blindly because every step or iteration is not dependent on the previous iteration. This means that the samples are independent identically distributed (i.i.d), and based on these samples, the mean and variance of the distribution can be approximated. An example of such simple methods is the crude MC, which will be explained in detail later. Other simple MC methods are importance sampling and rejection sampling. These methods should lower the variance of the crude method without increasing the number of samples [9].

Using the MC method, where n independent samples are randomly drawn from a distribution and are denoted by x_i. The approximated mean of the distribution is

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.13}$$

where $\overline{x}$ is the sample mean [7]. However, the sample mean will almost surely converge to the theoretical mean, μ, when the number of samples is large, $n \to \infty$. This assumption is based on the law of large number in statics [14].

The variance of the distribution can be approximated as

$$s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2 \tag{2.14}$$

[7]

Monte Carlo sampling is used either when the distribution is intractable, or it is too computationally inefficient to compute it exactly. However, there is also some

disadvantages to this method. Sampling methods do not consider the relationship between aleatoric and epistemic uncertainty, which can lead to the risk of underestimating uncertainties. Since we only want to consider aleatoric uncertainty in this thesis, sampling will be considered a useful approach [15]. Another issue with the MC methods is their high computational cost. Generally the error decreases when the amount of samples increase, but it also provides a larger computational cost. Therefore, accuracy must be assessed against the computational cost.

2.3.3 Gaussian Quadrature

The quadrature rule approximates the indefinite integral of a function and is a numerical integration method based on a deterministic choice of weighted points. A indefinite integral mean that the approximation is done over a infinite range $(-\infty, \infty)$ using the quadrature. Consider the following weighted integral of an integrable function $f(x)$ over an interval $[a, b]$:

$$I(f) = \int_a^b w(x)f(x)dx \tag{2.15}$$

Based on the chosen weights, $w(x)$, and a linear combination of a nonlinear transformation of weighted points, the quadrature is an approximation on the form:

$$I(f) \approx \sum_{i=1}^{n} w_i f(x_i) \tag{2.16}$$

where ξ_i is the quadrature points. Both the nonlinear transformation and the choice of points, also called abscissas or nodes, are dependent on the specific integral problem. The different integration problems are based on orthogonal polynomials, such as the Legendre polynomials or Hermit polynomial. Orthogonal polynomials are a class of polynomials $p_n(x)$ defined over a range $[a, b]$ that obey an orthogonality relation

$$\int_a^b w(x)p_m(x)p_n(x)dx = \delta_{mn}c_n, \tag{2.17}$$

where w(x) is a weighting function and δ_{mn} is the Kronecker delta which is a discrete version of the delta function

$$\delta_{ij} = \begin{cases} 0 \text{ for } i \neq j \\ 1 \text{ for } i = j \end{cases} \tag{2.18}$$

and c_n is given by

$$c_n = \int_a^b w(x)[p_n(x)]^2 dx. \tag{2.19}$$

The deterministically chosen weighted points are the reason for the generally high performance of the method. The deterministic way of choosing points minimizes the error in the approximation.

The General Fundamental Theorem of Gaussian Quadrature

The fundamental theorem of Gaussian Quadrature states that the abscissas, x_i, of the n-point Gaussian Quadrature formula is precisely the roots of the Orthogonal Polynomial for the same interval and weighting function. Mathematically the same is expressed as,

$$\int_a^b w(x)x^k p_n(x)dx = 0, \text{ for all } k = 0, 1, ..., n-1 \tag{2.20}$$

where p_n is a nontrivial polynomial of degree n, which again is orthogonal to the other polynomial p_j for $j \neq n$, the weighting function is $w(x)$ and x^k is the span of the set.

Let $h(x)$ be any polynomial in x. Then if n nodes x_i are chosen to be the zeroes of p_n, there will exist n weights w_i, which makes the Gaussian-quadrature computed integral exact for all polynomials $h(x)$ of maximum degree $2n-1$. This is why the quadrature rule yields an exact result for a polynomial of degree $2n-1$ or less. The function $h(x)$ can be divided by $p_n(x)$ to produce a quotient $q(x)$ of degree strictly lower than n, and a remainder $r(x)$ still of lower degree, so that both are orthogonal to $p_n(x)$. This gives

$$\underbrace{h(x)}_{2n-1} = \underbrace{p_n(x)}_{n}\underbrace{q(x)}_{n-1} + \underbrace{r(x)}_{n-1} \tag{2.21}$$

It is assumed that p_n is orthogonal to all monomials of degree less than n. A monomial is an algebraic term that only contains one term, e.g., x^3. This assumption gives that p_n is orthogonal to the quotient $q(x)$. This again gives

$$\int_a^b w(x)h(x)dx = \int_a^b w(x)(p_n(x)q(x) + r(x))dx = \int_a^b w(x)r(x)dx. \tag{2.22}$$

The remainder $r(x)$ is of degree $n-1$ or less. This means that it is possible to

interpolate it exactly using n interpolations using the Lagrange interpolation

$$r(x) = \sum_{i=1}^{n} r(x_i) \prod_{\substack{1 \le j \le n \\ j \neq i}} \frac{x - x_j}{x_i - x_j} \tag{2.23}$$

Then the integral will equal

$$\int_a^b w(x) r(x) dx = \sum_{i=1}^{n} r(x_i) \int_a^b w(x) \prod_{\substack{1 \le j \le n \\ j \neq i}} \frac{x - x_j}{x_i - x_j} dx = \sum_{i=1}^{n} r(x_i) w_i \tag{2.24}$$

with the general formula for the weights being $w_i = \frac{a_n}{a_{n-1}} \frac{\int_a^b w(x) p_{n-1}(x)^2 dx}{p_n'(x_i) p_{n-1}(x_i)}$. Where a_k is the coefficient of x^k of $p_k(x)$.

Together with equation,2.22 this gives the relation

$$\int_a^b w(x) h(x) dx = \int_a^b w(x) r(x) dx = \sum_{i=1}^{n} w_i r(x_i) = \sum_{i=1}^{n} w_i h(x_i) \tag{2.25}$$

based on the fact that x_i are roots of p_n. This again yields that

$$h(x_i) = r(x_i), \tag{2.26}$$

for all i [8].

Given m distinct quadrature points, one can calculate w_i by first computing the moments M_i of the integral

$$M_i = \int_a^b x^i w(x), \text{ for } i \in 0, 1, ..., (m-1) \tag{2.27}$$

Since the quadrature rule is Gauss-Hermite, one knows that the weight function is the standard Gaussian/normal distribution, and the moments will therefore be the same as for a standard normal distribution.

The moments are then used to solve the following Vandermonde system of equations:

$$\begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ \vdots & \vdots & & \vdots \\ x_1^{m-1} & x_2^{m-1} & \dots & x_m^{m-1} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{pmatrix} = \begin{pmatrix} M_0 \\ M_1 \\ \vdots \\ M_{m-1} \end{pmatrix} \tag{2.28}$$

This can be inverted to solve for w_i. This is an alternative to equation 2.24, to compute the weights [23].

2.4 Probability theory for uncertainty estimation

This hole chapter is a direct copy of chapter 2.4 in [26]. Probability theory was originally developed for analyzing the frequencies of events, where the events are repeatable. When the experiment is repeated an infinite number of times it is possible to say that a specific outcome has the probability p to occur. However, this kind of reasoning is not possible for experiments where there is no repeatable event. In these events, the probability is instead represented as a degree of belief. Here 1 indicates absolute certainty of the investigated case and 0 indicates the opposite. The last method goes by the name Bayesian probability and is the method within probability theory associated with DNN [10].

Bayesian probability is based on the well known Bayes Theorem:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} \tag{2.29}$$

The goal of this theorem is to estimate the posterior distribution $p(\theta|x)$ given the likelihood $p(x|\theta)$ and the prior distribution.

2.4.1 Probability distributions

A probability distribution is a description of how likely a random variable or a set of random variables are to take on each of its possible states. A random variable can take on different values randomly. A continuous random variable is associated with a real number.

In the domain of continuous random variables, the probability distribution is expressed as a Probability Density Function (PDF).

This thesis will look into two different continuous probability distributions, the Gaussian distribution, and the Uniform distribution.

Gaussian Distribution/Normal distribution

A well-used distribution for the continuous random variables is the Gaussian distribution, which also goes by the name normal distribution. The normal distribution is controlled by two parameters, $\mu \in R$ and $\sigma \in (0, \infty)$. The first parameter μ is the

mean value of the PDF and gives the coordinates where the peak of the function is located. The second parameter σ is the standard deviation of the distribution and controls the width of the distribution [10]. The distribution is given then N(x;μ, σ^2). The simplest case of the normal distribution is the standard normal distribution. This is the special case where $\mu = 0$ and $\sigma = 1$, which gives N(x;0, 1) See the figure 2.3 below for a plot of the normal distribution.

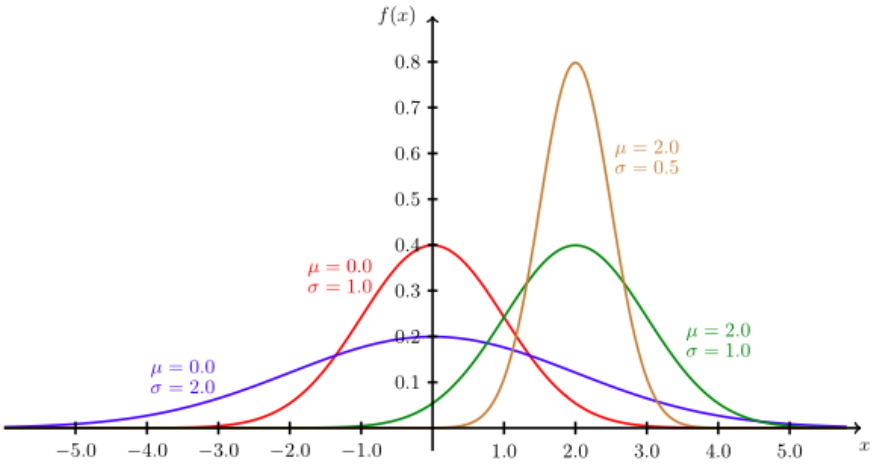


Figure 2.3: The normal distribution N(x;μ, σ^2). In this example, we depict the standard normal distribution with different values for μ and σ. [21]

Uniform Distribution

The uniform distribution is another continuous distribution which is defined over an interval of real numbers. The interval is specified by a and b, which are the endpoints of the interval, with $a < b$. The function is then defined as $u(x; a, b)$, and within the interval $[a, b]$ the function is given by $u(x; a, b) = \frac{1}{b-a}$. Outside this interval the function is always zero [10]. See the figure 2.4 below for a plot of the uniform distribution and how different values for a and b affect the function.

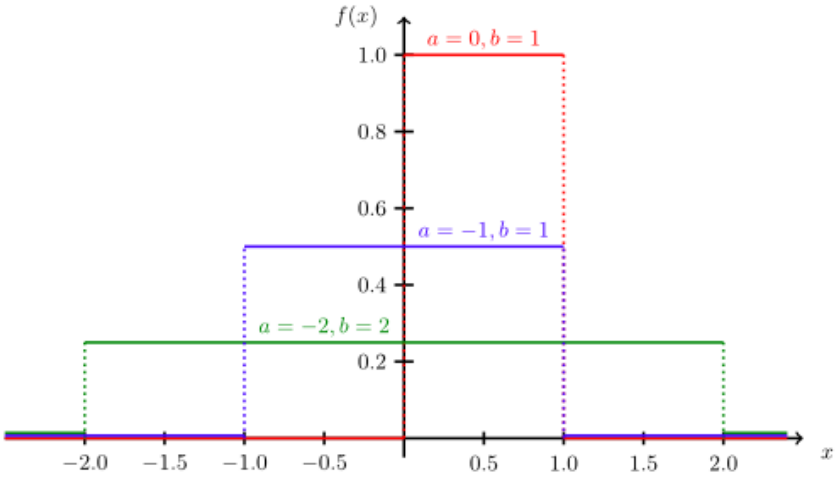


Figure 2.4: The uniform distribution Unif(x;a,b). In this example, we depict the standard normal distribution with different values for a and b. [29]

Expectation and Variance

The expected value of some function $f(x)$ with respect to a probability function $P(x)$ is the average, also called the mean value, that x can take when it is drawn from the probability function P [10]. For continuous random variables, it can be expressed with the following integral:

$$E_{x\sim P} = \int p(x)f(x)dx \tag{2.30}$$

The variance is given as the measurement of how the values of a function of a random variable x vary as different values x are sampled from the probability distribution [10]. The variance is given as:

$$Var(f(x)) = E[(f(x) - E[f(x)])^2] \tag{2.31}$$

from the equation above it is possible to see that the expected value is included in the variance. The relation between variance and expected value is that when the variance is low, the values of $f(x)$ cluster near the expected value. If the variance is high, the values will be spread out a long distance from the expected value.

Another useful measurement is the standard deviation. This is given as the square root of the variance:

$$std(f(x)) = \sqrt{var(f(x))} \tag{2.32}$$

Chapter 3

Methodology

This chapter presents three approaches for uncertainty based on estimating aleatoric uncertainty of a single input - single output perspective, where the input is a stochastic nonlinear function. However, the first section of the chapter is about the development of a benchmark. This benchmark is intended to be a ground truth, which the developed methods that can be compared against. In the next section, linearization is developed as an estimation method for uncertainty. The second method developed is presented in section 3.3 and is an MC sampling method. The last method, the Gaussian quadrature, is then explained in section 3.4.1. The following section goes further in detail about the implementation of the methods and how the results are presented. The last section of this chapter shows how the methods are tested and evaluated on a neural network.

3.1 Benchmark model

This whole chapter is based on chapter 3.1 in [26]. Computing an analytical expression for the output mean and variance for many nonlinear functions is challenging. Therefore instead of an analytical method, it is often only possible to obtain an approximation. To say something about the accuracy of the approximation, it is needed to have a benchmark considered to be the actual value of the output mean and variance. The approximation can then be compared against these actual values to decide the performance of the approximation. The benchmark in this project is an analytical computation of the mean and variance of a polynomial function.

In the following section, the approach to compute these true values will be presented using two different distributions, the Gaussian distribution/Normal distribution, and the Uniform distribution.

3.1.1 Mean and variance of a polynomial function with normally distributed inputs

As mentioned above, the goal is to find the output mean and output variance given a function $f(u)$ and input values for the expected mean, μ_u, and variance, σ_u^2. To obtain this when μ_u and σ_u^2 are normally distributed, the following variable is given:

$$x = au + b, \tag{3.1}$$

Where u is a standard normal variable, and a and b are any real constants. This makes x a normal variable scaled by a and shifted by b compared to u. Considering the information given about the normal distribution in the chapter 2.4.1, it is possible to consider a as the standard deviation, σ, and b as the expected value, μ, because it has the same effect on the shape of the distribution.

The mean and variance of x are then given by:

$$\begin{aligned} \mu_x &= E[x] \\ &= E[au + b] \\ &= aE[u] + b \\ &= am_1 + b \end{aligned} \tag{3.2}$$

and,

$$\begin{aligned} \sigma_x^2 &= E[(x - \mu_x)^2] \\ &= E[(au + b - \mu_x)^2] \\ &= a^2E[u^2] + 2a(b - \mu_x)E[u] + (b - \mu_x)^2 \\ &= a^2m_2 + 2a(b - \mu_x)m_1 + (b - \mu_x)^2 \end{aligned} \tag{3.3}$$

where the variables m_1 and m_2 are moments of the distribution. These moments depend on the given function, $f(u)$, and can be calculated considering the probability function of a standard normal distribution. This probability function is given by:

$$\phi_u(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} \tag{3.4}$$

Then the expected output of a function $f(u)$, where f is a function of the variable u drawn from a standard normal distribution, is given by the following integral:

$$E[f(u)] = \int_{-\infty}^{\infty} f(u)\phi_u(u)du \tag{3.5}$$

If the function $f(u)$ is represented as a monomial of order $n \geq 0$ (i.e. $f(u) = u^n$), then the n^{th} moment of the distribution is defined as:

$$m_n = E[u^n] = \int_{-\infty}^{\infty} u^n \phi_u(u) du \tag{3.6}$$

By using the probability function of the standard normal distribution given by the equation 3.4 in the equation above and computing the integral, the following values are given for m_1 and m_2, as well as for m_0

$$m_0 = 1, m_1 = 0, m_2 = 1 \tag{3.7}$$

For higher dimensions, the general formula for the moments is given by:

$$m_n = \begin{cases} 0 & \text{if n is odd} \\ (n-1)!! & \text{if n is even} \end{cases} \tag{3.8}$$

Using this formula, the first ten moments are:

m_0	1
m_1	0
m_2	1
m_3	0
m_4	3
m_5	0
m_6	15
m_7	0
m_8	105
m_9	0
m_{10}	945

Table 3.1: Ten first moments normal distribution

Going back to equation 3.2 and 3.3, inserting the calculated moment gives the following:

$$\mu_x = b, \sigma_x^2 = |a| \tag{3.9}$$

This means that the equation 3.1 may also be written as,

$$x = \sigma_x u + \mu_x \tag{3.10}$$

As mentioned initially, the main goal is to be able to compute the mean and variance of a polynomial function. This function can consist of several variables and coefficients, which can for example be the function $f(u) = u^5 + u^4 - 25u^3 - 36u^2 + 25u + 200$. To solve such functions, it is possible to use equation 3.10 and define:

$$y = x^n = (\sigma_x u + \mu_x)^n \tag{3.11}$$

Further it is possible to use this expression to calculate the mean and variance for each integer exponential of variables in the polynomial by equation 3.2 and 3.3. The combination of each result of integer exponential of variables together with their coefficient gives the output mean and variance of the polynomial function.

3.1.2 Mean and variance of a polynomial function with uniform distributed inputs

The same task as in the previous chapter is solved in this chapter. However, the input values for μ_u and σ_u^2 are now uniform distributed instead of normally distributed. The approach is very much the same, and the main difference is the moments of the distribution. To find a general expression for n^{th} moments of the distribution, let's start with the PDF of the uniform distribution:

$$\phi_x(x) = \begin{cases} c & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

Here a and b are the constants defined in chapter 2.4.1 and c is the constant defining the height of the function above the x-axis. See the figure 2.4for an illustration of the uniform distribution.

