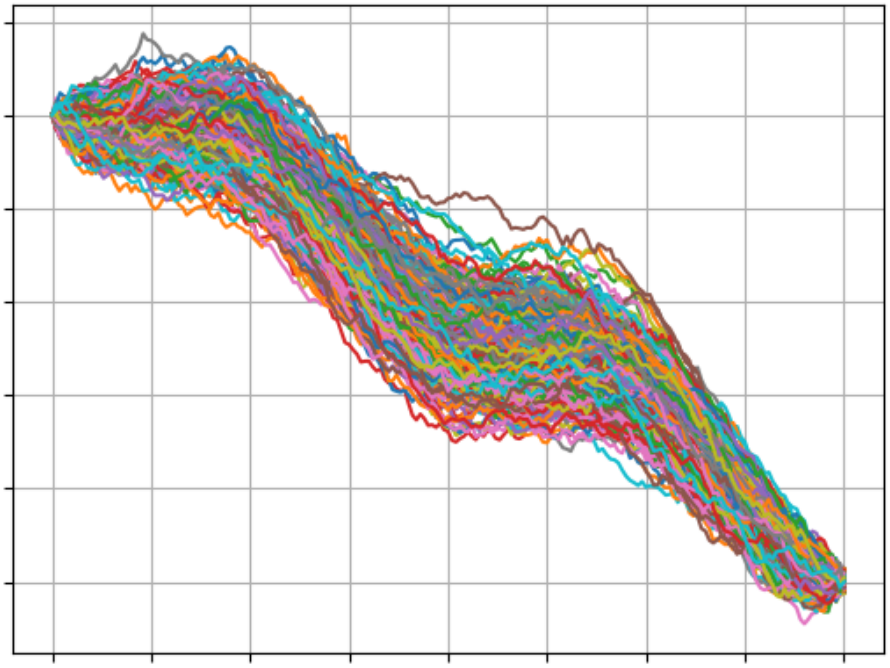


Prospects of neural SDEs for Bayesian smoothing

Investigating the possibility of using neural networks to parameterize generative models to approximate conditional probability distributions

Master's thesis in Engineering mathematics and computational science

Samuel Winqvist and Isak Wikman

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Prospects of neural SDEs for Bayesian smoothing

Investigating the possibility of using neural networks to parameterize generative models to approximate conditional probability distributions

Samuel Winqvist and Isak Wikman

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Saab AB
Chalmers University of Technology
Gothenburg, Sweden 2024

Prospects of neural SDEs for Bayesian smoothing
Investigating the possibility of using neural networks to parameterize generative models to approximate conditional probability distributions
Samuel Winqvist and Isak Wikman

Supervisors: Benjamin Svedung Wettervik and Adam Andersson, Saab AB
Examiner: Moritz Schauer, Department of Mathematical Sciences

Master's Thesis 2024
Department of Mathematical Sciences
Engineering mathematics and computational sciences
Saab AB
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone: Samuel +46 73 060 90 91, Isak +46 70 603 61 27

Cover: Paths simulated from a nonlinear SDE.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Prospects of neural SDEs for Bayesian smoothing
Investigating the possibility of using neural networks to parameterize generative models to approximate conditional probability distributions
Samuel Winqvist and Isak Wikman
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Due to the importance of airspace defence for national security, the pursuit of efficient and accurate methods for tracking airborne targets is of great interest. An essential part of target tracking is to estimate the state of a target given (noisy) observations from a radar. This involves calculating conditional probability distributions given measurements (e.g. smoothing), which is associated with large computational costs for high-dimensional and/or nonlinear target models. This thesis investigates the possibility of alleviating such bottlenecks by employing neural stochastic differential equations (neural SDEs) as generative models for the smoothing problem - possibly allowing the use of more complex models in real-time applications. As part of this work, it is first shown that SDE models can be used as generative models for conditional distributions of target states given noisy measurements obtained at discrete points of time. Secondly, this thesis presents a detailed description of how the neural SDE is utilized as the generator of a so called Wasserstein Generative Adversarial Network (WGAN) in signature space, meaning it can be trained to approximate conditional probability distributions for the smoothing problem. Thirdly, the SDE-model is evaluated with respect to a series of test-problems. Findings indicate that the neural SDE can reproduce qualitative features for conditional distributions for nonlinear problems, while further developments of the method and paradigms for training are needed to achieve trained SDE-models which are perceived to (in distribution) closely resemble their theoretically exact counterparts.

Keywords: smoothing, filtering, target identification, nonlinear, neural networks, generative AI, neural SDE, particle filter, signature, Wasserstein, GAN, generative adversarial network.

Acknowledgements

We would like to thank our supervisors Adam Andersson and Benjamin Svedung Wettervik at Saab AB. No matter the problem encountered, their knowledge has guided us through this project. For this, we thank you. We would also like to thank our examiner at Chalmers, Moritz Schauer, for providing valuable insights and providing another perspective of the project. Finally, we would like to direct a thank you to all people outside of this project for their support and encouragement.

Samuel Winqvist and Isak Wikman, Gothenburg, May 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

GAN	Generative Adversarial Network
MLP	Multilayer Perceptron
MTT	Multi-Target-Tracking
NN	Neural Network
ODE	Ordinary Differential Equation
ReLU	Rectified Linear Unit
SDE	Stochastic Differential Equation
SGD	Stochastic Gradient Descent
WGAN	Wasserstein Generative Adversarial Network

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

Military air defence systems play an essential role in keeping people safe. A vital part of the air defence is to have a good air situational awareness, which relies on radar sensor systems. These systems emit electromagnetic pulses that reflect off targets, where a small fraction of the reflected radiation is received and recorded by the radar. This data allow for locating targets, estimating where they are heading, identifying what threats they pose and distinguishing between friend and foe. The air situational picture has in recent years grown increasingly complex, featuring a diverse array of targets such as missiles, UAVs, drones, birds and various ground-based targets.

Multi-Target-Tracking (MTT) is the radar function that associates observations of targets over time, which gives rise to tracks. These tracks are crucial to form an accurate air situational picture. Since military does not allow for engagements against unidentified objects, early tracking, track continuity, classification and identification are key capabilities needed for fast and qualified decision making. The previously mentioned increased air situational complexity has introduced challenges, particularly in terms of increased computational costs which leads to approximations that may compromise with performance.

One approximation in MTT, and especially in the problem of estimating the state of a target, is the use of simplified state estimation models combined with models for uncertainty. While these models enable real-time operation, they also lead to non-optimal performance. One source of uncertainty comes from the simpler models themselves, since they may not accurately describe the true underlying dynamics. Incorporating noise with the state estimation models makes tracking possible by compensating for the model mismatch. These models have a low degree of predictive ability if targets follow nonlinear dynamics, which is often the case for real-life targets.

An optimal model for state estimation is a model with high degree of predictive ability, i.e., with the smallest possible amount of noise. Some form of uncertainty will however always be needed to model uncertainties such as measurement noise and lack of knowledge of steering signals. An accurate model comes at the price that different types of targets need different state estimation models and the type

of target may not a-priori be known. This has become an even larger problem due to the increased complexity of the air situational picture. Additionally, contemporary accurate models, even when assuming the true underlying dynamics are known, come with a large computational cost meaning they are not feasible for real-time applications.

In summary, different state estimation models are needed for different types of targets to increase their predictive ability, while they also must balance efficiency and accuracy. Achieving a good trade-off is challenging, prompting the investigation of using neural stochastic differential equations (neural SDEs) to solve the state estimation problem. SDEs incorporate uncertainties with a stochastic component and are almost instantaneous to sample from, while neural networks have shown great success in handling a great amount of data and can approximate almost any function, even nonlinear ones. Therefore, it could be both an efficient and accurate choice of model. The neural SDE could also simultaneously solve the target identification problem by being trained against a wide range of different targets, meaning that the neural SDE could be directly applied to any type of target to solve the state estimation problem.

For state estimation, there are three cases that could be considered: smoothing, filtering, and prediction. Smoothing is the problem of estimating the state of a target at previous time instances using all available information, filtering estimates the current state and prediction estimates the future state. In military usage, filtering and prediction are of particular interest. However, if using a neural SDE, these problems are solved in a similar manner, requiring only minor adjustments to transition from one task to another.

1.2 Problem description

Considering the given background, exploring the possibility of using neural SDEs in this area has many aspects to consider. Due to this being a new area, we focus on exploring the smoothing problem (and filtering in its special cases). First, assume target data is generated by

$$\begin{aligned} \mathrm{d}X_t &= \mu(t, X_t)\,\mathrm{d}t + \sigma(t, X_t)\,\mathrm{d}W_t, \quad t \in [0, T],\ X_0 = x_0 \\ Z_n &= X_{t_n} + V_n, \quad n = 1, ..., N. \end{aligned} \tag{1.1}$$

Here X_t is the hidden true state at time t and realizations of it, i.e., the sample path is denoted as x_t. Notice that the initial value, X_0, is assumed to be known and is equal to the constant x_0. In this thesis, for the observations we use the notation Z_n for the random variable and z_n as outcomes of Z_n. The random variable $V_n \sim \mathcal{N}(0, \sigma_z^2)$ corresponds to the measurement noise. Meanwhile, μ and σ are referred to as the drift and diffusion coefficients, respectively. As previously mentioned, we will solve the smoothing problem which is equivalent to estimating the distribution of X at previous time instances given observations $z_{1:N}$, and more specifically,

$$\pi((x_t)_{t\in[0,t_N]}|z_{1:N}). \tag{1.2}$$

Here π stands for the unknown distribution that is to be replicated. This estimation will be performed by simulating data from

$$\begin{aligned} \mathrm{d}\tilde{X}_t &= \tilde{\mu}_{\gamma_i}(t, \tilde{X}_t, z_{i:N})\,\mathrm{d}t + \tilde{\sigma}_{\gamma_i}(t, \tilde{X}_t, z_{i:N})\,\mathrm{d}W_t, \quad t \in (t_{i-1}, t_i],\ i = 1, 2, ..., N \\ \tilde{X}_0 &= x_0. \end{aligned} \tag{1.3}$$

Here $\tilde{\mu}_{\gamma_i}$ and $\tilde{\sigma}_{\gamma_i}$ are neural networks with parameters γ_i and we refer to (1.3) as a neural SDE. This is a Markov process and only the observations in the future have an impact for simulating the next value. This is shown to be true in Section 2.3. In that section we also analyse the dynamics of the conditional model. This is important in order to understand needed structure for the neural network. By training the neural networks, the drift and diffusion networks can hopefully represent (possibly nonlinear) conditional drift and diffusion coefficients that are difficult to derive analytically.

Notice that X_t and Z_n in our problem formulation are one-dimensional which is often not enough for applications. This simplification makes the analysis in the thesis simpler to interpret. Further discussion about this, and other extensions to make the neural SDE useful in real-life applications, can be seen in Section 6.3.

1.3 Similar work

This area is relatively new, and with our approach, to use the so called Signature Wasserstein Loss for training, there are relatively few works to take inspiration from. One text that uses this metric for conditional time series generation is [1]. However, their focus is on applications in finance, which means they are more interested in prediction rather than smoothing and filtering. More specifically, their approach is to use a Wasserstein Generative Adversarial Networks (GAN) defined in the signature space, which is exactly the neural network structure used in this thesis, to do stock predictions. The article provides a plausible introduction to signature theory and the Wasserstein distance in the signature space.

The text that sparked the idea of applying neural SDEs to this problem is [2]. The author, Kidger, introduces several kinds of neural differential equations and discusses them. One of the introduced neural differential equations is the neural SDE. Kidger describes the neural SDE as a generator for GAN networks, which is closely related to the concepts in [1]. Combining the concepts of using a neural SDE as a generator together with a Wasserstein GAN in signature space, lead to the core of this thesis. Thereby, much of the methodology is inspired by these articles.

To analyse the true conditional SDE theoretically, inspiration has been taken from [3]. It outlines the conditional SDE in several dimensions, in contrast to this thesis where it is done in one dimension. We use this concept to analyse the true conditional SDE in a specific setting (in the case of Brownian motion).

1.4 Contents of the report

In Chapter 2, necessary theory regarding stochastic differential equations is presented. Strong solutions to SDEs and the Markov property are defined, before progressing towards conditional SDEs. To conclude the chapter, the conditional SDE in the special case of a Brownian motion is derived explicitly.

Chapter 3 introduces the reader to signature theory, providing a necessary background before defining and interpreting the signature transformation. Furthermore, expected signatures and the Wasserstein distance in the signature space are discussed. Finally, interpolation methods and process augmentations of practical use connected to the signature are considered.

In Chapter 4, the concept of neural networks and how these are utilized are introduced. Furthermore, the network structure of Generative Adversarial Networks is discussed. The section is concluded by connecting the concepts of signature and Generative Adversarial Networks.

In Chapter 5 the main procedure for developing the neural SDE is discussed. It begins with a summary of the problem formulation, given the concepts presented in previous chapters. Furthermore, different approaches for implementing the neural SDE is discussed, both successful and unsuccessful ones. The chapter is then concluded by choosing the final implementation of the neural SDE.

In Chapter 6, the performance of the neural SDE is compared to that of a particle filter. This is done for two cases of nonlinear SDEs. Furthermore, the results are examined and proposals for future development are discussed. The chapter is concluded with a conclusion regarding the outcome of the thesis.

2
Stochastic Calculus

The focus of this chapter is to present elements of stochastic analysis. We start this chapter by, in Section 2.1, introducing stochastic differential equations. Stochastic differential equations in this text are assumed to have strong solutions, which is introduced in Section 2.1.1, and are as a consequence Markov processes. Therefore we, in Section 2.1.3, introduce the Markov property. This is important since it is a necessity when generating samples from stochastic differential equations, which is introduced in Section 2.2. Furthermore, SDEs conditioned on observations are, in Section 2.3, considered, and the special case of when the underlying SDE is a Brownian motion is derived explicitly in Section 2.3.3. This special case is crucial in the development of the neural SDE.

2.1 Stochastic differential equations

Consider the filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\geq 0}, \mathbb{P})$ equipped with a Brownian motion. A **stochastic differential equation** (an **SDE**) is a differential equation on the form

$$\mathrm{d}X_t = \mu(X_t, t)\,\mathrm{d}t + \sigma(X_t, t)\,\mathrm{d}W_t, \quad t \in (0, T], \quad X_0 \sim P. \tag{2.1}$$

Here $\mu : \mathbb{R}^2 \to \mathbb{R}$ is referred to as the drift coefficient and $\sigma : \mathbb{R}^2 \to \mathbb{R}$ as the diffusion coefficient. The initial value X_0 is taken from some distribution, P, however, in our thesis it is assumed to be known and more specifically a constant. Meanwhile, $W : \Omega \times [0, T] \to \mathbb{R}$ is a Brownian motion. The initial value X_0 is $\mathcal{F}_0$-measurable, the processes $\mu(X_t, t)$ and $\sigma(X_t, t)$ are $\mathcal{F}_t$-adapted, as well as $\int_0^T |\mu(X_s, s)|\,\mathrm{d}s < \infty$ and $\int_0^T \sigma^2(X_s, s)\,\mathrm{d}s < \infty$.

A solution to (2.1) is an adapted stochastic process satisfying $\sup_{t\in[0,T]} \mathbb{E}[|X_t|^2] < \infty$ and for all $t \in [0, T]$ satisfy $\mathbb{P}$-almost surely

$$X_t = X_0 + \int_0^t \mu(X_s, s)\,\mathrm{d}s + \int_0^t \sigma(X_s, s)\,\mathrm{d}W_s. \tag{2.2}$$

The second integral is the Itô integral.

2.1.1 Strong solutions

The Itô integral is defined for so called predictable square-integrable processes. This means that solutions to stochastic differential equations, when they exist, are processes of this kind. One setting which guarantees the existence and uniqueness of

strong solutions is when the following conditions are fulfilled:

- The SDE coefficients are globally Lipschitz in x with uniform constants, i.e., there is a constant K, such that for all $x, y \in \mathbb{R}$ and all $0 \leq t \leq T$ we have
$$|\mu(x,t) - \mu(y,t)| + |\sigma(x,t) - \sigma(y,t)| < K|x - y|. \tag{2.3}$$
- The SDE coefficients satisfy the linear growth condition in x with uniform constant, i.e., there is a constant K, such that for all $x \in \mathbb{R}$ and all $0 \leq t \leq T$ we have
$$|\mu(x,t)| + |\sigma(x,t)| \leq K(1 + |x|). \tag{2.4}$$
- The initial value X_0 is independent of W and $\mathbb{E}[X_0^2] < \infty$.

If these conditions are fulfilled, it is possible to show that X has continuous paths and the expected value of the squared process is bounded by the expected value of the squared initial value, more specifically

$$\mathbb{E}\Big[\sup_{0 \leq t \leq T} X_t^2\Big] < C(1 + \mathbb{E}[X_0^2]). \tag{2.5}$$

The proof of existence is similar for that of ODEs, i.e., by using Picard iterations, while uniqueness follows by Grönwall's lemma. In this report we assume strong solutions, but not necessarily that these conditions are fulfilled. Namely, there exists other settings for existence and uniqueness of strong solutions, but that is not the focus of this report. In [4] some other such settings are presented.

2.1.2 Itô theory

The existence of strong solutions does not give a methodology on finding them analytically. A useful tool for finding strong solutions is to consider Itô's lemma, which is presented below.

Lemma 1 (Itô's formula for $f(X_t, Y_t)$) *Let $f\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ have bounded and continuous derivatives up to order two and X, Y be Itô processes*

$$\begin{aligned} \mathrm{d}X_t &= \mu_X(X_t, t)\,\mathrm{d}t + \sigma_X(X_t, t)\,\mathrm{d}W_t \\ \mathrm{d}Y_t &= \mu_Y(Y_t, t)\,\mathrm{d}t + \sigma_Y(Y_t, t)\,\mathrm{d}W_t. \end{aligned} \tag{2.6}$$

Here $\mu_X, \mu_Y, \sigma_X, \sigma_Y\colon \mathbb{R} \times [0,T] \mapsto \mathbb{R}$ with suitable regularity properties, then

$$\begin{aligned} f(X_t, Y_t) = f(X_0, Y_0) &+ \int_0^t \frac{\partial f}{\partial x}(X_s, Y_s)\,\mathrm{d}X_s \\ &+ \int_0^t \frac{\partial f}{\partial y}(X_s, Y_s)\,\mathrm{d}Y_s \\ &+ \int_0^t \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(X_s, Y_s)\sigma_X^2(X_s, s)\,\mathrm{d}s \\ &+ \int_0^t \frac{1}{2}\frac{\partial^2 f}{\partial y^2}(X_s, Y_s)\sigma_Y^2(Y_s, s)\,\mathrm{d}s \\ &+ \int_0^t \frac{\partial^2 f}{\partial x \partial y}(X_s, Y_s)\sigma_X(X_s, s)\sigma_Y(Y_s, s)\,\mathrm{d}s \end{aligned} \tag{2.7}$$

When $Y_t = t$, it reads

$$\begin{aligned} f(X_t, t) =& f(X_0, 0) + \int_0^t \frac{\partial f}{\partial x}(X_s, s)\,\mathrm{d}X_s + \int_0^t \frac{\partial f}{\partial t}(X_s, s)\,\mathrm{d}s \\ &+ \int_0^t \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(X_s, s)\sigma_X^2(X_s, s)\,\mathrm{d}s. \end{aligned} \tag{2.8}$$

Notice, that the equations in Itô's formula are stated in the case when X is one-dimensional. This makes the derivation found in Section 2.3 easier to follow. However, Itô's formula in higher dimensions is not more cumbersome to derive; it looks as expected, i.e., sums of partial derivatives.

2.1.3 Markov property

Strong solutions to SDEs have the so called **Markov property**. The Markov property for a for a process X is formally defined as

$$\mathbb{P}[X_t \in A|\mathcal{F}_s] = \mathbb{P}[X_t \in A|\sigma(X_s)] \tag{2.9}$$

for $0 \leq s \leq t$ and $A \in \mathcal{B}$ (an element of the Borel set). Here $\mathcal{F}_s$ is the filtration at time s and $\sigma(X_s)$ stands for sigma-algebra generated by the random variable X_s. The meaning of (2.9) is that conditioning on all information up to s is the same as conditioning on the information at s. Written in a more informal way it means that

$$\mathbb{P}[X_t \in A|X_t, t \in [0, s]] = \mathbb{P}[X_t \in A|X_s]. \tag{2.10}$$

A sufficiently regular diffusion process which has the Markov property also has what is called a **transition density**. The transition density is a function denoted as $p(y, t + s, x, t)$ and it corresponds to the probability density for the process going from the value x at time t to the value y at time $t + s$. We also denote it as $p(y, t + s|x, t)$ to make it clear which state the process is going from and to.

We next introduce the **generator** of a Markov process. The generator, applied on a sufficiently regular function $\phi(x)$, is defined as

$$A\phi(x) = \lim_{\tau \to 0} \frac{\mathbb{E}[\phi(X_{t+\tau})|X_t = x] - \phi(x)}{\tau}. \tag{2.11}$$

For a strong solution X_t (with observed value x_t) to an SDE with coefficients μ and σ, the generator is given by

$$A\phi(x_t) = \phi'_x(x_t)\mu(x_t, t) + \frac{1}{2}\phi''_{xx}(x_t)\sigma^2(x_t, t). \tag{2.12}$$

The proof is straightforward, and it relies on Itô's formula. More precisely, for $t \geq 0$ and $\tau > 0$ we have

$$\begin{aligned} &\phi(X_{t+\tau}) \\ &\quad = \phi(x_t) + \int_t^{t+\tau} \phi'_x(X_s)\mu(X_s, s)\,\mathrm{d}s + \int_t^{t+\tau} \phi'_x(X_s)\sigma(X_s, s)\,\mathrm{d}W_s \\ &\qquad + \frac{1}{2}\int_t^{t+\tau} \phi''_{xx}(X_s)\sigma^2(X_s, s)\,\mathrm{d}s. \end{aligned} \tag{2.13}$$

Taking the expected value conditioned on $X_t = x_t$ yields

$$\mathbb{E}[\phi(X_{t+\tau})|X_t = x_t] = \phi(x_t) + \int_t^{t+\tau} \phi'_x(X_s)\mu(X_s, s)\,\mathrm{d}s + \frac{1}{2}\int_t^{t+\tau} \phi''_{xx}(X_s)\sigma^2(X_s, s)\,\mathrm{d}s. \tag{2.14}$$

This follows since the expected value of the term $\int_t^{t+\tau} \phi'_x(X_s)\sigma(X_s, s)\,\mathrm{d}W_s$ is zero. When dividing by τ and letting it go to zero the result follows by the mean value theorem.

It is also possible to consider the generator $\tilde{A}$ of the stochastic process X_t. This is defined through its action on a sufficiently smooth function $\phi(x, t)$ as followed

$$\tilde{A}\phi(x, t) = \lim_{\tau \to 0} \frac{\mathbb{E}[\phi(X_{t+\tau}, t + \tau)|X_t = x] - \phi(x, t)}{\tau}. \tag{2.15}$$

It can be shown, by using Itô's formula and the same reasoning,

$$\tilde{A}\phi(x_t, t) = \phi'_t(x_t, t) + \phi'_x(x_t, t)\mu(x_t, t) + \frac{1}{2}\phi''_{xx}(x_t, t)\sigma^2(x_t, t). \tag{2.16}$$

The only difference in the result for respective operator is the term $\phi'_t(x_t, t)$. This means that we have found a way of identifying an SDE by considering its generator.

2.2 Numerical schemes for solving SDEs

There are several approaches when it comes to solving SDEs, or rather approximate solutions to SDEs. A popular approach for solving SDEs, which is used in this report, is the Euler—Maruyama method. Consider a partition of the interval $[0, T]$, i.e.,

$$0 = t_0 < t_1 ... < t_n = T, \tag{2.17}$$

and note that the process X satisfies

$$X_{t_{i+1}} = X_{t_i} + \int_{t_i}^{t_{i+1}} \mu(X_s, s)\,\mathrm{d}s + \int_{t_i}^{t_{i+1}} \sigma(X_s, s)\,\mathrm{d}W_s, \quad i = 0, ..., n - 1. \tag{2.18}$$

With the Euler–Maruyama method, X_{t_i} in (2.18) is approximated by Y_i with the left rectangle method according to

$$Y_{i+1} = Y_i + \mu(Y_i, t_i)\Delta t + \sigma(Y_i, t_i)\Delta W_i. \tag{2.19}$$

Here $\Delta W_i = W_{t_{i+1}} - W_{t_i} = \mathcal{N}(0, t_{i+1} - t_i)$.

