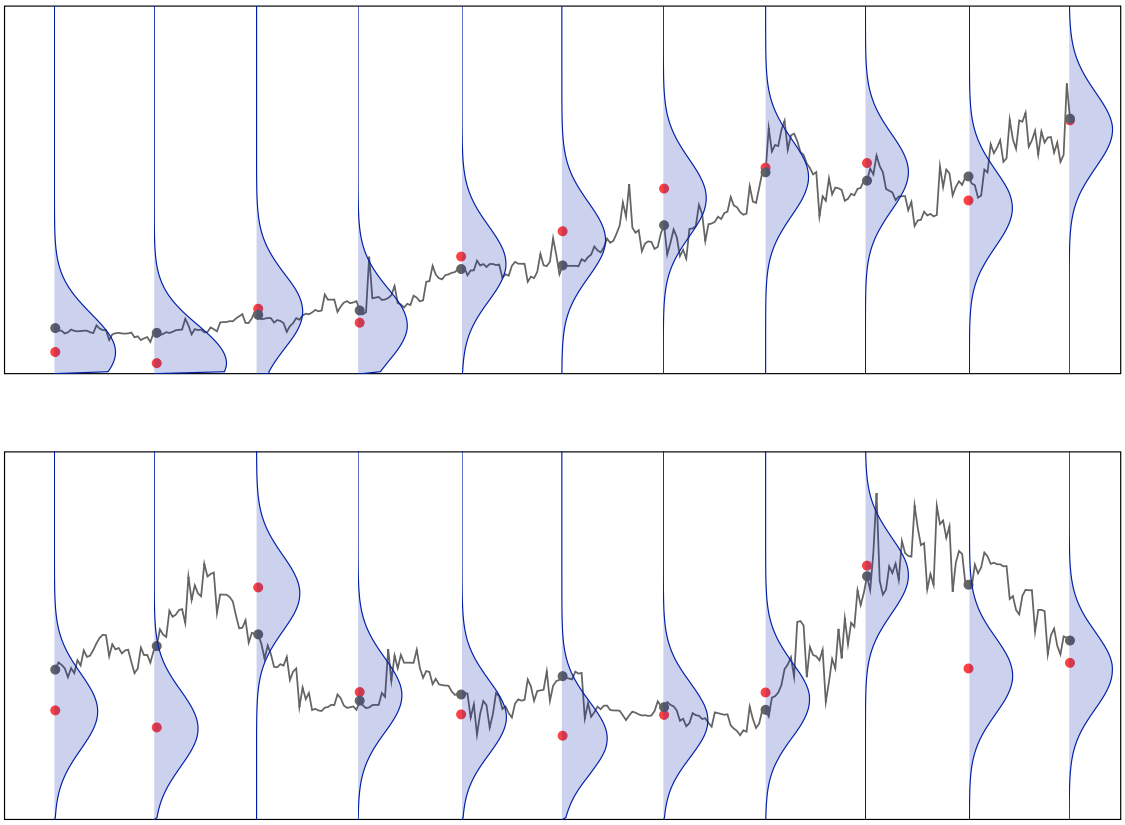

# A Deep Learning Method for Nonlinear Stochastic Filtering

Energy-Based Deep Splitting for Fast and Accurate Estimation of Filtering Densities

Master's Thesis in Engineering Mathematics and Computational Science

Filip Rydin

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg 2024
www.chalmers.se

Master's thesis 2024

# A Deep Learning Method for Nonlinear Stochastic Filtering

Energy-Based Deep Splitting for Fast and Accurate Estimation of Filtering Densities

Filip Rydin

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg 2024

A Deep Learning Method for Nonlinear Stochastic Filtering
Energy-Based Deep Splitting for Fast and Accurate Estimation of Filtering Densities
Filip Rydin



Supervisors:
Adam Andersson, Department of Mathematical Sciences and Saab AB
Kasper Bågmark, Department of Mathematical Sciences

Examiner:
Stig Larsson, Department of Mathematical Sciences

Master's Thesis 2024
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg

Cover: Two sample paths of geometric Brownian motion (grey) with noisy measurements (red) and EBDS filtering densities (blue).

Typeset in LaTeX
Gothenburg, Sweden 2024

A Deep Learning Method for Nonlinear Stochastic Filtering
Energy-Based Deep Splitting for Fast and Accurate
Estimation of Filtering Densities
Filip Rydin
Department of Mathematical Sciences
Chalmers University of Technology

# Abstract

In filtering the problem is to find the conditional distribution of a dynamically evolving state given noisy measurements. Critically, designing accurate filters for nonlinear problems that scale well with the state dimension is exceedingly difficult. In this thesis, a novel filtering method based on deep learning solutions to the Fokker–Planck partial differential equation is treated. Training can be performed offline, which results in a computationally efficient algorithm online, even in high dimensions. This is promising for applications which require good real-time performance, such as target-tracking.

The filtering method, referred to as Energy-Based Deep Splitting (EBDS), is presented in detail and implemented. The performance of EBDS on different example problems is then investigated and compared to benchmark filters, such as variants of the Kalman filter and particle filters. In one dimension EBDS seems to perform superbly, especially considering how fast the filter is at evaluation. In higher dimensions the method performs worse in comparison to the benchmarks, although it still yields sensible density estimates in most cases. Additionally, convergence for EBDS in the number of prediction steps is investigated empirically for two of the example problems. The results in both examples indicate strong convergence of order 1/2. Lastly, a neural network architecture based on Long Short-Term Memory (LSTM) encoders is proposed for EBDS. This architecture yields reduced errors compared to standard fully-connected networks.

In summary, the results indicate that the method is promising and should be examined further. This thesis can be viewed as a reference for future works that aim to apply EBDS in more specific settings or that aim to improve the method further.

Keywords: Nonlinear filtering, Scalable filter, Deep learning, Kalman filter, Particle filter, Fokker–Planck equation, Neural networks, Long short-term memory

# Acknowledgements

Firstly, I would like to express my sincerest gratitude to my supervisors Adam Andersson and Kasper Bågmark. Apart from proposing the topic they have both contributed massively to the quality of this thesis. I am particularly grateful to Kasper for his guidance and unwavering encouragement throughout the project. Our weekly meetings have been invaluable. I can truly say that both Adam and Kasper have exceeded the expectations I had when the project started. It is in no small part thanks to their help and sound advice that I will continue my career in academia this autumn with a PhD at Chalmers.

I would also like to thank my examiner Stig Larsson for his support and help in making this thesis work a seamless experience. Additional gratitude goes to my opponents Elias Stenhede Johansson and Valter Schütz. Thank you for your honest comments and for enduring all the theory and derivations in this report. Finally, this project would have been severely limited without access to compute resources. These were provided by Chalmers e-Commons, for which I am thankful.

This thesis marks the end of five years of studies at the Engineering Physics program at Chalmers. Without amazing friends, passionate teachers and continual support from family these years would not have passed by nearly as quickly as they did. A part of this thesis belongs to everyone who has supported, encouraged and inspired me throughout the journey that has been my time at Chalmers. Thank you.

Filip Rydin, Gothenburg, June 2024

# Contents

# 1
# Introduction

At its core, the stochastic filtering problem concerns finding the conditional distribution of a state that is not directly observable. Instead, the state can only be observed through noisy measurements and often only at specific times. The filtering problem naturally arises in a vast number of applications and its solutions play a pivotal role in fields such as target tracking, finance and biomedical engineering. As a result of its practical applicability and inherent complexity, a tremendous amount of work has been done to understand and solve the filtering problem. Crucially however, the design of accurate and fast filters for nonlinear problems, especially in high dimensions, remains an open research question.

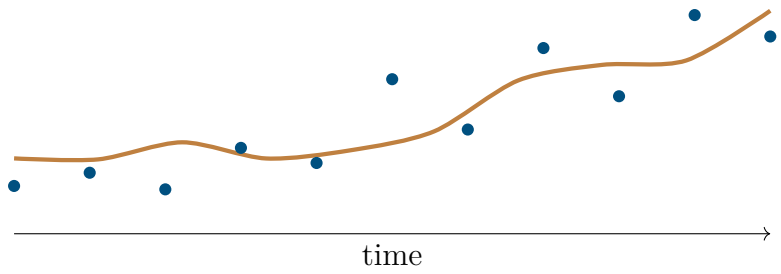


**Figure 1.1:** In filtering, the problem is to find the conditional distribution of a dynamically evolving state (brown) given noisy measurements (blue).

Among existing filtering methods, there exists no ”silver bullet”. Historically and in applications, the impact of the Kalman filter can not be overstated. This filter is exact for linear problems and its extensions, such as the extended and unscented Kalman filters, often yield good approximations for nonlinear problems. Moreover, these filters are easy to implement and computationally efficient. Yet, they do not guarantee satisfactory solutions in many strongly nonlinear cases often encountered in applications. More recently, particle filters, also called sequential Monte Carlo methods, have gained traction. These filters have many advantages, both theoretically and in practice [11]. One major disadvantage however is the complexity of the computations that need to be performed online, as the number of Monte Carlo samples needed to maintain accuracy increases exponentially with the state dimension [32]. In applications where real-time performance is essential, such as within target-tracking, this can be a major obstacle.

While there are clear disadvantages with these algorithms, stochastic filtering is

well understood from a theoretical perspective [12]. It turns out that if observations can be made arbitrarily frequently, i.e., the observation process is continuous, then the unnormalised filtering density solves a Stochastic Partial Differential Equation (SPDE) known as the Zakai equation. In real applications, observations are often only available at fixed times, however. This is the setting of most conventional filters and here, the famous Fokker–Planck equation is important. This is a Partial Differential Equation (PDE) that governs how the distribution of the state evolves between observations. Critically for applications, these equations are rarely tractable. Hence PDE- and SPDE-based filters are used very scarcely.

Recent advances could potentially change this fact. Utilisation of deep learning for solving PDEs and SPDEs is an active research topic and during the last years a number of interesting results have been published in the filtering context. In [5] from 2021, a splitting method is introduced and deep neural networks are used to solve the recursive optimisation problems that arise. The method is extended to SPDEs in [4] and applied to the Zakai equation for the filtering problem. A similar filtering method using neural networks for PDE solutions is proposed in [13] from 2022. A crucial step is taken in [2] from 2023, where the authors extend the method of [5] and [4] for the Zakai equation. The key feature of the method in this paper is that training is performed offline. As a result, the method offers a promising alternative for real-time applications when online performance is critical.

This thesis builds on the work of Bågmark, Andersson and Larsson in [2] and a new similar filtering method proposed by the same authors. The overall goal is to design a precise filter that is computationally light and scales well with the state dimension. In the new method, the observation process is modelled as discrete rather than continuous and the equation of interest is the Fokker–Planck equation, rather than the Zakai equation. This setting with discrete observations is considered more appropriate for applications.

## 1.1 Aim and Outline

The aim of this thesis is to conduct an initial numerical evaluation of the new filtering method proposed by the authors of [2], which is based on deep learning solutions to the Fokker–Planck PDE. As the method is a modification of the one in [2], it is referred to by the same name, namely Energy-Based Deep Splitting (EBDS). The method is implemented and benchmarked against other common filters, such as variants of the Kalman filter and particle filters. The convergence properties are also examined empirically. In parallel with the numerical study in this thesis, the authors of [2] are working on an article regarding the theoretical convergence properties of the same scheme.

To briefly outline the structure of the thesis, in Section 1.2 the filtering problem is introduced more in detail and the preliminaries of EBDS are presented. This is followed by a short review of some application areas of filtering in Section 1.3. Chapters 2 and 3 contain necessary background knowledge from stochastic analysis

and regarding machine learning while Chapter 4 presents other common filtering methods that are used for benchmarking in this thesis. The EBDS method is then derived in Chapter 5, which ends with a formulation of the algorithm in pseudo-code. A more detailed description of the implementation is presented in Chapter 6. In Chapter 7 EBDS is then tested on four different example problems and compared to other filters in terms of performance. Chapter 7 also contains runtime results as well as results when using a more advanced neural network architecture. Finally, in Chapter 8 the results are discussed briefly and possible directions of future research are suggested.

## 1.2 Preliminaries and General Approach

A model in stochastic filtering consists of two essential parts, both modelled as stochastic processes. The state process models the latent quantity of interest, whereas the observation process models measurements of the state. Both processes can be either discrete or continuous. In this thesis, the state is described by a continuous time $d$-dimensional stochastic process $X\colon [0,T] \times \Omega \to \mathbb{R}^d$ satisfying a time-homogeneous diffusion type stochastic differential equation

$$X_t = X_0 + \int_0^t \mu(X_s)\,\mathrm{d}s + \int_0^t \sigma(X_s)\,\mathrm{d}W_s, \quad t \in [0,T]. \tag{1.1}$$

The initial distribution $p_0(x)$ is assumed to be given. The observation process is modelled as a discrete time stochastic process $Y\colon \{0,1,\dots,K\} \times \Omega \to \mathbb{R}^{d'}$ satisfying

$$Y_k = h(X_{t_k}) + V_k, \quad k \in \{0,1,\dots,K\}, \tag{1.2}$$

for a grid of observation times $0 \leq t_0 < t_1 < \cdots < t_K \leq T$. We assume that the measurement noise is Gaussian according to $V_k \sim \mathrm{N}(0,\Sigma)$ for all $k$. See Figure 1.2 for an illustration of the state-observation model, where the arrows describe conditional dependence in the sense of Bayesian networks.

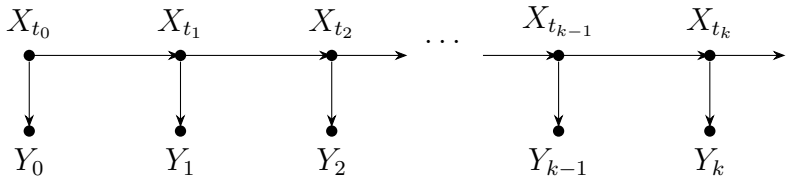


**Figure 1.2:** A simple overview of the state-observation model used in this thesis.

In filtering, the main task is to retrieve the conditional distribution of the state, given currently available observations. The objects of interest are $p(X_{t_k} \mid Y_{0:k})$ for $k = 0,\dots,K$. Similar problems concern smoothing, where $p(X_{t_k} \mid Y_{0:K})$ for $k = 0,\dots,K$ are of interest and prediction, where $p(X_t \mid Y_{0:k})$ for some $t > t_k$ is of interest. Another problem that often arises in applications is parameter estimation. Let $\theta$ be some parameters that influence the dynamics of the state. The objective of parameter estimation is to infer $p(\theta \mid Y_{0:k})$. These similar problems are not focused

on in this thesis. Instead the reader is referred to [35] for a comprehensive treatise.

In EBDS for discrete observations, the approach is to form the filtering density recursively using Bayes' law:

$$p(X_{t_k} \mid Y_{0:k}) = \frac{p(Y_k \mid X_{t_k}, Y_{0:k-1})\, p(X_{t_k} \mid Y_{0:k-1})}{p(Y_k \mid Y_{0:k-1})} \propto p(Y_k \mid X_{t_k})\, p(X_{t_k} \mid Y_{0:k-1}).$$

The main difficulty lies in calculating the predictive distribution $p(X_{t_k} \mid Y_{0:k-1})$ using the previous filtering distribution $p(X_{t_{k-1}} \mid Y_{0:k-1})$. This is done by approximately solving the Fokker–Planck equation. Deep learning is the method of choice since it has the potential to solve the PDE accurately in a reasonable time. For comparison, the Kalman family of filters rely on normality assumptions to calculate the predictive distribution together with different methods for estimating moments whereas particle filters use Monte Carlo techniques. Further details on these filters are presented in Chapter 4 while EBDS is treated in Chapter 5.

## 1.3 Applications

Due to its general formulation, the filtering problem occurs in a huge number of settings. Some example applications of linear and nonlinear filtering include

- **Target tracking**. Filtering is especially important to improve estimates of position and velocity in the field of target tracking, as observations often come from video, sonar, radar or lidar systems and contain large amounts of noise. In [14], the authors compare the performance of several classical nonlinear filtering algorithms when tracking ground targets using radar measurements.
- **Finance and Economics**. In filtering applications from finance and economics, the state often represents variables that are difficult to observe directly such as inflation, instantaneous interest rates or volatilities. The observations can be for instance prices of traded instruments, which are readily available. A vast number of example applications can be found in [47]. See [15] for a shorter overview.
- **Predictive maintenance**. If the state is made to represent a degradation process, filtering can be used to estimate remaining lifetime of system components. See for example [30], in which a general model is presented and applied to fatigue crack growth prediction. In [28], filtering is used for a maintenance policy when the sensor itself degrades as well as the monitored system.
- **Biomedical Engineering**. Filtering is widely utilised to process measurement data in biomedical engineering. In [16], ensemble Kalman filtering is used to improve the resolution of data from 4D Flow MRI. In [46] nonlinear filtering is used to process data from wearable devices, allowing for accurate monitoring of subject heart rates. A final example from literature is [40], where a particle filter is proposed to track harmonics in rhythmical signals such as blood pressure and electrocardiogram signals.

- **Epidemiology**. Filtering can also be used in the modelling of population dynamics in epidemiology. See for instance [3], in which the authors use an extended Kalman filter in a model for covid-19 transmission.
- **Weather forecasting**. Filtering can play a major role in weather forecasting systems and is used to improve the output of Numerical Weather Prediction (NWP) models. See [8], [9] and [18] for discussion on the subject.
- **Telecommunications**. Filtering is very useful for handling various signals in telecommunications. In [44] particle filters are used to counter frequency hopping, with applications in signals intelligence. Other examples from literature include [37], in which nonlinear filtering is used for a signal receiver algorithm, and [36], in which filtering is part of an algorithm to predict interference in networks.

# 2

# Concepts from Stochastic Analysis

As the state dynamics are described by a Stochastic Differential Equation (SDE) in the setting of EBDS, stochastic analysis is necessary in its derivation. This chapter treats the most important results relevant to this thesis and introduces some notation. Some prior knowledge in measure theoretic probability theory and stochastic processes is assumed. For a thorough introduction to the subject, see for instance [25] or [31]. We start the chapter by introducing some notation, in particular from functional analysis. In the following section, stochastic differential equations are introduced. Then, the Markov theory for diffusion processes is explained, after which numerical SDE solutions are treated.

## 2.1 Preliminary Elements and Notation

Denote the set of all functions $f\colon \mathbb{R}^n \to \mathbb{R}^m$ with continuous derivatives up to order $k$ by $C^k(\mathbb{R}^n;\mathbb{R}^m)$. For $k = 0$ the short-hand notation $C(\mathbb{R}^n;\mathbb{R}^m)$ is used. For a function $g\colon \mathbb{R}^n \times \mathbb{R}^{n'} \to \mathbb{R}^m$ with continuous derivatives up to order $k$ for the first argument and up to order $k'$ for the second argument, the notation $C^{k,k'}(\mathbb{R}^n \times \mathbb{R}^{n'};\mathbb{R}^m)$ is used. No assumption is then made on the order of the mixed derivatives. For functions from $\mathbb{R}^n \to \mathbb{R}$ that vanish as some variable tends to infinity a subscript 0 is added according to $C_0^k(\mathbb{R}^n;\mathbb{R})$. A function is said to be smooth if it is differentiable enough for the result at hand.

With the notation $\|\cdot\|_Y$ and $\langle\cdot,\cdot\rangle_Y$, we mean the norm and scalar product on a vector space $Y$. The notation $\|\cdot\|$ and $\langle\cdot,\cdot\rangle$ is reserved for the standard euclidean norm and scalar product respectively. In the case of matrices we let $\|\cdot\|$ denote the Frobenius norm. For a general measure space $(X, \mathcal{M}, \mu)$ and normed vector space $Y$, let $L^p(X;Y)$, $1 \leq p < \infty$ denote the set of all equivalence classes of measurable functions $f\colon X \to Y$ such that

$$\|f\|_{L^p(X;Y)} := \left(\int_X \|f\|_Y^p \,\mathrm{d}\mu\right)^{1/p} < \infty.$$

To clarify, two functions $f$ and $g$ belong to the same equivalence class and are thus considered the same element of $L^p(X;Y)$, if $\|f - g\|_{L^p(X;Y)} = 0$. This means that they can differ on a set of $\mu$-measure zero. In particular, define $L^p(\mathbb{R}^n;\mathbb{R}^m)$ as the set of equivalence classes of functions $f\colon \mathbb{R}^n \to \mathbb{R}^m$ with

$$\|f\|_{L^p(\mathbb{R}^n;\mathbb{R}^m)} := \left(\int_{\mathbb{R}^n} \|f(x)\|^p \,\mathrm{d}x\right)^{1/p} < \infty.$$

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. Another important $L^p$-space is $L^p(\Omega; \mathbb{R}^m)$, the set of equivalence classes of random variables $Z\colon \Omega \to \mathbb{R}^m$ with

$$\|Z\|_{L^p(\Omega;\mathbb{R}^m)} := \left(\int_\Omega \|Z\|^p \,\mathrm{d}\mathbb{P}\right)^{1/p} = \left(\mathbb{E}[\|Z\|^p]\right)^{1/p} < \infty.$$

We also have reason to define so called Bochner spaces $L^p(\Omega; L^{p'}(\mathbb{R}^n; \mathbb{R}^m))$, which consist of all equivalence classes of functions $g\colon \Omega \to L^{p'}(\mathbb{R}^n; \mathbb{R}^m)$ satisfying

$$\begin{aligned}\|g\|_{L^p(\Omega;L^{p'}(\mathbb{R}^n;\mathbb{R}^m))} &:= \left(\int_\Omega \|g\|^p_{L^{p'}(\mathbb{R}^n;\mathbb{R}^m)} \,\mathrm{d}\mathbb{P}\right)^{1/p} \\ &= \left(\mathbb{E}\left[\left(\int_{\mathbb{R}^n} \|g(x)\|^{p'} \,\mathrm{d}x\right)^{p/p'}\right]\right)^{1/p} < \infty.\end{aligned}$$

In the case of $p = \infty$, the space $L^\infty(X; Y)$ is the set of all equivalence classes of functions $f\colon X \to Y$ with

$$\|f\|_{L^\infty(X;Y)} := \sup_X \|f\|_Y < \infty.$$

Here, by sup we mean the essential supremum, although this is not written explicitly.

## 2.2 Stochastic Differential Equations

Stochastic differential equations are essential for constructing dynamic models in the presence of process noise. They are fundamental building blocks within financial mathematics, control theory, mathematical biology and physics. There are numerous types of stochastic differential equations, with different formulas and applications. In general, one can also include jump terms that capture discontinuities in the dynamics. Here, the theory is only presented for a specific type suitable for our needs.

We use a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with a filtration $\mathbb{F} = (\mathcal{F}_t)_{t\in[0,T]}$ and assume that it is complete. Let $\mu\colon [0,T] \times \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma\colon [0,T] \times \mathbb{R}^d \to \mathbb{R}^{d\times d}$ be measurable functions. Finally, let $W\colon [0,T] \times \Omega \to \mathbb{R}^d$ be a $d$-dimensional Brownian motion that is $\mathbb{F}$-adapted. A diffusion type stochastic differential equation is an equation of the form

$$X_t = X_0 + \int_0^t \mu(s, X_s) \,\mathrm{d}s + \int_0^t \sigma(s, X_s) \,\mathrm{d}W_s, \quad t \in [0,T].$$

A solution $X\colon [0,T] \times \Omega \to \mathbb{R}^d$ is called a diffusion process. At time $t = 0$, $X_0$ is $\mathcal{F}_0$-measureable and independent of $W$, but possibly random. From here, we assume that $X_0$ has a density $p_0(x)$. Further simplifications can be made by assuming that the stochastic differential equation is time-homogeneous. It is then of the form

$$X_t = X_0 + \int_0^t \mu(X_s) \,\mathrm{d}s + \int_0^t \sigma(X_s) \,\mathrm{d}W_s, \quad t \in [0,T], \tag{2.1}$$

with $\mu\colon \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma\colon \mathbb{R}^d \to \mathbb{R}^{d\times d}$. All results from here are presented for time-homogeneous SDEs.

There are two forms of solutions for SDEs. Let $\mathbb{F}^W$ be the filtration generated by $W$. Then, a strong solution to (2.1) is an $\mathbb{F}^W$-adapted process that satisfies the equation $\mathbb{P}$-a.s.. A weak solution, on the other hand, satisfies the equation for some Brownian motion $W$ and is adapted to the filtration generated by that $W$. Intuitively, the weak solution has the dynamics specified by the SDE, but is not driven by the particular Brownian motion. There are many results that guarantee the existence and uniqueness of strong and weak solutions to (2.1) under certain conditions. One well known such result is presented in Proposition 2.1.

**Proposition 2.1** (Existence and uniqueness of strong solution)**.** *Let $K_1, K_2 \in \mathbb{R}_{>0}$ be two constants. Under the regularity assumptions*

$$\mathbb{E}\left[\|X_0\|^2\right] < \infty,$$
$$\|\mu(x) - \mu(y)\| + \|\sigma(x) - \sigma(y)\| \leq K_1 \|x - y\|, \quad x, y \in \mathbb{R}^d, \tag{2.2}$$
$$\|\mu(x)\| + \|\sigma(x)\| \leq K_2(1 + \|x\|), \quad x \in \mathbb{R}^d, \tag{2.3}$$

*the stochastic differential equation* (2.1) *has a unique strong solution.*

Assumption (2.2) is called a global Lipschitz condition while the assumption (2.3) is a linear growth condition.

The time-homogeneous SDE (2.1) is associated with an operator $A$ containing the information of the equation. This is called the infinitesimal generator. Define $a(x) = \sigma(x)\sigma(x)^T$. For $f \in C_0^\infty(\mathbb{R}^d; \mathbb{R})$ the operator is given by

$$(Af)(x) = \sum_{i=1}^{d} \mu_i(x) \frac{\partial f(x)}{\partial x_i} + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} a_{ij}(x) \frac{\partial^2 f(x)}{\partial x_i \partial x_j}.$$

The infinitesimal generator has an adjoint $A^*$ on $L^2(\mathbb{R}^d; \mathbb{R})$, so that $\langle Af, g \rangle_{L^2(\mathbb{R}^d;\mathbb{R})} = \langle f, A^* g \rangle_{L^2(\mathbb{R}^d;\mathbb{R})}$. For $f \in C_0^\infty(\mathbb{R}^d; \mathbb{R})$, the adjoint is given by

$$(A^* f)(x) = -\sum_{i=1}^{d} \frac{\partial}{\partial x_i}\Big(\mu_i(x) f(x)\Big) + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \frac{\partial^2}{\partial x_i \partial x_j}\Big(a_{ij}(x) f(x)\Big). \tag{2.4}$$

These operators are essential in the Markov theory for diffusion processes and for the Kolmogorov backward and forward equations respectively, see below. Moreover, $A$ can be used to write Itô's formula in a concise way. This is done in Proposition 2.2. For a proof of this proposition, see for instance Theorem 3.3 in [33].

**Proposition 2.2** (Itô's formula)**.** *Under the same assumptions as in Proposition 2.1, for any $f \in C^{1,2}([0, T] \times \mathbb{R}^d; \mathbb{R})$ and $X$ solving* (2.1) *it holds that*

$$f(t, X_t) = f(0, X_0) + \int_0^t \left( \frac{\partial f}{\partial s}(s, X_s) + Af(s, X_s) \right) \mathrm{d}s + \int_0^t \langle \nabla_x f(s, X_s), \sigma(X_s)\, \mathrm{d}W_s \rangle.$$

While it sometimes makes sense to work directly with the underlying stochastic differential equation, often it can be advantageous to solve for the probability distribution of the solution instead. This is the approach used in EBDS for the filtering problem. As such, some background and important results in this area are presented next.

## 2.3 Markov Theory for Diffusion Processes

A process $X\colon [0,T]\times\Omega\to\mathbb{R}^d$ is said to have the Markov property if for all Borel sets $B$

$$\mathbb{P}(X_t\in B\mid\mathcal{F}_s^X)=\mathbb{P}(X_t\in B\mid\mathfrak{S}(X_s)),\quad 0\le s\le t\le T.$$

Here $\mathbb{F}^X=(\mathcal{F}_t^X)_{t\in[0,T]}$ is the filtration generated by $X$ and $\mathfrak{S}(X_t)$ is the $\sigma$-algebra generated by $X_t$. The intuitive interpretation is that the future behaviour of $X$ only depends on its present state. If a solution exists to (2.1), it is necessarily a Markov process. This can be seen by reformulating (2.1) as

$$X_t=X_s+\int_s^t\mu(X_\tau)\,\mathrm{d}\tau+\int_s^t\sigma(X_\tau)\,\mathrm{d}W_\tau,\quad 0\le s\le t\le T.$$

Now, $X_s$ is both $\mathcal{F}_s^X$-measurable and $\mathfrak{S}(X_s)$-measurable. Both integral terms are independent of $\mathcal{F}_s^X\supset\mathfrak{S}(X_s)$.

For processes with the Markov property, it makes sense to define a transition probability $\mathbb{P}(X_t\mid\mathfrak{S}(X_s))$ for $s\le t$. If a process is time-homogeneous, this probability only depends on the difference $t-s$. From here, assume that the setting is regular enough so that the transition probability is given by a density $p_{t-s}(x,y)$ according to

$$\mathbb{P}(X_t\in B\mid X_s=y)=\int_{x\in B}p_{t-s}(x,y)\,\mathrm{d}x,\tag{2.5}$$

for any Borel set $B$. See for instance Hörmander's theorem for conditions that guarantee the existence of such a density. Using the operator $A^*$ in (2.4), a partial differential equation for the transition density can be formulated, which is done next.

Assume that there is a density $p\colon[0,T]\times\mathbb{R}^d\times\mathbb{R}^d\to\mathbb{R}$ satisfying (2.5) and that this density is sufficiently smooth in the forward variable $x$ and time. Also assume that $\mu$ and $\sigma$ in (2.1) are sufficiently smooth. Then the transition density of $X$, the unique solution to (2.1), satisfies the partial differential equation

$$\frac{\partial p_t(x,y)}{\partial t}=A^*p_t(x,y),\tag{2.6}$$

known as the Fokker–Planck equation, with initial condition

$$p_0(x,y)=\delta(x-y).\tag{2.7}$$

For a proof in the one-dimensional case, see Theorem 2.2 in [31]. The multidimensional case can be done similarly. In mathematics, the Fokker–Planck equation is

commonly referred to as the Kolmogorov forward equation.

Next, let $p_t(x)$ be the (unconditional) density of $X_t$ and assume that it exists. We now derive the Fokker–Planck equation for this unconditional density, which is a more common formulation than the one for the transition density in (2.6)–(2.7). To derive this formulation, the Chapman–Kolmogorov equation is needed. This equation says that for $s \leq t$

$$p_t(x) = \int_{\mathbb{R}^d} p_{t-s}(x, y)\, p_s(y)\, \mathrm{d}y. \tag{2.8}$$

In the following short derivation of the Fokker–Planck PDE for the unconditional density some theoretical details are left out for brevity.

To start, assume that the solution $X$ of (2.1) has a density $p\colon [0, T] \times \mathbb{R}^d \to \mathbb{R}$ and that the density is given at some time $s$ as $p(x)$. For $t \geq s$, according to (2.6), we have

$$\frac{\partial p_{t-s}(x, y)}{\partial t} = A^* p_{t-s}(x, y).$$

Multiplying with $p_s(y)$ on both sides and integrating yields

$$\int_{\mathbb{R}^d} \frac{\partial}{\partial t} p_{t-s}(x, y)\, p_s(y)\, \mathrm{d}y = \int_{\mathbb{R}^d} A^* p_{t-s}(x, y)\, p_s(y)\, \mathrm{d}y.$$

Now, by changing the order of integration and differentiation and applying the Chapman–Kolmogorov equation (2.8), we arrive at

$$\frac{\partial p_t(x)}{\partial t} = A^* p_t(x), \quad t \geq s. \tag{2.9}$$

The initial condition $p_s(x) = p(x)$ follows from the Chapman–Kolmogorov equation and the initial condition for the transition density (2.7). Note that under certain regularity assumptions one can guarantee the existence and uniqueness of a solution to the Fokker-Planck equation (2.9). See for instance Theorem 4.1 in [31].