The integral of the PDF of any probability function is equal to one. This gives the value for the zero moment:

$$m_0 = \int_{-\infty}^{\infty} \phi_x(x)dx = \int_a^b cdx = c(b-a) = 1 \tag{3.13}$$

which holds if,

$$c = \frac{1}{b-a} \tag{3.14}$$

Using the expression for c, the n^{th} moments of the uniform distribution are given by:

$$m_n = \int_{-\infty}^{\infty} x^n \phi_x(x) dx = \int_a^b cx^n dx = \frac{c}{n+1}(b^{n+1} - a^{n+1}) \tag{3.15}$$

For this expression to be useful, it is convenient to introduce a standard uniform distribution. In the same way, the standard normal distribution is defined, i.e., $\mu_u = 0$ and $\sigma_u^2 = 1$. For a standard normal distribution the mean is respectively given as $\mu_u = E[u] = m_1 = 0$. Using the same for the standard uniform distribution gives the following

$$m_1 = \mu_u = \frac{c}{2}(b^2 - a^2) = 0 \tag{3.16}$$

which is further used to define a and b,

$$a = -b \tag{3.17}$$

On the other hand the variance is defined as $\sigma_u^2 = m_2 - \mu_u^2$. This gives

$$\sigma_u^2 = \frac{c}{3}(b^3 - a^3) - \mu_u^2 = \frac{b^2}{3}1 \tag{3.18}$$

Thus, the following values are obtained for the constants,

$$a = -\sqrt{3}, b = \sqrt{3}, c = \frac{1}{2\sqrt{3}} \tag{3.19}$$

The general expression for n^{th} moments for the standard uniform distribution is then,

$$m_n = \frac{b^{n+1} - a^{n+1}}{(n+1)(b-a)} = \begin{cases} 0 & \text{if n is odd} \\ \frac{\sqrt{3}^{n+1} - (-\sqrt{3})^{n+1}}{(n+1)(\sqrt{3} - (-\sqrt{3}))} & \text{if n is even} \end{cases} \tag{3.20}$$

Calculations with this formula give the ten following moments for uniform distribution:

m_0	1
m_1	0
m_2	1
m_3	0
m_4	$\frac{9}{5}$
m_5	0
m_6	$\frac{27}{7}$
m_7	0
m_8	9
m_9	0
m_{10}	$\frac{243}{11}$

Table 3.2: Ten first moments uniform distribution

When the moments are calculated, the same approach as for the normal distribution can be used. Equation 3.10 translates from the standard uniform distribution back to a uniform distribution. Then equation 3.11, together with 3.2 and 3.3 gives the output mean and variance of the polynomial function as described above.

3.2 Linearization

This chapter is a direct copy of chapter 3.2 *Linearization* in [26]. Linearization is implemented as the first method to evaluate a polynomial function's output mean and variance. It is a pretty straightforward method where the linearization of a polynomial function, $f(u)$, with input mean, μ_u, and input variance, σ_u^2, is calculated as follows:

$$l(u) = f(\mu_u) + \frac{df}{du}(\mu_u)(u - \mu_u) \tag{3.21}$$

From the equation above, it is possible to see that the function is linearized by the input mean.

The goal is then to calculate the mean and variance of the linearization, $l(u)$. The output mean, μ_y is derived as the expected value of the linearized function:

$$\begin{aligned} \mu_y &= E[l(u)] \\ &= E[f(\mu_u) + \frac{df}{du}(\mu_u)(u - \mu_u)] \\ &= f(\mu_u) + \frac{df}{du}(\mu_u)(E[u] - \mu_u) \\ &= f(\mu_u) \end{aligned} \tag{3.22}$$

The output variance, σ_y^2, is given by:

$$\begin{aligned} \sigma_y^2 &= E[(l(u) - \mu_y)^2] \\ &= E[(f(\mu_u) + \frac{df}{du}(\mu_u)(u - \mu_u) - \mu_y)^2] \\ &= (\frac{df}{du}(\mu_u))^2 E[(u - \mu_u)^2] \\ &= (\frac{df}{du}(\mu_u))^2 \sigma_u^2 \end{aligned} \tag{3.23}$$

3.3 Monte Carlo sampling

This chapter is based on chapter 3.3 *Monte Carlo sampling* in [26]. There are several different sampling algorithms, and one well-known collection of sampling algorithms is the MC sampling method. In this thesis, the crude MC approximation is implemented. This is a simple approach to the MC sampling technique. The approach is based on finding an approximation to an integral on the form:

$$I = \int f(u)du \tag{3.24}$$

As mentioned, the problem with this integration of the polynomial function $f(u)$ is that it is not possible to solve it analytically. The MC approximation uses a principle that estimates the average of the integrand $f(u)$ and uses this to approximate the integral instead of evaluating the indefinite integral in equation3.24. The equation for finding the average value of a function is given below:

$$f_{avg} = \frac{1}{b-a}\int_a^b f(u)du \tag{3.25}$$

where the bounds of the integral, a and b, are called the integration range. It is from this range the samples are randomly chosen. In this thesis, the range is either a given normal distribution or a given uniform distribution. Therefore, the first step in the algorithm is to get a random input value from the integration range and evaluate the integrand, $f(u)$, for this input. This first step is repeated n numbers of times before the average of all the samples multiplied by the range gives the final approximated average of the integrand. The result of the approximation will therefore vary with the number of samples. This is, however, not the only parameter affecting the result. Since the samples are chosen randomly from a range, the result will vary each time.

The variation in the estimated average is defined as the variance and says how accurate the estimate is, i.e., how much $f(u)$ varies in the domain of x. The variance is given by:

$$\sigma^2 = [\frac{b-a}{N}\sum_i^N f^2(u_i)] - [\sum_i^N \frac{b-a}{N} f(u_i)]^2 \tag{3.26}$$

3.4 Gaussian Quadrature

Gaussian quadrature is the third method implemented for calculating mean and variance. To clarify the theory given in section 2.3.3, an example will be presented of how the orthogonal polynomials, their roots, and corresponding weights can be

calculated. Calculating the roots and weights are necessary steps toward finding the mean and variance, and how these depend on each other is presented at the end of the section.

As mentioned, there are several different orthogonal polynomials to choose from, and each polynomial has its own quadrature rule. This means that each rule depends on the weight function $w(x)$ or the probability density function. The weight function used in the calculations in the following sections is based on the standard Gaussian density (standard normal distribution) with zero mean and unit variance $\mathcal{N}(x; 0, 1)$ and the interval of interest is $(-\infty, \infty)$. Such weight function corresponds to a Hermit polynomial, and the quadrature rule used in this polynomial goes by the name, Gauss-Hermite quadrature rule.

3.4.1 Calculating the Orthogonal Polynomials, the Root, and the Corresponding Weights

Calculating the orthogonal polynomials

The chosen example is based on a Hermite orthogonal polynomial of degree three. The general expression for the polynomials up to degree n is given by

$$p_n(x) = x^n + c_1 x^{n-1} + c_2 x^{n-2} + ... + c_n \tag{3.27}$$

However, the zero-order polynomial $p_0(x)$ will always be $p_0(x) = c_0 = m_0 = 1$, based on the first moment of a standard normal distribution.

From equation 3.27 one finds the first-order polynomial as $p_1(x) = x + c$, and to find the orthogonal polynomial, the following integral need to be solved

$$\int_{-\infty}^{\infty} w(x) p_1(x) p_0(x) dx = 0 \tag{3.28}$$

Again considering the moments of a standard normal distribution, one has

$$\int_{-\infty}^{\infty} w(x) p_1(x) p_0(x) dx = m_1 + c m_0 = 0 + c = c \tag{3.29}$$

The integral in equation 3.28 is equal zero, thus $c = 0$, which again yields $p_1(x) = x$.

Doing the same for second-order polynomials on the form $p_2(x) = x^2 + c_1 x + c_2$, gives the following two integrals to solve:

$$\begin{aligned}\int_{-\infty}^{\infty} w(x)p_2(x)p_0(x)dx = 0\\ \int_{-\infty}^{\infty} w(x)p_2(x)p_1(x)dx = 0\end{aligned} \tag{3.30}$$

Again using moments and solving the integrals give

$$\begin{aligned}\int_{-\infty}^{\infty} w(x)p_2(x)p_0(x)dx = m_2 + c_1m_1 + c_2m_0 = 1 + 0 + c_2 = 1 + c_2\\ \int_{-\infty}^{\infty} w(x)p_2(x)p_1(x)dx = m_3 + c_1m_2 + c_2m_1 = 0 + c_1 + 1 = c_1\end{aligned} \tag{3.31}$$

Knowing that both expression in the equation above should be equal zero, thus $c_1 = 0$ and $c_2 = -1$. Hence $p_2(x) = x^2 - 1$. The next step is then to do the same step, but know for the third-order polynomial $p_3(x) = x^3 + c_1x^2 + c_2x + c_3$. This means one has to solve the three following integrals

$$\begin{aligned}\int_{-\infty}^{\infty} w(x)p_3(x)p_0(x)dx = 0\\ \int_{-\infty}^{\infty} w(x)p_3(x)p_1(x)dx = 0\\ \int_{-\infty}^{\infty} w(x)p_3(x)p_2(x)dx = 0\end{aligned} \tag{3.32}$$

Then solving the integrals in equation 3.32 gives

$$\begin{aligned}\int_{-\infty}^{\infty} w(x)p_3(x)p_0(x)dx = m_3 + c_1m_2 + c_2m_1 + c_3m_0 = c_1 + c_3\\ \int_{-\infty}^{\infty} w(x)p_3(x)p_1(x)dx = m_4 + c_1m_3 + c_2m_2 + c_2m_1 = 3 + c_2\\ \int_{-\infty}^{\infty} w(x)p_3(x)p_2(x)dx = m_5 + c_1m_4 + c_2m_3 - m_3 - c_1m_2 + c_3m_2 - c_2m_1 - c_3m_0 = 2c_1\end{aligned} \tag{3.33}$$

which all equals zero and therefore yield $c_2 = -3$ and $c_1 = c_3 = 0$. This gives $p_3(x) = x^3 - 3x$

For higher-order polynomials, the procedure is continued. The example above uses the moment of a standard normal distribution. As long as the moments of another type of distribution (for example, uniform distribution) are known, the moments can easily be switched out, and the quadrature is calculated for the wanted distribution.

Calculating the roots of the orthogonal polynomials

The next step in finding the weights is calculating the root, x_i of the orthogonal polynomials. This is done by setting each of the polynomials, $p_n(x)$, found in the previous section equal to zero and solving for x

$$p_1(x) = x = 0 \tag{3.34}$$

this gives $x_1 = 0$. Then doing the same for second-order polynomial gives

$$p_2(x) = x^2 - 1 = 0 \tag{3.35}$$

gives $x_1 = -1$ and $x_2 = 1$. For the third order in the example above, one got

$$p_3(x) = x^3 - 3x = 0 \tag{3.36}$$

which has three roots $x_1 = 0$, $x_2 = -\sqrt{3}$ and $x_3 = \sqrt{3}$. Following the same approach gives roots for higher-order polynomials.

Calculating the weights

Now that the roots of the orthogonal polynomials are found, the last step is to calculate the corresponding weights. To calculate these weights, one uses equation 2.28 and inserts the calculated roots and the known moments of a distribution, in this case, the standard normal distribution. For the third-degree example used in this section, the Vandermond system will look like

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & -\sqrt{3} & \sqrt{3} \\ 0 & 3 & 3 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \tag{3.37}$$

Inverting this to solve for w_i gives the following equation

$$\begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \frac{-1}{3} \\ 0 & \frac{-\sqrt{3}}{6} & \frac{1}{6} \\ 0 & \frac{\sqrt{3}}{6} & \frac{1}{6} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \tag{3.38}$$

The corresponding weights are $w_1 = \frac{2}{3}$, $w_2 = \frac{1}{6}$ and $w_3 = \frac{1}{6}$.

3.4.2 Output mean and variance using Gaussian quadrature

When roots and weights are found for the orthogonal polynomials, these values can be used in calculations of output mean, μ_y, and variance σ_y^2, of a polynomial function, $f(u)$, with inputs values for mean, μ_u, and variance, σ_u^2.

$$\mu_y = E[h(y)] = \sum_{i=1}^{n} w_i f(\sigma_u x_i + \mu_u) \tag{3.39}$$

It is possible to see that the equation for output mean is the same as the equation for the approximation, 2.16, except that equation 3.39 also has the possibility to choose values for σ_u and μ_u, such that the output mean can be calculated for different distributions. The same goes for output variance, which also considers input values for mean and variance, as seen below

$$\sigma_y^2 = \sum_{i=1}^{n} w_i (f(\sigma_u x_i + \mu_u) - \mu_y)^2. \tag{3.40}$$

3.5 Methodology for evaluation of linearization, Monte Carlo sampling, and gaussian quadrature

This section gives an overview of the implementation needed for each method and how plots are generated to be able to present the result. All implementations are done using Python.

3.5.1 Script for analytical computation of mean and variance

This section is a direct copy of section 3.4.1 in [26]. A python-script was generated to make this benchmark easy to use for higher-order functions. This script will calculate the analytical output mean and variance for any polynomial function, both for normal and uniform distribution. The input values for mean and variance, together with the nonlinear function, can be changed to test for different cases. The benchmark is also made such that it is possible to see how the output mean or output variance behaves with a varying input mean or input variance.

3.5.2 Linearization

This section is a direct copy of section 3.4.2 in [26]. Linearization is implemented following the procedure in section 3.2. The linearization method has the same

functionality as the benchmark. This is so that the benchmark model and the linearization method can be plotted against each other to see how well the linearization performs against the exact solution. Since the benchmark model is linked to a specific distribution and the linearization method is not, the benchmark used to compare with linearization will always be based on the normal distribution.

3.5.3 Monte Carlo sampling

This section is based on section 3.4.3 in [26]. The implementation of the MC method is based on section 3.3. The random samples are generated using the built-in function in the NumPy package called *np.random.normal(loc=0.0, scale=1.0, size=None)* or *np.random.uniform(low=0.0, high=1.0, size=None)* dependent on the distribution. Using the default values for the built-in functions gives the standard normal distribution and standard uniform distribution. However, if wanted, the samples can be scaled to work for other normal/uniform distributions. If u is a sample from the standard normal/uniform distribution, it is scaled by taking $new\_sample = sigma * u + mean$, where choosing $sigma$ and $mean$ give the wanted normal/uniform distribution.

Generating plots for the MC method is somewhat more complicated than for the linearization method. This is due to of how the samples are drawn from the distribution and the number of samples that need to be considered.

First, an illustration of how the randomness of choosing samples affects the results, is made for both the normal and uniform distribution and for output mean and output variance. The number of samples follows a logarithmic scale, such that the number of samples goes logarithmic from 1 to 1 million. Each number of samples is run one hundred times to see if the randomness in drawing samples has a significant variation that needs to be considered.

When the MC method is plotted against the benchmark, it has the same functionality as the benchmark, i.e., the possibility to choose the distribution function, and input values for mean and variance. However, since there is some variation in the results for each iteration, the same samples are run ten times because of the randomness. Then an average over the ten different iterations is found, and the standard deviation is related to this average. This is done so that the randomness will have less effect on the final result. The standard deviation will show us how much variation there is between the ten iterations.

3.5.4 Gaussian Quadrature

As mentioned in earlier sections, the quadrature rule used in this thesis is the Gauss-Hermite quadrature rule. This rule is based on the probabilist's Hermite polynomials, which again consider a probability density function for the standard

normal distribution. This rule is also to be found inside the NumPy library in Python, which makes the implementation relatively straightforward. The function used is the *np.polynomial.hermit_ e.hermegauss(n)*, which takes in n number of quadrature points and returns both the roots and weights corresponding to the number of points. However, some scaling of the weights needs to be done so that they can be used in further calculations. This scaling is done by dividing each weight by the sum of all weights

$$new\_w_i = \frac{w_i}{\sum_{i=1}^{n} w_i} \tag{3.41}$$

When this scaling is completed, the sum of all the new weights should equal one. Then the roots and weight obtained can be used further in calculations to find the mean and variance, as explained in section 3.4.2.

However, finding the Gaussian quadrature based on the uniform distribution is not straightforward and something quite unusual. Therefore, no built-in functions exist for this in Python, and the implementation has to be done manually following the procedure in section 3.4.1. Because of the complexity of calculating the orthogonal polynomials, the calculations were done for polynomials up to degree five by hand. It is important in these calculations to use the moments from the uniform distribution to obtain the correct values for c_n. The obtained values for $c = [c_1 \dots c_n]$ is then used in the python implementation calculating the corresponding roots and weights. This last step is not dependent on the distribution. However, it needs to be manually implemented for the uniform distribution since Python does not have a package for the normal distribution.

The Gaussian quadrature method is also plotted against the benchmark described at the beginning of this chapter. It is given the same functionalities to vary both inputs mean and variance, as well as alter between normal and uniform distribution.

3.6 Neural Network implementation

In this section, a method for estimating aleatoric uncertainty using a neural network is presented. The neural network used is a network that is already trained. This means that with given weights and biases, the output can be calculated for each of the three methods already used for uncertainty estimation, namely linearization, MC sampling, and Gaussian Quadrature.

The section is organized as follows: First, the neural network already trained and used to test the methods will be described. Then details about how to use the network's output for calculating mean and variance are given. Finally, the combination of the neural network and the linearization, MC sampling, and Gaussian Quadrature method is presented.

3.6.1 Neural network details - Haldane net

The network used to check the performance of the three methods, linearization, MC sampling, and Gaussian quadrature, is a network based on the *Haldane law*. Therefore, this network will from now on be referred to as the *Haldane network*. To approximate the nonlinear function, the network considers the relation between the substrate concentration S and the specific growth rate μ of bacteria in a bioreactor. The relation between these two quantities can be described by the following nonlinear curve, which is taken from the book "Dynamical Modelling and Estimation in Wastewater Treatment Processes" by D. Dochain and Peter A. Vanrolleghem (2001) on page 30 [6]:

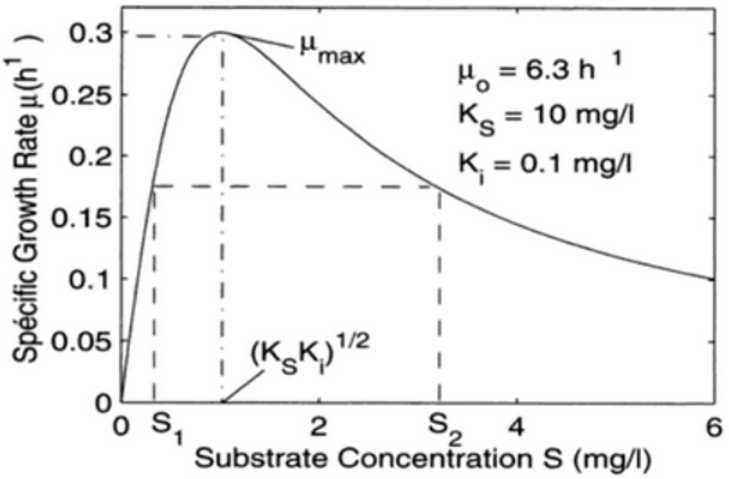


Figure 3.1: Haldane Law [6]

The growth rate is given by

$$\mu = \frac{\mu_0 S}{K_S + S + \frac{S^2}{K_i}} \tag{3.42}$$

The relation is described mathematically by equation 3.42 known as the *Haldane law* [16].