If convergence is too slow, the Milstein method could instead be considered, which is defined similarly

$$Y_{i+1} = Y_i + \mu(Y_i, t_i)\Delta t + b(Y_i, t_i)\Delta W_i + \frac{1}{2}\sigma(Y_i, t_i)\sigma'(Y_i, t_i)((\Delta W_i)^2 - (t_{i+1} - t_i)). \tag{2.20}$$

2.3 Conditional SDE-models

Consider the SDE

$$\mathrm{d}X_t = \mu(X_t, t)\,\mathrm{d}t + \sigma(X_t, t)\,\mathrm{d}W_t, \quad t \in [0, T],\ X_0 = x_0 \tag{2.21}$$

and the observations

$$Z_n = X_{t_n} + V_n, \quad n = 1, ..., N. \tag{2.22}$$

Here $V_n \sim \mathcal{N}(0, \sigma_Z^2)$. In this thesis we are interested in a generative model, such that its sampled paths are distributed according to the conditional distribution $\pi((x_t)_{t\in[0,t_N]}|z_{1:N})$. More specifically, since the project investigates neural SDEs, we want to answer whether the conditional generative model is an SDE. To prove that this is the case, we first introduce the Doob-h transform in a rather general setting. Using the Doob-h transform, the conditional SDE when conditioning on observations can be derived. Finally, the conditional SDE in the case when the underlying SDE is a Brownian motion is derived explicitly.

2.3.1 Doob-h transform

We present the doob-h transform in the following theorem, where we use the concept of the generator of a process defined in Section 2.1.3.

Theorem 2 (Doob-h transform) *Let X be defined by the following SDE*

$$\mathrm{d}X_t = \mu(X_t, t)\,\mathrm{d}t + \sigma(X_t, t)\,\mathrm{d}W_t, \qquad X_0 = x_0 \tag{2.23}$$

with transition density $p(y, t+s|x, t)$. Then consider the transition density

$$p^h(y, t+s|x, t) = p(y, t+s|x, t)\frac{h(t+s, y)}{h(t, x)}, \tag{2.24}$$

where $h(t, x)$ is any function such that

$$h(t, x) = \int p(y, t+s|x, t)h(t+s, y)\,\mathrm{d}y \tag{2.25}$$

Then the process, defined by the transition density in (2.24)*, is the following SDE*

$$\begin{aligned} \mathrm{d}X_t &= \left[\mu(X_t, t) + \sigma^2(X_t, t)\frac{\mathrm{d}}{\mathrm{d}x}\log h(t, X_t)\right]\mathrm{d}t + \sigma(X_t, t)\,\mathrm{d}W_t, \quad t \in [0, T] \\ X_0 &= x_0. \end{aligned} \tag{2.26}$$

Proof:
Consider the transition density

$$p^h(y, t+s|x, t) = p(y, t+s|x, t)\frac{h(t+s, y)}{h(t, x)}. \tag{2.27}$$

Notice that this is a valid transition density, since if we integrate over all possible y-values we get one,

$$\begin{aligned}\int p^h(y,t+s|x,t)\,\mathrm{d}y &= \int p(y,t+s|x,t)\frac{h(t+s,y)}{h(t,x)}\,\mathrm{d}y \\ &= \frac{1}{h(t,x)}\int p(y,t+s|x,t)h(t+s,y)\,\mathrm{d}y = 1.\end{aligned} \tag{2.28}$$

Before obtaining the model in (2.26), we prove that $\tilde{A}h(x,t) = 0$ (defined in (2.15)), since it is used in the proof. We consider the first term in the numerator, i.e.,

$$\begin{aligned}\mathbb{E}[h(t+s,X_{t+s})|X_t = x] &= \int_y h(t+s,X_{t+s}=y)\,\mathrm{d}\mathbb{P}(X_{t+s}=y|X_t=x) \\ &= \int_y h(t+s,y)p(y,t+s|x,t)\,\mathrm{d}y \\ &= h(t,x).\end{aligned}$$

Then it follows that $\tilde{A}h(x,t) = 0$ (since the numerator becomes zero). We consider the generator $A^h\phi(x)$ which is defined as in (2.11), but with the expected value taken with respect to the probability density $p^h(y,t+s|x,t)$, i.e.,

$$A^h\phi(x) = \lim_{s\to 0}\frac{\mathbb{E}^h[\phi(X_{t+s})|X_t=x]-\phi(x)}{s} \tag{2.29}$$

This can be simplified in order to identify the drift and diffusion term respectively

$$\begin{aligned}&\lim_{s\to 0}\frac{\mathbb{E}^h[\phi(X_{t+s})|X_t=x]-\phi(x)}{s} \\ &= \lim_{s\to 0}\frac{\mathbb{E}[\phi(X_{t+s})h(t+s,X_{t+s})|X_t=x]-\phi(x)h(t,x)}{s\,h(t,x)} \\ &= \frac{1}{h(t,x)}A_t\{h(t,x)\phi(x))\} \\ &= \frac{1}{h(t,x)}\Big\{\frac{\mathrm{d}}{\mathrm{d}t}\Big(h(t,x)\phi(x)\Big)+\frac{\mathrm{d}}{\mathrm{d}x}\Big(h(t,x)\phi(x)\Big)\mu(x,t) \\ &\quad+\frac{1}{2}\frac{\mathrm{d}^2}{\mathrm{d}x^2}\Big(h(t,x)\phi(x)\Big)\sigma^2(x,t)\Big\} \\ &= \frac{1}{h(t,x)}\Big\{\Big[\underbrace{h'_t(t,x)+h'_x(t,x)\mu(x,t)+\frac{1}{2}h''_{xx}(t,x)\sigma^2(x,t)}_{\tilde{A}h=0}\Big]\phi(x) \\ &\quad+h(t,x)\phi'_x(x)\mu(x,t)+\frac{1}{2}\Big[2h'_x(t,x)\phi'_x(x)+h(t,x)\phi''_{xx}(x)\Big]\sigma^2(x,t)\Big\} \\ &= \Big[\mu(x,t)+\sigma^2(x,t)\frac{h'_x(t,x)}{h(t,x)}\Big]\phi'_x(x)+\frac{1}{2}\sigma^2(x,t)\phi''_{xx}(x) \\ &= \Big[\mu(x,t)+\sigma^2(x,t)\frac{\mathrm{d}}{\mathrm{d}x}\log(h(t,x))\Big]\phi'_x(x)+\frac{1}{2}\sigma^2(x,t)\phi''_{xx}(x).\end{aligned} \tag{2.30}$$

From this we see that X_t given the observations is a time inhomogeneous process. By identifying the drift and diffusion terms respectively, it follows that the new

(conditional) h-transformed SDE is given by

$$\begin{aligned} \mathrm{d}X_t &= \left[\mu(X_t,t) + \sigma^2(X_t,t)\frac{\mathrm{d}}{\mathrm{d}x}\log h(t,X_t)\right]\mathrm{d}t + \sigma(X_t,t)\,\mathrm{d}W_t, \quad t \in [0,T] \\ X_0 &= x_0. \end{aligned} \tag{2.31}$$

This concludes the proof. To see how (2.31) is formulated in the case of a multivariate SDE, see [3]. This case is, however, not considered in this report.

2.3.2 Conditioning on observations

Connecting the Doob-h transform to our problem is not cumbersome. We consider the transition density $p(y, t+s|x,t)$, but now conditioned on the observations $z_{1:N}$, i.e.,

$$p^h(x_{t+s}, t+s|x_t,t) = p(x_{t+s}|x_t, z_{1:N}). \tag{2.32}$$

We further assume $t_i < t < t+s \leq t_{i+1}$ and make the following manipulations

$$\begin{aligned} p(x_{t+s}|x_t, z_{1:N}) &= \frac{p(x_{t+s}, x_t, z_{1:N})}{p(x_t, z_{1:N})} \\ &= \frac{p(z_{i+1:N}|x_{t+s}, x_t, z_{1:i})p(x_{t+s}, x_t, z_{1:i})}{p(z_{i+1:N}|x_t, z_{1:i})p(x_t, z_{1:i})} \\ &= \frac{p(z_{i+1:N}|x_{t+s})p(x_{t+s}|x_t, z_{1:i})p(x_t, z_{1:i})}{p(z_{i+1:N}|x_t, z_{1:i})p(x_t, z_{1:i})} \\ &= p(x_{t+s}|x_t)\frac{p(z_{i+1:N}|x_{t+s})}{p(z_{i+1:N}|x_t)}. \end{aligned}$$

If we define $h(t+s, x_{t+s}) = p(z_{i+1:N}|x_{t+s})$ we get

$$p^h(x_{t+s}, t+s|x_t,t) = p(x_{t+s}|x_t)\frac{h(t+s,x_{t+s})}{h(t,x_t)}. \tag{2.33}$$

This means that we achieve the h-transformed SDE, in this setup, if we can show that this choice of h fulfils the property in (2.25). It does, since,

$$\begin{aligned} &\int p(x_{t+s} = y, t+s|x_t,t)p(z_{i+1:N}|x_{t+s} = y)\,\mathrm{d}y \\ &\quad = \int p(x_{t+s} = y|x_t)p(z_{i+1:N}|x_{t+s} = y, x_t)\,\mathrm{d}y \\ &\quad = \int p(z_{i+1:N}, x_{t+s} = y|x_t)\,\mathrm{d}y \\ &\quad = p(z_{i+1:N}|x_t = x) = h(t,x). \end{aligned}$$

Using this, the (conditional) h-transformed SDE is given by

$$\begin{aligned} \mathrm{d}X_t &= \left[\mu(X_t,t) + \sigma^2(X_t,t)\frac{\mathrm{d}}{\mathrm{d}x}\log p(z_{i+1:N}|X_t)\right]\mathrm{d}t + \sigma(X_t,t)\,\mathrm{d}W_t \quad t \in [t_{i-1}, t_i] \\ X_0 &= x_0. \end{aligned} \tag{2.34}$$

The meaning of this result deserves some reflection. It is remarkable that we achieve an analytical formula for the conditional SDE. Notice, that in any interval $(t_i, t_{i+1}]$ the conditional model does not depend on past observations, but it only depends on observations in the future. This result is incredibly important for understanding the setting needed for the neural SDE.

2.3.3 Special case of Brownian motion

In this example, we assume that the underlying SDE can be described as

$$\mathrm{d}X_t = \mathrm{d}W_t, \quad t \in [0, T], \; X_0 = x_0, \tag{2.35}$$

and observations are gathered as before, see (2.22). This means more specifically, that we have a so called state space model with the following characteristics

$$\begin{aligned} \pi(X_{t_k}|X_{t_{k-1}}) &\sim N(X_{t_{k-1}}, t_k - t_{k-1}) \\ \pi(Z_k|X_{t_k}) &\sim N(X_{t_k}, \sigma_Z^2). \end{aligned} \tag{2.36}$$

It should be added, that the state space model with a non-zero drift term is not considerably more difficult to derive, since the only difference is in the mean.

It is possible to show that

$$p(Z_{i+1:N}|X_t) = \prod_{j=i+2}^{N} \mathcal{N}(Z_j; \mu_{j-1,(t)}, \sigma_Z^2 + \sigma_{j-1,(t)}^2 + \Delta t)\mathcal{N}(Z_{i+1}; X_t, \sigma_Z^2 + t_{i+1} - t). \tag{2.37}$$

Furthermore, it can be shown, that the drift and diffusion terms can iteratively be defined as

$$\mu_{j,(t)} = \frac{Z_j(\sigma_{j-1,(t)}^2 + \Delta t) + \sigma_Z^2 \mu_{j-1,(t)}}{\sigma_Z^2 + \sigma_{j-1,(t)}^2 + \Delta t} \tag{2.38}$$

$$\sigma_{j,(t)}^2 = \frac{\sigma_Z^2(\sigma_{j-1,(t)}^2 + \Delta t)}{\sigma_Z^2 + \sigma_{j-1,(t)}^2 + \Delta t} \tag{2.39}$$

$$\forall j = i+2, ..., N. \tag{2.40}$$

with the initial conditions

$$\mu_{i+1,(t)} = \frac{Z_{i+1}(t_{i+1} - t) + \sigma_Z^2 X_t}{\sigma_Z^2 + t_{i+1} - t} \tag{2.41}$$

$$\sigma_{i+1,(t)}^2 = \frac{\sigma_Z^2(t_{i+1} - t)}{\sigma_Z^2 + t_{i+1} - t}. \tag{2.42}$$

Taking the logarithm and then the derivative with respect to X_t we obtain

$$\frac{\mathrm{d}}{\mathrm{d}x} \log p(Z_{i+1:N}|X_t) = \sum_{j=i+2}^{N} \frac{Z_j - \mu_{j-1,(t)}}{(\sigma_Z^2 + \sigma_{j-1,(t)}^2 + \Delta t)} \frac{\mathrm{d}}{\mathrm{d}x} \mu_{j-1,(t)} + \frac{Z_{i+1} - X_t}{(\sigma_Z^2 + t_{i+1} - t)} \tag{2.43}$$

where the derivative of the iterative drift term can be calculated as

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}x}\mu_{j-1,(t)} &= \frac{\sigma_Z^2 \frac{\mathrm{d}}{\mathrm{d}x}\mu_{j-2,(t)}}{\sigma_Z^2 + \sigma_{j-2,(t)}^2 + \Delta t}, \quad j = i+3, ..., N \\ \frac{\mathrm{d}}{\mathrm{d}x}\mu_{i+1,(t)} &= \frac{\sigma_Z^2}{\sigma_Z^2 + t_{i+1} - t}.\end{aligned} \tag{2.44}$$

For full derivation, see Appendix A.

A special case is when considering one noise-free observation. In that case, by using (2.43), it is possible to show that the conditional model can be written as

$$\mathrm{d}X_t = \frac{b - X_t}{T - t}\,\mathrm{d}t + \,\mathrm{d}W_t, \quad X_0 = a, \quad X_T = b. \tag{2.45}$$

This process is known as a **Brownian bridge**.

3
Signature

To solve the smoothing problem with our chosen approach we need to introduce the signature. The signature is a transformation from the path space to a tensor algebra. This transformation has several nice properties - most importantly that (almost) any function applied on the signature can be approximated by a linear map. This leads to that the minimal Wasserstein distance can be expressed as a minimization problem only, instead of a min-max problem which can be harder to solve.

More specifically, in this chapter we introduce the concept of the signature, both from a theoretical and practical standpoint. We start by, in Section 3.1, presenting theory regarding tensor algebra and needed vector spaces for the signature. Afterwards, in Section 3.2, the formal definition of the signature transformation applied on a specific set of paths is presented. The signature is then, in Section 3.2.1, interpreted geometrically and necessary conditions for achieving an injective mapping from the path space to the signature space are discussed in Section 3.2.2. Important theoretical results for expected signatures and more specifically the Wasserstein distance in the signature space are stated in Section 3.3. The chapter is then concluded with interpolation methods (Section 3.4) and augmentations (Section 3.5) of the process. Here we also decide which augmentations are suitable to consider in this thesis.

3.1 Tensor algebra

Consider two finite-dimensional vector spaces V and W (having the same field). The tensor product $V \otimes W$ is a vector space together with a bilinear mapping $(v, w) \mapsto v \otimes w$, spanned by vectors $b \otimes b'$, where $b \in B_V$ and $b' \in B_W$, and B_V and B_W are arbitrary bases for V and W, respectively. For any two elements $v = \sum_{b \in B_v} v_b b \in V$ and $w = \sum_{b' \in B_w} w_b b' \in W$, we have $v \otimes w = \sum_{b \in B_v, b' \in B_w} (v_b w_b) b \otimes b'$. From this it is easy to see that the following relation holds

$$\dim(V \otimes W) = \dim(V)\dim(W). \tag{3.1}$$

One important example is for Euclidean vector spaces V and W, in which case the bilinear map can be taken as the outer product, i.e., $(v, w) \mapsto vw^T$. Thus in this case $v \otimes w$ is a rank-one matrix.

In this text we consider only one vector space, so for simplicity we denote it as

E for the rest of the text. We consider several orders of tensor powers (not only two as above), so we denote the k:th order as $E^{\otimes k}$. If E is spanned by the basis vectors $e_1, ..., e_d$, $E^{\otimes k}$ can be interpreted as a word of length k where each letter can attain d values (one of the letters $\{e_1, ...e_d\}$). If E has dimension n then $E^{\otimes k}$ has dimension n^k. Below we make a formal definition for the tensor space of all orders.

Definition 3 (Tensors series) *The space of all formal E-tensors series, denoted by $T(E)$, is defined as*

$$T(E) = \left\{ \mathbf{a} = (a_0, a_1, ...) \middle| a_n \in E^{\otimes n}, \forall n \geq 0 \right\}. \tag{3.2}$$

It is equipped with two operations, an addition and a product, defined as follows: for $\mathbf{a} = (a_0, a_1, ...)$, $\mathbf{b} = (b_0, b_1, ...) \in T(E)$ the binary operations $+$ and $\otimes$ perform the actions

$$\begin{aligned} \mathbf{a} + \mathbf{b} &= (a_0 + b_0, a_1 + b_1, ...) \\ \mathbf{a} \otimes \mathbf{b} &= (c_0, c_1, ...) \end{aligned} \tag{3.3}$$

where $c_n = \sum_{j=0}^{n} a_j \otimes b_{n-j}$.

The reason for introducing this tensor algebra is that the signature is an element of it. However, in implementations it is impossible to work with infinite series, which means we need to define a truncated version of this space which has the same properties. We define the truncation of order m of an element $\boldsymbol{a} \in T(E)$ as

$$\pi_m(\boldsymbol{a}) = (a_0, a_1, ..., a_m). \tag{3.4}$$

3.2 Signature definition

In order to define the signature we introduce the $k-$fold iterated integral of a path X. The paths considered are those that are continuous, d dimensional and of finite 1-variation over the time interval $[a, b]$. The set of these paths are denoted as $\Psi([a, b], \mathbb{R}^d)$. Let $I = (i_1, ..., i_k)$ be a multi-index of length k where $i_1, ..., i_k \in \{1, 2, 3, ...d\}$. Let $X^{(i)}$ denote the i^{th} coordinate of X, which is a real-valued function. The iterated integral of X over $[a, b]$ indexed by I is defined as

$$S_{a,b}^{(i_1,i_2,...,i_k)}(X) = \int_{a \leq t_1 \leq t_2 ... t_k \leq b} \mathrm{d}X_{t_1}^{(i_1)} \mathrm{d}X_{t_2}^{(i_2)} ... \mathrm{d}X_{t_k}^{(i_k)}. \tag{3.5}$$

The integrals here should be seen in the Riemann-Stieltjes sense. An equivalent way of representing it is by the recursive formula

$$S_{a,b}^{(i_1,i_2,...,i_k)}(X) = \int_a^b S_{a,t_k}^{(i_1,i_2,...,i_{k-1})}(X) \, \mathrm{d}X_{t_k}^{(i_k)}, \text{where } S^{\emptyset}(X) = 1. \tag{3.6}$$

We can write all possible iterated integrals with multi-indices of length k in a compact way using tensor algebra

$$\mathbf{X}_{a,b}^{k} = \int_{a \leq t_1 \leq t_2 ... \leq b} \mathrm{d}X_{t_1}^{(i_1)} \otimes \mathrm{d}X_{t_2}^{(i_2)} \otimes ... \otimes \mathrm{d}X_{t_k}^{(i_k)}. \tag{3.7}$$

This is an element in the tensor algebra of $E^{\otimes k}$. Now we can define the signature of a path.

Definition 4 (The signature of a path) *Consider the interval* $[a, b]$ *and* $X \in \Psi([a, b], \mathbb{R}^d)$ *such that the integration in* (3.5) *makes sense. The signature* $S(X)$ *of* X *over the time interval* $[a, b]$ *is the infinite series*

$$S(X)_{a,b} = (1, \boldsymbol{X}^1_{a,b}, ..., \boldsymbol{X}^n_{a,b}, ...) \in T(E) \tag{3.8}$$

where $\mathbf{X}^k_{a,b}$ *is defined as in* (3.7).

Meanwhile, the truncated signature of X of order m, denoted as $S^m(X)$, is defined as

$$S^m(X)_{a,b} := \pi_m(S(X)) = (1, \boldsymbol{X}^1_{a,b}, ..., \boldsymbol{X}^m_{a,b}). \tag{3.9}$$

Henceforth, the signature is denoted without the interval it is defined over. If it is relevant, it is clear from the context which interval is considered. We call the time interval J and generally speaking, in the thesis, we have $J = [0, T]$. It is also important to note that each dimension of X does not have the same meaning and range, but each dimension is connected to the same time interval and each dimension is in $\mathbb{R}$. Furthermore, we also define $S(\Psi(J, E))$ as the range of the signature defined over the time interval J.

3.2.1 Geometrical interpretation

The interpretation of the signature could initially be hard to grasp, especially the geometrical interpretation. Therefore, we show how the lower order terms of the signature of a simple process can be interpreted and visualized. We consider the example where X is the discrete time series that attains four values $[0, 4, 2, 5]$ at the time instances $[0, 1, 2, 3]$. Furthermore, we assume that the process is augmented with time, i.e., we have the process $\bar{X} = (X, t)$. We consider linear interpolation between the values (a piecewise linear function). Using the iterated integral in (3.5) we can calculate the first order terms of the signature as

$$S^{(0)}(\bar{X}) = X_3 - X_0 = 5 \qquad S^{(1)}(\bar{X}) = t_3 - t_0 = 3. \tag{3.10}$$

For the second order, there are four terms to consider: $S^{(0,0)}, S^{(0,1)}, S^{(1,0)}$ and $S^{(1,1)}$. The terms containing the same index can be calculated, according to [5], by the identity

$$S(\overbrace{i, i, ..., i}^{d})(\bar{X}) = \frac{(\bar{X}^{(i)}_T - \bar{X}^{(i)}_0)^d}{d!}. \tag{3.11}$$

This means we get

$$S^{(0,0)}(\bar{X}) = \frac{5^2}{2} \qquad S^{(1,1)}(\bar{X}) = \frac{3^2}{2}. \tag{3.12}$$

These terms have no immediate way to be visualized, however, the two remaining terms have nice geometrical interpretations. The two remaining terms can be

calculated by the iterative formula found in (3.6), e.g., for $S^{(0,1)}$ we get

$$S^{(0,1)}(\bar{X}) = \int_0^{t_3} X_t \, \mathrm{d}t. \tag{3.13}$$

This is the area under the curve and similarly we get that $S^{(1,0)}$ is the area over the curve. For visualization of $S^{(0,1)}$ and $S^{(1,0)}$, see Figure 3.1. Visualizing higher order terms is hard, but the idea is that each element in the signature captures a specific characteristic of the path such that the signature uniquely determines the path. This concept is examined in the following section.

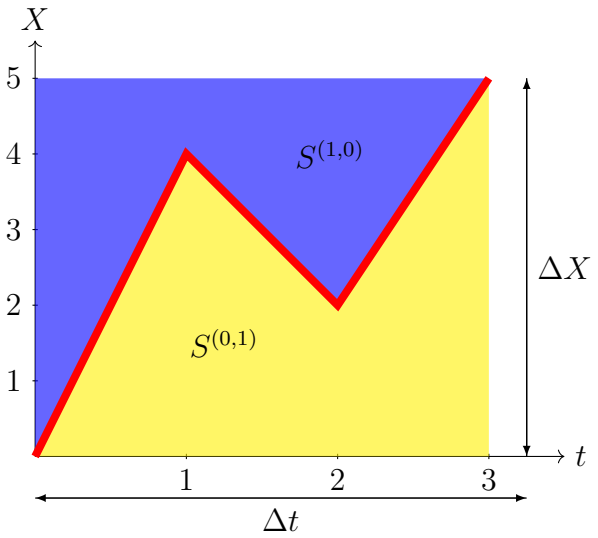


Figure 3.1: Two of the signature elements, $S^{(0,1)}$ and $S^{(1,0)}$, of the process $\bar{X} = (X, t)$ visualized. The process is defined as $X_0 = 0, X_1 = 4, X_2 = 2, X_3 = 5$ and values between have been calculated by considering linear interpolation.