As an extension of (2.9), it is clear that if the distribution of $X_t$ conditioned on some event is of interest, one only has to replace the initial condition. For example, let $p(x \mid z)$ be the density of $X_s$ conditioned on some variable $Z = z$. Then for $t \geq s$, the conditional distribution of $X_t$ satisfies

$$\begin{aligned} \frac{\partial p_t(x \mid z)}{\partial t} &= A^* p_t(x \mid z), \\ p_s(x \mid z) &= p(x \mid z). \end{aligned} \tag{2.10}$$

This fact plays a crucial role in the derivation of EBDS in Chapter 5, as we can use the Fokker–Planck equation to model the time evolution of the predictive distribution for the state conditioned on all available measurements.

## 2.4 Numerical Methods

Closed expressions for solutions to stochastic differential equations are rarely available. Instead, one needs to resort to numerical methods in most cases. In this study, we use the Euler–Maruyama method to simulate from (2.1). This is done using the scheme

$$\begin{aligned} &\bar{X}_{n+1} = \bar{X}_n + \mu(\bar{X}_n)\,(t_{n+1} - t_n) + \sigma(\bar{X}_n)\,(W_{t_{n+1}} - W_{t_n}), \quad n \in \{0, \ldots, N-1\}, \\ &\bar{X}_0 \sim p_0. \end{aligned} \tag{2.11}$$

Here, $0 = t_0 < t_1 < \cdots < t_N = T$ is a discrete grid and we note that $(W_{t_{n+1}} - W_{t_n}) \sim \mathrm{N}(0, I_d\,(t_{n+1} - t_n))$. The idea is that $\bar{X}_n \approx X_{t_n}$, which is made formal through Theorem 2.1. For the statement we need a continuous interpolation of the Euler–Maruyama approximation to all $t \in [0, T]$ given by

$$\bar{X}(t) = \bar{X}_n + \mu(\bar{X}_n)\,(t - t_n) + \sigma(\bar{X}_n)\,(W_t - W_{t_n}), \quad t \in [t_n, t_{n+1}). \tag{2.12}$$

We note that $\bar{X}(t_n) = \bar{X}_n$ for all $n$. Additionally, define $h$ as the maximum time step in the grid according to

$$h = \max_{n \in \{0,\ldots,N-1\}} t_{n+1} - t_n.$$

No proof is presented for Theorem 2.1 here, instead the reader is referred to Theorem 10.2.2 in [26].

**Theorem 2.1** (Strong Convergence of the Euler–Maruyama scheme)**.** *Let $X_t$ be given by* (2.1) *and $\bar{X}(t)$ by* (2.11)–(2.12)*. Assume that the regularity conditions for Proposition 2.1 apply with constants $K_1$, $K_2$ independent of $h$ and assume that*

$$\mathbb{E}\left[\left\|\bar{X}(0) - X_0\right\|^2\right] \leq K_3\, h,$$

*with $K_3 \in \mathbb{R}_{>0}$ independent of $h$. Then, for $C \in \mathbb{R}_{>0}$ independent of $h$*

$$\sqrt{\mathbb{E}\left[\sup_{0 \leq s \leq T} \left\|\bar{X}(s) - X_s\right\|^2\right]} \leq C\, h^{1/2}.$$

The bound in Theorem 2.1 for the Euler–Maruyama method is referred to as strong convergence of order 1/2. Note that as a trivial consequence, at a certain time $t$, $\bar{X}(t)$ converges to $X_t$ in the $L^2$-sense, i.e.,

$$\lim_{h \searrow 0} \|\bar{X}(t) - X_t\|_{L^2(\Omega;\mathbb{R}^d)} = \lim_{h \searrow 0} \sqrt{\mathbb{E}\left[\left\|\bar{X}(t) - X_t\right\|^2\right]} = 0.$$

The Euler–Maruyama method comes from the first order truncation of the so called Itô–Taylor expansion. Higher order methods can be derived by adding more terms. The most famous such method is the Milstein scheme, which generally has strong convergence of order 1. Implementation of this scheme in more than one dimension

is not easy though and is not done in this thesis. There are also certain stochastic differential equations, called stiff equations, that cannot be simulated in a stable way using explicit methods such as the Euler–Maruyama method or the Milstein method. One can then resort to implicit versions of these schemes. Once again, such methods are not used here however. For a longer treatise on higher order methods, implicit methods and numerical methods for stochastic differential equations in general, see [26].

# 3
# Concepts from Machine Learning

Neural networks are models inspired by the connectivity and simplistic function of of neurons in biology. During the last decades, they have proved to be tremendously capable for most tasks in machine learning. In this thesis, neural networks are used for a regression task with supervised learning. The goal is to train a neural network to minimise a given loss function. A vast amount of material on the subject of neural networks can be found by searching online, yet for completeness a short overview is presented here. The chapter starts with a description of the standard fully-connected architecture, after which long short-term memory networks are treated.

## 3.1 Neural Networks

A neural network consists of a series of layers, where each layer consists of several parallel neurons. A classic illustration is presented in Figure 3.1. In a standard fully-connected layer, each neuron is connected to all neurons in the previous layer. The i-th neuron in a layer is a function $f^i_{w_i,b_i}\colon \mathbb{R}^\ell \to \mathbb{R}$, parameterised by learnable weights $w_i \in \mathbb{R}^\ell$ and bias $b_i \in \mathbb{R}$. Here $\ell$ is the size of the previous layer. Given the output $z$ from the previous layer, the neuron performs the operation

$$f^i_{w_i,b_i}(z) = f(w_i^T z + b_i).$$

The function $f\colon \mathbb{R} \to \mathbb{R}$ is called the activation function. A typical choice for intermediate layers is the ReLU activation function $f(u) = \max(u, 0)$. For regression tasks, the activation in the final layer is typically the identity $f(u) = u$.

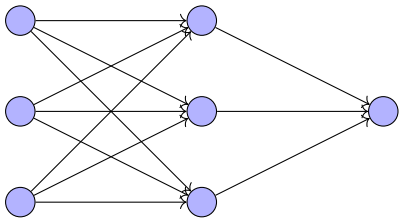

**Figure 3.1:** A very classic illustration of a small fully-connected neural network with two layers, input dimension 3 and output dimension 1. Neurons are drawn in blue, with connections in black. The first column of nodes represents input data.

Each layer then represents a function $L_{W,b}\colon \mathbb{R}^\ell \to \mathbb{R}^{\ell'}$ with weights $W = (w_1, w_2, \ldots, w_{\ell'})^T$ and biases $b = (b_1, b_2, \ldots, b_{\ell'})^T$. Explicitly, the layer is

$$L_{W,b}(z) = (f^1_{w_1,b_1}(z), f^2_{w_2,b_2}(z), \ldots, f^{\ell'}_{w_{\ell'},b_{\ell'}}(z))^T = f(Wz + b).$$

Here, the activation $f$ acts component-wise.

Now, denote the j-th layer in a network as $L^j_{\theta_j}$, where $\theta_j$ is a concatenation of the weights and biases in the layer. Let the number of layers be $L$. The network itself is a function $\mathcal{N}_\theta \colon \mathbb{R}^n \to \mathbb{R}^m$ parameterised by $\theta = (\theta_1, \theta_2, \ldots, \theta_L)$ satisfying

$$\mathcal{N}_\theta(x) = (L^L_{\theta_L} \circ \cdots \circ L^2_{\theta_2} \circ L^1_{\theta_1})(x).$$

All layers except layer $L$ are referred to as hidden layers. Typically, if there are more than two hidden layers, i.e., if $L > 2$, then the network is said to be deep.

To learn the parameters $\theta$, a so called loss function must be defined. In regression, the goal is for the network to learn the relationship between two random variables $X$ and $Y$ with a joint distribution $(X, Y) \sim Q$. The most natural interpretation of this is to learn to estimate $f^*(X) = \mathbb{E}[Y \mid X]$, which is the unique minimiser of

$$\|f(X) - Y\|^2_{L^2(\Omega;\mathbb{R}^m)} = \mathbb{E}[\|f(X) - Y\|^2]. \tag{3.1}$$

As $Q$ is almost always unknown, this expectation is rarely available directly. Instead, in supervised learning for regression, the network is trained on a data set consisting of input-label pairs $(x_i, y_i)_{i=1}^N$ sampled from $Q$. The Mean Squared Error (MSE) of the network on the data set is defined as

$$\mathrm{MSE}_\theta = \frac{1}{N} \sum_{i=1}^{N} \|\mathcal{N}_\theta(x_i) - y_i\|^2. \tag{3.2}$$

It is a Monte Carlo approximation of the $L^2$-error in (3.1). As such, the MSE is the most common loss function for regression tasks. The problem of training is formulated as finding optimal weights and biases $\theta^* = \operatorname{argmin}_\theta \mathrm{MSE}_\theta$.

Most commonly, the optimisation algorithm used to find optimal parameters is gradient descent, in which the parameters are updated as

$$\theta^{(j+1)} = \theta^{(j)} - \alpha \nabla_\theta \mathrm{MSE}_\theta.$$

The learning rate $\alpha$ controls how fast the parameters are updated. Due to the structure of the network, the gradient with respect to the parameters in layer $i$, $\nabla_{\theta_i}$, can be calculated recursively using $\nabla_{\theta_{i+1}}$. This is known as backpropagation and contrasts the forward propagation that occurs when predictions are made.

By calculating the gradient of the objective function on only some small part of the data, called a batch, one often arrives at better weights. One training cycle, called epoch, then consists of looping through and updating weights for each batch. This is known as (mini) batch gradient descent. If this is done separately for each data point, with batch size 1, then the famous Stochastic Gradient Descent (SGD) algorithm is obtained. Numerous extensions and adaptations of gradient descent exist. A notable scheme that often performs well in practice is the ADaptive Moment Estimation (ADAM) optimiser [24]. The idea behind this algorithm is to adapt the

learning rate for each parameter while also using momentum to speed up convergence.

The number of layers, activation function, number of neurons in each layer, batch size, learning rate and similar parameters are called hyperparameters. These need to be specified before training, but are often tuned to obtain better performance. For this purpose, the available data set is often split into a training set, a validation set and a test set. The hyperparameters with the best result on the validation partition are chosen as optimal, after which the test partition is used to estimate unbiased performance.

As a final note, typically overfitting is a major challenge in the context of supervised learning. This occurs when the model adapts excessively to the training data but loses generality. There exists many methods to prevent this, some of which include adding regularising terms to the loss function, utilising dropout layers and stopping the training early. See [6] for a review of various techniques to avoid overfitting.

## 3.2 Long Short-Term Memory Networks

There are many variations of the classic fully-connected neural networks presented in the last section. These different architectures vary in what tasks they have been designed for. Recurrent Neural Networks (RNNs) are particularly useful for the tasks performed in this thesis. Fundamentally, they are designed to handle the dependencies that arise with time series data. Instead of treating input samples as i.i.d., recurrent neural networks allow previous input instances to affect predictions on future ones.

A particular type of RNN that has become increasingly popular is the Long Short-Term Memory (LSTM) network. Crucially, it allows for learning dependencies far back in time by avoiding vanishing gradients. In this thesis, the motivation behind using LSTMs is to investigate if more exotic architectures can reduce the error and help produce a better filter. Here, the main components of these networks are highlighted, for a more comprehensive treatise see for instance [38].

An LSTM network can be viewed as a map $(x_1, \dots, x_T) \mapsto (h_1, \dots, h_T)$, consisting of $T$ cells. At a time $t$, the cell receives an input $x_t$ and provides an output $h_t$. The output $h_{t-1}$ from the previous cell is also used for the prediction. Moreover, the network stores a cell state, $c_t$, that acts as another input to the LSTM. At time $t$ the model operates on $c_{t-1}$. The LSTM cell consists of four components, called gates, a classic illustration is shown in Figure 3.2. Let $\sigma\colon \mathbb{R}^n \to \mathbb{R}^n$ denote the sigmoid activation function that for each component $i$ performs $x_i \mapsto 1/(1+\exp(-x_i))$ and let $\tanh\colon \mathbb{R}^n \to \mathbb{R}^n$ be the hyperbolic tangent applied in the same way. For weight matrices $W_{if}$, $W_{hf}$ and biases $b_{if}$, $b_{hf}$, the so called forget gate performs

$$f_t = \sigma(W_{if}\, x_t + b_{if} + W_{hf}\, h_{t-1} + b_{hf}).$$

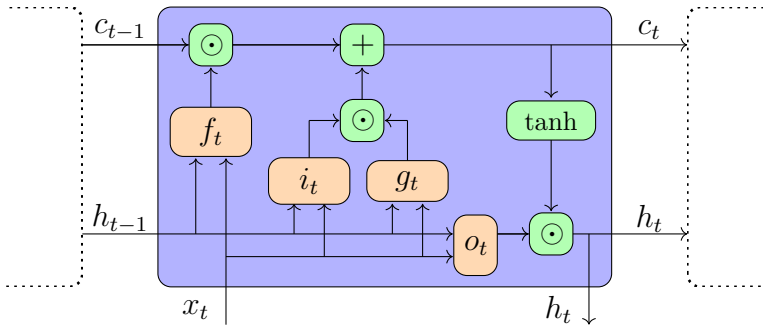


**Figure 3.2:** An illustration showing the different components in an LSTM cell. The gates are marked in orange. Besides the input $x_t$, the cell operates on the cell state $c_{t-1}$ and output $h_{t-1}$ from the previous cell.

For weights $W_{ii}$, $W_{hi}$ and biases $b_{ii}$, $b_{hi}$ the input gate performs

$$i_t = \sigma(W_{ii}\, x_t + b_{ii} + W_{hi}\, h_{t-1} + b_{hi}).$$

Similarly, for the cell gate

$$g_t = \tanh(W_{ig}\, x_t + b_{ig} + W_{hg}\, h_{t-1} + b_{hg}).$$

These three are then used to update the cell state. Let $\odot$ be the Hadamard product. The update is done with the operation

$$c_t = f_t \odot c_{t-1} + g_t \odot i_t.$$

The intuition is that $f_t \in [0,1]^n$ controls what parts of the long-term memory to forget while the second term adds new information to the long-term memory, with $i_t \in [0,1]^n$ modulating what to save. Finally, the output $h_t$ is formed through

$$\begin{aligned} o_t &= \sigma(W_{io}\, x_t + b_{io} + W_{ho}\, h_{t-1} + b_{ho}), \\ h_t &= o_t \odot \tanh(c_t), \end{aligned}$$

where $o_t$ denotes the output gate.

In a similar way as fully-connected networks, LSTM networks are trained using backpropagation. However, the temporal dependence complicates the procedure and the resulting algorithm is known as Backpropagation Through Time (BPTT). The reader is referred to [10] for further details.

Standard LSTM networks operate on a sequence of inputs $(x_1, \ldots, x_T)$ to output a sequence $(h_1, \ldots, h_T)$. In certain applications only a final output might be of interest and the network can then be viewed as a map $(x_1, \ldots, x_T) \mapsto h_T$. Moreover, additional layers can be concatenated to the LSTM cells to process $h_t$ further. One can even add intermediate cells between each input. The particular adaptation of LSTM networks used in this thesis is detailed further in Chapter 6.

# 4

# Bayesian Filtering Methods

In this chapter, the most common methods for Bayesian filtering are presented and discussed. These classical filtering methods are used to benchmark EBDS in Chapter 7. We begin by reformulating the continuous time model used for EBDS to a discrete time one. Then, three types of Kalman filters are presented, after which particle filters are treated.

## 4.1 Discrete Time State Model

Most classical methods within Bayesian filtering utilise a discrete time dynamic model for the state instead of the continuous time one defined in (1.1). Using the Euler–Maruyama scheme defined in (2.11), it is possible to translate a continuous time model into a discrete time model. If the time steps are small and the dynamics are regular enough (see Theorem 2.1), then the discretisation error will be small. As such, the Bayesian filtering methods presented below can be compared to EBDS in spite of the difference in underlying model.

In the literature on Bayesian filtering, most commonly the grid in the state model is defined based on the measurement times and no intermediate steps are assumed. To be able to control the discretisation error when replacing a continuous time model with a discrete time model, we allow the state model grid to be finer. Assume that the observations are made in times $t_0, \dots, t_K$. Each interval $[t_k, t_{k+1}]$ is then partitioned into a new grid $t_k = t_{k,0} < \cdots < t_{k,N} = t_{k+1}$.

In many scenarios a model with additive process and measurement noise is sufficient. Let $f_{k,n} \colon \mathbb{R}^d \to \mathbb{R}^d$ and $q_{k,n} \sim \mathrm{N}(0, Q_{k,n})$ for $n \in \{0 \dots N-1\}$. The model for the state is then

$$X_{t_{k,n+1}} = f_{k,n}(X_{t_{k,n}}) + q_{k,n}, \quad n \in \{0 \dots N-1\},\ k \in \{0 \dots K-1\}.$$

For the observations, let $h_k \colon \mathbb{R}^d \to \mathbb{R}^{d'}$ and $r_k \sim \mathrm{N}(0, R_k)$ for $k \in \{0 \dots K\}$. The model for the measurement process is

$$Y_k = h_k(X_{t_k}) + r_k, \quad k \in \{0 \dots K\}. \tag{4.1}$$

The model with additive noise is often not general enough, however, as it is the discrete time equivalent of assuming a constant diffusion coefficient in the continuous

time dynamic model. Instead, redefine $f_{k,n}\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$. A more general model for the state is then

$$X_{t_{k,n+1}} = f_{k,n}(X_{t_{k,n}}, q_{k,n}), \quad n \in \{0 \dots N-1\},\ k \in \{0 \dots K-1\}. \tag{4.2}$$

The measurement model with additive noise in (4.1) is adequate for our purposes. However, for completeness the Bayesian filters in this chapter are presented using the more general measurement model

$$Y_k = h_k(X_{t_k}, r_k), \quad k \in \{0 \dots K\}, \tag{4.3}$$

where we redefine $h_k\colon \mathbb{R}^d \times \mathbb{R}^{d'} \to \mathbb{R}^{d'}$.

Using the joint state-measurement model defined in (4.2)–(4.3), various Bayesian filtering algorithms are introduced in the sections below. First, the Kalman filter and some of its extensions are presented after which particle filters are treated.

## 4.2 Kalman Filters

Named after engineer and mathematician Rudolf E. Kálmán, the Kalman filter has been instrumental for filtering since its invention in the 1960s. Notably, Kalman filtering was used in the navigation system of the Apollo missions [39]. While the key disadvantage of the classical Kalman filter is that it can only be applied when the model is linear, several modifications exist. In this section, we begin by presenting the standard Kalman filter. Thereafter the extended and unscented variants are treated and general Gaussian filters are discussed in short.

### 4.2.1 Classical Kalman Filter

A Kalman filter can only be used when the dynamics of the model are linear. Let $A_{k,n} \in \mathbb{R}^{d\times d}$ and $q_{k,n} \sim \mathrm{N}(0, Q_{k,n})$ for $n \in \{0 \dots N-1\}$, $k \in \{0 \dots K-1\}$. The model for the state is of the form

$$X_{t_{k,n+1}} = A_{k,n}\, X_{t_{k,n}} + q_{k,n}. \tag{4.4}$$

Moreover, $X_0 \sim p_0$ needs to be Gaussian and the measurement needs to be of the form

$$Y_k = H_k\, X_{t_k} + r_k, \tag{4.5}$$

where $H_k \in \mathbb{R}^{d'\times d}$ and $r_k \sim \mathrm{N}(0, R_k)$. In the framework of (4.2)–(4.3), this means that

$$\begin{aligned} f_{k,n}(x, q) &= A_{k,n}\, x + q, \\ h_k(x, r) &= H_k\, x + r. \end{aligned}$$

Note that (1.1) with the Euler–Maruyama scheme in (2.11) can be written on the required form if $\mu(x)$ only has linear terms and $\sigma(x)$ is constant. It is also required

that $h(x)$ in (1.2) only has linear terms. If the exact model is given by (4.4)–(4.5), then the Kalman filter gives the exact solution to the filtering problem. If the exact model is given by a diffusion type stochastic differential equation of the required type, then the Kalman filter is exact with respect to the discretised model.

The Kalman filtering algorithm consists of two main steps, prediction and update, which are iterated. The prediction step yields the predictive distribution

$$p(X_{t_{k,n}} = x \mid Y_{0:k} = y_{0:k}) = \mathrm{N}(x \mid m_{k,n}^-, P_{k,n}^-), \quad n \in \{1, \ldots, N\},\ k \in \{0, \ldots, K\}. \tag{4.6}$$

Formulas for the mean $m_{k,n}^-$ and covariance $P_{k,n}^-$ are presented in Theorem 4.1. The prediction in $t_{k-1,N}$ is used by the update step to form the filtering distribution

$$p(X_{t_k} = x \mid Y_{0:k} = y_{0:k}) = \mathrm{N}(x \mid m_k, P_k), \quad k \in \{0, \ldots, K\}, \tag{4.7}$$

where the mean $m_k$ and covariance $P_k$ are computed according to Theorem 4.2. For proof of Theorem 4.1 and 4.2, see Theorem 4.2 in [35].

**Theorem 4.1** (Kalman filter prediction step)**.** *Let the state-measurement model be given by* (4.4)–(4.5)*. If the prior $p_0(x)$ is Gaussian, then the exact predictive distribution for the state at time $t_{k,n}$, given the observations $y_{0:k}$, is of the form* (4.6)*. For $n \in \{2, \ldots, N\}$ the mean $m_{k,n}^-$ and covariance $P_{k,n}^-$ are calculated recursively using*

$$\begin{aligned} m_{k,n}^- &= A_{k,n-1}\, m_{k,n-1}^-, \\ P_{k,n}^- &= A_{k,n-1}\, P_{k,n-1}^-\, A_{k,n-1}^T + Q_{k,n-1}. \end{aligned}$$

*For $n = 1$, $m_k$ and $P_k$ from Theorem 4.2 are used and*

$$\begin{aligned} m_{k,1}^- &= A_{k,0}\, m_k, \\ P_{k,1}^- &= A_{k,0}\, P_k\, A_{k,0}^T + Q_{k,0}. \end{aligned}$$

**Theorem 4.2** (Kalman filter update step)**.** *If the state-measurement model is given by* (4.4)–(4.5) *and the prior is Gaussian, then the exact filtering distribution for the state at time $t_k$, given the observations $y_{0:k}$, is of the form* (4.7)*. The mean $m_k$ and covariance $P_k$ are calculated with $m_{k-1,N}^-$ and $P_{k-1,N}^-$ from Theorem 4.1 using the formulas*

$$\begin{aligned} v_k &= y_k - H_k\, m_{k-1,N}^-, \\ S_k &= H_k\, P_{k-1,N}^-\, H_k^T + R_k, \\ K_k &= P_{k-1,N}^-\, H_k^T\, S_k^{-1}, \\ m_k &= m_{k-1,N}^- + K_k\, v_k, \\ P_k &= P_{k-1,N}^- - K_k\, S_k\, K_k^T. \end{aligned}$$

### 4.2.2 Extended Kalman Filter

The Kalman Filter is heavily restricted by only applying to linear models. A simple but often effective solution for many nonlinear problems is the extended Kalman

filter, which uses Taylor expansions and works for models of the form (4.2)–(4.3). As with the Kalman filter, the obtained predictive and filtering distributions are Gaussian. However, in the case of the extended Kalman filter, these only approximate the true distributions without any guarantee of accuracy in general. In the prediction step, the extended Kalman filter asserts that approximately

$$p(X_{t_{k,n}} = x \mid Y_{0:k} = y_{0:k}) = \mathrm{N}(x \mid m^-_{k,n}, P^-_{k,n}), \quad n \in \{1, \ldots, N\},\ k \in \{0, \ldots, K\}. \tag{4.8}$$

Similarly, in the update step the filter asserts that

$$p(X_{t_k} = x \mid Y_{0:k} = y_{0:k}) = \mathrm{N}(x \mid m_k, P_k), \quad k \in \{0, \ldots, K\}. \tag{4.9}$$

To derive the prediction step (4.8), firstly assume

$$p(X_{t_{k,n-1}} = x \mid Y_{0:k} = y_{0:k}) = \mathrm{N}(x \mid m^-_{k,n-1}, P^-_{k,n-1}).$$

This assumption may be accurate in certain settings, but inaccurate in others. Then, the first order Taylor expansion of (4.2) around $X_{t_{k,n-1}} = m^-_{k,n-1}$ and $q_{k,n-1} = 0$ is used as an approximate state model. Let $F^x_{k,n}$ and $F^q_{k,n}$ be the Jacobians of $f_{k,n}$ with respect to the first and second argument respectively. An approximate model is

$$\begin{aligned} X_{t_{k,n}} \approx & f_{k,n-1}(m^-_{k,n-1}, 0) + F^x_{k,n-1}(m^-_{k,n-1}, 0)\,(X_{t_{k,n-1}} - m^-_{k,n-1}) \\ & + F^q_{k,n-1}(m^-_{k,n-1}, 0)\,q_{k,n-1}, \end{aligned}$$

The formulas for the mean $m^-_{k,n}$ and covariance $P^-_{k,n}$ now follow directly from the properties of Gaussian random variables. They are

$$\begin{aligned} m^-_{k,n} &= f_{k,n-1}(m^-_{k,n-1}, 0), \\ P^-_{k,n} &= F^x_{k,n-1}\,P^-_{k,n-1}\,F^{x^T}_{k,n-1} + F^q_{k,n-1}\,Q_{k,n-1}\,F^{q^T}_{k,n-1}. \end{aligned}$$

Here, the arguments of the Jacobians are not written explicitly. Note that these formulas apply for $n \in \{2, \ldots, N\}$. For $n = 1$, simply replace $m^-_{k,n-1}$ with $m_k$ and $P^-_{k,n}$ with $P_k$.

The update step (4.9) is obtained by similarly Taylor expanding (4.3) around $X_{t_k} = m^-_{k-1,N}$ and $r_k = 0$ to obtain

$$Y_k \approx h_k(m^-_{k-1,N}, 0) + H^x_k(m^-_{k-1,N}, 0)\,(X_{t_k} - m^-_{k-1,N}) + H^r_k(m^-_{k-1,N}, 0)\,r_k,$$

where $H^x_k$ is the Jacobian of $h_k$ with respect to the first argument and $H^r_k$ is the Jacobian with respect to the second argument. In the following, we refrain from writing out the arguments of the Jacobians. To start the derivation, assume that approximately

$$p(X_{t_k} = x \mid Y_{0:k-1} = y_{0:k-1}) = \mathrm{N}(x \mid m^-_{k-1,N}, P^-_{k-1,N}).$$

Using the properties of Gaussian random variables, the joint conditional distribution of $Y_k$ and $X_{t_k}$ can be approximated as

$$p(X_{t_k} = x, Y_k = y \mid Y_{0:k-1} = y_{0:k-1}) = \mathrm{N}((x, y)^T \mid \bar{m}_k, \bar{P}_k),$$

with mean and covariance

$$\bar{m}_k = \begin{pmatrix} m^-_{k-1,N} \\ h_k(m^-_{k-1,N}, 0) \end{pmatrix}, \quad \bar{P}_k = \begin{pmatrix} P^-_{k-1,N} & P^-_{k-1,N} H_k^{x^T} \\ H_k^x P^-_{k-1,N} & H_k^x P^-_{k-1,N} H_k^{x^T} + H_k^r R_k H_k^{r^T} \end{pmatrix}.$$

The rest of the derivation is almost identical to the proof of the Kalman filter update step, see Algorithm 5.5 in [35] for details. In summary, the obtained formulas are

$$\begin{aligned} v_k &= y_k - h_k(m^-_{k-1,N}, 0), \\ S_k &= H_k^x P^-_{k-1,N} H_k^{x^T} + H_k^r R_k H_k^{r^T}, \\ K_k &= P^-_{k-1,N} H_k^{x^T} S_k^{-1}, \\ m_k &= m^-_{k-1,N} + K_k v_k, \\ P_k &= P^-_{k-1,N} - K_k S_k K_k^T. \end{aligned}$$

### 4.2.3 Unscented Kalman Filter

The unscented Kalman filter was originally proposed in the 1990s and is, as such, newer than the extended Kalman filter. Compared to the extended Kalman filter, which only accurately estimates the mean and covariance to the first order, the unscented Kalman filter accurately estimates the mean and covariance to the third order. At the same time, it is not significantly more demanding computationally.

For an example comparison between the unscented and extended Kalman filters within target tracking, see [23] by the original inventors. The unscented filter is found to be more accurate than the extended Kalman filter. More comparisons are presented in [45]. As a first experiment, the authors compare performance on a problem from mathematical biology. In a second experiment, joint parameter-state estimation is attempted. In both these examples, the unscented filter outperforms the extended filter.

The principle behind the unscented Kalman filter is to use the so called unscented transform for moment estimation, while assuming Gaussian distributions. To estimate the moments after a nonlinear function is applied to a random variable, a relatively small number of deterministic points, called sigma points, are chosen. The nonlinear function is then applied to each sigma point, after which the propagated sigma points are used to calculate mean and covariance.

More specifically, in the prediction step, we once again start by assuming

$$p(X_{t_{k,n-1}} = x \mid Y_{0:k} = y_{0:k}) = \mathrm{N}(x \mid m^-_{k,n-1}, P^-_{k,n-1}).$$

Additionally, instead of approximating the dynamics via linearisation as with the extended Kalman filter, the assumption

$$p(X_{t_{k,n}} = x \mid Y_{0:k} = y_{0:k}) = \mathrm{N}(x \mid m^-_{k,n}, P^-_{k,n}),$$

is made directly. Now, what remains is to estimate the mean and covariance in the next step using the unscented transform. Let $\alpha$, $\beta$ and $\kappa$ be algorithm parameters.

Standard values for these are $\alpha = 10^{-3}$, $\beta = 2$ and $\kappa = 0$. Further, let $\lambda = \alpha^2(2d + \kappa) - 2d$ and form the augmented mean and covariance for $(X_{t_{k,n-1}}, q_{k,n-1})^T$ as

$$\widetilde{m}^-_{k,n-1} = \begin{pmatrix} m^-_{k,n-1} \\ 0 \end{pmatrix}, \quad \widetilde{P}^-_{k,n-1} = \begin{pmatrix} P^-_{k,n-1} & 0 \\ 0 & Q_{k,n-1} \end{pmatrix}.$$

The $4d + 1$ sigma points are then given by

$$\begin{aligned}
\chi^{(0)}_{k,n-1} &= \widetilde{m}^-_{k,n-1}, \\
\chi^{(i)}_{k,n-1} &= \widetilde{m}^-_{k,n-1} + \sqrt{2d + \lambda} \left[ \sqrt{\widetilde{P}^-_{k,n-1}} \right]_i, \quad i = 1, \ldots, 2d, \\
\chi^{(2d+i)}_{k,n-1} &= \widetilde{m}^-_{k,n-1} - \sqrt{2d + \lambda} \left[ \sqrt{\widetilde{P}^-_{k,n-1}} \right]_i, \quad i = 1, \ldots, 2d.
\end{aligned}$$

Here, the notation $[.]_i$ is used for the i-th column of a matrix. Using these, the propagated sigma points $\hat{\chi}^{(i)}_{k,n}$ for $i = 0, \ldots, 4d$ are formed through

$$\hat{\chi}^{(i)}_{k,n} = f_{k,n-1}(\chi^{(i),x}_{k,n-1}, \chi^{(i),q}_{k,n-1}),$$

where $\chi^{(i),x}_{k,n-1}$ is the first $d$ components in $\chi^{(i)}_{k,n-1}$ and $\chi^{(i),q}_{k,n-1}$ is the last $d$ components. The mean and covariance in the next step are now given by

$$\begin{aligned}
m^-_{k,n} &= \sum_{i=0}^{4d} W^{(i)}_m \, \hat{\chi}^{(i)}_{k,n}, \\
P^-_{k,n} &= \sum_{i=0}^{4d} W^{(i)}_c \, (\hat{\chi}^{(i)}_{k,n} - m^-_{k,n}) \, (\hat{\chi}^{(i)}_{k,n} - m^-_{k,n})^T,
\end{aligned}$$

where the weights $W^{(i)}_m$ and $W^{(i)}_c$ are

$$\begin{aligned}
W^{(0)}_m &= \frac{\lambda}{\lambda + 2d}, \\
W^{(0)}_c &= \frac{\lambda}{\lambda + 2d} + (1 - \alpha^2 + \beta), \\
W^{(i)}_m &= \frac{\lambda}{2(\lambda + 2d)}, \quad i = 1, \ldots, 4d, \\
W^{(i)}_c &= \frac{\lambda}{2(\lambda + 2d)}, \quad i = 1, \ldots, 4d.
\end{aligned}$$

This description of the prediction step applies for $n \in \{2, \ldots, N\}$. For $n = 1$ simply replace $m^-_{k,n-1}$ with $m_k$ and $P^-_{k,n-1}$ with $P_k$ in all relevant formulas.