The Haldane network is trained to describe the relation between the substrate concentration S and the specific growth rate μ due to the data set. The data set is obtained by adding noise to the mathematical relation 3.42. The network has one hidden layer with three neurons shown in Figure 3.2

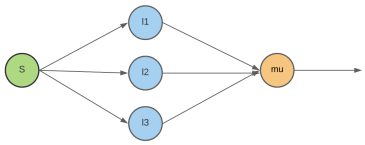


Figure 3.2: A neural network with one input, one hidden layer with three neurons, and one output.

The network's output μ is described by the following relations

$$l_1 = \phi(w_{1,1}S + b_{1,1}) \tag{3.43a}$$
$$l_2 = \phi(w_{1,2}S + b_{1,2}) \tag{3.43b}$$
$$l_3 = \phi(w_{1,3}S + b_{1,3}) \tag{3.43c}$$
$$\mu = w_{2,1}l_1 + w_{2,2}l_2 + w_{2,3}l_3 + b_2 \tag{3.43d}$$

where ϕ is the chosen activation function. In this thesis, ϕ chosen to be the ELU, given by:

$$\phi(u) = \begin{cases} e^u - 1 \text{ if } u < 0 \\ u \text{ for } u \geq 0 \end{cases} \tag{3.44}$$

The corresponding weights and biases in equation 3.43 are given by

$w_{1,1} = -2.3725$	$w_{2,1} = -0.3245$	$b_{1,1} = 0.6461$	$b_2 = 0.2990$
$w_{1,2} = -0.6090$	$w_{2,2} = 0.5539$	$b_{1,2} = -0.3452$	
$w_{1,3} = -0.0576$	$w_{2,3} = 0.1908$	$b_{1,3} = 0.4535$	

The reason for choosing the ELU activation function instead of the more known ReLU activation function is mainly because the linearization method will be used on this neural network. The ReLU activation function is much more used and is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. However, the function is not defined in the the breaking point when the function is going from a linear function $R(x) = x$, to being zero when $R(x) = 0$ [1]. This means that the ReLU function is not differentiable everywhere, and this can lead to a problem for the linearization method, which is based on derivation. The ELU activation function is, on the other side, smooth everywhere and is therefore also differentiable everywhere. This means that the problem with the linearization method and the ReLU does not hold for the ELU function. The ELU activation function works for all three methods. Apart from this mentioned difference in the two functions, ELU is in many ways similar to ReLU [1].

3.6.2 Calculations of output mean and variance using the output of the network

The output of the network itself does not give us much relevant information. However, when the output of the neural network is calculated for a given input, the output can be used further to calculate the mean and variance. This again tells us something about the uncertainty in the neural network. The output of the network is denoted $Y = [y_1, y_2, \ldots, y_n]$. The approximated mean is then

$$\mu_y = \frac{1}{N}\sum_{i=1}^{n} y_i \tag{3.45}$$

while the variance can be approximated as

$$\sigma_y^2 = \frac{1}{N-1}\sum_{i=1}^{n}(y_i - \mu_y)^2 \tag{3.46}$$

3.6.3 Testing each method on the neural network

Linearization, MC sampling, and Gaussian quadrature are all methods for approximating a nonlinear function. Now we want to use these methods as inputs to the neural network described above. The neural network also approximates a nonlinear function, and using one of the methods as inputs to the network will result in an approximation of the approximation. In figure 3.3 there is a visualization of the input-output relations in the network. As mentioned, the X input is a matrix of values generated from one of the three methods; linearization, MC sampling, and Gaussian quadrature. This matrix is sent through the Haldane network and results in the output matrix Y, before this matrix is used in the calculations of output mean and variance described above.

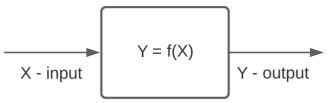


Figure 3.3: Input-output relation Neural Network

Below is a more detailed description of how each method is used as input to the network and how the X matrix is generated for each method. However, first, a new benchmark used to check the performance of each method is presented.

Neural network benchmark

Again a benchmark is needed to say something about the accuracy of the network's approximation. However, this time it is not that straightforward to define a bench-

mark. Therefore the benchmark used will not be a 100% ground truth, but only a result of the method one believes and theoretically should be close to the true ground truth. The benchmark also needs to be relatable to the methods it is to be compared with so that the results can be interpreted.

The benchmark used for the network approximation is the result of the MC sampling method as input to the method with i number of samples drawn equal to $i = 100million$. Since the benchmark is generated using the MC method as input to the network, it follows the same procedure as explained in the section below for MC sampling.

Linearization

Linearization is the first method tested on the network. However, this method is not only affecting the input to the network but the network itself. This is because we now want to linearize the nonlinearity in the network given by equation 3.43. This is done by a chain rule derivation of the function

$$\mu = F(S) = w_{2,1}\phi(w_{1,1}S+b_{1,1})+w_{2,2}phi(w_{1,2}S + b_{1,2})+w_{2,3}\phi(w_{1,3}S+b_{1,3})+b_2 \quad (3.47)$$

with respect to S. Performing this linearization gives

$$\begin{aligned}\frac{F(S)}{\partial S} = &\begin{cases} w_{2,1}e^{u_1}w_{1,1} & if\ u_1 < 0 \\ w_{2,1}w_{1,1} & if\ u_1 \geq 0 \end{cases} \\ &+ \begin{cases} w_{2,2}e^{u_2}w_{1,2} & if\ u_2 < 0 \\ w_{2,2}w_{1,2} & if\ u_2 \geq 0 \end{cases} \\ &+ \begin{cases} w_{2,3}e^{u_3}w_{1,3} & if\ u_3 < 0 \\ w_{2,3}w_{1,3} & if\ u_3 \geq 0 \end{cases}\end{aligned} \quad (3.48)$$

where $u_1 = (w_{1,1}S + b_{1,1})$, $u_2 = (w_{1,2}S + b_{1,2})$ and $u_3 = (w_{1,3}S + b_{1,3})$. S is the input to the network, and for linearization, we use two samples drawn from either a normal or a uniform distribution. The sample is drawn in the same way as for MC sampling, explained in more detail in the section below. However, only two samples are used, and not several samples as for in the MC method. The reason for using two samples and not one is that one sample will give an invalid result when calculating the variance of the network output given in equation 3.46 as $N - 1$ when $N = 1$ is zero, and we get an invalid result with a zero-division.

However, using only two samples when the samples are randomly drawn for a distribution gives some variations in the result. Since the sample drawn will be

different each time, the result of the linearization will not be constant either. This is important to remember when the result is presented.

MC sampling

The second method, the MC sampling, only affect the network by providing the input matrix, X, to the network. This matrix is given by

$$X_{j,i} = \begin{cases} \sigma_j u_i + \mu & \textit{for varying input } \sigma^2 \\ \sigma u_i + \mu_j & \textit{for varying input } \mu^2 \end{cases} \tag{3.49}$$

where u_i is the samples randomly drawn from either a normal distribution or a uniform distribution and i is the number of samples drawn. The matrix is generated such that it can contain samples with a varying mean or variance. The number of samples decides how many columns the matrix has, while for each new input value of mean or variance, a new row in the matrix is added. This functionality makes it possible to have varying input values for mean and variance to the network in the same way as for the MC sampling method alone.

Gaussian quadrature

The Gaussian Quadrature method as input to the network looks a lot like the MC method. The input X is now given by

$$S_{j,i} = \begin{cases} \sigma_j x_i + \mu & \textit{for varying input } \sigma^2 \\ \sigma x_i + \mu_j & \textit{for varying input } \mu^2 \end{cases} \tag{3.50}$$

where x_i is the roots calculated using the Gaussian quadrature method based on n, numbers of quadrature points. This is instead of randomly drawn samples u_i as for the MC method. The number of roots, i will always be equal to n, and different values for n can be chosen to test how the performance change with a different number of quadrature points. The roots are either calculated for the normal distribution or the uniform distribution. In the same way, as for the MC method, the input mean and variance may have different values. Only now, the number of quadrature points chosen decides the number of columns in the S matrix, while each input value of either mean or variance gives a new row in the matrix.

Chapter 4

Results

4.1 Linearization

This hole section is a direct copy of section 4.1 in [26]. In this section, the results of the linearization method are presented.

4.1.1 Linearization versus benchmark

To see how the linearization performs, different nonlinear functions and scenarios are used.

The calculations of output mean and output variance for the linearization method are not dependent on a specific distribution. However, since the benchmark comes in two different versions, normally distributed and uniformly distributed, the choice of the benchmark may have something to say on how well the linearization performs against the benchmark dependent on which distribution the benchmark is based on.

The benchmark is based on the normal distribution in the upcoming results.

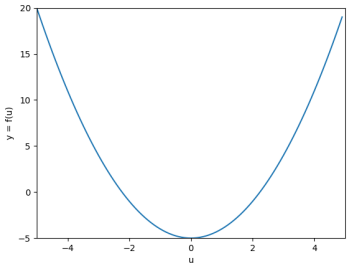


Figure 4.1: Illustration of function $f(u) = u^2 - 5$

Linearization of polynomial function - degree = 2

In this first example a polynomial function with $degree = 2$ i chosen. The function is given as $f(u) = u^2 - 5$ and has the shape shown in figure 4.1.

This function once with varying input mean and once with varying input variance is given as input to the method, and the output mean and output variance can be studied in the plots below.

Figure 4.3 shows how a varying input mean affects the result of the output mean. The mean vary from $\mu = -10$ to 10 while the input variance is kept constant with the value $\sigma^2 = 1$. The figure has two plots, the left plot has a complete overview of the result and the right plot is zoomed in around input mean $\mu = 0$. From the left graph, it looks like the approximated value and the exact value for output mean are the same. However, from the right plot, it is possible to see a deviation between the two graphs. The difference between the graphs is constantly equal to ≈ 1.0. To check if this is not only true for convex polynomial functions of degree two, it is also interesting to check for concave function. In figure 4.4 the result for the opposite concave function $f(u) = -u^2 + 5$ is displayed. The opposite function is displayed in figure 4.2. This yields the same result as for the convex function, except that approximation gives a lower instead of higher estimate compared to the exact value.

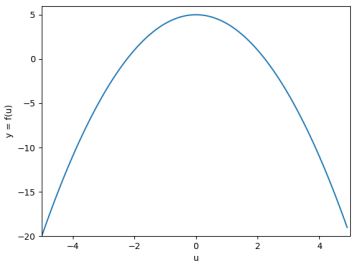


Figure 4.2: Illustration of function $f(u) = u^2 + 5$

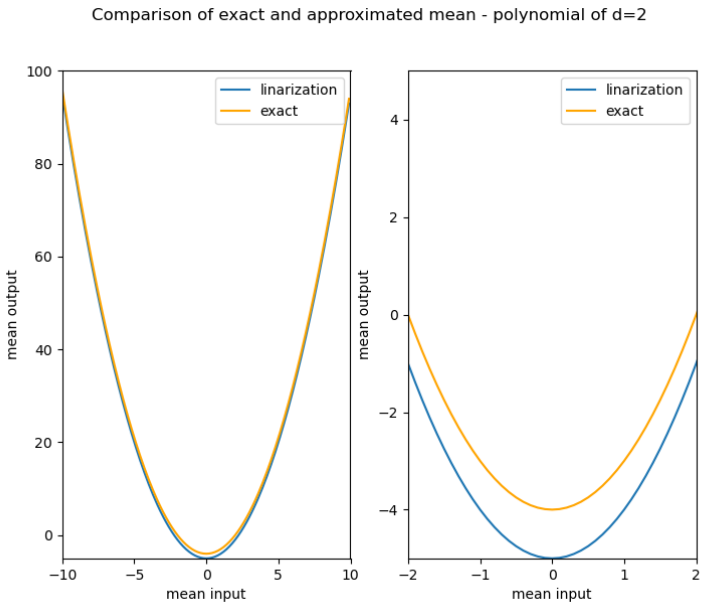


Figure 4.3: The plots show how the result of output mean changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$. The picture to the right is a zoomed-in version of the left picture.

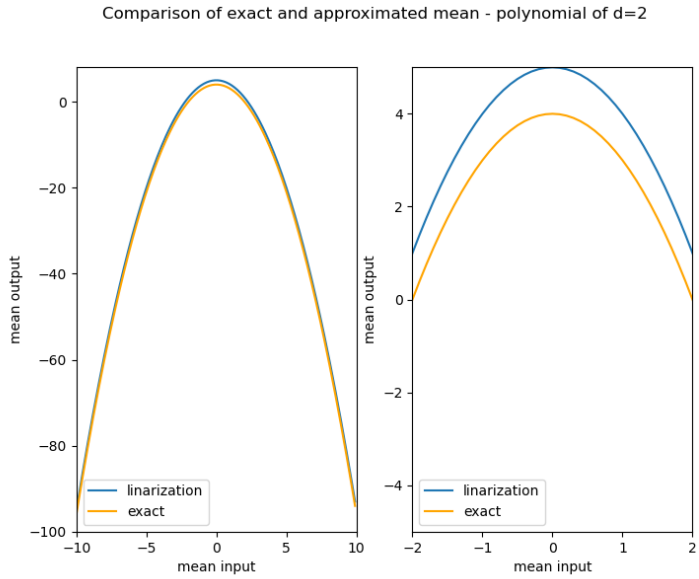


Figure 4.4: The plots show how the result of output mean changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$. The picture to the right is a zoomed-in version of the left picture.

Figure 4.5 shows how a varying input mean affects the result of the output variance. The input mean still varies at the same scale. The difference is now that the effect on the output variance is shown in the plot. The deviation between the approximated and exact values for output variance is now constantly equal ≈ 2.0. This apply to the convex and the concave function of degree 2.

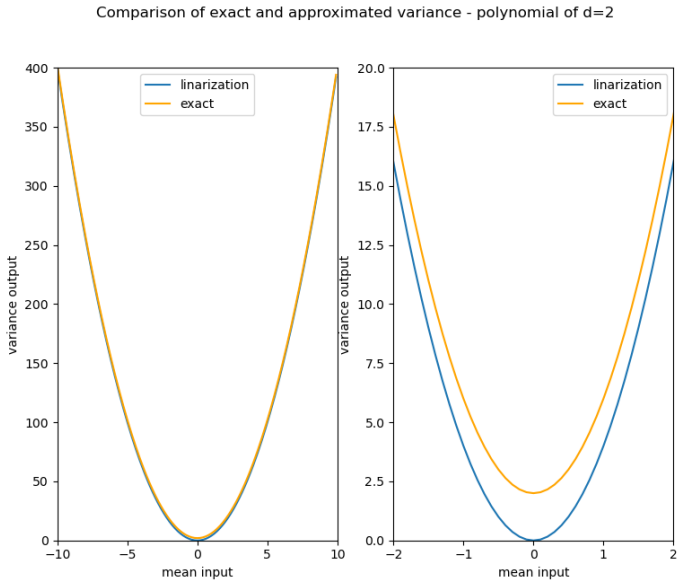


Figure 4.5: The plots show how the result of output variance changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$. The picture to the right is a zoomed-in version of the left picture.

Instead of varying the input mean, we now vary the input variance. The variance vary from $\sigma^2 = 0$ to 10., while the input mean is kept constant with the value $\mu = 1$. Figure 4.6 shows how varying the input variance affects the result of the output mean. This gives the same deviation between the approximation and the exact values for output mean, as for the output variance when the input mean was varying in figure 4.5. I.e. the deviation is equal ≈ 2.0.

This means that the value for the input mean does not affect the accuracy of the estimated output mean or output variance. This is because the deviation is the same for all input values of the mean.

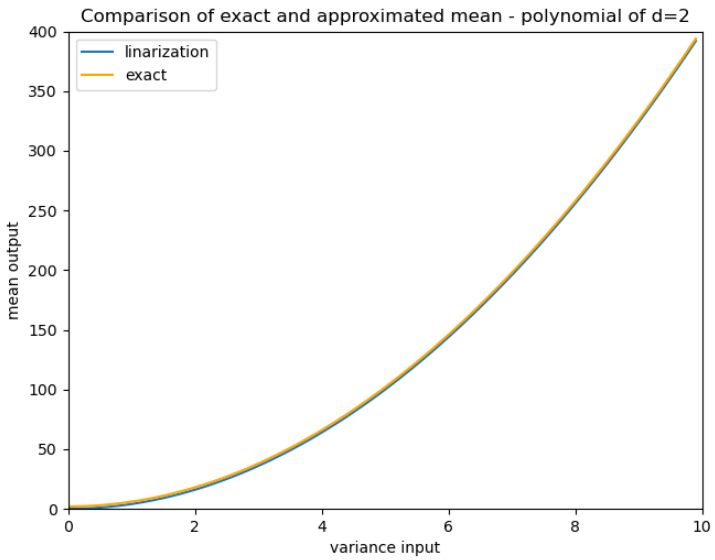


Figure 4.6: The plots show how the result of output mean changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 1$.

Figure 4.7 shows how varying the input variance affects the result of the output variance. This is the only combination of input and output which gives a deviation between the approximation and exact value that is not constant but increases with larger input variance. An input variance equal $\sigma^2 = 10$ gives a deviation between the approximated and exact value equal ≈ 196.0. The same apply for concave functions of degree 2.

This means that a varying input variance only affects the performance of the estimated value of output variance. The accuracy of the estimated value for output mean is the same for all input variance.

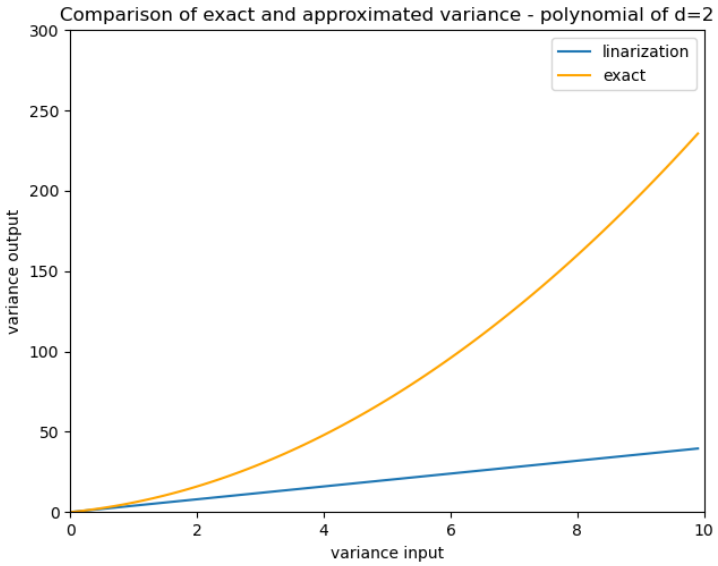


Figure 4.7: The plots show how the result of output variance changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 1$.