3.2.2 Characterization of paths by means of signatures

We want the mapping $S : \Psi(J, E) \to S(\Psi(J, E))$ to be injective. The reason is because we in this project consider signatures and through this determine distributions on the path space. For a specific set of paths this is true. Consider the set, denoted as $\Psi_{\text{mon}}(J, E)$, which consists of all path in $\Psi(J, E)$ where at least one coordinate of X is a monotone function. Then the injective property is true according to Theorem 5 which is proven in Lemma 2.14 in [6].

Theorem 5 (Uniqueness of signature) *Let $X_0 \in E$. For $X \in \Psi_{\text{mon}}(J, E)$ with $X_0 = x_0$, then the signature of X uniquely determines X.*

Theorem 5 requires a fixed starting point for the signature to be unique. This means, according to [7], that the signature is invariant to translation of paths. Notice that it does not need to be the same translation in each dimension. This means that the signature does not generally fulfil the desired property of injectivity. However, since the paths which are considered in this thesis are augmented with time (a monotone

function), see Section 3.5, and we assume the initial value of the process is known, our paths fulfil Theorem 5.

3.3 Wasserstein metric in signature space

In this section we introduce the Wasserstein distance in signature space, the metric used in this thesis to quantify similarity between two distributions. In order to understand this metric, we define it in Euclidean space. Afterwards, we make the analogue definition in signature space. In addition, important theorems regarding this distance in signature space are presented.

3.3.1 Wasserstein distance in Euclidean space

The Wasserstein distance is a metric which measures how similar two distributions, μ and ν, are. It is defined as

$$W_p(\mu,\nu) = \left(\inf_{\pi \in \Pi(\nu,\mu)} \int c(x,y)^p \, \mathrm{d}\pi(x,y) \right)^{1/p}. \tag{3.14}$$

Here $c(x,y)$ is the cost function corresponding to the cost of moving mass from x to y and $\Pi(\mu,\nu)$ is the set of distributions whose marginal distributions are μ and ν. In this text we consider the Wasserstein-1 distance, i.e.,

$$W_1(\mu,\nu) = \inf_{\pi \in \Pi(\nu,\mu)} \int c(x,y) \, \mathrm{d}\pi(x,y). \tag{3.15}$$

This can (according to Kantorovich-Rubinstein duality, see [8] for proof) be rewritten as a maximum problem

$$W_1(\mu,\nu) = \sup_{\varphi \in \mathrm{Lip}_1} \left(\int \varphi(x) \, \mathrm{d}\mu(x) - \int \varphi(x) \, \mathrm{d}\nu(x) \right) \tag{3.16}$$

where

$$\mathrm{Lip}_1 = \{ \varphi : |\varphi(x) - \varphi(y)| \leq c(x,y), x \in \mu, \ y \in \nu \}. \tag{3.17}$$

In this text we only consider the Euclidean distance as the cost function, i.e., $c(x,y) := ||x-y||$. Notice as well, that the function φ should have Lipschitz constant less than or equal to one.

The integrals can be seen as expected values and (3.16) can therefore equivalently be written

$$W_1(\mu,\nu) = \sup_{\varphi \in \mathrm{Lip}_1} \mathbb{E}_{X\sim\mu}[\varphi(X)] - \mathbb{E}_{X\sim\nu}[\varphi(X)]. \tag{3.18}$$

The Wasserstein distance can be used when the goal is to replicate a fixed distribution ν. Then an approximate μ is found by minimizing $W_1(\mu,\nu)$ with respect to μ, constituting a min-max problem.

3.3.2 Extending to signature space

Here we derive the Signature Wasserstein-1 (Sig-W_1) metric, which is a special case of the Wasserstein metric (defined in Section 3.3.1) in the signature space. We start by considering $f : M \to \mathbb{R}$, where (M, D) is a generic metric space and define

$$||f||_{\text{Lip},M} := \sup_{x \neq y, x, y \in M} \frac{f(x) - f(y)}{D(x, y)}. \tag{3.19}$$

Here D is a metric defined on M and in this thesis the ℓ^p norm is considered. We assume that μ and ν are two compactly supported measures on the path space $\Psi(J, E)$ such that their corresponding induced measures (pushforward measures) on the signature space $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ respectively have a compact support $\mathcal{K} \subset S(\Psi(J, E)) \subset \boldsymbol{T}^p(E)$. With the pushforward measure we mean $\boldsymbol{\mu} := \mu(S^{-1}(B))$ for any B in the σ-algebra of $S(\Psi(J, E))$, meanwhile $\boldsymbol{T}^p(E)$ is defined as the set of all elements in $T(E)$ with finite ℓ^p norm. In the signature space the Wasserstein distance is defined as

$$W_1^{\text{Sig space}}(\mu, \nu) = \sup_{||\boldsymbol{f}||_{\text{Lip},\mathcal{K}} \leq 1} \mathbb{E}_{S \sim \boldsymbol{\mu}}[\boldsymbol{f}(S)] - \mathbb{E}_{S \sim \boldsymbol{\nu}}[\boldsymbol{f}(S)]. \tag{3.20}$$

We introduce the following theorem which states that it is possible to approximate every continuous function on the signature space arbitrarily well by a linear functional.

Theorem 6 (Universality of Signature) *Consider a compact set $\mathcal{K} \subset S(\Psi(J, E))$. Let $\boldsymbol{f} : \mathcal{K} \mapsto \mathbb{R}$ be any continuous function. Then for any $\epsilon > 0$, there exists a linear functional $\boldsymbol{L} \in T(E)^*$ (the dual space) acting on a signature such that*

$$\sup_{S \in \mathcal{K}} |\boldsymbol{f}(S) - \boldsymbol{L}(S)| < \epsilon \tag{3.21}$$

This motivates us to consider the Sig-$W_1(\mu, \nu)$ metric which is defined as

$$\text{Sig-}W_1(\mu, \nu) = \sup_{||\boldsymbol{L}||_{\text{Lip}} \leq 1, \text{L is linear}} \mathbb{E}_{S \sim \boldsymbol{\mu}}[\boldsymbol{L}(S)] - E_{S \sim \boldsymbol{\nu}}[\boldsymbol{L}(S)]. \tag{3.22}$$

We state Lemma 7 which states that it is possible to disregard explicit consideration of the linear map when maximizing the Wasserstein distance in signature space.

Lemma 7 (Duality) *Fix $p, q > 1$ such that $\frac{1}{p} + \frac{1}{q} = 1$, then for any linear functional $\boldsymbol{L} \in \mathbf{T}^p(E)^*$, it holds that*

$$\sup_{||a||_p = 1} |\boldsymbol{L}a| = ||\boldsymbol{L}||_q, \tag{3.23}$$

Similarly, for any $a \in \mathbf{T}^p(E)$, it holds that

$$\sup_{||\boldsymbol{L}||_q \leq 1} |\boldsymbol{L}a| = \sup_{||\boldsymbol{L}||_q = 1} |\boldsymbol{L}a| = ||a||_p \tag{3.24}$$

This means that we, in this setup, have the following relation

$$||\boldsymbol{L}||_{\text{Lip}} = \sup_{x \neq y, x, y \in \boldsymbol{T}^p(E)} \frac{|\boldsymbol{L}(x - y)|}{||x - y||_p} = \sup_{||a||_p = 1} |\boldsymbol{L}a| = ||\boldsymbol{L}||_q. \tag{3.25}$$

Moving $\boldsymbol{L}$ outside the expectation in (3.22), and using the second relation of Lemma 7, gives

$$\text{Sig-}W_1(\mu,\nu) = \sup_{||\boldsymbol{L}||_q \leq 1} \boldsymbol{L}\Big(\mathbb{E}_{S\sim\boldsymbol{\mu}}[S] - \mathbb{E}_{S\sim\boldsymbol{\nu}}[S]\Big) = ||\mathbb{E}_{S\sim\boldsymbol{\mu}}[S] - \mathbb{E}_{S\sim\boldsymbol{\nu}}[S]||_p. \quad (3.26)$$

It should be added that the fact that $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ have compact support on $\mathcal{K}$ makes sure that $\mathbb{E}_{S\sim\boldsymbol{\mu}}(S)$ and $\mathbb{E}_{S\sim\boldsymbol{\nu}}(S)$ are in $\boldsymbol{T}^p(E)$ which was a necessary condition for using Lemma 7. In this thesis we always consider data on the path space and we therefore introduce the following Lemma.

Lemma 8 *For two measures μ, ν on the path space $\Psi([0,T],E)$ such that their induced (pushforward) measures $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ have a compact support $\mathcal{K} \subset S(\Psi([0,T],E))$. Then it holds that*

$$\text{Sig-}W_1(\mu,\nu) = ||\mathbb{E}_{S\sim\boldsymbol{\mu}}[S] - \mathbb{E}_{S\sim\boldsymbol{\nu}}[S]||_p = ||\mathbb{E}_{X\sim\mu}[S(X)] - \mathbb{E}_{X\sim\nu}[S(X)]||_p. \quad (3.27)$$

This means that we have motivated that it is possible to interchangeably consider measures defined on signature and path space.

Remember, it is not possible to consider the signature transform in implementations, and therefore the truncated signature needs to be considered. Therefore, we define

$$\text{Sig-}W_1^m(\mu,\nu) = ||\mathbb{E}_{X\sim\mu}[S^m(X)] - \mathbb{E}_{X\sim\nu}[S^m(X)]||_p \quad (3.28)$$

which is approximately equal to Sig-$W_1(\mu,\nu)$ in the sense of distributions on the path space.

3.3.3 Expected signature

Considering the derivation in the previous section regarding quantifying similarity between two measures in the signature space, we need to determine if a zero Wasserstein distance in signature space leads to equality between distributions in the path space (since it is the path space that is of interest in this project). This means that we want to have the following property

$$\mathbb{E}_{S\sim\boldsymbol{\mu}}[S(X)] = \mathbb{E}_{S\sim\boldsymbol{\nu}}[S(X)] \implies \mu = \nu. \quad (3.29)$$

This property is true, according to Theorem 9, in some specific settings. The specific setting considered in the theorem is that of infinite radius of convergence. When this is fulfilled or not, is not of focus in this report. Nevertheless, it is important to highlight that this property is fulfilled in specific settings.

Theorem 9 (Equivalent measures in expected signature) *Let μ and ν be two measures on the path space $\Psi(J,E)$. Let $\boldsymbol{\mu}(B) = \mu(S^{-1}(B))$ and $\boldsymbol{\nu} = \nu(S^{-1}(B))$ for B in the σ-algebra of $S(\Psi(J,E))$. Suppose that $\mathbb{E}_{S\sim\boldsymbol{\mu}}[S]$ exists and has infinite radius of convergence. If $\mathbb{E}_{S\sim\boldsymbol{\mu}}[S] = \mathbb{E}_{S\sim\boldsymbol{\nu}}[S]$, then $\mu = \nu$.*

3.4 Process interpolation

In our report we consider discretized SDEs which are sampled using numerical schemes. This means we cannot calculate the signature, nor the truncated signature, since the generated paths from the SDE are not continuously defined over the time interval. To evaluate the signature, we need to interpolate the values of the simulated paths. Generally speaking, the only condition on the interpolation is that it should guarantee an injective mapping from the path space to the signature space.

There are several approaches for this, but we have decided on piecewise linear interpolation, since it is simple to implement and interpret. Furthermore, it also seems to be the standard approach used in similar projects, see for example the approach in [2]. The author of [2] has also created a python package, which we use, for fast signature calculations with piecewise linear interpolation as chosen interpolation method. However, no reason for why it is chosen is mentioned. In [5] another method is mentioned - rectilinear interpolation. This is similar to linear interpolation, but the interpolated path instead attains the last obtained value and then makes a jump to the new value when passing it.

3.5 Process augmentations

Due to considering truncated signatures, important characteristics captured in higher order terms could be lost. Therefore, it is reasonable to consider augmentations that can help highlighting these characteristics in lower order terms. One such important augmentation of the process is the quadratic variation, denoted as $[X]_t$ in this thesis. Due to discretization of paths, we approximate $[X]_t$ as $\sum_{i=1}^{j}(X_{t_i} - X_{t_{i-1}})^2$ at t_j. Between t_j and t_{j+1} it is defined by the linear interpolation between the discrete values. The quadratic variation is (obviously) strongly related to the volatility of a path, meaning the signature can easily capture this characteristic when augmented.

Another reasonable augmentation is time. This augmentation is useful when learning from irregularly sampled data and especially for when designing training methods that desirably should work with high and ultra-high frequency data [1]. Another reason to augment with time is that it is a strict monotone function. A monotone function, in at least one of the dimensions of the process, is a necessity for the signature transform to be an injective mapping, as seen in Theorem 5.

3.5.1 Effect of quadratic variation augmentation

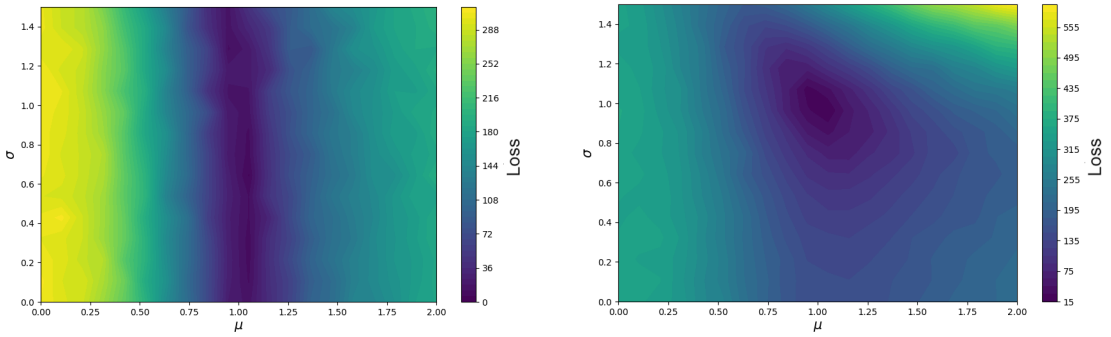
To visualize the importance of augmenting a process with the quadratic variation, we consider the SDE

$$\mathrm{d}X_t^{(\mu,\sigma)} = -\mu X_t^{(\mu,\sigma)}\,\mathrm{d}t + \sigma\,\mathrm{d}W_t, \quad t \in [0,2],\ X_0 = 0. \tag{3.30}$$

We compare different μ and σ values against the base case of $\mu = 1$ and $\sigma = 1$ for the objective function

$$\mathcal{L}(\mu,\sigma) = ||\mathbb{E}[S^4(X^{(1,1)})] - \mathbb{E}[S^4(X^{(\mu,\sigma)})]||_2. \tag{3.31}$$

$\mathcal{L}(\mu, \sigma)$ is visualized in Figure 3.2 for two augmented processes. The first process is augmented with only time and the second process is augmented with time and the quadratic variation.



(a) The augmented process (X, t). **(b)** The augmented process $(X, t, [X]_t)$.

Figure 3.2: The landscape of (3.31) for varying μ and σ. To approximate each expected value, the Monte Carlo method with 500 samples has been used.

When the quadratic variation is not included we can see that $\mathcal{L}(\mu, \sigma)$ is insensitive to different σ values. In contrast, when augmenting with the quadratic variation, there is a clear minimum of the objective function for the correct σ. The reason for considering this objective function is its relation to the Wasserstein metric in signature space introduced in Section 3.3.

Concluding this section, we state that the signature throughout the rest of this thesis should be seen as it is applied on the augmented process $\bar{X} = (X_t, t, [X]_t)$.

4

Neural Networks and GANs

Recall that the drift and diffusion coefficients of the neural SDE in (1.3) are parameterized as neural networks (NN). Therefore, we introduce the concept of NN in this chapter. Neural networks constitute a family of machine learning models constructed to imitate the functions of the human brain. We focus on a specific branch of NN, the Multilayer Perceptron (MLP), which is the most commonly used NN. We define it in Section 4.1, and the general procedure regarding training it is introduced in Section 4.2.

The goal of the neural SDE is to replicate an unknown distribution, which is why the concept of Generative Adversarial Networks (GAN) is introduced in Section 4.3. This is a special network structure that utilizes a generator (the neural SDE) to generate samples. The generator is trained by comparing its samples against samples from the true distribution, such that the distribution of the generator can replicate the true distribution. The introduction of GAN networks is done chronologically, starting with the Vanilla GAN (Section 4.3.1), followed by the Wasserstein GAN (Section 4.3.2). From this, we, in Section 4.4, introduce the conditional Wasserstein GAN and, in Section 4.4.2, we make the definition analogously in signature space, which is the final network concept used in this thesis. We end this chapter by, in Section 4.5, introducing a method for approximating conditional expectations using neural networks.

4.1 Multilayer perceptron

The **multilayer perceptron** (MLP) is a fully connected feedforward neural network consisting of N layers. It maps an input $\mathbf{x}_0 \in \mathbb{R}^m$ to an output $\mathbf{x}_N \in \mathbb{R}^n$ by propagating it through the network. The MLP is said to be parameterized by weight matrices $\theta = (\theta_1, ..., \theta_N)$, and bias vectors, $\mathbf{b} = (\mathbf{b}_1, ..., \mathbf{b}_N)$, where each matrix θ_i describes the transition weights between each layer and $\mathbf{b_i}$ is an additive term for each neuron in layer i. The forward propagation of the MLP is defined as

$$\begin{aligned} \mathbf{x}_i &= g(\theta_i \mathbf{x}_{i-1} + \mathbf{b}_i), \quad i = 1, \dots, N-1 \\ \mathbf{x}_N &= \theta_N \mathbf{x}_{N-1} + \mathbf{b}_N. \end{aligned} \tag{4.1}$$

Here g is an **activation function** that operates elementwise on the vector input. The layers between the input and output layer are called hidden layers, which means in the above introduced notation there are $N-1$ hidden layers.

The main purpose of the activation function is to allow the network to learn nonlinearities, which is possible according to the universal approximation theorem. This theorem states that for any sufficiently regular function, there exists a feedforward network that approximates it with an arbitrary tolerance. For the interest of the reader, we specify the universal approximation theorem explicitly. Consider the set of feedforward neural networks, with any continuous non-polynomial activation function φ, with d neurons in the input layer, one neuron in the output layer and a single hidden layer with an arbitrary number of neurons, and call this set $\mathcal{N}_d^\varphi$. Then, for any compact set $\mathcal{K} \subseteq \mathbb{R}^d$ and function $f \in C(K)$, there exists a sequence of functions $(f_n)_{n=1}^\infty \subset \mathcal{N}_d^\varphi$ such that for any ϵ there exists a N such that for all $n > N$ and $\mathbf{x} \in \mathcal{K}$ it holds

$$|f_i(\mathbf{x}) - f(\mathbf{x})| < \epsilon.$$

4.2 Training a network

For a network to learn, it needs a set of data to fit its parameters, $\mathbf{W} = (\theta, \mathbf{b})$, to. Define the training data, $\mathcal{T} = \{\mathbf{x}_0^i\}_{i=1}^N$, where N is the number of observations. The goal is to choose the parameters of the network in such a way that the network produces values that are optimal according to an objective function. This function is often called the **loss function** and we denote it as L. More mathematically, the optimal parameters are chosen by minimizing

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} L(\mathbf{W}; \mathcal{T}). \tag{4.2}$$

Usually, the optimal $\mathbf{W}^*$ is approximated using some form of gradient descent algorithm. There are several possible choices, and we could write an entire report on the importance of the choice of optimizer. However, that is not the focus of this thesis. For the interested reader, a few are stated and briefly discussed.

Gradient descent, which is the simplest one, can be formulated as

$$\mathbf{W}_{k+1} \leftarrow \mathbf{W}_k - \eta \nabla_W L(\mathbf{W}_k; \mathcal{T}), \quad k \geq 0, \quad W_0 \sim \mathcal{D}. \tag{4.3}$$

Here η is the **learning rate** which is a so-called **hyper-parameter**, meaning it is chosen prior to running the algorithm. The most common approach for choosing the initial state W_0 is to randomly assign values from some distribution $\mathcal{D}$.

One of the advantages with using the gradient descent optimizer is that it is simple to understand and implement. However, it is more prone to get stuck in local optimas compared to other optimizers, due to it being deterministic given each input. A solution is to consider **mini-batches**. A mini-batch means that instead of considering the entire dataset $\mathcal{T}$, a random subset of $\mathcal{T}$ is chosen each iteration. We denote this random subset of size n as $\mathcal{T}^n$. Gradient descent, with the usage of mini-batches, is known as the **stochastic gradient descent** (**SGD**) algorithm. The update step for it can be formulated as

$$\mathbf{W}_{k+1} \leftarrow \mathbf{W}_k - \frac{\eta}{n} \nabla_W L(\mathbf{W}_k; \mathcal{T}^n). \tag{4.4}$$

To be precise, for the SGD-optimizer, only one datapoint ($n = 1$) should be considered, but we do not make that distinction here. This is one of the most popular algorithms, likely because of its simplicity and its ability to avoid local optimas. However, for any problem, it is still cumbersome to choose the best learning rate. If a small learning rate is chosen, convergence might be slow and if a large learning rate is chosen, the global optima might be missed. These two concepts are visualized for a one-dimensional problem (one weight) in Figure 4.1.

A solution to the problems regarding choosing the learning rate is to implement an **adaptive learning rate**. This is a learning rate that depends on the gradients. An optimizer which is currently popular and that implements this is the **ADAM**-optimizer. It updates exponential moving averages of the gradient and the squared gradient, where there are two hyper-parameters, β_1 and β_2, controlling each one of these. The moving averages are estimates of the first moment (the mean) and the raw second moment (the uncentered variance). These moments are initialized to zero and are then updated as

$$\begin{aligned} m_k &= \beta_1 \cdot m_{k-1} + (1 - \beta_1) \cdot g_k \\ v_k &= \beta_2 \cdot v_{k-1} + (1 - \beta_2) \cdot g_k^2 \end{aligned} \tag{4.5}$$

where $g_k = \nabla_W L(W_k; \mathcal{T}^n)$. The bias corrected estimates are calculated

$$\begin{aligned} \tilde{m}_k &= \frac{m_k}{1 - \beta_1^k} \\ \tilde{v}_k &= \frac{v_k}{1 - \beta_2^k} \end{aligned} \tag{4.6}$$

where β_i^k is β_i to the power k. The weight update is then formulated as

$$W_k = W_{k-1} - \frac{\alpha \cdot \tilde{m}_k}{\sqrt{\tilde{v}_k} + \epsilon}. \tag{4.7}$$

Here there are two additional hyper-parameters: α and ϵ. According to [9] a good initialization for the hyper-parameters found in ADAM is: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$, but of course it is problem specific.

In addition to the optimizers mentioned, there are several more optimizers that could be considered, for example the **Adadelta**-optimizer. It is namely, often difficult to choose the best optimizer for a specific problem, due to the complexity of the problem and poor understanding of the loss landscape [10].

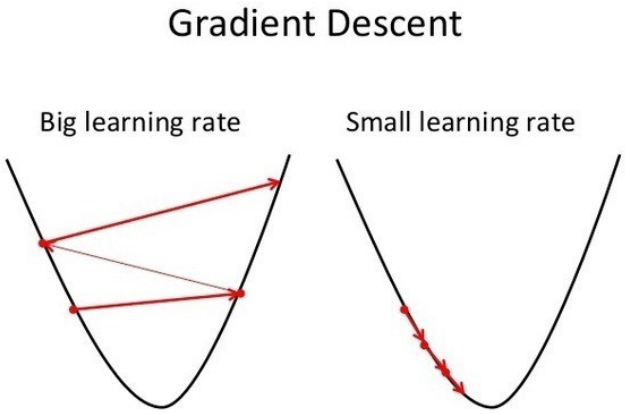


Figure 4.1: The picture on the left illustrates the consequence of a learning rate that is too big, while the picture on the right shows the same but for a too small learning rate.