The update step is performed similarly to the prediction step. Assume that, approximately

$$\begin{aligned}
p(X_{t_k} = x \mid Y_{0:k-1} = y_{0:k-1}) &= \mathrm{N}(x \mid m^-_{k-1,N}, P^-_{k-1,N}), \\
p(X_{t_k} = x \mid Y_{0:k} = y_{0:k}) &= \mathrm{N}(x \mid m_k, P_k).
\end{aligned}$$

Let $\lambda' = \alpha^2(d + d' + \kappa) - d - d'$ and form the augmented mean and covariance for $(X_{t_k}, r_k)^T$ as

$$\widetilde{m}^-_{k-1,N} = \begin{pmatrix} m^-_{k-1,N} \\ 0 \end{pmatrix}, \quad \widetilde{P}^-_{k-1,N} = \begin{pmatrix} P^-_{k-1,N} & 0 \\ 0 & R_k \end{pmatrix}.$$

The $2d + 2d' + 1$ sigma points before propagation are then

$$\begin{aligned}
\chi_k^{(0)} &= \widetilde{m}^-_{k-1,N}, \\
\chi_k^{(i)} &= \widetilde{m}^-_{k-1,N} + \sqrt{d + d' + \lambda'} \left[\sqrt{\widetilde{P}^-_{k-1,N}}\right]_i, \quad i = 1, \ldots, d + d', \\
\chi_k^{(d+d'+i)} &= \widetilde{m}^-_{k-1,N} - \sqrt{d + d' + \lambda'} \left[\sqrt{\widetilde{P}^-_{k-1,N}}\right]_i, \quad i = 1, \ldots, d + d',
\end{aligned}$$

The measurement function is now applied to each sigma point. Consequently, the propagated sigma points $\mathcal{Y}_k^{(i)}$, $i = 0, \ldots, 2d + 2d'$ are

$$\mathcal{Y}_k^{(i)} = h_k(\chi_k^{(i),x}, \chi_k^{(i),q}),$$

where $\chi_k^{(i),x}$ is the first $d$ components in $\chi_k^{(i)}$ and $\chi_k^{(i),q}$ is the last $d'$ components. The mean and covariance of the filtering density are then calculated using the formulas

$$\begin{aligned}
K_k &= C_k \, S_k^{-1}, \\
m_k &= m^-_{k-1,N} + K_k \, (y_k - \mu_k), \\
P_k &= P^-_{k-1,N} - K_k \, S_k \, K_k^T,
\end{aligned} \tag{4.10}$$

with $\mu_k$, $S_k$ and $C_k$ given by

$$\begin{aligned}
\mu_k &= \sum_{i=0}^{2d+2d'} \widetilde{W}_m^{(i)} \, \mathcal{Y}_k^{(i)}, \\
S_k &= \sum_{i=0}^{2d+2d'} \widetilde{W}_c^{(i)} \, (\mathcal{Y}_k^{(i)} - \mu_k) \, (\mathcal{Y}_k^{(i)} - \mu_k)^T, \\
C_k &= \sum_{i=0}^{2d+2d'} \widetilde{W}_c^{(i)} \, (\chi_k^{(i),x} - m^-_{k-1,N}) \, (\mathcal{Y}_k^{(i)} - \mu_k)^T.
\end{aligned}$$

Here, the weights $\widetilde{W}_m^{(i)}$ and $\widetilde{W}_c^{(i)}$ are

$$\begin{aligned}
\widetilde{W}_m^{(0)} &= \frac{\lambda'}{\lambda' + d + d'}, \\
\widetilde{W}_c^{(0)} &= \frac{\lambda'}{\lambda' + d + d'} + (1 - \alpha^2 + \beta), \\
\widetilde{W}_m^{(i)} &= \frac{\lambda'}{2(\lambda' + d + d')}, \quad i = 1, \ldots, 2d + 2d', \\
\widetilde{W}_c^{(i)} &= \frac{\lambda'}{2(\lambda' + d + d')}, \quad i = 1, \ldots, 2d + 2d'.
\end{aligned}$$

To derive (4.10), note that approximately

$$p(X_{t_k} = x, Y_k = y \mid Y_{0:k-1} = y_{0:k-1}) = \mathrm{N}((x, y)^T \mid \bar{m}_k, \bar{P}_k),$$

with mean and covariance

$$\bar{m}_k = \begin{pmatrix} m^-_{k-1,N} \\ \mu_k \end{pmatrix}, \quad \bar{P}_k = \begin{pmatrix} P^-_{k-1,N} & C_k \\ C_k^T & S_k \end{pmatrix}.$$

The rest of the derivation is almost identical to the proof of the Kalman filter update step, see Algorithm 5.5 in [35] for details. For a detailed explanation of the unscented transform the reader is referred to [22]. See Appendix A.2 and A.3 in this reference for proof that the unscented transform approximates the mean and covariance exactly to the third order.

### 4.2.4 Gaussian Filters

Both the extended and unscented Kalman filters are specific types of Gaussian filters. These are characterised by the assumption

$$p(X_{t_k} = x \mid Y_{0:k} = y_{0:k}) = \mathrm{N}(x \mid m_k, P_k),$$

and simply deviate in their respective methods for calculating the mean $m_k$ and covariance $P_k$. Other filters in this class include the Gauss–Hermite Kalman filter and the cubature Kalman filter. Both of these employ methods to directly approximate the integrals involved in the computation of the mean and covariance, see [35] for details. Another filter that could be said to belong to this class is the Ensemble Kalman Filter (EnKF), which relies on Monte Carlo samples to estimate moments. This last method is particularly useful for problems in very high dimensions as it avoids the need to propagate and store covariance matrices.

In certain settings the Gaussian assumption is accurate, in which case Gaussian filters work well. In other settings, such as when the distributions are multimodal or non-symmetrical, the Gaussian assumption is less suitable. For such settings one has to employ other methods, such as the particle filters described in the next section.

## 4.3 Particle Filters

Since the seminal paper [19] from 1993, particle filters have become tremendously popular for filtering. The most simple form, called the bootstrap particle filter, is both easy to implement and understand. Yet, there exists a vast number of extensions and adaptations tailored to different tasks, see for instance [7], [17] and [32].

In this thesis, only bootstrap particle filters are used, but the theory in this section treats more general variants. The model is assumed to be of the form (4.2)–(4.3). Note that typically no intermediate prediction steps are assumed for particle filters,

which makes the presentation below quite different compared to many in literature.

Particle filters are Monte Carlo methods, meaning that they utilise random samples, typically called particles, propagated through the state dynamics for estimation. Assume there are $M$ such samples and denote them in time $t_{k,n}$ as $(x^m_{k,n})_{m=1}^M$. Moreover, each sample is paired with a weight $w^m_{k,n}$. Given the sample-weight pairs, an estimation of the conditional expectation of some function $g$ is calculated through

$$\mathbb{E}[g(X_{t_{k,n}}) \mid Y_{0:k} = y_{0:k}] \approx \sum_{m=1}^{M} w^m_{k,n}\, g(x^m_{k,n}).$$

Note that for filtering, $n = 0$ is of main interest. The sample-weight pairs in all time steps are obtained through recursive iteration of prediction and update steps. Additionally, a resampling step is added. This is to prevent particle degeneracy, which happens when most particles have weights close to zero.

In the prediction for $n \in \{1, \dots N-1\}$, the particles are propagated by sampling from an importance distribution

$$x^m_{k,n+1} \sim \pi(x^m_{k,n+1} \mid x^m_{k,n}, y_{0:k}).$$

One intuitive choice is to use the distribution for the state update, but sometimes other distributions are used, see below, and this is compensated by in the weight calculations, next explained. We assume for simplicity that the importance distribution is Markovian, i.e., only dependent on $x^m_{k,n}$ and no previous state, although this is not necessary. Let the true conditional distribution be given by $p(x^m_{k,n+1} \mid x^m_{k,n})$. Then, the weights are updated according to

$$w^m_{k,n+1} = c\, \frac{p(x^m_{k,n+1} \mid x^m_{k,n})}{\pi(x^m_{k,n+1} \mid x^m_{k,n}, y_{0:k})}\, w^m_{k,n}. \tag{4.11}$$

The constant $c$ is chosen so that the weights sum to unity. For $n = 0$, the prediction step is done analogously but replacing $(x^m_{k,0}, w^m_{k,0})_{m=1}^M$ with $(\tilde{x}^m_{k,0}, \tilde{w}^m_{k,0})_{m=1}^M$, see the resampling step below.

As for the update step, it becomes quite simple with this formulation of the particle filter. For each particle we perform

$$\begin{aligned} x^m_{k,0} &= x^m_{k-1,N}, \\ w^m_{k,0} &= c\, p(y_k \mid x^m_{k-1,N})\, w^m_{k-1,N}, \end{aligned}$$

where, again, $c$ is a constant so that the weights sum to 1.

If the number of ineffective particles, i.e., particles with weight very close to zero, becomes high, resampling is performed after the update step. A typical condition is to resample if

$$M_{\text{eff}} \approx \frac{1}{\sum_{m=1}^{M} (w^m_{k,0})^2} < M/10.$$

While different resampling algorithms can be used (see [27] for an outlook) the overall idea is to replace the ineffective particles by sampling with replacement from the samples themselves. In the simplest algorithm, each $x_{k,0}^m$ is chosen with probability $w_{k,0}^m$. All weights are then set to $1/M$. Whether or not resampling has actually been performed, denote the sample-weight pairs after resampling as $(\tilde{x}_{k,0}^m, \tilde{w}_{k,0}^m)_{m=1}^M$. Note that here, we assume that the updated weights and states before resampling are used for estimation. This is because the resampling can introduce additional variance to estimates.

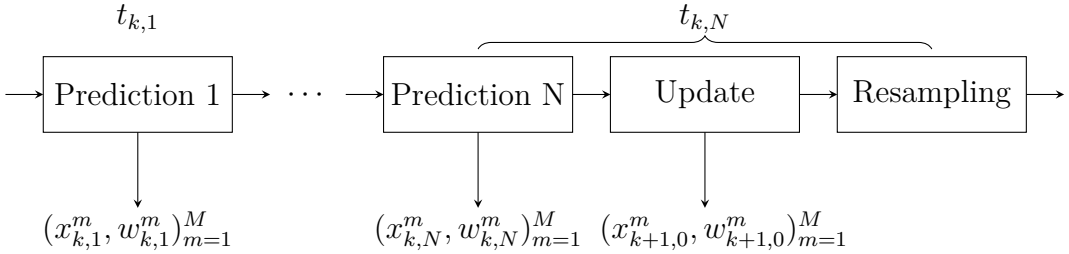


**Figure 4.1:** A schematic presentation of the steps in the particle filter algorithm. The sample-weight pairs $(\tilde{x}_{k+1,0}^m, \tilde{w}_{k+1,0}^m)_{m=1}^M$ from the resampling step are used in the first prediction step for $k+1$. The process is started by sampling from $p_0(x)$.

To start the algorithm, $M$ samples are taken from the prior $p_0(x)$ and given equal weights. The prediction, update and resampling steps are then iterated to obtain all sample-weight pairs $(x_{k,n}^m, w_{k,n}^m)$. An illustration of this process is shown in Figure 4.1. Remark that in this section the particle filter algorithm has only been presented and not derived. The reader is referred to Chapter 7 in [35] for details on the derivation when there are no intermediate steps.

Typically, the importance distribution is essential for the accuracy of a particle filter. A common choice is to let it be given by some Kalman filter, for instance the extended or unscented ones, or some distribution with heavier tails. From the algorithm presented above, the bootstrap filter can be obtained by resampling in every update step and by using the true distribution as sampling distribution. As such, the resulting algorithm is quite simple. Most notably, the weights are never updated in a prediction step, see (4.11), and the sampling in the prediction step is simply done through

$$x_{k,n+1}^m = f_{k,n}(x_{k,n}^m, q_{k,n}^m),$$

where $q_{k,n}^m$ is a sample from $\mathrm{N}(0, Q_{k,n})$.

While particle filters have many advantages, such as simplicity and ability to approximate highly nonlinear dynamics, two inherent difficulties associated with them are degeneracy and sample impoverishment. The latter occurs when all samples tend to have the same value. Consequently, estimates can be dominated by a small number of particles. While resampling prevents degeneracy to some extent, the impoverishment problem often remains. These problems become more apparent in

high dimensions, which is the reason behind why these filters scale poorly [32], [42]. Consequently, particle filters can be infeasible in high-dimensional problems when rapid real-time filtering is required. As an alternative, the method referred to as energy-based deep splitting is presented in the next chapter.

# 5

# Energy-Based Deep Splitting Derivation

In this chapter EBDS is derived in detail. The key results are (5.20)–(5.22) as well as Algorithm 1 and 2 in Section 5.2.4. Firstly however, the chapter starts with the derivation of a recursive optimisation scheme for approximate densities in Section 5.1. Section 5.2 then treats how to solve this problem using deep learning and some adjustments are made to the optimisation scheme to obtain better properties. The final part of this chapter discusses the convergence of the method. As a general guide, a reader who is only interested in the algorithm from an implementation perspective could start with the preliminaries in Section 5.1.1 and then move directly to 5.2.4.

Note that the derivation in this chapter follows that in [2] closely. Throughout, we assume enough regularity for probability densities to exist and that these are sufficiently smooth. In particular, we want to be able to apply the results from Section 2.3 regarding the Fokker–Planck equation. Some technical details in the proofs are also left out for the purpose of brevity.

## 5.1 Derivation of Recursive Optimisation Scheme

The goal of this section is to obtain an optimisation scheme for approximate solutions to the Fokker–Planck equation. The section ends with Theorem 5.2, which contains this scheme. Our approach is then to solve the infinite-dimensional optimisation problem using neural networks, which is presented in Section 5.2.

### 5.1.1 Preliminaries

We start by introducing the setting, notation and general idea of the method. Assume that we have a complete filtered probability space $(\Omega, \mathcal{F}, \mathbb{F} = (\mathcal{F}_t)_{t\in[0,T]}, \mathbb{P})$. The state is modelled as a continuous time stochastic process $X\colon [0,T] \times \Omega \to \mathbb{R}^d$, whereas the observations are modelled as a discrete time stochastic process $Y\colon \{0,1,\dots,K\} \times \Omega \to \mathbb{R}^{d'}$. For the state, let $\mu\colon \mathbb{R}^d \to \mathbb{R}^d$ be the drift function and let $\sigma\colon \mathbb{R}^d \to \mathbb{R}^{d\times d}$ be the diffusion function. The process noise is given by the $d$-dimensional $\mathbb{F}$-adapted Brownian motion $W\colon [0,T]\times\Omega \to \mathbb{R}^d$. For the observations, define $h\colon \mathbb{R}^d \to \mathbb{R}^{d'}$ as the measurement function and $V\colon \{0,1,\dots,K\} \times \Omega \to \mathbb{R}^{d'}$ as the measurement noise. Assume that $V_k \sim \mathrm{N}(0,\Sigma)$ is $\mathcal{F}_{t_k}$-measurable for all $k$.

The joint state-observation model is then

$$\begin{aligned} X_t &= X_0 + \int_0^t \mu(X_s)\,\mathrm{d}s + \int_0^t \sigma(X_s)\,\mathrm{d}W_s, \quad t \in [0,T], \\ Y_k &= h(X_{t_k}) + V_k, \quad k \in \{0,1,\dots,K\}. \end{aligned} \tag{5.1}$$

At time $t = 0$, assume that $X_0$ is $\mathcal{F}_0$-measurable but independent of $W$ with known density $p_0(x)$. Again, note that the observations are only available at fixed times. We assume enough regularity for a unique strong solution to exist, see Proposition 2.1.

For $k \in \{0,\dots K-1\}$ let $p_k(t,\, x \mid y_{0:k})\colon [t_k, t_{k+1}] \times \mathbb{R}^d \to \mathbb{R}$ be the probability density function satisfying

$$\mathbb{P}(X_t \in B \mid Y_{0:k} = y_{0:k}) = \int_{x\in B} p_k(t,\, x \mid y_{0:k})\,\mathrm{d}x, \tag{5.2}$$

given a sample path $y = Y(\omega)$, any Borel set $B$ and $t \in [t_k, t_{k+1}]$. The objects of interest are the filtering densities $p_k(t_k,\, x \mid y_{0:k})$ for $k \in \{0,\dots,K\}$.

In EBDS the approach is to solve for these recursively. According to Bayes' theorem we have

$$p(X_{t_{k+1}} \mid Y_{0:k+1}) = \frac{p(Y_{k+1} \mid X_{t_{k+1}}, Y_{0:k})\, p(X_{t_{k+1}} \mid Y_{0:k})}{p(Y_{k+1} \mid Y_{0:k})}.$$

Using the notation defined above, as well as the law of total probability for the marginal distribution $p(Y_{k+1} \mid Y_{0:k})$, this can be rewritten as

$$p_{k+1}(t_{k+1},\, x \mid y_{0:k+1}) = \frac{p(Y_{k+1} = y_{k+1} \mid X_{t_{k+1}} = x)\, p_k(t_{k+1},\, x \mid y_{0:k})}{\int_{\mathbb{R}^d} p(Y_{k+1} = y_{k+1} \mid X_{t_{k+1}} = x)\, p_k(t_{k+1},\, x \mid y_{0:k})\,\mathrm{d}x}. \tag{5.3}$$

The likelihood factor $p(Y_{k+1} = y_{k+1} \mid X_{t_{k+1}} = x)$ is given from the normality assumption of the measurement. The main difficulty now lies in calculating the predictive distribution $p_k(t_{k+1},\, x \mid y_{0:k})$. This is done using a deep learning approach for approximately solving the Fokker–Planck PDE. Remark that the initial condition for the Fokker–Planck equation is given by the filtering density in the previous step, hence the method is recursive. The derivation starts by introducing a splitting-up method for the operator in the PDE, which is presented next.

### 5.1.2 Splitting Approach

Define $a(X_t) = \sigma(X_t)\sigma(X_t)^T$. For $f \in C_0^\infty(\mathbb{R}^d;\mathbb{R})$, the operator associated with the stochastic differential equation (5.1) and its adjoint are

$$\begin{aligned} (Af)(x) &= \sum_{i=1}^d \mu_i(x)\frac{\partial f(x)}{\partial x_i} + \frac{1}{2}\sum_{i=1}^d\sum_{j=1}^d a_{ij}(x)\frac{\partial^2 f(x)}{\partial x_i \partial x_j}, \\ (A^* f)(x) &= -\sum_{i=1}^d \frac{\partial}{\partial x_i}\Big(\mu_i(x) f(x)\Big) + \frac{1}{2}\sum_{i=1}^d\sum_{j=1}^d \frac{\partial^2}{\partial x_i \partial x_j}\Big(a_{ij}(x) f(x)\Big). \end{aligned}$$

The time evolution of the conditional density in (5.2) for $t \in [t_k, t_{k+1}]$ is governed by the Fokker-Planck equation, see (2.10) in Section 2.3. The Fokker–Planck equation on the interval $[t_k, t_{k+1}]$ is

$$\frac{\partial p_k(t,\, x \mid y_{0:k})}{\partial t} = A^* p_k(t, x \mid y_{0:k}), \tag{5.4}$$

with initial condition given by (5.3) as

$$p_k(t_k,\, x \mid y_{0:k}) = \frac{p(Y_k = y_k \mid X_{t_k} = x)\, p_{k-1}(t_k,\, x \mid y_{0:k-1})}{\int_{\mathbb{R}^d} p(Y_k = y_k \mid X_{t_k} = x)\, p_{k-1}(t_k,\, x \mid y_{0:k-1})\, \mathrm{d}x}.$$

We now rewrite (5.4) in a form more suitable for approximations. Define a finer grid of time points $t_{k,0} < t_{k,1} < \cdots < t_{k,N}$ such that $t_k = t_{k,0}$ and $t_{k,N} = t_{k+1}$.

**Lemma 5.1** (Reformulation of the Fokker–Planck equation)**.** *For $t \in [t_{k,n}, t_{k,n+1}]$ and $n \in \{0, \ldots, N-1\}$, the density $p_k(t,\, x \mid y_{0:k})$ satisfies*

$$p_k(t,\, x \mid y_{0:k}) = p_k(t_{k,n},\, x \mid y_{0:k}) + \int_{t_{k,n}}^{t} (A + F)\, p_k(s,\, x \mid y_{0:k})\, \mathrm{d}s, \tag{5.5}$$

*where the operator $F$ is defined for $f \in C^2(\mathbb{R}^d; \mathbb{R})$ as*

$$\begin{aligned}(Ff)(x) = &-2\sum_{i=1}^{d} \mu_i(x) \frac{\partial f(x)}{\partial x_i} - \sum_{i=1}^{d} f(x) \frac{\partial \mu_i(x)}{\partial x_i} \\ &+ \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} f(x) \frac{\partial^2 a_{ij}(x)}{\partial x_i \partial x_j} + \sum_{i=1}^{d} \sum_{j=1}^{d} \frac{\partial f(x)}{\partial x_i} \frac{\partial a_{ij}(x)}{\partial x_j}.\end{aligned}$$

*Proof.* Integrating both sides in (5.4) yields

$$p_k(t,\, x \mid y_{0:k}) = p_k(t_k,\, x \mid y_{0:k}) + \int_{t_k}^{t} A^* p_k(s,\, x \mid y_{0:k})\, \mathrm{d}s.$$

Next, we note that by differentiation $A^* p_k(s,\, x \mid y_{0:k}) = (A + F)\, p_k(s,\, x \mid y_{0:k})$. The transition to (5.5) is now trivial. □

Solving (5.5) is equivalent to solving the Fokker–Planck equation. To continue the derivation of the filtering method, we now introduce a first approximation. The idea behind this approximation is to treat the two operator terms in (5.5) differently. The term involving the operator $A$ is kept for now, but vanishes later in the derivation. The term involving $F$, on the other hand, is dealt with by a forward Euler scheme. We introduce approximate densities $\pi_{k,n}(t,\, x \mid y_{0:k})$ for $n = 1, \ldots, N$. For $t \in (t_{k,n}, t_{k,n+1}]$ the density $\pi_{k,n+1}$ satisfies

$$\begin{aligned}\pi_{k,n+1}(t,\, x \mid y_{0:k}) = \pi_{k,n}(t_{k,n},\, x \mid y_{0:k}) &+ \int_{t_{k,n}}^{t} A \pi_{k,n+1}(s,\, x \mid y_{0:k})\, \mathrm{d}s \\ &+ F \pi_{k,n}(t_{k,n},\, x \mid y_{0:k})\, (t_{k,n+1} - t_{k,n}).\end{aligned} \tag{5.6}$$

Note that the difference between (5.5) and (5.6) is the term involving the operator $F$. For $n = 0$ we naturally define

$$\pi_{k,0}(t_{k,0},\, x \mid y_{0:k}) = \frac{p(Y_k = y_k \mid X_{t_k} = x)\, \pi_{k-1,N}(t_{k-1,N},\, x \mid y_{0:k-1})}{\int_{\mathbb{R}^d} p(Y_k = y_k \mid X_{t_k} = x)\, \pi_{k-1,N}(t_{k-1,N},\, x \mid y_{0:k-1})\, \mathrm{d}x}.$$

Informally, the idea is that $\pi_{k,n+1}(t,\, x \mid y_{0:k}) \approx p_k(t,\, x \mid y_{0:k})$ for $t \in (t_{k,n}, t_{k,n+1}]$. This method for solving the Fokker–Planck equation is referred to as a splitting approach since we essentially split the original equation in two parts, one involving the operator $A$ and the other $F$, and solve these parts separately. The separate treatment of the two operators can further be seen in the next sections. The next step in the derivation is to derive a Feynman–Kac formula to replace the PDE in (5.6) with an expected value problem.

### 5.1.3 Feynman–Kac Formula

In this section, a Feynman–Kac formula is derived for the solution to (5.6). To repeat, the purpose is to later in the derivation arrive at a recursive optimisation problem for approximate filtering densities. This optimisation problem is then solved using sampling and deep learning.

For the Feynman–Kac formula, we define an auxiliary process $\widetilde{X}\colon [0,T] \times \Omega \to \mathbb{R}^d$ satisfying the same dynamics as the state, namely

$$\widetilde{X}_t = \widetilde{X}_0 + \int_0^t \mu(\widetilde{X}_s)\, \mathrm{d}s + \int_0^t \sigma(\widetilde{X}_s)\, \mathrm{d}\widetilde{W}_s, \quad t \in [0,T].$$

Here $\widetilde{W}$ is another Brownian motion independent of $W$ and $V$. Let $\widetilde{\mathbb{F}} = \{\widetilde{\mathcal{F}}_t\}_{t\in[0,T]}$ be the filtration generated by $\widetilde{W}$.

**Theorem 5.1** (Feynman–Kac formula for approximate densities)**.** *For $n \in \{0, \dots, N-1\}$ the approximate density $\pi_{k,n+1}(t,\, x \mid y_{0:k})$ defined in* (5.6) *satisfies the recursive Feynman–Kac type formula*

$$\begin{aligned}\pi_{k,n+1}(t_{k,n+1}, \widetilde{X}_{t_{k+1}-t_{k,n+1}} \mid y_{0:k}) \\ =\mathbb{E}\Big[\pi_{k,n}(t_{k,n},\, \widetilde{X}_{t_{k+1}-t_{k,n}} \mid y_{0:k}) \\ + F\pi_{k,n}(t_{k,n},\, \widetilde{X}_{t_{k+1}-t_{k,n}} \mid y_{0:k})\,(t_{k,n+1} - t_{k,n}) \mid \widetilde{\mathcal{F}}_{t_{k+1}-t_{k,n+1}}\Big].\end{aligned} \tag{5.7}$$

*Proof.* Differentiating both sides in (5.6) with respect to time, we obtain

$$\frac{\partial \pi_{k,n+1}(t,\, x \mid y_{0:k})}{\partial t} = A\pi_{k,n+1}(t,\, x \mid y_{0:k}).$$

Now let $t = t_{k+1} - t'$. If $t \in (t_{k,n}, t_{k,n+1}]$, then $t' \in [t_{k+1} - t_{k,n+1}, t_{k+1} - t_{k,n})$. Inserting this reparameterisation of time, we obtain

$$\frac{\partial \pi_{k,n+1}(t_{k+1} - t',\, x \mid y_{0:k})}{\partial t'} + A\pi_{k,n+1}(t_{k+1} - t',\, x \mid y_{0:k}) = 0. \tag{5.8}$$

This is the Kolmogorov backward equation. The terminal condition as $t' \nearrow t_{k+1}-t_{k,n}$ is given by (5.6) as

$$\begin{aligned}\lim_{t' \nearrow t_{k+1}-t_{k,n}} \pi_{k,n+1}(t_{k+1}-t',\, x \mid y_{0:k}) =& \pi_{k,n}(t_{k,n},\, x \mid y_{0:k}) \\ &+ F\pi_{k,n}(t_{k,n},\, x \mid y_{0:k})\,(t_{k,n+1}-t_{k,n}).\end{aligned} \tag{5.9}$$

Applying Itô's formula (see Proposition 2.2) to $\pi_{k,n+1}(t_{k+1}-t',\, \widetilde{X}_{t'} \mid y_{0:k})$ for $t' \in [t_{k+1}-t_{k,n+1}, t_{k+1}-t_{k,n})$, we obtain

$$\begin{aligned}&\pi_{k,n+1}(t_{k+1}-t',\, \widetilde{X}_{t'} \mid y_{0:k}) = \\ &\quad \pi_{k,n+1}(t_{k,n+1},\, \widetilde{X}_{t_{k+1}-t_{k,n+1}} \mid y_{0:k}) \\ &\quad + \int_{t_{k+1}-t_{k,n+1}}^{t'} \langle \nabla_x \pi_{k,n+1}(t_{k+1}-s,\, \widetilde{X}_s \mid y_{0:k}), \sigma(\widetilde{X}_s)\, \mathrm{d}\widetilde{W}_s \rangle \\ &\quad + \int_{t_{k+1}-t_{k,n+1}}^{t'} \left( \frac{\partial \pi_{k,n+1}(t_{k+1}-s,\, \widetilde{X}_s \mid y_{0:k})}{\partial s} + A\pi_{k,n+1}(t_{k+1}-s,\, \widetilde{X}_s \mid y_{0:k}) \right) \mathrm{d}s.\end{aligned} \tag{5.10}$$

Now, the third term vanishes, according to (5.8). Taking the conditional expectation of both sides with respect to $\widetilde{\mathcal{F}}_{t_{k+1}-t_{k,n+1}}$, we obtain

$$\mathbb{E}\left[\pi_{k,n+1}(t_{k+1}-t',\, \widetilde{X}_{t'} \mid y_{0:k}) \mid \widetilde{\mathcal{F}}_{t_{k+1}-t_{k,n+1}}\right] = \pi_{k,n+1}(t_{k,n+1},\, \widetilde{X}_{t_{k+1}-t_{k,n+1}} \mid y_{0:k}). \tag{5.11}$$

The expectation on the first term in (5.10) vanishes, since $\widetilde{X}_t$ is $\widetilde{\mathcal{F}}_t$-measurable. As the Itõ integral process

$$\left\{ \int_{t_{k+1}-t_{k,n+1}}^{t'} \langle \nabla_x \pi_{k,n+1}(t_{k+1}-s,\, \widetilde{X}_s \mid y_{0:k}), \sigma(\widetilde{X}_s)\, \mathrm{d}\widetilde{W}_s \rangle \right\}_{t' \in [t_{k+1}-t_{k,n+1}, t_{k+1}-t_{k,n})}$$

is a zero mean martingale with respect to $\widetilde{\mathbb{F}}$, the second term in (5.10) becomes zero under the conditional expectation.

Returning to (5.11), we take the limit of both sides as $t' \nearrow t_{k+1}-t_{k,n}$. With some theoretical arguments, we can express the left hand side as

$$\begin{aligned}&\lim_{t' \nearrow t_{k+1}-t_{k,n}} \mathbb{E}\left[\pi_{k,n+1}(t_{k+1}-t',\, \widetilde{X}_{t'} \mid y_{0:k}) \mid \widetilde{\mathcal{F}}_{t_{k+1}-t_{k,n+1}}\right] \\ &= \mathbb{E}\left[ \lim_{t' \nearrow t_{k+1}-t_{k,n}} \pi_{k,n+1}(t_{k+1}-t',\, \widetilde{X}_{t'} \mid y_{0:k}) \mid \widetilde{\mathcal{F}}_{t_{k+1}-t_{k,n+1}} \right] \\ &= \mathbb{E}\Big[\pi_{k,n}(t_{k,n},\, \widetilde{X}_{t_{k+1}-t_{k,n}} \mid y_{0:k}) \\ &\qquad + F\pi_{k,n}(t_{k,n},\, \widetilde{X}_{t_{k+1}-t_{k,n}} \mid y_{0:k})\,(t_{k,n+1}-t_{k,n}) \mid \widetilde{\mathcal{F}}_{t_{k+1}-t_{k,n+1}}\Big].\end{aligned}$$

In the last step we have used (5.9). □

Theorem 5.1 gives a formula for the approximate probability density function of the state in the next time step $t_{k,n+1}$ given the approximate density in the current time step $t_{k,n}$. The idea is to recursively solve this equation approximately in each time step. The resulting optimisation problem is derived and presented in the next section.

### 5.1.4 Optimisation Problem

In order to feasibly solve (5.7), we first introduce the new process $Z$. This is the Euler–Maruyama approximation of $\widetilde{X}$. Here we first begin by restricting the time settings of the problem for simplicity. This restriction is not needed after later modifications, see Section 5.2.2.

**Setting 5.1.** *We consider $t_{k+1} - t_k = \Delta t$ for all $k \in \{0, \dots, K-1\}$ and $t_0 = 0$. Furthermore, the partition on each interval $[t_k, t_{k+1}]$ is identical and equidistant, so that $(t_{k,n+1} - t_{k,n}) = \Delta t / N$ for all $k$ and $n$.*

Using this setting, if we define $t'_n = t_{0,n}$, we then have $t_{k,n'} - t_{k,n} = t'_{n'-n}$ for $n' \geq n$ and $n, n' \in \{0, \dots, N\}$. Now, define $Z$ as

$$Z_{n+1} = Z_n + \mu(Z_n)\,(t'_{n+1} - t'_n) + \sigma(Z_n)\,(\widetilde{W}_{t'_{n+1}} - \widetilde{W}_{t'_n}).$$

Thus $Z_n$ is the Euler–Maruyama approximation of $\widetilde{X}_{t'_n}$. Furthermore, as $t_{k+1} - t_{k,n+1} = t'_{N-(n+1)}$ and $t_{k+1} - t_{k,n} = t'_{N-n}$ we have $\widetilde{X}_{t_{k+1}-t_{k,n+1}} \approx Z_{N-(n+1)}$ and $\widetilde{X}_{t_{k+1}-t_{k,n}} \approx Z_{N-n}$. In the next approximation, the auxiliary process $\widetilde{X}$ in (5.7) is replaced with its Euler–Maruyama approximation $Z$ accordingly. We introduce the approximate densities $\widetilde{\pi}_{k,n}(x \mid y_{0:k})$ for $n = 0, \dots, N$ satisfying

$$\begin{aligned}
\widetilde{\pi}_{k,n+1}(Z_{N-(n+1)} \mid y_{0:k}) =& \mathbb{E}\Big[\widetilde{\pi}_{k,n}(Z_{N-n} \mid y_{0:k}) \\
&+ F\widetilde{\pi}_{k,n}(Z_{N-n} \mid y_{0:k})\,(t_{k,n+1} - t_{k,n}) \mid \widetilde{\mathcal{F}}_{t_{k+1}-t_{k,n+1}}\Big], \\
\widetilde{\pi}_{k,0}(x \mid y_{0:k}) =& \frac{p(Y_k = y_k \mid X_{t_k} = x)\,\widetilde{\pi}_{k-1,N}(x \mid y_{0:k-1})}{\int_{\mathbb{R}^d} p(Y_k = y_k \mid X_{t_k} = x)\,\widetilde{\pi}_{k-1,N}(x \mid y_{0:k-1})\,\mathrm{d}x}.
\end{aligned} \tag{5.12}$$

Note that, while $\pi_{k,n}(t, x \mid y_{0:k})$ is defined in the entire time interval $t \in (t_{k,n-1}, t_{k,n}]$, the new approximation $\widetilde{\pi}_{k,n}(x \mid y_{0:k}) \approx \pi_{k,n}(t_{k,n}, x \mid y_{0:k})$ is defined exclusively in $t_{k,n}$. The next result allows us to solve for $\widetilde{\pi}$ in each time step by solving an optimisation problem.

**Theorem 5.2** (Recursive optimisation problem for approximate densities)**.** *For $n \in \{0, \dots, N-1\}$, the density $\widetilde{\pi}_{k,n+1}(x \mid y_{0:k})$ defined in* (5.12) *satisfies the recursive optimisation problem*

$$\begin{aligned}
\Big(\widetilde{\pi}_{k,n+1}(x \mid y_{0:k})\Big)_{x \in \mathbb{R}^d} = \operatorname*{argmin}_{u \in C(\mathbb{R}^d;\mathbb{R})} \mathbb{E}\Big[\Big|u(Z_{N-(n+1)}) - \Big(&\widetilde{\pi}_{k,n}(Z_{N-n} \mid y_{0:k}) \\
&+ F\widetilde{\pi}_{k,n}(Z_{N-n} \mid y_{0:k})\,(t_{k,n+1} - t_{k,n})\Big)\Big|^2\Big].
\end{aligned} \tag{5.13}$$

*Proof.* Since $\mathfrak{S}(Z_{N-(n+1)}) \subset \widetilde{\mathcal{F}}_{t_{k+1}-t_{k,n+1}}$, applying the conditional expectation with respect to $\mathfrak{S}(Z_{N-(n+1)})$ on both sides of (5.12) yields

$$\begin{aligned}
\widetilde{\pi}_{k,n+1}(Z_{N-(n+1)} \mid y_{0:k}) =& \mathbb{E}\Big[\widetilde{\pi}_{k,n}(Z_{N-n} \mid y_{0:k}) \\
&+ F\widetilde{\pi}_{k,n}(Z_{N-n} \mid y_{0:k})\,(t_{k,n+1} - t_{k,n}) \mid \mathfrak{S}(Z_{N-(n+1)})\Big],
\end{aligned}$$

by utilisation of the tower property of conditional expectation. In general, the conditional expectation with respect to a $\sigma$-field $\mathcal{G}$ is the $\mathcal{G}$-measurable random variable satisfying $L^2$-minimality. Hence we have

$$\begin{aligned}\left(\widetilde{\pi}_{k,n+1}(x \mid y_{0:k})\right)_{x\in\mathbb{R}^d} = \underset{u\in U}{\operatorname{argmin}}\ \mathbb{E}\Big[\Big|u(Z_{N-(n+1)}) - \big(\widetilde{\pi}_{k,n}(Z_{N-n} \mid y_{0:k}) \\ + F\widetilde{\pi}_{k,n}(Z_{N-n} \mid y_{0:k})\,(t_{k,n+1} - t_{k,n})\big)\Big|^2\Big].\end{aligned}$$

The optimisation is performed over the set $U = \{u : \mathbb{R}^d \to \mathbb{R} \mid u(Z_{N-(n+1)}) \in L^2(\Omega, \mathfrak{S}(Z_{N-(n+1)}); \mathbb{R})\}$. This can be changed to a minimisation over all functions in $C(\mathbb{R}^d; \mathbb{R})$. □

The optimisation problem (5.13) allows us to recursively solve for the density in the next time step given the current one. However, note that the optimisation problem is performed over the infinite dimensional space $C(\mathbb{R}^d; \mathbb{R})$. Moreover, the expectation is difficult, if not impossible, to calculate analytically in most examples. As such, the approach is to solve (5.13) using Monte Carlo estimation as well as deep learning. The next section treats how this is done in practice.

## 5.2 Deep Learning for the Optimisation Problem

The previous section culminates with (5.13), which is the foundation for the EBDS filtering method. In this section, the aim is to solve the optimisation problem using deep learning. However, if this was done without further steps, the obtained filter would have certain properties that are either unpractical or problematic from a computational point of view. These properties are detailed and corrected in the following three sections. After these corrections the final method is presented in Section 5.2.4. Note again that we follow the approach taken in [2] closely.

### 5.2.1 Observations as Input

The optimisation problem (5.13) gives labels that can be used to sequentially train neural networks with the MSE loss function. Crucially however, the training is performed online for a given sequence of observations $y_{0:k}$. This is computationally demanding and generally infeasible in many applications. The first modification to the initial scheme is to perform the training offline.

Let $p_k(t, x, y)\colon [t_k, t_{k+1}] \times \mathbb{R}^d \times \mathbb{R}^{d'\times(k+1)} \to \mathbb{R}$ be the conditional probability density function $p_k(t, x, y_{0:k}) = p_k(t,\, x \mid y_{0:k})$ with the previously defined notation, see (5.2). Similarly, we define $\widetilde{\pi}_{k,n}(x, y)\colon \mathbb{R}^d \times \mathbb{R}^{d'\times(k+1)} \to \mathbb{R}$ such that $\widetilde{\pi}_{k,n}(x, y_{0:k}) = \widetilde{\pi}_{k,n}(x \mid y_{0:k})$, see (5.12). This slight change in notation is to emphasize that observations are now taken as input.

Returning to the original optimisation problem in (5.13), with the new notation

$$\begin{aligned}\left(\widetilde{\pi}_{k,n+1}(x, y_{0:k})\right)_{x\in\mathbb{R}^d} = \underset{u\in C(\mathbb{R}^d;\mathbb{R})}{\operatorname{argmin}}\ \mathbb{E}\Big[\Big|u(Z_{N-(n+1)}) - \big(\widetilde{\pi}_{k,n}(Z_{N-n}, y_{0:k}) \\ + F\widetilde{\pi}_{k,n}(Z_{N-n}, y_{0:k})\,(t_{k,n+1} - t_{k,n})\big)\Big|^2\Big].\end{aligned} \tag{5.14}$$

In order to avoid solving new optimisation problems individually for each $y$, the natural choice is to let $\widetilde{\pi}_{k,n}(x, y)$ satisfy

$$\begin{aligned}&\left(\widetilde{\pi}_{k,n+1}(x, y)\right)_{(x,y)\in\mathbb{R}^d\times\mathbb{R}^{d'\times(k+1)}} \\ &\quad= \underset{u\in C(\mathbb{R}^d\times\mathbb{R}^{d'\times(k+1)};\mathbb{R})}{\operatorname{argmin}}\ \mathbb{E}\Big[\mathbb{E}\Big[\Big|u(Z_{N-(n+1)}, Y_{0:k}) - \big(\widetilde{\pi}_{k,n}(Z_{N-n}, Y_{0:k}) \\ &\qquad + F\widetilde{\pi}_{k,n}(Z_{N-n}, Y_{0:k})\,(t_{k,n+1} - t_{k,n})\big)\Big|^2 \mid \mathfrak{S}(Y_{0:k})\Big]\Big].\end{aligned}$$

According to the tower property, the expectation can be rewritten as a single unconditional expectation

$$\begin{aligned}&\left(\widetilde{\pi}_{k,n+1}(x, y)\right)_{(x,y)\in\mathbb{R}^d\times\mathbb{R}^{d'\times(k+1)}} \\ &\quad= \underset{u\in C(\mathbb{R}^d\times\mathbb{R}^{d'\times(k+1)};\mathbb{R})}{\operatorname{argmin}}\ \mathbb{E}\Big[\Big|u(Z_{N-(n+1)}, Y_{0:k}) - \big(\widetilde{\pi}_{k,n}(Z_{N-n}, Y_{0:k}) \\ &\qquad + F\widetilde{\pi}_{k,n}(Z_{N-n}, Y_{0:k})\,(t_{k,n+1} - t_{k,n})\big)\Big|^2\Big].\end{aligned} \tag{5.15}$$

This expectation is later in the derivation approximated with a Monte Carlo estimate, for which both $Z$ and $Y$ are sampled. As is shown next, this natural extension is the appropriate one. The proof for the following theorem was original presented in [2].

**Theorem 5.3** (Equivalence of optimisation problems)**.** *If a conditional density $u^*$ solves the optimisation problem* (5.15)*, then it solves the optimisation problem* (5.14) *for almost every sequence $y_{0:k}$.*

*Proof.* First note that the optimisation problem (5.15) has a unique solution with objective value 0. Moreover, the auxiliary process $Z$ and observations $Y$ are independent, by assumption. Both of these also permit densities. Let

$$g(u, z, y) = \left|u(z, y) - \left(\widetilde{\pi}_{k,n}(z, y) + F\widetilde{\pi}_{k,n}(z, y)\,(t_{k,n+1} - t_{k,n})\right)\right|^2.$$

For the optimal $u$, we have

$$\int_y \int_z g(u^*, z, y)\, p_Z(z)\, p_Y(y)\, \mathrm{d}z\, \mathrm{d}y = 0.$$

As all factors in the integrand are $\geq 0$, we must have for almost every $y$

$$\int_z g(u^*, z, y)\, p_Z(z)\, \mathrm{d}z = \mathbb{E}[g(u^*, Z, y)] = 0.$$

Note that $\mathbb{E}[g(u^*, Z, y)]$ is the objective in (5.14). Hence $u^*$ clearly solves (5.14) for almost every $y$. □

### 5.2.2 Reducing Sample - Density Temporal Difference

Next, we note that in (5.15), there will often be a significant temporal difference between the Euler–Maruyama simulated auxiliary process $Z_{N-n}$ in $t_{0,N-n}$ and the density $\widetilde{\pi}_{k,n}(x,y)$ in time $t_{k,n}$. This will pose a problem if we try to approximate the expectation with a Monte Carlo estimate. The reason being that if the dynamics of the state tends to shift the distribution, most of the samples of $Z$ will fall outside of the main probability mass. See Section 3.4 in [2] for an illustration of this phenomenon. As an additional consequence, the neural network training focuses on areas in $\mathbb{R}^d$ not of chief interest.

The solution is to use samples as close as possible in time to $t_{k,n}$. For this purpose, we revisit Setting 5.1. This restriction is no longer needed. Most notably, the time between observations is now allowed to vary. However, in most cases one would most likely want the partition of $[t_k, t_{k+1}]$ to be equidistant.

The key correction is done in Theorem 5.1, where the auxiliary process $\widetilde{X}$ is evaluated in $t_{k+1} - t_{k,n+1}$ and $t_{k+1} - t_{k,n}$. During the proof of this theorem, a reparameterisation of time is done, such that $t = t_{k+1} - t'$ with $t' \in [t_{k+1} - t_{k,n+1}, t_{k+1} - t_{k,n})$. Note however, that while $t_{k+1}$ is a natural choice to subtract $t'$ from, the derivation does not rely on this. In fact, any time $\tau$ greater than $t_{k,n+1}$ can be chosen. Thus, make the more deliberate choice of $\tau = t_{k,n} + t_{k,n+1}$ such that

$$\begin{aligned} \tau - t_{k,n+1} &= t_{k,n}, \\ \tau - t_{k,n} &= t_{k,n+1}. \end{aligned}$$

The auxiliary process $\widetilde{X}$ is once again simulated using the Euler–Maruyama scheme according to

$$Z_{k,n+1} = Z_{k,n} + \mu(Z_{k,n})\,(t_{k,n+1} - t_{k,n}) + \sigma(Z_{k,n})\,(\widetilde{W}_{t_{k,n+1}} - \widetilde{W}_{t_{k,n}}). \tag{5.16}$$

As such, the approximation of $\widetilde{X}_{\tau - t_{k,n+1}}$ is $Z_{k,n}$ and the approximation of $\widetilde{X}_{\tau - t_{k,n}}$ is $Z_{k,n+1}$. The updated optimisation problem, which should be compared to (5.15), reads

$$\begin{aligned} &\big(\widetilde{\pi}_{k,n+1}(x,y)\big)_{(x,y)\in\mathbb{R}^d\times\mathbb{R}^{d'\times(k+1)}} \\ &\quad = \operatorname*{argmin}_{u\in C(\mathbb{R}^d\times\mathbb{R}^{d'\times(k+1)};\mathbb{R})} \mathbb{E}\Big[\Big|u(Z_{k,n}, Y_{0:k}) - \big(\widetilde{\pi}_{k,n}(Z_{k,n+1}, Y_{0:k}) \\ &\qquad\qquad + F\widetilde{\pi}_{k,n}(Z_{k,n+1}, Y_{0:k})\,(t_{k,n+1} - t_{k,n})\big)\Big|^2\Big]. \end{aligned} \tag{5.17}$$

The initial condition for $n = 0$ remains as

$$\widetilde{\pi}_{k,0}(x, y_{0:k}) = \frac{p(Y_k = y_k \mid X_{t_k} = x)\,\widetilde{\pi}_{k-1,N}(x, y_{0:k-1})}{\int_{\mathbb{R}^d} p(Y_k = y_k \mid X_{t_k} = x)\,\widetilde{\pi}_{k-1,N}(x, y_{0:k-1})\,\mathrm{d}x}. \tag{5.18}$$

### 5.2.3 On Normalisation During Training

In this final section before the neural network approximation is introduced, we focus on the necessity of calculating the denominator in (5.18). While no adjustments are

made to the method here, some alternatives are discussed.

The initial condition comes from Bayes' law and this far the normalising constant has been kept. This is potentially problematic however. The reason is that computing this constant in high dimensions is computationally demanding and could slow down the training process. As of now, $\widetilde{\pi}$ from the previous sections is a normalised density that integrates to 1. If instead the initial condition is updated to

$$\widetilde{\pi}_{k,0}(x, y_{0:k}) = p(Y_k = y_k \mid X_{t_k} = x)\, \widetilde{\pi}_{k-1,N}(x, y_{0:k-1}),$$

the resulting densities become unnormalised. This in itself is not a problem, as the normalisation can be performed when the filtering distribution is evaluated. In many situations the normalisation constant is not even needed. One notable example is when sampling from $\widetilde{\pi}$ using Markov Chain Monte Carlo (MCMC).

The problem with replacing the initial condition instead arises during training. Note that $\widetilde{\pi}_{k-1,N}(x, y_{0:k-1})$ in (5.18) represents a predictive distribution for the state in $t_k$ conditioned on $Y_{0:k-1} = y_{0:k-1}$. The overlap in mass for this distribution and the likelihood $p(Y_k = y_k \mid X_{t_k} = x)$ might be low and a direct multiplication of these without normalisation thus causes $\widetilde{\pi}_{k,0}(x, y_{0:k})$ to be significantly smaller than $\widetilde{\pi}_{k-1,N}(x, y_{0:k-1})$. This has significant consequences, since $\widetilde{\pi}_{k,0}(x, y_{0:k})$ is used to calculate the next density. The loss in size can accumulate through the training process, causing the integrals of later distributions to be orders of magnitude smaller than earlier ones. Eventually, this can causes numerical errors and make the training convergence difficult. As such, the normalisation constant in the initial condition is kept here. Some results indicating the necessity of this choice are presented in Section 7.7.

### 5.2.4 Final Algorithm

In summary, the recursive optimisation problem obtained this far is (5.17) for $n \in \{0, \dots, N-1\}$ with initial condition for $n = 0$ given in (5.18). To solve the optimisation problem the expectation is not calculated analytically. Instead it is estimated using Monte Carlo simulation. We define approximate densities $\hat{\pi}_{k,n}(x, y)$ for $n = 0, \dots, N$ satisfying the modified optimisation problem

$$\begin{aligned}
&\left(\hat{\pi}_{k,n+1}(x, y)\right)_{(x,y)\in\mathbb{R}^d\times\mathbb{R}^{d'\times(k+1)}} \\
&\quad = \operatorname*{argmin}_{u\in C(\mathbb{R}^d\times\mathbb{R}^{d'\times(k+1)};\mathbb{R})} \frac{1}{M}\sum_{m=1}^{M}\Big[\Big|u(z^m_{k,n}, y^m_{0:k}) - \big(\hat{\pi}_{k,n}(z^m_{k,n+1}, y^m_{0:k}) \\
&\qquad\qquad + F\hat{\pi}_{k,n}(z^m_{k,n+1}, y^m_{0:k})\,(t_{k,n+1} - t_{k,n})\big)\Big|^2\Big],
\end{aligned} \tag{5.19}$$

with initial condition

$$\hat{\pi}_{k,0}(x, y) = \frac{p(Y_k = y_k \mid X_{t_k} = x)\, \hat{\pi}_{k-1,N}(x, y_{0:k-1})}{\int_{\mathbb{R}^d} p(Y_k = y_k \mid X_{t_k} = x)\, \hat{\pi}_{k-1,N}(x, y_{0:k-1})\,\mathrm{d}x}.$$

Here $z^m = Z(\omega_m)$ is a specific sample path of $Z$ and $y^m = Y(\omega_m)$ a sample path of the observation process $Y$. The number of Monte Carlo sample paths used is $M$.

In practice, to sample from $Y$ we use the Euler–Maruyama method to simulate the state $X$ with the same grid as $Z$ and then form observations using the model (5.1).

Finally, in order for this problem to be solveable in practice, we replace the optimisation over the infinite-dimensional $C(\mathbb{R}^d \times \mathbb{R}^{d' \times (k+1)}; \mathbb{R})$ with an optimisation over the finite-dimensional parameter space $\Theta$ of a neural network $\mathcal{N}_\theta \colon \mathbb{R}^d \times \mathbb{R}^{d' \times (k+1)} \to \mathbb{R}$ with some sufficient architecture. We have $\{\mathcal{N}_\theta \mid \theta \in \Theta\} \subset C(\mathbb{R}^d \times \mathbb{R}^{d' \times (k+1)}; \mathbb{R})$ and the idea is that the network is versatile enough that, with the right parameters, it can approximate the true solution to (5.19) reasonably well. Additionally, to guarantee non-negativity of the solution, we let the neural network output energy. This approach is exactly the same as in [2] and means that the neural network $u(x, y) = \mathcal{N}_\theta(x, y)$ is replaced with $u(x, y) = \exp(-\mathcal{N}_\theta(x, y))$.

Now, let $\bar{\pi}_{k,n}$ denote the neural network approximation of the state probability density function in time $t_{k,n}$ for $n = 0, \dots, N$. With these modifications, (5.19) is approximated with

$$\begin{aligned}
&\left(\bar{\pi}_{k,n+1}(x, y)\right)_{(x,y) \in \mathbb{R}^d \times \mathbb{R}^{d' \times (k+1)}} \\
&\quad = \operatorname*{argmin}_{u \in \{\exp(-\mathcal{N}_\theta) \,|\, \theta \in \Theta\}} \frac{1}{M} \sum_{m=1}^{M} \Big[ \Big| u(z^m_{k,n}, y^m_{0:k}) - \Big( \bar{\pi}_{k,n}(z^m_{k,n+1}, y^m_{0:k}) \\
&\qquad\qquad + F\bar{\pi}_{k,n}(z^m_{k,n+1}, y^m_{0:k}) \, (t_{k,n+1} - t_{k,n}) \Big) \Big|^2 \Big].
\end{aligned} \tag{5.20}$$

This is the final recursive optimisation scheme and constitutes the most important equation in this thesis. Note that the objective function is the mean squared error, see (3.2), of the neural network output compared to given labels. As such, solving (5.20) fits into the framework of supervised learning well. The initial condition is once again

$$\bar{\pi}_{k,0}(x, y) = \frac{p(Y_k = y_k \mid X_{t_k} = x) \, \bar{\pi}_{k-1,N}(x, y_{0:k-1})}{\int_{\mathbb{R}^d} p(Y_k = y_k \mid X_{t_k} = x) \, \bar{\pi}_{k-1,N}(x, y_{0:k-1}) \, \mathrm{d}x}. \tag{5.21}$$

Assuming that $t_0 = 0$, i.e., the first observation is at time 0, the neural network training chain is started for $k = 0$ using

$$\bar{\pi}_{0,0}(x, y) = \frac{p(Y_0 = y \mid X_0 = x) \, p_0(x)}{\int_{\mathbb{R}^d} p(Y_0 = y \mid X_0 = x) \, p_0(x) \, \mathrm{d}x}. \tag{5.22}$$

As a complement to (5.20)–(5.22), the training algorithm is given in pseudo-code in Algorithm 1. Moreover, the procedure when evaluating the filtering density is given in Algorithm 2. However, these descriptions are given without much detail. Instead, the reader is referred to Chapter 6 for the implementation used in this thesis. Note that, as desired, the filter derived in this chapter is computationally light to evaluate online, as neural network evaluation can be done rather quickly. Training can be performed offline if the dynamics of the problem are known, which is often the case.

**Algorithm 1** Training of EBDS model

Simulate $M$ sample paths of $Z$ with (5.16)
Simulate $M$ sample paths of $X, Y$ using the Euler–Maruyama method.
**for** $k = 0, 1, \dots, K-1$ **do**
    Calculate $M$ labels using (5.21) or (5.22) inserted into:
    $\ell_m = \bar{\pi}_{k,0}(z^m_{k,1}, y^m_{0:k}) + F\bar{\pi}_{k,0}(z^m_{k,1}, y^m_{0:k})(t_{k,1} - t_{k,0})$
    Train $\bar{\pi}_{1,k}$ using (5.20)
    **for** $n = 1, \dots, N-1$ **do**
        Calculate $M$ labels using $\bar{\pi}_{k,n}$ inserted into:
        $\ell_m = \bar{\pi}_{k,n}(z^m_{k,n+1}, y^m_{0:k}) + F\bar{\pi}_{k,n}(z^m_{k,n+1}, y^m_{0:k})\,(t_{k,n+1} - t_{k,n})$
        Train $\bar{\pi}_{k,n+1}$ using (5.20)
    **end for**
**end for**

**Algorithm 2** Evaluation of EBDS filtering density in $t_k$ given $y_{0:k}$

Select the neural network in $t_k$
Perform one forward propagation $\bar{\pi}_{k-1,N}(x, y_{0:k-1})$
Calculate likelihood $p(Y_k = y_k \mid X_{t_k} = x)$
Find normalising constant $c$
**return** $\bar{\pi}_{k-1,N}(x, y_{0:k-1})\, p(Y_k = y_k \mid X_{t_k} = x)/c$

## 5.3 On Approximations and Convergence

In the derivation of the method, there are four main approximation steps. The first one is the transition from $p$, the exact density, to $\pi$ by utilisation of a forward Euler scheme in (5.6). The second step is when approximating $\pi$ with $\widetilde{\pi}$ defined in (5.12), by simulating the auxiliary process with the Euler–Maruyama scheme. The third step is in the Monte Carlo approximation of $\widetilde{\pi}$ with $\hat{\pi}$ in (5.19). Finally, $\hat{\pi}$ is approximated with $\bar{\pi}$ in (5.20) using neural networks. It can be seen that the quality of the first two approximations depends on the number of prediction steps $N$, whereas the third depends on the number of samples $M$ and the final on the accuracy and training of the neural network.

For further discussion, we introduce the notation $p\colon \{0, \dots, K\} \times \Omega \to L^\infty(\mathbb{R}^d; \mathbb{R})$ for the density valued stochastic process representing the exact filtering distribution in time $t_k$, $k \in \{0, \dots, K\}$. Given $\omega \in \Omega$ with associated $y_{0:k}$, $p_k(x)$ is simply the true filtering distribution in $t_k$. The same notation is used for the filters associated with $\pi$, $\widetilde{\pi}$, $\hat{\pi}$ and $\bar{\pi}$. Now, a natural approach for quantifying the error when using $\bar{\pi}$ instead of $p$ is to consider the $L^2(\Omega; L^\infty(\mathbb{R}^d; \mathbb{R}))$-error. Assuming all density valued stochastic processes are in this $L^p$-space, we want to establish a bound for

$$\|p_k - \bar{\pi}_k\|_{L^2(\Omega; L^\infty(\mathbb{R}^d; \mathbb{R}))} = \sqrt{\mathbb{E}\left[\sup_{x \in \mathbb{R}^d} |p_k(x) - \bar{\pi}_k(x)|^2\right]}, \quad k \in \{0, \dots, K\}. \tag{5.23}$$

According to the triangle inequality, this error can be split into three terms according to

$$\|p_k - \bar{\pi}_k\|_{L^2(\Omega;L^\infty(\mathbb{R}^d;\mathbb{R}))} \leq \|p_k - \widetilde{\pi}_k\| + \|\widetilde{\pi}_k - \hat{\pi}_k\| + \|\hat{\pi}_k - \bar{\pi}_k\|.$$

The first term is dependent on the number of intermediate prediction steps $N$. The size of the second term depends entirely on the quality of the Monte Carlo estimate, whereas the size of the third term depends on the how well the neural network can approximate the given function. Neural networks are generally used for their ability to accurately approximate functions and the number of samples can be increased so that the second term is small. Hence, it is a reasonable assumption that the first term dominates and is key in reducing the error.

In a forthcoming paper by Andersson, Bågmark and Larsson, the authors of [2], the first term is proved to be bounded according to

$$\|p_k - \widetilde{\pi}_k\|_{L^2(\Omega;L^\infty(\mathbb{R}^d;\mathbb{R}))} \leq CN^{-1/2}, \tag{5.24}$$

with a constant $C \in \mathbb{R}_{>0}$ independent of both $N$ and $T$. Similar to the Euler–Maruyama method, see Theorem 2.1, this can be referred to as strong convergence of order 1/2. If the Monte Carlo error and neural network approximation error is kept small enough, then the bound applies for $\|p_k - \bar{\pi}_k\|_{L^2(\Omega;L^\infty(\mathbb{R}^d;\mathbb{R}))}$ as a whole. One of the main goals of this thesis is to verify this result empirically. For this purpose and to generally benchmark performance against other methods, EBDS was implemented and tested on several example problems. The details are presented in the next chapters. It must be noted that the proof of (5.24) assumes properties that are not satisfied in all example problems treated in this thesis. One notable such assumption is uniform ellipticity of the matrix $a(x) = \sigma(x)\sigma(x)^T$, which is not true for geometric Brownian motion. Overall though we are interested in testing the method even when such assumptions are violated.

# 6
# Implementation

In the previous chapter the filtering method referred to as Energy-Based Deep Splitting (EBDS) was derived. The chapter ends with (5.20)–(5.22), which describe how the neural networks for the filter should be trained. Moreover, Algorithm 1 describes the training process in pseudo-code and Algorithm 2 describes how to evaluate the filter after training. The purpose of this chapter is to complement these descriptions further, with more detailed explanations of the filter implementation as done in this thesis. In the first section, the training is explained more in depth and in the following section the neural network architectures are described. Lastly, the metrics used for evaluation are presented.

## 6.1 Training Algorithm

The implementation and evaluation of EBDS was done in Python using the deep learning framework PyTorch. After training, the result is a sequence of neural networks $(\bar{\pi}_{k,n})_{k=0,n=1}^{K,N}$ representing approximations of the normalised predictive distributions in $(t_{k,n})_{k=0,n=1}^{K,N}$ given the available observations. Here, the key steps in the training of this neural network sequence are outlined from an implementation perspective. Note that in this section and henceforth, we treat the exponential function $x \mapsto \exp(-x)$ in (5.20) as part of the networks.

The meaning of (5.20) is that the neural network $\bar{\pi}_{k,n+1}(x, y)$ for time $t_{k,n+1}$ is trained using the MSE loss function with input-label pairs $((z_{k,n}^m, y_{0:k}^m), \ell_m)_{m=1}^M$ such that

$$\ell_m = \bar{\pi}_{k,n}(z_{k,n+1}^m, y_{0:k}^m) + F\bar{\pi}_{k,n}(z_{k,n+1}^m, y_{0:k}^m)\,(t_{k,n+1} - t_{k,n}). \tag{6.1}$$

As such, the operator $F$, defined in Lemma 5.1, is critical. To repeat, for $f \in C^2(\mathbb{R}^d; \mathbb{R})$ the operator is

$$\begin{aligned}(Ff)(x) = &-2\sum_{i=1}^{d}\mu_i(x)\frac{\partial f(x)}{\partial x_i} - \sum_{i=1}^{d} f(x)\frac{\partial \mu_i(x)}{\partial x_i} \\ &+ \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d} f(x)\frac{\partial^2 a_{ij}(x)}{\partial x_i \partial x_j} + \sum_{i=1}^{d}\sum_{j=1}^{d}\frac{\partial f(x)}{\partial x_i}\frac{\partial a_{ij}(x)}{\partial x_j}.\end{aligned}$$

As the drift and diffusion are assumed to be known, the gradients of these can be precomputed analytically. Moreover, since the training is performed sequentially, $\bar{\pi}_{k,n}(z_{k,n+1}^m, y_{0:k}^m)$ is available for evaluation. What remains is to evaluate partial derivatives of $\bar{\pi}_{k,n}(x, y)$ with respect to the first argument. For $n \in \{1, \ldots, N-1\}$,

$\bar{\pi}_{k,n}$ simply consists of a network and these derivatives can be calculated numerically using standard PyTorch functionality. While this can be done for $\bar{\pi}_{k,0}$ as well, instead the product rule was utilised together with analytic formulas for the likelihood gradient.

More in detail, the filtering density in $t_k$, $\bar{\pi}_{k,0}$, is formed according to

$$\bar{\pi}_{k,0}(x, y_{0:k}) = \frac{1}{c}\,\mathrm{N}(y_k \mid h(x), \Sigma)\,\bar{\pi}_{k-1,N}(x, y_{0:k-1}).$$

As a reminder, $h\colon \mathbb{R}^d \to \mathbb{R}^{d'}$ is the measurement function and $\Sigma \in \mathbb{R}^{d'\times d'}$ is the measurement noise covariance matrix. The normaliser $c$, the integral in (5.21), is a constant with respect to $x$ but dependent on $y_{0:k}$. Now, to calculate $\nabla_x \bar{\pi}_{k,0}(x, y_{0:k})$ for the label function, the product rule is used by

$$\begin{aligned}\nabla_x \bar{\pi}_{k,0}(x, y_{0:k}) = \frac{1}{c}\Big[&\bar{\pi}_{k-1,N}(x, y_{0:k-1})\,\nabla_x \mathrm{N}(y_k \mid h(x), \Sigma) \\ &+ \mathrm{N}(y_k \mid h(x), \Sigma)\,\nabla_x \bar{\pi}_{k-1,N}(x, y_{0:k-1})\Big].\end{aligned} \tag{6.2}$$

The second term is calculated using automatic differentiation with PyTorch. The first term can be calculated analytically using the formula

$$\nabla_x \mathrm{N}(y_k \mid h(x), \Sigma) = H(x)^T\,\Sigma^{-1}(y_k - h(x))\,\mathrm{N}(y_k \mid h(x), \Sigma),$$

where $H(x)$ is the Jacobian of the measurement function. In the very first step, when calculating $\nabla_x \bar{\pi}_{0,0}(x, y_0)$, the previous network $\bar{\pi}_{k-1,N}(x, y_{0:k-1})$ in (6.2) is replaced with the prior $p_0(x)$. The gradient of the prior $\nabla p_0(x)$ was also calculated analytically in the implementation.

It can be seen that another critical part of the training process is the calculation of the normalising constant $c$ in the update step. As the constant depends on $y_{0:k}$, this calculation must be performed for each training sample $m$. Explicitly, we want to calculate

$$c_m = \int_{\mathbb{R}^d} \mathrm{N}(y_k^m \mid h(x), \Sigma)\,\bar{\pi}_{k-1,N}(x, y_{0:k-1}^m)\,\mathrm{d}x, \quad m = 1, \ldots, M. \tag{6.3}$$

A fast method for computing these integrals numerically is essential, especially in high dimensions. Another desirable property is for the numerical method to be grid-free (in space), as in general the area of maximal probability mass can change from sample to sample. This is especially important if the samples can be expected to drift away from each other, for example when there is no state mean-reversion. For simplicity, in one-dimensional examples when the dynamics are mean-reverting we calculate (6.3) using quadrature. In higher dimensions or when the dynamics are more complicated, we use a more advanced method.

The task of numerically computing the integral in (6.3) can be compared to estimation of marginal likelihood, also known as evidence, in Bayesian statistics. This estimation problem is well explored in literature. A simple approach would be to

draw samples $(x_i)_{i=1}^I$ from the normalised predictive density $\bar{\pi}_{k-1,N}(x, y_{0:k-1}^m)$ and estimate the integral as the sample mean of $\mathrm{N}(y_k^m \mid h(x_i), \Sigma)$. Similarly, one could utilise that $\mathrm{N}(y_k^m \mid h(x), \Sigma) = \mathrm{N}(h(x) \mid y_k^m, \Sigma)$. Now, if the measurement function is invertible, the change of variable $x' = h(x)$ in (6.3) can be made to see that

$$c_m = \mathbb{E}_{X' \sim \mathrm{N}(y_k^m, \Sigma)} \left[ \bar{\pi}_{k-1,N}(h^{-1}(X'), y_{0:k-1}^m)\, J(X') \right],$$

where $J(x')$ is the determinant of the Jacobian of $h^{-1}(x')$. This expectation is then easily estimated using sampling. Note that this is essentially the approach taken in [13], in which a similar filtering method to ours is proposed. Both of these estimation methods suffer from the same problem however: it could happen that the two factors in (6.3) do not overlap to a large extent. This leads to an unstable estimate. Our approach is instead to estimate the integral using importance sampling from a wider distribution. Let this importance distribution be given by $\mathrm{N}(x \mid \mu^m, \Sigma^m)$ for some $\mu^m$ and $\Sigma^m$. The integral in (6.3) is then

$$c_m = \mathbb{E}_{X'' \sim \mathrm{N}(\mu^m, \Sigma^m)} \left[ \frac{\mathrm{N}(y_k^m \mid h(X''), \Sigma)\, \bar{\pi}_{k-1,N}(X'', y_{0:k-1}^m)}{\mathrm{N}(X'' \mid \mu^m, \Sigma^m)} \right].$$

Approximate normalisation constants for $m = 1, \ldots, M$ are thus each calculated using $I$ samples $(x_i'')_{i=1}^I$ with

$$\hat{c}_m = \frac{1}{I} \sum_{i=1}^{I} \left[ \frac{\mathrm{N}(y_k^m \mid h(x_i''), \Sigma)\, \bar{\pi}_{k-1,N}(x_i'', y_{0:k-1}^m)}{\mathrm{N}(x_i'' \mid \mu^m, \Sigma^m)} \right]. \tag{6.4}$$

Good choices for the mean $\mu^m$ and covariance $\Sigma^m$ depend on the problem at hand. In the particular example when $h$ is invertible $\mu^m = h^{-1}(y_k^m)$ and $\Sigma^m = a\Sigma$ for $a > 1$ are natural choices. One could also let the mean and covariance be given by some Kalman filter, for instance the extended or unscented variants. In the future, if this importance sampling scheme proves to be insufficient and an even more stable approach is required, so called bridge sampling would be a possible next step, see for instance [20].

Apart from the calculation of normalising constants in the update steps, before training each network the labels from (6.1) are normalised according to

$$\ell_m' = \frac{\ell_m}{\frac{1}{M} \sum_{m=1}^{M} \ell_m}. \tag{6.5}$$

The purpose of this is mainly to prevent the labels from becoming too small in high dimensions. An important detail is that in the training stage the denominator in (6.5) is saved for each network. During evaluation, the network output is then multiplied with this constant, to maintain the property of $(\bar{\pi}_{k,n})_{k=0,n=1}^{K,N}$ being normalised predictive distributions. However, note that if a filtering distribution is of interest, a normalising constant will have to be calculated when performing evaluation, see Algorithm 2.

## 6.2 Neural Architectures

In Chapter 3, classical neural networks and long short-term memory networks were introduced. In this section, the particular architectures employed in this thesis are detailed.

Starting with the standard fully-connected network, the architecture is shown schematically in Figure 6.1. Note that this architecture is used for the network in all time steps. This allows us to copy the learned weights and biases from the previous network when training the next in the sequence, which speeds up training, improves consistency and ultimately seems to yield better results compared to when networks of varying architecture are used. As a consequence, the input size is kept constant and inputs not currently utilised are set to zero. Additionally, as can be seen in the figure, we use three hidden layers with constant size $n_h$, dependent on the example in question. The output layer has one neuron and linear activation. Finally, as mentioned, we consider the exponential function $x \mapsto \exp(-x)$ as part of the network.

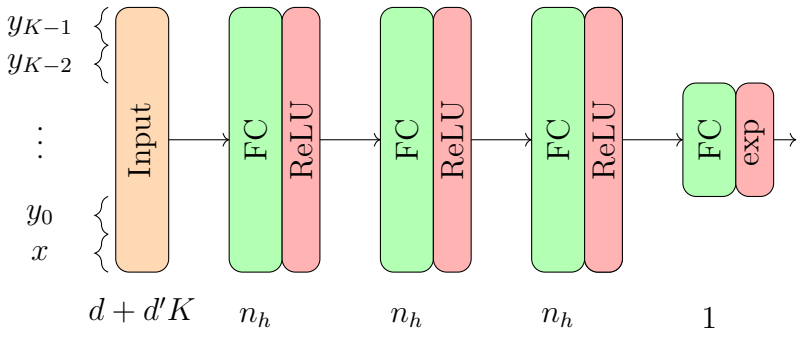


**Figure 6.1:** A schematic view of the fully-connected neural network used. Layer sizes are displayed under the respective layer. Most notably, the size of the input is kept constant, so that all networks have the same architecture.

Apart from a fully-connected architecture, we also use an LSTM-based architecture. The purpose is to examine if more advanced architectures can reduce the neural network approximation error. In particular, we use an LSTM architecture because the observations can be viewed as dependent samples from a time series. An additional advantage is that the cell structure of the LSTM allows the network to process sequences of varying lengths naturally. As such, we can initialise each network as a copy of the previous network without having to fix a constant input size and the somewhat artificial setting of future observations to zero can be avoided.

Critically however, the $x$-input which controls the point of evaluation of the density should be treated differently compared to the observations. One approach would be to concatenate $x$ with each observation and use these joint vectors as input to the LSTM cells. Instead, we use the LSTM layers as an encoder to produce a latent representation of the observations. This latent representation is then used together

with $x$ as input to a decoder consisting of a fully-connected network. LSTM encoders are common in sequence-to-sequence tasks, such as time series prediction, where they have proved to be effective, see for instance [29] and [41]. For an example architecture utilising an LSTM encoder together with auxiliary non-sequential input in the decoder, see [21]. However, note that although the approach of concatenating LSTM output with non-sequential input is similar to our approach, the decoder architecture in this article is quite different compared to ours.

An overview of the proposed hybrid LSTM and fully-connected model is shown in Figure 6.2. As can be seen, the output of the last LSTM cell is concatenated with the output from a layer that only operates on the $x$-input. This concatenated representation is then used as input to another hidden layer, whose output is fed to the final layer. Note that more LSTM cells are added for networks later in time. However, as the cells are identical in terms of weights and biases this does not pose a problem. A new network can always be initialised with the same parameters as the already trained previous one, resulting in faster training times and better results.

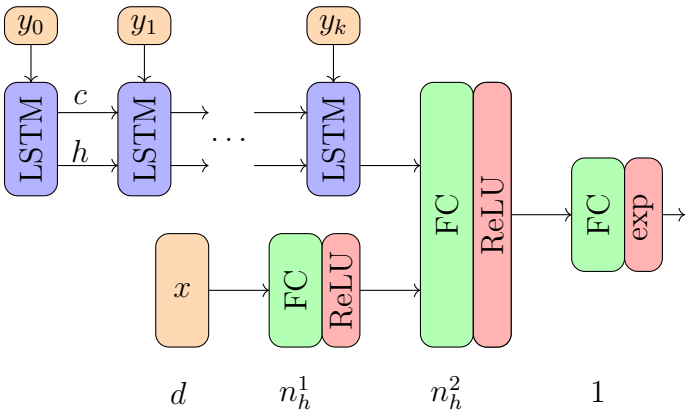


**Figure 6.2:** The hybrid model based on an LSTM encoder shown schematically. Again, the layer dimensions are specified below the respective layers. As for the LSTM output and cell state, they both have dimension $n_h^1$. This means that the input to the last hidden layer has length $2n_h^1$. For an explanation of the LSTM cell, see Section 3.2 and Figure 3.2.

For both the standard and LSTM architectures, the ADAM optimiser with standard parameters, apart from the learning rate, was used. In each separate example, initial tuning was performed to determine suitable hyperparameters, such as learning rate and layer sizes. In all examples however, a batch size of 512 was used and the networks were trained for a maximum of 100 epochs. Additionally, a validation set of size $0.1M$ was taken from the initial training set of size $M$. This validation set was utilised for early stopping of the training, with a patience of five epochs. Critically, the same $0.1M$ samples were used for validation across all times. This is important due to the fact that the previous network is copied. As such, including samples from its training set in the validation set of the current network would break independence assumptions.

In the output layer, additional neurons can be added if necessary to for instance control the tail behaviour. This is done in [2] to ensure that the networks extrapolate well. The idea is to use known information about the problem at hand, for example about the stationary distribution, and incorporate it into the model architecture. To give an example in one dimension, the basic output layer is

$$O(u) = \exp\left(-\xi_1(u)\right), \tag{6.6}$$

where $u$ is the output from the last hidden layer and $\xi_1$ is a neuron with linear activation. This can be replaced with

$$\widetilde{O}(u, x) = \exp\Big(-\xi_1(u) - (x - \xi_2(u))^2 \, \mathbb{1}_{|x|>\xi_2(u)}\Big). \tag{6.7}$$

As can be seen, $x$ is copied from the input of the network to the last layer and used in the second energy term. This second term ensures the tails of the distribution are Gaussian. In this thesis adding extra output terms is generally refrained from, but used in some examples, see Chapter 7.

## 6.3 Method Evaluation

In order to evaluate how well EBDS performs and compare it to the benchmark filters, certain metrics are used. These metrics are calculated by averaging over $M'$ test sample paths that are not part of the training data for EBDS. As before, the test sample paths are simulated using the Euler–Maruyama method. Note that we limit ourselves to evaluation of the performance in the measurement times, i.e., we only consider the accuracy of the filtering distributions and not the predictive distributions.

Most of the metrics require some reference filter that should be either exact or known to be close to the exact solution. In linear cases, we use the Kalman filter for this purpose whereas in nonlinear cases we use a particle filter with a large number of particles. To obtain a distribution with continuous support from the particle filter, a so called Gaussian Kernel Density Estimate (Gaussian KDE) is used.

The first metric of interest is the so called Mean Absolute Error (MAE). Let $x_k^m$ denote the state at time $t_k$ for sample path $m$ and let $\hat{\mu}_k^m$ denote the approximate mean given by the filter of interest. The MAE is then defined as

$$\mathrm{MAE}_k = \frac{1}{M'} \sum_{m=1}^{M'} \|x_k^m - \hat{\mu}_k^m\|, \quad k \in \{0, \ldots, K\}. \tag{6.8}$$

Note that this is not generally going to be zero even for exact filters. This means that it is formally not an error. Nevertheless, it helps to evaluate how well a filter performs, especially when accompanied with the next metrics.

The First Moment Error (FME) measures the deviation of an approximate filter

mean from the exact filter mean. Let $\mu_k^m$ denote the true mean of the filtering distribution at time $t_k$ for sample path $m$. The FME for the filter of interest is then defined as

$$\text{FME}_k = \frac{1}{M'} \sum_{m=1}^{M'} \|\mu_k^m - \hat{\mu}_k^m\|, \quad k \in \{0, \ldots, K\}. \tag{6.9}$$

Remark that this only captures the deviation of the first moment for an approximate filter.

The Kullback–Leibler Divergence (KLD), on the other hand, measures how well the density of the approximate filter matches the true density in its entirety. Also known as the relative entropy, the KLD is not strictly a metric, as it is not symmetric. Instead it exists in two variants: the forward KLD and the backward KLD. Here, we use the forward divergence denoted as $D_{\text{KL}}(p\|\hat{p})$, where $p(X_{t_k} \mid Y_{0:k} = y_{0:k}^m)$ is the exact density at time $t_k$ for sample path $m$ and $\hat{p}(X_{t_k} \mid Y_{0:k} = y_{0:k}^m)$ is the approximate density. Using the shorter notation $p_k^m(x) = p(X_{t_k} = x \mid Y_{0:k} = y_{0:k}^m)$ and $\hat{p}_k^m(x) = \hat{p}(X_{t_k} = x \mid Y_{0:k} = y_{0:k}^m)$, we define the forward KLD as

$$\text{KLD}_k = D_{\text{KL}}(p_k\|\hat{p}_k) = \frac{1}{M'} \sum_{m=1}^{M'} \int_{\mathbb{R}^d} p_k^m(x) \log\left(\frac{p_k^m(x)}{\hat{p}_k^m(x)}\right) \mathrm{d}x, \quad k \in \{0, \ldots, K\}. \tag{6.10}$$

Next, to examine convergence we use a metric approximating the $L^2(\Omega; L^\infty(\mathbb{R}^d; \mathbb{R}))$-error in (5.23). The expectation is approximated using a sample mean and the supremum is replaced with maximum. The resulting metric, henceforth called $L^2L^\infty$-error is

$$L^2L^\infty\text{-error}_k = \sqrt{\frac{1}{M'} \sum_{m=1}^{M'} \max_{x \in \mathbb{R}^d} |p_k^m(x) - \hat{p}_k^m(x)|^2}, \quad k \in \{0, \ldots, K\}. \tag{6.11}$$

While this metrics is mainly used to evaluate convergence, it is also interesting in itself because it measures the expected maximum difference between two filter distributions.

The last metric of interest is an approximation of the $L^2(\Omega; L^2(\mathbb{R}^d; \mathbb{R}))$-error, which we refer to as the $L^2L^2$-error. This metric is defined as

$$L^2L^2\text{-error}_k = \sqrt{\frac{1}{M'} \sum_{m=1}^{M'} \int_{x \in \mathbb{R}^d} |p_k^m(x) - \hat{p}_k^m(x)|^2 \, \mathrm{d}x}, \quad k \in \{0, \ldots, K\}. \tag{6.12}$$

We use this partly to examine convergence and partly because it is interesting in itself, similarly to the $L^2L^\infty$-error.

To obtain some idea of the convergence rate in an error when going from a number of prediction steps $N_i$ to a larger number $N_{i+1}$, the Empirical Order of Convergence (EOC) is utilised. This is formed as

$$\text{EOC}(N_i) = -\frac{\log(\text{error}(N_{i+1})) - \log(\text{error}(N_i))}{\log(N_{i+1}) - \log(N_i)}.$$

In this thesis, the error is always the $L^2L^\infty$-error. Using the EOC, the hypothesis that EBDS converges with order 1/2 in practice can be tested. For this hypothesis to hold the EOC should be above 1/2 for all values of $N$. Remark that this method for estimating convergence rate is well established in literature, see for instance [1].

From an implementation perspective, the metrics pose some challenges. Firstly, the evaluation of (6.10) and (6.12) require the calculation of possibly high-dimensional integrals, which is computationally demanding. Similarly, solving the optimisation problem in (6.11) and calculating the approximate mean in (6.8)–(6.9) in high dimensions is not trivial. Finally, for EBDS, the normalising constant for the filtering distribution must be calculated according to (6.3).

In one dimension when the state is mean-reverting, we use simple quadrature on a pre-specified grid to calculate the integrals in (6.10) and (6.12), as well as the mean and normalising constant for EBDS. The optimisation problem (6.11) is solved by considering the maximum on the grid. This is not feasible in high dimensions however and, as discussed in Section 6.1, grid-free methods are generally preferred.

In high dimensions and for dynamics that are not mean-reverting, we instead employ sampling to estimate the integrals. It is clear that (6.10) can be written as an expectation with respect to the distribution of the reference filter. Similarly, for (6.12), we have

$$\int_{x\in\mathbb{R}^d} |p_k^m(x) - \hat{p}_k^m(x)|^2 \,\mathrm{d}x = \mathbb{E}_{X\sim p_k^m}\left[\frac{|p_k^m(X) - \hat{p}_k^m(X)|^2}{p_k^m(X)}\right]. \tag{6.13}$$

Thus, we can estimate both the integral in (6.12) and the one in (6.10) by sampling from the exact filtering distribution, which is simple if it is given by a particle filter or Kalman filter. For the optimisation problem in the $L^2L^\infty$-error, we use a standard gradient based solver, started in the sample from $p_k^m$ with the largest objective value. Estimation of the normalising constant is done using importance sampling as specified in Section 6.1. Estimation of the mean for EBDS is done completely analogously. With the same notation as in (6.4), the formula is

$$\hat{\mu}_k^m = \frac{1}{\hat{c}_m\, I}\sum_{i=1}^{I}\left[\frac{x_i''\,\mathrm{N}(y_k^m \mid h(x_i''),\Sigma)\,\bar{\pi}_{k-1,N}(x_i'', y_{0:k-1}^m)}{\mathrm{N}(x_i'' \mid \mu^m, \Sigma^m)}\right]. \tag{6.14}$$

By using this procedure for evaluation of the metrics samples are reused where possible and duplicate calculations are avoided. Note, however, that if only one metric is of interest, there might be more efficient alternatives. For example, the mean can be evaluated using MCMC, which does not require the normalising constant.

# 7

# Results

The energy-based deep splitting method is here evaluated on four example problems, two in one dimension and two in higher dimensions. The examples are chosen to be quite different, in order to investigate the versatility of the method. In one dimension, the first example is an Ornstein–Uhlenbeck process and the second one is an SDE with bimodal dynamics. Convergence is investigated in both these examples. For the higher dimensional cases, the first example is a predator-prey model from mathematical biology and the second one is a stochastic linear spring-mass system. Each section begins with a more thorough description of the problem at hand after which EBDS performance is shown. After the specific examples, some results are presented pertaining to various parts in the EBDS method. In Section 7.5, the LSTM-model performance is evaluated and compared to the standard model for one example problem. This is followed by a runtime analysis in Section 7.6. Lastly, we evaluate whether there is a need for calculating the normalisation constant during the training.

For the first two example problems, we start the simulations at $t_0 = 0$ and end at $T = 1$. Observations are made starting at time 0 with 0.1 units of time in between, resulting in 11 observations in total. In the last two examples, which are in three and eight dimensions respectively, we instead let $T = 0.5$ and consider six observations. Note that all results except those in Section 7.5 were generated using the standard fully-connected neural network architecture. Model hyperparameters and other information necessary for reproducing the results can be found in the Appendix B. In all results for EBDS we average over five instances with different training data to obtain reliable metric estimates, although EBDS instances generally do not deviate significantly in terms of performance.

Two additional example problems are presented in Appendix A. In the first example the state is given by a one-dimensional geometric Brownian motion. The second example is in two dimensions and semi-inspired by stochastic volatility models from financial mathematics. Qualitatively the results in these two examples are similar to the ones presented in this chapter. As such, Appendix A can be ignored by a reader only interested in the overall performance of EBDS and not specific examples.

## 7.1 Ornstein–Uhlenbeck Process

The famous Ornstein–Uhlenbeck process is the solution to the Langevin equation

$$X_t = X_0 + \int_0^t -\theta\, X_s \,\mathrm{d}s + \int_0^t \sigma \,\mathrm{d}W_s, \quad t \in [0, T],$$

for some parameters $\theta, \sigma \in \mathbb{R}_{>0}$. The process is mean-reverting to mean zero and it can easily be shown that $\mathrm{N}(0, \sigma^2/\theta)$ is a stationary and limiting distribution. As this state dynamic, together with a linear measurement function and Gaussian prior, fits into the framework of the Kalman filter, it is a natural first example for evaluating EBDS.

In particular, we use the parameter values $\theta = 3$ and $\sigma = 1$. As mentioned we consider the final time $T = 1$. Furthermore, the standard normal distribution is chosen as the prior and we assume the measurement noise has variance 1. As the measurement function is $h(x) = x$, it should be noted that the signal-to-noise ratio is quite low. One could argue that this makes the filtering problem more difficult.

To investigate convergence, we evaluate EBDS performance on the test set using six different numbers of prediction steps $N$: $1, 2, 4, 8, 16$ and 32 steps. Additionally, as mentioned, for each number of steps and each time, the metrics are averaged over five instances to obtain more stable performance estimates. Using the Kalman filter as reference, the $L^2L^2$- and $L^2L^\infty$-errors over time are presented in Figure 7.1. Note that in both errors, there seems to be a clear convergence trend. The exception is for 32 steps at later times. An interesting detail is that the error seems to reach a maximum after some time has passed and then decreases later. A likely explanation is that the process tends towards the stationary distribution, which is easier for the model to learn than when the process is non-stationary.

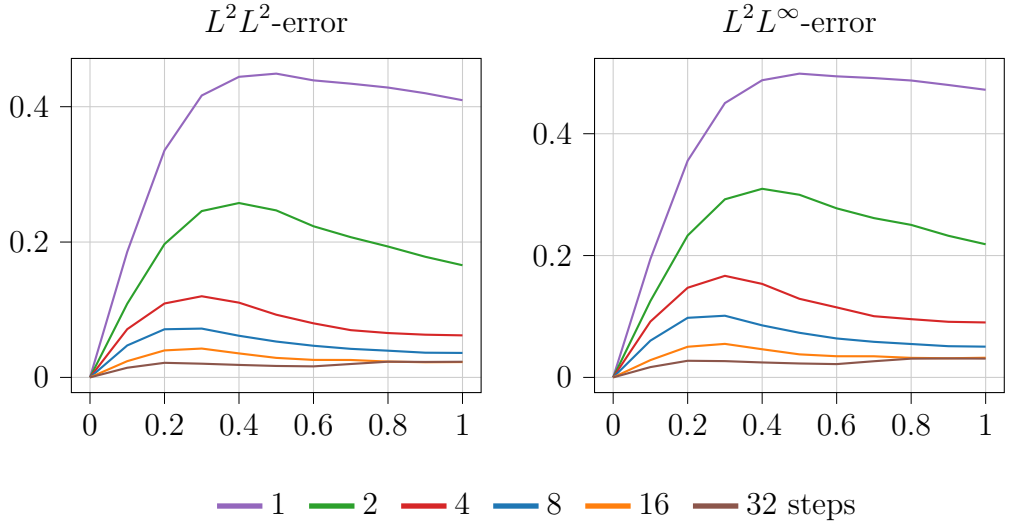


**Figure 7.1:** $L^2$-errors over time in the Ornstein–Uhlenbeck example for different numbers of prediction steps. As can be seen, there is a clear convergence.

The order of convergence for the $L^2L^\infty$-error is shown more clearly in Figure 7.2 at times 0.1 and 1. The corresponding error values and EOC can be found in Table 7.1. At time 0.1, convergence of at least approximately order 1/2 can be seen for all values of $N$. The pattern is less clear at time 1, but overall the EOC is above 0.5 for all $N$ except 32. A likely explanation for the error not decreasing significantly between 16 and 32 steps is that it is dominated by either the Monte Carlo term or the neural network approximation term for $N = 32$. Although not shown here, for fewer training samples the convergence starts to deteriorate already at 8 and 16 steps, which supports this theory.

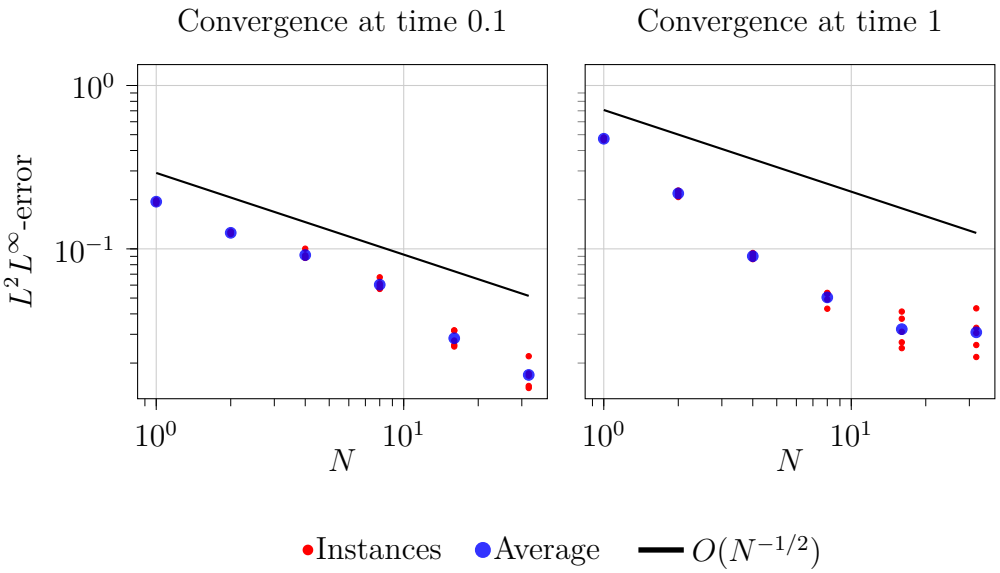


**Figure 7.2:** Convergence plots for the $L^2L^\infty$-error in the Ornstein–Uhlenbeck example. Apart from the average error, the error for each model instance is shown in red. The black line corresponds to convergence of order 1/2.

**Table 7.1:** Error and Experimental Order of Convergence (EOC) for different discretisation steps $N$ at the two time points in the Ornstein–Uhlenbeck example.

| | **Time 0.1** | | **Time 1** | |
|---|---|---|---|---|
| $N$ | $L^2L^\infty$-error | EOC | $L^2L^\infty$-error | EOC |
| 1 | 0.1945 | 0.6329 | 0.4721 | 1.1114 |
| 2 | 0.1254 | 0.4522 | 0.2185 | 1.2767 |
| 4 | 0.0917 | 0.6029 | 0.0902 | 0.8376 |
| 8 | 0.0603 | 1.0890 | 0.0505 | 0.6460 |
| 16 | 0.0284 | 0.7456 | 0.0323 | 0.0639 |
| 32 | 0.0169 | - | 0.0309 | - |

To benchmark EBDS, its average performance with $N = 32$ is compared to bootstrap particle filters with $10^2$, $10^3$, $10^4$ and $10^5$ particles. These are all set to have the same number of intermediate prediction steps, for a fair comparison. The MAE, FME, KLD and $L^2L^\infty$-error over time for the filtering distributions are displayed in Figure 7.3. The plot for the MAE shows that all filters generally manage to track the mean of the reference filter and that the average error compared to the true state decreases over time. In terms of the FME, EBDS is slightly worse than a particle filter with $10^4$ particles. For the KLD and $L^2L^\infty$-error, our filter generally outperforms the particle filter with $10^4$ particles but not the one with $10^5$. As EBDS takes a fraction of the time to evaluate the filtering density compared to the benchmarks, these results in the Ornstein–Uhlenbeck example are quite promising.

For a more intuitive demonstration of EBDS performance the reader is referred to Figure 7.4. This figure shows some filtering distributions for one randomly selected example in both one-dimensional problems. The Ornstein–Uhlenbeck example can be found in the first column and it can be seen that the EBDS solution matches the exact solution very well in all times. Next however, a more difficult example is investigated to see if EBDS performs as well in nonlinear cases.

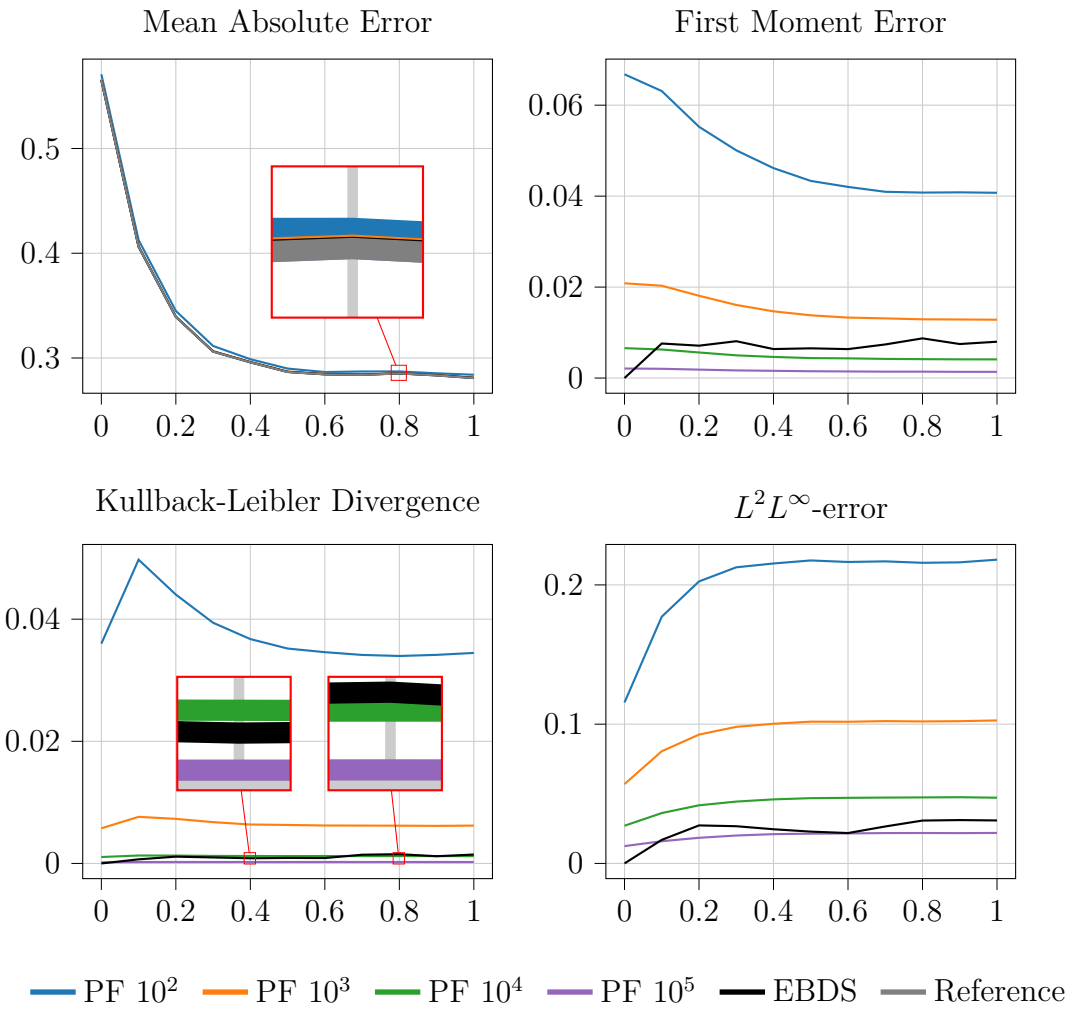


**Figure 7.3:** Four metrics over time for EBDS compared to benchmark particle filters in the Ornstein–Uhlenbeck case. The reference filter (a Kalman filter) is shown in the MAE plot.

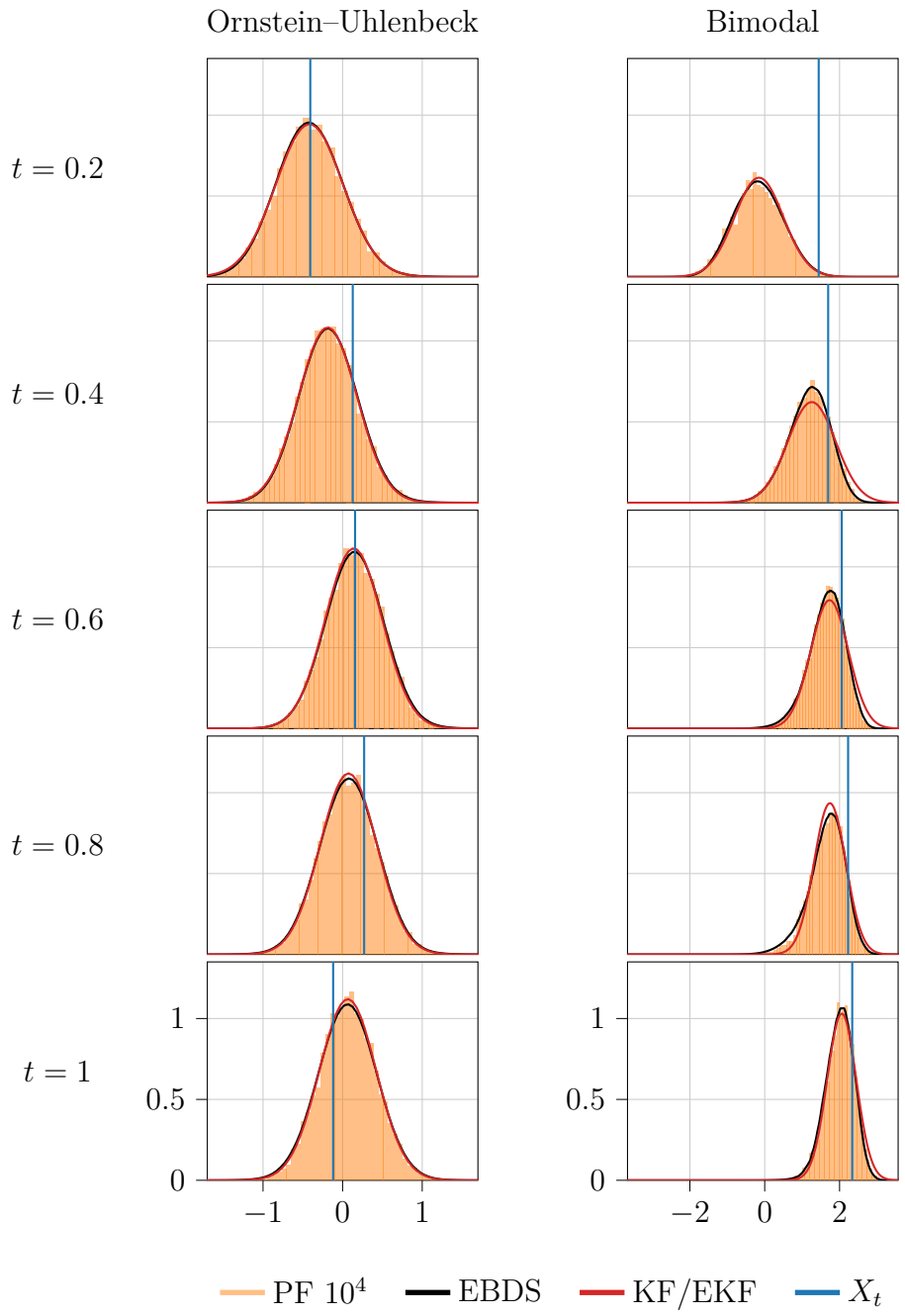


**Figure 7.4:** Filtering densities in one example for both one-dimensional problems. The true state is marked with a blue line. Note that axis scales are provided in the bottom plot in each column. In the first column, the Kalman filter is shown in red and represents the exact solution. It can be seen that EBDS matches this solution quite well. In the second column red instead represents the extended Kalman filter. Here, EBDS seems to align well with the particle filter, while the extended Kalman filter provides a less satisfactory solution

## 7.2 Bimodal SDE

As a first nonlinear example, we consider a case with significantly more complicated drift function. We now assume the state-measurement model is of the form

$$\begin{aligned} X_t &= X_0 + \int_0^t \frac{2}{5}(5X_s - X_s^3)\,\mathrm{d}s + \int_0^t \mathrm{d}W_s, \qquad t \in [0,1], \\ Y_k &= X_{t_k} + V_k, \qquad\qquad\qquad\qquad\qquad\quad k \in \{0,\dots,10\}. \end{aligned} \tag{7.1}$$

Once again, the prior is chosen as the standard normal distribution and $V_k$ has variance 1. As can be seen in Figure 7.5, which shows the histogram for $10^5$ simulated states at various times, the state tends to accumulate around $x = \pm\sqrt{5}$. Because of this bimodality in the unconditional distribution, one can expect the Kalman filters to perform poorly in this example. Instead, a bootstrap particle filter with $10^5$ particles is used as reference filter for the calculation of the metrics.

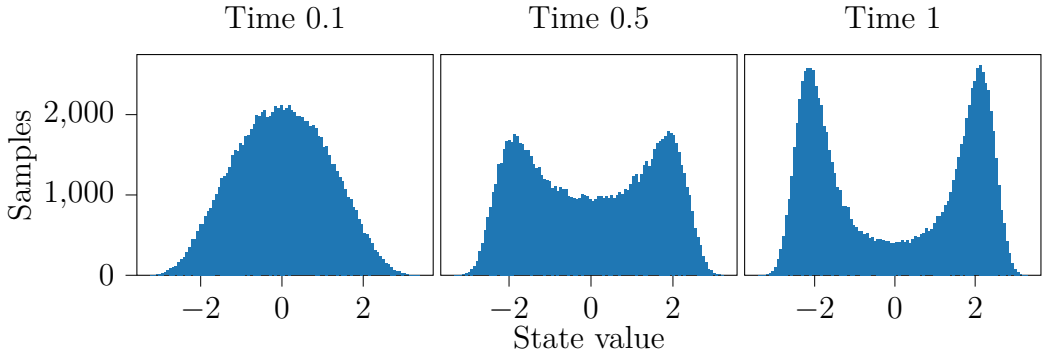


**Figure 7.5:** Histogram simulated using the Euler–Maruyama scheme and the dynamics (7.1) at various times.

Again, we start by examining convergence. Figure 7.6 shows the $L^2L^2$- and $L^2L^\infty$-error for $N = 1, 2, 4, 8, 16$ and 32 steps. Similar to the Ornstein–Uhlenbeck example, there seems to be convergence in both metrics. Moreover, for smaller numbers of steps the error increases drastically over time while more steps results in only slight or no increase over time. The order of convergence can be seen more clearly in Figure 7.7 as well as Table 7.2 and we note that overall the black line corresponding to order 1/2 fits the data quite well although singular EOC values are lower. There could be several reasons for the rather uneven EOC-values. As in the last example, the Monte–Carlo error or neural network approximation error could be dominating for larger $N$. Further, it can be seen in Figure 7.7 that EBDS instances with the same $N$ deviate quite much in terms of the error. It is possible that averaging over more instances would yield better EOC estimates. Finally, while the results indicate that EBDS does converge in this example, it must be noted that the dynamics neither satisfy the assumptions for convergence of the Euler–Maruyama scheme in Theorem 2.1 nor, in fact, the assumptions in the proof of EBDS convergence.

Evidently the best model is the one with $N = 32$ and we compare its performance with the benchmark filters, for which we also use 32 intermediate steps. In this case

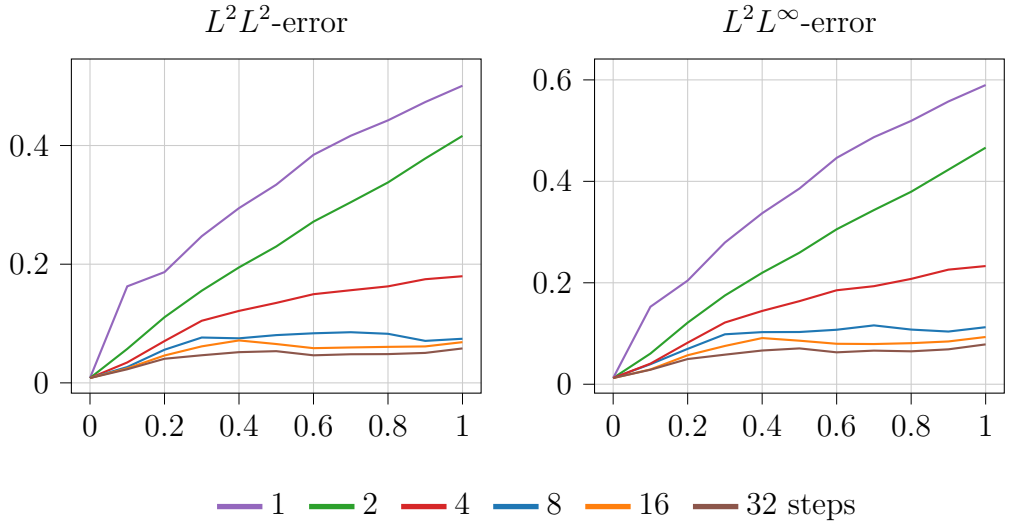


**Figure 7.6:** EBDS $L^2L^2$- and $L^2L^\infty$-error over time for different values of $N$ in the bimodal case. It can be seen that there is a clear convergence trend.

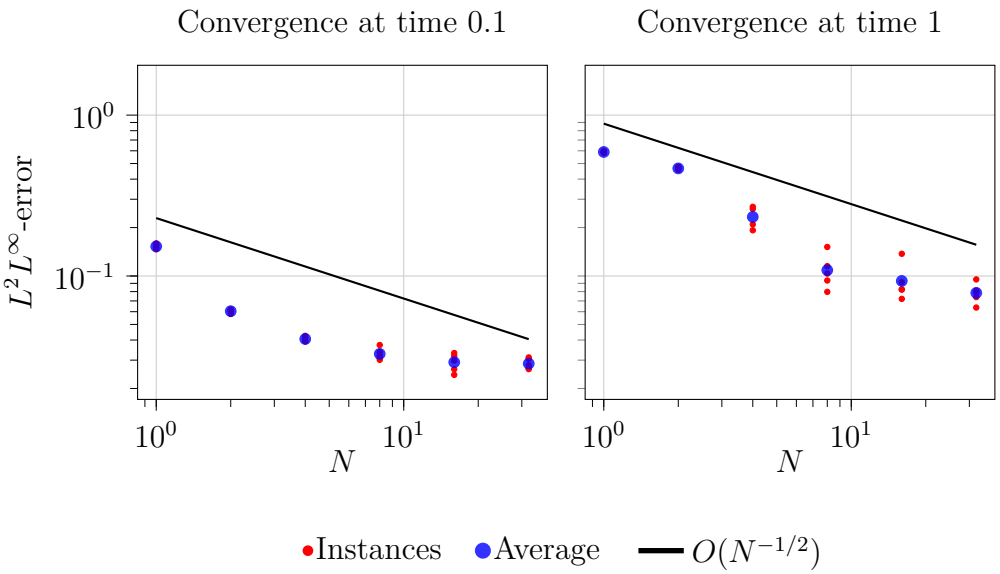


**Figure 7.7:** Convergence plots for the $L^2L^\infty$-error in the bimodal example. The error for each model instance is shown in red while the black line corresponds to convergence of order 1/2.

we benchmark against bootstrap particle filters as well as unscented and extended Kalman filters. The comparison is shown in Figure 7.8 and EBDS turns out to be significantly more accurate than the Kalman filters in all metrics. In terms of particle filters, our method consistently performs better than one with $10^3$ particles but worse than one with $10^4$ particles. The accuracy of EBDS, especially when compared to the extended Kalman filter, can further be seen in the second column of Figure 7.4. While EBDS aligns well with the particle filter histogram in all times the EKF deviates somewhat, particularly for $t = 0.4$, $t = 0.6$ and $t = 0.8$.

**Table 7.2:** Error and Experimental Order of Convergence (EOC) for different discretisation steps $N$ at the two time points in the bimodal example.

| | **Time 0.1** | | **Time 1** | |
|---|---|---|---|---|
| $N$ | $L^2L^\infty$-error | EOC | $L^2L^\infty$-error | EOC |
| 1 | 0.1527 | 1.3374 | 0.5899 | 0.3389 |
| 2 | 0.0604 | 0.5740 | 0.4664 | 1.0013 |
| 4 | 0.0406 | 0.3074 | 0.2330 | 1.0994 |
| 8 | 0.0328 | 0.1718 | 0.1087 | 0.2231 |
| 16 | 0.0291 | 0.0289 | 0.0932 | 0.2458 |
| 32 | 0.0285 | - | 0.0786 | - |

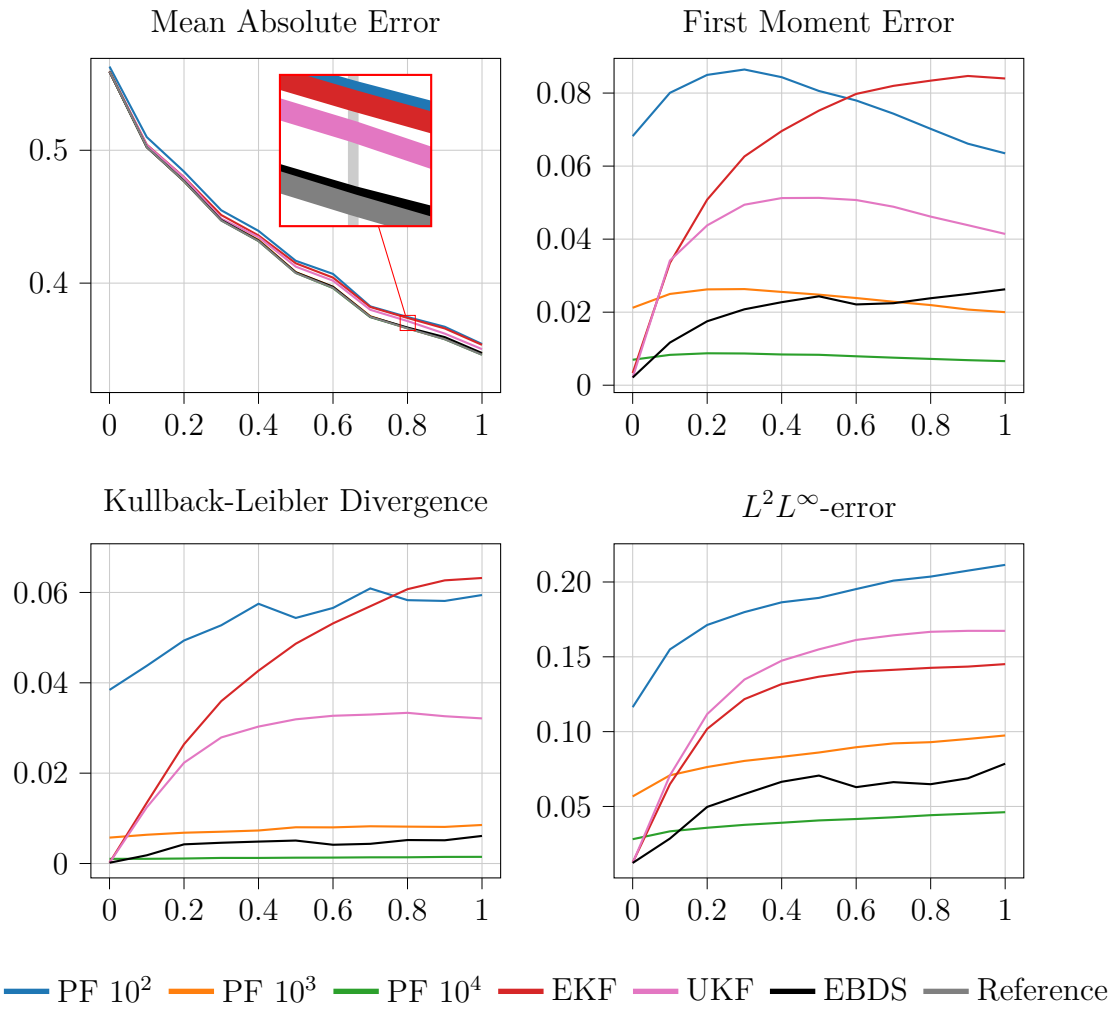


**Figure 7.8:** EBDS performance over time compared to the benchmarks on four metrics in the bimodal example. As can be seen, the Kalman filters are inaccurate compared to our method and the particle filters with many particles.

To summarise the one-dimensional examples, EBDS shows promising results. In the relatively simple Ornstein–Uhlenbeck example performance is excellent and this applies to a large extent in the bimodal example as well. The geometric Brownian motion example presented in the appendix is more difficult but EBDS remains relatively accurate, although performance is worse in relation to the benchmark particle filters. As for the convergence, both the Ornstein–Uhlenbeck example and bimodal example have some EOC-values below 0.5. Nevertheless, the convergence trends are clear and the lines corresponding to order 1/2 fit the data quite well. Next we start to investigate how the method scales with the state dimension by considering an example in three dimensions followed by one in eight dimensions. Additionally, a problem with state dimension two is presented in Appendix A.

## 7.3 Stochastic Predator-Prey Model

The Lotka–Volterra predator-prey model is a system of ordinary differential equations proposed in the early 1900s commonly used to model populations of interacting animals. Various methods for incorporating randomness into the model exist and three different diffusion models are suggested in [43]. In this article two stochastic processes for populations are considered: one for rabbits, $R_t$, and one for foxes $F_t$. One of the stochastic models that is proposed reads

$$\begin{aligned} R_t &= R_0 + \int_0^t R_s \left(2 - \alpha F_s\right) \mathrm{d}s + \int_0^t \sigma_1 R_s \, \mathrm{d}W_s^{(1)}, \\ F_t &= F_0 + \int_0^t F_s \left(\alpha R_s - 1\right) \mathrm{d}s + \int_0^t \sigma_2 F_s \, \mathrm{d}W_s^{(2)}, \end{aligned}$$

for some parameters $\alpha$, $\sigma_1$ and $\sigma_2$. Here, we extend this model for a third population, namely mice, and denote the corresponding process $M_t$. Mice are eaten by the foxes, but do not directly interact with the rabbits (we assume). Additionally, we consider population densities $r, m$ and $f$ rather than absolute sizes. As such, the state model used in this example is for $t \in [0, 0.5]$

$$\begin{aligned} r_t &= r_0 + \int_0^t r_s \left(\alpha_1 - \alpha_2 f_s\right) \mathrm{d}s + \int_0^t \sigma_1 r_s \, \mathrm{d}W_s^{(1)}, \\ m_t &= m_0 + \int_0^t m_s \left(\beta_1 - \beta_2 f_s\right) \mathrm{d}s + \int_0^t \sigma_2 m_s \, \mathrm{d}W_s^{(2)}, \\ f_t &= f_0 + \int_0^t f_s \left(\alpha_2 r_s + \beta_2 m_s - \gamma\right) \mathrm{d}s + \int_0^t \sigma_3 f_s \, \mathrm{d}W_s^{(3)}. \end{aligned} \tag{7.2}$$

The parameter values considered are $\alpha_1 = \beta_1 = \beta_2 = 1$, $\alpha_2 = 0.75$, $\gamma = 2$ as well as $\sigma_1 = 0.8$, $\sigma_2 = 1$ and $\sigma_3 = 0.5$. Each starting population is assumed to be independent of the other ones and the prior is normal with means $1, 1.5$ and $0.5$ for the rabbits, mice and foxes respectively. The standard deviation for each prior is 1/5 of the mean. These parameters ensure that each population remains approximately close to the prior mean over time, although significant deviations can occur for individual samples. This fact can be seen in Figure 7.9, which shows histograms for the three populations initially and at $t = 0.5$. As for the observations, we assume all populations can be measured directly, i.e., $h(X_t) = (r_t, m_t, f_t)^T$. The

standard deviation of the measurement noise is set to 0.5, which is unrealistically high in reality but offers an interesting challenge for filtering.

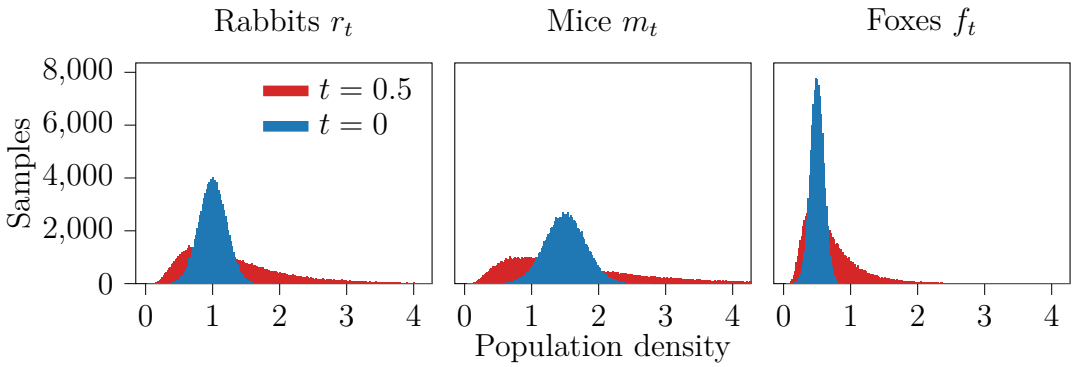


**Figure 7.9:** Histograms for the population densities from (7.2) at the initial and final times.

A crucial detail in this case is that we alter the neural network output layer with problem specific information. This is needed, as there are few, if any, training samples with negative state. As such, the neural network will struggle to learn that the density should always be zero for these states. We alter the output layer according to

$$O(u) = \exp\left(-\xi_1(u)\right) \to \widetilde{O}(u, r, m, f) = \exp\left(-\xi_1(u)\right) \mathbb{1}_{r,m,f>0},$$

with $x = (r, m, f)^T$ being the state value and other notation as in (6.6)–(6.7).

In this and the next example we refrain from investigating convergence. Instead we fix $N = 16$ and examine EBDS performance compared to the benchmark filter directly. We let the benchmark filters have 16 intermediate steps so that the comparison is fair. This value for $N$ is chosen as it constitutes a good trade-off between training time and performance for EBDS. While the last examples show that the error can be reduced by using $N = 32$, this involves training twice as many neural networks.

With the settings described above we investigate EBDS performance compared to benchmark particle and Kalman filters. The metrics using a bootstrap particle filter with $10^5$ particles as reference are displayed in Figure 7.10. It can be seen that EBDS outperforms the extended and unscented Kalman filters for all metrics at almost all times. In the FME, EBDS has a lower error than the particle filter with $10^2$ particles but not the ones with more particles. In the KLD and $L^2L^\infty$-error performance is better and EBDS often has a higher accuracy than a particle filter with $10^3$ particles. The particle filter with $10^4$ particles is almost always better than EBDS however.

Marginal filtering densities from EBDS and the extended Kalman filter in one test instance is shown in Figure 7.11. Histograms from a particle filter with $10^5$ particles are also included for reference. In the presented example, EBDS seems to yield

densities that align well with the particle filter histograms. One notable exception is the left tail for the third component at $t = 0.3$, which EBDS estimates as too heavy. While not shown here, overestimation of tail probabilities, particularly in the left tail, for EBDS was observed recurringly for all components.

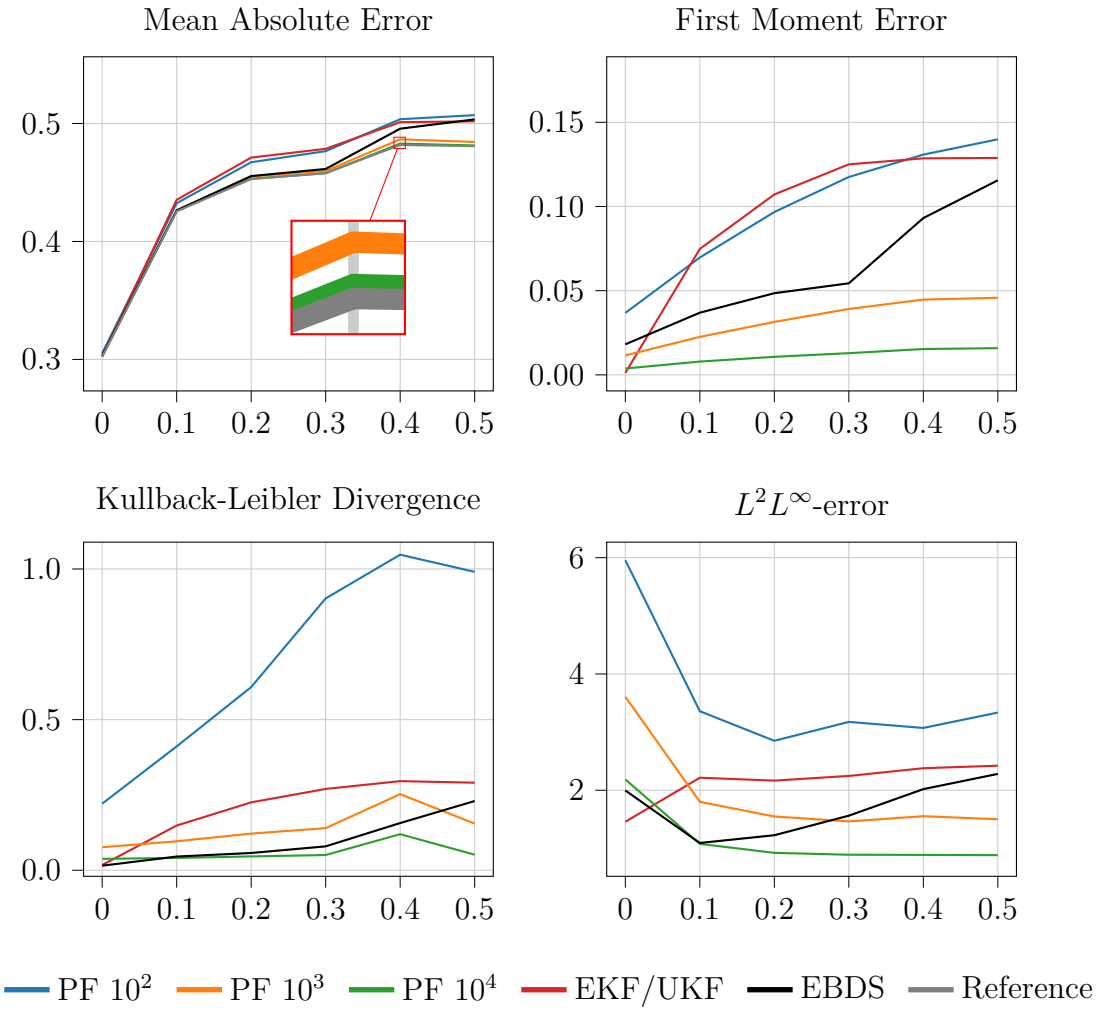


**Figure 7.10:** Metrics over time in the predator-prey example. Again, note that in this and the next example we consider the final time $T = 0.5$ and six observations. The extended and unscented Kalman filters perform identically in this example and hence they are shown together. To summarise, when analysing the metrics it can be seen that EBDS outperforms the Kalman filters, but not particle filters with many particles.

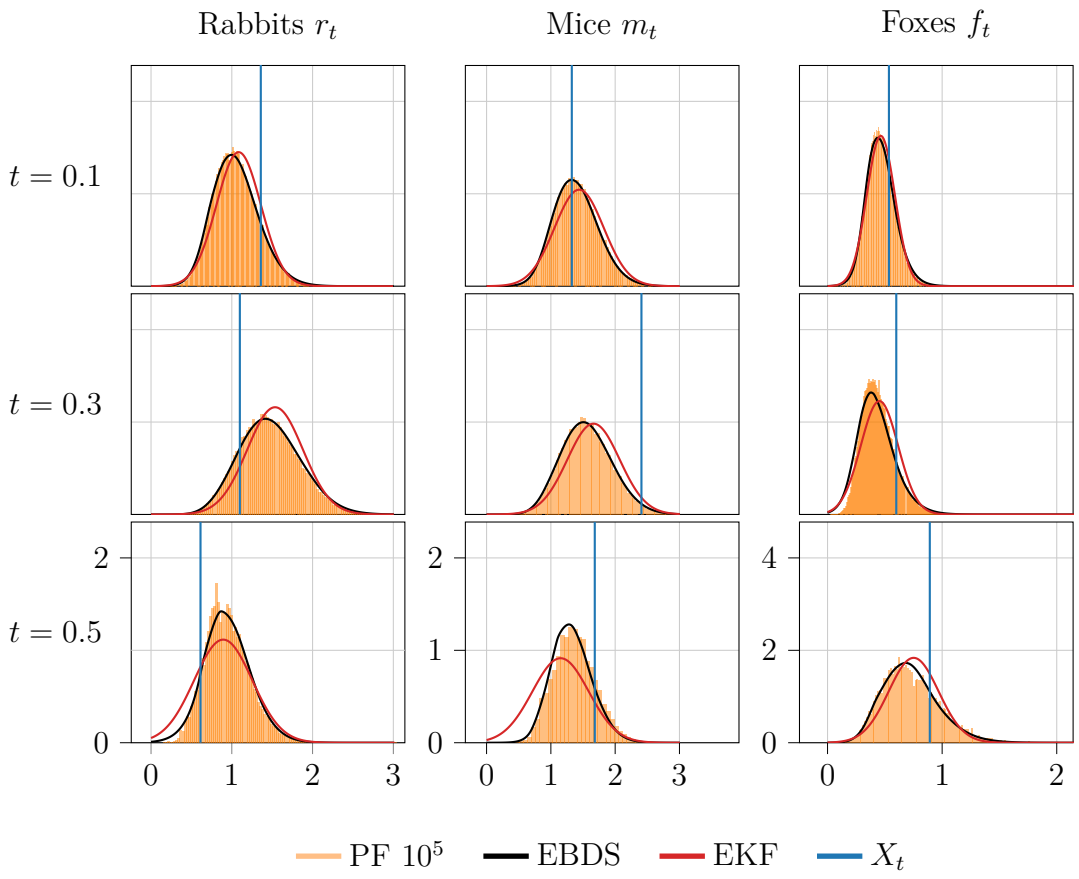


**Figure 7.11:** Marginal filtering densities in one instance for the stochastic predator-prey example. The densities from EBDS are shown in black. Densities from an extended Kalman filter are included for comparison and shown in red. As can be seen, the densities from EBDS align well with the particle filter histograms while those from the EKF deviate significantly in some cases.

## 7.4 Turbulent Spring-Mass System

As a last example, a spring-mass system affected by random turbulence is considered. We assume there are four masses, five springs and five dampers connected according to Figure 7.12. The state vector consists of the positions $X_t$ and velocities $V_t$ for these masses. To make the filtering problem more difficult we assume that positions and velocities can be observed for only two of the masses.

For constant matrices $A_{21}, A_{22} \in \mathbb{R}^{4\times4}$ and $t \in [0, 0.5]$ the state model in this example is thus

$$X_t = X_0 + \int_0^t V_s \,\mathrm{d}s + \int_0^t \sigma_1 \,\mathrm{d}W_s^{(1)},$$
$$V_t = V_0 + \int_0^t (A_{21}\, X_s + A_{22}\, V_s) \,\mathrm{d}s + \int_0^t \sigma_2 \,\mathrm{d}W_s^{(2)}.$$

Here, $W^{(1)}$ and $W^{(2)}$ are independent four-dimensional Brownian motions. As can be seen, this model is linear and the Kalman filter can be used as as reference. Let $k_1, k_2, k_3, k_4, k_5$ be the spring constants, $c_1, c_2, c_3, c_4, c_5$ the damping coefficients and $m_1, m_2, m_3, m_4$ the masses. The matrices $A_{21}$ and $A_{22}$ then have the form

$$A_{21} = \begin{bmatrix} -\frac{k_1+k_2}{m_1} & \frac{k_2}{m_1} & 0 & 0 \\ \frac{k_2}{m_2} & -\frac{k_2+k_3}{m_2} & \frac{k_3}{m_2} & 0 \\ 0 & \frac{k_3}{m_3} & -\frac{k_3+k_4}{m_3} & \frac{k_4}{m_3} \\ 0 & 0 & \frac{k_4}{m_4} & -\frac{k_4+k_5}{m_4} \end{bmatrix},$$

$$A_{22} = \begin{bmatrix} -\frac{c_1+c_2}{m_1} & 0 & 0 & 0 \\ 0 & -\frac{c_2+c_3}{m_2} & 0 & 0 \\ 0 & 0 & -\frac{c_3+c_4}{m_3} & 0 \\ 0 & 0 & 0 & -\frac{c_4+c_5}{m_4} \end{bmatrix}.$$

In the experiments, we use the parameter values $m = 1$, $k = 1$ and $c = 0.2$ for all masses, springs and dampers. Moreover, we let $\sigma_1 = \sigma_2 = 1$. These are slightly different parameter values compared to [2], where the same experiment is attempted but with ten masses. We assume that the observations are made on the velocities and positions of the first two masses and let the measurement noise variance be 1. Lastly, the starting value for each component is independent and standard normally distributed.

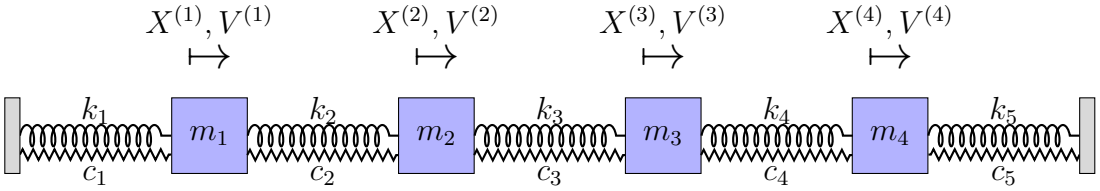


**Figure 7.12:** Schematic overview of the model for the last example, consisting of four masses, five springs and five dampers. The state in this example is the concatenation of all positions $X$ and velocities $V$, shown in the figure.

Again, we do not investigate convergence and instead benchmark EBDS directly using $N = 16$ steps. The resulting MAE, FME, KLD and $L^2L^2$-error over time are shown in Figure 7.13. The $L^2L^2$-error is shown in this case and not the $L^2L^\infty$-error because that metric proved to be more prone to outliers, particularly for the bootstrap particle filters with a small number of particles. To analyse the results, EBDS performs better than a particle filter with $10^3$ particles but worse than one with $10^4$ particles in the metrics that concern the first moment, i.e., the MAE and FME. For the KLD and $L^2L^2$-error EBDS is almost always better than a particle filter with $10^4$ particles and sometimes better than one with $10^5$. To examine how well EBDS scales with the state dimension it is particularly interesting to compare this linear example in eight dimensions to the one-dimensional linear Ornstein–Uhlenbeck example. When comparing the MAE to the one in Figure 7.3 it can clearly be seen that all filters deviate significantly more from the reference in this high-dimensional example. In particular, the particle filters with low numbers of particles have severely diminishes accuracy. Moreover, it can be seen that EBDS also performs worse in the

high dimension, but remains quite accurate and comparable to the particle filters with many particles.

Some marginal filtering densities for EBDS in one instance are shown in Figure 7.14. We also show densities from a Kalman filter and histograms from a particle filter. It can be seen in the figure that EBDS gives quite accurate results for the shown components, while the particle filter histograms deviate more significantly from the Kalman filter reference densities. This can be seen for $X_t^{(1)}$ at both times and $X_t^{(3)}$ as well as $V_t^{(3)}$ at $t = 0.4$. In particular, we see that EBDS manages to capture the tails of the reference densities rather well in this example.

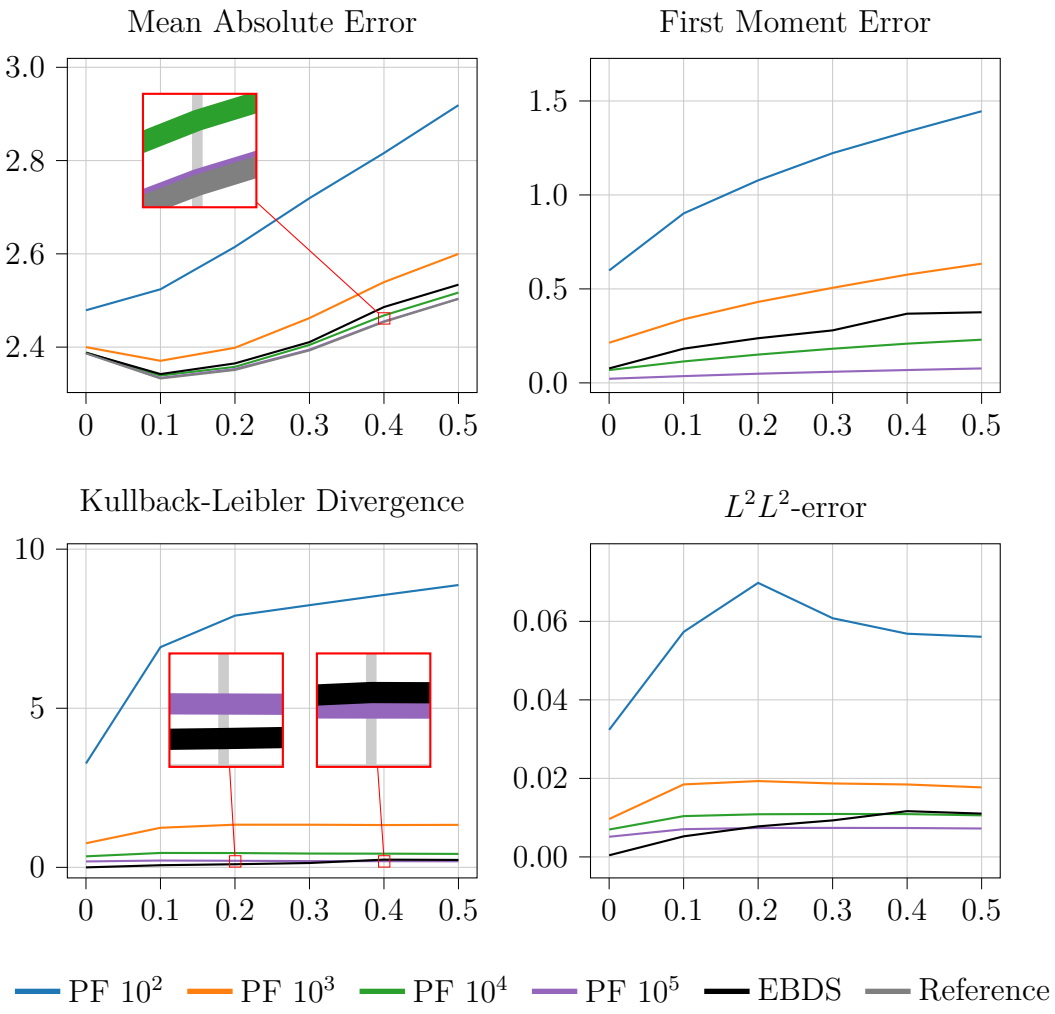


**Figure 7.13:** EBDS performance over time compared to benchmark filters in the spring-mass example. In terms of the first moment, EBDS generally performs slightly worse than a particle filter with $10^4$ particles while it performs slightly better than this filter in terms of the KLD and $L^2L^2$-error.

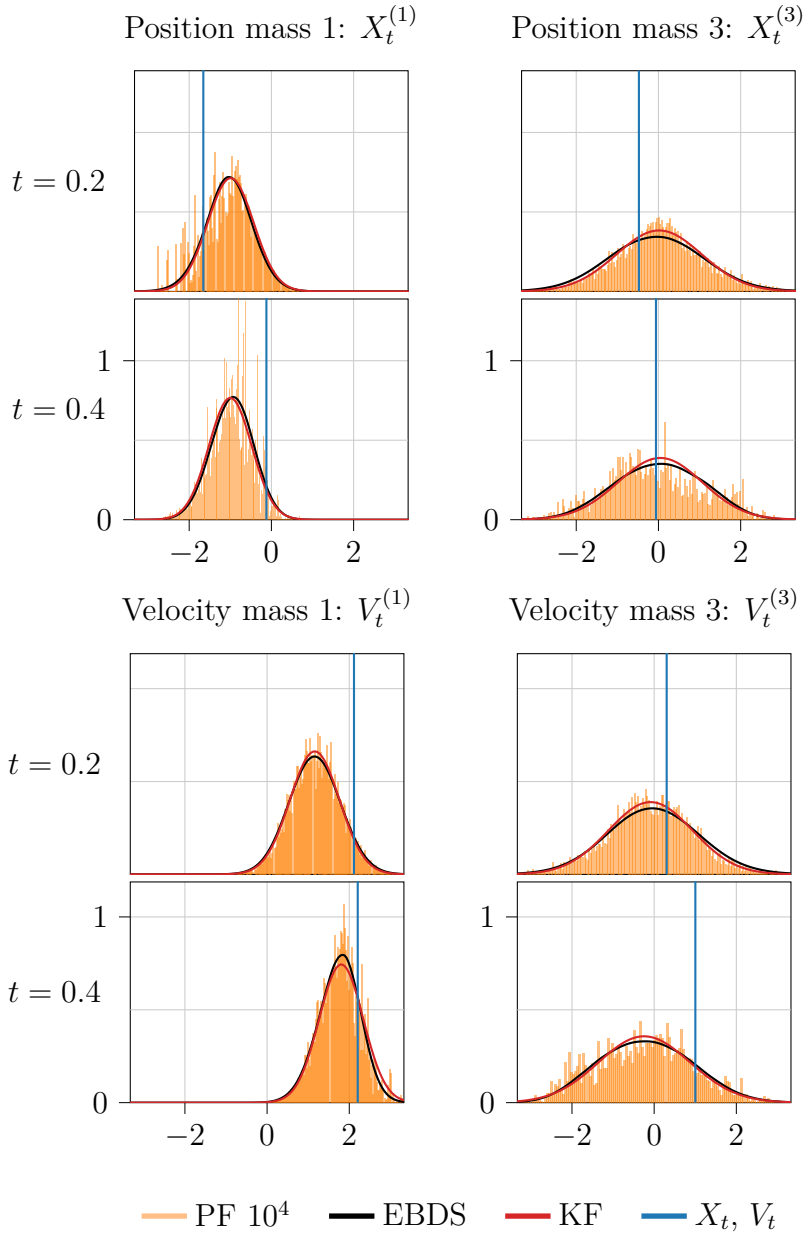


**Figure 7.14:** Some marginal densities for one instance of the spring-mass problem. The position and velocity in the left column can be observed whereas the position and velocity in the right column cannot. As can be seen, the EBDS densities align quite well in most times with those from the Kalman filter. The particle filter, on the other hand, seems less accurate.

To summarise the examples in more than one dimension, EBDS gives decent results although the problems seem more difficult than in one dimension. This is especially the case for the nonlinear predator-prey problem and to an extent the nonlinear two-dimensional problem treated in Appendix A. However, for the linear eight-dimensional example performance is quite satisfactory compared to the benchmarks and this indicates that EBDS works in high dimensions. Analysis of EBDS marginal densities in the three higher-dimensional examples indicates that

tail-behaviour can be difficult to learn. The linear example does not seem to suffer from this problem, see for instance Figure 7.14, but the nonlinear problems do, see for example Figure 7.11 and A.5. As such handling the tails better seems to be a clear next step. This is discussed further in Chapter 8. Nevertheless, in the nonlinear examples our filter performs better than the Kalman filters and it must still be noted that EBDS is very computationally light online. As such, the method as it is still shows some promise. In the following sections some complementary results are provided, starting with an exploration of the neural network architecture.

## 7.5 Reduction of Error Using LSTM-Model

To investigate whether the LSTM hybrid model can reduce the neural network approximation error and thereby contribute to a more accurate filter, its performance is here compared to the standard architecture in the bimodal example. We compare the two architectures using $N = 16$ and choose the LSTM hyperparameters such that the number of trainable parameters are the same.

Comparisons of the FME, KLD, $L^2L^2$-error and $L^2L^\infty$-error over time are shown in Figure 7.15. Note that for this figure all models were trained with $10^6$ sample paths. The figures in Section 7.2 were generated with $10^7$ training samples, which is why the numbers for the standard architecture are different in this case. Apart from at early times, the LSTM-based architecture performs significantly better on average. The exception is for the KLD at $t = 1$, where one of the LSTM instances has a particularly high error. The difference in performance is most noticeable in the two $L^2$-errors and for the KLD at $t = 0.2$ to $t = 0.8$, while the two architectures are more similar in the FME. It can be seen that for all errors the best instance is almost always LSTM-based at later times. As such, a natural conclusion is that this architecture is to prefer in the bimodal example if the error is to be minimised. Note though that this architecture is both more computationally demanding to train and to evaluate.

Additional preliminary experiments were performed to compare the two architectures in the other examples and with more training samples. Because of time constraints these results could not be included in this thesis. However, generally the experiments indicated favourable results for the LSTM-model. In the future, a more comprehensive study of the LSTM hybrid model and other neural architectures can be conducted.

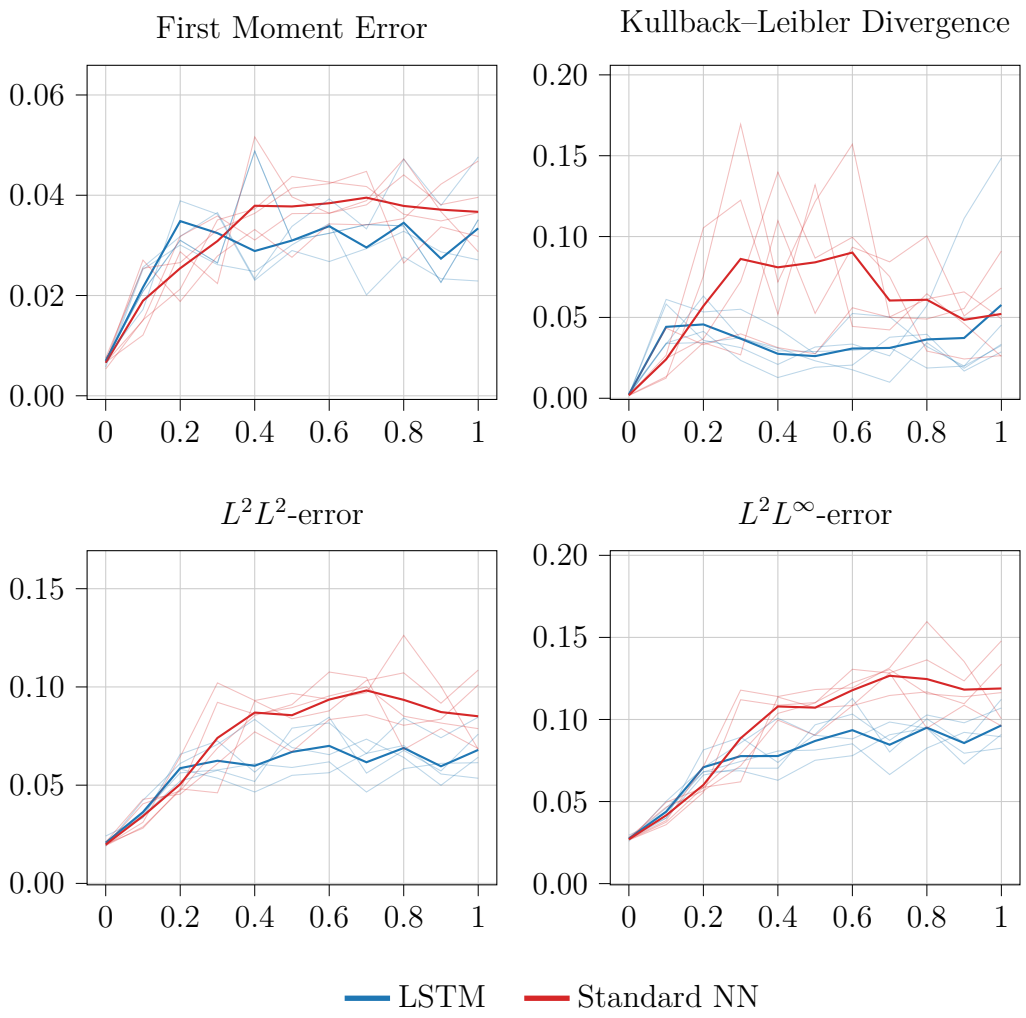


**Figure 7.15:** Metrics over time in the bimodal example for the standard architecture compared to the LSTM-based architecture. The thin lines are model instances (five for each architecture) whereas the thick lines are averages over the instances. On average, the LSTM-based models perform better for almost all time steps and all metrics.

## 7.6 Runtime Analysis

In this section some runtime results are provided. The purpose is to show that EBDS is efficient to use in real time for estimation and that the filter scales well with the state dimension in terms of computation times. For comparison, the filtering times for particle filters and Kalman filters in the relevant examples are also provided.

It must clearly be noted however that performance depends heavily on implementation. While a fairly large amount of optimisation has been performed for all filters, additional work could be done in this regard. A reader interested in more details regarding implementation is referred to the project GitHub repository [34]. Also note that all computations were performed on a compute cluster and that interference

from background processes was minimal, allowing for consistent runtime estimation. The hardware was an NVIDIA A40 GPU with 48 GB of VRAM, an Intel(R) Xeon(R) Gold 6338 CPU and 512 GB of RAM. The computational resources were provided by Chalmers e-Commons at Chalmers.

Tables 7.3–7.6 display the average filtering times and average time to compute first moments for the four example problems in this chapter. The filtering time is the time required to perform all computations for one entire sample path, i.e., all prediction and update steps. The First Moment (FM) calculation time, similarly, is the time required to compute first moments for the filtering distributions in all observation times for one sample path. The total time is thus the time required to estimate all filtering means for one sample path. Note that EBDS does not have a filtering step, as it is pretrained, and the benchmark filters offer almost instantaneous computation of the first moment. Also remark that the first moment calculation time for EBDS depends on the number of samples or points used to estimate the integral for the mean. In the Ornstein–Uhlenbeck, bimodal and spring-mass examples 2000 points were used while the nonlinear predator-prey example required 20000 samples.

**Table 7.3:** Average time for filtering and calculating first moments in the Ornstein–Uhlenbeck case. The reference filter is marked in bold and has a very large number of prediction steps. Hence its runtime should not be compared directly to the other filters.

| **Filter** | **FM calc. time** [s] | **Filter time** [s] | **Total time** [s] |
|---|---|---|---|
| **KF** | 0.0001 | 0.0514 | 0.0515 |
| PF $10^2$ | 0.0001 | 0.0172 | 0.0174 |
| PF $10^3$ | 0.0003 | 0.0338 | 0.0341 |
| PF $10^4$ | 0.0018 | 0.2122 | 0.2139 |
| PF $10^5$ | 0.0158 | 2.1682 | 2.1840 |
| EBDS | 0.0113 | - | 0.0113 |

**Table 7.4:** Average time for filtering and calculating first moments in the bimodal case. Again, the reference filter is in bold.

| **Filter** | **FM calc. time** [s] | **Filter time** [s] | **Total time** [s] |
|---|---|---|---|
| **PF $10^5$** | 0.0186 | 16.1989 | 16.2175 |
| PF $10^2$ | 0.0002 | 0.0181 | 0.0184 |
| PF $10^3$ | 0.0007 | 0.0515 | 0.0522 |
| PF $10^4$ | 0.0034 | 0.4066 | 0.4100 |
| EKF | 0.0001 | 0.0109 | 0.0110 |
| UKF | 0.0001 | 0.0666 | 0.0667 |
| EBDS | 0.0122 | - | 0.0122 |

**Table 7.5:** Average time for filtering and calculating first moments in the three-dimensional predator-prey case. As a reminder, we only consider six observations for this example and the spring-mass example in Table 7.6, hence the times in these tables are for fewer estimations than the times in Table 7.3 and 7.4.

| **Filter** | **FM calc. time** [s] | **Filter time** [s] | **Total time** [s] |
|---|---|---|---|
| **PF $10^5$** | 0.0373 | 15.3075 | 15.3448 |
| PF $10^2$ | 0.0005 | 0.0080 | 0.0085 |
| PF $10^3$ | 0.0007 | 0.0230 | 0.0237 |
| PF $10^4$ | 0.0067 | 0.1891 | 0.1958 |
| EKF | 0.0001 | 0.0043 | 0.0044 |
| UKF | 0.0001 | 0.0346 | 0.0347 |
| EBDS | 0.0301 | - | 0.0301 |

**Table 7.6:** Average time for filtering and calculating first moments in the eight-dimensional spring-mass case.

| **Filter** | **FM calc. time** [s] | **Filter time** [s] | **Total time** [s] |
|---|---|---|---|
| **KF** | 0.0001 | 0.0291 | 0.0292 |
| PF $10^2$ | 0.0003 | 0.0080 | 0.0083 |
| PF $10^3$ | 0.0011 | 0.0443 | 0.0454 |
| PF $10^4$ | 0.0085 | 0.4324 | 0.4409 |
| PF $10^5$ | 0.0828 | 5.7126 | 5.7955 |
| EBDS | 0.0751 | - | 0.0751 |

It can be seen from the tables that EBDS is very fast considering its accuracy. In the Ornstein–Uhlenbeck example shown in Table 7.3 our method is 20 times faster than a particle filter with $10^4$ particles, which has comparable performance in terms of the FME. In the bimodal example EBDS is five times faster than a particle filter with comparable performance and even as fast as the extended Kalman filter, see Table 7.4. As can be seen in Figure 7.10, EBDS performance in terms of FME in the predator-prey example is quite bad, hence comparable particle filters are faster in this case, see Table 7.5. Yet, EBDS is still faster than particle filters with many particles as well as the unscented Kalman filter. For the eight-dimensional example shown in Table 7.6 EBDS has slightly higher FME than a particle filter that is six times slower with $10^4$ particles. Lastly, note that the first moment calculation time for EBDS seems to scale well with the state dimension.

The analysis in this section only covers the computation of the first moment, which is a typical use-case for filtering algorithms since the mean is a natural choice for a state point-estimate. If the task instead is to calculate some value of the filtering density, then the results would be even more favourable for EBDS compared to the particle filters. This is especially true using the method in this thesis, since the Gaussian kernel density estimate is very computationally demanding.

## 7.7 Alternatives to Training Normalisation

Table 7.7 shows the average proportion of time for key tasks during training in the four examples. As can clearly be seen, neural network training and calculation of normalisation constants in update steps, i.e., calculation of the integral in (6.3), take by far the most time. The proportion between these vary, depending on the dimensionality of the problem and the number of prediction steps $N$, however it is clear that avoiding the update step normalisation would save considerable time.

**Table 7.7:** Fraction of time for main training tasks in all examples. We use the abbreviation OU for the Ornstein–Uhlenbeck example, BM for the bimodal example, LV for the stochastic Lotka–Volterra problem and SM for the spring-mass problem.

| **Task** | **OU** | **BM** | **LV** | **SM** |
|---|---|---|---|---|
| Data simulation | 0.09% | 0.14% | 0.17% | 0.59% |
| Label calculation | 0.05% | 0.04% | 0.02% | 0.05% |
| Network training | 67.67% | 70.34% | 32.70% | 36.59% |
| Update normalisation | 32.19% | 29.48% | 67.11% | 62.77% |

In Chapter 5 it was argued that this normalisation is necessary, which is shown here. We investigate four different normalisation schemes:

1. Not including the normalisation constant in the update step nor dividing with the average label size in (6.5).

2. Not including the normalisation constant in the update step but dividing with the average label size in each step according to (6.5) during training.

3. Calculating the integral (6.3) in the update step for a subset of training samples (one tenth of all samples) and dividing by the average integral while not performing (6.5).

4. Calculating the integral (6.3) for each training sample while also performing (6.5).

These schemes are sorted in order from least computationally expensive to most. Note that the last alternative was used to generate all other results.

For $N = 32$ and the Ornstein–Uhlenbeck example the $L^2L^2$- and $L^2L^\infty$-error in these four cases are displayed in Figure 7.16. As can be seen, without any normalisation the error increases drastically after some time. Using some normalisation makes the curve smoother, but scheme 4 yields significantly lower error than number 2 or 3. While not shown here, the pattern is similar for other examples. As such, although it is time consuming, normalising exactly seems to be the best approach. Note that fewer training samples were used here compared to Figure 7.1, which is why the performance for scheme 4 is worse in this plot. The reason behind why some

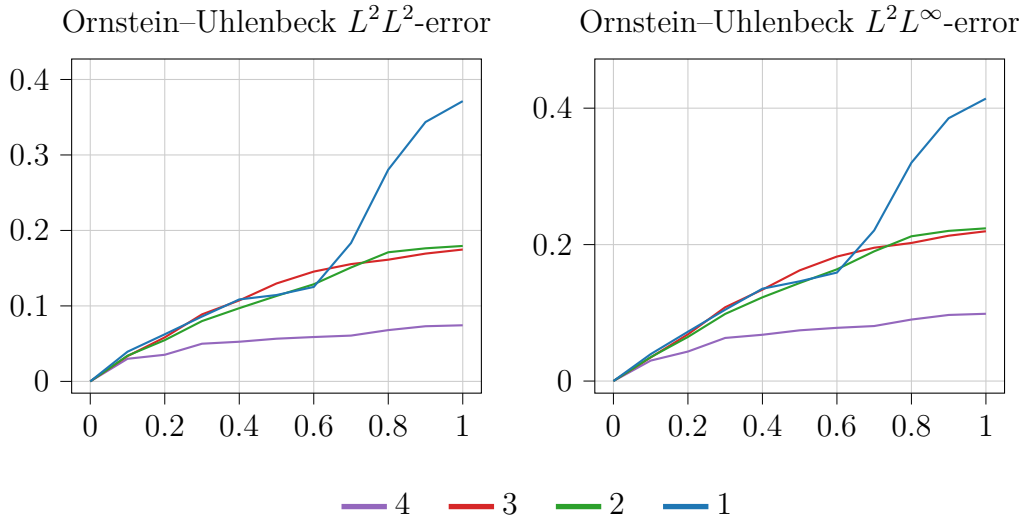


**Figure 7.16:** $L^2L^2$- and $L^2L^\infty$-error over time when using different normalisation schemes in the training process. As can be seen, scheme 4 yields significantly better results than the others.

normalisation is needed can be seen in Figure 7.17. Using the four normalisation schemes, this figure shows the average over the test samples of the integral

$$\int_{\mathbb{R}} \bar{\pi}_{k,n}(x, y_{0:k}^m)\,\mathrm{d}x,$$

for all networks $(\bar{\pi}_{k,n})_{k=0,n=1}^{K,N}$. Without any normalisation the integral decreases significantly after each update step. Eventually, this leads to the training labels becoming too small for efficient neural network training. It can be seen in the figure that all of schemes 2, 3 and 4 mitigate this effect. Moreover, scheme 4 and to some extent scheme 3 ensure the prediction densities are approximately normalised. Why scheme 4 is significantly better than 2 or 3 is less clear. One theory is that the estimate of an average normalisation constant in both methods will be dominated by a small number of labels that are particularly large. This could result in inaccurate scaling for small labels.

Irrespective of the underlying reason, this experiment indicates that sample-wise normalisation should be used if accuracy is the main goal. However, in scenarios where a shorter training time is preferable schemes 2 or 3 might be better. Note again though that by using scheme 4 the prediction densities become approximately normalised, which is useful if these need to be evaluated rapidly. Moreover, the normalising constants from the training could possibly be used downstream during the evaluation of filtering densities online. Although not attempted here, the constants during training could be saved together with corresponding observation chains. Then, during evaluation, similarity search can be performed to determine which observation chain in training data is the closest and the constant for that chain can be used to approximate the true normaliser. This approach would allow for almost instantaneous evaluation of filtering densities and constitutes a possible direction for future research.

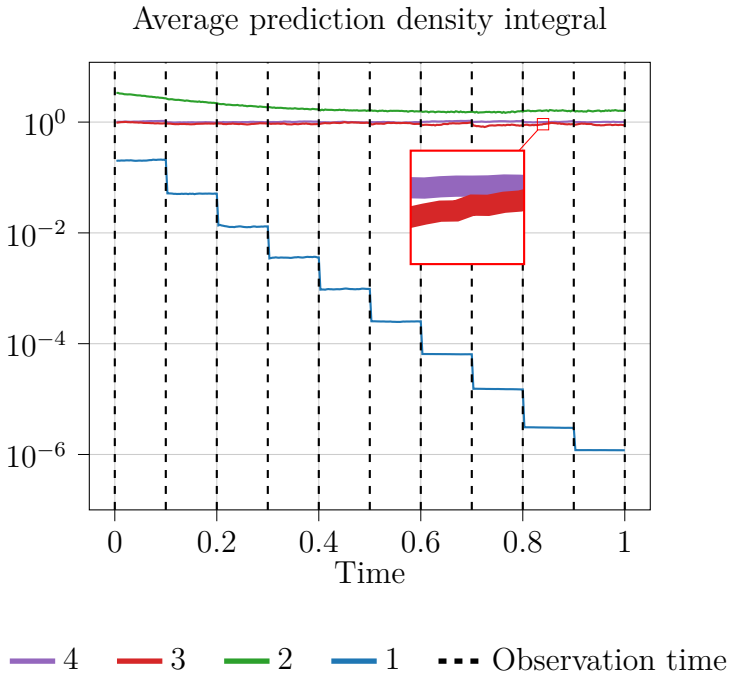


**Figure 7.17:** Average integrals of the prediction densities over all test samples. Apparently, using scheme 1 results in the integral becoming several orders of magnitude smaller over time. Utilisation of any other scheme prevents this from occuring.

# 8

# Discussion

The main task undertaken in this thesis is an initial numerical evaluation of the energy-based deep splitting method for discrete observations. The results in one dimension show that the method is promising for accurate filtering since EBDS consistently performs as well as many of the benchmark filters. Additionally, the results for the one-dimensional examples indicate strong convergence of order 1/2 in the number of intermediate prediction steps. Although performance is worse for the nonlinear problems in more dimensions, EBDS remains on par with some benchmarks. Moreover, the linear eight-dimensional problem shows that EBDS can work rather well in high dimensions. As the method is computationally light online even in the higher-dimensional problems considered, EBDS seems promising for further evaluation. Chapter 5, Chapter 6 and Appendix B are purposefully detailed, so that the results can be reproduced and the method can be improved in future works. As mentioned, the source code can also be found in the project GitHub repository [34]. There are many potential directions for future research.

Firstly, Section 7.7 in the results clearly highlights the possibility of reducing training time by either avoiding or improving the normalisation in the update steps. Here, the only methods attempted were exact normalisation using numerical integration, dividing by the mean label magnitude and dividing by the average integral of a subset of training samples. One way to potentially reduce the time for this task would be to only normalise in some update steps, although the performance effects of this would have to be examined.

Overall however, a large improvement to the training process that could yield superb performance even in high dimensions would be to device some scheme for improved sampling of training data. At the moment, the sampling is tied to the dynamics of the state since the auxiliary process $Z$ follows the same SDE. This means that the neural networks will be accurate on samples typically seen when simulating the dynamics, but that tail behaviour might not be well-explored as neural network generally do not extrapolate. Adding extra terms to the neural network output is a potential solution which does seem to yield positive results, although it also seems to complicate the training process and sometimes leads to numerical problems, which is why it was not used extensively here. A simple solution would be to use stratified sampling from the prior, although even better results could probably be achieved by some type of importance sampling strategy in each training step.

It would also be interesting to examine other methods for simulating the under-

lying stochastic differential equations. The Euler–Maruyama method has worked well in the examples tested here, but implicit schemes might be better in certain scenarios. Moreover, a numerical study regarding the convergence when using the Milstein scheme is possible.

Section 7.5 shows that state-of-the-art architectures can be used to improve the EBDS filtering method. In the future, it is possible to explore this direction further by utilising convolutional neural networks or even transformer-based models. It would also be interesting to investigate transfer learning, to see whether EBDS trained on some problem and fine-tuned on some other problem (perhaps with the same dynamics but different parameters) has a competitive accuracy. From an implementation perspective, the approach of pre-simulating all training samples before the neural network training works for small training sets. However, simulation during training is also an option. Apart from enabling truly large training sets, it could be interesting to see whether iteratively simulating more data until each network converges yields better results.

In addition to improving the method, it could also be applied to other problems. In this thesis filtering is the focus but, as mentioned in the introduction, similar problems concern smoothing and parameter estimation. Adapting EBDS for parameter estimation in particular is another interesting direction for future research.

Finally, as treated in the introduction, filtering has a vast number of applications in various fields. EBDS is especially suitable for problems with dynamics that are known beforehand and for which online performance is critical. Focusing more on such problems, for example in target tracking or biomedical engineering, is highly interesting for the future of the method. Then, EBDS can be tailored to the specific setting and its performance can be evaluated in truth on non-simulated data.

# Bibliography


[1] K. Andersson, A. Andersson, and C. W. Oosterlee, "Convergence of a robust deep FBSDE method for stochastic control," *SIAM Journal on Scientific Computing*, 2023.

[2] K. Bågmark, A. Andersson, and S. Larsson, "An energy-based deep splitting method for the nonlinear filtering problem," *Partial Differential Equations and Applications*, 2023.

[3] R. Bansal *et al.*, "Stochastic filtering based transmissibility estimation of novel coronavirus," *Digital Signal Processing*, 2021.

[4] C. Beck *et al.*, "Deep learning based numerical approximation algorithms for stochastic partial differential equations and high-dimensional nonlinear filtering problems," 2020. arXiv: 2012.01194.

[5] C. Beck *et al.*, "Deep splitting method for parabolic PDEs," *SIAM Journal on Scientific Computing*, 2021.

[6] M. M. Bejani and M. Ghatee, "A systematic review on overfitting control in shallow and deep neural networks," *Artificial Intelligence Review*, 2021.

[7] A. Boopathy *et al.*, "Resampling-free particle filters in high-dimensions," 2024. arXiv: 2404.13698.

[8] M. Buehner, R. McTaggart-Cowan, and S. Heilliette, "An ensemble Kalman filter for numerical weather prediction based on variational data assimilation: VarEnKF," *Monthly Weather Review*, 2017.

[9] F. Cassola and M. Burlando, "Wind speed and wind energy forecast through Kalman filtering of numerical weather prediction model output," *Applied Energy*, 2012.

[10] G. Chen, "A gentle tutorial of recurrent neural network with error backpropagation," 2018. arXiv: 1610.02583.

[11] Z. Chen, "Bayesian filtering: From Kalman filters to particle filters, and beyond," *Statistics*, 2003.

[12] D. Crisan, "The stochastic filtering problem: A brief historical account," *Journal of Applied Probability*, 2014.

[13] D. Crisan, A. Lobbe, and S. Ortiz-Latorre, "An application of the splitting-up method for the computation of a neural network representation for the solution for the filtering equations," *Stochastics and Partial Differential Equations: Analysis and Computations*, 2022.

[14] N. Cui, L. Hong, and J. R. Layne, "A comparison of nonlinear filtering approaches with an application to ground target tracking," *Signal Processing*, 2005.

[15] P. Date and K. Ponomareva, "Linear and non-linear filtering in mathematical finance: a review," *IMA Journal of Management Mathematics*, 2010.

[16] D. De Marinis and D. Obrist, "Data assimilation by stochastic ensemble Kalman filtering to enhance turbulent cardiovascular flow data from under-resolved observations," *Frontiers in Cardiovascular Medicine*, 2021.

[17] X. Deng *et al.*, "PoseRBPF: A Rao–Blackwellized particle filter for 6-D object pose tracking," *IEEE Transactions on Robotics*, 2021.

[18] G. Galanis *et al.*, "Applications of Kalman filter based on non-linear functions to numerical weather predictions," *Annales Geophysicae*, 2006.

[19] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proceedings F (Radar and Signal Processing)*, 1993.

[20] Q. F. Gronau *et al.*, "A tutorial on bridge sampling," *Journal of Mathematical Psychology*, 2017.

[21] D. Jacoby, J. Ostrometzky, and H. Messer, "Short-term prediction of the attenuation in a commercial microwave link using LSTM-based RNN," in *2020 28th European Signal Processing Conference*, 2021.

[22] S. J. Julier and J. K. Uhlmann, "A general method for approximating nonlinear transformations of probability distributions," 1996.

[23] S. J. Julier and J. K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," in *Signal Processing, Sensor Fusion, and Target Recognition VI*, 1997.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. arXiv: `1412.6980`.

[25] F. C. Klebaner, "Introduction to Stochastic Calculus with Applications," 3rd. Imperial College Press, 2012.

[26] P. E. Kloeden and E. Platen, "Numerical Solution of Stochastic Differential Equations." Springer, 1992.

[27] T. Li, M. Bolic, and P. M. Djuric, "Resampling methods for particle filtering: Classification, implementation, and strategies," *IEEE Signal Processing Magazine*, 2015.

[28] B. Liu *et al.*, "Stochastic filtering approach for condition-based maintenance considering sensor degradation," *IEEE Transactions on Automation Science and Engineering*, 2020.

[29] P. Malhotra *et al.*, "LSTM-based encoder-decoder for multi-sensor anomaly detection," 2016. arXiv: `1607.00148`.

[30] E. Myötyri, U. Pulkkinen, and K. Simola, "Application of stochastic filtering for lifetime prediction," *Reliability Engineering  System Safety*, 2006.

[31] G. A. Pavliotis, "Stochastic Processes and Applications," 1st. Springer, 2014.

[32] J. Quinn, "A high-dimensional particle filter algorithm," 2019. arXiv: 1901.10543.

[33] D. Revuz and M. Yor, "Continuous Martingales and Brownian Motion," 3rd. Springer, 1999.

[34] F. Rydin, "Deep-Stochastic-Filtering," 2024. [Online]. Available: https://github.com/filiprydin/Deep-Stochastic-Filtering.

[35] S. Särkkä, "Bayesian Filtering and Smoothing." Cambridge University Press, 2013.

[36] J. F. Schmidt *et al.*, "Interference prediction in wireless networks: Stochastic geometry meets recursive filtering," *IEEE Transactions on Vehicular Technology*, 2021.

[37] R. I. Sokolov, D. A. Dolmatov, and R. R. Abdullin, "Efficient signal processing algorithms for radar and telecommunication systems," in *2016 International Conference on Advances in Computing, Communications and Informatics*, 2016.

[38] R. C. Staudemeyer and E. R. Morris, "Understanding LSTM – a tutorial into long short-term memory recurrent neural networks," 2019. arXiv: 1909.09586.

[39] J. H. Suddath, R. H. Kidd III, and A. G. Reinhold, "A linearized error analysis of onboard primary navigation systems for the Apollo lunar module," NASA, 1967.

[40] K. Sunghan, L. Holmstrom, and J. McNames, "Tracking of rhythmical biomedical signals using the maximum a posteriori adaptive marginalized particle filter," *British Journal of Health Informatics and Monitoring*, 2015.

[41] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, 2014.

[42] S. Thrun, "Particle filters in robotics," in *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, 2002.

[43] F. Vadillo, "Comparing stochastic Lotka–Volterra predator-prey models," *Applied Mathematics and Computation*, 2019.

[44] A. Valyrakis *et al.*, "Stochastic modeling and particle filtering algorithms for tracking a frequency-hopped signal," *IEEE Transactions on Signal Processing*, 2009.

[45] E. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*, 2000.

[46] K. Xu *et al.*, "Stochastic modeling based nonlinear Bayesian filtering for photoplethysmography denoising in wearable devices," *IEEE Transactions on Industrial Informatics*, 2020.

[47] Y. Zeng and S. Wu, "State-Space Models." Springer, 2013.

# A

# Additional Examples

In this appendix EBDS performance is presented for two additional example problems as a complement to the results in Chapter 7. Both examples are nonlinear and the first is in one dimension whereas the second is in two dimensions. In each section, the relevant example is first introduced in detail after which the performance of EBDS is shown. For both examples we consider 11 observations and these occur with 0.1 units of time in between starting at $t_0 = 0$. The simulations are ended at $T = 1$. Hyperparameters are listed in Appendix B and as in Chapter 7, we always average the metrics over five EBDS instances.

## A.1 Geometric Brownian Motion

This example in one dimension is quite different to the ones presented in Chapter 7. Now, we assume the state satisfies the SDE

$$X_t = X_0 + \int_0^t \mu\, X_s \,\mathrm{d}s + \int_0^t \sigma\, X_s \,\mathrm{d}W_s, \quad t \in [0, 1].$$

The solution to this equation is the famous geometric Brownian motion. This process is very important in financial mathematics as it is used to model the underlying security in the famous Black–Scholes model for option pricing. It can be seen from the equation that the process noise increases with larger $x$ and that the state tends to increase exponentially over time. In fact, when started in a deterministic $X_0$, the expected value at a time $t$ is $\mathbb{E}[X_t] = X_0 \exp(\mu t)$.

The tendency for the state to drift away as well as the non-constant diffusion is why this example is used here. It can be expected that sample paths differ widely, in contrast to the other two one-dimensional examples where the state tends to some value over time. This should make the filtering problem more difficult, especially for EBDS as outliers should occur more frequently with no direct correspondence in training data. We utilise the parameter values $\mu = \sigma = 1$ as well as a linear measurement function and measurement noise variance 1. In this example, the half-normal distribution is used as prior, with original standard deviation 1. The Kalman filters cannot be expected to perform well in this scenario and a particle filter with $10^5$ particles is thus used as reference. Finally, similarly to the Lotka–Volterra example we enhance the output layer of the neural networks so that if the state is negative the output is set to zero.

Here, we do not investigate convergence. Instead, EBDS performance with $N = 16$ is benchmarked directly against the extended and unscented Kalman filters as well as bootstrap particle filters. The resulting MAE, FME and $L^2L^2$-error can be seen in Figure A.1. The KLD and $L^2L^\infty$-error are not shown in this case. This is because the reference particle filter together with the Gaussian KDE yields inaccurate density estimates in this scenario, especially for negative states. Since we use the half-normal prior, meaning many state sample paths start close to $x = 0$ and then drift away, the problem is most noticeable at early times. The $L^2L^2$-error also suffers from the same problem, but the effect is less apparent, which is why this metric is shown here. It should still be interpreted with care however, particularly for early times. The reader should focus more on the MAE and FME. Then it can be observed that EBDS performs as well as a particle filter with 1000 particles and significantly better than both the extended and unscented Kalman filters. Evidently, EBDS works rather well in spite of the inherent difficulties with this problem.

Some example filtering densities compared to benchmarks are presented in Figure A.2. It can be seen that the EBDS densities align quite well with the histograms from the particle filter, although there are some deviations particularly in $t = 0.4$ and $t = 0.6$ for the peak. Note that the distribution changes significantly at $t = 0.8$ and $t = 1$, but EBDS seems to remain accurate.

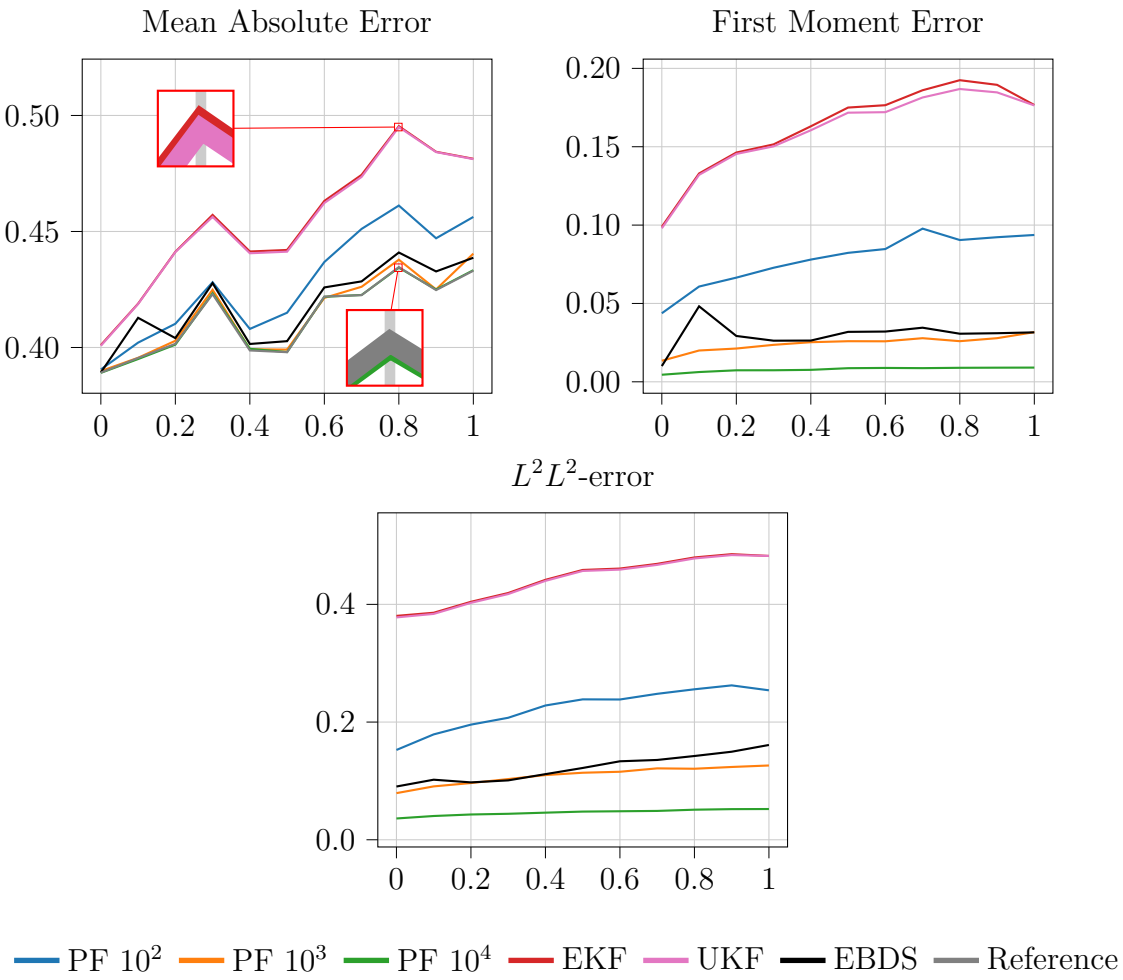


**Figure A.1:** Metrics over time in the geometric Brownian motion example. Note that the plot of the $L^2L^2$-error should be interpreted with care, especially at early times, as the reference filter density estimates can be considered somewhat inaccurate.

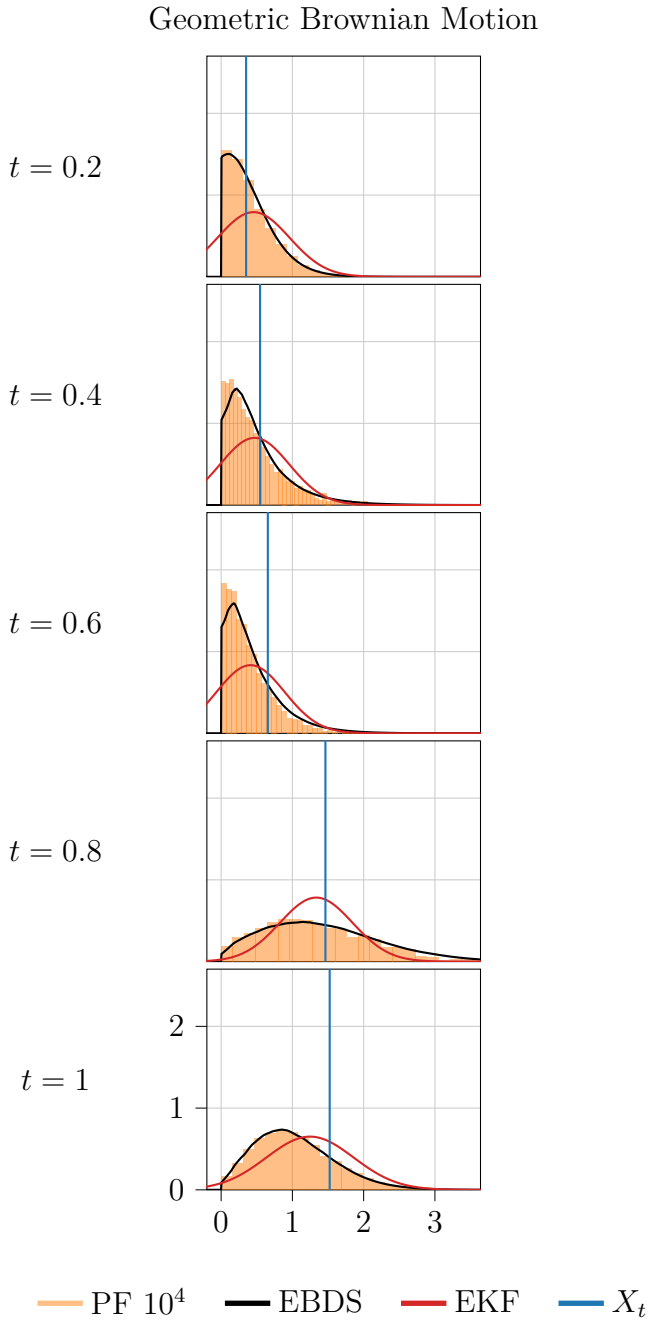


**Figure A.2:** Some filtering densities in one example for the geometric Brownian motion problem. As can be seen, the EBDS densities generally align with the particle filter histograms while those from the extended Kalman filter differ significantly.

## A.2 Sine Drift with Stochastic Diffusion

Stochastic volatility models are used extensively in finance, especially in the context of option pricing. Two particularly famous such models are the Heston model commonly used for stocks and the Longstaff—Schwartz model for the short interest rate. These models let the diffusion of the instrument price be a function of

a second stochastic process, which is, in general, unobservable. Motivated by this setup, we introduce the next example. It must be noted though that the drift and diffusion in this example are not commonly used in financial modelling, but do offer an interesting challenge for testing EBDS.

Consider the following model for the state $X_t = (r_t, \sigma_t)^T$ for $t \in [0, 1]$:

$$\begin{aligned} r_t &= r_0 + \int_0^t -\alpha_1 \sin(r_s)\,\mathrm{d}s + \int_0^t \beta_1 \sqrt{1+\sigma_s^2}\,\mathrm{d}W_s^{(1)}, \\ \sigma_t &= \sigma_0 + \int_0^t -\alpha_2\,\sigma_s^3\,\mathrm{d}s + \int_0^t \beta_2\,\mathrm{d}W_s^{(2)}. \end{aligned} \tag{A.1}$$

Then, assume that only the first component $r$ can be observed and let $h(X) = r$. For the parameters, we use $\alpha_1 = 2$, $\alpha_2 = 0.1$, $\beta_1 = 0.5$ and $\beta_2 = 1$. The variance of $V$ is again set to 1 and the prior is standard normal for both components of the state.

To analyse the dynamics at hand, the drift of $\sigma_t$ ensures that it is mean reverting. The first order Taylor expansion of the drift for the first component tells us that the same is true for $r$ close to zero. However, more generally, the first component drifts towards the closest multiple of $2\pi$. Some sample paths are shown in Figure A.3. It can be seen that $\sigma_t$ is only weakly mean-reverting, which is caused by $\alpha_2$ being rather small, and that sample paths with larger value in the second component exhibit a greater degree of random behaviour.

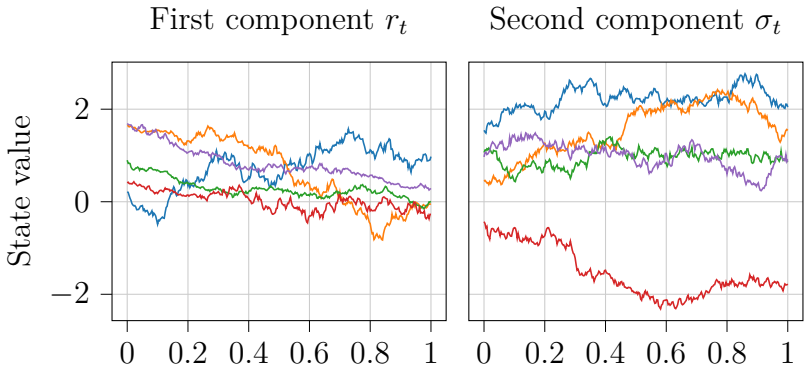


**Figure A.3:** Some simulated sample paths from (A.1). Equal colour in the two components indicate corresponding paths.

As in the last example, we only investigate the performance for EBDS using $N = 16$. For evaluating the metrics, the reference filter in this case is a bootstrap particle filter with $10^5$ particles. Using these settings, the MAE, FME, KLD and $L^2L^\infty$-error on test data is shown in Figure A.4 for EBDS and the benchmark filters. It can be noted that the Kalman filters perform well in terms of the mean, but lose accuracy severely over time for the KLD and $L^2L^\infty$-error. EBDS, on the other hand, has approximately the same error as the particle filter with $10^3$ particles for the first moment (slightly better at early times and slightly worse at late times) but clearly performs better than this filter terms of the KLD and $L^2L^\infty$-error.

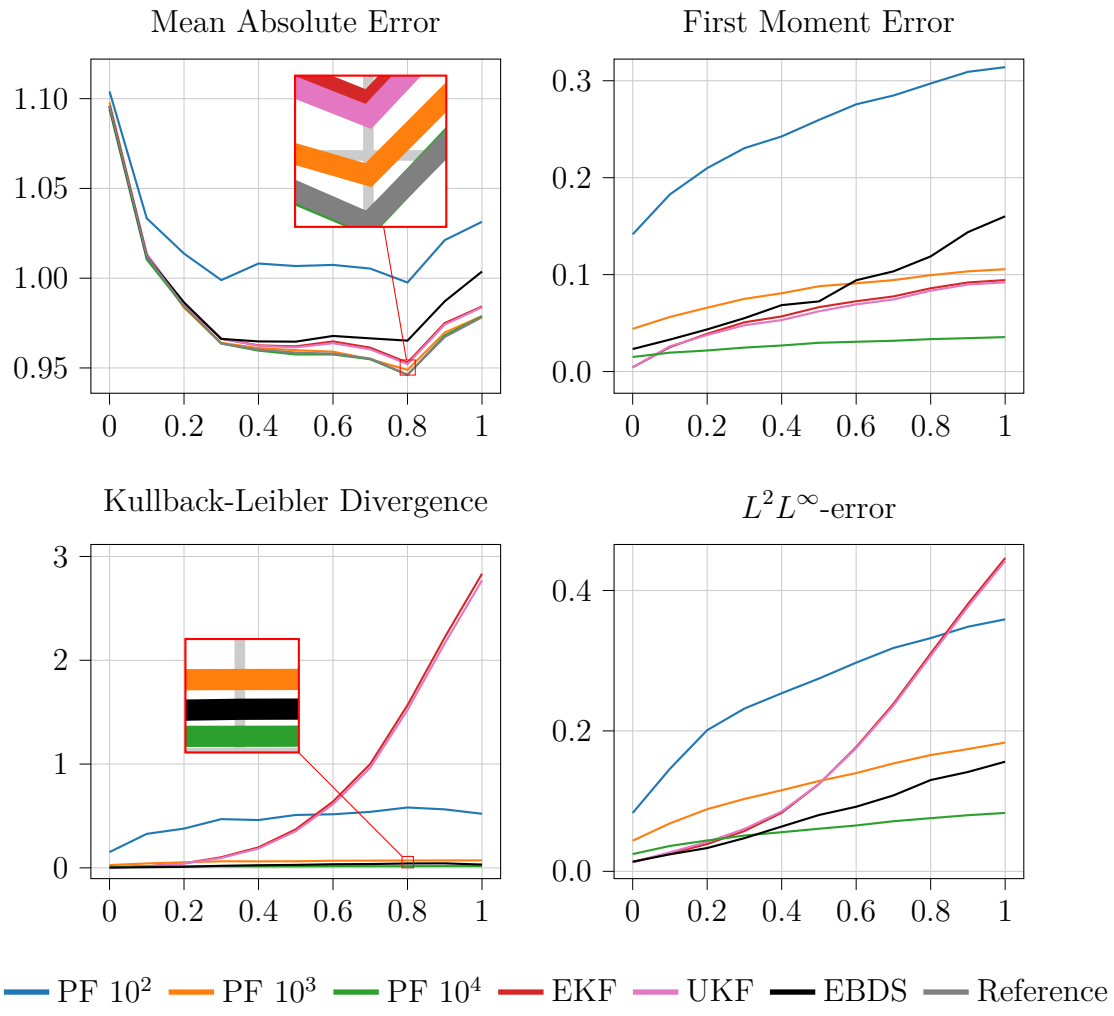


**Figure A.4:** Metrics over time for EBDS compared to benchmark particle filters and Kalman filters in the two-dimensional example.

Some example filtering densities in this example are provided in Figure A.5. For both components EBDS generally performs well in the shown example, especially compared to the extended Kalman filter and in the first component. However, the tails could be more accurate. This can be seen in particular for the second component at later times.

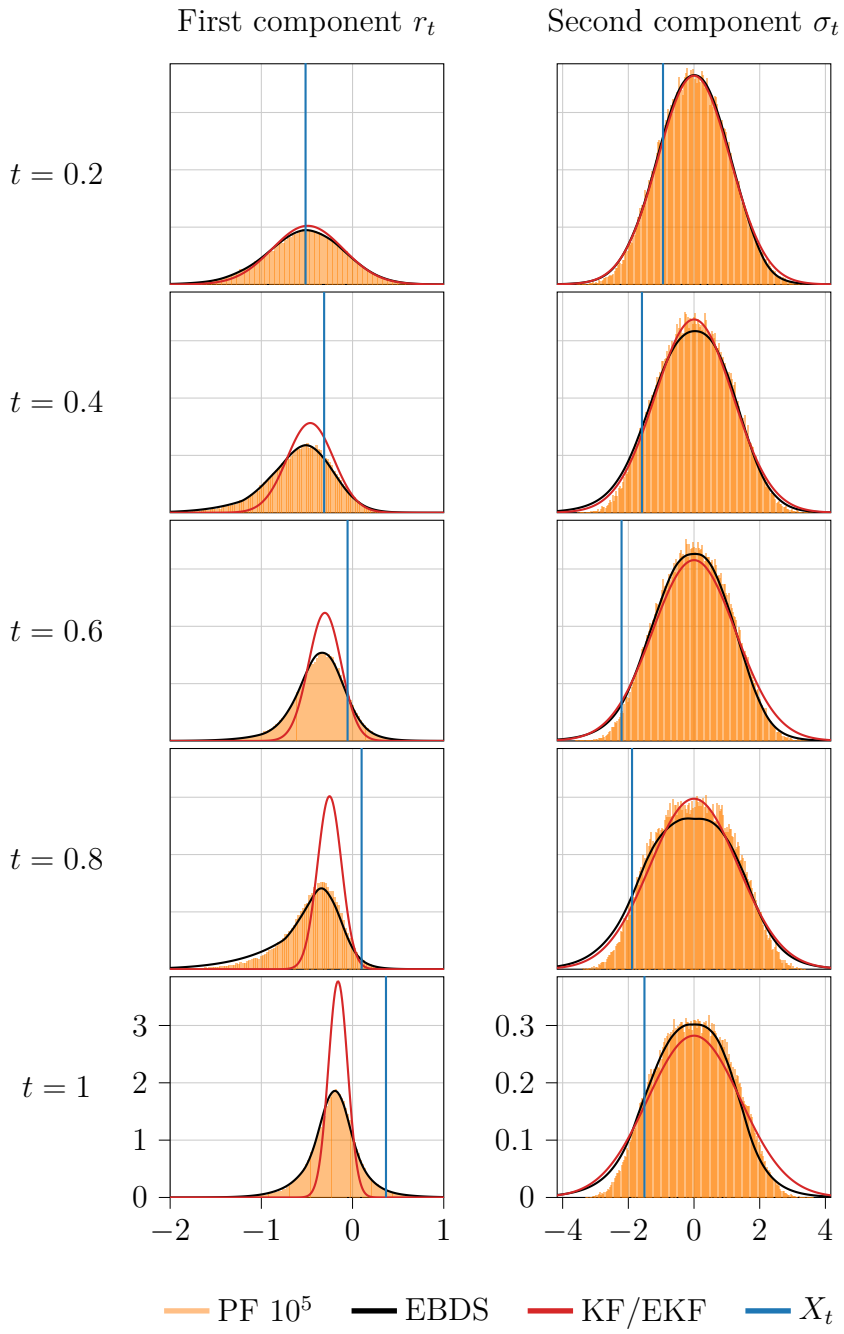


**Figure A.5:** Some filtering densities in one example for the two-dimensional problem. EBDS generally manages to capture the true density, but struggles somewhat with estimating tails. This can particularly be seen at later times for the second component.

# B

# Model Hyperparameters

All hyperparameters used in the six examples for evaluating EBDS are presented here. In the following text, OU represents the Ornstein–Uhlenbeck example, BM the bimodal example, GBM geometric Brownian motion, SIN the problem in two dimensions with stochastic diffusion, LV the stochastic Lotka–Volterra problem and SM the spring-mass problem.

Table B.1 shows hyperparameters related to training and network architecture. Moreover, a key step in the training stage is the update step normalisation detailed in Section 6.1. In the OU and BM examples this normalisation was done using quadrature with 2000 integration points in a uniform grid on the interval $[-10, 10]$. In the other examples the importance sampling scheme presented in (6.4) was used with varying numbers of samples $I$. In the GBM, SIN, LV and SM examples we used $I = 4000$, 8000, 10000 and 20000 samples respectively. Additionally, the method for calculating the importance density parameters varied for the different examples. In the GBM and LV examples the measurement function is invertible and we thus used $\mu^m = h^{-1}(y_k^m)$ as well as $\Sigma^m = 5\,\Sigma$. In the SIN and SM examples, for which the measurement function is not invertible, we let $\mu^m$ be given by the filter mean of an extended Kalman filter and $\Sigma^m$ by the covariance of the extended Kalman filter multiplied by 2.

Table B.2 shows the settings when testing the method. Again, a key step in the evaluation is to calculate normalising constants for EBDS filtering densities. The first moments for these same densities also need to be calculated. Calculation of normalising constants and first moments was done using quadrature for the OU and BM examples with 2000 points on the interval [-10, 10]. For the other examples importance sampling was used, see (6.4) and (6.14), with $I = 4000$ for GBM, $I = 2000$ for SIN, $I = 20000$ for LV and $I = 2000$ for SM. The same parameter calculation methods were used as in the training stage, see above. Additionally, the integrals for the KLD and $L^2L^2$-error for the OU and BM examples were calculated using quadrature with the same number of points and interval as the normalising constants and means. In the other four examples, these integrals were estimated using the importance sampling scheme in Section 6.3, see for instance (6.13). The number of samples used then was $10^4$ for all examples.

Finally, for the LSTM experiment in Section 7.5 we used $M = 10^6$ training samples and $M' = 10^3$ test samples. The reference filter was a particle filter with $10^4$ particles. Otherwise, the same hyperparameters were used for the standard neural

architecture as in the BM example detailed above. As mentioned in Chapter 7, we use hyperparameters for the LSTM architecture such that it has approximately the same number of trainable parameters as the standard architecture. For this purpose we utilise $n_h^1 = 64$ and $n_h^2 = 128$, see Figure 6.2. The same normalisation settings were used for both architectures in the training and evaluation stages. These were again the same hyperparameters as in the standard BM example presented above.

**Table B.1:** Hyperparameters pertaining to training and network architecture. The number of prediction steps is $N$, the number of training samples $M$, the hidden layer size $n_h$ and the learning rate $\alpha$.

| **Example** | $\boldsymbol{N}$ | $\boldsymbol{M}$ | $\boldsymbol{n_h}$ | $\boldsymbol{\alpha}$ |
|---|---|---|---|---|
| OU | $1, 2, 4, 8, 16, 32$ | $10^7$ | 128 | $10^{-4}$ |
| BM | $1, 2, 4, 8, 16, 32$ | $10^7$ | 128 | $10^{-3}$ |
| GBM | 16 | $10^7$ | 128 | $10^{-3}$ |
| SIN | 16 | $5 \cdot 10^6$ | 128 | $10^{-3}$ |
| LV | 16 | $5 \cdot 10^6$ | 128 | $10^{-3}$ |
| SM | 16 | $5 \cdot 10^6$ | 256 | $10^{-3}$ |

**Table B.2:** Hyperparameters pertaining to testing and evaluation. The prediction steps for the reference is $N_r$, the prediction steps for the benchmark filters $N_b$ and the number of test samples $M'$.

| **Example** | **Reference** | $\boldsymbol{N_r}$ | $\boldsymbol{N_b}$ | $\boldsymbol{M'}$ |
|---|---|---|---|---|
| OU | KF | 128 | 32 | $10^4$ |
| BM | PF $10^5$ | 128 | 32 | $5 \cdot 10^3$ |
| GBM | PF $10^5$ | 128 | 16 | $10^3$ |
| SIN | PF $10^5$ | 128 | 16 | $10^3$ |
| LV | PF $10^5$ | 128 | 16 | $10^3$ |
| SM | KF | 128 | 16 | $10^3$ |

**DEPARTMENT OF MATHEMATICAL SCIENCES**
**CHALMERS UNIVERISITY OF TECHNOLOGY**
Gothenburg, Sweden
www.chalmers.se


UNIVERSITY OF
GOTHENBURG


CHALMERS
UNIVERSITY OF TECHNOLOGY