Linearization of polynomial function - degree = 3

Following the same procedure as in the section above, but now for a function of *degree*3 instead. The function used is given by $f(u) = -u^3 + 5$ and is illustrated in figure 4.8.

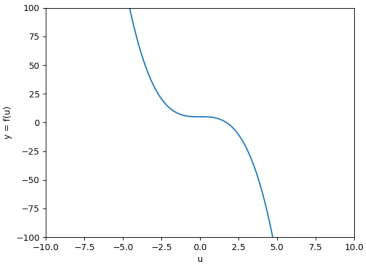


Figure 4.8: Illustration of function $f(u) = -u^3 + 5$

Based on figure 4.9, 4.10, 4.11 and 4.12 it is possible to see that increasing the degree of a polynomial function with one degree, from two to three, does not affect the result much, at least for figure 4.10, 4.11 and 4.12. On the other hand, figure 4.9 has got a completely new shape compared to figure 4.3. For the function of degree 2, the deviation between the approximated and exact values was constant. For degree 3, the approximated value gives a lower value for output mean than the exact value for negative values of input mean. When the input mean is zero, the deviation between approximated value and the exact value is ≈ 0. Then the approximated value gives higher output mean compared to the exact value for positive input mean. The deviation increases between the approximated value and exact value with increasing distance from $/mu = 0$. When $\mu = \pm 10$ the deviation is $\approx \pm 30.0$

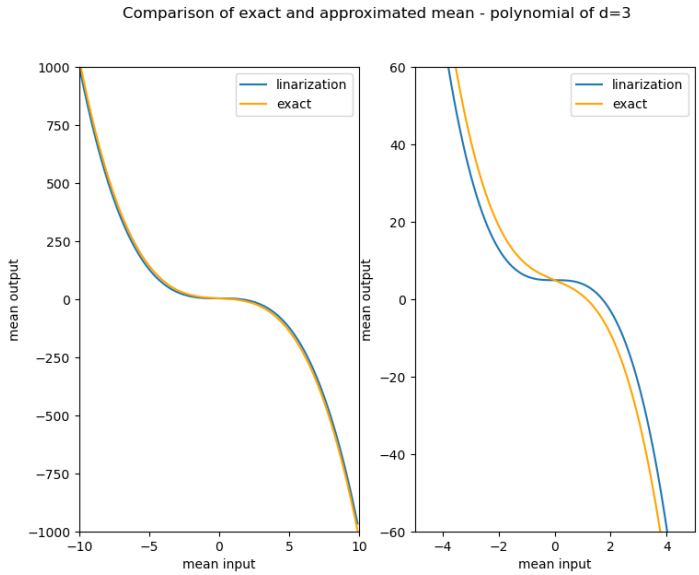


Figure 4.9: The plots show how the result of output mean changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$. The picture to the right is a zoomed-in version of the left picture.

The shape of figure 4.10 is the same compared to figure 4.5. However again the deviation between the approximated and exact values is no longer constant. When the input mean is equal to zero, the deviation is ≈ 15.0. With increasing input mean in a negative and positive direction, the deviation also increases. When $\mu = \pm 10$ the deviation is ≈ 3615.0.

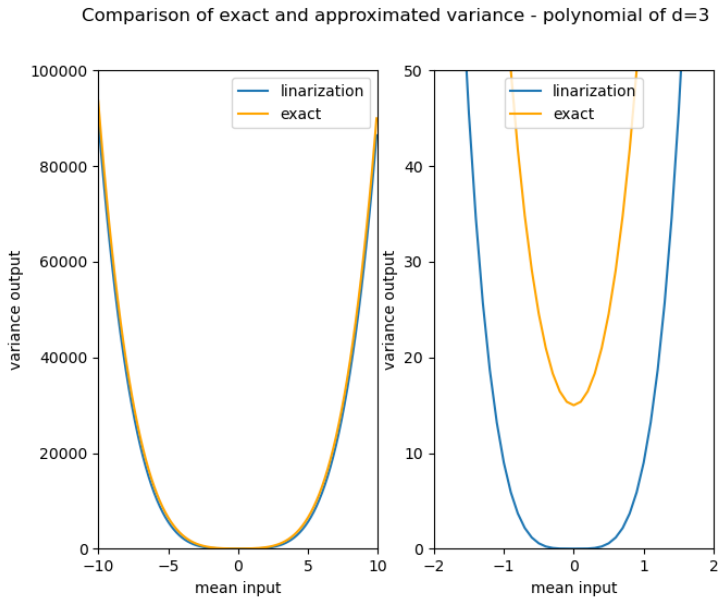


Figure 4.10: The plots show how the result of output variance changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$. The picture to the right is a zoomed-in version of the left picture.

The same change between degree 2 and degree 3 for varying input mean is possible to observe for varying input variance. The shape of figure 4.11 is the same compared to figure 4.6, but again, the deviation between approximated and exact values is increasing with increase in input variance. For $\sigma^2 = 0$ the deviation is ≈ 15.0 and increase to ≈ 3615.0 for $\sigma^2 = 10$. Again it is possible to see that the deviations are the same for output variance when varying input mean, and for output mean when varying input variance.

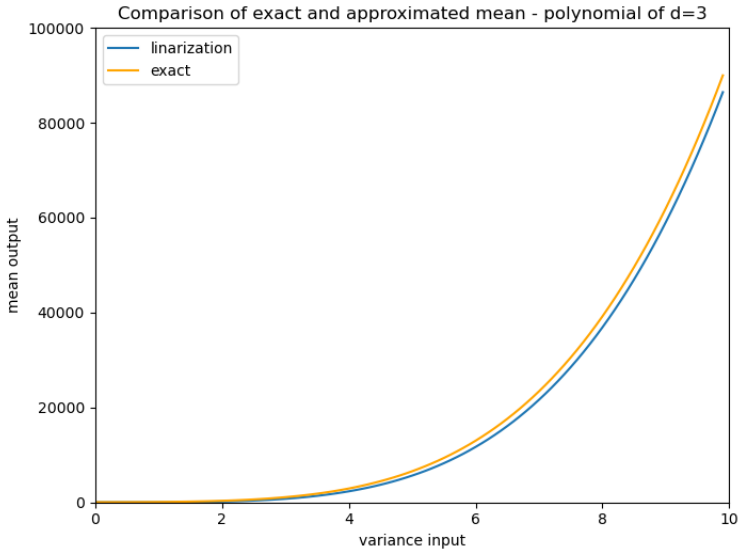


Figure 4.11: The plots show how the result of output mean changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 1$.

The deviation between approximated value and exact value did also increase with a higher value on the input variance in figure 4.7 for the function of degree 2. However, the deviation is even larger in figure 4.12. The deviation is ≈ 0 when input variance is zero, and increase to ≈ 18082.8 when $\sigma^2 = 10$.

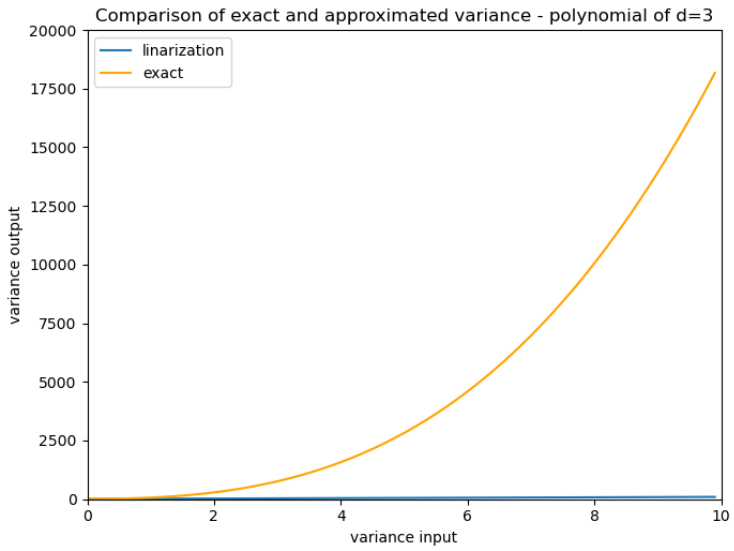


Figure 4.12: The plots show how the result of output variance changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 1$.

4.2 Monte Carlo sampling

This hole section is a direct copy of section 4.2 in [26]. In this section, the results of MC sampling will be presented.

4.2.1 Effect of randomly chosen samples

As briefly mentioned in chapter 3.3, drawing random samples from the given distribution gives different results of the method each time. To see how much random draw of samples has to say for the variation in the method some plots are made.

The plots below are generated for the function $f(u) = u^2 - 5$ and have input mean, $\mu = 0$, and input variance, $\sigma^2 = 4$. The function is illustrated in figure 4.1. In figure 4.13, the effect random samples chosen from a normal distribution has on output mean is illustrated for a different number of samples. It is possible to see that a low number of samples has a lot of variations in the estimated output. This again means that the estimate has low accuracy. It is possible that increasing the number of samples has a good effect on this issue. When the number of samples is 10^5 or higher, the samples are reasonably well centered around one value.

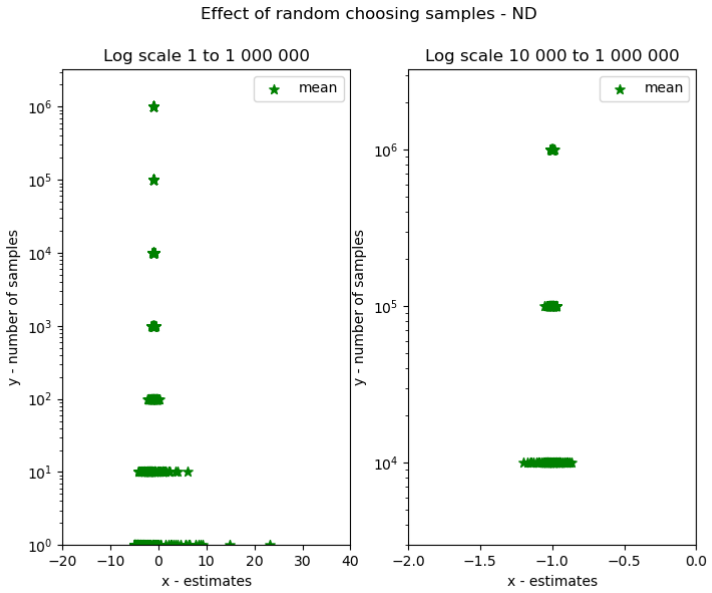


Figure 4.13: Effect of a random draw of samples from a normal distribution. The plot shows the variation in the output mean for a different number of samples. The y-axis represents the number of samples in a logarithmic scale from 1 to 1 million and the x-axis is the estimated value for output mean.

It is possible to see in figure 4.14 that the same tendency holds for output variance as it does for output mean. However, for variance, the spread in estimated values is even larger.

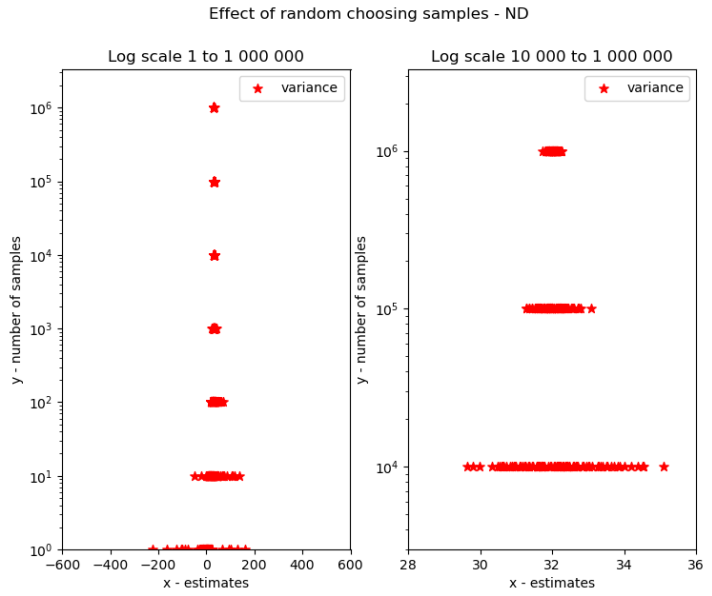


Figure 4.14: Effect of a random draw of samples from a normal distribution. The plot shows the variation in the output mean for a different number of samples. The y-axis represents the number of samples in a logarithmic scale from 1 to 1 million and the x-axis is the estimated value for output variance.

Figure 4.15 and 4.16show the same as 4.13 and 4.14, but now for the uniform distribution. When comparing the two results from the two distributions it is possible to see that the uniform distribution has a lower spread in the estimated values compared to the normal distribution. This holds especially for the variance. However, the tendency is quite clear for all the plots. Increasing the number of samples, decrease the spread in estimated value, which gives higher accuracy.

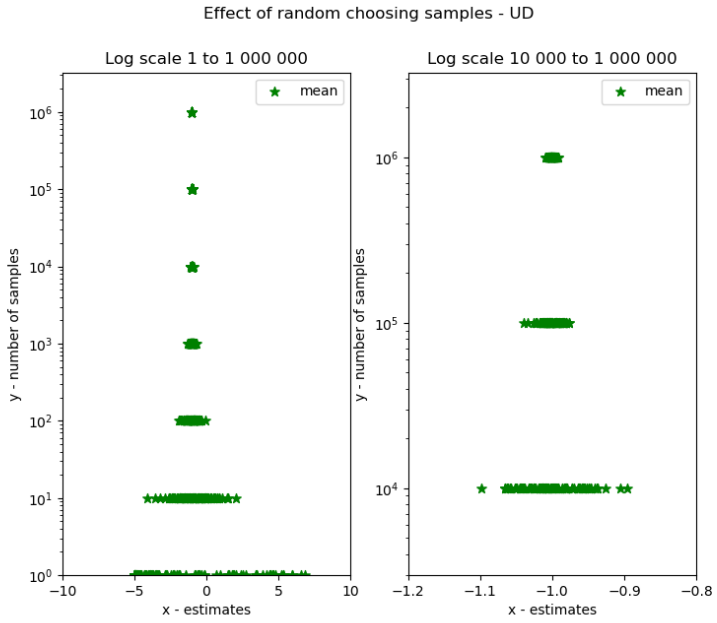


Figure 4.15: Effect of a random draw of samples from a uniform distribution. The plot shows the variation in the output mean for a different number of samples. The y-axis represents the number of samples in a logarithmic scale from 1 to 1 million and the x-axis is the estimated value for output mean.

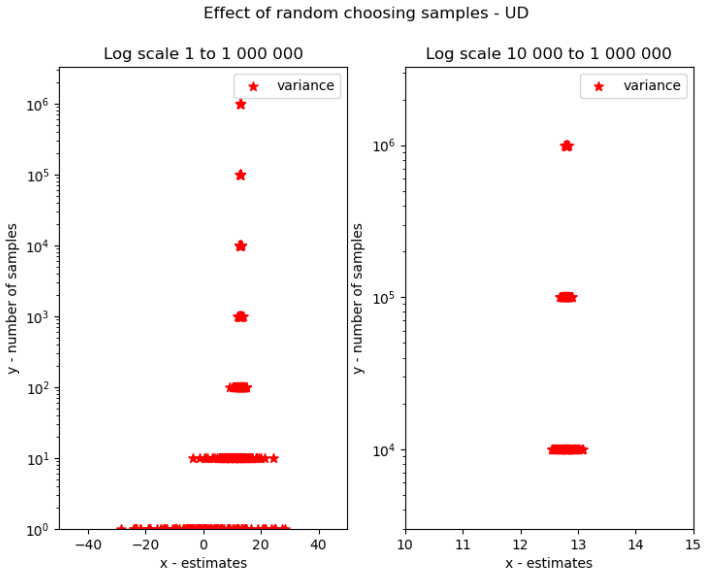


Figure 4.16: Effect of a random draw of samples from a normal distribution. The plot shows the variation in the output mean for a different number of samples. The y-axis represents the number of samples in a logarithmic scale from 1 to 1 million and the x-axis is the estimated value for output variance.

4.2.2 Monte Carlo sampling versus the benchmark

The same strategy is used here for linearization. One of the variables, input mean or input variance, is varying while the other input stays constant. However, since each number of samples is run ten times, instead of plotting each of these ten runs as a separate graph, the average over the samples at each input value is found and then the standard deviation of this average. Together with the exact value, this average over samples is plotted with the standard deviation.

All figures shown in this section are based on the normal distribution and not the uniform distribution. There are only small differences between the distributions. There are no cases where the deviation is approximated and the exact value is significantly bigger for one of the distributions compared to the other. The main focus will therefore be the normal distribution.

To begin with, the same polynomial function of degree 2, used for linearization is also used to check the performance of sampling. This function is $f(u) = u^2 - 5$ and has the shape shown in figure 4.1.

1 million samples - polynomial function of degree 2

Figure 4.17, 4.18, 4.19 and 4.20 is run with 1 million samples. In general, for the four plots, it's hard to distinguish the approximation from the exact value. This

mean that the approximation is quite accurate. The standard deviation is also very close or approximately lying on the exact value.

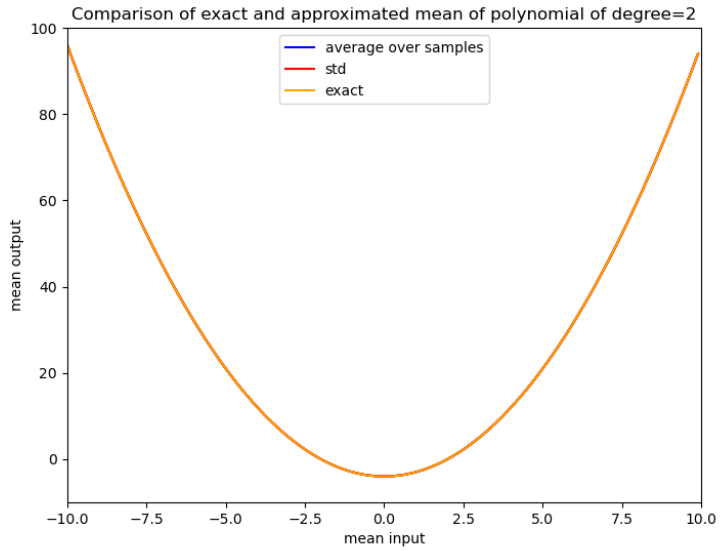


Figure 4.17: The plots show how the result of output mean changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$.