4.3 Estimate unknown distribution from data

Consider the data, $\{x^{(i)}\}_{i=1}^{N}$, generated from a distribution μ_{ref}. In this example the distribution is unknown, but it is assumed to exist and the dataset is assumed rich enough to approximately represent it. The goal is to find this distribution, or rather approximate it, using the data. An initial approach would be to define a parametric family of densities $(\mu_\theta)_{\theta\in\mathbb{R}^d}$ and then find the density maximizing the expected likelihood of our data, i.e.,

$$\max_{\theta\in\mathbb{R}^d} \frac{1}{N}\sum_{i=1}^{N} \log \mu_\theta(x^{(i)}). \tag{4.8}$$

According to [11], finding the best approximation to the reference distribution, in the sense of this cost function, is equivalent to minimizing the Kullback-Leibler divergence $KL(\mu_{\text{ref}}||\mu_\theta)$, which is defined as

$$KL(\mu_{\text{ref}}||\mu_\theta) = \int_{\mathcal{X}} \mu_{\text{ref}}(x) \log \frac{\mu_{\text{ref}}(x)}{\mu_\theta(x)}\, \mathrm{d}x. \tag{4.9}$$

Minimizing this is possible without the knowledge of μ_{ref}, since

$$\int_{\mathcal{X}} \mu_{\text{ref}}(x) \log \frac{\mu_{\text{ref}}(x)}{\mu_\theta(x)}\, \mathrm{d}x = \int_{\mathcal{X}} \mu_{\text{ref}}(x) \log \mu_{\text{ref}}(x)\, \mathrm{d}x - \int_{\mathcal{X}} \mu_{\text{ref}}(x) \log \mu_\theta(x)\, \mathrm{d}x, \tag{4.10}$$

and when minimizing with respect to θ by Monte Carlo methods it reads

$$\arg\min_\theta KL(\mu_{\text{ref}}||\mu_\theta) \approx \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} -\log \mu_\theta(x^{(i)}). \tag{4.11}$$

The Kullback-Leibler divergence, however, has its drawbacks. For example, the assumption that the parametric family accurately describes the reference distribution might be bad. One case of this is that it breaks down in the rather common scenario

where we have distributions supported by low dimensional manifolds (data concentrated at lower dimensions subsets that could be hard to capture). This would lead to a model not respecting the manifold constraint. [11]

In this thesis, the approach to solve this is by the usage of generative neural networks and more specifically **Generative Adversarial Networks (GAN)**. However, depending on how the GAN is designed (which distance metric that is considered), this problem is solved with varying results.

To arrive at the GAN considered in this text, which is the Wasserstein GAN, we first introduce the Vanilla GAN so that the concept of GAN is clear.

4.3.1 Vanilla GAN

Consider the setup of two separate neural networks: the **generator network** G and the **discriminator network** D. The generator network takes as input a noise vector $\mathbf{z} \sim p_z$ (from the so called **latent space**) and outputs a generated sample $\mathbf{x} = G(\mathbf{z}; \mathbf{W_G})$, where $\mathbf{W_G}$ are the generator weights. This is equivalent to saying that $\mathbf{x}$ is sampled from the distribution of the generator μ_G. In addition to this we have data that are generated from the reference distribution μ_{ref}. The goal of the discriminator is to distinguish between these distributions; more specifically it generates the probability that the data is generated from μ_{ref} and a complementary probability that it is generated from μ_G. The probability is generated from the network $D(\mathbf{x}; \mathbf{W_D})$, where $\mathbf{W_D}$ are the discriminator weights. For a common visualization of the GAN, see Figure 4.2.

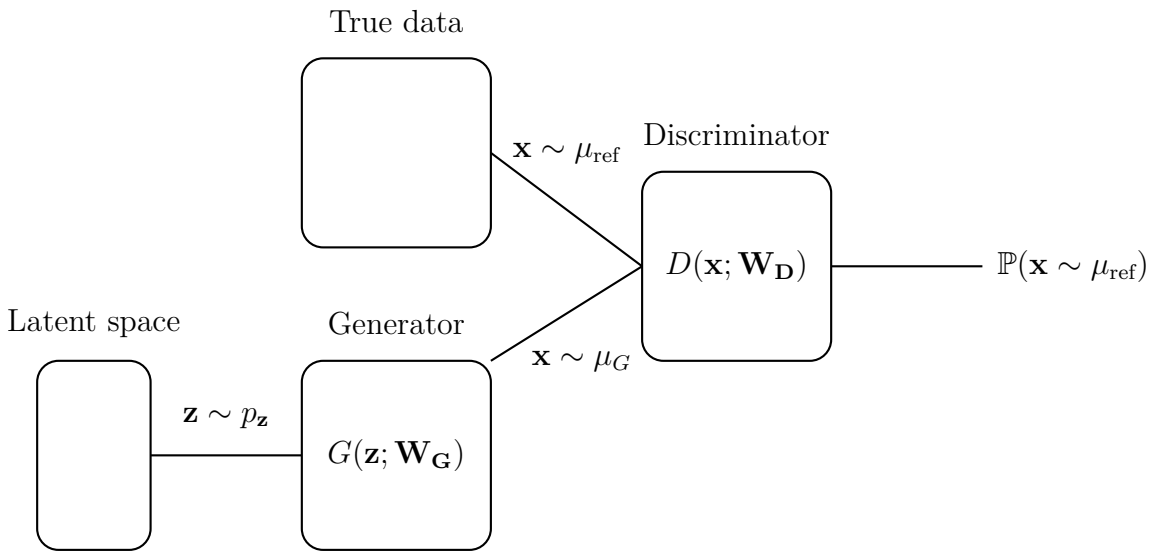


Figure 4.2: Figure showing the concept of GAN.

The setup described above leads us to consider the usage of the binary cross entropy loss function for the discriminator, which is defined as

$$L(y_t, y_p) = -y_t \log(y_p) - (1 - y_t) \log(1 - y_p). \tag{4.12}$$

Here y_t is the true label and y_p is the predicted label. However, we do not explicitly have labels in this case, which means we need to analyse what happens for each sample type. A sample from the true distribution ($y_t = 1$) yields

$$L(1, y_p) = -\log(y_p) \tag{4.13}$$

which we want to minimize. Furthermore, when inputting a false sample ($y_t = 0$), we achieve

$$L(0, y_p) = -\log(1 - y_p) \tag{4.14}$$

which we also want to minimize. This yields (in our problem formulation) the following expression to minimize

$$-\mathbb{E}_{\mathbf{x}\sim\mu_{\text{ref}}}[\log(D(\mathbf{x}))] - \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))], \tag{4.15}$$

or equivalently the maximization of

$$\mathbb{E}_{\mathbf{x}\sim\mu_{\text{ref}}}[\log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))] \tag{4.16}$$

The variables $\boldsymbol{W_G}$ and $\boldsymbol{W_D}$ parameterizing the networks have been omitted for more readable expressions. For the generator, we want the samples generated to be as close as possible to the reference distribution, meaning that we want to minimize

$$\mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))]. \tag{4.17}$$

This means, we have to the following two player min-max game with value function $L(\mu_G, \mu_D)$,

$$\min_{\mu_G}\max_{\mu_D} L(\mu_G, \mu_D) = \min_{\mu_G}\max_{\mu_D} \mathbb{E}_{\mathbf{x}\sim\mu_{\text{ref}}}[\log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))]. \tag{4.18}$$

If we instead consider μ_G directly, we instead achieve

$$\min_{\mu_G}\max_{\mu_D} L(\mu_G, \mu_D) = \min_{\mu_G}\max_{\mu_D} \mathbb{E}_{\mathbf{x}\sim\mu_{\text{ref}}}[\log(D(\mathbf{x}))] + \mathbb{E}_{x\sim\mu_G}[\log(1 - D(\mathbf{x}))]. \tag{4.19}$$

This approach is often referred to as Vanilla GAN. According to [12], it is efficient in generating samples from high-dimensional data, which is generally cumbersome. However, it does not solve the above-mentioned problem (low dimensional manifolds) very well.

It can be shown that the optimal solution for a Vanilla GAN when fixing the generator, i.e., the optimal discriminator distribution when doing the maximizing step is

$$\mu_D^* \propto \frac{\mu_{\text{ref}}}{\mu_{\text{ref}} + \mu_G}. \tag{4.20}$$

This is easily proven, since

$$\begin{aligned} L(\mu_G, \mu_D) &= \int_{\mathbf{x}} \mu_{\text{ref}}(\mathbf{x}) \log(D(\mathbf{x}))\, \mathrm{d}\mathbf{x} + \int_{\mathbf{z}} p_z(\mathbf{z}) \log(1 - D(G(\mathbf{z})))\, \mathrm{d}\mathbf{z} \\ &= \int_{\mathbf{x}} \mu_{\text{ref}}(\mathbf{x}) \log(D(\mathbf{x})) + \mu_G(\mathbf{x}) \log(1 - D(\mathbf{x}))\, \mathrm{d}\mathbf{x}. \end{aligned} \tag{4.21}$$

Using this together with the fact that the function $y \mapsto a\log(y) + b\log(1-y)$ for $(a,b) \in \mathbb{R}^2 \backslash \{0,0\}$ has its maximum in $[0,1]$ at $\frac{a}{a+b}$, proves the result. This yields the following value for the loss function

$$\begin{aligned} C(\mu_D) =& \max_{\mu_D} L(\mu_G, \mu_D) \\ =& \mathbb{E}_{\mathbf{x}\sim\mu_{\text{ref}}}\Big[\log \frac{\mu_{\text{ref}}(\mathbf{x})}{\mu_{\text{ref}}(\mathbf{x}) + \mu_G(\mathbf{x})}\Big] + \mathbb{E}_{\mathbf{x}\sim\mu_G}\Big[\log \frac{\mu_G(\mathbf{x})}{\mu_{\text{ref}}(\mathbf{x}) + \mu_G(\mathbf{x})}\Big]. \end{aligned} \tag{4.22}$$

According to [12] it can be simplified to

$$C(G) = -\log(4) + KL\Big(\mu_{\text{ref}} || \frac{\mu_{\text{ref}} + \mu_G}{2}\Big) + KL\Big(\mu_G || \frac{\mu_{\text{ref}} + \mu_G}{2}\Big). \tag{4.23}$$

This is in turn can be written as

$$C(G) = -\log(4) + 2 \cdot JSD(\mu_{\text{ref}} || \mu_G) \tag{4.24}$$

where JSD stands for Jensen-Shannon divergence. However, this means that the process of finding the optimal solution deals with the distance between two distributions defined in the Jensen-Shannon divergence metric, which is connected to the Kullback-Leibler divergence, and therefore the problem regarding low dimensional manifolds is poorly addressed. A distance that handles this better is the Wasserstein distance.

4.3.2 Wasserstein GAN

Wasserstein GAN (WGAN) has a slightly different formulation. It is formulated using the Wasserstein distance between two distributions [11]. Using the same notation for distributions as in this chapter, the minimal Wasserstein-1 distance with respect to μ_G is formulated as

$$\min_{\mu_G} W_1(\mu_{\text{ref}}, \mu_G) = \min_{\mu_G} \sup_{\varphi \in \text{Lip}_1} \Big(\int \varphi(x)\, \mathrm{d}\mu_{\text{ref}}(x) - \int \varphi(x)\, \mathrm{d}\mu_G(x) \Big). \tag{4.25}$$

This is the loss function for Wasserstein GANs. With this design, the discriminator is not classifying, but instead for a fixed generator strategy, the optimal solution for the discriminator is the Lipschitz function maximizing the difference between the integrals. The integrals can be seen as expected values (as previously mentioned) and is therefore equivalently written as

$$\min_{\mu_G} W_1(\mu_{\text{ref}}, \mu_G) = \min_{\mu_G} \sup_{\varphi \in \text{Lip}_1} \Big(\mathbb{E}_{X\sim\mu_{\text{ref}}}[\varphi(X)] - \mathbb{E}_{X\sim\mu_G}[\varphi(X)] \Big). \tag{4.26}$$

Furthermore, since in this project we are conditioning on data, we introduce the concept of conditional WGAN in the next section. This is done both in Euclidean space and in signature space.

4.4 Conditional WGAN

In this section we assume that X and Z are defined in the usual manner considered in this thesis. This means that X is a stochastic process and we denote its Law as μ_X. Meanwhile, we consider the case of N observations of Z, at predetermined time instances, which has previously been written as $Z_{1:N} = (Z_1, Z_2, ...Z_N)$. We, in order to make it more readable, use the notation $\mathbf{Z}_N$ for $Z_{1:N}$. The quantity, $\mathbf{Z}_N$, has a joint distribution which we denote as $\mu_{\mathbf{Z}_N}$ and realizations of $\mathbf{Z}_N$ are denoted as $\mathbf{z}_N$. The goal is to learn the distribution of the path given a set of observations $\mathbf{z}_N$. This is achieved by training the conditional Law

$$\mu_X(\mathbf{z}_N) = \text{Law}(X|\mathbf{Z}_N = \mathbf{z}_N) \tag{4.27}$$

to replicate the true conditional Law denoted as $\mu_X^t(\mathbf{z}_N)$.

4.4.1 Euclidean space

In Euclidean space the Wasserstein distance, with the conditional Laws considered, is written as

$$W_1(\mu_X^t(\mathbf{z}_N), \mu_X(\mathbf{z}_N)) = \sup_{\varphi \in \text{Lip}_1} \mathbb{E}_{\mu_X^t(\mathbf{z}_N)}[\varphi(X)] - \mathbb{E}_{\mu_X(\mathbf{z}_N)}[\varphi(X)]. \tag{4.28}$$

Since $\mathbf{z}_N$ are realizations of random variables at specific time instances, we want to evaluate this over all such possible realizations, we therefore consider

$$\mathbb{E}_{\mathbf{Z}_N \sim \mu_{\mathbf{Z}_n}}[W_1(\mu_X^t(\mathbf{Z}_N), \mu_X(\mathbf{Z}_N))]. \tag{4.29}$$

As mentioned, when introducing the concept of GAN, μ_X is approximated by a neural network and we denote its parameters as $\gamma^{(g)}$. The general approach is to use a neural network for φ as well. This network is parameterised by $\gamma^{(d)}$. We then get the following loss function

$$l(\gamma^{(g)}, \gamma^{(d)}) = \mathbb{E}_{\mathbf{Z}_N \sim \mu_{\mathbf{Z}_N}} \Big[\mathbb{E}_{\mu_X^t(\mathbf{Z}_N)}[\varphi(X; \gamma^{(d)})] - \mathbb{E}_{\mu_X(\mathbf{Z}_N; \gamma^{(g)})}[\varphi(X; \gamma^{(d)})] \Big]. \tag{4.30}$$

This would lead to the following min-max problem for the conditional WGAN.

$$\min_{\gamma^{(g)}} \max_{\gamma^{(d)}} l(\gamma^{(g)}, \gamma^{(d)}). \tag{4.31}$$

There are several reasons for not using this approach. Firstly, it is problematic having two networks competing with each other, and there can be slow convergence for the optimization problem. Secondly, Wasserstein GAN has, in some settings, a hard time capturing temporal dependence of joint probability distributions, according to [1]. This is important as we are considering entire paths and not just marginal distributions. Thirdly, it is cumbersome to enforce a neural network to be a Lipschitz function. However, there exists several approaches to address this, for example gradient penalty, weight clipping, or more advanced approaches such as spectral normalization found in [13] and Lipschitz constant constrainer found in [14]. Lastly, even if it possible to enforce Lipschitz continuity, the network might not cover the entire space of Lipschitz functions.

4.4.2 Signature space

With these problems in mind, we consider another metric which has previously been introduced - the signature Wasserstein-1 (Sig-W_1) metric. As previously seen, using the notation of μ and ν for distributions, it is defined as

$$W_1^{\text{Sig space}}(\mu, \nu) = \sup_{||\mathbf{f}||_{\text{Lip}} \leq 1} \mathbb{E}_{S\sim\mu}[\mathbf{f}(S)] - \mathbb{E}_{S\sim\nu}[\mathbf{f}(S)]. \tag{4.32}$$

In Section 3.3 it is shown it can be approximated as

$$\text{Sig-}W_1(\mu, \nu) = ||\mathbb{E}_{X\sim\mu}[S(X)] - \mathbb{E}_{X\sim\nu}[S(X)]||_p \tag{4.33}$$

and by considering the truncated signature it reads

$$\text{Sig-}W_1(\mu, \nu) \approx \text{Sig-}W_1^m(\mu, \nu) = ||\mathbb{E}_{X\sim\mu}[S^m(X)] - \mathbb{E}_{X\sim\nu}[S^m(X)]||_p. \tag{4.34}$$

Changing back to the notation for conditional distributions, we get the following loss function for a conditional WGAN in signature space

$$\begin{aligned} &\mathbb{E}_{\mathbf{Z}_N\sim\mu_{\mathbf{Z_N}}}\Big[\text{Sig-}W_1^m(\mu_X^t(\mathbf{Z}_N), \mu_X(\mathbf{Z}_N;\gamma))\Big] \\ &\quad = \mathbb{E}_{\mathbf{Z}_N\sim\mu_{\mathbf{Z}_N}}\Big[||\mathbb{E}_{X\sim\mu_X^t(\mathbf{Z}_N)}[S^m(X)] - \mathbb{E}_{X\sim\mu_X(\mathbf{Z}_N;\gamma)}[S^m(X)]||_p\Big]. \end{aligned} \tag{4.35}$$

The difference in the signature space is that we only have one network to train, which is parameterised by γ. This means that the problem with slow convergence coming from two networks competing against each other is eliminated.

To evaluate the expression in (4.35) we need to, in some way, evaluate true conditional expected values. In some settings it is possible to analytically derive the conditional model that generated the data; one such example is when the data is generated by a Brownian motion (derivation of this is found in Section 2.3.3). However, if the model is assumed to be unknown (as in this project), or it is impossible to derive it analytically, another way of evaluating this needs to be considered. The approach considered in this report is presented in the following section.

4.5 Approximation of conditional expected value by a neural network

We consider the setting where we have a random variable X, such that $\mathbb{E}[X^2] < \infty$, and Y is $\mathcal{F}$-measurable with $E[Y^2] < \infty$. Then we have the following relation (which is proved in [15])

$$\mathbb{E}[(X - Y)^2] \geq \mathbb{E}[(X - \mathbb{E}[X|\mathcal{F}])^2], \tag{4.36}$$

where equality is achieved if and only if $Y = \mathbb{E}[X|\mathcal{F}]$. The interpretation of this is that the conditional expectation is the orthogonal projection of X to the space of squared integrable random variables on the probability space with $\mathcal{F}$ as σ-algebra.

This means the conditional expectation can be obtained by considering the left hand side as a minimization problem, i.e.,

$$\min_{\text{g is measurable}} \mathbb{E}[(X - g(Y))^2] = \mathbb{E}[(X - \mathbb{E}[X|Y])^2] \tag{4.37}$$

The left hand side is considered as the loss function for the neural network $\Phi(Y)$ which is trained to replicate $g(Y)$. In our setting, since we are approximating the true signature conditioned on $\mathbf{z}_N$, $\Phi(\mathbf{z}_N)$ is trained by the loss function

$$\frac{1}{N}\sum_{i=1}^{N} ||S^m(X^{(i)}) - \Phi(\mathbf{z}_N^{(i)})||^2, \tag{4.38}$$

where N is the sample size used to approximate the expected value using the Monte Carlo method.

5

Investigations for conditional Brownian motion

In this section we present and discuss the implementation of the neural SDE and all intermediate results leading up to the final design. This highlights troubles that were encountered during the process and motivates choices made along the way.

5.1 Summary of the problem

Remember, we assume the data are generated by the following model

$$\begin{aligned} \mathrm{d}X_t &= \mu(X_t, t)\,\mathrm{d}t + \sigma(X_t, t)\,\mathrm{d}W_t \quad t \in [0, T],\ X_0 = x_0 \\ Z_n &= X_{t_n} + V_n, \quad n = 1, ..., N. \end{aligned} \tag{5.1}$$

Here $V_n \sim \mathcal{N}(0, \sigma_Z^2)$ is the measurement noise. The goal is, given a set of observations $\mathbf{z}_N$ of the random variable $\mathbf{Z}_N$, to train a neural SDE that can generate paths that approximate the Law $\mu_X^t(\mathbf{z}_N)$. The neural SDE is defined by

$$\begin{aligned} \mathrm{d}\tilde{X}_t &= \tilde{\mu}_{\gamma_i}(t, \tilde{X}_t, z_{i:N})\,\mathrm{d}t + \tilde{\sigma}_{\gamma_i}(t, \tilde{X}_t, z_{i:N})\,\mathrm{d}W_t, \quad t \in (t_{i-1}, t_i],\ i = 1, 2, ..., N \\ \tilde{X}_0 &= x_0 \end{aligned} \tag{5.2}$$

and the Law of its generated paths is denoted as $\mu_X(\mathbf{z}_N; \gamma_i)$. More specifically, the neural SDE is represented by the two sets of networks $\tilde{\mu}_{\gamma_i}$ and $\tilde{\sigma}_{\gamma_i}$, corresponding to the drift and diffusion networks in each interval $(t_{i-1}, t_i]$, respectively. The reasoning for the notation $\tilde{X}_t$ is to show that it is a new process, conditioned on the realizations $\mathbf{z}_N$. This means that when the notation $\tilde{X}_t$ is used in the text, it should be seen as paths sampled from the neural SDE.

The neural SDE can in theory handle an arbitrary amount of future observations due to the different parameterization γ_i for each possible number of future observations. However, due to computational reasons, in this project we assume that only a fixed number of future observations have an impact. This means that only one network for the drift and diffusion each needs to be implemented. Henceforth, these are denoted to be parameterized by γ and the Law of its generated paths as $\mu_X(\mathbf{z}_N; \gamma)$. For the first investigations in Section 5.6, only one future observation is considered. The model is extended to handle four future observations in Section 5.6.2.5.

According to (2.31) only the drift term, if the diffusion term is assumed to be known, needs to be modelled as a neural network. However, in the case of no noise and a larger time step, the dynamics can be extreme and numerical approximations can be bad, which is discussed in Section 5.5. By modelling σ as a neural network, the neural SDE is allowed more flexibility to describe the dynamics. The conclusion is therefore to always model σ as a neural network.

The networks are trained using the Sig-W_1^m metric defined in (4.35). This means we have the following loss function

$$\gamma \mapsto \mathbb{E}_{\mathbf{z}_N \sim \mu_{\mathbf{z}_N}} \Big[||\Phi(\mathbf{z}_N) - \mathbb{E}_{X \sim \mu_X(\mathbf{z}_N;\gamma)}[S^m(\tilde{X})]||_p \Big]. \tag{5.3}$$

Here $\Phi(\mathbf{z}_N)$ is trained beforehand (the true conditional expected signature) as described in Section 4.5. We make the following Monte Carlo approximation for the outer expectation

$$\frac{1}{N} \sum_{j=1}^{N} ||\Phi(\mathbf{z}_N^{(j)}) - \mathbb{E}_{\tilde{X} \sim \mu_X(\mathbf{z}_N^{(j)};\gamma)}[S^m(\tilde{X})]||_p \tag{5.4}$$

where N stands for the number of samples, henceforth referred to as the **batch size** and m is the order of the truncation considered. The inner expectation is also approximated using the Monte Carlo method and more specifically as

$$\mathbb{E}_{\tilde{X} \sim \mu_X(\mathbf{z}_N^{(j)};\gamma)}[S^m(\tilde{X})] \approx \frac{1}{M} \sum_{i=1}^{M} S^m(\tilde{X}^{(i,j)}). \tag{5.5}$$

Henceforth, M is referred to as the number of **Monte Carlo samples**.

Using the above, we end up with the loss function for training our neural SDE

$$\gamma \mapsto \frac{1}{N} \sum_{j=1}^{N} ||\Phi(\mathbf{z}_N^{(j)}) - \frac{1}{M} \sum_{i=1}^{M} S^m(\tilde{X}^{(i,j)})||_p. \tag{5.6}$$

From the derivation of the Wasserstein distance in the signature space, we have the parameter p, which could be arbitrarily chosen, at least such that $p > 1$. However, in this project we do not focus on finding the best p, but we have simply chosen the most natural one $p = 2$.

5.2 Programming language and packages

We are using Python for all implementations. Python is known to not perform at par with low-level programming languages such as C. However, since most of the computational resources used are in existing packages which has been written in C, this is not problematic. The primary packages that are used are PyTorch and signatory. PyTorch is a machine learning package that utilizes CUDA, which enables the possibility of making fast parallel computations on the GPU. It is user-friendly and contains a considerable amount of already implemented network architecture. The other crucial package, signatory, utilizes fast computation on the GPU for calculating the (truncated) signature transformation of paths being piecewise linear interpolations of discrete time series.

5.3 General network architecture

All networks, if nothing else is mentioned, are parameterised as MLPs, possibly with different number of layers and neurons in each layer. Each network could have different input size, depending on which inputs are deemed necessary and how the dependency of $\mathbf{z}_N$ is implemented. This dependency is discussed in Section 5.6. The chosen activation function for the MLPs is the rectified linear unit (**ReLU**) activation function which is defined as

$$\mathrm{ReLU}(x) = \max(0, x). \tag{5.7}$$

ReLU is a popular choice, mostly due to its efficient computation, sparse activation and scale-invariancy [16]. It also performed well in this project, which is why no other activation function is used.

All networks are trained using the ADAM-optimizer, utilizing mini-batches and adaptive learning rates. This means the training is not as prone to choosing a bad learning rate and therefore avoids local optimas, which was discussed in Section 4.2. The hyper-parameters are set to the standard values presented in that section. If nothing else is mentioned, a mini-batch size of 64 is always used.

5.4 Generating data

The first step of training the neural SDE is to generate the training data. This is done by sampling from an SDE with the Euler–Maruyama method, introduced in Section 2.2. Notice that the noise level for observations and time step size are easily adjustable.

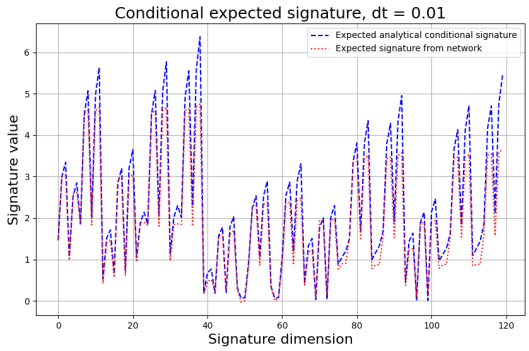
5.5 Implementation of the true conditional expected signature

The neural SDE should replicate the true distribution by evaluating signatures. This means that the network Φ for estimating the true conditional expected signature needs to perform well, i.e., it needs to be expressive enough to represent signatures. The chosen approach is to make Φ adaptive to the dimension of the signature. More specifically, it is built using three layers, each with size $3 \cdot \dim(S^m(\tilde{X}))$. If nothing else is specified, a truncation of degree four is used ($m = 4$). This means that, as each path is augmented with time and quadratic variation, each layer consists of 360 neurons. The network is trained with the loss function described in (4.38).

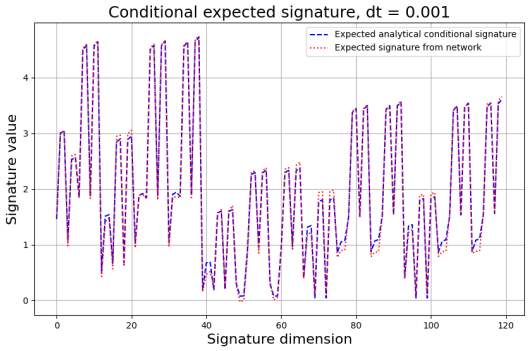
It is difficult to quantify how well this network performs, since we generally cannot sample from the true conditional distribution. However, as derived in Section 2.3.3, in the case when the underlying SDE is that of a Brownian motion, we can sample from the conditional SDE. By sampling from this SDE, we can estimate the conditional expected signature with the Monte Carlo method. In this section, Brownian

motions in the time range [0,3] and with 7 observations (every half time unit) are used.

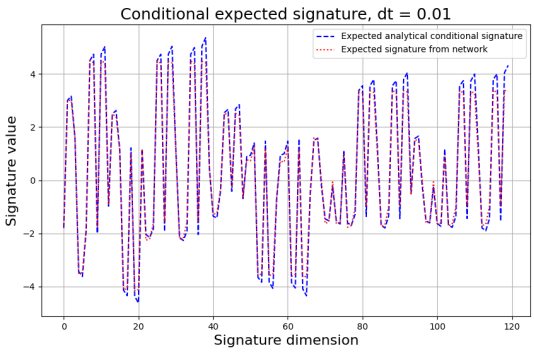
In Table 5.1 the average distance (ℓ^2 distance) between the output of the network and the expected signature from conditionally generated paths for two different time steps and two different noise levels are shown. This is done relative to the ℓ^2 norm of the expected signature from conditionally generated paths. To visualize these results we can, given a specific series of observations, plot the output of the network and the expected signature of conditionally generated paths. This is done in Figure 5.1. Note that the network Φ is always trained using paths sampled with the larger time step, however, it is possible to use Φ with observations from a path sampled with a smaller time step, given that the observations occur at the same time instances.



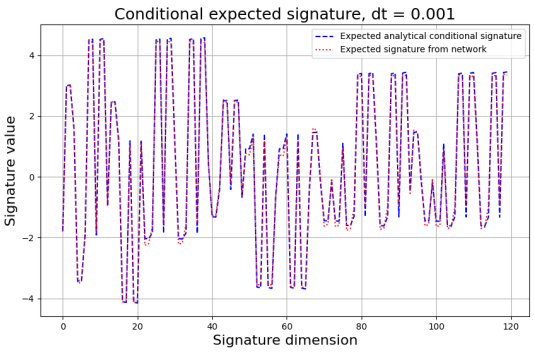
(a) Noise level $\sigma_Z = 0$, time step $\mathrm{d}t = 0.01$.



(b) Noise level $\sigma_Z = 0$, time step $\mathrm{d}t = 0.001$.



(c) Noise level $\sigma_Z = 0.2$, time step $\mathrm{d}t = 0.01$.



(d) Noise level $\sigma_Z = 0.2$, time step $\mathrm{d}t = 0.001$.

Figure 5.1: Comparison of the conditional expected signature approximated using the network Φ and using conditionally generated paths.

Table 5.1: Table showing the average relative distance in the ℓ^2-norm (and its standard deviation) between the output of Φ and the expected signature of conditionally generated paths. This is done for two noise levels and two time steps. The underlying paths are Brownian motions in the time interval [0,3] with 7 observations. 100 samples are used for computing the average. All values are multiplied with a factor 10^4.

l^2 distance	$\mathrm{d}t = 0.01$	$\mathrm{d}t = 0.001$
$\sigma_Z = 0$	3.2 (1.6)	0.44 (1.0)
$\sigma_Z = 0.2$	0.99 (0.65)	0.38 (0.56)

There is a clear difference in performance when comparing the different time steps. In the case of the smaller time step $\mathrm{d}t = 0.001$, the output from the network and the expected signature from conditionally generated paths are similar, while in the case of the larger time step $\mathrm{d}t = 0.01$, they are not as similar. The reason it differs is due to the approximation errors of the quadratic variation, which is discussed below.

The network Φ is trained using unconditional paths, which on average obtains the correct quadratic variation (three in this case). The generated conditional paths consistently acquire bad approximations of it, due to numerical errors. The size of these errors are shown in Table 5.2. The extreme dynamics of the conditional paths mean they obtain almost singular drift terms near observations. When the path closes into an observation, the drift goes to infinity (in mean-reverting sense), while the diffusion term is constant. The extreme drift term leads to the Euler–Maruyama method being a bad numerical approximation, and in turn the quadratic variation is badly approximated. When adding noise, the drift does not become singular, although still extreme, which leads to the approximation not being as bad. Since the quadratic variation is on average correct when training Φ, it is reasonable to assume that the output of this network is more correct than what is obtained from the generated conditional paths.

Table 5.2: Table showing the average quadratic variation (and standard deviation) for two noise levels and two time step sizes. The quadratic variation is estimated by sampling both unconditional and conditional paths. The underlying paths are Brownian motions in the time interval [0,3] with 7 observations. 100 samples are used for the computations.

Quadratic Variation	$\sigma_Z = 0$		$\sigma_Z = 0.2$	
	$\mathrm{dt} = 0.01$	$\mathrm{dt} = 0.001$	$\mathrm{dt} = 0.01$	$\mathrm{dt} = 0.001$
Unconditional Paths	3.00 (0.24)	3.00 (0.07)	3.00 (0.24)	3.00 (0.08)
Conditional Paths	3.32 (0.27)	3.05 (0.08)	3.15 (0.26)	3.02 (0.08)

5.6 Implementation of the neural SDE

To be able to sample paths from the neural SDE an initial value is needed. In this project, the initial condition is assumed to be known. The networks for the

drift and diffusion terms need to be explanatory enough to be able to represent the structures of a conditional SDE. They have both been constructed using six layers with 60 neurons in each. This has shown to be large enough to learn desired patterns.

The neural SDE is trained using the loss function in (5.6), with a varying amount of Monte Carlo samples. With a low amount of Monte Carlo samples, the variability in the approximated expected signature is higher, which could help avoiding local optimas. Therefore, by incorporating a training scheme with different Monte Carlo sample sizes, it is possible to in a more consistent way avoid getting stuck in local optimas. The scheme makes use of Monte Carlo sample sizes in the range of 16 to 2048. During training, a time step size of 0.01 is used. Training with a smaller time step is not possible in this project due to the computational load.

The purpose of the neural SDE is to replicate the conditional distribution given a series of observations. Naturally, one of the main problems of implementing the neural SDE is how to incorporate the dependency of the observations. Several approaches to this have been tried. Before describing our final implementation and evaluating its performance, approaches considered during the project are next discussed.

5.6.1 Initial approach for handling observations

The initial plan was to make use of an LSTM network to handle the observations. This is a special type of recursive neural network that updates its so called "hidden state" at the time of each new observation [17]. It would be able to handle an arbitrary number of observations, since the number of parameters in the network is not affected by adding more observations. The idea was to use a bidirectional LSTM such that the neural SDE can use information both from the past and future. More technically, at time steps between observation Z_i and Z_{i+1}, $\tilde{\mu}_\gamma$ and $\tilde{\sigma}_\gamma$ would take the i:th hidden state from the forward LSTM (all past information) and the $i+1$:th hidden state from the backward LSTM (all future information) as input. This approach is quite intuitive and is illustrated in Figure 5.2. However, training the neural SDE, when constructed this way, proved to be difficult. The exact reason is though not clear; one possible explanation being that an LSTM cannot represent all the characteristics of the observations needed. We have chosen not to add results from this approach since they are deemed too poor.

5.6.2 Using the observations directly

As the initial approach failed, the neural SDE was simplified to use the observations directly as input. Three approaches are presented and discussed in the following (sub-)sections.

In the development of the following approaches, Brownian motions

$$\mathrm{d}X_t = \mathrm{d}W_t, \quad t \in [0,1]; \tag{5.8}$$

$$Z_1 = X_1 + \mathcal{N}(0, \sigma_Z^2), \tag{5.9}$$

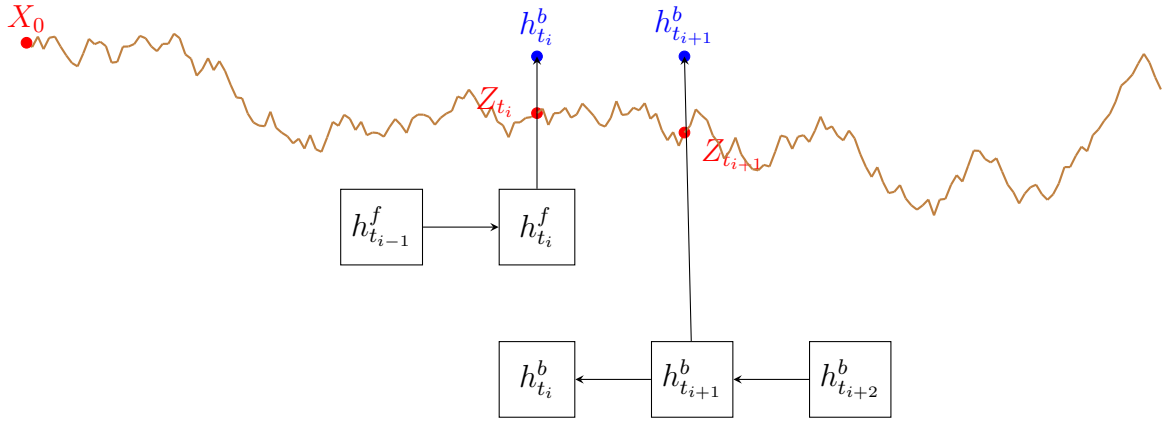


Figure 5.2: The concept of the Neural SDE using LSTM networks for observations visualized.

are used as training example. The analytical conditional SDE can be derived in this case. In this derivation, we have two settings to consider: observations with and without noise. In the case without measurement noise, the only dependency of the drift term is the current state and the successive observation. However, in the case with measurement noise, the dependency is on all future observations.

The drift term for the neural SDE to learn in this first test case ($t_1 = 1$) is

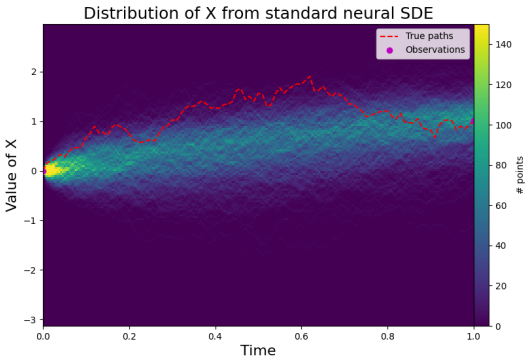
$$\bar{X}_t = \frac{z_1 - X_t}{\sigma_Z^2 + t_1 - t}. \tag{5.10}$$

The term is taken from (2.43).

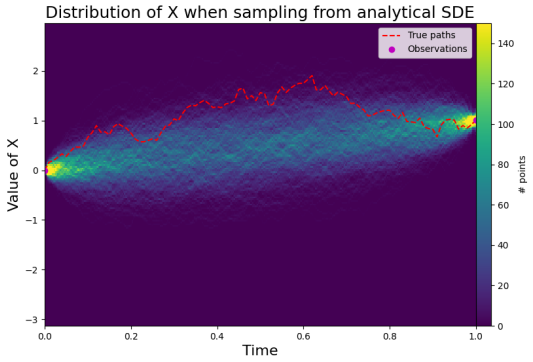
5.6.2.1 Standard approach

The first approach, which we refer to as the *standard* approach, is implemented with the drift and the diffusion networks receiving the tuple (X_t, t, z_1, t_1) as input. Notice that σ_Z is not given as input. Optimally, the drift network should learn to combine the inputs to replicate the drift coefficient in (5.10). Remember that the drift term becomes singular (in a mean-reverting sense) at $t = t_1$ if $\sigma_Z = 0$. This means that all paths should go exactly through z_1, but, due to the discretization, this is not the case.

The distribution through time, in the case of no noise, when sampling from the neural SDE and when sampling from the analytical conditional SDE, can be seen in Figure 5.3. In the figures, the true underlying path that generated the observation is also visualized. This path is, however, unknown for both the neural SDE and the analytical conditional SDE. The resulting distributions when adding measurement noise ($\sigma_Z = 0.2$) can be seen in Figure 5.4.

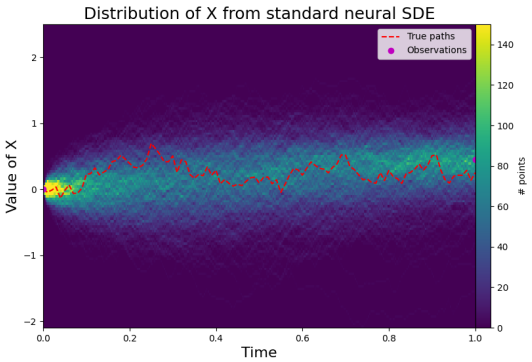


(a) Distribution of generated paths from neural SDE when trained with input tuple (X_t, t, z_1, t_1).

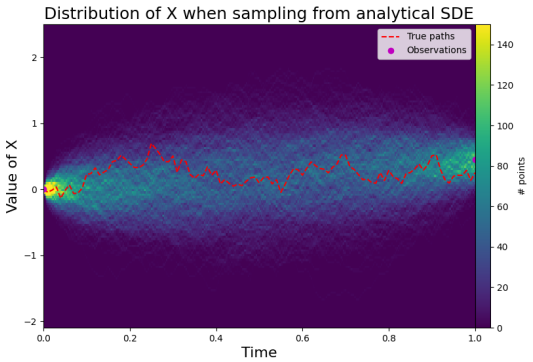


(b) Distribution of generated paths from analytical conditional SDE when conditioning on z_1.

Figure 5.3: Distribution of neural SDE and analytical conditional SDE when sampling with time step $\mathrm{d}t = 0.01$. No measurement noise.



(a) Distribution of generated paths from neural SDE when trained with input tuple (X_t, t, z_1, t_1).



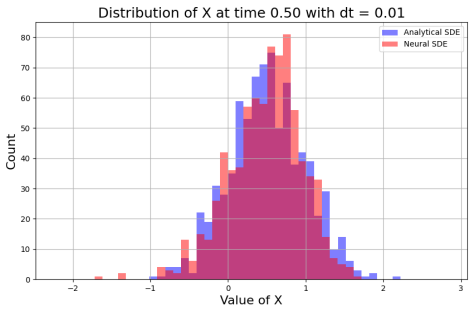
(b) Distribution of generated paths from analytical conditional SDE when conditioning on z_1.

Figure 5.4: Distribution of neural SDE and analytical conditional SDE when sampling with time step $\mathrm{d}t = 0.01$. Measurement noise $\sigma_Z = 0.2$.

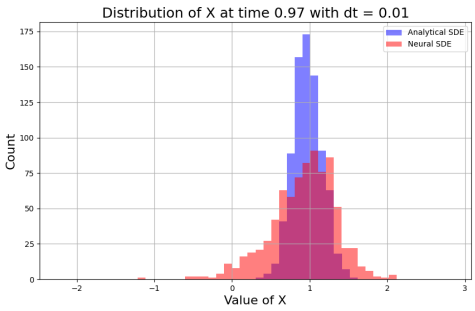
We can see that the neural SDE does not learn to shrink the distribution towards the observation at $t = 1$ in the case with no noise. The case when measurement noise is added, should be easier to solve, since measurement noise has a regularizing effect on the drift term. However, the neural SDE does not show any tendency to shrink the distribution in this case either. To visualize this better, the distributions in the case of no noise, at two different times $t = 0.5$ and $t = 0.97$, is shown in Figure 5.5. From this it is clear that the neural SDEs has difficulties narrowing the distribution.

The reason for why the neural SDE does not perform as expected is examined by the following approaches considered. An initial guess is that the optimizer gets

stuck in a local optima. To avoid this, a possible solution is to consider giving the neural SDE help during training, such that it finds the correct solution.



(a) Distribution at $t = 0.5$.



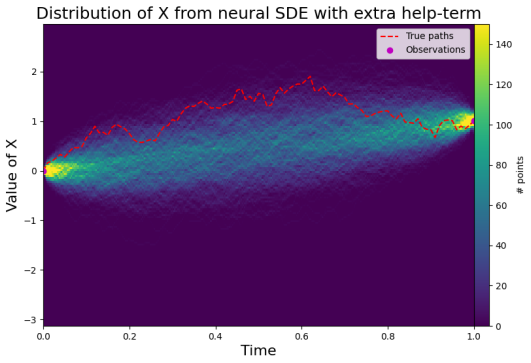
(b) Distribution at $t = 0.97$.

Figure 5.5: Distribution at two different times of the standard neural SDE and the analytical conditional SDE when sampling with time step $\mathrm{d}t = 0.01$. Measurement noise $\sigma_Z = 0.0$.

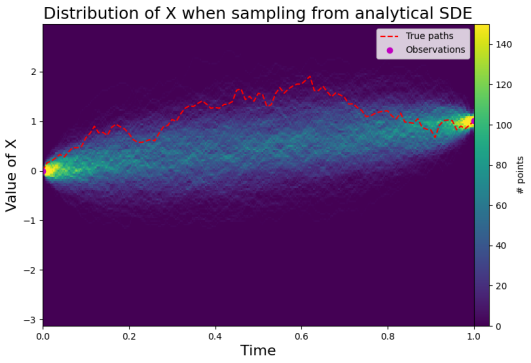
5.6.2.2 Help term approach

Given the result from the *standard* approach, a natural approach would be to give the analytically calculated drift term as input to the neural SDE. In other words, the tuple $(X_t, t, z_1, t_1, \bar{X}_t)$ where $\bar{X}_t$ is calculated as in (5.10), is used as input to the neural SDE. This approach will be called the *help term* approach. The training procedure is the same as for the *standard* approach.

The distribution through time, in the case of no noise, when sampling from the neural SDE with the help term and when sampling from the analytical conditional SDE, can be seen in Figure 5.6. The resulting distributions when adding measurement noise ($\sigma_Z = 0.2$) can be seen in Figure 5.7.

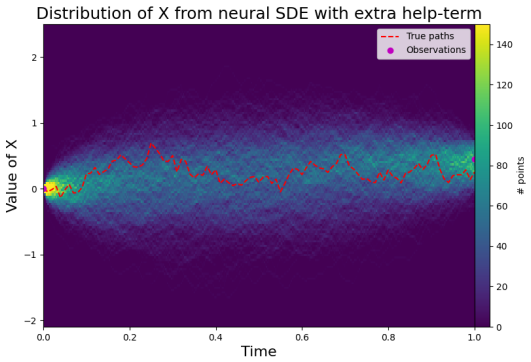


(a) Distribution of generated paths from neural SDE when trained with input tuple $(X_t, t, z_1, t_1, \bar{X}_t)$.

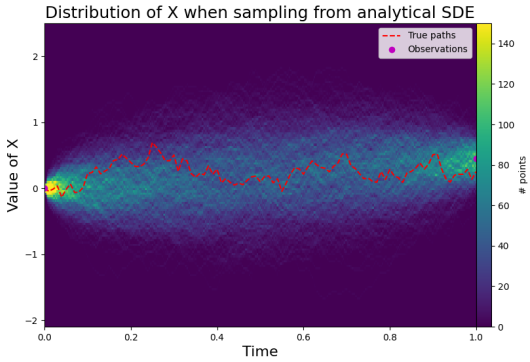


(b) Distribution of generated paths from analytical conditional SDE when conditioning on z_1.

Figure 5.6: Distribution of neural SDE and analytical conditional SDE when sampling with time step $\mathrm{d}t = 0.01$. No measurement noise.

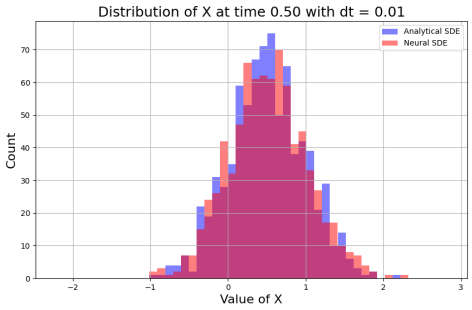


(a) Distribution of generated paths from neural SDE when trained with input tuple $(X_t, t, Z_1, t_1, \bar{X}_t)$.

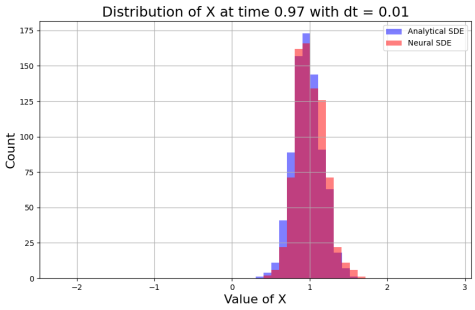


(b) Distribution of generated paths from analytical conditional SDE when conditioning on z_1.

Figure 5.7: Distribution of neural SDE and analytical conditional SDE when sampling with time step $\mathrm{d}t = 0.01$. Measurement noise $\sigma_Z = 0.2$.



(a) Distribution at $t = 0.5$.



(b) Distribution at $t = 0.97$.

Figure 5.8: Distribution at two different times of the neural SDE with the extra help term and the analytical conditional SDE when sampling with time step $\mathrm{d}t = 0.01$. Measurement noise $\sigma_Z = 0.0$.

The distributions are very similar, both with and without noise. This suggests that the neural SDE prefers to squeeze the distribution (achieves lower loss), since it otherwise would ignore the input $\bar{X}$ and perform as in the *standard* approach. To better visualize the narrowing of the distribution, two histograms at times $t = 0.5$ and $t = 0.97$ are shown in Figure 5.8. These results indicate that the problem either lies within the optimization (meaning that the neural SDE gets stuck in a local optima) or that the network cannot represent the structure of the conditional drift term.

Notice that the *help term* approach is, however, not valid as final implementation. First of all, we assume that the measurement noise is known which generally is not true. We also assume that all paths are generated from a Brownian motion and only has one observation in the end, which means that the performance of the neural SDE

in another setting, could be poor. An approach that helps the training of the neural SDE, without these assumptions, is introduced next.