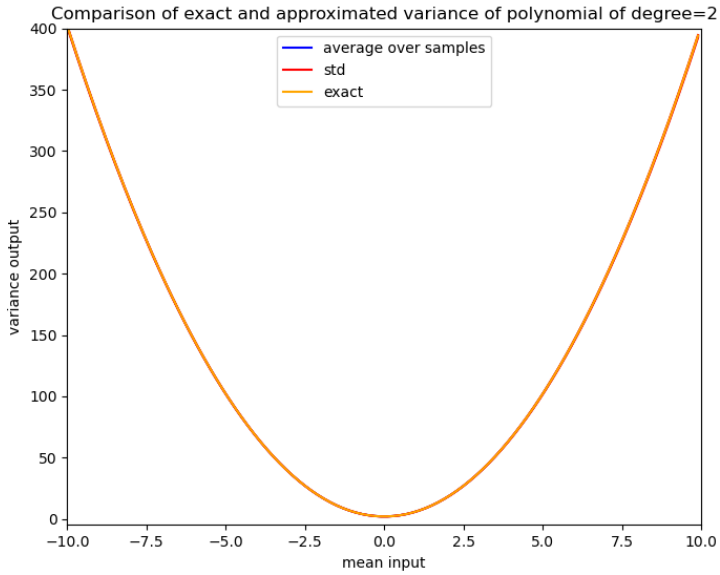


Figure 4.18: The plots show how the result of output variance changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$.

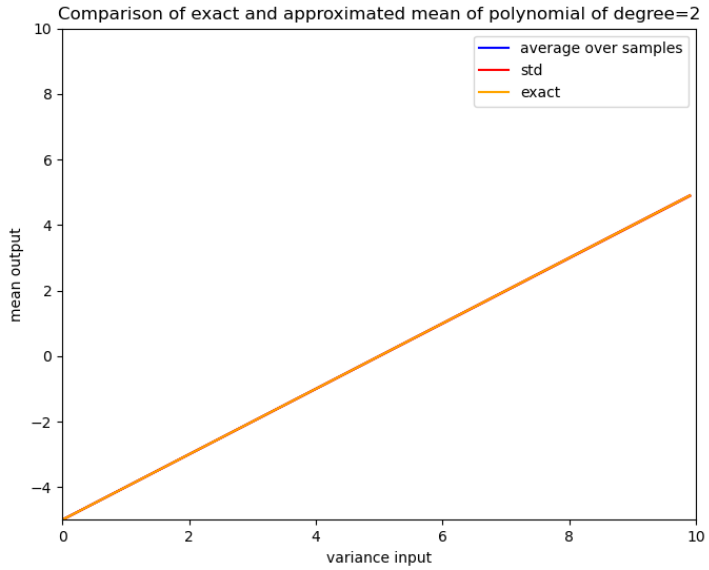


Figure 4.19: The plots show how the result of output mean changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 0$.

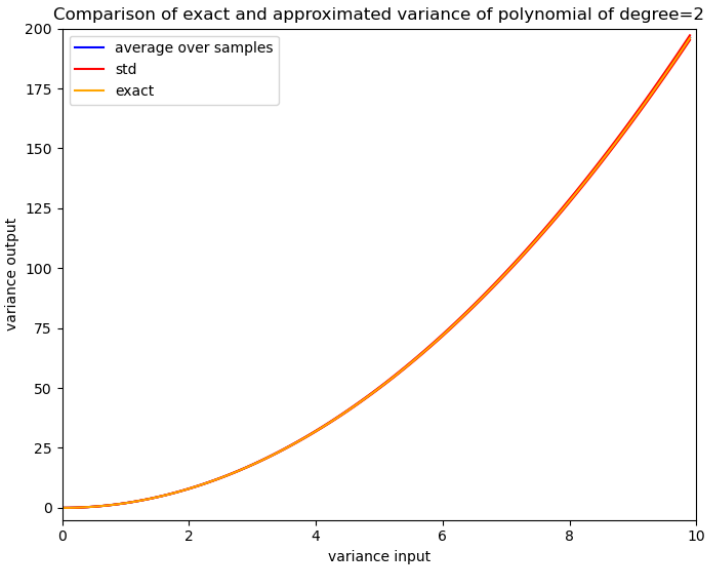


Figure 4.20: The plots show how the result of output variance changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 0$.

1000 samples - polynomial function of degree 2

Figure 4.21, 4.22, 4.23 and 4.24 use 1000 samples instead of 1 million as the figures above. It is now clear that an increase in input mean in both the negative and positive direction away from $\mu = 0$ gives a larger deviation between the approximated and exact value for both output mean and output variance. However, the effect is larger for variance compared to mean.

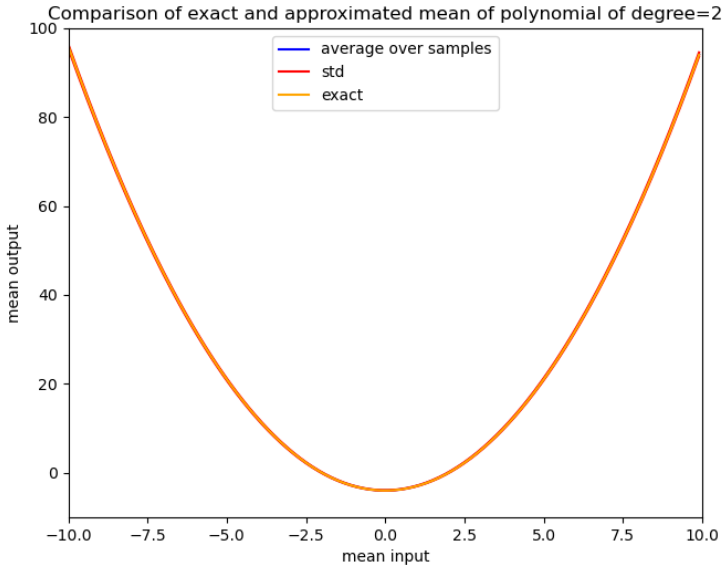


Figure 4.21: The plots show how the result of output mean changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$.

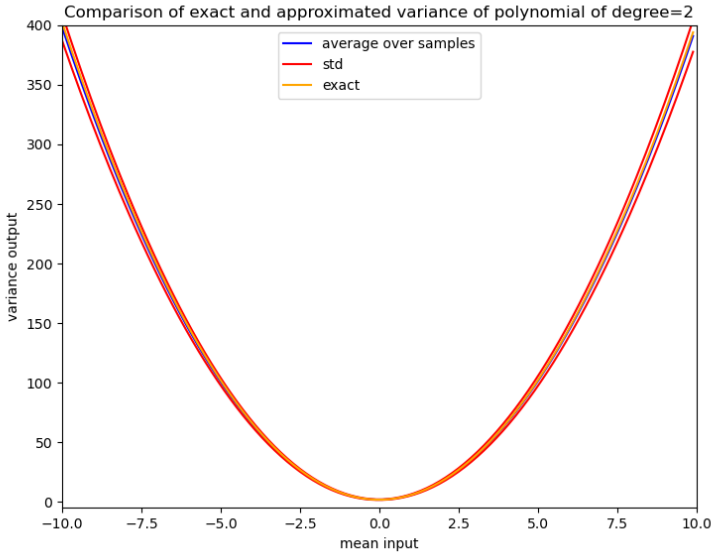


Figure 4.22: The plots show how the result of output variance changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$.

It is also possible to see in Figure 4.23 and 4.24 that an increase in input variance gives a larger deviation between the approximated and exact value for output mean. However, for output variance, the approximated value is still close to the exact value. For both output mean and output variance an increase in standard deviation is observed when the input variance increases.

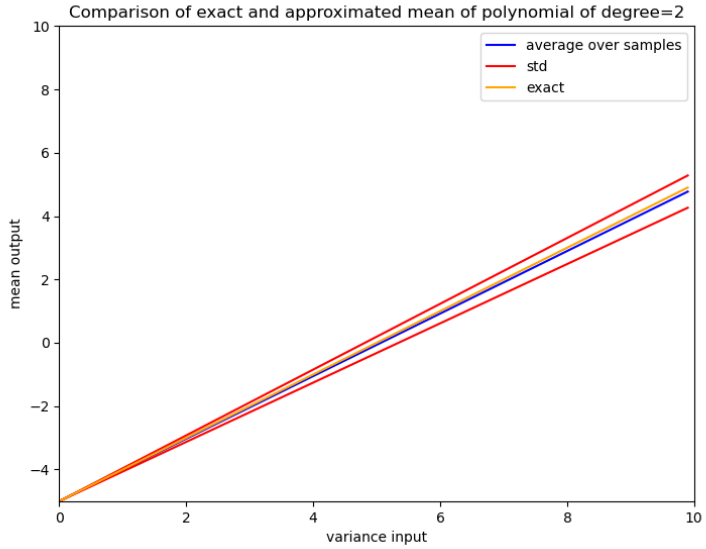


Figure 4.23: The plots show how the result of output variance changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 0$.

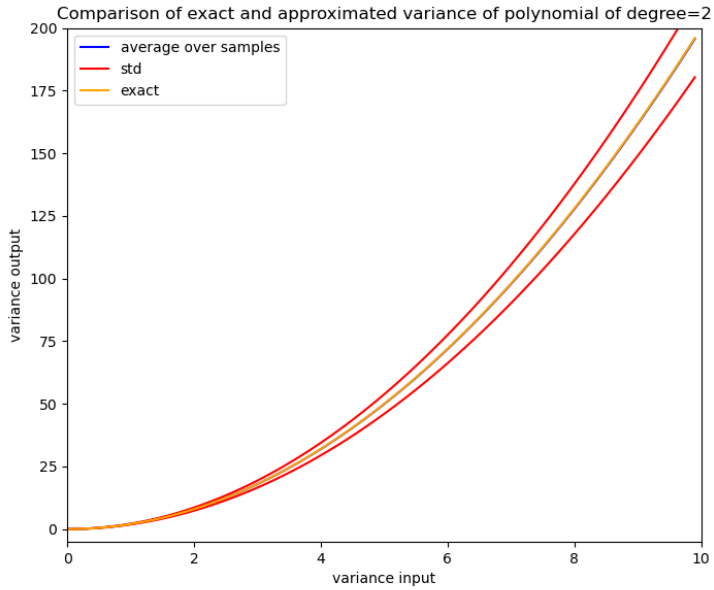


Figure 4.24: The plots show how the result of output variance changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 0$.

When performing linearization it was also checked if the concave function of a polynomial with degree 2 behaved differently than the convex. Doing the same

for MC sampling, with the function $f(u) = -u^2 + 5$ (figure 4.2) the result are displayed in figure 4.25. It is possible to see that the behavior is the same as for the convex version in 4.24 with an increase in output variance when the input variance is increasing. This is also the case for the other combinations of input and output. If the function is convex or concave has nothing to say for this behavior. However, it is possible to see a larger deviation between approximated and exact values for the concave function in figure 4.25 compared to the convex function.

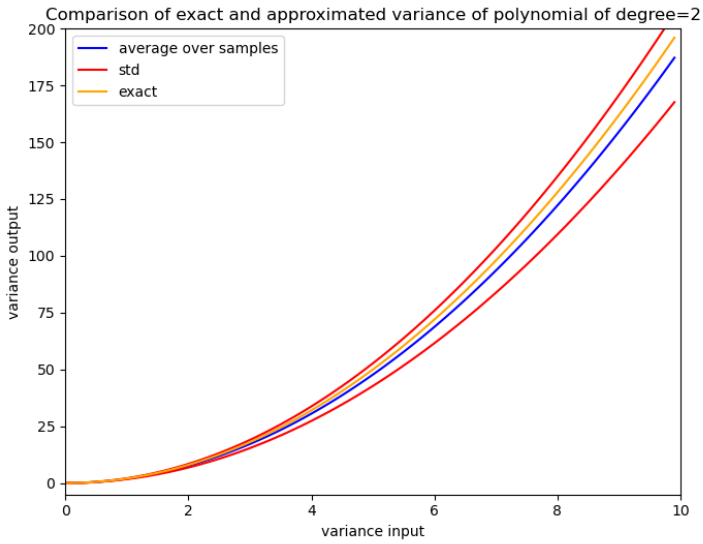


Figure 4.25: The plots show how the result of output variance changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 0$.

1000 samples - polynomial function of degree 3

As for linearization, the MC method is also implemented for polynomial function of degree 3, given by $f(u) = -u^3 + 5$ and illustrated in figure 4.8. The result compared to 1000 samples with the polynomial function of degree 2 is quite similar. Figure 4.26 and 4.27 seems to have the same deviation between the approximated and exact value since figure 4.21 and 4.22 has smaller interval on the y-axis. The deviation is larger for the degree 3 function compared to the degree 2 function.

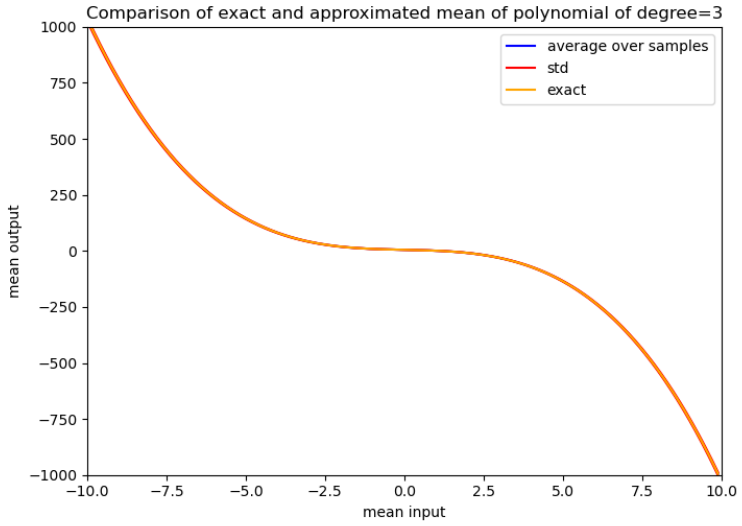


Figure 4.26: The plots show how the result of output mean changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$.

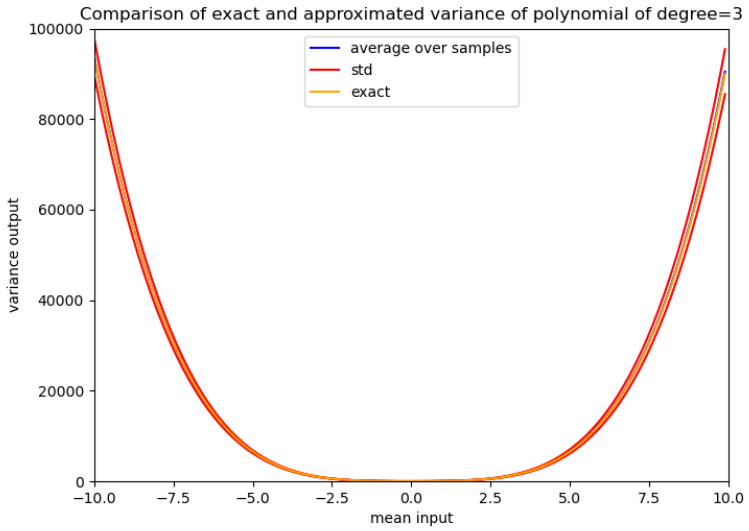


Figure 4.27: The plots show how the result of output variance changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$.

Figure 4.28 shows quite a large deviation between the approximated and exact value. This also gives a large standard deviation. Compared to the similar case for degree 2, the deviation has increased with the complexity of the function.

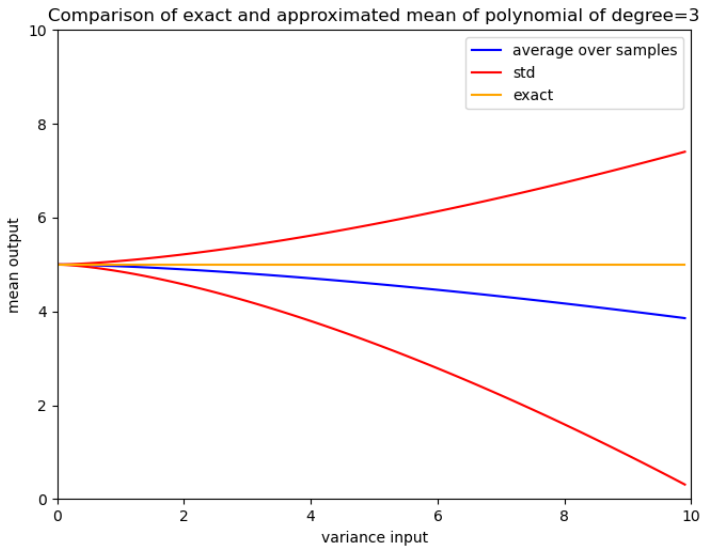


Figure 4.28: The plots show how the result of output mean changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 0$.

Figure 4.29 is also quite similar to the corresponding figure 4.24 for degree 2. The scale on the y-axis still has a much larger interval, such that the real deviation is larger for the degree 3 functions.

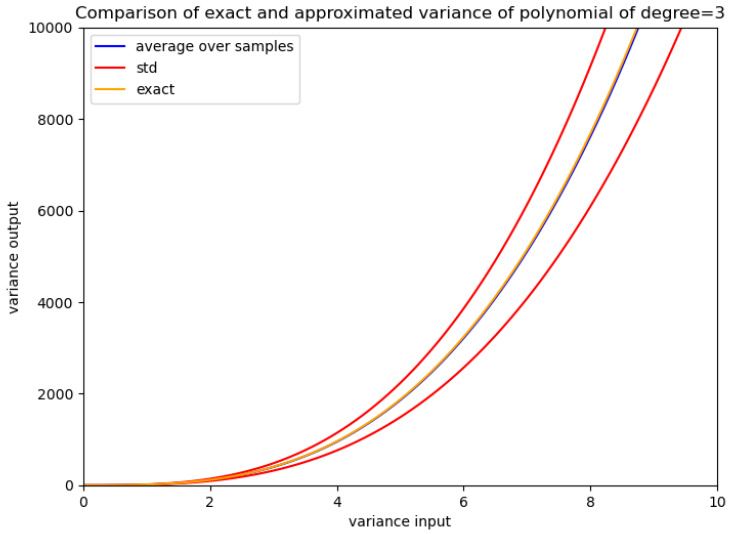


Figure 4.29: The plots show how the result of output variance changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 0$.

4.3 Gaussian Quadrature

In this section, the results of the Gaussian quadrature method are presented.

4.3.1 Gaussian quadrature versus the benchmark

The same procedure as for the two previous methods is also used for Gaussian Quadrature for generating results. Varying one input variable, mean or variance, at a time will show how the method responds to different values and how the output mean or variance is affected by the change. Since the same procedure as for the two other methods is used, the layout of the plots will be quite similar. However, a new variable n, known as the quadrature point, is also an important variable affecting the outcome of the Gaussian method. Therefore, different values for this variable are used, and the effect will be shown in the upcoming plots.