5.6.2.3 Pre-trained approach

The third approach considered, called the *pre-trained* approach, is to train the drift network beforehand to replicate the analytical drift term of a Brownian motion conditioned on measurements. The idea is, that by providing a good initial state for the parameters (of the drift network), the neural SDE can be adequately trained against other SDEs without getting stuck in local optimas.

The pre-training of the drift term, $\tilde{\mu}_\gamma$, is performed by minimizing the expression

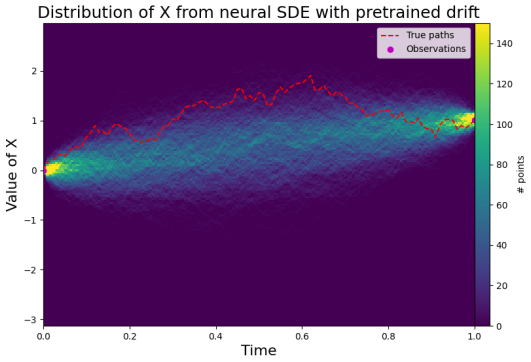
$$\gamma \mapsto \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N_t} \left(\tilde{\mu}_\gamma(X_{t_j}^i, t_j^i, z_1^i, t_1) - \frac{z_1^i - X_{t_j}^i}{\sigma_Z^2 + t_1 - t_j^i} \right)^2 . \tag{5.11}$$

Here N is the batch size and N_t is number of time instances considered. During this training, a value of $\sigma_Z = 0.1$ is used, even though the correct value in this context is either $\sigma_Z = 0.0$ or $\sigma_Z = 0.2$. The reason for using an approximately correct σ_Z is that in practical applications, it is often possible to achieve a reasonable approximation of this value.

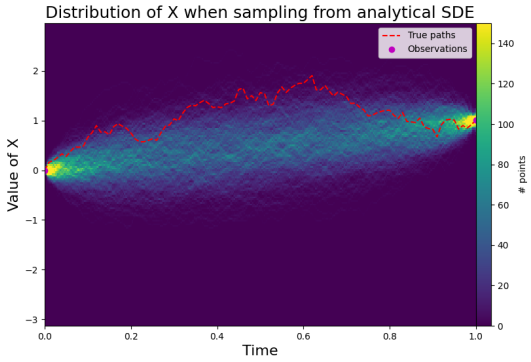
After having pre-trained $\tilde{\mu}_\gamma$, the neural SDE is trained using the same procedure as the two previous approaches. The hope is that, despite being pre-trained with an incorrect σ_Z, the neural SDE will able to learn the correct dynamics without getting stuck in local optimas.

The reason for not pre-training $\tilde{\sigma}$ is because it can easily learn the correct one due to augmenting with the quadratic variation. Additionally, we do not know the desired structure of $\tilde{\sigma}$ in each setting, due to numerical simulation errors (which has been discussed).

The distribution, in the case of no noise, when sampling from the neural SDE with a pre-trained drift and when sampling from the analytical conditional SDE, can be seen in Figure 5.9. To better visualize that the pre-trained neural SDE squeezes the distribution, the distribution at times $t = 0.5$ and $t = 0.97$ are shown in Figure 5.11. The resulting distribution when adding measurement noise ($\sigma_Z = 0.2$) can be seen in Figure 5.10. The *pre-trained* neural SDE replicates the desired distribution in a similar manner as the *help term* approach.

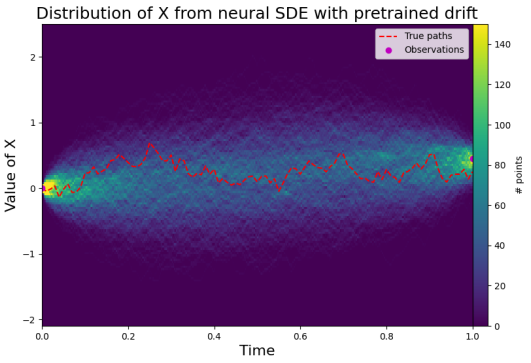


(a) Distribution of generated paths from neural SDE when drift-network is pre-trained.

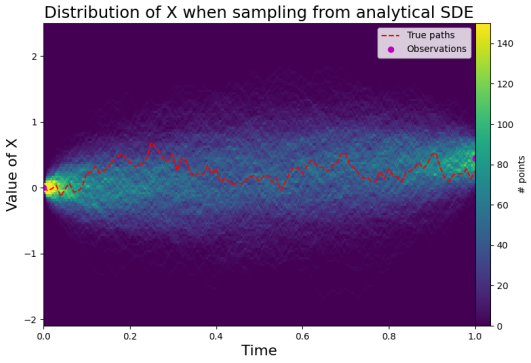


(b) Distribution of generated paths from analytical conditional SDE when conditioning on z_1.

Figure 5.9: Distribution of neural SDE and analytical conditional SDE when sampling with time step $\mathrm{d}t = 0.01$. No measurement noise.

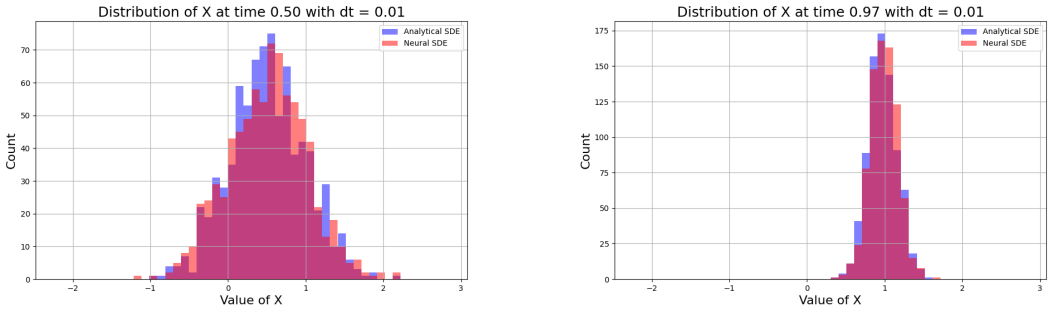


(a) Distribution of generated paths from neural SDE when drift-network is pre-trained.



(b) Distribution of generated paths from analytical conditional SDE when conditioning on Z_1.

Figure 5.10: Distribution of neural SDE and analytical conditional SDE when sampling with time step $\mathrm{d}t = 0.01$. Measurement noise $\sigma_Z = 0.2$.



(a) Distribution at $t = 0.5$. **(b)** Distribution at $t = 0.97$.

Figure 5.11: Distribution at two different times of the neural SDE with the extra help term and the analytical conditional SDE when sampling with time step $\mathrm{d}t = 0.01$. Measurement noise $\sigma_Z = 0.0$.

5.6.2.4 Comparing the approaches

We also compare the three approaches quantitatively. This is done by evaluating the average test loss for each approach. We also compare them against paths sampled from the analytical conditional SDE.

The average test loss in the four cases with a time step $\mathrm{d}t = 0.01$ are visualized in Figure 5.12. As previously mentioned we can sample paths from the neural SDEs with a smaller time step (if observations occur at the same time instances). We therefore also consider the average test loss when paths are sampled with $\mathrm{d}t = 0.001$. The result from this can be seen in Figure 5.13.

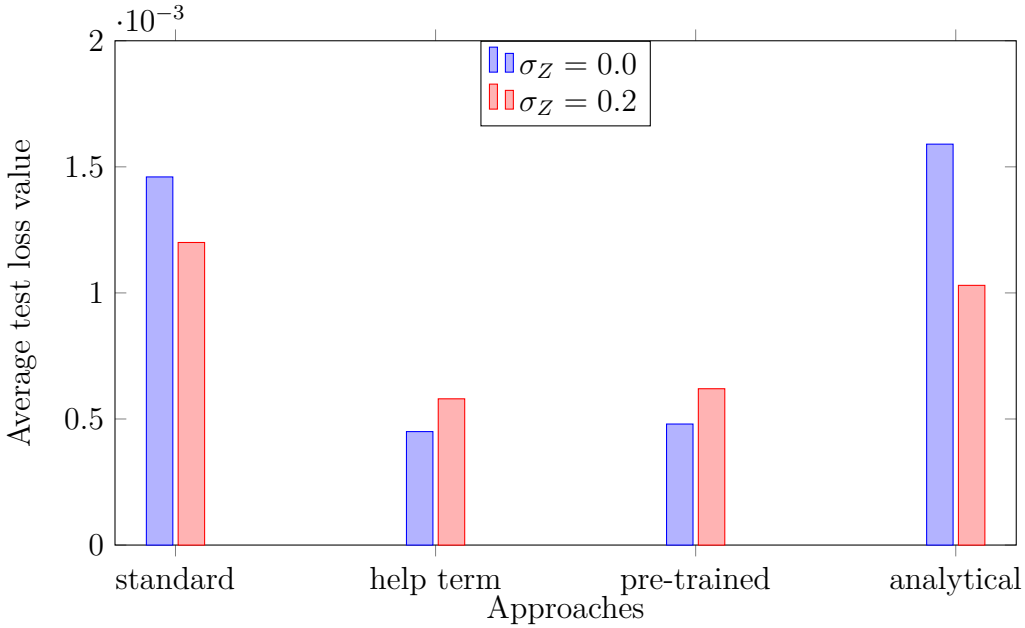


Figure 5.12: Average test losses of the three approaches for the neural SDE and for analytically generated conditional paths. Time step size $\mathrm{d}t = 0.01$.

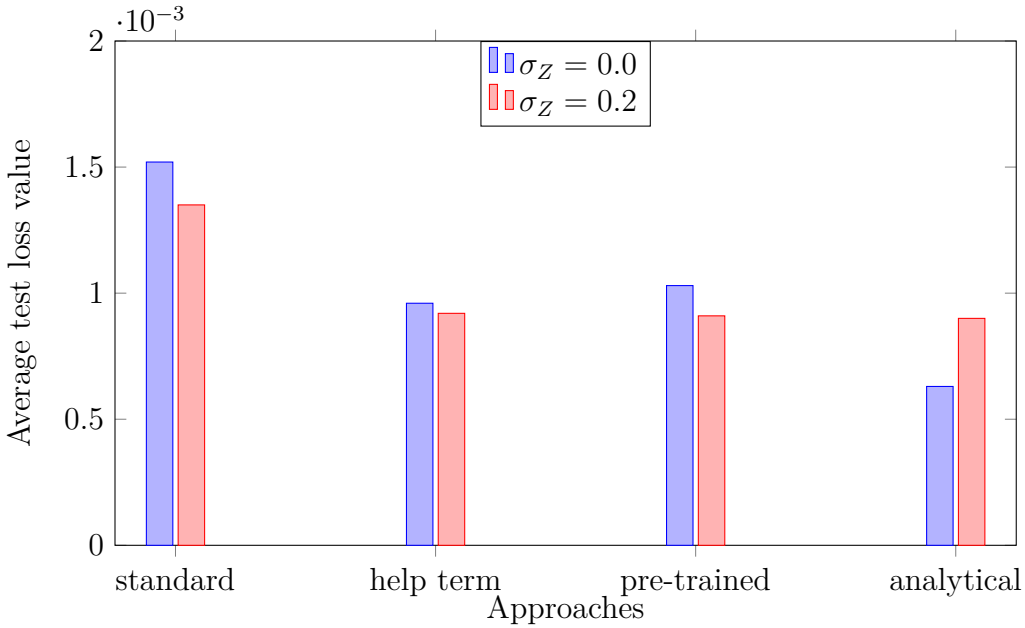
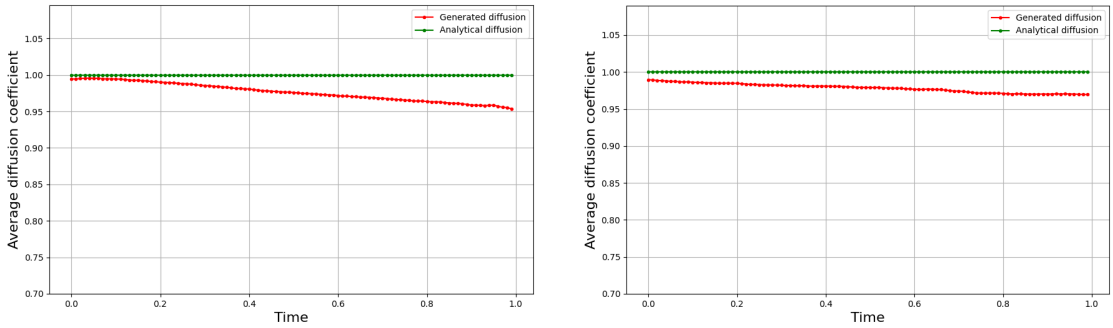


Figure 5.13: Average test losses of the three approaches for the neural SDE and for analytically generated conditional paths. Time step size $\mathrm{d}t = 0.001$.

In the case of the time step $\mathrm{d}t = 0.01$, it is clear that the *help term* and *pre-trained* approaches have a lower test loss than the *standard* approach. This strengthens the idea that the optimizer cannot find the global optima without help.

Interestingly, the case when paths are generated from the analytical conditional SDE, and the time step size is 0.01, results in the highest loss (almost in the case of noise as well). As discussed in Section 5.5, this is most likely due to numerical errors in the approximation of the quadratic variation, since analytically generated conditional paths consistently obtain a poor approximation of the quadratic variation. The poor approximation is due to the Euler–Maruyama method being used with a too large time step in relation to the drift values. The neural SDE aims to obtain the correct quadratic variation. It achieves this by slightly lowering the value of $\tilde{\sigma}_\gamma$ component. This behaviour is visualized in Figure 5.14, both for the cases with and without noise. This leads to, according to the loss function, a better approximation. The conclusion of this analysis is that it is smart to, even if σ is assumed to be known, parameterize the diffusion coefficient as a neural network such that it can adapt according to what is best for replicating the data for a specific setting.

In the setting where $\mathrm{d}t = 0.001$, the analytical conditional SDE obtains the lowest test loss. In this scenario, it does not suffer as much from the numerical errors. At the same time, the neural SDEs have not been trained with this time step size, meaning that they do not perform as well as when $\mathrm{d}t = 0.01$. It is, however, still clear that the neural SDE performs better when receiving help in training.



(a) No measurement noise **(b)** Measurement noise $\sigma_Z = 0.2$

Figure 5.14: Average diffusion value of pre-trained neural SDE and analytical solution for two different noise levels. Time step size $\mathrm{d}t = 0.01$.

The conclusion of the attempted approaches is that the *pre-trained* approach, due to its good performance and flexibility, is the best one. Before choosing this as our final implementation, the example is extended to more than one observation.

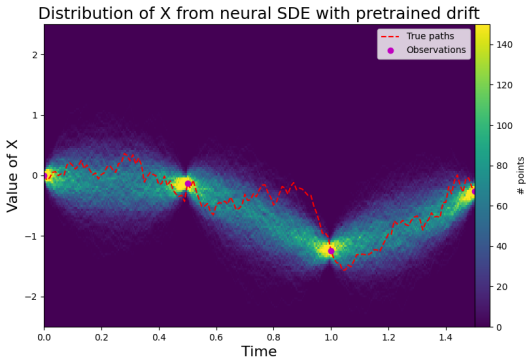
5.6.2.5 More than one observation

Extending the neural SDE to handle more than one observation requires modifications. First of all, since the drift term at any time $t \in (t_{i-1}, t_i]$ depends on all future observations $z_{i:N}$ (unless $\sigma_Z = 0$), the neural SDE would need different sized inputs in each interval. In other words, several drift and diffusion networks need to be implemented. As mentioned earlier, due to time limitations of the project, it was decided to do a simplified version of this; only one network for each term (one for the drift coefficient and one for the diffusion coefficient), that always takes a fixed number of future observations. This is a reasonable approach due to the fact that the most vital information for simulating the next value of a path lies in the closest (future) observations. We have chosen to always consider four future observations. The neural SDE can be written as

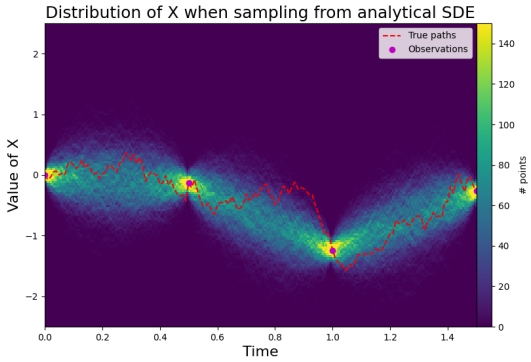
$$\begin{aligned} \tilde{X}_t &= \mu_\gamma(t, \tilde{X}_t, z_{i:i+3})\,\mathrm{d}t + \sigma_\gamma(t, \tilde{X}_t, z_{i:i+3}), \quad t \in (t_{i-1}, t_i],\ i = 1, 2, ...N-3 \\ \tilde{X}_0 &= x_0 \end{aligned} \tag{5.12}$$

The pre-training is also modified, such that the term corresponding to the analytical drift term in the loss function in (5.11), is replaced with the drift term corresponding to four future observations. For the terms added, see (2.43).

For the following results, the underlying paths are Brownian motions extended to the time frame $t \in [0, 3]$, with 7 observations (at each half time unit). This means we can consider paths up to $t = 1.5$, since we always need four future observations. The distribution, in the case of no noise, when sampling from the neural SDE with a pre-trained drift and when sampling from the analytical conditional SDE, can be seen in Figure 5.15. Notice that the drift term in this case only depends on the next observation. The resulting distributions when adding measurement noise ($\sigma_Z = 0.2$) can be seen in Figure 5.16.

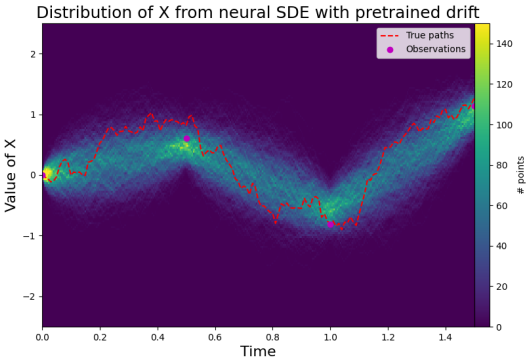


(a) Distribution of generated paths from neural SDE when drift-network is pre-trained.

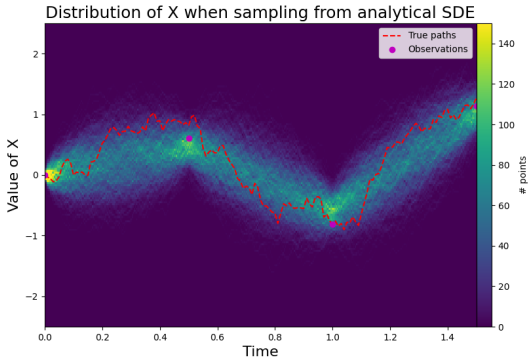


(b) Distribution of generated paths from analytical conditional SDE when conditioning on Z_1.

Figure 5.15: Distribution of neural SDE and analytical conditional SDE when sampling with time step $\mathrm{d}t = 0.01$. No measurement noise.



(a) Distribution of generated paths from neural SDE when drift-network is pre-trained.



(b) Distribution of generated paths from analytical conditional SDE when conditioning on Z_1.

Figure 5.16: Distribution of neural SDE and analytical conditional SDE when sampling with time step $\mathrm{d}t = 0.01$. Measurement noise $\sigma_Z = 0.2$.

The pre-trained neural SDE clearly obtains a similar distribution as the analytical conditional SDE, both with and without noise. The conclusion is that the *pre-trained* approach is an adequate implementation of the neural SDE, and is therefore our chosen approach.

6

Results for conditional nonlinear SDE

In this chapter the performance of the neural SDE is compared to that of a particle filter. Since most applications involve targets moving according to a nonlinear trajectory, the underlying dynamics considered here are nonlinear. More specifically, we consider two nonlinear examples. The most interesting example of these is when the data are described by a bimodal distribution, which could correspond to a target turning either left or right. We conclude this section with a discussion regarding further developments that could be considered and make a conclusion of the entirety of the project.

6.1 Comparison against particle filter

A particle filter is a sequential Monte Carlo method to solve the filtering problem. It does this by propagating particles forward in time, using an SDE model, and assigning them weights based on the likelihood of an observation given each particle. Resampling is then performed to focus on particles with larger weight. The method can be extended to also solve the smoothing problem by, after filtering, propagating the particles backwards, computing smoothed weights. This method is called Forward-Backward smoothing. The smoothed particles approximate the smoothing distribution. For further explanation of particle filters and Forward-Backward smoothing, see Appendix B.

In this project, a particle filter and smoother with 5000 particles was used. This provided an adequate solution to the filtering and smoothing problems. Given that the particle filter and smoother knows the exact underlying model, the problems are solved exactly, at least as the number of particles goes to infinity. This means that the particle filter and smoother is a tough benchmark for the neural SDE.

As concluded in Chapter 5, the *pre-trained* approach is the final implementation of the neural SDE. All results in this chapter are generated with this approach and the time step $\mathrm{d}t = 0.01$.

6.1.1 Nonlinear dynamics

To determine whether the neural SDE can learn nonlinear dynamics, the SDE

$$\mathrm{d}X_t = (-10X_t^3 - 5X)\,\mathrm{d}t + 0.5\,\mathrm{d}W_t, \quad t \in [0, 1.5]; \tag{6.1}$$
$$Z_1 = X_{1.5} + \mathcal{N}(0, \sigma_Z^2), \tag{6.2}$$

is used. We condition on one observation z_1 at $t = 1.5$ with the measurement noise set to $\sigma_Z = 0.2$. This is a mean reverting SDE centered around $X_t = 0$. The drift term grows in a nonlinear manner as the path drifts away from zero, meaning that to replicate the true distribution, the neural SDE has to learn a nonlinear drift term.

The unconditional distribution of the SDE is visualized in Figure 6.1. Meanwhile, the resulting distribution of the neural SDE and the conditional distribution of X when estimated with a particle filter can be seen in Figure 6.2. In Figure 6.3 we can see the marginal distributions at time $t = 1.5$. For a quantitative measure of performance, on the path space, the average Wasserstein-1 distance between the marginal distributions of the neural SDE and the particle filter is calculated. This is shown in Figure 6.4.

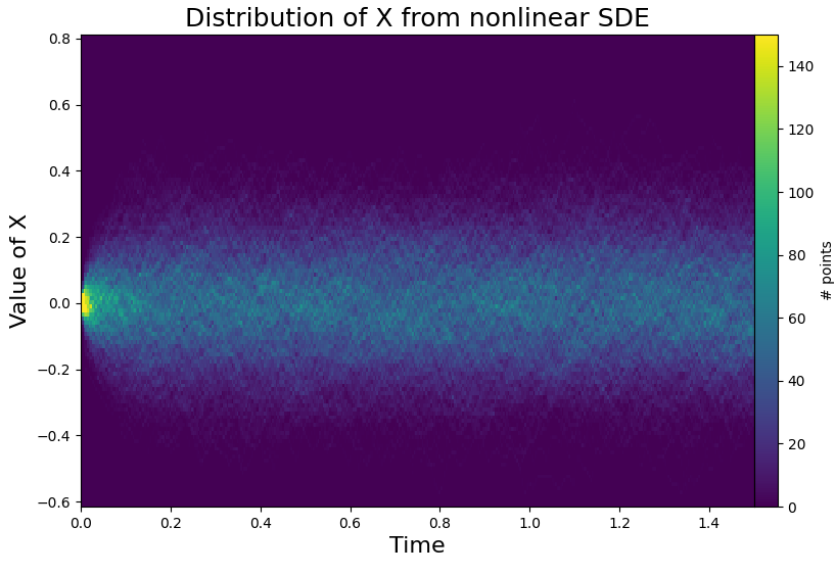
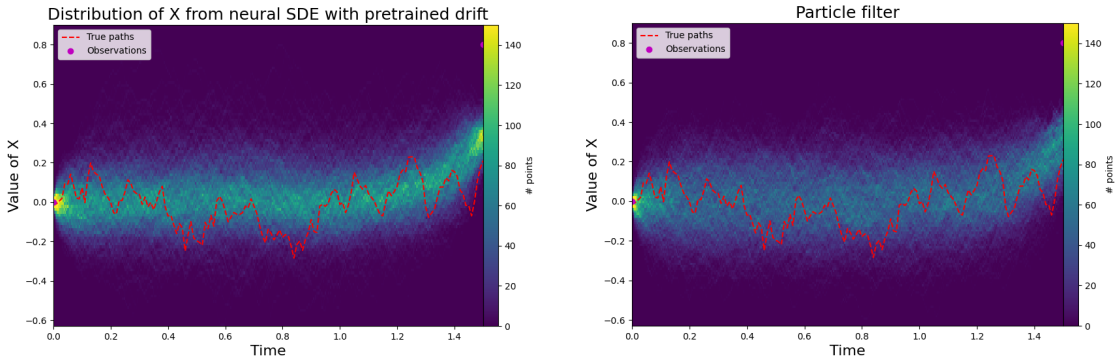


Figure 6.1: Distribution of X from the nonlinear SDE in (6.1).