Results for both the normal distribution and uniform distribution will be displayed.

4.3.2 Normal distribution

Gaussian Quadrature of polynomial function - degree = 3

In this first example a polynomial function with $degree = 3$ i chosen. The function used is the same as for the two previous methods and is given by $f(u) = -u^3 + 5$ and is illustrated in figure 4.8. The plots below show how well the approximation is for different input values for mean and variance, but the resulting approximation is also given for two different numbers of quadrature points. The result displayed in the left plot in each figure is approximated using two quadrature points, while the right one uses four quadrature points.

In figure 4.30 is it possible to see that the Gaussian quadrature approximation is performing well compared to the benchmark, as the two graphs are perfectly aligned for all input values of the mean. It is not possible to spot the blue line that shows the approximation at any point because it is at all times covered by the benchmark. This holds for both two and four quadrature points.

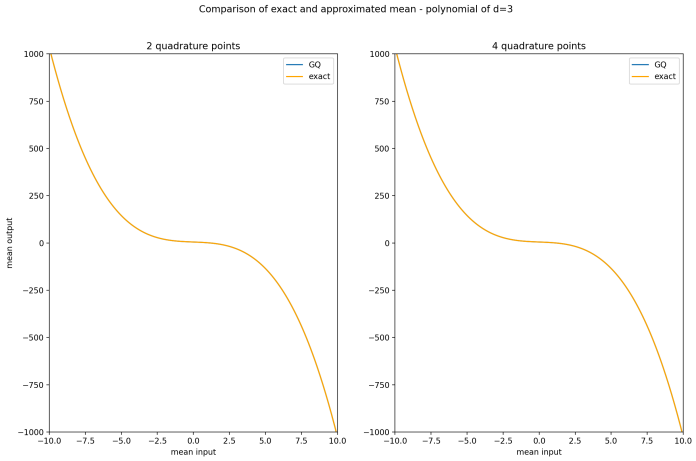


Figure 4.30: The plots show how the result of output mean changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$.

The same holds for the four quadrature points in figure 4.31. However, for two quadrature points, the blue graph is now visible, and there is a deviation between the approximation and the benchmark. Figure 4.31 shows how the output variance is affected by varying input mean. By experience from the two previous methods, the output variance is usually more difficult to get a good approximation of, and it is now possible to see that the chosen number of quadrature points has something to say for the accuracy.

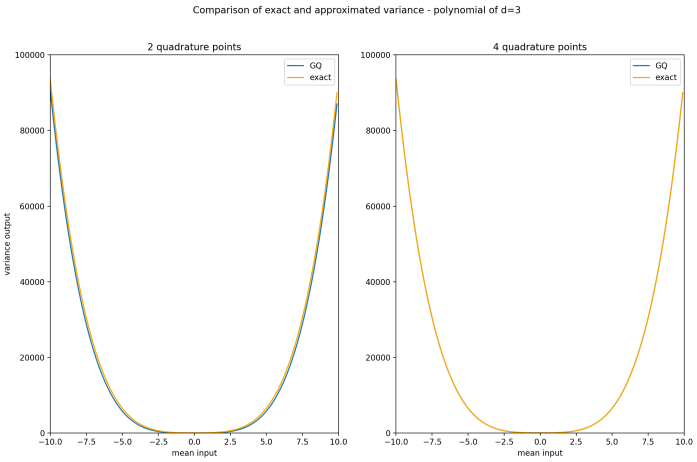


Figure 4.31: The plots show how the result of output variance changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$.

Figure 4.32 again shows that it is easier to get a good approximation for the output mean. This time-varying input variance for both two and four quadrature points gives a perfect approximation for both values of n.

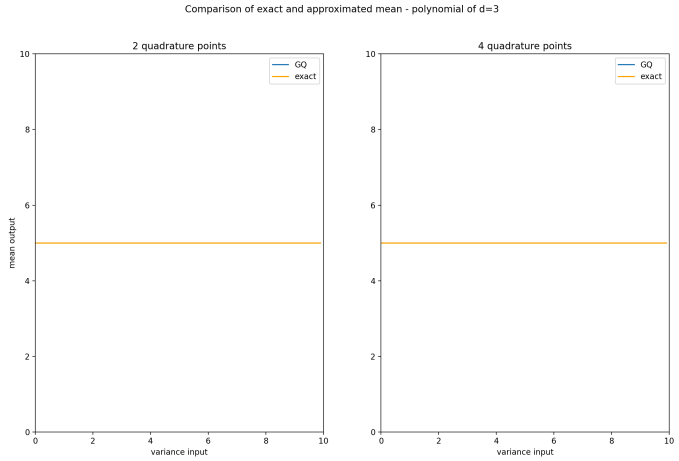


Figure 4.32: The plots show how the result of output mean changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 0$.

The larges deviation between the approximation and benchmark is possible to observe in figure 4.33 for two quadrature points. However, with four quadrature points, the Gaussian quadrature method performs well and gives an accurate approximation compared to the benchmark. Again observing that the output variance is harder to approximate.

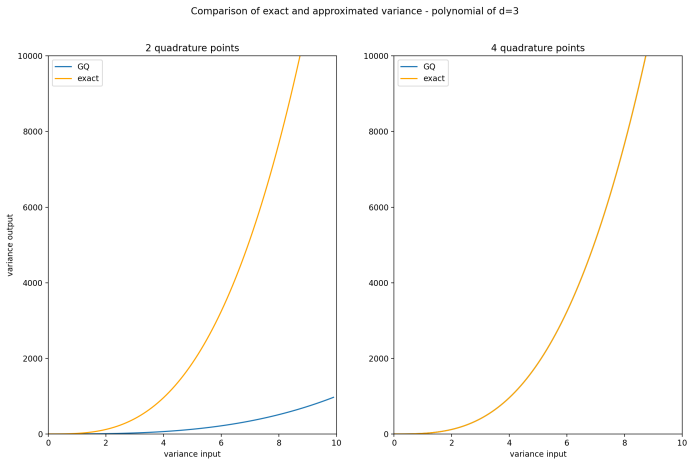


Figure 4.33: The plots show how the result of output variance changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 0$.

Gaussian quadrature of polynomial - degree = 5

In the next example, the polynomial used has more degree than the examples up to this point. In this section a polynomial of $degree = 5$ i used. The function is given by $f(u) = u^5 + u^4 - 25u^3 - 36u^2 + 25u + 200$ and is displayed in figure 4.34.

This time the approximation is made for three different numbers of quadrature points to get an even better visual view of how this n variable affects the result. For the polynomial of $degree = 5$, the result is shown for two, three, and six quadrature points.

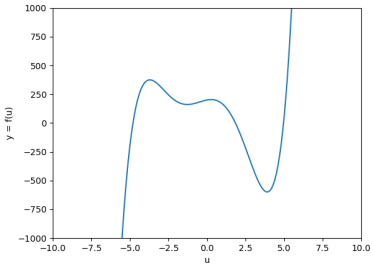


Figure 4.34: Illustration of function $f(u) = u^5 + u^4 - 25u^3 - 36u^2 + 25u + 200$

In figure 4.35 it is possible to see that the Gaussian quadrature with a higher degree polynomial still has a quite good approximation for the output mean. Based on theory, we know that six quadrature points will give an accurate approximation. However, it is possible to see that three quadrature points also perform well and is hard to separate from the benchmark. The quadrature point has to be reduced to only $n = 2$ before it is more obvious that the approximation not always corresponds to the benchmark.

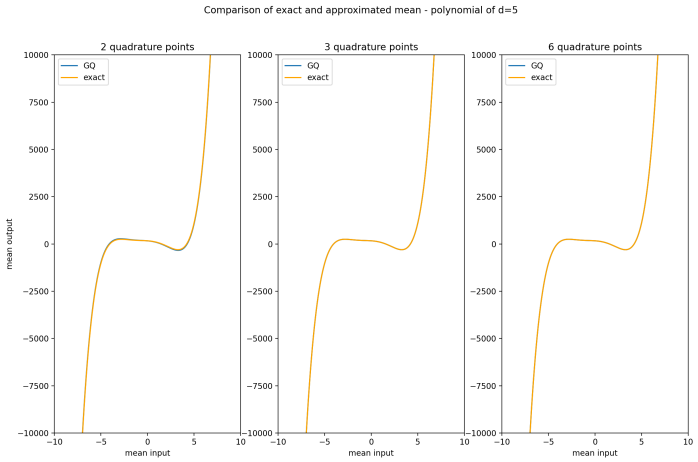


Figure 4.35: The plots show how the result of output mean changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$.

Again the Gaussian quadrature method has more difficulties with approximating variance output. However, the increase in performance is quite visual when it comes to increasing numbers in the chosen quadrature point for 4.36. With $n = 6$ the method performs as assumed, and the approximation has the same result as the benchmark. There is also a visual difference between the performance of two and three quadrature points. With three points, the approximation is close to the shape of the benchmark. However, with some deviations along the line. With two quadrature points, the Gaussian quadrature approximation struggles with getting a result close to the benchmark.

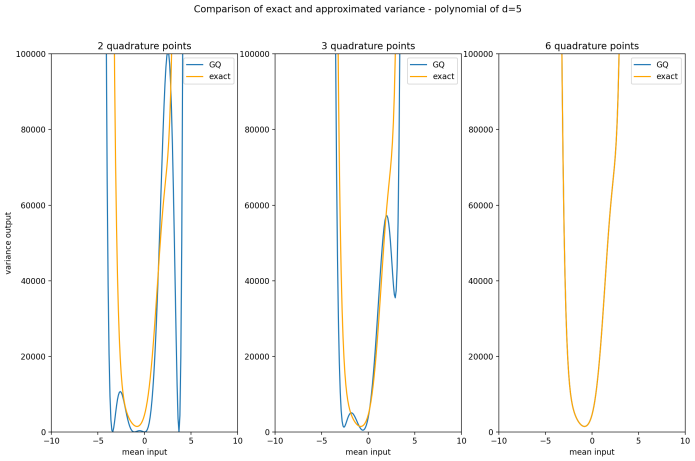


Figure 4.36: The plots show how the result of output variance changes with a varying input mean. The input mean vary from $\mu = -10$ to 10 and the input variance is constant $\sigma^2 = 1$.

Figure 4.32 again shows that for output mean, it is not necessary to have $n > d$ to get an approximation that is close to the benchmark, as we see for three quadrature points on this five-degree polynomial. However, it is also important that this is not assumed for every example and number of quadrature points. In this example, there is quite an obvious difference using two and three quadrature points. Without the ability to check the performance, choosing a too low number for n can give a very bad approximation.

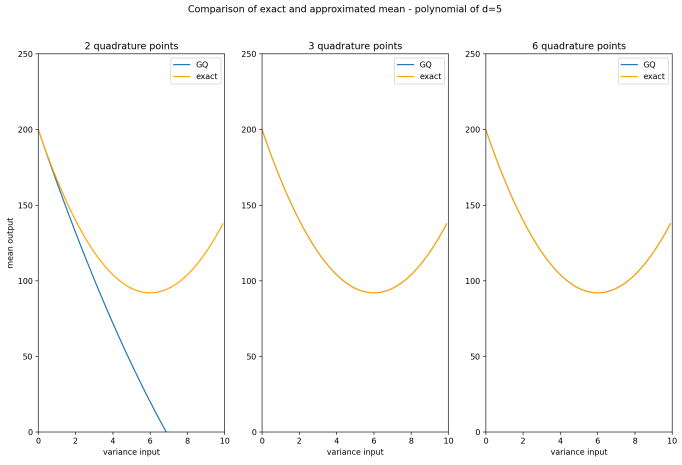


Figure 4.37: The plots show how the result of output mean changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 0$.

For the last input-output relation, in figure 4.38, we see that the observations done for other input-output relations are also in line with this example.

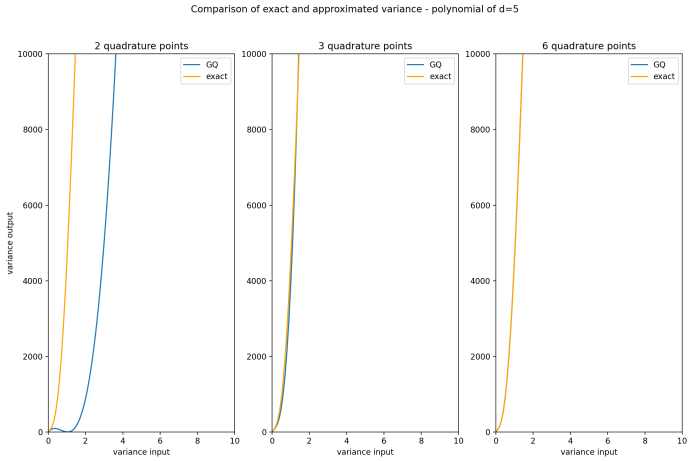


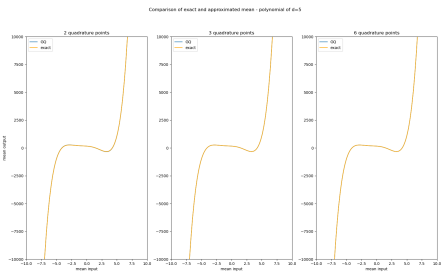
Figure 4.38: The plots show how the result of output variance changes with a varying input variance. The input variance vary from $\sigma^2 = 0$ to 10 and the input mean is constant $\mu = 0$.

4.3.3 Uniform distribution

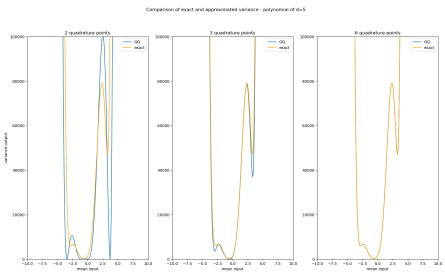
Gaussian quadrature on polynomial function - degree = 5

As mentioned, the Gaussian quadrature method is also adapted to the uniform distribution. It is therefore relevant to see if there are some differences in performance between the Gaussian quadrature method compared with the normal distribution. The Gaussian quadrature with uniform distribution is used on the same polynomial function of $degree = 5$ as in the section above, i.e, $f(u) = u^5 + u^4 - 25u^3 - 36u^2 + 25u + 200$. The same three quadrature points are also used, i.e, $n = 2, n = 3$ and $n = 6$. Other parameters are also the same, so the presentation of the results is the same.

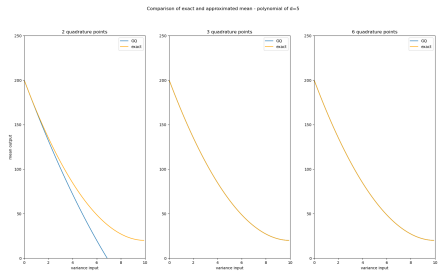
In figure 4.39 the approximations generated from the method are shown. Compared to the methods using the normal distribution, the result for each input-output relation is identical or nearly identical to the corresponding input-output relation generated from the normal distribution. The method is still accurate for six quadrature points, and the same decrease in performance is visual with the decrease in the number of chosen quadrature points.



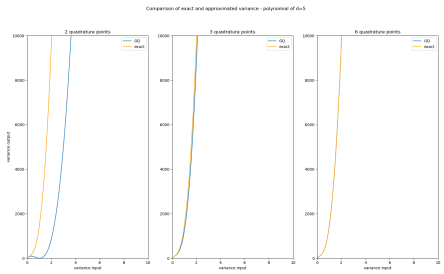
(a) output mean for varying input mean



(b) output variance for varying input mean



(c) output mean for varying input variance



(d) output variance for varying input variance

Figure 4.39: Uniform distributed Gaussian quadrature on polynomial function degree 5 with quadrature point n = 2,3,6

4.4 Neural Network

Up to this point, the results for each method alone are presented. Now the result of each method, in combination with the neural network defined in section 3.6 will be presented.

Linearization

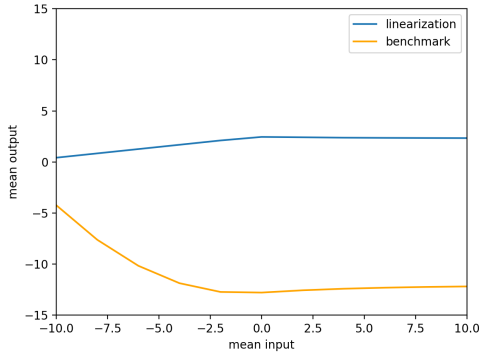
In this section, the results of linearizing the non-linearity in the Haldane network are presented. As mentioned in section 3.6.3, the main contribution of the linearization method is inside the neural network, and the input to the neural network is chosen to be two samples. These samples are either drawn from the normal or the uniform distribution. The samples will also be drawn randomly from distributions with varying inputs. The input values for the samples are either

1. $\mu = -10$ to 10 with stepsize 2.0, and constant input variance $\sigma^2 = 1$, or
2. $\sigma^2 = 0$ to 10 with stepsize 2.0 and constant input mean $\mu = 0$

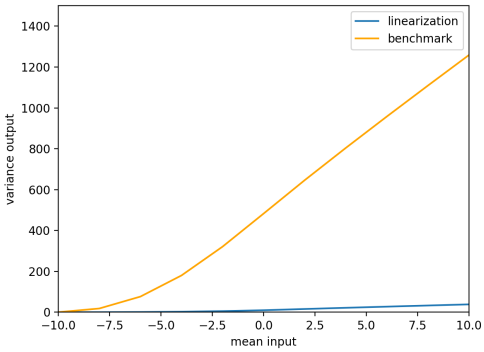
This gives the result of the linearization with normally distributed samples as input to the Haldane network presented in figure 4.40 compared to the benchmark defined in

section 3.6.3. Plot a), and b) are generated with input 1. and c) and d) is generated using input 2.

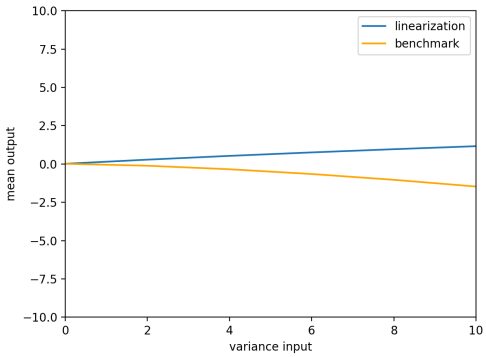
The deviations seen for both the output mean and the output variance are probably due to both the benchmark it is compared with and the low number of samples used as input to the network. As mentioned, the low number of samples gives randomness to the result since the sample is chosen from a distribution. However, this is not the only reason for the deviation compared to the benchmark. The benchmark used is also based on the sampling method. When a linearization is performed inside the network, the output shape changes compared to if the linearization had not happened. When a linearization is performed inside the network, the output shape changes compared to if the linearization had not occurred. In this case, it can be assumed that this change has led to a greater difference than if the linearization had not been done. It is important to point out that this is only an assumption.