(a) Distribution of generated paths from neural SDE.

(b) Distribution of generated paths from particle filter.

Figure 6.2: Distribution of neural SDE and particle filter when sampling with time step $\mathrm{d}t = 0.01$. Measurement noise $\sigma_Z = 0.2$.

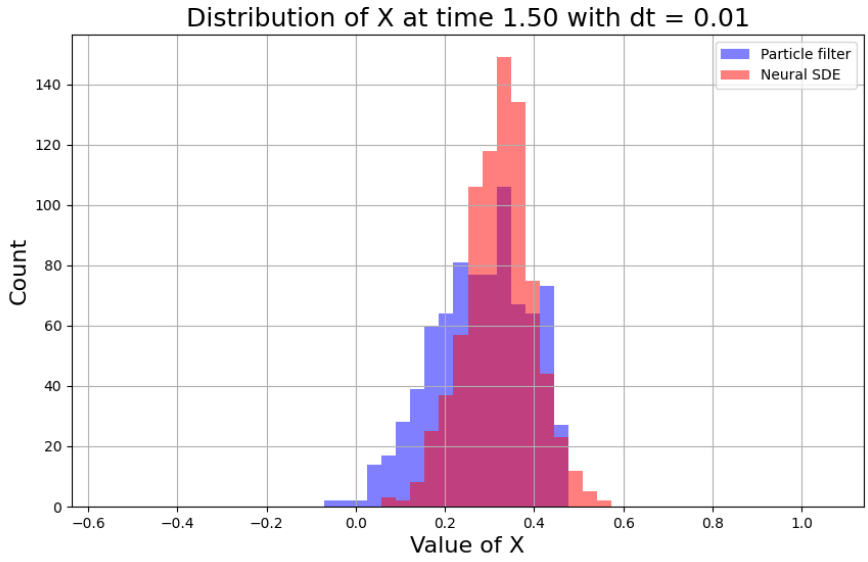


Figure 6.3: Distribution at time $t = 1.5$ of the neural SDE and particle filter.

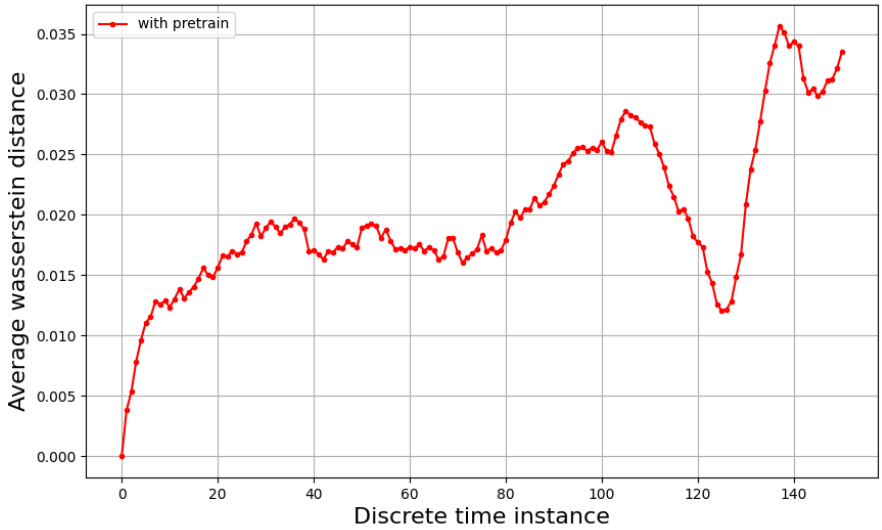


Figure 6.4: Average Wasserstein-1 distance between the marginal distributions of the pre-trained neural SDE and particle filter in the case of the nonlinear mean-reverting SDE in (6.1). Due to the large computational cost of the particle filter, only 10 samples are used to calculate the average.

We can see that the resulting distribution of paths generated from the neural SDE is similar, but not identical, to that of the particle filter. The distribution is slightly more concentrated (lower variance) for the neural SDE compared to the particle filter. The conclusion, keeping in mind that the particle filter knows the underlying model, is that the neural SDE solves the nonlinear smoothing (and filtering in this case) problem well. A method to improve the capability to capture nonlinear dependencies is discussed in Section 6.2.

6.1.2 Bimodal distribution

To test whether the neural SDE can learn a bimodal distribution, the nonlinear SDE

$$\begin{aligned} \mathrm{d}X_t &= (-4X_t^3 + 6X_t)\,\mathrm{d}t + 0.5\,\mathrm{d}W_t, \quad t \in [0, 1.5]; \\ Z_1 &= X_{1.5} + \mathcal{N}(0, \sigma_Z^2), \end{aligned} \tag{6.3}$$

is considered. This SDE has a clear bimodal distribution which is visualized in Figure 6.5. Again, one measurement z_1 at $t = 1.5$ is considered. However, in this case the measurement noise is set to $\sigma_Z = 2.0$. By considering larger measurement noise it is possible to achieve an observation which is in the middle of modes. Conditioning on this observation yields almost no information of which mode the true path belongs to, meaning the conditional distribution is almost the same as the unconditional one.

When conditioning on such an observation, the resulting distribution when sampled from the neural SDE and the conditional distribution estimated with a particle filter, can be seen in Figure 6.6. Notice that even though the underlying path is in

the lower mode, the particle filter finds it more probable that the path was in the upper node, which is due to the observation being closer to that mode. In Figure 6.7 we can see the marginal distributions at time $t = 1.0$. The average Wasserstein-1 distance between the marginal distributions of the neural SDE and the particle filter is shown in Figure 6.8.

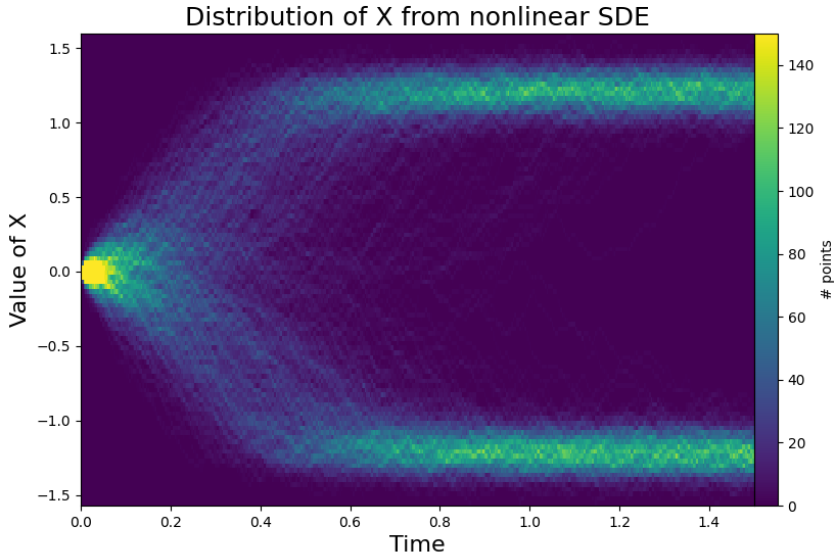
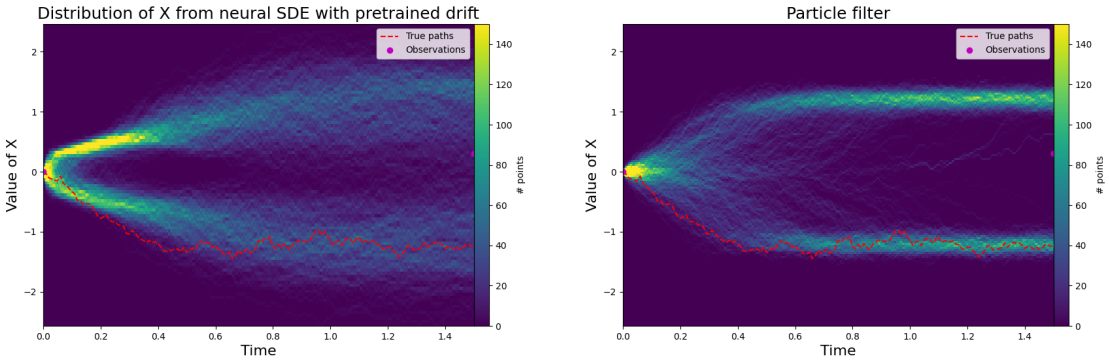


Figure 6.5: Distribution of X from bimodal SDE in (6.3).



(a) Distribution of generated paths from neural SDE.

(b) Distribution of generated paths from particle filter.

Figure 6.6: Distribution of neural SDE and particle filter when sampling with time step $\mathrm{d}t = 0.01$. Measurement noise $\sigma_Z = 2.0$.

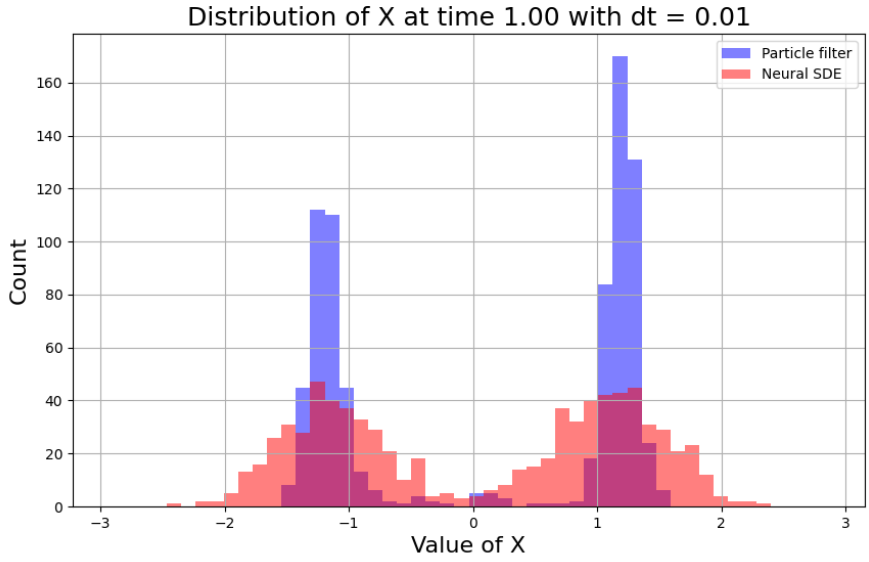


Figure 6.7: Distribution at time $t = 1.0$ of the neural SDE and particle filter.

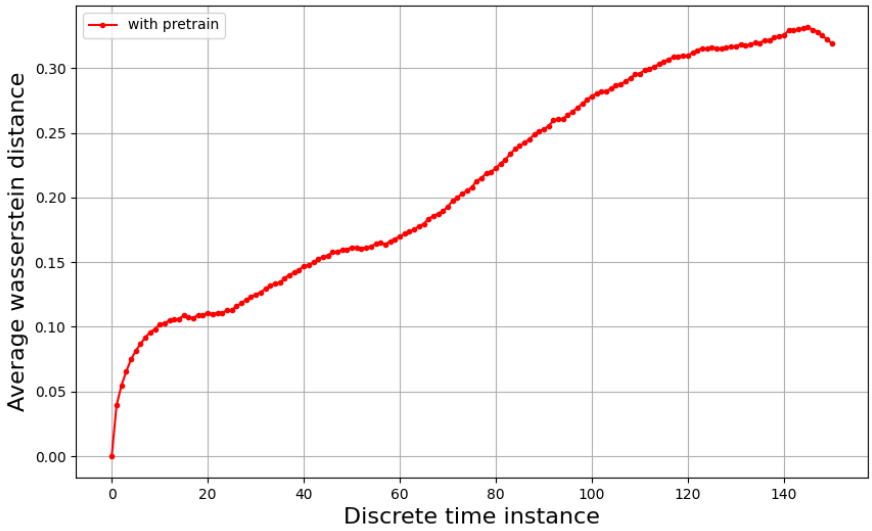


Figure 6.8: Average Wasserstein-1 distance between the marginal distributions of the pre-trained neural SDE and particle filter in the case of the bimodal SDE (6.3). Due to the large computational cost of the particle filter, only 10 samples are used to calculate the average.

We can see that the neural SDE does learn to represent a bimodal distribution, although not as good as the particle filter. The fact that the neural SDE is able to represent any form of bimodal distribution is seen as a success, as it was unclear whether expected signatures could capture this behaviour. Still, there is room for improvement. The average Wasserstein-1 distances are large, at least compared to

those seen in Section 6.1.1, which confirms that the bimodal distributions are not very similar.

An important take from the evaluation of the bimodal nonlinear SDE, is the amount of time needed to train the neural SDE. To achieve the results presented, the neural SDE had to train for a considerable amount of more epochs than in previous examples. This means that, as the underlying SDE becomes increasingly complex, the neural SDE has to train for longer periods to represent the complex structure. Although training the model may need some time, evaluating it for a set of observations is almost instantaneous. The particle filter on the other hand is computationally heavy. If it is possible to increase the accuracy in the nonlinear cases to the level of a particle filter, the neural SDE could be a superior modelling choice due to the possibility of instantaneous sampling.

6.2 Sweeping window

The results show that the neural SDE has a somewhat difficult time learning nonlinear dynamics, especially in the case of the bimodal distribution. The reason for this is that the loss while training is not significantly lower for paths generated with the particle filter compared to paths generated from the neural SDE, meaning that the correct solution is not the best one according to the loss function. One possible reason for this could be that truncated signatures are considered. The signature transformation is always considered for entire paths, and characteristics that are captured in the higher order terms could be lost due to truncation, despite augmentations made to mitigate this.

A possible solution to further mitigate this loss might be to consider the signature on smaller parts of the path. The (truncated) signature would then be able to capture more of the dynamics specific to that part of the path in lower order terms. This would make the information loss when truncating not as severe, and could allow for easier and more precise learning for the neural SDE. Our proposed approach is a moving window that calculates the signature of the segment of the path that is inside the window. The window would then sweep over the entirety of the path. The loss for each window is calculated in the same way as previously described in the thesis, but the sum of the losses from each window is considered. This means that we end up with the following loss function

$$\gamma \mapsto \sum_{k=1}^{N_W} \frac{1}{N} \sum_{j=1}^{N} ||\Phi^k(\mathbf{z}_N^{(j)}) - \frac{1}{M} \sum_{i=1}^{M} S^m(\tilde{X}_k^{(i,j)})||_p, \tag{6.4}$$

where N_W is the number of windows. In this scenario Φ^k estimates the conditional signature of the segment of the path that is inside the k:th window. The segment of the path inside the window is denoted as $\tilde{X}_k$. Notice that N_W depends on two parameters: the window size and the stride. The window size is how many data points of the path the signature transformation is applied on and the stride is the number of steps the window take each time. This means that each data point is (if

the stride is less than the size of window) considered more than one time. Notice as well that this method is similar to the approach considered in [1]. They obtained promising results, which advocates for this approach.

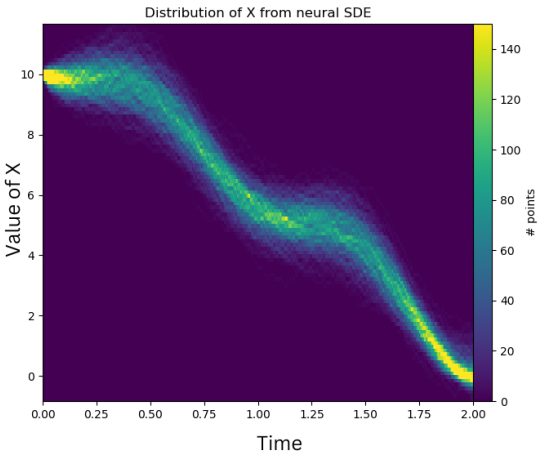
Implementing the neural SDE with this sweeping window would most likely improve the results obtained in Section 6.1.1 and Section 6.1.2. However, it would also come with a greater computational cost. Several Φ-networks would need to be trained and the number of signature calculations would increase. This is mainly the reason why the sweeping window was not utilized in this thesis. However, we did an example of a nonlinear SDE in time to see if the approach should be examined further. The SDE considered is

$$\begin{aligned} \mathrm{d}X_t &= \left(1 + 5\sin(2\pi t) - \frac{X_t}{2-t}\right)\mathrm{d}t + \mathrm{d}W_t, \quad t \in [0,2],\, X_0 = 10 \\ Z_1 &= 0. \end{aligned} \tag{6.5}$$

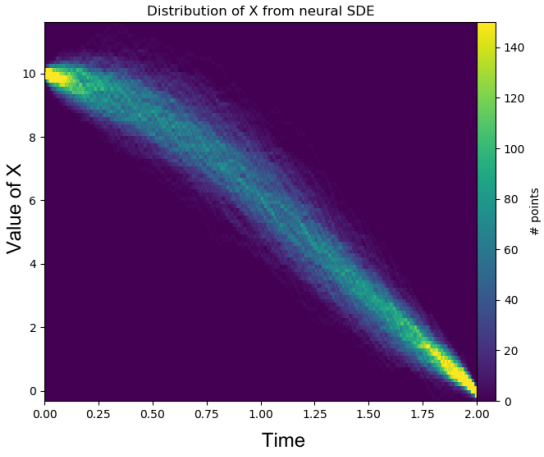
This unconditional SDE could be seen as a conditional SDE where each path goes through 0 at $t = 2$. The pre-training was performed in the usual manner as described in Section 5.6.2.3.

The resulting distribution from the neural SDEs implemented with a sweeping window, the distribution from the *pre-trained* neural SDE and the true distribution can be seen in Figure 6.9.

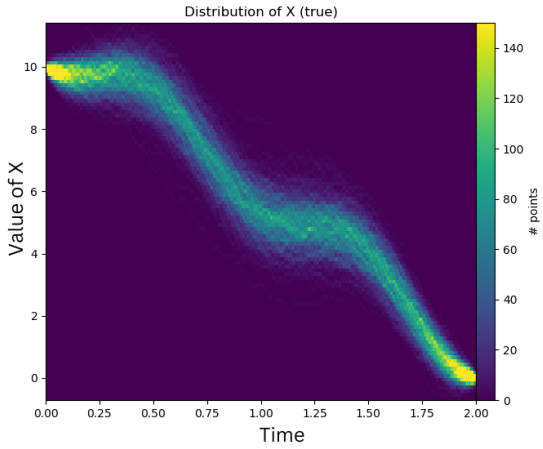
The results strengthen the idea to further investigate the sweeping window approach. However, the sweeping window method has a difficult time squeezing the distribution towards $t = 2$, much like the *standard* approach. Incorporating it together with pre-training could prove successful and is something to consider in further work. Furthermore, if implemented in such a way that is considered in [1], having to train several Φ networks could be avoided, but that is just speculative.



(a) Distribution of paths from Neural SDE, sweeping window (size 11 with stride 2).



(b) Distribution of paths from Neural SDE, *pre trained* approach.



(c) Distribution of true paths sampled from (6.5).

Figure 6.9: Distribution of two approaches to the neural SDE and the true distribution. Time step $\mathrm{d}t = 0.01$.

6.3 Further development

The final implementation of the neural SDE has limitations compared what is needed for real-life application. This means that there are several further developments to consider.

One of the more straight-forward developments of the current model would be to have the neural SDE trained against a richer set of SDEs. This would in practice correspond to having the neural SDE trained against several types of targets. As a consequence of this, it would not only solve the smoothing problem, but also identify

what kind of SDE the target is following. Thereby, the neural SDE could solve the classification problem as well.

Another limitation in this thesis was that X and Z were one-dimensional. Only extending to a richer space of SDEs would not itself make the neural SDE useful in practice. The model also need to be scaled up to dimensions representing relevant physical models. In practice, Z would reside in a lower dimensional (measurement) space (position, velocity), while X, most likely, would reside in a higher dimensional (state) space (position, velocity, acceleration). This would introduce problems regarding the current approach. More explicitly, the loss function for the pre-training in (5.11) cannot be used. Another approach for the pre-training would need to be implemented. Our proposed solution is to estimate X at the time instances where measurements are made with a neural network. Using these estimates, a similar pre-training approach could be developed.

Furthermore, a straight-forward extension would be, as discussed in Section 5.6.2.5, to introduce several drift and diffusion networks to handle a varying number of observations. This would make the neural SDE able to solve the filtering problem as well. Additionally, if one introduces networks for the case when there are no future observations, the prediction problem can be solved too.

Lastly, the SDEs considered in this thesis do not allow for non-continuous jumps. In real life problems that is plausible, for example if we model acceleration, non-continuity could be the effect of turning the engine on or off. Considering Lévy noise (having jumps) would make the neural SDE more expressive, however, since we discretize the data, it is difficult to see how the neural SDE would differentiate between this and the stochastic term (if the Lévy noise is not too great).

6.4 Concerns regarding learning uncertainty connected to ODEs

The neural SDEs in this thesis have only been tested when the underlying model is an SDE. Since SDEs, per definition, have uncertainty in form of a stochastic component, the neural SDE can learn to replicate this uncertainty. When applying the neural SDEs to targets in a tracking algorithm, it would need to be trained against paths that follow real-life dynamics. These dynamics are more similar to that of an ODE, at least if one disregards small effects such as wind. Given observations of a path of this kind, the goal would be to model the uncertainty with the stochastic component. Since there is no stochastic part of the data, the neural SDE would, however, learn to set the diffusion coefficient to 0, especially since the paths are augmented with quadratic variation. This means that the neural SDE would not be able to model any form of uncertainty unless the underlying model actually is an SDE. When developing the neural SDE towards real-life applications, this is a problem that needs to be addressed.

6.5 Conclusion

The purpose of this project was to train a neural SDE such that its distribution is approximately equal to that of the conditional distribution. Although limited and simplified, this has been done successfully. With further development needed to improve the accuracy when the dynamics are nonlinear, the results obtained indicates that a neural SDE could be a plausible approach for solving the smoothing problem. However, questions remain whether this will be a valid approach for real-life applications.

Bibliography

[1] S Liao et al. “Sig-Wasserstein GANs for Conditional Time Series Generation”. In: *arXiv e-prints* (2023).

[2] P Kidger. “On Neural Differential Equations”. In: *arXiv e-prints* (2021).

[3] S Särkkä and A Solin. “Applied Stochastic Differential Equations”. In: *Cambridge University Press* (2019).

[4] X Mao. *Stochastic Differential Equations and Applications*. Second edition. 2008.

[5] I Chevyrev and A Kormilitzin. “A Primer on the Signature Method in Machine Learning”. In: *arXiv e-prints* (2016).

[6] D Levin, T Lyons, and H Ni. “Learning from the past, predicting the statistics for the future, learning an evolving system”. In: *arXiv e-prints* (2016).

[7] T Lyons and H Ni. “EXPECTED SIGNATURE OF BROWNIAN MOTION UP TO THE FIRST EXIT TIME FROM A BOUNDED DOMAIN”. In: *arXiv e-prints* (2015).

[8] B Piccoli and F Rossi. “On properties of the Generalized Wasserstein distance”. In: *arXiv e-prints* (2014).

[9] DP Kingma and JL Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv e-prints* (2017).

[10] RM Schmidt, F Schneider, and P Hennig. “Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers”. In: *arXiv e-prints* (2021).

[11] M Arjovsky, S Chintala, and L Bottou. “Wasserstein GAN”. In: *arXiv e-prints* (2017).

[12] IJ Goodfellow et al. “Generative Adversarial Nets”. In: *arXiv e-prints* (2014).

[13] T Miyato et al. “SPECTRAL NORMALIZATION FOR GENERATIVE ADVERSARIAL NETWORKS”. In: *arXiv e-prints* (2018).

[14] H Gouk et al. “Regularisation of neural networks by enforcing Lipschitz continuity”. In: *arXiv e-prints* (2020).

[15] A Klenke. *Probability Theory: A Comprehensive Course*. 2nd. 2013.