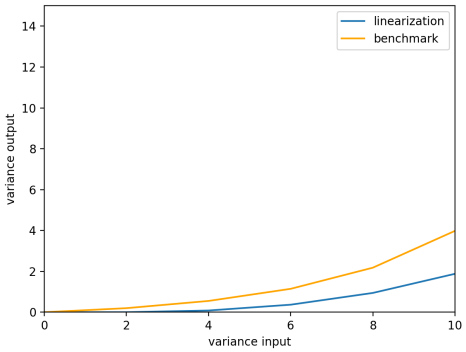
(a) output mean for varying input mean



(b) output variance for varying input mean



(c) output mean for varying input variance

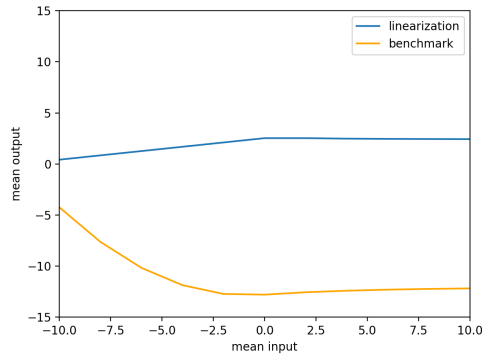


(d) output variance for varying input variance

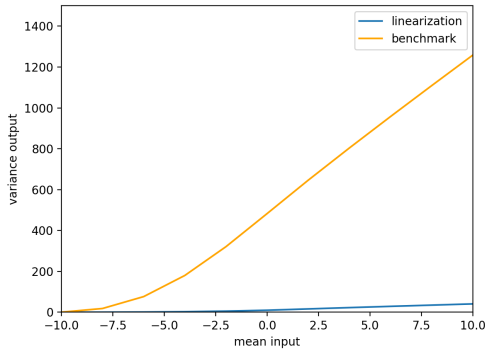
Figure 4.40: Linearization of the Haldane network with normally distributed samples as input is compared against the benchmark for different input values for mean and variance

In the figure 4.41, the same plot is shown once more. Only the selected distribution used for the samples as inputs to the Haldane network has been changed. This time the samples are drawn from a uniform distribution. From the two figures in this

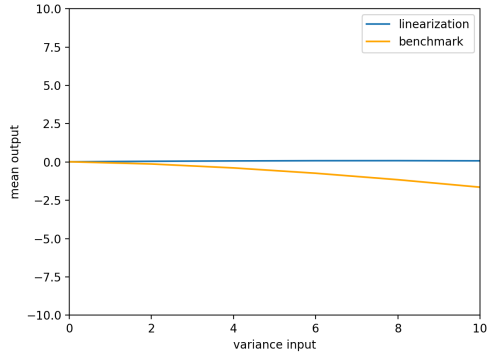
section, we see that using a uniform distribution instead of a normal distribution does not have much effect on the results. The results vary slightly for the different input-output relations, but overall the results are way off the benchmark.



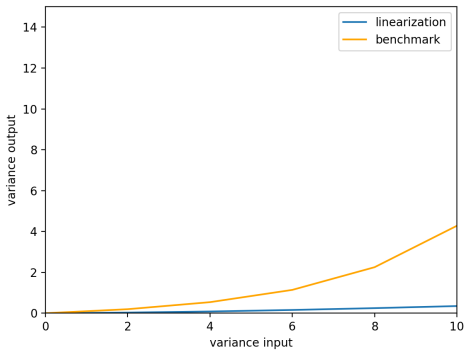
(a) output mean for varying input mean



(b) output variance for varying input mean



(c) output mean for varying input variance



(d) output variance varying input variance

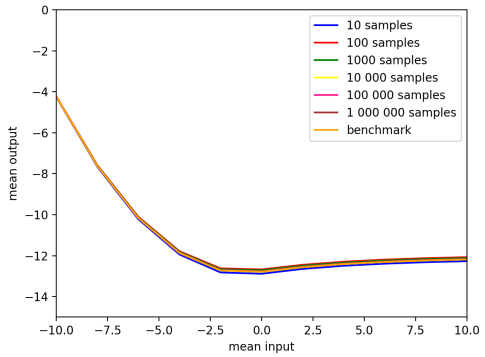
Figure 4.41: Linearization of the Haldane network with uniformly distributed samples as input is compared against the benchmark for different input values for mean and variance

4.4.1 Monte Carlo sampling

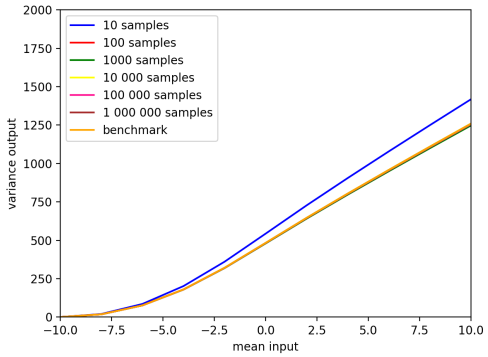
In this section, the MC sampling is used as input to the Haldane network, and the result is presented. Again, the samples are either subtracted from the normal distribution or the uniform distribution, and the input values for mean and variance are given the same way as for linearization and listed in 2.

In figure 4.42, the results for a different number of samples drawn from a normal distribution are plotted against the benchmark. From the plots displayed in the figure, it is possible to see that a higher number of samples generally gives a result closer to the benchmark. The method also does generally well, and you can see that the shape of the graphs agrees relatively well with the shape of the benchmark. This is initially not very surprising as the benchmark is based on the sampling method

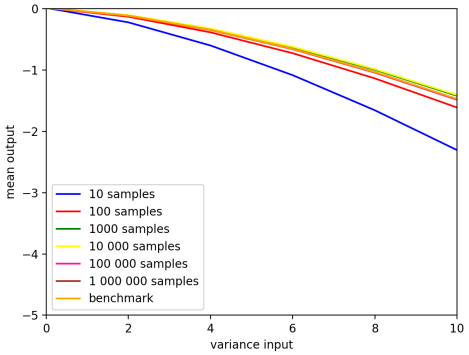
with an even higher number of samples than those produced in the experiment. It is therefore natural that the increase in the number of samples also produces a result closer to the benchmark. It is also possible to see that it is not necessary with a very large amount of sample to approach the benchmark. From approximately 100 thousand samples, the resulting graph is so close to the benchmark that it is hard to separate from the benchmark for all four examples given in the figure.



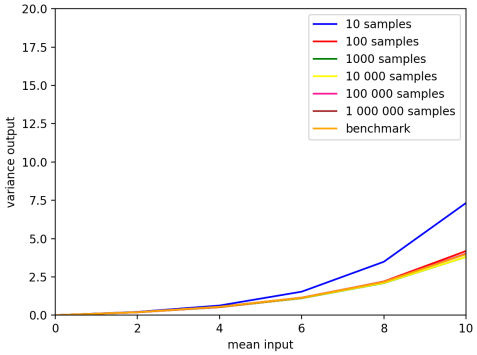
(a) output mean for varying input mean



(b) output variance for varying input mean



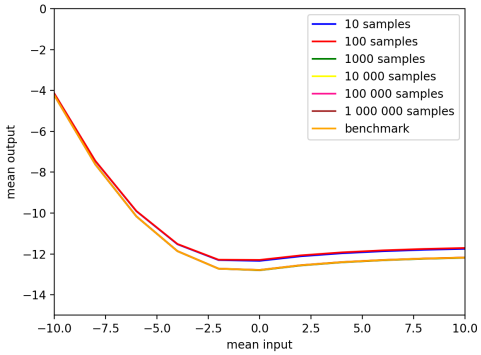
(c) output mean for varying input variance



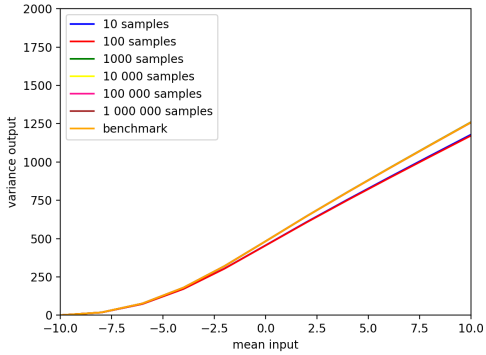
(d) output variance for varying input variance

Figure 4.42: Monte Carlo sampling with normally distributed samples as input to Haldane network compared against the benchmark for different input values for mean and variance

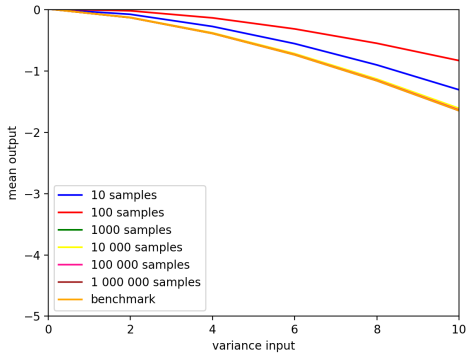
In the figure 4.43, the distribution is changed from normally distributed to uniformly distributed. This does not seem to mean much for the performance of the method. However, for plot c), the behavior of 100 samples is unexpected. The result of 100 samples lies further away from the benchmark compared to 10 samples. In general, we see that the performance increases with more samples. Nevertheless, we see the tendency also in a) and b) where the result of 10 and 100 samples is very similar.



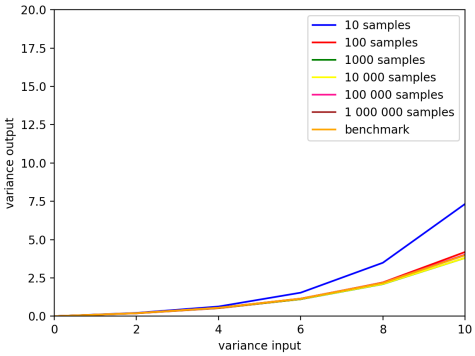
(a) output mean for varying input mean



(b) output variance for varying input mean



(c) output mean for varying input variance



(d) output variance for varying input variance

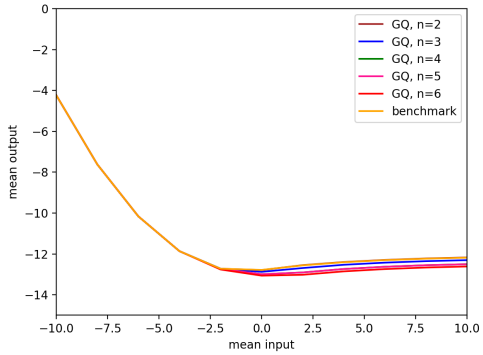
Figure 4.43: Monte Carlo sampling with uniformly distributed samples as input to Haldane network compared against the benchmark for different input values for mean and variance

4.4.2 Gaussian quadrature

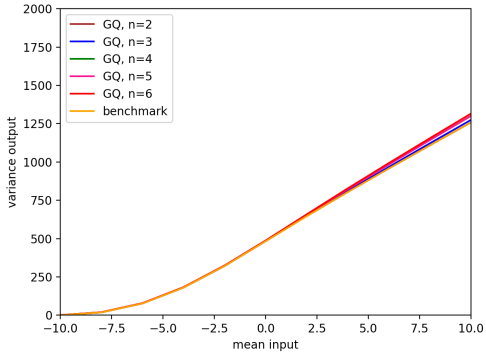
The last experiment is done using the roots calculated from the Gaussian quadrature as input to the Haldane network. These roots are calculated for both normal and uniform distribution for varying input mean or variance. The same variations in mean and variance are used for Gaussian quadrature as for the two previous methods, see list 2.

The result for the normal distributed Gaussian quadrature is given in figure 4.44. The results are surprising because a higher number of quadrature points used for calculating roots give a result further away from the benchmark compared to a smaller number of quadrature points. In other words, increasing the number of quadrature points results in a larger deviation from the benchmark. In section 4.3, where the performance of the Gaussian quadrature method is studied alone, the opposite was observed. A higher number of quadrature points gave a smaller deviation to the benchmark. Choosing the correct number of quadrature points for

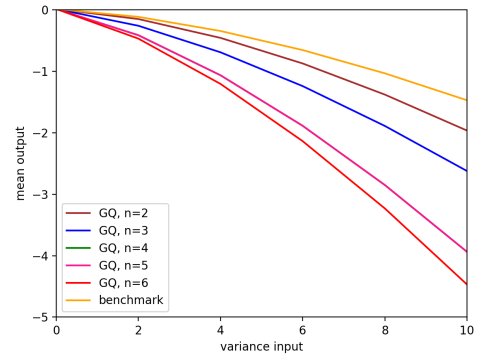
the specific task also gave a perfect approximation. Therefore, observing the opposite result for Gaussian quadrature as input to the Haldane network is quite surprising.



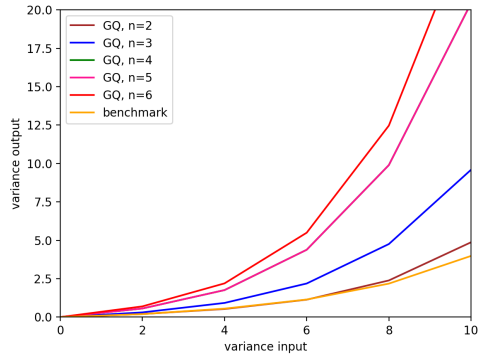
(a) output mean for varying input mean



(b) output variance for varying input mean



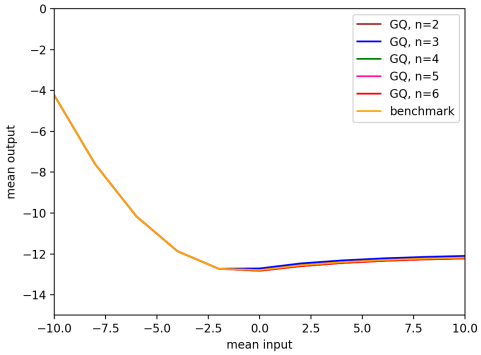
(c) output mean for varying input variance



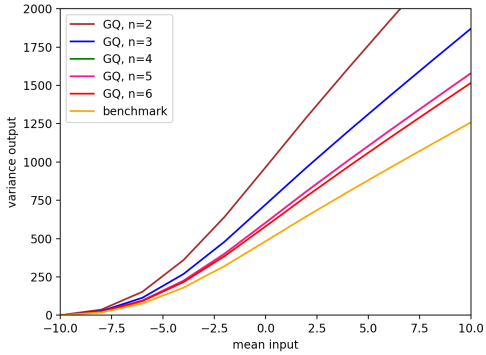
(d) output variance for varying input variance

Figure 4.44: Gaussian quadrature with normally distributed roots as input to Haldane network compared against the benchmark for different input values for mean and variance

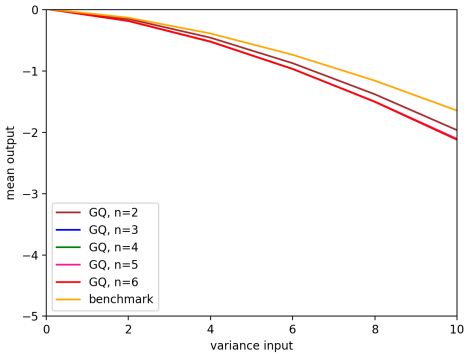
In figure 4.45, the distribution is again changed to a uniform distribution. Now you can see that the results behave to a greater extent as expected. In that, a higher number of quadrature points gives a performance closer to the benchmark compared to a small number of points. It is also possible to assume that if one chooses a high enough number of points, the method will overlap or be very close to overlapping with the benchmark.



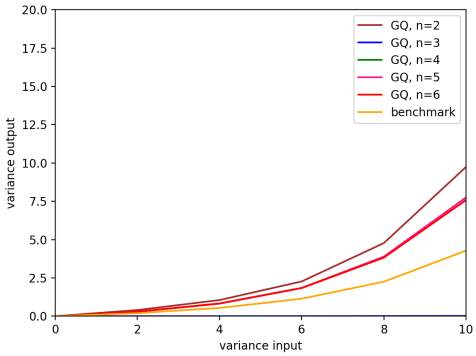
(a) output mean for varying input mean



(b) output variance for varying input mean



(c) output mean for varying input variance



(d) output variance for varying input variance

Figure 4.45: Gaussian quadrature with uniformly distributed roots as input to Haldane network compared against the benchmark for different input values for mean and variance

Chapter 5

Discussion

In this chapter I will discuss the presented results from chapter 4. I will start by looking at each of the methods separately, then a comparison of the methods will be made. Both for the methods alone and in combination with the neural network.

5.1 Linearization

This section is a direct copy of section 5.1 in [26]. Linearization is an easy method to understand and implement. The implementation is mainly based on a function derivation, which has a low computational cost. This makes it appropriate for real-time applications, which the land-based process industry is an example of. However, low computational cost and a simple implementation are usually benefits that go at the expense of high accuracy, and in this case, it is no exception.

For the polynomial of degree 2, the deviation is small and constant for output mean and variance when varying the input mean. If the deviation is constant and known, it is possible to consider it when using the output of the approximation at other places in the overall system. However, the deviation between the approximated and exact output variance increases as the input variance increases. The deviation is no longer constant and increases with a high rate at the variance increase. This approximation is not good enough to be used further in a system that has safety-critical applications.

When the polynomial degree is increased to 3, the performance decreases drastically. The deviation increases for all four cases when the mean or variance takes a higher value. The deviation also increases rapidly, so the approximation will not be sufficient for use.

The linearization method is, therefore, doubtfully useful for uncertainty estimation of neural networks, especially for the land-based process industry, which has high demands for safety.

5.2 Monte Carlo sampling

This section is a direct copy of section 5.2 in [26]. There exist several types of MC sampling methods. In this thesis, a crude MC method was implemented, and the results are shown in chapter 4.

The number of samples used to approximate the output mean, and output variance is essential for the outcome of the approximation. If the number of samples chosen is too small, the variation of each iteration in the algorithm will be significant. This variation will again lead to a bad approximation compared to the exact value. However, it is important to be aware that a large number of samples leads to high computational costs. For a polynomial function of degree 2, it is possible to assume that more samples than 1 million are redundant based on the results. However, 1 million samples use a lot of time to compute the approximation and will not be feasible for real-time application.

Nonlinearities that one encounters in connection with neural networks will probably also be more complex than our chosen polynomial of degree 2. The polynomial function of degree 3 had more significant deviations between approximated and exact values than the degree 2 function. An assumption is that a higher degree on the polynomial needs more samples. Since the computational cost is already too high to work for the intended application, a more complex function and a higher number of samples will worsen the situation.

From the results in chapter 4, it is possible to see that the method is more sensitive to variation in the input variance compared to variation in the input mean. This probably has to do with a higher variation for the variance when samples are chosen randomly.

5.3 Gaussian Quadrature

The Gaussian quadrature method demonstrates that it is possible to obtain an exact approximation if the chosen number of quadrature points n is $n > d$, such that $2n - 1 \geq 2d$, where d is the degree of the polynomial function. When the number of quadrature points required for an exact approximation is known, there is a lower risk in using the method as this parameter can be chosen without assumptions. Nevertheless, it is also seen in section 4.3 that a slightly too low number of points does not necessarily result in a poor approximation compared to the benchmark. The advantage of this method is that an increase in the number of quadrature points does not significantly increase the computational cost. The method is, therefore, both very efficient and reliable as long as the nonlinear problem is known so that the correct number of points is chosen. It is also an advantage that the method is not limited by the number of degrees on the polynomial function, as the processes in the land-based process industry can be quite extensive.

5.4 Comparison of linearization, Monte Carlo sampling, and Gaussian Quadrature

This section is based on section 5.3 in [26]. A comparison of the linearization method and MC sampling method shows that the pros and cons of linearization and MC are quite the opposite. The linearization method has a low computational cost but struggles to give a good approximation of the input mean and input variance for quite simple functions. On the other hand, the MC sampling method is very computationally demanding, but with enough samples, it achieves a good approximation for more complex functions than the linearization method. The MC method depends on the number of samples and if the samples are drawn correctly. This is a critical point for the MC method and can potentially lead to a worse approximation than expected. The linearization method has no such critical points that could lead to unforeseen calculations. However, since there are several possible ways to draw samples in the MC method, it allows the method to be associated with the desired probability distribution, where the linearization is more general.

The Gaussian quadrature, on the other hand, is both computationally efficient and gives an accurate approximation. Suppose one sees the methods in the light of the requirements for safety and other important parameters such as computational costs in connection with the land-based process industry. In that case, it is pretty clear that the Gaussian quadrature appears to be the most optimal method.

5.5 Uncertainty in Neural Network

This section will discuss the three different methods in combination with the neural network.

5.5.1 Linearization

The linearization method, in combination with the Haldane network, has a bad performance compared to the benchmark. This holds for both the uniform and the normal distribution. As shortly mentioned in 3.2, the bad performance might be because of how the samples were chosen as inputs to the Haldane network. These samples are drawn randomly and will therefore be different for each test. A better solution would probably be to use a higher number of samples instead of the overall average samples as input. This approach will reduce the randomness in the input. However, the sampling method alone, with a low number of samples as input, still does much better than the linearization method. The graph has a fairly similar shape to the benchmark even with a low number of samples. This does not apply to linearization, and it is, therefore, possible to assume that it is not only the input to the network, when the nonlinearity in the network is linearized; that is the reason

for the low performance. On the other hand, it can be assumed that linearizing the network in itself is a bad method, especially against the benchmark used in this experiment.

5.5.2 Monte Carlo sampling

The experiment with MC sampling as input to the Haldane network has a good performance when choosing a high enough number of samples. From the plots in section 4.4.1, is it possible to conclude that 100 thousand samples have a performance close to the benchmark and may be a sufficient number of samples for both distributions. On the other hand, one must be a little extra critical of the results of this method, as the benchmark is also based on sampling. This will automatically result in certain similarities between the tests made with fewer samples and the benchmark. The fact that the benchmark is based on 100 million samples may initially also be too low a number of samples to achieve the required accuracy expected from a benchmark. This number was nevertheless chosen as a higher number of samples was too time-consuming to produce.

Although the method appears to perform well, it is essential to remember that sampling has a high computational cost, so it is poorly suited for real-time applications as land-based process industries.

5.5.3 Gaussian quadrature

The results obtained by combining the Gaussian quadrature method and the Haldane network are a bit difficult to interpret because of the contradictions in the results between the two distributions. Initially, it would be most natural to assume that the results based on the normal distribution should have been the opposite. A higher number of quadrature points would have given the best result, with the least deviation compared to the benchmark. The results for the normal distribution should have resembled those for the uniform distribution. This is most natural as the Gaussian quadrature method alone gives a more accurate result for a higher number of quadrature points.

If the above were the case, it looks like choosing a high enough number of quadrature points will result in a reasonably accurate approximation also for Gaussian quadrature in combination with the Haldane network. We know this at least applies to Gaussian quadrature based on uniform distribution as shown from the results in section 4.4.2.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The main topic of this thesis has been to look into methods for uncertainty estimation of neural networks used for the land-based process industry. The first objective was to develop various uncertainty estimation methods for a non-linear function. This is done in chapter 3. Linearization and MC sampling are implemented for uncertainty estimation together with a benchmark corresponding to an exact value. The necessary theoretical framework for these methods is given in chapter 2.

The second objective was to compare the developed methods based on assumptions and performance. The results of the developed methods are presented in chapter 4. For the three methods, linearization, MC sampling, and Gaussian quadrature, relevant cases are studied to identify the advantages and disadvantages of the methods. These cases involve polynomial functions with different complexity. It also involves varying the input mean and input variance separately to see how that affects the output mean and output variance. For the MC method, it is also investigated how the randomness in choosing samples influences the approximations.

The third objective in this thesis was to combine the methods developed in the second objective with a static feed-forward neural network. This was done for all three methods. Using the methods as input to the neural network is the same as performing an approximation of the method. This makes it possible to identify new advantages and disadvantages, as well as perform uncertainty estimation on an actual neural network.

The main conclusion from the discussion of the three methods on its own is that neither the implemented MC method nor the implemented linearization is performing sufficiently well enough. For both the methods, the approximation is not accurate enough or the method has too high a computational cost to be reliable and useful for the intended application, which is a safety-critical application, namely the land-based

process industry. However, the Gaussian quadrature has an exact approximation of the non-linear function as long as the correct number of quadrature points is chosen. This method also has a low computational cost compared to the MC method. The Gaussian quadrature method, therefore, stands out as a method that can work in practice. Using this method as input to the Haldane network gives an ambiguous result that makes it difficult to assess whether this is a method to invest in further. Suppose the results from Gaussian quadrature with normally distributed roots are ignored and we choose to rely on the results from Gaussian quadrature with uniformly distributed roots. In that case, we conclude that this method can work well. This method clearly has fewer disadvantages than the other two, so assuming the Gaussian quadrature method with uniformly distributed roots is correct, it may be desirable to look at this method further.

However, the MC method also performs well as input to the Haldane network compared to the benchmark. Nevertheless, this method still has the drawback of a high computational cost, and it is possible to question whether the method does as well as it looks, as the benchmark is also based on this method.

The fourth objective is to identify the areas for further research with uncertainty estimation for neural networks. This is done in the upcoming section.

6.2 Future work

There are several opportunities to continue on the results obtained in this thesis. A place to start is to do new experiments with linearization, MC sampling, and Gaussian quadrature on a different and more complex neural network. This can either be a trained network as in this thesis, based on a more complex model, or it can be a neural network built and trained from scratch.

It is also possible to introduce entirely new methods. Another interesting method to investigate is the Assumed Density Filtering (ADF) method. This method is not an entire-network uncertainty propagation method, but a layer-to-layer method for uncertainty propagation. This method is based on applying ADF to the nodes in the DNN. This will give the overall network a tractable approximation, incorporating one neuron at a time. Considering one layer at a time provides a recipe that converts any layer with neurons into a layer that propagates uncertainty. This method has shown reasonable uncertainty estimates, with a high correlation to the empirical error for regression and classification tasks.

When one or several methods for uncertainty estimation has obtained a good enough performance for static feed-forward neural networks, the next step is to identify challenges applying these methods to dynamical state-space models based on neural networks. When these challenges are identified, new methods for overcoming these have to be implemented, such that it is possible to use the methods for uncertainty estimation in neural networks for dynamical state-space models. This can, for

example, be based on the Moore-Greitzer model [20] or the Continuous stirred tank reactor model [25]. When this is done, the developed methods for uncertainty estimation in dynamical state-space models should be compared. The goal is to find practical methods for uncertainty estimation for dynamical state-space models based on neural networks. This is an essential step before a data-driven model based on DL can be used with traditional control theory in application areas with high safety demands.

References

[1] *Activation Functions — ML Glossary documentation*. URL: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html#relu (visited on 08/20/2022).

[2] *Aleatoric Uncertainty - an overview | ScienceDirect Topics*. URL: https://www.sciencedirect.com/topics/engineering/aleatoric-uncertainty (visited on 11/15/2021).

[3] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI". en. In: *arXiv:1910.10045 [cs]* (Dec. 2019). arXiv: 1910.10045. URL: http://arxiv.org/abs/1910.10045 (visited on 10/19/2021).

[4] B. YEGNANARAYANA. *ARTIFICIAL NEURAL NETWORKS - B. YEGNANARAYANA - Google Bøker*. Twelfth Printing. Sept. 2006. ISBN: 81-203-1253-8. URL: https://books.google.no/books?hl=no&lr=&id=RTtvUVU_xL4C&oi=fnd&pg=PR9&dq=Artificial+neural+networks&ots=Gd93AjBKSw&sig=P3r2bGh8TuAV4yUqGjlQ2CpHusQ&redir_esc=y#v=onepage&q&f=false (visited on 11/21/2021).

[5] George Cvetkovich and Timothy C. Earle. "Classifying hazardous events". en. In: *Journal of Environmental Psychology* 5.1 (Mar. 1985), pp. 5–35. ISSN: 02724944. DOI: 10.1016/S0272-4944(85)80036-0. URL: https://linkinghub.elsevier.com/retrieve/pii/S0272494485800360 (visited on 11/15/2021).

[6] D. Dochain and Peter A. Vanrolleghem. "Dynamical Modelling & Estimation in Wastewater Treatment Processes". In: 2001, p. 30.

[7] David B. Thomas and Wayne Luk. "Estimation of Sample Mean and Variance for Monte-Carlo Simulations". In: *Imperial College London* (). URL: https://cas.ee.ic.ac.uk/people/dt10/research/thomas-08-sample-mean-and-variance.pdf (visited on 08/07/2022).

[8] *Gaussian quadrature*. URL: https://pluto.huji.ac.il/~mszucker/BINARY/gaussian_quadrature.pdf.

[9] Benyamin Ghojogh et al. "Sampling Algorithms, from Survey Sampling to Monte Carlo Methods: Tutorial and Literature Review". en. In: *arXiv:2011.00901 [physics, stat]* (Nov. 2020). arXiv: 2011.00901. URL: http://arxiv.org/abs/2011.00901 (visited on 12/19/2021).

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[11] Jens G.Balchen and Trond Andresen. *Reguleringsteknikk*. 6th ed. Trondheim: Institutt for teknisk kybernetikk, 2016. ISBN: 978-82-7842-202-1.

[12] Ji Weiqi, Zhuyin Ren, and Chung K. Law. *Uncertainty Propagation in Deep Neural Network Using Active Subspace*. URL: https://arxiv.org/pdf/1903.03989.pdf.

[13] Alex Kendall and Yarin Gal. "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" en. In: *arXiv:1703.04977 [cs]* (Oct. 2017). arXiv: 1703.04977. URL: http://arxiv.org/abs/1703.04977 (visited on 11/15/2021).

[14] *law of large numbers | statistics | Britannica*. URL: https://www.britannica.com/science/law-of-large-numbers (visited on 08/07/2022).

[15] Antonio Loquercio, Mattia Segù, and Davide Scaramuzza. "A General Framework for Uncertainty Estimation in Deep Learning". en. In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020). arXiv: 1907.06890, pp. 3153–3160. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2020.2974682. URL: http://arxiv.org/abs/1907.06890 (visited on 10/19/2021).

[16] Mark Haring. *Network approximation of a nonlinear function*. (Visited on 06/21/2022).

[17] Marvin Minsky and Saymour Papert. *Perceptrons - An introduction to Computational Geometry*. Massachusetts institute of technology, 1998. ISBN: 0-262-63111-3.

[18] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[19] Maria Moreno de Castro. *Uncertainty Quantification and Explainable Artificial Intelligence*. other. pico, Mar. 2020. DOI: 10.5194/egusphere-egu2020-21281. URL: https://meetingorganizer.copernicus.org/EGU2020/EGU2020-21281.html (visited on 12/19/2021).

[20] Åse Neverlien, Signe Moe, and Jan T. Gravdahl. "Compressor Surge Control Using Lyapunov Neural Networks". en. In: *Modeling, Identification and Control: A Norwegian Research Bulletin* 41.2 (2020), pp. 41–49. ISSN: 0332-7353, 1890-1328. DOI: 10.4173/mic.2020.2.1. URL: http://www.mic-journal.no/ABS/MIC-2020-2-1.asp (visited on 08/21/2022).

[21] *Normalfordeling*. URL: https://tma4245.math.ntnu.no/viktige-kontinuerlige-fordelinger/normalfordeling/ (visited on 12/18/2021).

[22] *Python Release Python 3.9.7 | Python.org*. URL: https://www.python.org/downloads/release/python-397/ (visited on 08/19/2022).

[23] *Quadrature – from Wolfram MathWorld*. URL: https://mathworld.wolfram.com/Quadrature.html (visited on 08/21/2022).

[24] D Sculley et al. "ON PACE, PROGRESS, AND EMPIRICAL RIGOR". en. In: (2018), p. 4.

[25] Katrine Seel et al. "Neural Network-Based Model Predictive Control with Input-to-State Stability". en. In: *2021 American Control Conference (ACC)*. New Orleans, LA, USA: IEEE, May 2021, pp. 3556–3563. ISBN: 978-1-66544-197-1. DOI: 10.23919/ACC50511.2021.9483190. URL: https://ieeexplore.ieee.org/document/9483190/ (visited on 08/21/2022).

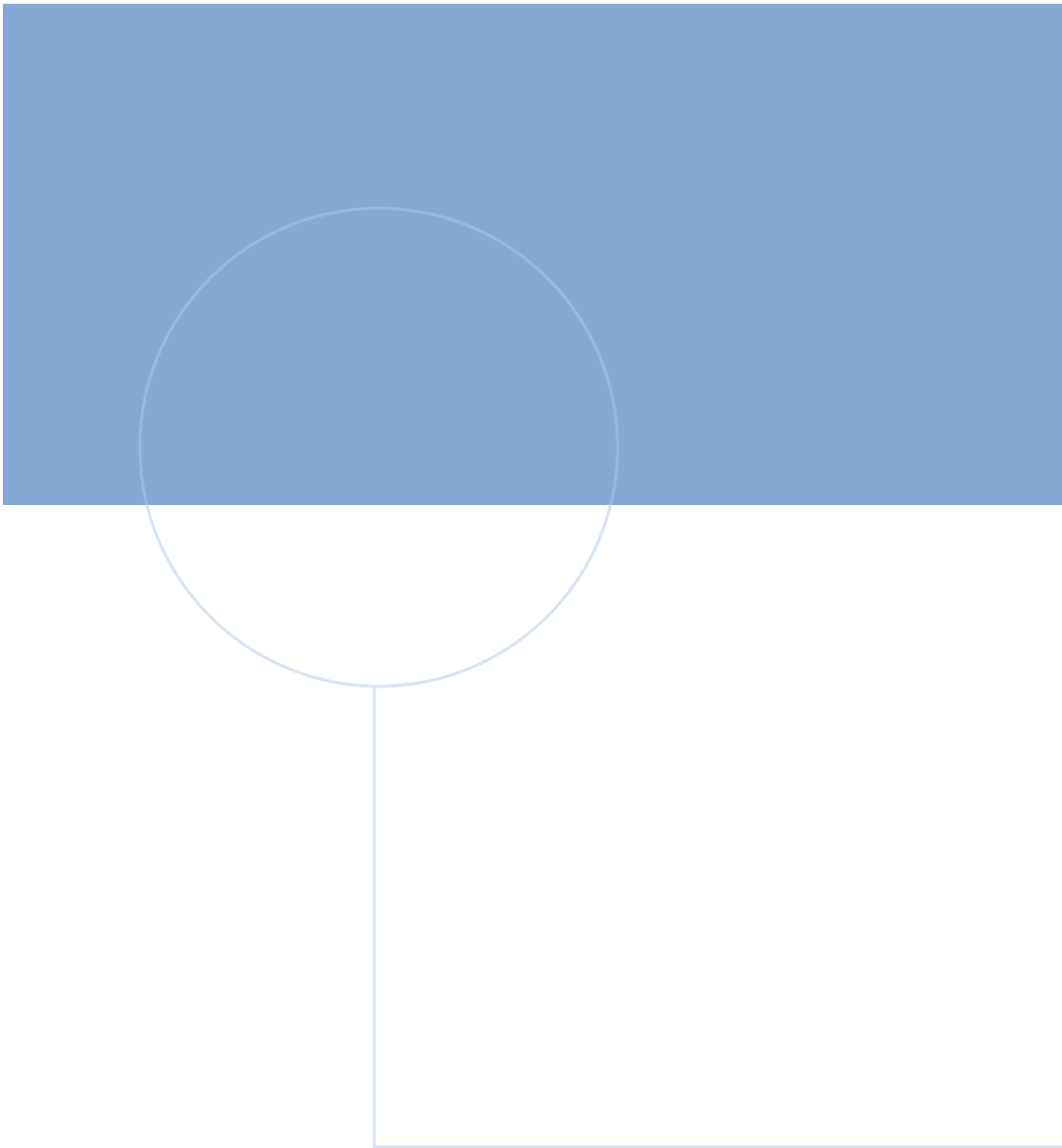
[26] Siri Holde Hegvold. "Uncertainty estimation in Neural Networks". In: (2021).

[27] *TAPI (Towards Autonomy in Process Industries) - SINTEF*. URL: https://www.sintef.no/en/projects/2019/tapi-towards-autonomy-in-process-industries/ (visited on 10/19/2021).

[28] Jessica S. Titensky, Hayden Jananthan, and Jeremy Kepner. "Uncertainty Propagation in Deep Neural Networks Using Extended Kalman Filtering". en. In: *arXiv:1809.06009 [cs, math, stat]* (Sept. 2018). arXiv: 1809.06009. URL: http://arxiv.org/abs/1809.06009 (visited on 12/19/2021).

[29] *Uniformfordeling*. URL: https://tma4245.math.ntnu.no/viktige-kontinuerlige-fordelinger/uniformfordeling/ (visited on 12/18/2021).

[30] *USA TODAY*. URL: https://eu.usatoday.com/story/tech/2015/07/01/google-apologizes-after-photos-identify-black-people-as-gorillas/29567465/ (visited on 11/15/2021).



NTNU

Norwegian University of
Science and Technology