[16] X Glorot, A Bordes, and Y Bengio. “Deep Sparse Rectifier Neural Networks”. In: vol. 15. 2010.

[17] S Hochreiter and J Schmidhuber. “Long Short-term Memory”. In: *Neural computation* 9 (1997), pp. 1735–80.

[18] J Zuo, B Yan, and W Lian. “Forward–backward particle smoother for nonlinear systems with one-step random measurement delay”. In: *IET Signal Processing* 12.7 (2018), pp. 836–843.

A

Derivation of the special case of the Brownian motion and soem proofs

Consider the setup where we have

$$\begin{aligned} \mathrm{d}X_t &= \mathrm{d}W_t \\ Z_n &= X_{t_n}, \quad n = 1, 2, ..., N. \end{aligned}$$

where we want to determine

$$\frac{\mathrm{d}}{\mathrm{d}x} \log p(Z_{i+1:N}|X_{t+s}). \tag{A.1}$$

To prove this we use the following identity

$$p(Z_{i+1:N}|X_t) = p(Z_{i+1}|X_t) \prod_{j=i+2}^{N} p(Z_j|Z_{i+1:j-1}, X_t). \tag{A.2}$$

With more generic notation for distributions, we have

$$\pi(Z_{i+1:N}|X_t) = \pi(Z_{i+1}|X_t) \prod_{j=i+2}^{N} \pi(Z_j|Z_{i+1:j-1}, X_t). \tag{A.3}$$

We first consider the term in (A.3) that is outside the product, i.e.,

$$\pi(Z_{i+1}|X_t). \tag{A.4}$$

This term can be written as

$$\pi(Z_{i+1}|X_t) = \int \pi(Z_{i+1}, X_{t_{i+1}}|X_t)\,\mathrm{d}X_{t_{i+1}} = \int \pi(Z_{i+1}|X_{t_{i+1}})\pi(X_{t_{i+1}}|X_t)\,\mathrm{d}X_{t_{i+1}}. \tag{A.5}$$

Solving this is done by considering the distribution for each term inside the integral. The first term is normally distributed with mean $X_{t_{i+1}}$ and variance σ_Z^2. Meanwhile, the second term is normally distributed with mean X_t and variance $t_{i+1} - t$. According to Theorem 10 in Appendix A, we get

$$\pi(Z_{i+1}|X_t) = \frac{1}{\sqrt{2\pi(\sigma_Z^2 + (t_{i+1} - t))}} e^{-\frac{(Z_{i+1} - X_t)^2}{2(\sigma_Z^2 + (t_{i+1} - t))}}. \tag{A.6}$$

This is the pdf for the normal distribution with mean X_t and variance $\sigma_Z^2+(t_{i+1}-t)$. There is only one term left to derive since the generic term in the product can be defined iteratively. To be specific we want to find

$$\pi(Z_j|Z_{i+1:j-1}, X_t), \forall j = i+2, i+3, ..., N. \tag{A.7}$$

When doing this we assume that

$$\pi(X_{t_{j-1}}|Z_{i+1:j-1}, X_t) \tag{A.8}$$

is known, and we define it as

$$\pi(X_{t_{j-1}}|Z_{i+1:j-1}, X_t) \sim N(\mu_{j-1,(t)}, \sigma^2_{j-1,(t)}), \tag{A.9}$$

which is shown to be true further down in this section. Notice that the mean and the variance is time dependent. The mean is also dependent on X_t, however that is left out notation wise to make it more readable. We consider (A.7) which can be written as

$$\pi(Z_j|Z_{i+1:j-1}, X_t) = \int \pi(Z_j|X_{t_{j-1}})\pi(X_{t_{j-1}}|Z_{i+1:j-1}, X_t)\,\mathrm{d}X_{t_{j-1}}. \tag{A.10}$$

We use the same approach, i.e., we consider the terms separately. The first term is written as

$$\pi(Z_j|X_{t_{j-1}}) = \int \pi(Z_j|X_{t_j})\pi(X_{t_j}|X_{t_{j-1}})\,\mathrm{d}X_{t_j}. \tag{A.11}$$

This integral is solvable, and the resulting distribution is (by using Theorem 10 again)

$$\pi(Z_j|X_{t_{j-1}}) = \text{Normal}(Z_j; X_{t_{j-1}}, \sigma_Z^2 + t_j - t_{j-1}). \tag{A.12}$$

Because we are considering equidistant time intervals, it can be written as

$$\pi(Z_j|X_{t_{j-1}}) = \text{Normal}(Z_j; X_{t_{j-1}}, \sigma_Z^2 + \Delta t). \tag{A.13}$$

Here Δt stands for the size of the time step. The second term was assumed to be known and therefore we get the following integral to solve

$$\int \text{Normal}(Z_j; X_{t_{j-1}}, \sigma_Z^2 + \Delta t)\text{Normal}(X_{t_{j-1}}; \mu_{j-1,(t)}, \sigma^2_{j-1,(t)})\,\mathrm{d}X_{t_{j-1}} \tag{A.14}$$

and with the same argumentation as before, it reads

$$\pi(Z_j|Z_{i+1:j-1}, X_t) = \text{Normal}(Z_j; \mu_{j-1,(t)}, \sigma_Z^2 + \sigma^2_{j-1,(t)} + \Delta t). \tag{A.15}$$

For this derivation to be valid, we need to prove that the distribution found in (A.9) is the correct one, and an iterative formula for the mean and variance needs to be determined. The term (or rather the term for the following j) can be written as

$$\begin{aligned}\pi(X_{t_j}|Z_{i+1:j}, X_t) =& \frac{\pi(Z_j|X_{t_j})\pi(X_{t_j}|Z_{i+1:j-1}, X_t)\pi(Z_{i+1:j-1}|X_t)}{\pi(Z_j|Z_{i+1:j-1}, X_t)\pi(Z_{i+1:j-1}|X_t)} \\ =& \frac{\pi(Z_j|X_{t_j})\pi(X_{t_j}|Z_{i+1:j-1}, X_t)}{\pi(Z_j|Z_{i+1:j-1}, X_t)}\end{aligned} \tag{A.16}$$

and if disregard the denominator we have

$$\pi(X_{t_j}|Z_{i+1:j}, X_t) = \pi(Z_j|X_{t_j})\pi(X_{t_j}|Z_{i+1:j-1}, X_t). \tag{A.17}$$

This is possible since the denominator does not depend on X_{t_j} and therefore does not affect the distribution. It merely works as a normalization constant and therefore the equality sign should be seen as equality in distribution. The second term can be written as

$$\pi(X_{t_j}|Z_{i+1:j-1}, X_t) = \int \pi(X_{t_j}|X_{t_{j-1}})\pi(X_{t_{j-1}}|Z_{i+1:j-1}, X_t)\,\mathrm{d}X_{t_{j-1}}. \tag{A.18}$$

Both these terms are known which therefore reads

$$\pi(X_{t_j}|Z_{i+1:j-1}, X_t) = \int \text{Normal}(X_{t_j}; X_{t_{j-1}}, \Delta t)\text{Normal}(X_{t_{j-1}}; \mu_{j-1,(t)}, \sigma^2_{j-1,(t)})\,\mathrm{d}X_{t_{j-1}} \tag{A.19}$$

and yields the result

$$\pi(X_{t_j}|Z_{i+1:j-1}, X_t) = \text{Normal}(X_{t_j}; \mu_{j-1,(t)}, \sigma^2_{j-1,(t)} + \Delta t). \tag{A.20}$$

(A.17) can therefore be written as

$$\pi(X_{t_j}|Z_{i+1:j}, X_t) = \text{Normal}(Z_j; X_{t_j}, \sigma^2_Z)\text{Normal}(X_{t_j}; \mu_{j-1,(t)}, \sigma^2_{j-1,(t)} + \Delta t) \tag{A.21}$$

which with the Normal-Normal conjugacy, see Theorem 11 in Appendix A, it reads

$$\pi(X_{t_j}|Z_{i+1:j}, X_t) = \text{Normal}\Big(X_{t_j}; \frac{Z_j(\sigma^2_{j-1,(t)} + \Delta t) + \sigma^2_Z\mu_{j-1,(t)}}{\sigma^2_Z + \sigma^2_{j-1,(t)} + \Delta t}, \frac{\sigma^2_Z(\sigma^2_{j-1,(t)} + \Delta t)}{\sigma^2_Z + \sigma^2_{j-1,(t)} + \Delta t}\Big). \tag{A.22}$$

So to summarise, in each step we have the following:

$$\pi(Z_j|Z_{i+1:j-1}, X_t) = \text{Normal}(Z_j; \mu_{j-1}, \sigma^2_Z + \sigma^2_{j-1,(t)} + \Delta t) \tag{A.23}$$

and the update step

$$\begin{aligned}&\pi(X_{t_j}|Z_{1:j}, X_t) = \\ =&\text{Normal}\Big(X_{t_j}; \frac{Z_j(\sigma^2_{j-1,(t)} + \Delta t) + \sigma^2_Z\mu_{j-1,(t)}}{\sigma^2_Z + \sigma^2_{j-1,(t)} + \Delta t}, \frac{\sigma^2_Z(\sigma^2_{j-1,(t)} + \Delta t)}{\sigma^2_Z + \sigma^2_{j-1,(t)} + \Delta t}\Big)\end{aligned}$$

which more specifically for the mean and variance reads

$$\mu_{j,(t)} = \frac{Z_j(\sigma^2_{j-1,(t)} + \Delta t) + \sigma^2_Z\mu_{j-1,(t)}}{\sigma^2_Z + \sigma^2_{j-1,(t)} + \Delta t} \tag{A.24}$$

$$\sigma^2_{j,(t)} = \frac{\sigma^2_Z(\sigma^2_{j-1,(t)} + \Delta t)}{\sigma^2_Z + \sigma^2_{j-1,(t)} + \Delta t} \tag{A.25}$$

$$\forall j = i + 2, ..., N. \tag{A.26}$$

In order for this to be valid, or even make sense, we need an initial condition, i.e., determine $\mu_{i+1,(t)}$ and $\sigma_{i+1,(t)}$. These are found when considering

$$\begin{aligned}\pi(X_{t_{i+1}}|Z_{i+1}, X_t) &\propto \pi(Z_{i+1}|X_{t_{i+1}})\pi(X_{t_{i+1}}|X_t) \\ &= \text{Normal}(Z_{i+1}; X_{t_{i+1}}, \sigma_Z^2)\text{Normal}(X_{t_{i+1}}; X_t, t_{i+1} - t) \\ &= \text{Normal}\Big(X_{t_{i+1}}; \frac{Z_{i+1}(t_{i+1} - t) + \sigma_Z^2 X_t}{\sigma_Z^2 + t_{i+1} - t}, \frac{\sigma_Z^2(t_{i+1} - t)}{\sigma_Z^2 + t_{i+1} - t}\Big)\end{aligned} \tag{A.27}$$

which leads to the initial conditions

$$\mu_{i+1,(t)} = \frac{Z_{i+1}(t_{i+1} - t) + \sigma_Z^2 X_t}{\sigma_Z^2 + t_{i+1} - t} \tag{A.28}$$

$$\sigma_{i+1,(t)}^2 = \frac{\sigma_Z^2(t_{i+1} - t)}{\sigma_Z^2 + t_{i+1} - t}. \tag{A.29}$$

(A.27) also shows that the terms are normally distributed, which was assumed. This concludes the first part of the formula for the Brownian motion. Using this we can take the derivative of the logarithm of the expressions, i.e., calculate

$$\frac{\mathrm{d}}{\mathrm{d}x} \log p(Z_{i+1:N}|X_t). \tag{A.30}$$

The expression to consider can be written as

$$p(Z_{i+1:N}|X_t) = \prod_{j=i+2}^{N} \mathcal{N}(Z_j; \mu_{j-1,(t)}, \sigma_Z^2 + \sigma_{j-1,(t)}^2 + \Delta t)\mathcal{N}(Z_{i+1}; X_t, \sigma_Z^2 + t_{i+1} - t) \tag{A.31}$$

and applying the logarithm converts the product to a sum, which yields

$$\begin{aligned}\log p(Z_{i+1:N}|X_t) &= \log \prod_{j=i+2}^{N} \mathcal{N}(Z_j; \mu_{j-1,(t)}, \sigma_Z^2 + \sigma_{j-1,(t)}^2 + \Delta t)\mathcal{N}(Z_{i+1}; X_t, \sigma_Z^2 + t_{i+1} - t) \\ &= \sum_{j=i+2}^{N} \log \mathcal{N}(Z_j; \mu_{j-1,(t)}, \sigma_Z^2 + \sigma_{j-1,(t)}^2 + \Delta t) \\ &\quad + \log \mathcal{N}(Z_{i+1}; X_t, \sigma_Z^2 + t_{i+1} - t) \\ &\sim_{X_t} \sum_{j=i+2}^{N} -\frac{1}{2}\frac{(Z_j - \mu_{j-1,(t)})^2}{\sigma_Z^2 + \sigma_{j-1,(t)}^2 + \Delta t} - \frac{1}{2}\frac{(Z_{i+1} - X_t)^2}{\sigma_Z^2 + t_{i+1} - t}.\end{aligned} \tag{A.32}$$

where $\sim_{X_t}$ in this setting means that we disregard any term not containing X_t. Taking the derivative with respect to X_t yields

$$\frac{\mathrm{d}}{\mathrm{d}x} \log p(Z_{i+1:N}|X_t) = \sum_{j=i+2}^{N} \frac{Z_j - \mu_{j-1,(t)}}{(\sigma_Z^2 + \sigma_{j-1,(t)}^2 + \Delta t)} \frac{\mathrm{d}}{\mathrm{d}x}\mu_{j-1,(t)} + \frac{Z_{i+1} - X_t}{(\sigma_Z^2 + t_{i+1} - t)}. \tag{A.33}$$

Remember that $\mu_{j-1,(t)}$ depends on both t and X_t meanwhile as $\sigma_{j-1,(t)}$ only depends on t, which is the reason for the iterative derivative of μ. Given the expression for

$\mu_{j-1,(t)}$ derived in the previous section we can derive the iterative derivative formula

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}x}\mu_{i+1,(t)} &= \frac{\sigma_Z^2}{\sigma_Z^2 + t_{i+1} - t} \\ \frac{\mathrm{d}}{\mathrm{d}x}\mu_{j-1,(t)} &= \frac{\sigma_Z^2 \frac{\mathrm{d}}{\mathrm{d}x}\mu_{j-2,(t)}}{\sigma_Z^2 + \sigma_{j-2,(t)}^2 + \Delta t}, \quad j = i+2, ..., N.\end{aligned} \tag{A.34}$$

Theorem 10 (Distribution of Compound Gaussian) *Consider two distributions* $\pi(x|y) = Normal(x; y, \sigma_x^2)$ *and* $\pi(y) = Normal(y; \mu_y, \sigma_y^2)$ *and we want to find the distribution of* $\pi(x)$*, which is defined as,*

$$\pi(x) = \int \pi(x|y)\pi(y)\,\mathrm{d}y \tag{A.35}$$

the resulting distribution will be

$$\pi(x) = Normal(x; \mu_y, \sigma_x^2 + \sigma_y^2) \tag{A.36}$$

which is a compound Gaussian distribution.

Proof:

We write the integral with the pdf:s

$$\begin{aligned}&\int_y \frac{1}{\sqrt{2\pi\sigma_x^2}} e^{-\frac{(x-y)^2}{2\sigma_x^2}} \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(y-\mu_y)^2}{2\sigma_y^2}}\,\mathrm{d}y \\ =&\frac{1}{\sqrt{2\pi\sigma_x^2}}\frac{1}{\sqrt{2\pi\sigma_y^2}} \int_y e^{-\frac{1}{2(\sigma_x^2\sigma_y^2)}((\sigma_x^2+\sigma_y^2)y^2 - 2y(\sigma_y^2 x + \sigma_x^2\mu_y) + \sigma_y^2 x^2 + \sigma_x^2\mu_y^2)}\,\mathrm{d}y \\ =&\frac{1}{\sqrt{2\pi\sigma_x^2}}\frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{\sigma_y^2 x^2 + \sigma_x^2\mu_y^2}{2\sigma_x^2\sigma_y^2}} \int_y e^{-\frac{\sigma_x^2+\sigma_y^2}{2(\sigma_x^2\sigma_y^2)}\left(y - \frac{\sigma_y^2 x + \sigma_x^2\mu_y}{\sigma_x^2+\sigma_y^2}\right)^2} e^{\frac{1}{2}\frac{(\sigma_y^2 x + \sigma_x^2\mu_y)^2}{\sigma_x^2\sigma_y^2(\sigma_x^2+\sigma_y^2)}}\,\mathrm{d}y \\ =&\sqrt{2\pi\frac{\sigma_x^2\sigma_y^2}{\sigma_x^2+\sigma_y^2}}\frac{1}{\sqrt{2\pi\sigma_x^2}}\frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{\sigma_y^2 x^2 + \sigma_x^2\mu_y^2}{2\sigma_x^2\sigma_y^2}} e^{\frac{1}{2}\frac{(\sigma_y^2 x + \sigma_x^2)\mu_y^2}{\sigma_x^2\sigma_y^2(\sigma_x^2+\sigma_y^2)}} \int_y e^{-\frac{\sigma_x^2+\sigma_y^2}{2(\sigma_x^2\sigma_y^2)}\left(y - \frac{\sigma_y^2 x + \sigma_x^2\mu_y}{\sigma_x^2+\sigma_y^2}\right)^2} \frac{1}{\sqrt{2\pi\frac{\sigma_x^2\sigma_y^2}{\sigma_x^2+\sigma_y^2}}}\,\mathrm{d}y\end{aligned}$$

If we consider separately and simplify

$$\begin{aligned}&e^{-\frac{\sigma_y^2 x^2 + \sigma_x^2\mu_y^2}{2\sigma_x^2\sigma_y^2}} e^{\frac{1}{2}\frac{(\sigma_y^2 x + \sigma_x^2\mu_y)^2}{\sigma_x^2\sigma_y^2(\sigma_x^2+\sigma_y^2)}} = e^{-\frac{\sigma_x^2\sigma_y^2 x^2 + \sigma_y^4 x^2 + \sigma_x^4\mu_y^2 + \sigma_x^2\sigma_y^2\mu_y - \sigma_y^4 x^2 + 2\sigma_x^2\sigma_y^2 x\mu_y - \sigma_x^4\mu_y^2}{2\sigma_x^2\sigma_y^2(\sigma_x^2+\sigma_y^2)}} \\ =&e^{-\frac{\sigma_x^2\sigma_y^2 x^2 + 2\sigma_x^2\sigma_y^2 x\mu_y + \sigma_x^2\sigma_y^2\mu_y^2}{2\sigma_x^2\sigma_2^y(\sigma_x^2+\sigma_y^2)}} = e^{-\frac{(x-\mu)^2}{2(\sigma_x^2+\sigma_y^2)}}\end{aligned}$$

which together with the fact that the integral is the pdf for a normal distributed random variable integrated over the entire domain, i.e. the integral is equal to one, yields the following pdf

$$\frac{1}{\sqrt{2\pi(\sigma_x^2+\sigma_y^2)}} e^{-\frac{(x-\mu_y)^2}{2(\sigma_x^2+\sigma_y^2)}} \tag{A.37}$$

which therefore completes the proof.

Theorem 11 (Normal-Normal) *Given the prior* $\pi(\theta) = Normal(\theta; \mu, \sigma_\theta^2)$ *and the likelihood* $\pi(x|\theta) = Normal(x; \theta, \sigma_x^2)$ *the posterior reads* $\pi(\theta|x) = Normal(\theta; \frac{\sigma_\theta^2 x + \sigma_x^2 \mu}{\sigma_x^2 + \sigma_y^2}, \frac{\sigma_x^2 \sigma_\theta^2}{\sigma_x^2 + \sigma_\theta^2})$

Proof: The posterior is proportional to the likelihood and the prior by

$$\pi(\theta|x) \propto_\theta \pi(x|\theta)\pi(\theta) \propto e^{-\frac{1}{2\sigma_x^2}(x-\theta^2)} e^{-\frac{1}{2\sigma_\theta^2}(\theta-\mu)^2} \propto_\theta e^{-\frac{1}{2\sigma_x^2\sigma_\theta^2}(\theta^2(\sigma_x^2+\sigma_\theta^2)-2\theta(\sigma_\theta^2 x+\sigma_x^2\mu))}$$
$$\propto_\theta e^{-\frac{\sigma_x^2+\sigma_\theta^2}{2\sigma_x^2\sigma_\theta^2}\left(\theta-\frac{\sigma_\theta^2 x+\sigma_x^2\mu}{\sigma_x^2+\sigma_\theta^2}\right)^2}$$

which is proportional to the pdf for the normal distribution with mean $\frac{\sigma_\theta^2 x + \sigma_x^2 \mu}{\sigma_x^2 + \sigma_y^2}$ and variance $\frac{\sigma_x^2 \sigma_\theta^2}{\sigma_x^2 + \sigma_\theta^2}$ so the proof is completed.

B

Particle filter

A particle filter is a sequential Monte Carlo method to solve the filtering problem. In this Chapter, the key parts of particle filtering, as well as particle smoothing with the Forward-Backward method, is introduced. For a more detailed description, see [18].

B.1 Setting

First, assume that the underlying SDE is known and described by

$$\begin{aligned} \mathrm{d}X_t &= \mu(X_t, t)\,\mathrm{d}t + \sigma(X_t, t)\,\mathrm{d}W_t, \quad t \in [0, T],\ X_0 = x_0 \\ Z_t &= X_t + V_t. \end{aligned} \tag{B.1}$$

Notice that according to this, observations are gathered in each time instance, which is not the case in this thesis. This problem is mitigated by V_t corresponding to colossal measurement noise in the time instances where observations are not usually gathered, while being the normal measurement noise when observations should be relevant.

Given the underlying SDE, the corresponding state space model can be defined as

$$\begin{aligned} X_0 &= x_0, \\ p(X_t|X_{t-1}) &\sim \mathcal{N}(X_{t-1} + \mu(X_{t-1}, t)\,\mathrm{d}t,\ \sigma(X_{t-1}, t)^2\,\mathrm{d}t), \\ p(Z_t|X_t) &\sim \mathcal{N}(X_t,\ \sigma_{Zt}^2), \end{aligned} \tag{B.2}$$

where σ_{Zt} is the measurement noise at time instance t.

B.2 Filtering

First, N particles $\{X_0^i\}_{i=1}^N$ are initialized from the prior distribution, which in this setting is just the constant x_0. Initial weights $\{w_0^i\}_{i=1}^N$ are assigned to each particle, where each $w_0^i = \frac{1}{N}$.

Next, each particle at time $t-1$ is propagated forward in time by sampling a new state X_t^i according to

$$X_t^i \sim p(X_t^i|X_{t-1}^i) \tag{B.3}$$

For the new state of each particle, its weight is updated according to

$$w_t^i = w_{t-1}^i p(Z_t|X_{t-1}^i). \tag{B.4}$$

The weights $\{w_t^i\}_{i=1}^N$ are then normalized such that they sum to one. If the measurement noise is colossal at this t, $p(Z_t|X_{t-1}^i)$ will be approximately equal for any X_{t-1}^i. This means that this observation will not affect the weights significantly, which is exactly what was desired in this project.

The particle filter then performs resampling to avoid degeneracy (where most particles have negligible weights). This is done by resampling N particles $\{X_t^i\}_{i=1}^N$ with replacement, according to their weights $\{w_t^i\}_{i=1}^N$.

These steps are repeated until $t = T$. The particles and their weights at the terminal time represent the conditional distribution.

B.3 Smoothing

The particle filter can be extended to solve the smoothing problem as well, and there are many different approaches to this. In this thesis, the utilized method is Forward-Backward smoothing. This method requires that the particle filter is run, as described above, such that particles X_t^i and weights w_t^i are obtained for each t.

For each particle, the smoothed weights $\{w_{T|T}^i\}_{i=1}^N$ at time $t = T$ are initialized to the terminal filtering weights $\{w_T^i\}_{i=1}^N$. Then, for each t from $T-1$ to 0 and for each particle, the smoothed weights are updated according to

$$w_{t|T}^i = w_t^i \sum_{j=1}^{N} \frac{w_{t+1|T}^j p(X_{t+1}^j|X_t^i)}{\sum_{k=1}^N w_t^k p(X_{t+1}^j|X_t^k)} \tag{B.5}$$

The smoothed distribution at all t is given by the particles and their smoothed weights.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY