CHALMERS
UNIVERSITY OF TECHNOLOGY

Deep-learning-accelerated Bayesian inference for state-space models

Master's thesis in Mathematics

Elias Hölén Hannouch
Oskar Holmstedt

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Deep-learning-accelerated Bayesian inference for state-space models

Elias Hölén Hannouch
Oskar Holmstedt

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2020

Deep-learning-accelerated Bayesian inference for state-space models
Elias Hölén Hannouch
Oskar Holmstedt

Supervisor: Adam Andersson, Smartr
Supervisor: Umberto Picchini, Department of Mathematical Sciences
Examiner: Moritz Schauer, Department of Mathematical Sciences

Master's Thesis 2020
Department of Mathemathical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

Deep-learning-accelerated Bayesian inference for state-space models
Elias Hölén Hannouch
Oskar Holmstedt
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Bayesian inference is an important statistical tool for estimating uncertainties in model parameters from data. One very important method is the Metropolis-Hastings algorithm, which allows for parameter inference when analytical solutions are intractable. The only requirement is that the likelihood function can be evaluated. However, it is a computationally expensive algorithm, as it is usually run for several thousand iterations. This is especially true for inference in state-space models, where the likelihood is computed via Bayesian filtering, which is a costly operation in and of itself. We propose a new method for doing Bayesian inference via Metropolis-Hastings for state-space models by replacing the standard likelihood computation with a neural network. The network is trained on data generated by a much shorter earlier run of Metropolis-Hastings.

We show, both qualitatively and quantitatively, that our method produces comparable results to the traditional method for several models. Moreover, our results indicate that the performance of our method is consistent as the dimensionality of the state-space model increases.

Finally, we show that our method is much more computationally efficient than the traditional method for large runs. We investigate at what point our method becomes the preferable alternative and find that the threshold occurs at quite small runs, both in terms of computational time and desired output size.

Keywords: Bayesian inference, MCMC, Metropolis-Hastings, state-space models, Bayesian filtering, surrogate likelihood, deep learning

Acknowledgements

Many thanks to our supervisors Adam Andersson of Smartr and Umberto Picchini of Chalmers for their support and feedback throughout the project. They truly went the extra mile for us. We would also like to thank the people at Smartr for welcoming us to their workplace. Furthermore, we must also thank both the Department of Mathematical Sciences and AI Innovation of Sweden for allowing us to use their computational resources. Finally, we extend our thanks to friends and family who have helped us through this strange time.

Elias Hölén Hannouch and Oskar Holmstedt, Gothenburg, June 2020

Contents

List of Figures

List of Tables

List of Algorithms

1
Introduction

A powerful tool for assessing model uncertainties in a probabilistic way is *Bayesian statistics*. Bayesian inference enables uncertainty quantification for unknown parameters $\boldsymbol{\theta}$ of a model, given observed data $\mathbf{y}$ of that model. The information in the data is encoded via the *likelihood function* $\mathcal{L}(\boldsymbol{\theta}|\mathbf{y})$. It can be used to update the experimenter's prior knowledge of the parameters via *Bayes' theorem*. The resulting quantity is the *posterior distribution* of the unknown parameters. This can sometimes be difficult to compute for various reasons. For instance, it requires a normalizing constant which is often unknown and may be difficult to approximate.

When we want to sample from a posterior distribution with a difficult or unknown algebraic form, we can instead use the *Metropolis-Hastings algorithm* (MH). It is a *Markov Chain Monte Carlo* (MCMC) method, which iteratively samples from a Markov chain, and converges in distribution to the sought posterior distribution.

In every step of the algorithm we need to compute the likelihood function [1]. However, while the likelihood function is central in obtaining this posterior distribution, computing it can be challenging for complex models. Hence the notion of a *surrogate likelihood*, which acts as an approximate and computationally more tractable replacement of the likelihood.

In this report the application is within the framework of Bayesian parameter estimation in *probabilistic state-space models*. State-space models describe the dynamics of an unobservable *hidden state* coupled with an observable *measured state*. These kind of models are used in a wide range of disciplines, including object tracking, biochemistry, and finance [2, 3, 4]. We use synthetic data simulated from several different theoretical models.

In order to compute the likelihood function of a state-space model, we need to use *Bayesian filtering*. The filters used in this report are the *Kalman filter* (KF), the *extended Kalman filter* (EKF) and the *bootstrap filter* (BF). These are computationally expensive and do not scale well with the dimensionality of the state-space. This is especially true for the bootstrap filter, which is a so-called particle filter.

The traditional approach for doing Bayesian parameter inference in this setting is to use the MH algorithm. In each iteration the likelihood is either approximated or computed exactly via a filter. Typically, you want to run the MH algorithm for a lot of iterations in order to get good results. Due to the complexity of the filters, this becomes very computationally heavy.

To alleviate this problem we propose a new algorithm for doing Bayesian inference in such environments, wherein *deep learning* is used to train an *artificial neural network* to approximate the likelihood function. Deep learning can be used to perform both regression and classification among other things. There are plenty of

different neural network architectures specialized for different problems. The kind of network used in this report is called a *fully connected feedforward neural network*, or a *multilayer perceptron*, and it is the most commonly used network for regression [5].

Similar studies have been made wherein alternative methods were used to approximate the likelihood function, for instance using Gaussian processes [6, 7]. Moreover, deep learning have been used before in the context of Bayesian inference to learn the posterior distribution directly [8], or to learn the likelihood function [9].

Broadly, the thesis aims to explore if deep learning can be used to accelerate the MH algorithm without sacrificing performance, with regards to accuracy and efficiency. To this end, we consider different state-space models, both linear and non-linear, and compare the results and computational times between our new method and the traditional method.

Specifically, we look at how the accuracy and efficiency of the method depends on the dimensionality of the state-space and the amount of training data used for the neural network. Moreover, we investigate at what point it becomes beneficial to use our method instead of the traditional one, both in terms of desired sample size and computational time.

The project is limited in its scope when it comes to the tested models, filters and network architectures. Furthermore, all data used is synthetic and produced by ourselves. This has the benefit of us knowing the true values of the parameters that we are trying to estimate, but it could have been of interest to test the methods on data from a real life experiment. Finally, time constraints also limited how far we were able to push the method.

The outline of the report is as follows. We begin in Chapter 2 by covering the basics of Bayesian inference and the specific elements of it that we use. Next, the models that we use for our experiments are presented in Chapter 3. Chapter 4 contains a brief introduction to the kind of neural network that we deploy. In Chapter 5, we give a walk-through of our deep-learning-accelerated Metropolis-Hastings algorithm, and present the experiments used to test it. The results of our experiments are detailed in Chapter 6 and discussed in Chapter 7. Finally, in Chapter 8 we draw some conclusions from our results and also suggest some future work on the topic.

2

Bayesian Inference

In this chapter we present the theory of Bayesian inference that is used in our experiments. We begin with a short summary of the Bayesian framework. Then, we introduce the important Metropolis-Hastings algorithm, which allows us to do Bayesian inference for difficult problems. Finally, we will also present elements of Bayesian filtering.

2.1 The Bayesian framework

The core of Bayesian inference is *Bayes' theorem*. Let A and B be events, satisfying $\mathrm{P}(B) \neq 0$. It holds that

$$\mathrm{P}(A|B) = \frac{\mathrm{P}(B|A)\mathrm{P}(A)}{\mathrm{P}(B)}. \tag{2.1}$$

It relates the conditional probability $\mathrm{P}(A|B)$ of observing A conditioned on B to the conditional probability $\mathrm{P}(B|A)$ and the marginal probabilities $\mathrm{P}(A)$ and $\mathrm{P}(B)$. The analogue of (2.1) for probability densities is

$$f_{X|Y}(x|y) = \frac{f_{Y|X}(y|x)f_X(x)}{f_Y(y)}, \tag{2.2}$$

where X and Y are random variables. Equation (2.2) also holds for probability mass functions and cumulative distribution functions.

Consider a model that depends on some parameters $\boldsymbol{\theta} \in \mathbb{R}^d$. In a Bayesian framework, we consider $\boldsymbol{\theta}$ to be a random variable, instead of a fixed constant as in the frequentist approach. We are interested in the distribution of the parameters $\boldsymbol{\theta}$ conditioned on observed data $\mathbf{y}$ from the model. This can be computed with Bayes' theorem

$$p(\boldsymbol{\theta}|\mathbf{y}) = \frac{p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{y})} = \frac{\mathcal{L}(\boldsymbol{\theta}|\mathbf{y})p(\boldsymbol{\theta})}{p(\mathbf{y})}. \tag{2.3}$$

Before conducting any experiments we have a distribution $p(\boldsymbol{\theta})$ which we believe holds for the parameters. This is called the *prior distribution*. The prior may come from an intuition about the model or knowledge from previous studies. After observing the data, we update our prior according to Bayes' theorem, giving us the *posterior distribution* $p(\boldsymbol{\theta}|\mathbf{y})$.

The function $p(\mathbf{y}|\boldsymbol{\theta})$ is called the *likelihood function* and is often denoted $\mathcal{L}(\boldsymbol{\theta}|\mathbf{y})$. We note that the order of $\boldsymbol{\theta}$ and $\mathbf{y}$ has changed. The reason is simply a matter of change in perspective. The likelihood $\mathcal{L}(\boldsymbol{\theta}|\mathbf{y})$ is considered a function of $\boldsymbol{\theta}$ for a

fixed $\mathbf{y}$, describing the probability density of observing the fixed data for a variable choice of parameters.

Equation (2.3) describes a density. By definition it must integrate to 1. The normalising constant is the marginal distribution of the data, also known as the *evidence*. It can be computed as $p(\mathbf{y}) = \int \mathcal{L}(\boldsymbol{\theta}|\mathbf{y})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}$. Often it is enough to know the likelihood and prior [1, chapter 2], and in that case we can omit the evidence and simply write

$$p(\boldsymbol{\theta}|\mathbf{y}) \propto \mathcal{L}(\boldsymbol{\theta}|\mathbf{y})p(\boldsymbol{\theta}).$$

How we choose the prior is a significant part of Bayesian inference. A smart choice of prior can result in a posterior distribution with a known analytical form. Furthermore, in some cases we can choose the prior in such a way that the posterior will be of the same form. Such distributions are called *conjugating distributions*, and we say that the prior is a *conjugate prior* to the model.

For instance, if we have independent identically distributed Gaussian data $\mathbf{x} = (x_1, \ldots, x_n)$, with known variance σ^2, and want to estimate the mean μ, then choosing a Gaussian prior, with mean μ_0 and variance σ_0^2, will in turn yield a Gaussian posterior with mean μ_1 and variance σ_1 given by

$$\mu_1 = \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right)^{-1}\left(\frac{\mu_0}{\sigma_0^2} + \frac{n\bar{\mathbf{x}}}{\sigma^2}\right), \quad \sigma_1^2 = \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right)^{-1},$$

where $\bar{\mathbf{x}}$ is the sample mean [10, chapter 3]. Figure 2.1 shows how the posterior distribution results from the interaction between the prior and the likelihood for this example with a specific choice of parameters.

However, in most cases the posterior distribution will not have a known analytical form. For those scenarios we can use numerical methods to estimate the posterior or to sample from it. These include several different Monte Carlo methods.

In contrast to the frequentist approach to statistical inference, Bayesian inference is primarily aimed at estimating distributions instead of point estimates for the parameters of interest. In the frequentist framework, the parameters are considered fixed. They can then be estimated by point estimators, which are random variables dependent on the random data. If we are interested in quantifying the uncertainty of our estimates, we can compute the standard error of the estimator. If the estimator has a known distribution or if it converges in distribution to a known asymptotic distribution, this allows us to form confidence intervals, giving us a measure of the uncertainty in our estimations. However, the posterior distribution obtained in Bayesian inference gives a more complete picture of the parameter uncertainty [11].

Even though the strength in Bayesian inference lies in creating a posterior distribution for the parameters, we are sometimes also interested in point estimates. There are multiple choices of point estimates, but the most common are *maximum a posteriori* (MAP), *posterior median* and *posterior mean* (also know as *minimum mean squared error* or MMSE). All three of these minimize the posterior expected value of different loss functions. The first one means the point with highest probability density in the posterior distribution, also known as the mode. The latter two are simply the median and mean of the posterior distribution, respectively.

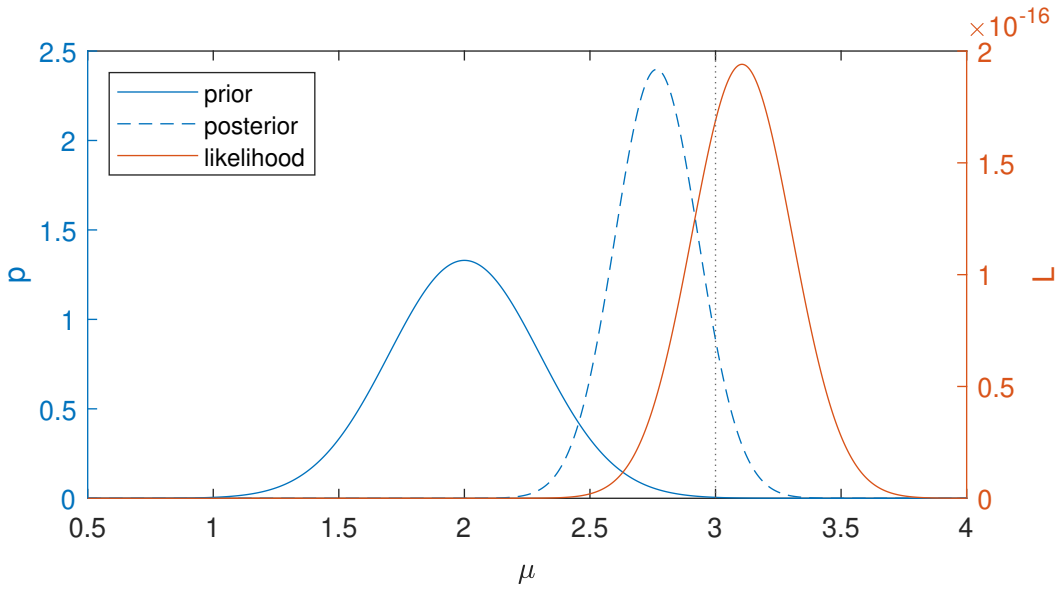


Figure 2.1: Demonstration of how the posterior is related to the prior and the likelihood. In this example, we have Gaussian data $X \sim \mathcal{N}(3, 1^2)$, and use a conjugate prior $\mu \sim \mathcal{N}(2, 0.3^2)$. After an observation of 25 independent samples, the prior is updated by Bayes' theorem. The resulting posterior is $\mathcal{N}(2.8, 0.17^2)$.

Finally, when we have the posterior distribution of the parameters we can also compute the predictive posterior distribution

$$p(\mathbf{y}^*|\mathbf{y}) = \int P(\mathbf{y}^*|\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{y})\mathrm{d}\boldsymbol{\theta}.$$

It gives us the distribution of a new observation $\mathbf{y}^*$ from the model, conditioned on the observed data $\mathbf{y}$ [1, chapter 2].

2.2 Metropolis-Hastings

In the previous section we mentioned that when the posterior distribution is not available analytically, we can use numerical methods to estimate it instead. One such method is the *Metropolis-Hastings* algorithm (MH), which can be used to sample from the posterior.

MH is a *Markov chain Monte Carlo* (MCMC) method for sampling from a distribution. It allows us to sample from distributions for which the closed form is unknown, or hard to compute. The only requirement is that we know that the target distribution, $p(\mathbf{x})$, is proportional to some known function $f(\mathbf{x})$, i.e.

$$p(\mathbf{x}) \propto f(\mathbf{x}). \tag{2.4}$$

As all densities must integrate to 1, the proportionality constant is $\int f(\mathbf{x})\mathrm{d}\mathbf{x}$, where the integration is taken over the entire domain of f. There are multiple reasons why sampling from such a distribution can be difficult. First of all, the normalizing constant can be very difficult to compute or unobtainable. Secondly,

even if the normalizing constant can be computed efficiently it is not guaranteed that the resulting distribution is something which we know how to sample from directly.

The Metropolis-Hastings algorithm bypasses these difficulties and instead iteratively produces a Markov chain which, under some conditions, asymptotically converges in distribution to the target distribution $p(\mathbf{x})$ as its stationary distribution. The method is presented in Algorithm 1 [12, 13].

The algorithm starts by assigning an initial $\mathbf{x}_0$ and setting $t = 0$. At each consecutive iteration a new value $\mathbf{x}^*$ is sampled from a so-called *proposal distribution* $g(\mathbf{x}^*|\mathbf{x}_t)$. We either want to accept or reject this new proposal depending on how likely it is to belong to the target distribution. To this end, we compute an *acceptance probability* given by

$$\alpha = \min\left(1, \frac{f(\mathbf{x}^*)}{f(\mathbf{x}_t)} \frac{g(\mathbf{x}_t|\mathbf{x}^*)}{g(\mathbf{x}^*|\mathbf{x}_t)}\right). \tag{2.5}$$

The acceptance probability consists of two ratios. The first one is the ratio between the posterior density of the new and previous sample. The normalizing factor in the target distribution is not needed because

$$\frac{f(\mathbf{x}^*)}{f(\mathbf{x}_t)} = \frac{f(\mathbf{x}^*)/\int f(\mathbf{x})\mathrm{d}\mathbf{x}}{f(\mathbf{x}_t)/\int f(\mathbf{x})\mathrm{d}\mathbf{x}} = \frac{p(\mathbf{x}^*)}{p(\mathbf{x}_t)}.$$

Thus, the acceptance probability gets larger if the proposed $\mathbf{x}^*$ has higher density than the previous sample, and smaller if it has lower density. The second ratio is taken between the proposal density for transitioning from the new sample to the previous one, and vice versa. This punishes the proposal for moving to regions from where it would be more difficult to come back from. Note that if g is a symmetric distribution, then this ratio will always be equal to 1.

The new proposal is accepted with probability α. This can be done by sampling from the uniform distribution on the unit interval, $u \sim \mathcal{U}(0, 1)$, and comparing the value to the acceptance probability. If $u \leq \alpha$, then the proposal is accepted, and we set $\mathbf{x}_{t+1} := \mathbf{x}^*$. Otherwise, it gets rejected, and we keep the previous value, i.e. we set $\mathbf{x}_{t+1} := \mathbf{x}_t$. The step counter t is then increased, and the algorithm repeats. The algorithm does not have any stopping criteria, rather it stops after a fixed number, T_{MH}, of iterations.

This procedure defines a Markov chain, in which in each step a new point is proposed, based on the previous point, and is either accepted or rejected. The chain converges asymptotically to a unique stationary distribution which is equal to the target distribution. In other words, the distribution of the state $\mathbf{x}_t$ at iteration t converges to the target distribution $p(\mathbf{x})$ as $t \to \infty$.

Thus, once the chain has converged sufficiently close to the target distribution, the method can be used to approximately sample from it. It should however be noted that subsequent samples will not be independent. If independent samples are desired, one can for example pick every jth sample for some j dependent on the autocorrelation of the chain. The period before the chain has converged sufficiently close to the target distribution is called the *burn-in period*. If all states from the burn-in period are discarded, the empirical distribution of the chain can be considered as an estimate of the target distribution.

In terms of user input, the burn-in period will be affected by the choice of initial sample $\mathbf{x}_0$. The more unlikely the initial sample is to belong to the target distributions the longer it will take for the algorithm to approach stationarity. Furthermore, the rate of convergence depends mainly on the shape of the proposal function. In general, the closer the proposal distribution is to the target distribution the better. Slow convergence happens particularly when the order of magnitude of the covariance of the target distribution is unknown, where a too large or small proposal function can severely inhibit the convergence. We will also present an MH algorithm which uses a proposal function g which adapts its shape during the course of the algorithm to alleviate this issue.

Algorithm 1 Metropolis-Hastings (MH)

Input: $\mathbf{x}_0, f(\mathbf{x}), g(\mathbf{x}'|\mathbf{x}), T_{\text{MH}}$
Output: $(\mathbf{x}_t)_{t=1}^{T_{\text{MH}}}$

1. **Initialization**
   (a) Set initial sample $\mathbf{x}_0$.
   (b) Set $t := 0$.
2. **Proposal**
   (a) Sample a proposal $\mathbf{x}^* \sim g(\mathbf{x}^*|\mathbf{x}_t)$.
   (b) Compute the acceptance probability $\alpha := \min\left(1, \frac{f(\mathbf{x}^*)}{f(\mathbf{x}_t)} \frac{g(\mathbf{x}_t|\mathbf{x}^*)}{g(\mathbf{x}^*|\mathbf{x}_t)}\right)$.
3. **Acceptance or rejection**
   (a) Draw a sample $u \sim \mathcal{U}(0, 1)$.
   (b) If $u \leq \alpha$, then accept $\mathbf{x}^*$ and set $\mathbf{x}_{t+1} := \mathbf{x}^*$.
   (c) If $u > \alpha$, then reject $\mathbf{x}^*$ and set $\mathbf{x}_{t+1} := \mathbf{x}_t$.
4. **Next step**
   (a) Set $t := t + 1$.
   (b) If $t < T_{\text{MH}}$ then return to step 2.
   (c) If $t = T_{\text{MH}}$ then break.

2.2.1 Adaptive Metropolis-Hastings algorithm

In the previous section we mentioned that the burn-in period of the Metropolis-Hastings algorithm is the number of samples until the algorithm has sufficiently converged to the stationary distribution. During this period, the shape of the distribution of accepted samples may change dramatically. If the initial sample x_0 is unlikely to come from the stationary distribution, then the proposal function may need to be larger to allow for the chain to converge within a reasonable time-frame. However, the draw-back is that the proposal may be too wide to propose points in a narrow stationary distribution. This leads to lower acceptance rates and thus more auto-correlation in the chain.

Algorithm 2 is an *adaptive Metropolis-Hasting* (AMH) algorithm which is proven to asymptotically converge to the target distribution, while being able to adapt the proposal distribution to ensure a higher acceptance rate [14].

We specify the proposal distribution to be a multivariate Gaussian distribution with covariance Σ_t, centered around the latest accepted proposal $\mathbf{x}^t$, as

$$g_{\Sigma_t}(\mathbf{x}^*|\mathbf{x}^t) = \mathcal{N}(\mathbf{x}^*|\mathbf{x}^t, \Sigma_t).$$

Note that the proposal function is symmetric and can now be removed from the calculation of the acceptance probability. In addition to choosing an initial sample $\mathbf{x}_0$, we also choose an initial covariance matrix Σ_0 and a time t_A at which to start the adaptive process. For the first t_A iterations we keep the initial covariance matrix. Then, starting at time step t_A, we compute a new covariance matrix based on the chain of accepted proposals $\mathbf{x}_0, \ldots, \mathbf{x}_t$ up until that point

$$\Sigma_t := s_d \operatorname{cov}(\mathbf{x}_0, \ldots, \mathbf{x}_t) + s_d \, \varepsilon \, I_d, \tag{2.6}$$

where s_d is a constant dependent on the dimension d of the parameter space. A suggested rule of thumb is $s_d = 2.4^2/d$ [14]. A small diagonal matrix $s_d \varepsilon I_d$ is added to avoid eventual numerical instability, where $0 < \varepsilon \ll 1$. The sample covariance is computed as

$$\operatorname{cov}(\mathbf{x}_0, \ldots, \mathbf{x}_t) = \frac{1}{t} \sum_{i=0}^{t} (\mathbf{x}_i - \bar{\mathbf{x}}_t)^T (\mathbf{x}_i - \bar{\mathbf{x}}_t),$$

where $\bar{\mathbf{x}}_t = \frac{1}{t+1} \sum_{i=0}^{t} \mathbf{x}_i$ is the sample mean of the chain up to that point.

At each subsequent time step the new mean vector and covariance matrix can be computed recursively according to the following formulas

$$\bar{\mathbf{x}}_t = \frac{1}{t+1} \left(t \bar{\mathbf{x}}_{t-1} + \mathbf{x}_t \right), \tag{2.7}$$

$$\Sigma_t = \frac{t-1}{t} \Sigma_{t-1} + \frac{s_d}{t} \left(t \bar{\mathbf{x}}_{t-1} \bar{\mathbf{x}}_{t-1}^T - (t+1) \bar{\mathbf{x}}_t \bar{\mathbf{x}}_t^T + \varepsilon I_d \right), \tag{2.8}$$

which greatly reduces the computational complexity [14].

2.2.2 Using Metropolis-Hastings for Bayesian inference

For our purposes the target distribution will be the posterior distribution $p(\boldsymbol{\theta}|\mathbf{y})$ of the parameters $\boldsymbol{\theta}$ given some observed data $\mathbf{y}$. As seen in section 2.1, the posterior is proportional to the likelihood times the prior,

$$p(\boldsymbol{\theta}|\mathbf{y}) = \frac{\mathcal{L}(\boldsymbol{\theta}|\mathbf{y}) \pi(\boldsymbol{\theta})}{p(\mathbf{y})} \propto \mathcal{L}(\boldsymbol{\theta}|\mathbf{y}) \pi(\boldsymbol{\theta}).$$

We can thus use MH to sample from the posterior distribution, as we do not need to know the normalizing factor $p(\mathbf{y})$. We let $\boldsymbol{\theta}$ be our $\mathbf{x}$ and $\mathcal{L}(\boldsymbol{\theta}|\mathbf{y})\pi(\boldsymbol{\theta})$ be our f as described in Equation (2.4) [1]. An example of this is shown in Figure 2.2.

When working with likelihood functions, it is often convenient to consider the logarithm of the likelihood function, $\ell(\boldsymbol{\theta}|\mathbf{y}) = \log \mathcal{L}(\boldsymbol{\theta}|\mathbf{y})$, which we call the *log-likelihood function*. The reason is that the likelihood function can have an extremely large range. This is in part because the likelihood function is often a product of probability densities. The large range in values can cause computational issues, as

Algorithm 2 Adaptive Metropolis-Hasting (AMH)

Input: $\mathbf{x}_0, \Sigma_0, f, t_\text{A}, s_d, \varepsilon, T_\text{MH}$
Output: $(\mathbf{x}_t)_{t=1}^{T_\text{MH}}$

1. **Initialization**
   (a) Set initial sample $\mathbf{x}_0$ and covariance Σ_0.
   (b) Set $t := 0$.
2. **Proposition**
   (a) Sample a proposal $\mathbf{x}^* \sim g_{\Sigma_t}(\mathbf{x}^*|\mathbf{x}_t)$.
   (b) Compute the acceptance probability $\alpha := \min\left(1, \frac{f(\mathbf{x}^*)}{f(\mathbf{x}_t)}\right)$.
3. **Acceptance or rejection**
   (a) Draw a sample $u \sim \mathcal{U}(0,1)$.
   (b) If $u \leq \alpha$ then accept $\mathbf{x}^*$ and set $\mathbf{x}_{t+1} := \mathbf{x}^*$.
   (c) If $u > \alpha$ then reject $\mathbf{x}^*$ and set $\mathbf{x}_{t+1} := \mathbf{x}_t$.
4. **Covariance**
   (a) Set $t := t + 1$.
   (b) If $t < t_\text{A}$, let $\Sigma_t := \Sigma_{t-1}$
   (c) If $t = t_\text{A}$, compute $\Sigma_t := s_d \operatorname{cov}(\mathbf{x}_0, \ldots, \mathbf{x}_t) + s_d \varepsilon I_d$.
   (d) If $t > t_\text{A}$, compute Σ_t recursively according to (2.8).
5. **Next step**
   (a) If $t < T_\text{MH}$ then return to step 2.
   (b) If $t = T_\text{MH}$ then break.

likelihoods may range from from smaller than the machine epsilon to larger than the maximum floating point number. Using the log-likelihood function significantly shrinks the range of values. Algorithms 1 and 2 can be reformulated to work with the log-likelihood. The acceptance probability expressed in terms of the log-likelihood is given by

$$\log \alpha := \min\left(0, \ell(\boldsymbol{\theta}^*|\mathbf{y}) - \ell(\boldsymbol{\theta}_t|\mathbf{y}) + \log \pi(\boldsymbol{\theta}^*) - \log \pi(\boldsymbol{\theta}_t)\right), \tag{2.9}$$

and to determine if the proposal is accepted or not, we check if $\log u \leq \log \alpha$.

Finally, we note that when using MH for Bayesian inference it is important that any proposal not within the support of the prior is rejected. Theoretically this is ensured by the fact that for such proposals $\pi(\boldsymbol{\theta}) = 0$, yielding an acceptance probability of $\alpha = 0$. However, when working on the log-scale, this will yield an undefined $\log \alpha$. Numerically, this is not an issue as it can be set to $-\infty$, effectively guaranteeing rejection.

2.3 Bayesian filtering

In this section we present the topic of Bayesian filtering. It is the process of estimating the states of a dynamic system, conditioned on observed noisy measurements of those states. In this report we use three different kinds of filters: the Kalman filter, the extended Kalman filter, and the Bootstrap filter. The Kalman filter is an exact solution to the Bayesian Filtering equations for a linear dynamical system. The extended Kalman filters is an approximate Gaussian filter that extends to non-linear

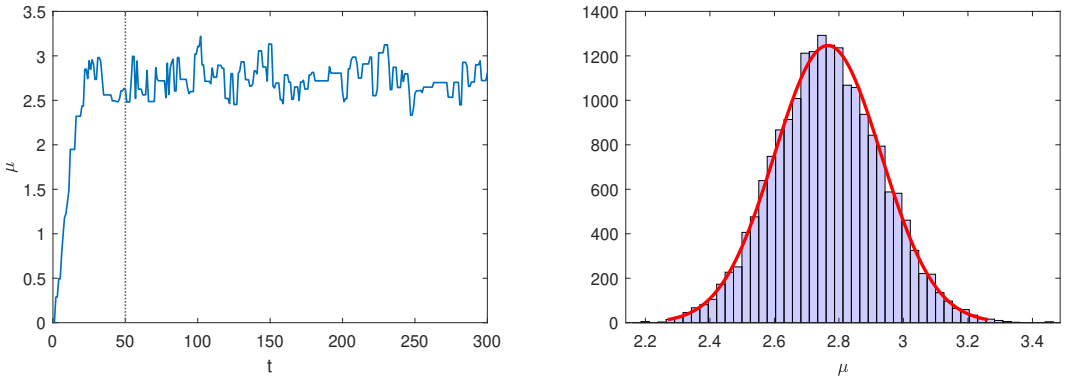


Figure 2.2: MH run, with the target distribution being the posterior of μ for Gaussian data, using a conjugate prior. The set up is the same as in the example shown in Figure 2.1. The proposal distribution is $\mathcal{U}(\mu_k - 0.5, \mu_k + 0.5)$. In the left-hand figure the trajectory of the chain is plotted, for the first 300 iterations. The burn-in period is marked by a dotted line. Note the short periods in which the function is constant. This equates to proposals being rejected. In the right-hand plot, the histogram of 10^4 samples from MH, where the burn-in has been removed, is shown. Compare this histogram to the true posterior in Figure 2.1.

dynamical systems. Finally, the bootstrap filter is a particle filter, and can be used for non-linear systems and when the transition model is poorly approximated by a Gaussian distribution, for example if it is multimodal or if it is not smooth.

2.3.1 State-space models

Discrete time filtering relies on so-called *probabilistic state-space models*. The general form for such a model is

$$\mathbf{x}_0 \sim p(\mathbf{x}_0), \tag{2.10a}$$
$$\mathbf{x}_k \sim p(\mathbf{x}_k|\mathbf{x}_{k-1}), \quad k = 1, \ldots, T, \tag{2.10b}$$
$$\mathbf{y}_k \sim p(\mathbf{y}_k|\mathbf{x}_k), \quad k = 1, \ldots, T. \tag{2.10c}$$

We call $\mathbf{x}$ and $\mathbf{y}$ the *hidden* and *measured states*, respectively. Only the measured states are observed, whereas the hidden states are unknown. At each time step, $\mathbf{x}_k$ and $\mathbf{y}_k$ are vectors in $\mathbb{R}^N$ and $\mathbb{R}^M$, respectively. We will use the notation $\mathbf{x}_{0:k} = (\mathbf{x}_0, \ldots, \mathbf{x}_k)$ to mean all states from time steps 0 to k. T is the final time step. Note that y_0 is not defined since x_0 represents an initial state.

The distribution (2.10a) is the prior density of $\mathbf{x}_0$. It describes the initial state of the system. The distribution (2.10b) is called the *transition* or *dynamic model*, and describes the dynamics of the hidden states. The distribution (2.10c) is called the *measurement model* and describes the distribution of the measured state given the corresponding hidden state.

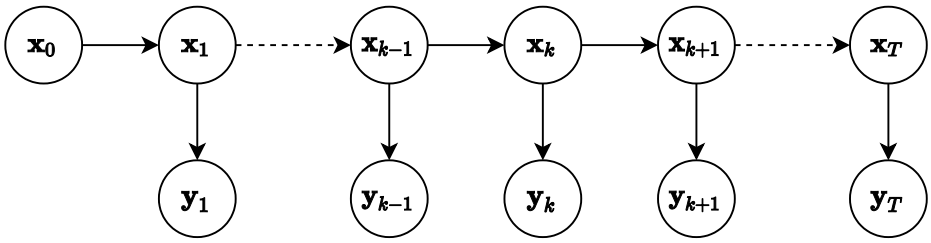


Figure 2.3: Graph representation of a state-space model with dependencies outlined. The values $x_0, \ldots, x_T$ represent the hidden states and $y_1, \ldots, y_T$ represent the measured states. We see that x_k and y_k are directly dependant on x_{k-1} and x_k, respectively.

State-space models are *Markovian*, meaning that the following properties hold:

$$p(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{y}_{1:k-1}) = p(\mathbf{x}_k|\mathbf{x}_{k-1}), \tag{2.11a}$$
$$p(\mathbf{x}_{k-1}|\mathbf{x}_{k:T}, \mathbf{y}_{k:T}) = p(\mathbf{x}_{k-1}|\mathbf{x}_k), \tag{2.11b}$$
$$p(\mathbf{y}_k|\mathbf{x}_{1:k}, \mathbf{y}_{1:k-1}) = p(\mathbf{y}_k, \mathbf{x}_k). \tag{2.11c}$$

Equation (2.11a) says that the distribution of $\mathbf{x}_k$ conditioned on all previous hidden and measured states, $\mathbf{x}_{0:k-1}$ and $\mathbf{y}_{1:k-1}$, only depends on the latest hidden state, $\mathbf{x}_{k-1}$. This is similar to the traditional Markov property, with the inclusion that it is also independent of the measured states. Similarly, equation (2.11b) states that the distribution of $\mathbf{x}_{k-1}$ conditioned on all future hidden and measured states only depends on the next hidden state. Finally, equation (2.11c) states that the distribution of the measured state $\mathbf{y}_k$ conditioned on all previous hidden and measured states as well as the current hidden state, only depends on the current hidden state [1, chapter 4]. Figure 2.3 shows a graph representation of the dependencies in a state-space model.

In general, state-space models depend on some parameters $\boldsymbol{\theta}$. We elect not to write out explicitly the conditioning on $\boldsymbol{\theta}$ throughout this section, but all distributions presented should be regarded as implicitly conditioned on the parameters of the model.

There are many different forms of state-space models. Among others, they include the very popular Hidden Markov models, which are discrete-valued. However, we will look exclusively at continuous-valued models with additive Gaussian noise. For such models, the distributions (2.10b) and (2.10c) can be expressed as the recurrence relations

$$\mathbf{x}_k = \mathbf{f}(k-1, \mathbf{x}_{k-1}) + \mathbf{q}_{k-1}, \tag{2.12a}$$
$$\mathbf{y}_k = \mathbf{h}(k, \mathbf{x}_k) + \mathbf{r}_k, \tag{2.12b}$$

where the functions $\mathbf{f} : \mathbb{R}\times\mathbb{R}^N \to \mathbb{R}^N$ and $\mathbf{h} : \mathbb{R}\times\mathbb{R}^N \to \mathbb{R}^M$ are called *transition* and *measurement* functions, respectively. The noise terms $\mathbf{q}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1})$ and $\mathbf{r}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$ are zero-mean multivariate Gaussian random variables with covariance matrices $\mathbf{Q}_{k-1}$ and $\mathbf{R}_k$, respectively.

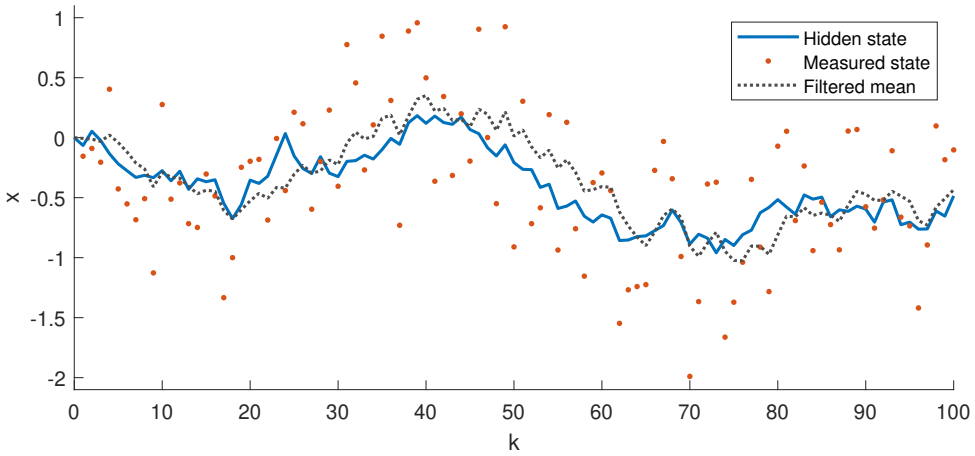


Figure 2.4: Linear dynamical system with $x_0 = 0$, $P_0 = 0$, $A = H = 1$, $Q_{k-1} = 0.1^2$, and $R_k = 0.5^2$. The hidden states are shown in blue solid line, the measured states are shown in red points. The filtered means from the KF are shown in gray dotted line.

These kind of systems are sometimes referred to as *dynamical systems*. If either $\mathbf{f}$ or $\mathbf{h}$ is nonlinear, we say that it is a *nonlinear dynamical system*, and if they are both linear, we say that it is a *linear dynamical system*. In the linear case the equations (2.12a) and (2.12b) simplify to

$$\mathbf{x}_k = \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{q}_{k-1}, \tag{2.13a}$$
$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{r}_k, \tag{2.13b}$$

where $\mathbf{A}_{k-1} \in \mathbb{R}^{N\times N}$ is called *transition matrix* and $\mathbf{H}_k \in \mathbb{R}^{M\times N}$ is called *measurement matrix* [1, chaper 4].

Two examples of one-dimensional state-space models are presented in Figures 2.4 and 2.5. In both cases the number of time steps is $T = 100$. The first model, depicted in 2.4, is a linear dynamical system with $A = H = 1$, $Q = 0.1^2$ and $R = 0.5^2$. This transition model describes a simple Gaussian random walk. The hidden and measured states are shown on top of another, in blue solid line an red points, respectively.

The second one, shown in 2.5, is a nonlinear dynamical system with $f(x) = x + e^{-x}$, $h(x) = e^x$, $Q = 0.1^2$ and $R = 10^2$. The same color coding was used for this case, but the hidden and measured states have been split up into two separate plots with different scales.

In both cases, the filtered means are also displayed in gray dotted line. These are Bayesian point estimates for the hidden states, and can be obtained by solving the Bayesian filtering equations, either exactly or approximately.

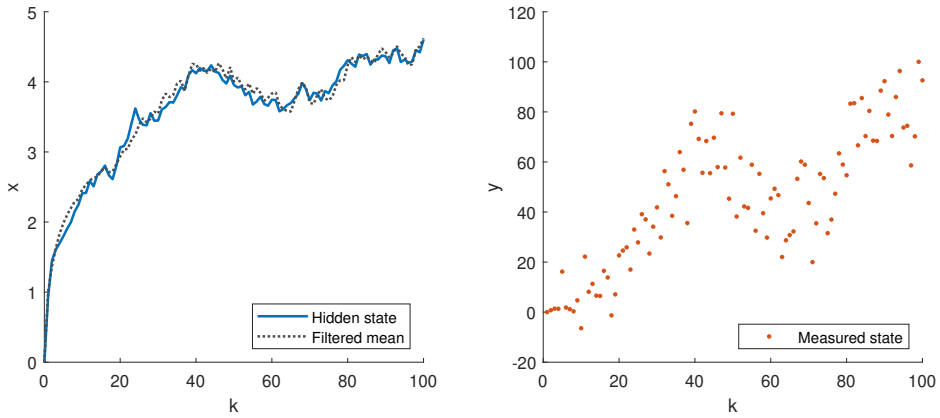


Figure 2.5: Nonlinear dynamical system with $x_0 = 0$, $P_0 = 0$, $f(x) = x + e^{-x}$, $h(x) = e^x$, $Q = 0.1^2$ and $R = 10^2$. In the left-hand figure the hidden states are shown in blue solid line and the filtered means are shown in gray dotted line. The measured states are shown in red points in the right-hand figure.

2.3.2 Bayesian filtering equations

Given an observation $\mathbf{y}_{1:T}$ of a state-space model, an obvious problem is to estimate the hidden states $\mathbf{x}_{0:T}$. This can be done by solving the *Bayesian filtering equations*. They are recursive formulas given by

$$p(\mathbf{x}_k|\mathbf{y}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{y}_{1:k-1})\mathrm{d}\mathbf{x}_{k-1}, \tag{2.14}$$

$$p(\mathbf{x}_k|\mathbf{y}_{1:k}) = \frac{p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{y}_{1:k-1})}{\int p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{y}_{1:k-1})\mathrm{d}\mathbf{x}_k}. \tag{2.15}$$

Equation (2.14) gives us the so-called *prediction distribution*. It describes the distribution of the next hidden state conditioned on all previous measured states. The formula can be derived by marginalizing and by using the Markov property (2.11a). If we also condition on the current measured state as in (2.15), we get the so-called *filtering distribution*. It can be derived using Bayes' theorem and the Markov property (2.11c). In the linear case these have exact solutions given by the Kalman filter. For nonlinear systems we can use approximate solutions to them, such as the extended Kalman filter or particle filters. As a point estimate for the hidden state at time k, we can use the mean of the filtering distribution [1, chapter 4].

Sometimes the distribution of interest is the marginal distribution $p(\mathbf{y}_{1:T})$. This is relevant if you want to do Bayesian inference with MH, as this becomes the likelihood function $p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$ when we explicitly condition on the parameters of the model. It can be obtained by filtering. First, it is decomposed in the following manner

$$p(\mathbf{y}_{1:T}) = \prod_{k=1}^{T} p(\mathbf{y}_k|\mathbf{y}_{1:k-1}). \tag{2.16}$$

Then, each contribution to this product can be computed in a similar manner to the prediction distribution, by

$$p(\mathbf{y}_k|\mathbf{y}_{1:k-1}) = \int p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|y_{1:k-1})\mathrm{d}\mathbf{x}_k. \tag{2.17}$$

Again, in the linear case this has a closed form solution, otherwise we can use approximate filters [1, chapter 12].

2.3.3 Kalman filter

The Kalman filter is the solution of the Bayesian filtering equations (2.14) and (2.15) for a linear dynamical system (2.13a)-(2.13b), with Gaussian prior $\mathbf{x_0} = \mathcal{N}(\mathbf{m}_0, \mathbf{P}_0)$. It gives us the following exact posterior distributions

$$\mathbf{x}_k|\mathbf{y}_{1:k-1} \sim \mathcal{N}(\tilde{\mathbf{m}}_k, \tilde{\mathbf{P}}_k) \tag{2.18a}$$
$$\mathbf{x}_k|\mathbf{y}_{1:k} \sim \mathcal{N}(\mathbf{m}_k, \mathbf{P}_k) \tag{2.18b}$$
$$\mathbf{y}_k|\mathbf{y}_{1:k-1} \sim \mathcal{N}(\mathbf{H}_k\tilde{\mathbf{m}}_k, \mathbf{S}_k) \tag{2.18c}$$

where $\tilde{\mathbf{m}}_k, \mathbf{m}_k, \tilde{\mathbf{P}}_k, \mathbf{P}_k$ and $\mathbf{S}_k$ are calculated iteratively according to Algorithm 3.

The algorithm can be derived by using two fundamental lemmas concerning joint and conditional distributions of multivariate Gaussian random variables. They are included as Lemma 1 and 2 in Appendix A.1.

The first two steps (2.19a) and (2.19b) are called the prediction steps. They give us the mean $\tilde{\mathbf{m}}_k$ and covariance $\tilde{\mathbf{P}}_k$ of the prediction distribution (2.18a). In (2.19a), the next state is predicted by applying the deterministic part of the transition model to the previously filtered mean.

The last four steps (2.19c)-(2.19g) are called the update steps or filtering steps. The vector $\mathbf{v}_k$ computed in (2.19c) is called the *innovation* and measures the difference between the observed measured state $\mathbf{y}_k$ and the expected measured state given the predicted mean. The matrix $\mathbf{S}_k$ is the measurement covariance or equivalently the covariance of the innovation $\mathbf{v}_k$. In (2.19e), we compute the so-called *optimal Kalman gain* $\mathbf{K}_k$, which determines how much the innovation will affect the filtering. In the next two steps (2.19f) and (2.19g), it is used to update the predicted mean and covariance. This gives us the mean $\mathbf{m}_k$ and covariance $\mathbf{P}_k$ of the filtering distribution (2.18b) [1, chapter 4].

If the transition function is an affine transformation $\mathbf{f}(\mathbf{x}_{k-1}) = \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{u}_{k-1}$ instead of a linear, we can still use the KF. We only need to replace step (2.19a) with $\tilde{\mathbf{m}}_k = \mathbf{A}_{k-1}\mathbf{m}_{k-1} + \mathbf{u}_{k-1}$.

The marginal distribution $p(\mathbf{y}_{1:T})$ (or the likelihood $p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$) can be computed as $p(\mathbf{y}_{1:T}) = \prod_{k=1}^{T} \mathcal{N}(\mathbf{y}_k|\mathbf{H}_k\tilde{\mathbf{m}}_k, \mathbf{S}_k)$. Note that this computation is exact and not an estimate.

Each iteration of KF involves several matrix multiplications and one matrix inversion. The fastest known algorithms for both of these problems have a computational complexity of $\mathcal{O}(N^{2.376})$, however in most implementations the time complexity is $\mathcal{O}(N^3)$. Thus, the complexity for filtering an observation $\mathbf{y}_{1:T}$ of length T is $\mathcal{O}(TN^3)$. In other words, the complexity of KF has cubic growth with the state-space dimensionality [15].

In figure 2.4 an example of KF is shown. The true hidden states are shown in blue solid line, whereas the filtered estimates are shown in gray dotted line. The model that is filtered is a simple Gaussian random walk model described in section 2.3.1. For this simple one-dimensional case, equations (2.19a)-(2.19g) can be simplified to

$$\begin{aligned}
\text{Predict} \quad & \tilde{m}_k = m_{k-1}, \\
& \tilde{P}_k = P_{k-1} + Q, \\
\text{Update} \quad & m_k = \tilde{m}_k + \frac{\tilde{P}_k}{\tilde{P}_k + R}(y_k - \tilde{m}_k), \\
& P_k = \tilde{P}_k - \frac{\tilde{P}_k^2}{\tilde{P}_k + R}.
\end{aligned}$$

Algorithm 3 Kalman filter (KF)

Input: $\mathbf{m}_0, \mathbf{P}_0, \mathbf{y}_{1:T}$
Output: $\tilde{\mathbf{m}}_{1:T}, \mathbf{m}_{1:T}, \tilde{\mathbf{P}}_{1:T}, \mathbf{P}_{1:T}, \mathbf{S}_{1:T}$,

1. Initial state and covariance $\mathbf{m}_0, \mathbf{P}_0$
2. For $k = 1, \ldots, T$
   (a) Predict mean
   $$\tilde{\mathbf{m}}_k = \mathbf{A}_{k-1}\mathbf{m}_{k-1} \tag{2.19a}$$
   (b) Predict covariance
   $$\tilde{\mathbf{P}}_k = \mathbf{A}_{k-1}\mathbf{P}_{k-1}\mathbf{A}_{k-1}^T + \mathbf{Q}_{k-1} \tag{2.19b}$$
   (c) Innovation vector
   $$\mathbf{v}_k = \mathbf{y}_k - \mathbf{H}_k\tilde{\mathbf{m}}_k \tag{2.19c}$$
   (d) Innovation covariance
   $$\mathbf{S}_k = \mathbf{H}_k\tilde{\mathbf{P}}_k\mathbf{H}_k^T + \mathbf{R}_k \tag{2.19d}$$
   (e) Kalman gain
   $$\mathbf{K}_k = \tilde{\mathbf{P}}_k\mathbf{H}_k^T\mathbf{S}_k^{-1} \tag{2.19e}$$
   (f) Update mean
   $$\mathbf{m}_k = \tilde{\mathbf{m}}_k + \mathbf{K}_k\mathbf{v}_k \tag{2.19f}$$
   (g) Update covariance
   $$\mathbf{P}_k = \tilde{\mathbf{P}}_k - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T \tag{2.19g}$$

2.3.4 Extended Kalman filter

For a non-linear system (2.12a)-(2.12b), the Bayesian filtering equations do not have a closed form solution. However, under certain assumptions we can use a filter called the *extended Kalman filter* (EKF). It gives an approximate solution to the Bayesian filtering equations, by using local linearization techniques to apply the same steps as in the KF.

The method is presented in Algorithm 4. The results are approximate Gaussian posterior distributions of the same form as (2.18a)-(2.18c). The derivation of the

EKF is similar to the one made for the KF, except we utilize approximate Gaussian joint distributions given by Approximation 1 in Appendix A.1, instead of the exact lemmas used to derive the KF. These approximations are constructed by first order Taylor expansions of the transition and measurement functions around the estimated posterior means of the previous iteration.

Comparing the steps of the EKF (2.20a)-(2.20g) with the KF, we see that the major differences is that in the prediction step (2.20a) and in the computation of the innovation vector (2.20c), states are propagated via the nonlinear functions $\mathbf{f}$ and $\mathbf{h}$ instead of via linear transforms. Moreover, the matrices $\mathbf{A}$ and $\mathbf{H}$ have been replaced by the Jacobian matrices $\mathbf{F}_{\mathbf{x}}(\mathbf{m}_{k-1})$ and $\mathbf{H}_{\mathbf{x}}(\tilde{\mathbf{m}}_k)$ in the computations of the covariances, (2.20b) and (2.20d), and the Kalman gain (2.20e).

Just like for the KF, the computational complexity of the matrix operations in EKF is $\mathcal{O}(N^3)$. However, the evaluation of the functions $\mathbf{f}$ and $\mathbf{h}$, as well as the Jacobian matrices $\mathbf{F}_{\mathbf{x}}$ and $\mathbf{H}_{\mathbf{x}}$, can potentially be of higher complexity. Thus, the best case complexity of EKF is $\mathcal{O}(TN^3)$ [15].

An example of EKF is visualized in Figure 2.5. The model is described in section 2.3.1. In this one-dimensional case the Jacobian matrices are reduced to just the derivatives. We get $f'(x) = 1 - e^{-x}$ and $h'(x) = e^x$.

Algorithm 4 Extended Kalman filter (EKF)

Input: $\mathbf{m}_0, \mathbf{P}_0, \mathbf{y}_{1:T}$
Output: $\tilde{\mathbf{m}}_{1:T}, \mathbf{m}_{1:T}, \tilde{\mathbf{P}}_{1:T}, \mathbf{P}_{1:T}, \mathbf{S}_{1:T}$

1. Initial state and covariance $\mathbf{m}_0, \mathbf{P}_0$
2. For $k = 1, \ldots, T$
   (a) Predict mean
   $$\tilde{\mathbf{m}}_k = \mathbf{f}(\mathbf{m}_{k-1}) \tag{2.20a}$$
   (b) Predict covariance
   $$\tilde{\mathbf{P}}_k = \mathbf{F}_{\mathbf{x}}(\mathbf{m}_{k-1})\mathbf{P}_{k-1}\mathbf{F}_{\mathbf{x}}^T(\mathbf{m}_{k-1}) + \mathbf{Q}_{k-1} \tag{2.20b}$$
   (c) Innovation vector
   $$\mathbf{v}_k = \mathbf{y}_k - \mathbf{h}(\tilde{\mathbf{m}}_k) \tag{2.20c}$$
   (d) Innovation covariance
   $$\mathbf{S}_k = \mathbf{H}_{\mathbf{x}}(\tilde{\mathbf{m}}_k)\tilde{\mathbf{P}}_k\mathbf{H}_{\mathbf{x}}^T(\tilde{\mathbf{m}}_k) + \mathbf{R}_k \tag{2.20d}$$
   (e) Kalman gain
   $$\mathbf{K}_k = \tilde{\mathbf{P}}_k\mathbf{H}_{\mathbf{x}}^T(\tilde{\mathbf{m}}_k)\mathbf{S}_k^{-1} \tag{2.20e}$$
   (f) Update mean
   $$\mathbf{m}_k = \tilde{\mathbf{m}}_k + \mathbf{K}_k\mathbf{v}_k \tag{2.20f}$$
   (g) Update covariance
   $$\mathbf{P}_k = \tilde{\mathbf{P}}_k - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T \tag{2.20g}$$

2.3.5 Particle filter

If either the transition or measurement model is not approximated well by a Gaussian approximation, then the EKF will perform badly. In such cases we can use a *particle filter*. Particle filters are Monte Carlo methods in which the basic idea is to generate new samples of the hidden states and weigh them based on how close to the observed data $\mathbf{y}_{1:T}$ they are. Thus, we will be able to produce Monte Carlo approximations to the distributions mentioned above.

Imagine that we want to compute the expected value of a function $\mathbf{g}(\mathbf{x})$ of a random variable $\mathbf{x}$, given some observed data $\mathbf{y}$. The Monte Carlo approximation of this is

$$\mathrm{E}(\mathbf{g}(\mathbf{x})|\mathbf{y}) = \int \mathbf{g}(\mathbf{x}) p(\mathbf{x}|\mathbf{y}) \mathrm{d}\mathbf{x} \approx \frac{1}{n} \sum_{i=1}^{n} \mathbf{g}(\boldsymbol{\xi}_i),$$

where $\xi_i \sim p(\mathbf{x}|\mathbf{y})$, for $i = 1, \ldots, n$. If the distribution $p(\mathbf{x}|\mathbf{y})$ is hard to sample from, we can use a technique called *importance sampling* where we instead sample from an *importance distribution* $q(\mathbf{x}|\mathbf{y})$. The samples are then weighed in the Monte Carlo approximation to account for that they come from another distribution. The approximation is then

$$\begin{aligned}\mathrm{E}(\mathbf{g}(\mathbf{x})|\mathbf{y}) = \int \mathbf{g}(\mathbf{x}) p(\mathbf{x}|\mathbf{y}) \mathrm{d}\mathbf{x} = \int \mathbf{g}(\mathbf{x}) \frac{p(\mathbf{x}|\mathbf{y})}{q(\mathbf{x}|\mathbf{y})} q(\mathbf{x}|\mathbf{y}) \mathrm{d}\mathbf{x} \approx \\ \approx \frac{1}{n} \sum_{i=1}^{n} \mathbf{g}(\boldsymbol{\xi}_i) \frac{p(\boldsymbol{\xi}_i|\mathbf{y})}{q(\boldsymbol{\xi}_i|\mathbf{y})} = \sum_{i=1}^{n} w_i \mathbf{g}(\boldsymbol{\xi}_i),\end{aligned}$$

where $\boldsymbol{\xi}_i \sim q(\mathbf{x}|\mathbf{y})$ and $w_i = \frac{1}{n} \frac{p(\boldsymbol{\xi}_i|\mathbf{y})}{q(\boldsymbol{\xi}_i|\mathbf{y})}$, for $i = 1, \ldots, n$. Although we avoid sampling from $p(\mathbf{x}|\mathbf{y})$, we still need to evaluate this density, which might also be hard. To avoid this, we use Bayes' theorem to get

$$\begin{aligned}\mathrm{E}(\mathbf{g}(\mathbf{x})|\mathbf{y}) = \int \mathbf{g}(\mathbf{x}) p(\mathbf{x}|\mathbf{y}) \mathrm{d}\mathbf{x} = \int \mathbf{g}(\mathbf{x}) \frac{p(\mathbf{y}|\mathbf{x}) p(\mathbf{x})}{\int p(\mathbf{y}|\mathbf{x}') p(\mathbf{x}') \mathrm{d}\mathbf{x}'} \mathrm{d}\mathbf{x} = \\ = \frac{\int \mathbf{g}(\mathbf{x}) \frac{p(\mathbf{y}|\mathbf{x}) p(\mathbf{x})}{q(\mathbf{x}|\mathbf{y})} q(\mathbf{x}|\mathbf{y}) \mathrm{d}\mathbf{x}}{\int \frac{p(\mathbf{y}|\mathbf{x}') p(\mathbf{x}')}{q(\mathbf{x}'|\mathbf{y})} q(\mathbf{x}'|\mathbf{y}) \mathrm{d}\mathbf{x}'} \approx \frac{\frac{1}{n} \sum_{i=1}^{n} \mathbf{g}(\boldsymbol{\xi}_i) \frac{p(\mathbf{y}|\boldsymbol{\xi}_i) p(\boldsymbol{\xi}_i)}{q(\boldsymbol{\xi}_i|\mathbf{y})}}{\frac{1}{n} \sum_{j=1}^{n} \frac{p(\mathbf{y}|\boldsymbol{\xi}_j) p(\boldsymbol{\xi}_j)}{q(\boldsymbol{\xi}_j|\mathbf{y})}}\end{aligned}$$

where $\boldsymbol{\xi}_i \sim q(\mathbf{x}|\mathbf{y})$, for $i = 1, \ldots, n$. By letting $v_i = p(\mathbf{y}|\boldsymbol{\xi}_i) p(\boldsymbol{\xi}_i) / q(\boldsymbol{\xi}_i|\mathbf{y})$ and defining the *importance weights* as $w_i = v_i / \sum_{j=1}^{n} v_j$, we get

$$\frac{\frac{1}{n} \sum_{i=1}^{n} \mathbf{g}(\boldsymbol{\xi}_i) \frac{p(\mathbf{y}|\boldsymbol{\xi}_i) p(\boldsymbol{\xi}_i)}{q(\boldsymbol{\xi}_i|\mathbf{y})}}{\frac{1}{n} \sum_{j=1}^{n} \frac{p(\mathbf{y}|\boldsymbol{\xi}_j) p(\boldsymbol{\xi}_j)}{q(\boldsymbol{\xi}_j|\mathbf{y})}} = \frac{\sum_{i=1}^{n} v_i \mathbf{g}(\boldsymbol{\xi}_i)}{\sum_{j=1}^{n} v_j} = \sum_{i=1}^{n} w_i \mathbf{g}(\boldsymbol{\xi}_i).$$

Since this holds for any function $\mathbf{g}$, another interpretation of the result is that we form an empirical approximate distribution to the posterior distribution

$$p(\mathbf{x}|\mathbf{y}) \approx \hat{p}(\mathbf{x}|\mathbf{y}) = \sum_{i=1}^{n} w_i \delta_{\boldsymbol{\xi}_i}(\mathbf{x}),$$

where $\delta_{\boldsymbol{\xi}_i} = \delta(\mathbf{x} - \boldsymbol{\xi}_i)$ is the Dirac delta function centered in $\boldsymbol{\xi}_i$.

For a general state-space model on the form (2.10a)-(2.10c), importance sampling can be performed sequentially to get Monte Carlo approximations to the Bayesian filtering equations (2.14)-(2.15). The method is called *sequential importance sampling* (SIS) and is shown in Algorithm 5. It is a particle filter. We consider n *particles* $(\boldsymbol{\xi}_k^{(i)})_{i=1}^n$ propagating over time $k = 0, \ldots, T$. For each particle and each time step there is an associated weight $w_k^{(i)}$. These produce the following approximation to the filtering distribution

$$p(\mathbf{x}_k|\mathbf{y}_{1:k}) = \hat{p}(\mathbf{x}_k|\mathbf{y}_{1:k}) \approx \sum_{i=1}^{n} w_k^{(i)} \delta_{\boldsymbol{\xi}_k^{(i)}}(\mathbf{x}_k).$$

The algorithm starts by sampling the initial states of the particles $\boldsymbol{\xi}_0^{(i)} \sim p(\mathbf{x_0})$ from the prior distribution. The initial weights are uniform. For each subsequent time step, we sample from the importance distribution $q(\mathbf{x}_{0:k}|\mathbf{y}_{1:k})$, via the construction

$$q(\mathbf{x}_{0:k}|\mathbf{y}_{1:k}) = q(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{y}_{1:k}) q(\mathbf{x}_{0:k-1}|\mathbf{y}_{1:k-1}).$$

Thus, we can simply propagate each particle according to the conditional importance distribution $q(\mathbf{x}_k|\boldsymbol{\xi}_{0:k-1}^{(i)}, \mathbf{y}_{1:k})$. The new weights are computed by the recursive formula

$$v_k^{(i)} = w_{k-1}^{(i)} \frac{p(\mathbf{y}_k|\boldsymbol{\xi}_k^{(i)}) p(\boldsymbol{\xi}_k^{(i)}|\boldsymbol{\xi}_{k-1}^{(i)})}{q(\boldsymbol{\xi}_k^{(i)}|\boldsymbol{\xi}_{0:k-1}^{(i)}, \mathbf{y}_{1:k})}, \quad i = 1, \ldots, n,$$

and then normalized so they sum to 1, i.e $w_k^{(i)} = v_k^{(i)} / \sum_{j=1}^n v_k^{(j)}$ [1, chapter 7].

If we want to approximate the marginal distribution $p(\mathbf{y}_{1:T})$ (or the likelihood $p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$), we can do so by inserting

$$p(\mathbf{y}_k|\mathbf{y}_{1:k-1}) \approx \hat{p}(\mathbf{y}_k|\mathbf{y}_{1:k-1}) = \sum_{i=1}^{n} v_k^{(i)},$$

into equation (2.16), where $v_k^{(i)}$ are taken before normalization has been performed [1, chapter 12]. This approximation is obtained by Monte Carlo integrating

$$\begin{aligned} p(\mathbf{y}_k|\mathbf{y}_{1:k-1}) &= \int p(\mathbf{y}_k|\mathbf{x}_k) p(\mathbf{x}_k|\mathbf{y}_{1:k-1}) \mathrm{d}\mathbf{x}_k = \\ &= \int p(\mathbf{y}_k|\mathbf{x}_k) p(\mathbf{x}_k|\mathbf{x}_{k-1}) p(\mathbf{x}_{k-1}|\mathbf{y}_{1:k-1}) \mathrm{d}\mathbf{x}_{k-1:k}. \end{aligned}$$

Intuitively, we can think of SIS as sampling new solutions to the transition model, that are guided by the measured states $\mathbf{y}_{1:T}$ via the importance distribution. The paths of the particles are then weighed, based on how likely they are to produce $\mathbf{y}$.

SIS have some issues, the largest being something called *particle degeneracy*. This happens when the particles have weights that are near zero, which makes the approximation worse. To alleviate this problem we introduce the *sequential importance resampling* algorithm (SIR), presented in Algorithm 6. At the end of an iteration, we resample n times with replacement from the empirical distribution on the discrete set $\{\boldsymbol{\xi}_k^{(i)}\}_{i=1}^n$ with probabilities given by the weights $w_k^{(1)}, \ldots, w_k^{(n)}$. Thus, particles with low weights are likely to be discarded, and particles with high weights are likely to be picked more than once. After this has been done the weights

must be reset uniformly, to account for the resampling. Resampling can be done either at each step, periodically or when needed.

There are two things that must be chosen in a particle filter: the number of particles and the importance distribution. It is common to choose a Markovian importance distribution, by which we mean that $q(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{y}_{1:k}) = q(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{y}_{1:k})$. Taking $q(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{y}_{1:k}) = p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{y}_k)$ gives us the *optimal importance distribution* in terms of variance of the importance weights [16, chapter 3]. For more on the optimal importance distribution, see Appendix A.2.

In Algorithm 7 we present the *bootstrap filter* (BF). It is a variation of SIR, for a particularly simple importance distribution, $q(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{y}_{1:k}) = p(\mathbf{x}_k|\mathbf{x}_{k-1})$. This is the state transition density and it is independent of $\mathbf{y_{1:k}}$. This choice also simplifies the weight formula to be just the measurement density $p(\mathbf{y}_k|\mathbf{x}_k)$. Since the importance distribution does not take the observed data into account, the particles are essentially propagated blindly. This leads to more unstable approximations. Therefore, resampling is done at each step and more particles are usually required than with other particle filters. However, this is compensated by the relative simplicity of the algorithm [1, chapter 7].

Choosing the number of particles can be a difficult task. It is clear that as we increase the number of particles, the approximations get better. However, this leads to an increased complexity. In general, the number of particles required grows exponentially with the dimension N of the system, giving particle filter a computational complexity of $\mathcal{O}(\lambda^N)$, for some $\lambda > 0$ [15]. In practice, the number of particles depends highly on the given model and is often tuned manually.

When using particle filters for likelihood computation in MH, it has been suggested that the number of particles should be chosen so that the standard deviation of the log-likelihood estimate should be in the range $[0.5, 1.5]$. This would give an optimal balance between computational time for the particle filter and the acceptance rate of MH [17].

Algorithm 5 Sequential importance sampling (SIS)

Input: $\mathbf{y}_{1:T}, n$
Output: $(\boldsymbol{\xi}_{0:T}^{(i)})_{i=1}^{n}, (w_{0:T}^{(i)})_{i=1}^{n}$,

1. Draw n initial samples $(\boldsymbol{\xi}_0^{(i)})_{i=1}^{n}$ from the initial prior distribution.

$$\boldsymbol{\xi}_0^{(i)} \sim p(\mathbf{x}_0), \quad i = 1, \ldots, n. \tag{2.21}$$

2. Set initial weights uniformly, $w_0^{(i)} = \frac{1}{n}$, $i = 1, \ldots, n$.
3. For $k = 1, \ldots, T$
   (a) Propagate samples according to the importance distribution.

$$\boldsymbol{\xi}_k^{(i)} \sim q(\mathbf{x}_k | \boldsymbol{\xi}_{0:k-1}^{(i)}, \mathbf{y}_{1:k}), \quad i = 1, \ldots, n. \tag{2.22}$$

   (b) Compute weights according to the weight formula

$$v_k^{(i)} = w_{k-1}^{(i)} \frac{p(\mathbf{y}_k | \boldsymbol{\xi}_k^{(i)}) p(\boldsymbol{\xi}_k^{(i)} | \boldsymbol{\xi}_{k-1}^{(i)})}{q(\boldsymbol{\xi}_k^{(i)} | \boldsymbol{\xi}_{0:k-1}^{(i)}, \mathbf{y}_{1:k})}, \quad i = 1, \ldots, n. \tag{2.23}$$

   (c) Compute the normalized weights $w_k^{(i)} = \frac{v_k^{(i)}}{\sum_{i=j}^{n} v_k^{(j)}}$, $i = 1, \ldots, n$.

Algorithm 6 Sequential importance resampling (SIR)

Input: $\mathbf{y}_{1:T}, n$
Output: $(\boldsymbol{\xi}_{0:T}^{(i)})_{i=1}^{n}, (w_{0:T}^{(i)})_{i=1}^{n}$,

1. Draw n initial samples $(\boldsymbol{\xi}_0^{(i)})_{i=1}^{n}$ from the initial prior distribution.

$$\boldsymbol{\xi}_0^{(i)} \sim p(\mathbf{x}_0), \quad i = 1, \ldots, n. \tag{2.24}$$

2. Set initial weights uniformly, $w_0^{(i)} = \frac{1}{n}$, $i = 1, \ldots, n$.
3. For $k = 1, \ldots, T$
   (a) Propagate samples according to the importance distribution.

$$\boldsymbol{\xi}_k^{(i)} \sim q(\mathbf{x}_k | \boldsymbol{\xi}_{0:k-1}^{(i)}, \mathbf{y}_{1:k}), \quad i = 1, \ldots, n. \tag{2.25}$$

   (b) Compute weights according to the weight formula

$$v_k^{(i)} = w_{k-1}^{(i)} \frac{p(\mathbf{y}_k | \boldsymbol{\xi}_k^{(i)}) p(\boldsymbol{\xi}_k^{(i)} | \boldsymbol{\xi}_{k-1}^{(i)})}{q(\boldsymbol{\xi}_k^{(i)} | \boldsymbol{\xi}_{0:k-1}^{(i)}, \mathbf{y}_{1:k})}, \quad i = 1, \ldots, n, \tag{2.26}$$

   (c) Compute the normalized weights $w_k^{(i)} = \frac{v_k^{(i)}}{\sum_{j=1}^{n} v_k^{(i)}}$, $i = 1, \ldots, n$.
   (d) Resample n times with replacement from the discrete distribution on $\{\boldsymbol{\xi}_k^{(i)}\}_{i=1}^{n}$ with probabilities given by the weights $w_k^{(1)}, \ldots, w_k^{(n)}$.
   (e) Reset the weights uniformly, $w_k^{(i)} = \frac{1}{n}$, $i = 1, \ldots, n$.

Algorithm 7 Bootstrap Filter (BF)

Input: $\mathbf{y}_{1:T}, n$
Output: $(\boldsymbol{\xi}_{0:T}^{(i)})_{i=1}^{n}, (w_{0:T}^{(i)})_{i=1}^{n}$,

1. Draw n initial samples $(\boldsymbol{\xi}_0^{(i)})_{i=1}^{n}$ from the initial prior distribution.

$$\boldsymbol{\xi}_0^{(i)} \sim p(\mathbf{x}_0), \quad i = 1, \ldots, n. \tag{2.27}$$

2. Set initial weights uniformly, $w_0^{(i)} = \frac{1}{n}$, $i = 1, \ldots, n$.
3. For $k = 1, \ldots, T$
   (a) Propagate samples according to the state-transition model.

$$\boldsymbol{\xi}_k^{(i)} \sim p(\mathbf{x}_k | \boldsymbol{\xi}_{k-1}^{(i)}), \quad i = 1, \ldots, n. \tag{2.28}$$

   (b) Compute weights according to the measurement model

$$v_k^{(i)} = p(\mathbf{y}_k | \boldsymbol{\xi}_k^{(i)}), \quad i = 1, \ldots, n, \tag{2.29}$$

   (c) Compute the normalized weights $w_k^{(i)} = \frac{v_k^{(i)}}{\sum_{j=1}^{n} v_k^{(i)}}$, $i = 1, \ldots, n$.
   (d) Resample from the discrete distribution on $\{\boldsymbol{\xi}_k^{(i)}\}_{i=1}^{n}$ with probabilities given by the weights $w_k^{(1)}, \ldots, w_k^{(n)}$.
   (e) Reset the weights uniformly, $w_k^{(i)} = \frac{1}{n}$, $i = 1, \ldots, n$.

3
Models

As stated above, the purpose of this project is to do Bayesian parameter inference in probabilistic state-space models of the form given by (2.12a)-(2.12b). In this chapter we present the models that we use for our experiments. The models are all derived from discretizations of stochastic differential equations, but exhibit different properties.

3.1 Stochastic differential equations

Stochastic differential equations (SDE) describe the dynamics of a continuous-time system with random perturbations. There are many different definitions of SDEs, but the most famous one is the Itô formulation. The general form is

$$\mathrm{d}\mathbf{X}_t = \mathbf{b}(t, \mathbf{X}_t)\mathrm{d}t + \boldsymbol{\sigma}(t, \mathbf{X}_t)\mathrm{d}\mathbf{W}_t, \tag{3.1}$$

where $\mathbf{W}_t$ is the standard Wiener process in $\mathbb{R}^N$. The function $\mathbf{b} : \mathbb{R} \times \mathbb{R}^N \to \mathbb{R}^N$ is called the *drift coefficient* and the function $\boldsymbol{\sigma} : \mathbb{R} \times \mathbb{R}^N \to \mathbb{R}^{N \times N}$ is called the *diffusion coefficient*. Equation (3.1) is convenient notation for the integral equation

$$\mathbf{X}_t = \mathbf{X}_{t_0} + \int_{t_0}^{t} \mathbf{b}(s, \mathbf{X}_s)\mathrm{d}s + \int_{t_0}^{t} \boldsymbol{\sigma}(s, \mathbf{X}_s)\mathrm{d}\mathbf{W}_s. \tag{3.2}$$

Here, the first integral is a Lebesgue integral and the second one is an Itô integral.

By a solution to (3.1) we mean a stochastic process $\mathbf{X}_t$ satisfying the integral equation (3.2). We call such a solution an *Itô diffusion*. It exists if the functions $\mathbf{b}$ and $\boldsymbol{\sigma}$ fulfill some conditions, for example linear growth and Lipschitz with regards to the spatial variable, uniformly in time [18].

In some special cases there exists closed form analytical solutions to (3.1). However, for most SDEs this is not the case and numerical approximations are required. The most basic numerical approximation method is the *Euler-Maruyama* method. The Euler-Maruyama approximation of (3.1) reads

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{b}(t_{k-1}, \mathbf{x}_{k-1})\Delta t_{k-1} + \boldsymbol{\sigma}(t_{k-1}, \mathbf{x}_{k-1})\Delta \mathbf{W}_{k-1}, \tag{3.3}$$

where $\Delta t_{k-1} = t_k - t_{k-1}$ and $\Delta \mathbf{W}_{k-1} = \mathbf{W}_k - \mathbf{W}_{k-1} \sim \mathcal{N}(\mathbf{0}, \Delta t_{k-1} I)$. In the ODE setting $\boldsymbol{\sigma} = 0$, the Euler-Maruyama method simplifies to the Euler method [19].

We recognise (3.3) as a transition model of the form (2.10b). Moreover, if we assume the diffusion coefficient matrix $\boldsymbol{\sigma}$ to be independent of $\mathbf{X}_t$, we get the transition model of a nonlinear dynamical system (2.12a).

To get a full state-space model, we include a general measurement model with additive noise. The result is

$$\begin{aligned}\mathbf{x}_k &= \mathbf{x}_{k-1} + \mathbf{b}(t_{k-1}, \mathbf{x}_{k-1})\Delta t_{k-1} + \boldsymbol{\sigma}(t_{k-1})\Delta \mathbf{W}_{k-1},\\ \mathbf{y}_k &= \mathbf{h}(k, \mathbf{x}_k) + \mathbf{r}_k,\end{aligned}$$

where our transition function is $\mathbf{f}(k-1, \mathbf{x}_{k-1}) = \mathbf{x}_{k-1} + \mathbf{b}(t_{k-1}, \mathbf{x}_{k-1})\Delta t_{k-1}$, and the transition noise is $\mathbf{q}_{k-1} = \boldsymbol{\sigma}(t_{k-1})\Delta \mathbf{W}_{k-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}(t_{k-1})\boldsymbol{\sigma}^T(t_{k-1})\Delta t_{k-1})$. The measurement function $\mathbf{h}$ can be any function and the measurement noise is $\mathbf{r}_k \sim \mathcal{N}(\mathbf{0}, R_k)$. We limit ourselves to SDEs where $\mathbf{b}$ and $\boldsymbol{\sigma}$ are independent of t and we only consider constant time steps Δt.

3.2 The Ornstein-Uhlenbeck process

The first model we consider is a one-dimensional *Ornstein-Uhlenbeck* (OU) process. It is the simplest model that we work with. The SDE that describes such a process is

$$\mathrm{d}X_t = -\beta(X_t - \alpha)\mathrm{d}t + \sigma \mathrm{d}W_t, \tag{3.4}$$

where $\beta > 0$, $\alpha \in \mathbb{R}$ and $\sigma > 0$ are constants. It has the exact solution

$$X_t = X_{t_0} e^{-\beta(t-t_0)} + \alpha(1 - e^{-\beta(t-t_0)}) + \sigma \int_{t_0}^{t} e^{-\beta(t-s)} \mathrm{d}W_s.$$

The solution consists of a deterministic part and an Itô integral. The expected value of the solution is $\mathrm{E}(X_t) = X_{t_0} e^{-\beta(t-t_0)} + \alpha(1 - e^{-\beta(t-t_0)})$, which corresponds to the solution in the deterministic case. We see that the process converges in expected value to α. It is therefore called a *mean reverting* process, as the further away it is from the steady state α the stronger it is pulled towards it. The parameter β determines the rate of mean reversion.

Another parameterization of the same model, with $\gamma = \alpha\beta$, is

$$\mathrm{d}X_t = -\beta X_t \mathrm{d}t + \gamma \mathrm{d}t + \sigma \mathrm{d}W_t. \tag{3.5}$$

If we discretize this equation using the Euler-Maruyama method we get

$$x_k = x_{k-1} - \beta x_{k-1}\Delta t + \gamma \Delta t + \sigma \Delta W_{k-1}. \tag{3.6}$$

Coupling this with a simple measurement model $y_k = x_k + r_k$, $r_k \sim \mathcal{N}(0, R)$, we get a state-space model. It is not quite a linear dynamical system, as the deterministic part of the transition model is an affine function. However, given some adjustments the Kalman filter can be applied to this kind of system as well.

In our experiments, the unknown parameters are β, γ and σ. The variance R of the measurement noise is considered fixed. A sample path of the Ornstein-Uhlenbeck process is shown in Figure 3.1. The parameters are $\gamma = 2$, $\beta = 0.1$, $\sigma = 0.01$. The number of time steps is $T = 100$ and the step length is $\Delta t = 1$. The measurement noise had the variance $R = 0.1^2$. The blue graph represents the hidden states, whereas the red graph represents the measured states.

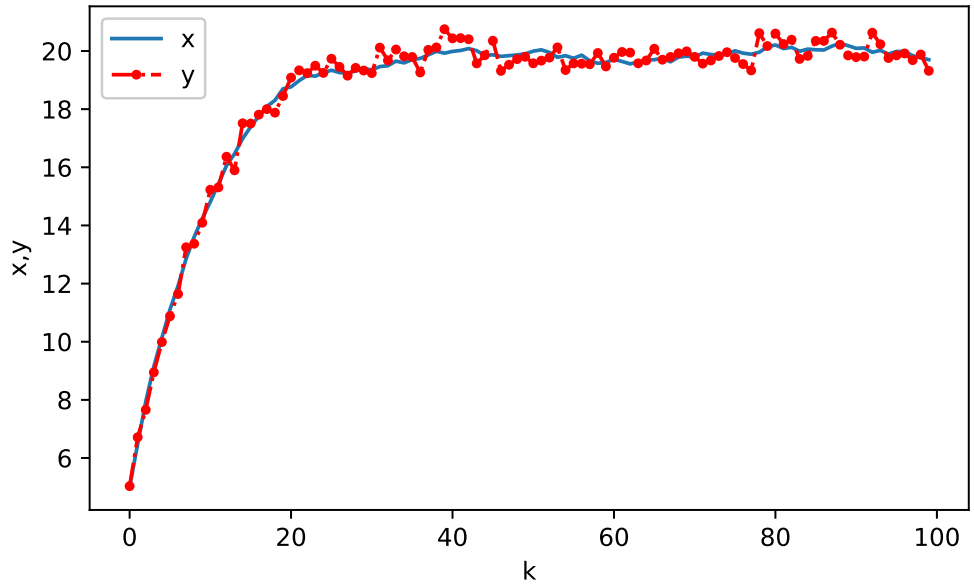


Figure 3.1: Sample path of an Ornstein-Uhlenbeck process, with $\gamma = 2$, $\beta = 0.1$, $\sigma = 0.1$, $T = 100$ and $\Delta t = 1$. The hidden state is shown in blue and the measured state is shown in red.

3.3 A linear spring-mass system

The second model we consider is a particular instance of the linear dynamical system, that can easily be scaled to higher dimensions. We call this model the *linear spring-mass* (LSM) system. The model describes the motion of M masses connected sequentially by springs. Their movement is determined by the following ODE

$$m_i\ddot{p}_i + (c_i + c_{i+1})\dot{p}_i + k_i(p_i - p_{i-1}) + k_{i+1}(p_i - p_{i+1}) = 0, \quad i = 1, \dots, M, \tag{3.7}$$

where p_i is the position of the ith mass relative to its steady state. The chain of masses is connected to fix walls at each end. Mathematically, this is expressed as $p_0 = p_{M+1} = 0$. The parameters of the model are the masses $m_1, \dots, m_M$, stiffness constants $k_1, \dots, k_{M+1}$ and damping constants $c_1, \dots, c_{M+1}$, all of which are assumed positive. A graphical representation of the system is presented in figure 3.2.

By letting $\mathbf{x} = (p_1, \dots, p_M, \dot{p}_1, \dots, \dot{p}_M)^T$, we can rewrite the second-order equation as a first-order equation

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x},$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{0}_{M\times M} & \mathbf{I}_{M\times M} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}, \tag{3.8}$$

and

$$\mathbf{A}_{21} = \begin{pmatrix} -\frac{k_1+k_2}{m_1} & \frac{k_2}{m_1} & & \\ \frac{k_2}{m_2} & \ddots & \ddots & \\ & \ddots & \ddots & \frac{k_N}{m_{N-1}} \\ & & \frac{k_M}{m_M} & -\frac{k_M+k_{M+1}}{m_M} \end{pmatrix}, \quad \mathbf{A}_{22} = \begin{pmatrix} -\frac{c_1+c_2}{m_1} & & \\ & \ddots & \\ & & -\frac{c_M+c_{M+1}}{m_M} \end{pmatrix}.$$

If we add a noise term, we get a stochastic differential equation

$$\mathrm{d}\mathbf{X}_t = \mathbf{A}\mathbf{X}_t\mathrm{d}t + \boldsymbol{\sigma}\mathrm{d}\mathbf{W}_t. \tag{3.9}$$

The diffusion coefficient matrix $\boldsymbol{\sigma}$ is assumed to be constant and of the form

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_1\mathbf{I}_{M\times M} & \mathbf{0}_{M\times M} \\ \mathbf{0}_{M\times M} & \sigma_2\mathbf{I}_{M\times M} \end{bmatrix}, \tag{3.10}$$

where $\sigma_1, \sigma_2 \geq 0$ are the variances of the noise on the positions and velocities, respectively.

The formal solution to (3.9) is

$$\mathbf{X}_t = e^{\mathbf{A}(t-t_0)}\mathbf{X}_{t_0} + \boldsymbol{\sigma}\int_{t_0}^{t} e^{\mathbf{A}(t-s)}\mathrm{d}\mathbf{W}_s.$$

Note that $e^{\mathbf{A}}$ is the matrix exponential function. The solution is similar to the solution (3.2) for the one-dimensional OU process. In fact, (3.9) describes a multivariate OU process with stationary mean $\mathbf{0}$.

If we assume that all parameters of the same type are equal, i.e. the masses $m_i = m$ for $i = 1, \ldots, M$, and the stiffness constants $k_i = k$ and the damping constants $c_i = c$ for $i = 1, \ldots, M+1$, we get a simplified model where

$$\mathbf{A}_{21} = \frac{k}{m}\begin{pmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{pmatrix}, \quad \mathbf{A}_{22} = -\frac{2c}{m}\mathbf{I}_{M\times M}. \tag{3.11}$$

We see that there are multiple choices of m, k and c which give equivalent systems. Therefore, in this case we consider the masses to be known and equal to 1.

Discretizing (3.9) using the Euler-Maruyama method and adding a linear measurement model with additive noise gives us a linear dynamical system

$$\begin{aligned} \mathbf{x}_k &= (\mathbf{I} + \mathbf{A}\Delta t)\,\mathbf{x}_{k-1} + \boldsymbol{\sigma}\Delta\mathbf{W}_{k-1}, \\ \mathbf{y}_k &= \mathbf{x}_k + \mathbf{r}_k, \end{aligned}$$

where $\boldsymbol{\sigma}\Delta\mathbf{W}_{k-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}^2\Delta t)$ and $\mathbf{r}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$. We choose $\mathbf{R} = R\mathbf{I}$ to be a constant diagonal matrix. In Figure 3.3 a sample path of the LSM system is shown.

3.4 A cubic spring-mass system

The linear spring-mass system (3.7) can be generalized by adding a cubic term. We call this new model the *cubic spring-mass* (CSM) system. The new equation is

$$m_i\ddot{p}_i + (c_i + c_{i+1})\dot{p}_i + h_i(p_i - p_{i-1}) + h_{i+1}(p_i - p_{i+1}) = 0, \quad i = 1, \ldots, N, \tag{3.12}$$

where $h_i(p) = k_i p - l_i p^3$. The nonlinear functions h_i replaces the stiffness constants of each spring in the linear model. The coefficients k_i fill the same role as the stiffness constants k_i in the linear model. However, the cubic terms decrease the effective stiffness of the springs. If all the coefficients l_i are zero, then the model reduces to the linear model. Using the same reparameterization as for the linear spring-mass system, we get a first order equation $\dot{\mathbf{x}} = \mathbf{b}(\mathbf{x})$, where

$$\mathbf{b}(\mathbf{x}) = \begin{pmatrix} x_{M+1} \\ \vdots \\ x_{2M} \\ -\frac{1}{m_1}(h_1(x_1) + h_2(x_1 - x_2) + (c_1 + c_2)x_{M+1}) \\ -\frac{1}{m_2}(h_2(x_2 - x_1) + h_3(x_2 - x_3) + (c_2 + c_3)x_{M+2}) \\ \vdots \\ -\frac{1}{m_{M-1}}(h_{M-1}(x_{M-1} - x_{M-2}) + h_M(x_{M-1} - x_M) + (c_{M-1} + c_M)x_{2M-1}) \\ -\frac{1}{m_M}(h_M(x_M - x_{M-1}) + h_{M+1}(x_M) + (c_M + c_{M+1})x_{2M}) \end{pmatrix} \tag{3.13}$$

or written differently

$$\mathbf{b}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \frac{1}{m_1}(l_1 x_1^3 + l_2(x_1 - x_2)^3) \\ \frac{1}{m_2}(l_2(x_2 - x_1)^3 + l_3(x_2 - x_3)^3) \\ \vdots \\ \frac{1}{m_{M-1}}(l_{M-1}(x_{M-1} - x_{M-2})^3 + l_M(x_{M-1} - x_M)^3) \\ \frac{1}{m_M}(l_M(x_M - x_{M-1})^3 + l_{M+1}x_M^3) \end{pmatrix}, \tag{3.14}$$

where $\mathbf{A}$ is the same matrix as for the linear spring-mass system, given by (3.8).

Analogous to the linear spring-mass system, we turn the ODE into an SDE by adding a noise term of the same form

$$\mathrm{d}\mathbf{X}_t = \mathbf{b}(\mathbf{X}_t) + \boldsymbol{\sigma}\mathrm{d}\mathbf{W}_t. \tag{3.15}$$

As opposed to the linear spring-mass system, this equation does not have a closed-form solution. We must therefore use numerical methods to solve it. After discretization and adding a measurement model we get a nonlinear dynamical system

$$\begin{aligned} \mathbf{x}_k &= \mathbf{x}_{k-1} + \mathbf{b}(\mathbf{x}_{k-1})\Delta t + \boldsymbol{\sigma}\Delta\mathbf{W}_{k-1}, \\ \mathbf{y}_k &= \mathbf{x}_k + \mathbf{r}_k. \end{aligned}$$

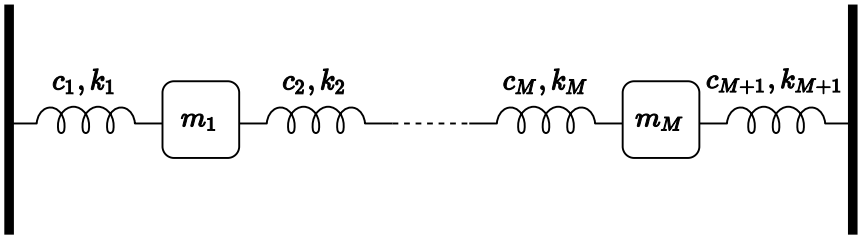


Figure 3.2: Schematics of the system described by (3.7). M masses with mass $m_1, \ldots, m_M$ connected sequentially by springs with stiffness constants $k_1, \ldots, k_{M+1}$ and damping constants $c_1, \ldots, c_{M+1}$.

In order to apply the EKF to an observation of this system, the Jacobian matrix of the transition and measurement functions are needed. The full transition function is $\mathbf{f}(\mathbf{x}) = \mathbf{x} + \mathbf{b}(\mathbf{x})\Delta t$. Its Jacobian matrix is

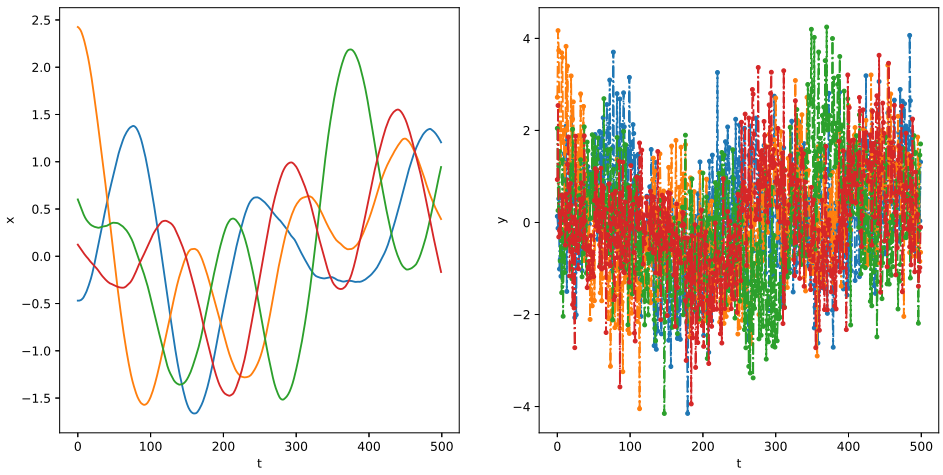
$$\mathbf{F}_{\mathbf{x}}(\mathbf{x}) = \mathbf{I} + \mathbf{A}\Delta t + \begin{bmatrix} \mathbf{0}_{M\times M} & \mathbf{0}_{M\times M} \\ \mathbf{B}(\mathbf{x}) & \mathbf{0}_{M\times M} \end{bmatrix} \Delta t,$$

where

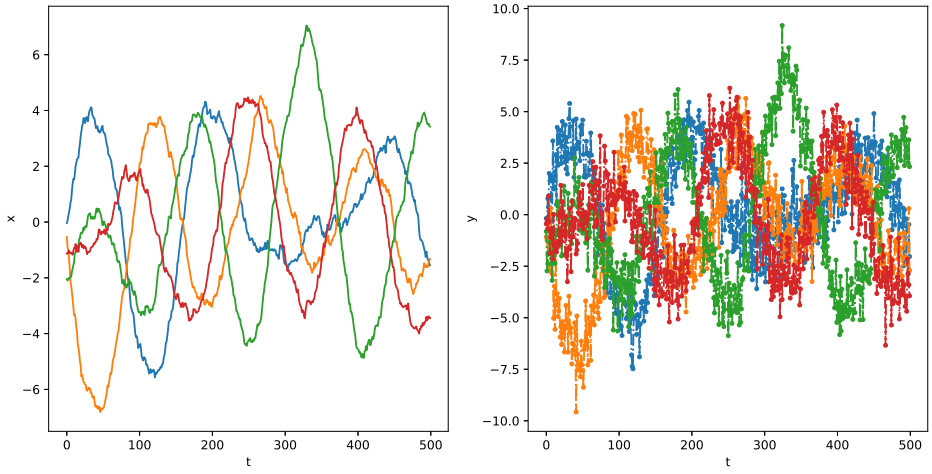
$$\mathbf{B}(\mathbf{x}) = 3 \begin{bmatrix} \frac{1}{m_1}(l_1 x_1^2 + l_2(x_1 - x_2)^2) & -\frac{l_2}{m_1}(x_1 - x_2)^2 & & \\ -\frac{l_2}{m_2}(x_2 - x_1)^2 & \ddots & \ddots & \\ & \ddots & \ddots & -\frac{l_M}{m_{M-1}}(x_{M-1} - x_M)^2 \\ & & -\frac{l_M}{m_M}(x_M - x_{M-1})^2 & \frac{1}{m_M}(l_M(x_M - x_{M-1})^2 + l_{M+1}x_M^2) \end{bmatrix}.$$

The chosen measurement function is the identity map $\mathbf{h}(\mathbf{x}) = \mathbf{x}$, and its Jacobian is simply $\mathbf{H}_{\mathbf{x}}(\mathbf{x}) = \mathbf{I}$.

Again, for our experiments we consider a simplified version of the model where all parameters of the same type are equal, i.e. $m_i = m$ for $i = 1, \ldots, M$, $k_i = k$, $l_i = l$, $c_i = c$ for $i = 1, \ldots, M + 1$. Furthermore, we consider the mass, and the covariances as known and fixed. The final parameters of the model are k,l and c.



(a)



(b)

Figure 3.3: Sample path of the linear sping-mass system with $k = 5$, $c = 0.1$, $m = 1$, $\sigma_1 = 0$, $\sigma_1 = 10$, $R = 1$, $M = 4$, $T = 500$ and $\Delta t = 0.01$. The left-hand plots of Figure 3.3a and Figure 3.3b show the positions and velocities of the masses, respectively. Each graph represents one of the 4 masses. The right-hand plots show the corresponding noisy measurements. 3.3b

4

Artificial neural networks

An artificial neural network is a model mainly used for regression and classification. The network is a structure made up of individual nodes and associated *weights* and *biases* which forwards input, and outputs a response value. It is sometimes thought of as a simplistic digital model for the biological neural activities of the brain. According to the universal approximation theorem, for any given tolerance level there exist a neural network which can approximate a given continuous function f on a compact subset of $\mathbb{R}^n$ with an absolute error smaller than this tolerance level.

Most commonly, a neural network will initially have randomised weights and biases. To properly map the input on the desired output, the network must be trained with some optimisation algorithm. This is called deep learning. For this project, we consider a feedforward neural network trained using the Adam optimiser, a stochastic gradient descent algorithm. It is the most commonly used network for regression purposes.

4.1 Feedforward neural network

In a feedforward neural network, the input is propagated through a number of *layers*. Each layer is the composition of an affine transformation and a nonlinear so-called *activation* function. We consider neural networks with vector inputs.

For a neural network with $L \geq 1$ number of layers of sizes $n_1, \ldots, n_L$, each layer $i \in \{1, \ldots, L\}$ has a weight matrix $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$ and a bias vector $\mathbf{b}_i \in \mathbb{R}^{n_i}$ associated with it. Each layer also has an associated activation function $f_i : \mathbb{R}^{n_i} \mapsto \mathbb{R}^{n_i}$ which enables nonlinear relations to be approximated. The output vectors from each layer are known as activations $\mathbf{a}_i \in \mathbb{R}^{n_i}$ and are defined as

$$\mathbf{a}_i := F_i(\mathbf{a}_{i-1}) = f_i(W_i\mathbf{a}_{i-1} + \mathbf{b}_i),$$

where the functions F_i are called layer maps. Note that the first activations are defined via the input $\mathbf{x} \in \mathbb{R}^{n_0}$ to the first input layer $\mathbf{a}_0 := \mathbf{x}$. The output from the neural network can thus be defined as

$$y := (F_L \circ \ldots \circ F_1)(\mathbf{x}).$$

We define the composition of the layer maps $\mathcal{F} := F_L \circ \ldots \circ F_1$ as the *neural network function*, which maps the input to the output.

4.2 Activation function

As previously mentioned, the network makes use of activation functions at every layer. The simplest activation function is the identity function. This function simply propagates the linear relationships formed in the layer. For the neural network to learn nonlinear relations, nonlinear activation functions are required. We utilise the most commonly used activation function, the *rectifier activation function* (ReLU)

$$f(\mathbf{x}) = \max(\mathbf{0}, \mathbf{x}), \tag{4.1}$$

which simply maps the output element-wise onto itself for positive contributions and zero elsewhere. Most activation functions are continuous and differentiable almost everywhere, which is required for the gradient calculation in the *backpropagation algorithm*. The gradient is used during the *stochastic gradient descent* which aims to minimise the loss described by a certain *loss function*.

4.3 Loss function

Consider a *training set* Ξ of tuples on the form (x, y) of corresponding input x and output y. There are various different loss functions, each with their own particular use. In this work, we utilise the mean smoothed L_1 loss function over this set as

$$C(\Xi) := \frac{1}{|\Xi|} \sum_{(x,y)\in\Xi} L_{1;\text{smooth}}(x, y), \tag{4.2}$$

where $L_{1;\text{smooth}}$ is defined as

$$L_{1;\text{smooth}}(x, y) := \begin{cases} \frac{1}{2}|y - \mathcal{F}(x)|, & \text{if } |y - \mathcal{F}(x)| \geq 1, \\ |y - \mathcal{F}(x)|^2 - \frac{1}{2}, & \text{if } |y - \mathcal{F}(x)| < 1. \end{cases} \tag{4.3}$$

The mean smoothed L_1 loss has the advantage of not having quadratic scaling of the errors, which can cause issues with outputs of varying sizes [20]. However, it is differentiable near the center as opposed to regular L_1 loss.

4.4 Stochastic gradient descent

Stochastic gradient descent (SGD) is a family of algorithms for updating the weights and biases in a neural network in an effort to minimise a *loss function*. These algorithms are greedy, which means that they take locally optimal steps at each stage. When learning, SGD makes iterative updates to the weights and biases in the directions derived from the gradient of the loss function. The method for updating the direction is decided by the optimiser.

Consider the set of all weights and biases

$$\omega := (W_1, \ldots, W_L, \mathbf{b}_1, \ldots, \mathbf{b}_L). \tag{4.4}$$

The backpropagation algorithm produces the estimates recursively for the gradient g_t at iteration step t, defined as

$$g_t := \frac{\partial C}{\partial \omega}. \tag{4.5}$$

The stochasticity is introduced through considering subsets of the data, often referred to as *minibatches*. The gradients are calculated for different subsets of Ξ and the average gradient is obtained, which is more robust in training. The optimiser then proceeds to form an update direction from the gradient g_t with the *learning rate* η and updates the weights and biases, for a given number of *epochs*. An epoch is defined as the amount of times all minibatches have been used to make an update step. In this report we use the *Adam optimiser*.

4.5 Adam optimiser

The Adam optimiser is an adaptive optimisation scheme which utilises the moving averages of the gradient estimated at each iteration [21]. It makes use of the first and second moments in proportion to the specification parameters $0 \leq \beta_1 \leq 1$ and $0 \leq \beta_2 \leq 1$ respectively. For the gradient g_t at iteration t, the first and second moments, m_t and v_t are updated according to

$$m_t := \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{4.6}$$
$$v_t := \beta_2 v_{t-1} + (1 - \beta_2) g_t^2. \tag{4.7}$$

Note that g_t^2 is produced as an element-wise multiplication of two g_t. Note also that m_t and v_t are biased estimators of g_t and g_t^2. We correct these biases with the following update

$$\hat{m}_t := \frac{m_t}{1 - \beta_1^t}, \tag{4.8}$$
$$\hat{v}_t := \frac{v_t}{1 - \beta_2^t}. \tag{4.9}$$

Consider using the learning rate η and a small constant ϵ, to avoid division by zero, whereafter the parameters are updated according to

$$\omega_t := \omega_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \tag{4.10}$$

The parameters β_1 and β_2 along with η decide how much the most recent gradient should be taken into consideration. In saddle points and other difficult regions with regards to convergence, it may be good to have high values of β_1 and β_2 along with a small η. There are some known default values which work in most situations but it is also possible to make *hyperparameter optimisation*, where the best values are deduced for the specific neural network structure and datasets considered.

4.6 Hyperparameter optimisation

In our hyperparameter optimisation scheme we consider the Tree Parzen Estimator [22]. It is a greedy algorithm based on the notion of expected improvement.

Define the space of hyperparameters as $\mathcal{X}$ and denote an element x in this set as a choice of hyperparameters. We define a fitness function $f : \mathcal{X} \to \mathbb{R}$ which we aim to maximise. A sound fitness function is the negative of the $L_{1;\text{smooth}}$ produced by the neural network. Proceed by introducing the probability of producing fitness value y by considering a hyperparameter $x \in \mathcal{X}$ as a function of the *history* $\mathcal{H}$, via $p_{\mathcal{H}}(y|x)$. The history $\mathcal{H}$ contains all previously considered hyperparameters $\{x_i\}_{i=1}^n$ and their corresponding fitness values $\{f(x_i)\}_{i=1}^n$. Define a threshold $y^*_{\mathcal{H}}$ as a function of the history $\mathcal{H}$ such that $P(y < y^*_{\mathcal{H}}) = \gamma$, for some quantile $\gamma \in (0, 1)$.

Consider the definition of the Expected Improvement measure for history $\mathcal{H}$

$$\text{EI}_{\mathcal{H}}(x) = \int_{\mathbb{R}} \max(y^*_{\mathcal{H}} - y, 0)\, p_{\mathcal{H}}(y|x)\text{d}y.$$

This measure estimates the expectation of producing a fitness value above $y^*_{\mathcal{H}}$ as a function of a hyperparameter choice $x \in \mathcal{X}$ and our history $\mathcal{H}$. However, we now utilise Bayes' theorem (2.1) to obtain

$$p_{\mathcal{H}}(y|x) = \frac{p_{\mathcal{H}}(x|y)p(y)}{p_{\mathcal{H}}(x)},$$

where we define

$$p_{\mathcal{H}}(x|y) = \begin{cases} l_{\mathcal{H}}(x) & \text{if } y < y^*_{\mathcal{H}}, \\ g_{\mathcal{H}}(x) & \text{if } y \geq y^*_{\mathcal{H}}. \end{cases}$$

The function $l_{\mathcal{H}}(x)$ is the density formed from the observations $\{x_i\}_{i=1}^n$ such that $f(x_i) < y^*_{\mathcal{H}}$ and $g_{\mathcal{H}}(x)$ is the density formed from the remaining ones. The distributions $l_{\mathcal{H}}$ and $g_{\mathcal{H}}$ are fitted to the data $\mathcal{H}$ using user-selected family of distributions. Available distributions are *uniform*, *log-uniform* and *categorical*. For the specifics regarding the implementation, see [23].

We derive the marginal distribution of x via

$$\begin{aligned} p_{\mathcal{H}}(x) &= \int_{\mathbb{R}} p_{\mathcal{H}}(x|y)p(y)\text{d}y = \int_{-\infty}^{y^*_{\mathcal{H}}} p_{\mathcal{H}}(x|y)p(y)\text{d}y + \int_{y^*_{\mathcal{H}}}^{\infty} p_{\mathcal{H}}(x|y)p(y)\text{d}y \\ &= l_{\mathcal{H}}(x) \int_{-\infty}^{y^*_{\mathcal{H}}} p(y)\text{d}y + g_{\mathcal{H}}(x) \int_{y^*_{\mathcal{H}}}^{\infty} p(y)\text{d}y = \gamma l_{\mathcal{H}}(x) + (1-\gamma)g_{\mathcal{H}}(x). \end{aligned}$$

Using Bayes' formula (2.1), we rewrite the expression as

$$\begin{aligned} EI_{\mathcal{H}}(x) &= \int_{\mathbb{R}} \max(y^*_{\mathcal{H}} - y, 0)\, p_{\mathcal{H}}(y|x)\, \text{d}y = \int_{\mathbb{R}} \max(y^*_{\mathcal{H}} - y, 0)\, \frac{p_{\mathcal{H}}(x|y)p(y)}{p_{\mathcal{H}}(x)}\, \text{d}y \\ &= \int_{-\infty}^{y^*_{\mathcal{H}}} (y^*_{\mathcal{H}} - y) \frac{l_{\mathcal{H}}(x)p(y)}{p_{\mathcal{H}}(x)}\, \text{d}y = \frac{l_{\mathcal{H}}(x)}{p_{\mathcal{H}}(x)} \int_{-\infty}^{y^*_{\mathcal{H}}} (y^*_{\mathcal{H}} - y)p(y)\, \text{d}y \\ &= \frac{\gamma y^*_{\mathcal{H}} l_{\mathcal{H}}(x) - l_{\mathcal{H}}(x) \int_{-\infty}^{y^*_{\mathcal{H}}} yp(y)\, \text{d}y}{\gamma l_{\mathcal{H}}(x) + (1-\gamma)g_{\mathcal{H}}(x)} \propto_x \left(\gamma + \frac{g_{\mathcal{H}}(x)}{l_{\mathcal{H}}(x)}(1-\gamma)\right)^{-1}. \end{aligned}$$

This suggests that to obtain the $x \in \mathcal{X}$ with regards to history $\mathcal{H}$ we should maximise $l_{\mathcal{H}}(x)$ and minimise $g_{\mathcal{H}}(x)$. On each iteration, the value x^* which maximises the derived expression is obtained and added,alongside the estimated fitness value $f(x^*)$, to $\mathcal{H}$. Note also that the choice of prior distribution $p(y)$ is less important for the convergence [22]. The algorithm thus iteratively maximises $\mathrm{EI}_{\mathcal{H}}(x)$, which works as surrogate model for maximising $f(x)$.

5
Method

In this chapter, we present a deep-learning-accelerated adaptive Metropolis-Hastings algorithm (AAMH) in which the log-likelihood is partly replaced by a neural network driven surrogate log-likelihood function. It is presented in a setting of probabilistic state-space models, but the general methodology is not restricted to this setting. First we present a version of the AMH algorithm where the log-likelihoods are estimated by a Bayesian filter, which we refer to as the *classic method* or *classic Metropolis-Hastings* (CMH). Then we present a scheme where the AMH algorithm utilises a neural network which has trained on the proposed parameters and their associated log-likelihoods from an initial run using the aforementioned classic method. We refer to this as the *mixed method* or *mixed Metropolis-Hastings* (MMH).

5.1 Metropolis-Hastings using Bayesian filters

Consider parameter inference for some unknown parameters of a state-space model. Define the *model parameters* of interest as $\boldsymbol{\theta} \in \mathbb{R}^d$, and an associated prior π_Θ with $\text{supp}(\pi_\Theta) = \Theta \subseteq \mathbb{R}^d$. Consider an observation $\mathbf{y}_{1:T}$ generated from this model, with the true parameters $\tilde{\boldsymbol{\theta}} \in \Theta$.

We are interested in using the AMH algorithm to sample from the resulting posterior distribution $p(\boldsymbol{\theta}|\mathbf{y}_{1:T})$. During the algorithm, we must estimate the log-likelihood value $\ell(\boldsymbol{\theta}^*|\mathbf{y}_{1:T})$ for every proposed parameter $\boldsymbol{\theta}^* \in \Theta$. For this purpose, we propose three different Bayesian filters, each of which can be used to estimate the log-likelihood via the general expression from (2.16) as

$$\ell(\boldsymbol{\theta}|\mathbf{y}_{1:T}) = \log\left(\prod_{k=1}^{T} p(\mathbf{y}_k|\mathbf{y}_{1:k-1}, \boldsymbol{\theta})\right) = \sum_{k=1}^{T} \log p(\mathbf{y}_k|\mathbf{y}_{1:k-1}, \boldsymbol{\theta}). \tag{5.1}$$

The KF and the EKF estimate these values via an exact and an approximate Gaussian density respectively, as

$$p(\mathbf{y}_k|\mathbf{y}_{1:k-1}, \boldsymbol{\theta}) \approx \mathcal{N}(\mathbf{y}_k|H_k\tilde{\mathbf{m}}_k, S_k), \text{ for } k \in \{1, \ldots, T\},$$

where $\mathbb{H}_k$, $\tilde{\mathbf{m}}_k$, and $\mathbf{S}_k$ are quantities computed by the filter. The BF approximates the expressions $p(\mathbf{y}_k|\mathbf{y}_{1:k-1})$ via Monte Carlo approximation

$$p(\mathbf{y}_k|\mathbf{y}_{1:k-1}, \boldsymbol{\theta}) \approx \frac{1}{n}\sum_{i=1}^{n} p(\mathbf{y}_k|\boldsymbol{\xi}_k^{(i)}, \boldsymbol{\theta}), \text{ for } k \in \{1, \ldots, T\},$$

where $\boldsymbol{\xi}_k^{(i)}$ denotes the i^{th} particle at time k.

As is explained in the AMH, Algorithm 2, one must provide an initial covariance guess Σ_0 and also a first sample guess $\boldsymbol{\theta}_0$. The algorithm parameters s_d and ε presented in Section 2.2.1, are set to $s_d = 2.4^2/d$ and $\varepsilon = 10^{-8}$. We also have the parameters T_C and t_a, specifying the number of posterior samples to be produced and the iteration at which the proposal function adaptation starts, respectively. The method is presented it in Algorithm 8.

The algorithm returns the sampled posterior points $(\boldsymbol{\theta}_k)_{k=0}^{T_C}$. The last instance of the covariance matrix Σ_{T_C} is also returned since continuing the sampling can then be done without going through another burn-in period. Note that during the run we are also generating and outputting the collections of proposal $(\boldsymbol{\theta}^{(t)})_{t=0}^{T_C}$ and their associated log-likelihood values $(\ell(\boldsymbol{\theta}^{(t)}|\mathbf{y}_{1:T}))_{t=0}^{T_C}$, which can then be used as training data for fitting a surrogate log-likelihood. $\boldsymbol{\theta}^{(k)}$ is the proposal from iteration k of the algorithm.

Algorithm 8 Classic Metropolis-Hastings using Bayesian filter (CMH)

Input $\mathbf{y}_{1:T}, T_C, \boldsymbol{\theta}_0, \Sigma_0, t_a$
Output $(\boldsymbol{\theta}_t)_{t=0}^{T_C}, (\boldsymbol{\theta}^{(t)})_{t=0}^{T_C}, (\ell(\boldsymbol{\theta}^{(t)}|\mathbf{y}_{1:T}))_{t=0}^{T_C}, \Sigma_{T_C}$

1. **Initialization**
   (a) Set initial sample $\boldsymbol{\theta}_0$ and covariance Σ_0
   (b) Set $t := 0$.
   (c) Calculate $\ell(\boldsymbol{\theta}_0|\mathbf{y}_{1:T})$ according to (5.1).
2. **Proposal**
   (a) Sample a proposal $\boldsymbol{\theta}^{(t+1)} \sim g_{\Sigma_t}(\boldsymbol{\theta}^*|\boldsymbol{\theta}_t)$.
   (b) Calculate the associated log-likelihood $\ell(\boldsymbol{\theta}^{(t+1)}|\mathbf{y}_{1:T})$ according to (5.1).
   (c) Calculate the logarithm of the acceptance rate α according to (2.9)

$$\log \alpha := \min\left(0, \ell(\boldsymbol{\theta}^{(t+1)}|\mathbf{y}_{1:T}) - \ell(\boldsymbol{\theta}_t|\mathbf{y}_{1:T}) + \log \pi_\Theta(\boldsymbol{\theta}^{(t+1)} - \log \pi_\Theta(\boldsymbol{\theta}_t)\right).$$

3. **Acceptance or rejection**
   (a) Draw a sample $u \sim \mathcal{U}(0,1)$.
   (b) If $\log u \leq \log \alpha$ then accept $\boldsymbol{\theta}^{(t+1)}$
      i. Set $\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}^{(t+1)}$.
      ii. Set $\ell(\boldsymbol{\theta}_{t+1}|\mathbf{y}_{1:T}) := \ell(\boldsymbol{\theta}^{(t+1)}|\mathbf{y}_{1:T})$.
   (c) If $\log u > \log \alpha$ then reject $\boldsymbol{\theta}^{(t+1)}$
      i. Set $\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t$.
      ii. Set $\ell(\boldsymbol{\theta}_{t+1}|\mathbf{y}_{1:T}) := \ell(\boldsymbol{\theta}_t|\mathbf{y}_{1:T})$.
4. **Covariance**
   (a) Set $t := t + 1$.
   (b) If $t < a$, let $\Sigma_t := \Sigma_{t-1}$.
   (c) If $t = t_a$, calculate $\Sigma_t := s_d \operatorname{cov}(\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_t) + s_d \varepsilon I_d$.
   (d) If $t > t_a$, calculate Σ_t recursively according to (2.8)
5. **Next step**
   (a) If $t < T_C$ then return to step 2.
   (b) If $t = T_C$ then break.

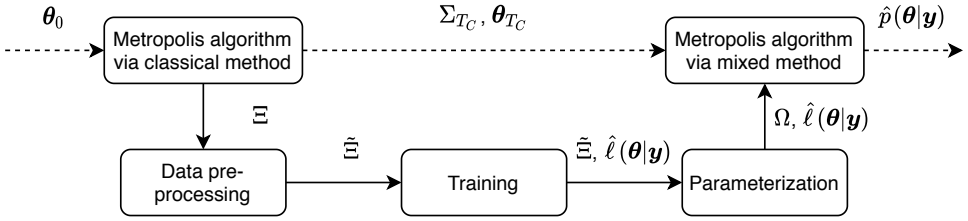


Figure 5.1: Flow chart of the AAMH algorithm. An initial run using CMH produces the data Ξ, which is pre-processed to produce the training data $\tilde{\Xi}$. The training data is used to parameterise a space Ω, centered around the data, and to train the neural network. Then the algorithm proceeds with MMH, in which the neural network acts as a surrogate log-likelihood estimator for proposals $\boldsymbol{\theta}^* \in \Omega$, except for $\boldsymbol{\theta}^* \notin \Omega$ when a Bayesian filter is used.

5.2 Overview of the AAMH algorithm

We are now ready to present our proposed method. We call it the accelerated adaptive Metropolis-Hasting algorithm (AAMH). The AAMH algorithm begins with an initial run of CMH, where the log-likelihood function $\ell(\boldsymbol{\theta}|\mathbf{y}_{1:T})$ is estimated using a Bayesian filter, see Algorithm 8. This produces a training data set Ξ of proposed parameters and associated log-likelihoods. The data set Ξ is then pre-processed, producing the training data set $\tilde{\Xi}$. A neural network is then trained on $\tilde{\Xi}$ and works as a function approximator $\mathcal{F}(\boldsymbol{\theta})$ of the log-likelihood for $\boldsymbol{\theta} \in \Omega$.

We introduce the region Ω as an approximation of the space that the training data set $\tilde{\Xi}$ spans. The region Ω is realised through a parameterization of the training data set $\tilde{\Xi}$, along with a classification scheme for determining if $\boldsymbol{\theta} \in \Omega$. Outside of this region the log-likelihood is computed by Bayesian filtering.

The AAMH then proceeds with an adjusted AMH algorithm, which utilises a mixed surrogate log-likelihood function presented in (5.4). This is called the *mixed Metropolis-Hastings* (MMH) and is presented in Algorithm 9.

The whole procedure is presented in Algorithm 10. A visual overview of the steps of the method is presented in Figure 5.1. Note that it is the full algorithm, including the creation of the training data and fitting of the neural network, which is referred to as AAMH, whereas the MMH only refers to the subroutine of running MH with a surrogate log-likelihood.

We will now present the method in detail. First,we describe how the data is generated. Then we describe the training of the neural network and the parameterization of the training data set. Lastly, we present the mixed method.

5.2.1 Generating and pre-processing the training data

We generate the training data Ξ through an initial run of CMH described in Algorithm 8. Since we are using an adaptive proposal distribution, the proposals become gradually more distributed according to the stationary distribution as the algorithm goes on. The proposals along with their associated log-likelihoods are stored as tuples $(\boldsymbol{\theta}^*, \ell(\boldsymbol{\theta}^*|\mathbf{y})) \in \Xi$. Note that this includes both rejected and accepted proposals.

The run is sufficiently long for the algorithm to converge to the stationary distribution and to produce enough proposed parameters to constitute a training data set. The required size of the data set might depend on how correlated the data is, the dimension of Θ and the quality of the data (for example when using very few particles in the BF).

Since the proposed parameters and log-likelihood values in Ξ constitutes raw data, it needs to be processed. The first step involves removing the burn-in period of the algorithm. The burn-in period can, in the unimodal case, be visually deduced as the point in the algorithm where the trajectory plot has seemingly converged to a relatively narrow strip. If the burn-in period accounts for the first t_b iterations of the CMH algorithm, the corresponding proposals $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(t_b)}$ are removed from the data set.

The second step involves removing any outliers. We consider two kinds of outliers. The first is proposals outside of Θ. These are removed as they will be rejected by the prior distribution π_Θ, regardless of their associated log-likelihood value. The second one is proposals with extremely large negative log-likelihood values. These are removed by systematically removing a set percentage of the most negative log-likelihood values from the training data. In the end, we are left with a processed training data set $\tilde{\Xi}$, which will be fed to the neural network.

5.2.2 Training the neural network

In the field of neural network, many times the design is a matter of intuition. In our case, we aimed to impose certain properties to the neural network and we present a design which takes some of the following properties in mind.

We first note that the dimension d of the domain Θ (which is also the input dimension to $\mathcal{F}$), is the number of dimensions to span. To properly account for the interaction between all components, the neural network may need to be larger for larger dimensions. Secondly, the state-space dimension N may lead to a more complex and informative log-likelihood, which means that the function $\mathcal{F}$ may need to learn to represent a more rapidly changing relationship.

These points leads us to consider a neural network design which increases in size according to Figure 5.2. The input size is d. The size of the hidden layers are given by $\lfloor 10\sqrt{N\,d} \rfloor$, where $\lfloor \cdot \rfloor$ is the floor operation. The output size is one-dimensional since it aims to approximate the univariate log-likelihood function.

The network is trained on the data set $\tilde{\Xi}$ using the Adam optimizer. The proposals and log-likelihood values are mean centered and normalized, which often have a positive impact on the training of the neural network.

5.2.3 Parameterization of the training region

The AMH algorithm utilises a Gaussian proposal distribution which has support on the whole of $\mathbb{R}^d$. This means that there is a chance of proposing parameters far outside the region spanned by the training data set $\tilde{\Xi}$. The neural network will not extrapolate well outside of this region. If log-likelihood values are systematically poorly estimated outside of this region, the AMH may diverge or get stuck.

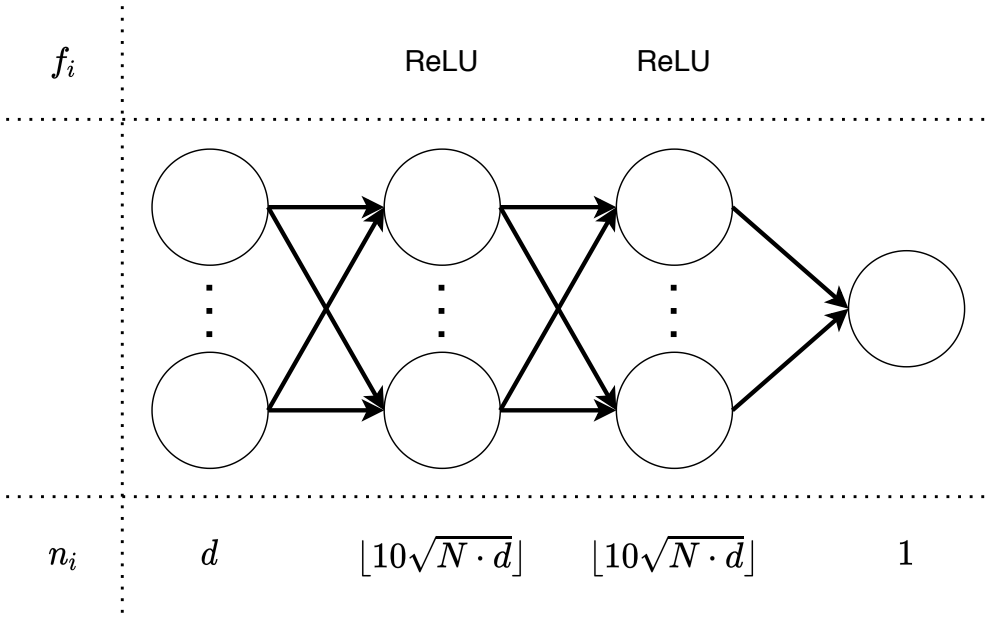


Figure 5.2: Illustration of the neural network structure considered in the project. The number of nodes in each layer is listed as n_i and the activation function (if any) as f_i for $i = 1, \ldots, 4$. The values N and d denotes the state-space dimensionality of the observation $\mathbf{y}$, and the number of parameters in Θ respectively.

We want to define a region Ω that roughly covers the data set. To this end, we assume that all the proposals come from one single multivariate Gaussian distribution with mean $\boldsymbol{\mu}_\Omega$ and covariance Σ_Ω. These are estimated by the sample mean and sample covariance, respectively. The data set $\tilde{\Xi}$ contains a subset of the proposals originating from different multivariate Gaussian distributions, since the proposal function g_Σ changes during the course of the AMH. However, assuming that the AMH have converged to the stationary distribution, the distribution should be approximately Gaussian.

The *Mahalanobis distance* is a metric that measures how far a point is from the mean of a Gaussian distribution. It is defined as

$$D(\boldsymbol{\theta}^*) = (\boldsymbol{\theta}^* - \boldsymbol{\mu}_\Omega)^T \Sigma_\Omega^{-1} (\boldsymbol{\theta}^* - \boldsymbol{\mu}_\Omega). \tag{5.2}$$

Under the Gaussian assumption the Mahalanobis distance is χ^2_d-distributed with d levels of freedom [24]. We can determine if a proposal $\boldsymbol{\theta}^*$ belongs to the distribution $\mathcal{N}(\boldsymbol{\mu}_\Omega, \Sigma_\Omega)$ with confidence level p by checking if

$$D(\boldsymbol{\theta}^*) \leq \chi^2_d(p), \tag{5.3}$$

where $\chi^2_d(p)$ is the pth quantile of the χ^2_d distribution. We define $\Omega_p = \{\boldsymbol{\theta} \in \mathbb{R}^d | D(\boldsymbol{\theta}) \leq \chi^2_d(p)\}$. This describes an ellipsoid centered in μ_Ω and with shape determined by Σ_Ω. By increasing p, we increase the size of the ellipsoid. In the limiting case of $p = 1$, we get $\Omega_p = \mathbb{R}^d$. See Figure 5.3 for an exemplification of the parameterization of Ω_p.

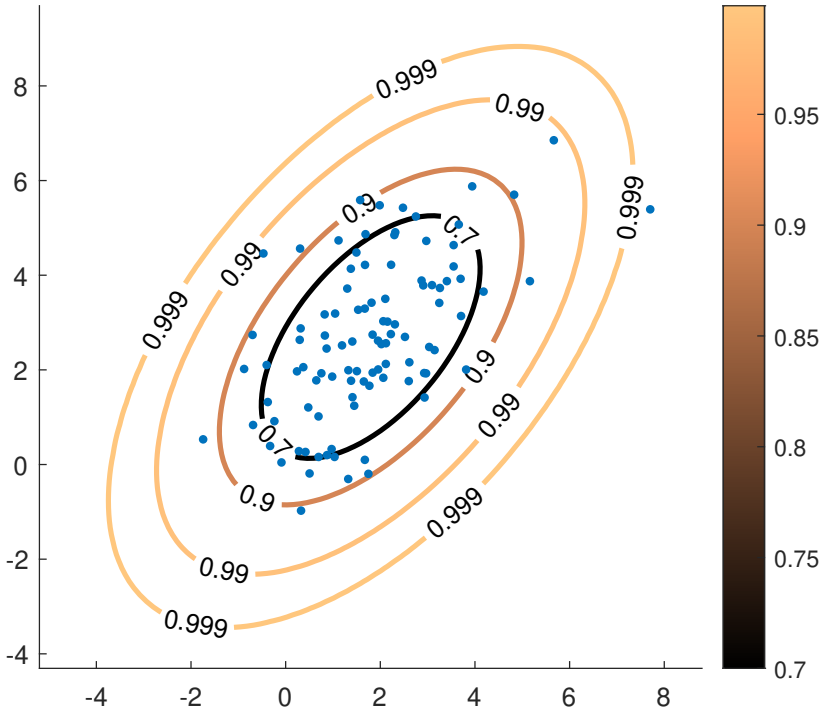


Figure 5.3: Exemplification of Mahalanobis distances. The ellipsoids define the region Ω_p based on some confidence level p and a 2D training data set $\tilde{\Xi}$.

5.2.4 Metropolis via the mixed method

We define the mixed surrogate log-likelihood $\hat{\ell}(\boldsymbol{\theta}|\mathbf{y})$ function, which utilises $\mathcal{F}$ only for $\boldsymbol{\theta} \in \Omega_p$, according to

$$\hat{\ell}(\boldsymbol{\theta}|\mathbf{y}) := \begin{cases} \mathcal{F}(\boldsymbol{\theta}) & \text{if } \boldsymbol{\theta} \in \Omega_p, \\ \ell(\boldsymbol{\theta}|\mathbf{y}) & \text{otherwise.} \end{cases} \tag{5.4}$$

The AMH algorithm is now continued using the surrogate log-likelihood $\hat{\ell}(\boldsymbol{\theta}|\mathbf{y})$. The adaptive covariance is started using Σ_{T_C} from the previous run and the proposal function is centered around the last accepted proposal $\boldsymbol{\theta}_{T_C}$. We call this the mixed Metropolis-Hastings, and it is presented in Algorithm 9.

The mixed method utilises the surrogate log-likelihood $\hat{\ell}(\boldsymbol{\theta}|\mathbf{y})$ which calls on the neural network for proposals in Ω_p. For this to be effective, it presupposes that the region Ω_p corresponds well to the stationary distribution, in other words the posterior distribution. The contrary would imply that the algorithm leaves the region Ω_p and thus stops using the neural network as a surrogate log-likelihood. The crucial feature of the AAMH algorithm is thus that, whenever $\boldsymbol{\theta}^* \in \Omega_p$, we compute the surrogate log-likelihood via deep learning prediction, rather than using the possibly expensive filtering procedure.

The MMH is run until a satisfactory number, T_M, of posterior samples are obtained. The chain produced by the MMH can be merged with the chain from the previous run of the CMH, to produce the final output of the AAMH $(\boldsymbol{\theta}_t)_{t=0}^{T_M+T_C}$.

Algorithm 9 Mixed Metropolis-Hasting using surrogate log-likelihood (MMH)

Input $\mathbf{y}, T_C, T_M, \boldsymbol{\theta}_{T_C}, \ell(\boldsymbol{\theta}_{T_C}|\mathbf{y}), \Sigma_{T_C}, \mu_\Omega, \Sigma_\Omega, p, \mathcal{F}$
Output $(\boldsymbol{\theta}_t)_{t=T_C}^{T_M+T_C}$

1. **Initialization**
   (a) Set $t := T_C$.
   (b) Set $\boldsymbol{\theta}_t := \boldsymbol{\theta}_{T_C}$ and $\Sigma_t := \Sigma_{T_C}$.
   (c) Set $\hat{\ell}(\boldsymbol{\theta}_t|\mathbf{y}_{1:T}) := \ell(\boldsymbol{\theta}_{T_C}|\mathbf{y}_{1:T})$.
2. **Proposal and classification**
   (a) Sample a proposal $\boldsymbol{\theta}^{(t+1)} \sim g_{\Sigma_t}(\boldsymbol{\theta}^*|\boldsymbol{\theta}_t)$.
   (b) Calculate the Mahalanobis distance of $\boldsymbol{\theta}^{(t+1)}$ (5.2)

$$D(\boldsymbol{\theta}^{(\boldsymbol{t+1})}) = (\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\mu}_{\boldsymbol{\Omega}})^T \boldsymbol{\Sigma}_\Omega^{-1} (\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\mu}).$$

   (c) Determine if $\boldsymbol{\theta}^{(\boldsymbol{t+1})} \in \Omega_p$ with confidence level p (5.3)

$$D(\boldsymbol{\theta}^{(t+1)}) \leq \chi_d^2(p).$$

3. **Log-likelihood estimation**
   (a) Calculate the associated surrogate log-likelihood

$$\hat{\ell}(\boldsymbol{\theta}^{(t+1)}|\mathbf{y}_{1:T}) := \begin{cases} \mathcal{F}(\boldsymbol{\theta}^{(t+1)}) & \text{if } \boldsymbol{\theta}^{(t+1)} \in \Omega_p, \\ \ell(\boldsymbol{\theta}^{(t+1)}|\mathbf{y}_{1:T}) & \text{otherwise.} \end{cases}$$

   (b) Calculate the logarithm of the acceptance rate α according to (2.9)

$$\log \alpha := \min\left(0, \ell(\boldsymbol{\theta}^{(t+1)}|\mathbf{y}_{1:T}) - \ell(\boldsymbol{\theta}_t|\mathbf{y}_{1:T}) + \log \pi_\Theta(\boldsymbol{\theta}^{(t+1)}) - \log \pi_\Theta(\boldsymbol{\theta}_t)\right).$$

4. **Acceptance or rejection**
   (a) Draw a sample $u \sim \mathcal{U}(0,1)$.
   (b) If $\log u \leq \log \alpha$ then accept $\boldsymbol{\theta}^{(t+1)}$
      i. Set $\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}^{(t+1)}$.
      ii. Set $\hat{\ell}(\boldsymbol{\theta}_{t+1}|\mathbf{y}_{1:T}) := \hat{\ell}(\boldsymbol{\theta}^{(t+1)}|\mathbf{y}_{1:T})$.
   (c) If $\log u > \log \alpha$ then reject $\boldsymbol{\theta}^{(t+1)}$
      i. Set $\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t$.
      ii. Set $\hat{\ell}(\boldsymbol{\theta}_{t+1}|\mathbf{y}_{1:T}) := \hat{\ell}(\boldsymbol{\theta}_t|\mathbf{y}_{1:T})$.
5. **Covariance**
   (a) Set $t := t + 1$.
   (b) Calculate Σ_t recursively according to (2.8).
6. **Next step**
   (a) If $t < T_M + T_C$ then return to step 2.
   (b) If $t = T_M + T_C$ then break.

Algorithm 10 Accelerated adaptive Metropolis-Hastings (AAMH)

Input: $\mathbf{y}_{1:T}, T_C, T_M, t_a, t_b, \boldsymbol{\theta}_0, \Sigma_0, p, \lambda$
Output: $(\boldsymbol{\theta}_k)_{k=0}^{T_M+T_C}$

1. **Initial classic AMH run**
   (a) Start an initial run of CMH, presented in Algorithm 8, for T_C iterations, with initial sample $\boldsymbol{\theta}_0$, covariance Σ_0 and adaptation start time t_a.
   (b) Save the proposals $(\boldsymbol{\theta}^{(k)})_{k=0}^{T_C}$ and the log-likelihoods $(\ell(\boldsymbol{\theta}^{(k)}|\mathbf{y}))_{k=0}^{T_C}$ in Ξ. Save also the chain of posterior samples $(\boldsymbol{\theta}_k)_{k=0}^{T_C}$ and the final proposal covariance Σ_{T_C}.
2. **Process data set**
   (a) Remove the burn-in period, $0, \ldots, t_b$, from the data set Ξ.
   (b) Remove proposals outside of Θ from Ξ.
   (c) Remove a percentage, λ, of proposals with the most negative log-likelihood values from Ξ.
3. **Train neural network**
   (a) Train a neural network $\mathcal{F}$ on the processed data set $\tilde{\Xi}$.
4. **Parameterise** Ω_p
   (a) Compute μ_Ω and Σ_Ω via the sample mean and covariance on the proposals in $\tilde{\Xi}$.
   (b) Define $\Omega_p = \{\boldsymbol{\theta} \in \mathbb{R}^d | D(\boldsymbol{\theta}) \leq \chi_d^2(p)\}$.
5. **Final mixed AMH run**
   (a) Start a final run of MMH, presented in Algorithm 9, for T_M iterations, starting at time T_C, with initial sample $\boldsymbol{\theta}_{T_C}$ and covariance Σ_{T_C}, using the surrogate log-likelihood defined by $\mathcal{F}$ and Ω_p.
   (b) Concatenate the sampled posterior points from the classic and mixed method into $(\boldsymbol{\theta}_k)_{k=0}^{T_M+T_C}$.

5.3 Performance metrics

To ensure that the algorithm produces satisfactory result we need to estimate its efficiency and accuracy. The efficiency is with regards to the number of samples produced per time second. However, not every sample can be considered independent in the output from MH. Thus, it's imperative to determine the *effective sample size*. The performance is assessed using the *Wasserstein distance*, also known as the *earth movers distance*, which measures the similarity between two distributions.

5.3.1 Effective sample size

The *effective sample size* (ESS) is defined as the number of samples that can be considered independent in a set of samples. The formula can vary, but is often defined as

$$\text{ESS} = \frac{n}{1 + 2\sum_{k=1}^{K} \rho(k)}, \tag{5.5}$$

where n is the total number of sample. The function ρ is defined as the autocorrelation of the samples from the posterior at time lag k. The upper limit K is defined at the point where the autocorrelation no longer exceeds the empirically derived value 0.05 [12]. If we divide the ESS by the sample size n, we get the ESS *ratio*.

5.3.2 Wasserstein distance

Assume that μ and ν are two probability measures on a so-called Polish metric space (χ, d). For technical details we refer to [25]. The *Wasserstein distance* (WD) of order p between μ and ν is then defined as

$$W_p(\mu, \nu) := \left(\inf_{\gamma \in \Pi(\mu,\nu)} \int_{\chi \times \chi} d(x, y)^p \mathrm{d}\pi(x, y) \right)^{\frac{1}{p}}, \tag{5.6}$$

where the collection $\Pi(\mu, \nu)$ is the set of all joint probability measures on $\chi \times \chi$ and the marginals are μ and ν respectively, see [25]. It measures the average distance that points from one distribution are moved in order to transform it into the other distribution, when considering the optimal movement scheme which minimises the distance d. For empirical distributions, the special case W_1 measure is often referred to as the *earth mover's distance*. This specific case can be defined as an optimisation problem, a special case of the optimal transport problem [26].

Consider that we have samples are P and Q, both of size n, from the distributions $\mathcal{P}$ and $\mathcal{Q}$, respectively. We wish to determine the optimal flows $f_{ij} \in [0, 1]$, from points $i = 1, \ldots, n$ to points $j = 1, \ldots, n$. The distance d_{ij} between points i and j are calculated beforehand. We consider the empirical distributions generated by each sample. Each point have the same probabilistic weight, which we set to $w_i = n^{-1}$ and $w_j = n^{-1}$ for all i and j. The optimization problem can then be expressed as

$$\begin{aligned}
\text{minimise} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}d_{ij} && && && \text{(5.7a)}\\
\text{subject to} \quad & f_{ij} \geq 0, && 1 \leq i \leq n, && 1 \leq j \leq n, && \text{(5.7b)}\\
& \sum_{j=1}^{n} f_{ij} \leq w_i, && 1 \leq i \leq n, && && \text{(5.7c)}\\
& \sum_{i=1}^{n} f_{ij} \leq w_j, && && 1 \leq j \leq n, && \text{(5.7d)}\\
& \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij} = 1. && && && \text{(5.7e)}
\end{aligned}$$

For the empirical distributions given by the sets P and Q, the Wasserstein distance can be calculated via the expression

$$\text{EMD}(P,Q) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij}d_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij}}, \tag{5.8}$$

where $f_{ij}, i, j = 1, \ldots, n$ are solutions to the optimization problem (5.7a)-(5.7e).

5.4 Experiment setup

In this section we present the setup that we use for our experiments. We wish to compare the results from using AAMH to the results from using only CMH. First, we want to check that our method generates a posterior distribution which is similar to the one produced by CMH. Secondly, we want to compare the computational efficiency of the two methods.

To check the accuracy of our method we use the Wasserstein distance. We estimate the WD from two posterior distributions produced using the CMH. This works as a benchmark for how much the posterior distribution varies inherently. Then, we estimate the WD between posterior distributions produced by CMH and MMH. Furthermore, we use the effective sample size to check that the sampling procedure of MMH is of the same quality as CMH. Again, this is done by calculating a benchmark ESS from CMH and comparing this to the ESS of MMH. To ensure that all calculations are done using the same conditions, we run them from the same start point where the initial CMH is sufficiently close to the stationary distribution. All of our experiments are run several times, in order to generate averages of the performance metrics. See Figure 5.4 for a visualisation of the experiments setup.

Once we have ensured that our method is accurate, we want to compare the computational efficiency of the two methods. To this end we measure the time that it takes for both the CMH and AAMH to produce a posterior sample of the same size. Moreover, we measure the time it takes to reach different ESS for each method, in order to see when the AAMH becomes preferable to only running CMH.

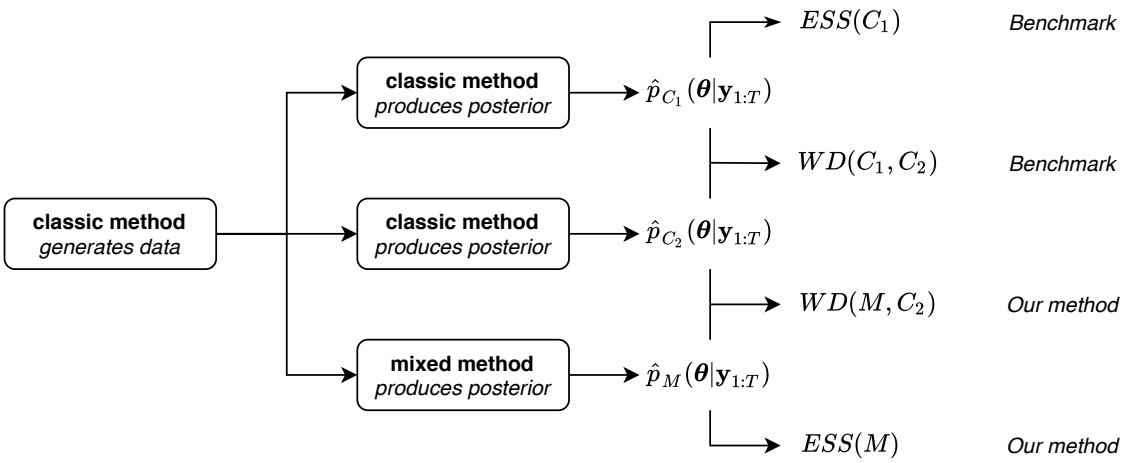


Figure 5.4: The experiments setup for performance assessment. First we run CMH until it has converged sufficiently close to the stationary distribution and generated training data for the neural network, which we train. Then the CMH is run twice from this point and the MMH once. The ESS values and WD values are calculated according to the graph to ensure that we get an assessment of both the efficiency and accuracy of our method as compared to a benchmark value.

These experiments are made for the LSM and CSM models, see sections 3.3 and 3.4. For the LSM model we use KF to calculate the log-likelihood, whereas for the CSM model we use both EKF and BF. The parameters that we vary are the state-space dimensionality N, the number of model parameters d, the confidence level p of the Mahalanobis classifier and the amount of training data.

6
Results

We begin by presenting exemplifying results of the accelerated adaptive Metropolis-Hastings algorithm (AAMH) using the Ornstein-Uhlenbeck model (OU) as a proof of concept. We then proceed by applying the method on the linear spring-mass model (LSM) where we test, among other things, how the state-space dimensionality affects the accuracy and efficiency. The same analysis is done for the cubic spring-mass model (CSM), where we utilise both the extended Kalman filter (EKF) and the bootstrap filter (BF).

6.1 Ornstein-Uhlenbeck model

Here we present the results of running the AAMH algorithm for the OU-model described in section 3.2. The purpose is to give a proof of concept for the method, for a well-known model. We first present the results from running the CMH algorithm, presented in Algorithm 8. The neural network is then trained on the pre-processed training data. Lastly, we proceed with running the MMH algorithm, presented in Algorithm 9.

6.1.1 The CMH algorithm

Consider an observation $\mathbf{y}_{1:T}$ of the OU-model, simulated using the model parameters presented in Table 6.1. For the posterior distribution we consider the parameters of interest $\boldsymbol{\theta} = (\gamma, \beta, \sigma)$. The prior distribution π_Θ is a truncated multivariate normal distribution, with its defining parameters presented in Table 6.2. The mean is chosen so that the prior is not centered around the true model parameters, and the covariance is chosen large enough for the prior to be considered uninformative.

The CMH algorithm is given an initial sample $\boldsymbol{\theta}_0$ close to the true model parameters $\tilde{\boldsymbol{\theta}} = (\tilde{\gamma}, \tilde{\beta}, \tilde{\sigma})$ and an initial narrow and diagonal proposal covariance Σ_0 before the adaptation begins. These two properties are likewise found in Table 6.2. We run the CMH algorithm for $T_C = 10^5$ iteration steps, where the log-likelihood values are estimated using the KF, and present the histogram and trajectory plots in Figure 6.1. We see that the chain reaches stationarity quickly, and that the method produces empirical posteriors centered around values close to the true ones.

Table 6.1: The model parameters used to simulate the observation $\mathbf{y}_{1:T}$. The parameters T and Δt denotes the number of time steps and their lengths in the Euler-Maruyama method (3.6). The value R is the standard deviation of the measurement noise in the model. We consider γ, β and σ to be unknown quantities.

parameters	$\tilde{\gamma}$	$\tilde{\beta}$	$\tilde{\sigma}$	T	Δt	R
values	2	0.1	0.1	100	1	0.1

Table 6.2: Parameters used for the CMH algorithm on the OU-model. The region Θ is the support of the prior distribution π_Θ. The parameters Σ_π and $\boldsymbol{\mu}_\pi$ denotes the covariance and the mean of the prior distribution. The parameters Σ_0 and $\boldsymbol{\theta}_0$ denotes the initial proposal function covariance and the initial posterior sample.

parameters	Θ	$\boldsymbol{\mu}_\pi$	Σ_π	$\boldsymbol{\theta}_0$	Σ_0
values	$(-\infty, \infty) \times [0, \infty)^2$	$10 \cdot \tilde{\boldsymbol{\theta}}$	$10^3 \cdot \text{diag}(\tilde{\boldsymbol{\theta}})$	$2 \cdot \tilde{\boldsymbol{\theta}}$	$10^{-1} \cdot \text{diag}(\tilde{\boldsymbol{\theta}})$

6.1.2 Data pre-processing and training the neural network

Since the stationary distribution is reached quickly, we have a lot of high quality data at our disposal, stored in Ξ. However, we make a generous burn-in cutoff at $t_b = 10^4$ iteration points. We also remove all negative σ values as they are not within the support Θ. In Figure 6.2 we can see that the proposal parameters and log-likelihood values are narrowed down greatly. We now have a training data set $\tilde{\Xi}$ consisting of the proposal parameter and log-likelihood pairs.

We proceed to parameterize the region $\tilde{\Xi}$, producing the region Ω_p. We set the confidence level to to $p = 0.999$. Since we have a lot of training data points, the data is of such good quality that we may set such a high confidence level, without risk of diverging. We define the neural network differently for this model than the one specified by Figure 5.2, instead we present the properties in Table 6.3. The neural network is trained on $\tilde{\Xi}$.

6.1.3 The MMH algorithm

We now run the MMH algorithm. The algorithm is run for a subsequent $T_M = 10^5$ iterations. The run statistics for both the CMH algorithm and the MMH algorithm (which make up the AAMH algorithm) are presented in Table 6.4. We see that the rate of accepted proposals in both methods are similar, and that the mean posterior values align well. Both methods perform almost as fast in this one-dimensional case.

The comparative results of the initial $T_C = 10^5$ iterations using the CMH algorithm and the subsequent $T_M = 10^5$ iterations using the MMH algorithm are presented in Figure 6.3. We see that the empirical posterior distributions align very well. However, this is only the case in a one-dimensional system. Next, we will test higher-dimensional models. However, before exploring the more complex models we perform a hyperparameter optimization for our neural network.

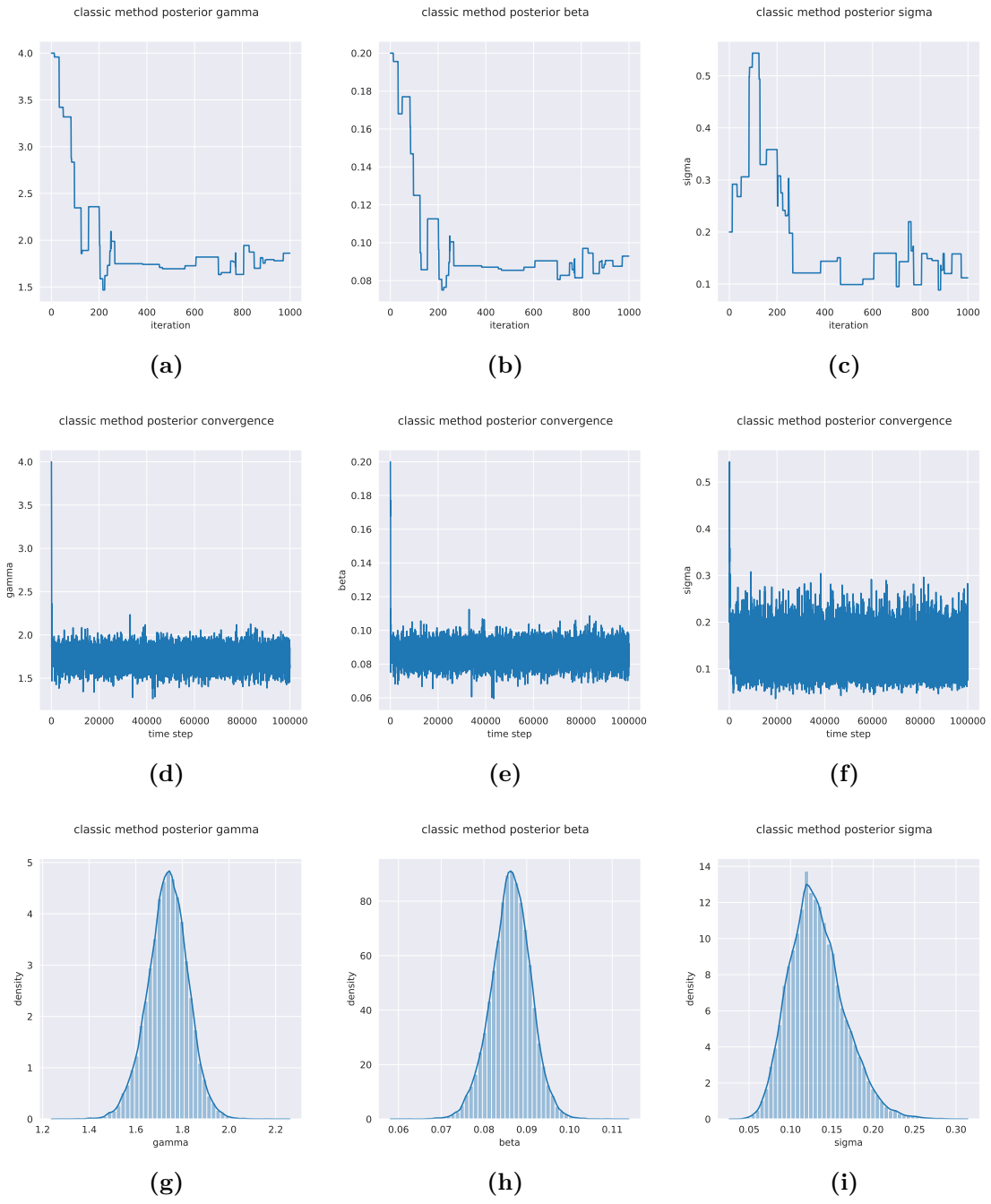


Figure 6.1: The results of using the CMH algorithm on the OU-model. The marginal values of the parameters γ, β, and σ are plotted separately. In Figures 6.1a, 6.1b and 6.1c, we see the trajectory of the first 10^3 sampled posterior points. In Figures 6.1d, 6.1e and 6.1f, we see the corresponding plots for all 10^5 posterior points. In Figures 6.1g, 6.1h and 6.1i, we see the marginal posterior distributions of all $T_C = 10^5$ posterior points.

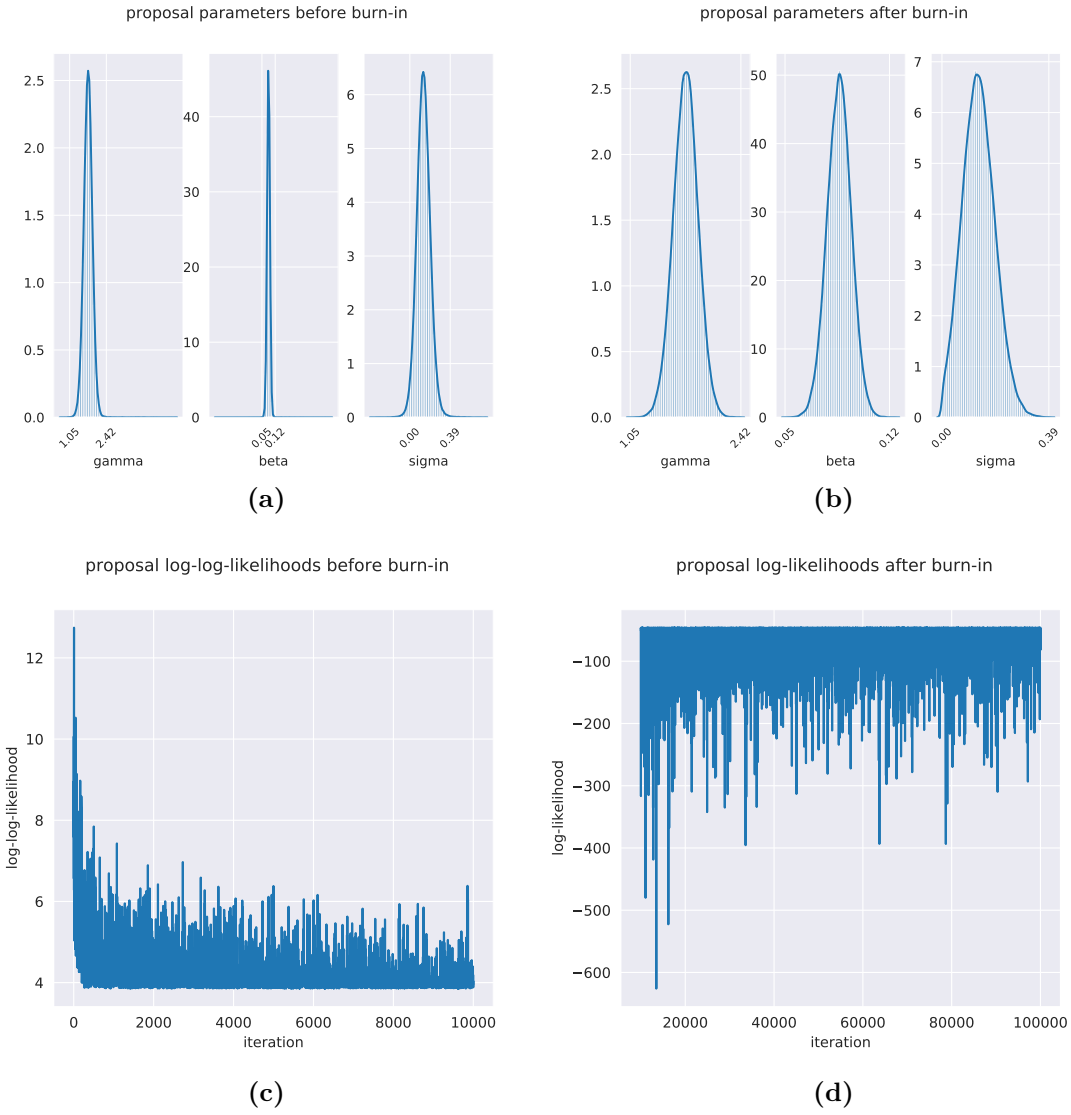


Figure 6.2: The training data before and after pre-processing on the OU-model. In Figures 6.2a and 6.2b we see the marginal distributions of the proposed parameters in the training data sets Ξ and $\tilde{\Xi}$ respectively. In Figure 6.2c we see the *logarithm* of the *absolute* log-likelihood values in Ξ. In Figure 6.2d we see the log-likelihood values in $\tilde{\Xi}$.

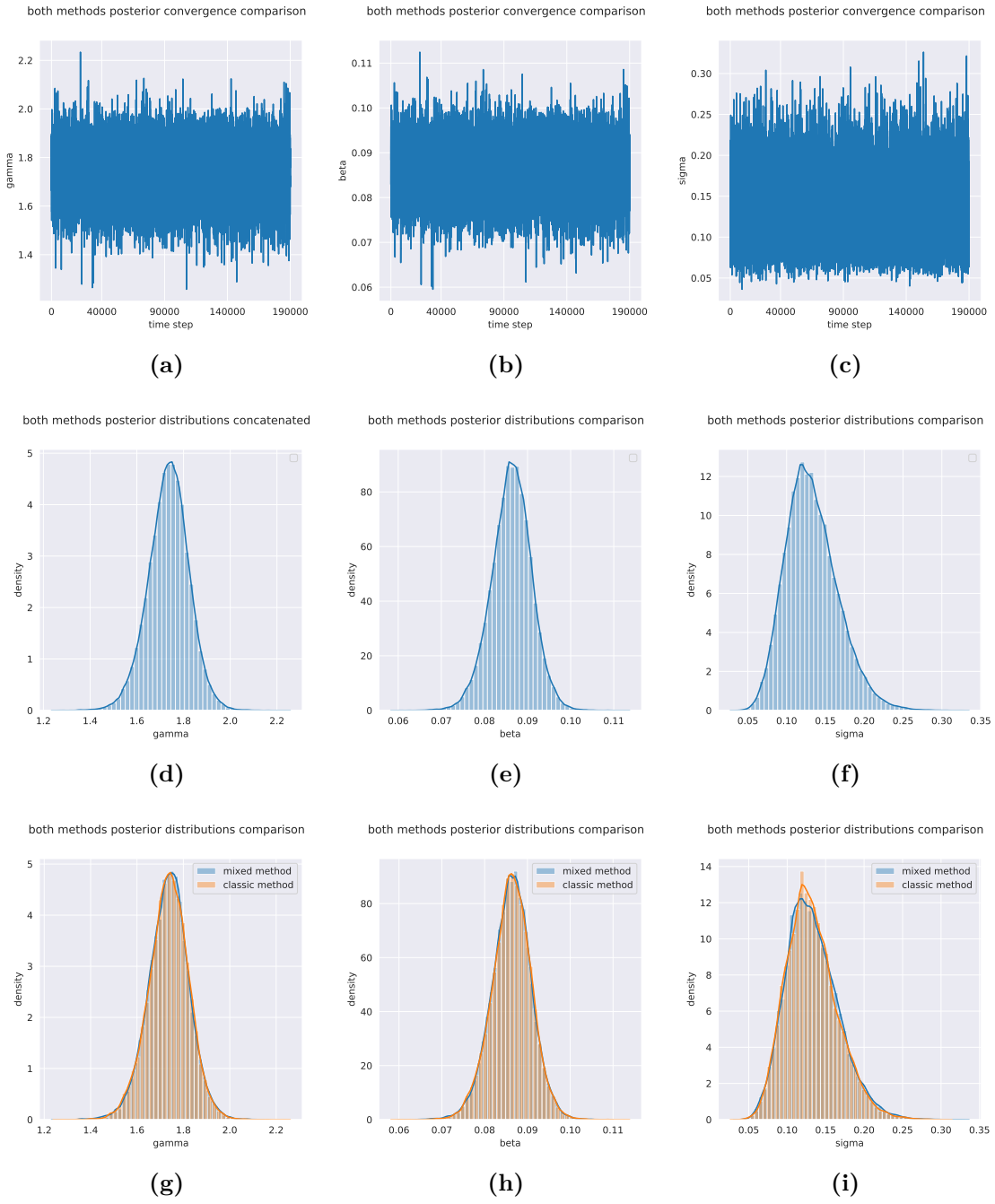


Figure 6.3: The results of the AAMH algorithm on the OU-model. In Figures 6.3a, 6.3b and 6.3c, we see the sampled posterior points from the CMH algorithm concatenated with the points from the MMH algorithm. Note that the burn-in phase is excluded. The concatenated posterior distributions are plotted in Figures 6.3d, 6.3e and 6.3f. In Figures 6.3g, 6.3h and 6.3i, the posterior distribution of the points from the CMH algorithm are plotted above the distribution of the points from the MMH algorithm.

Table 6.3: Parameters for the neural network for the OU-model. The values n_0, n_1, n_2, and n_3 are the sizes of each layer. The parameters β_1, β_2 and η are the moment scaling parameters and the learning rate respectively, used by the Adam optimiser. These values were selected from empirical testing.

properties	n_0	n_1	n_2	n_3	β_1	β_2	η	epochs
values	3	100	100	1	0.999	0.999	0.0005	100.000

Table 6.4: The run statistics of the AAMH algorithm for the OU-model.

Algorithm	acceptance rate	mean posterior values	time per sample
CMH	0.2464	$(1.74, 0.0864, 0.133)$	2.908 ms per sample
MMH	0.2800	$(1.74, 0.0862, 0.133)$	2.883 ms per sample

6.2 Hyperparameter optimisation

To ensure that the training of the neural network always performs well and does not get stuck at local optima we wish to optimise the hyperparameter choices.

We run the hyperparameter optimization for the LSM-model for the parameters $(\beta_1, \beta_2, \eta) \in \mathcal{X}$ in the Adam optimiser, see Section 4.5. We define the prior distribution of β_1 and β_2 as uniform distributions between 0.9 and 1, and between 0.99 and 1 respectively. The prior distribution of the learning rate η is defined as a log-uniform distribution between $\log(10^{-5})$ and $\log(10^{-1})$.

The TPE algorithm is iterated for 10^4 steps while defining the fitness function f as the negative loss function of the neural network. The training data $\tilde{\Xi}$ is constituted of an initial run using the CMH algorithm with the parameters in Table 6.5. We set the number of epochs to a constant 10^4, and the state-space dimension as $N = 10$. The optimal values derived are $\beta_1 = 0.990$, $\beta_2 = 0.996$ and $\eta = 5.406 \cdot 10^{-4}$. See Appendix A.5 for a presentation of the hyperparameter optimisation results.

6.3 Linear spring-mass model

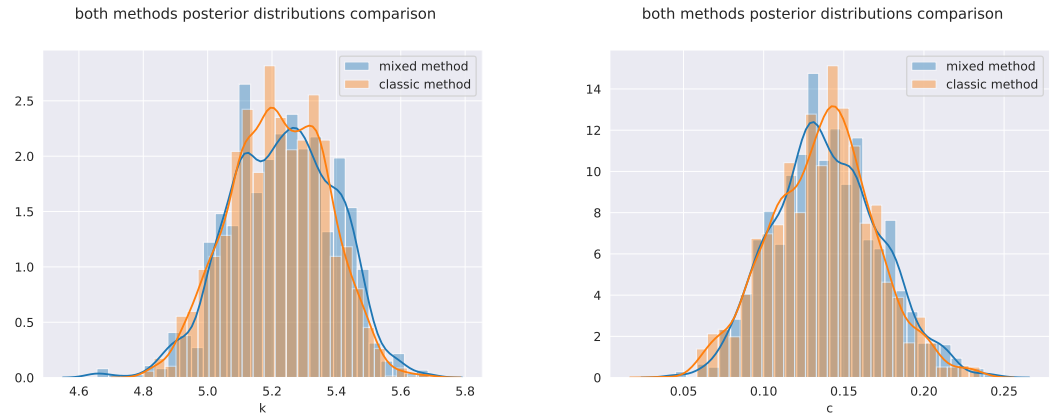
We wish to examine the performance of the AAMH algorithm when considering a linear model, wherein the log-likelihood estimation using the KF is replaced by our surrogate log-likelihood. The primary interest is to see how the efficiency and accuracy is affected by the state-space dimensionality. We also examine the effect of varying the training data size, the classification confidence level, and the number of parameters considered for the posterior distribution.

For the linear model, we consider the LSM-model presented in Section 3.3. During these tests, we consider only the stiffness and damping constants k and c, while setting the masses to constant values $m = 1$. We also consider all stiffness constants to be equal as well as all damping constants. See Table 6.5 for the model parameters $\tilde{\boldsymbol{\theta}} = (\tilde{k}, \tilde{c})$ considered, as well as the other parameters of the observation.

The prior π_Θ is defined on $\Theta := (0, \infty)^2$ and is a truncated multivariate normal distribution. To ensure fast convergence, we allow the initial guesses $\boldsymbol{\theta}_0$ to the CMH algorithm to be very close to the true model parameters $\tilde{\boldsymbol{\theta}}$, and the covariance

Table 6.5: The model parameters used in the LSM-model. The parameters k and c are the stiffness and damping coefficients, and m is the mass. The parameters σ_1 and σ_2 refer to the system noise exerted on the positions and velocities, respectively. The parameter R is the measurement noise. N is the state-space dimensionality.

parameters	k	c	m	T	ΔT		$\sigma_1\sigma_2$	R	N
values	5	0.1	1	100	0.01	0	0.1	1	6



(a) **(b)**

Figure 6.4: Results of the AAMH algorithm on the LSM-model. Figures 6.4a and 6.4b compares the results of the CMH algorithm with $T_C = 1000$ and the MMH algorithm with $T_M = 1000$. The model parameters are presented in Table 6.5.

matrix for the proposal function Σ_0 to be very narrow. In Figure 6.4 we see an exemplified run of the whole procedure described in Algorithm 10. The estimated average effective sample ratios of parameters k and c are 1.16 and 1.13. Since these values are similar, we need only consider one parameter as a representative for both ratios, namely parameter k. However, the Wasserstein distances (WDs) are still calculated while considering both parameters.

6.3.1 Varying the training data sizes

We have demonstrated that the AAMH algorithm can produce satisfactory results for the linear model. However, we wish to investigate how many training data points are required by the neural network to produce good results.

For the LSM-model with state-space dimension $N = 10$, we run the CMH algorithm for $T_C = 1000$ iterations, while considering the model parameters in Table 6.5. The burn-in period is set to $t_b = 800$, and the size of the training data is varied from $K = 15$ to $K = 200$. The training data is randomly selected without replacement from $\tilde{\Xi}$. Then the MMH algorithm is run with confidence level $p = 0.9999$, for a subsequent $T_M = 1000$ number of points.

The average WD and effective sample ratio, over 10 runs, associated to the CMH and the MMH algorithms are calculated according to the experiment setup presented in Figure 5.4. The results are presented in Figure 6.5.

(a) (b)

Figure 6.5: The performance metrics over training data sizes. Figure 6.5a illustrates the *ratio* of effective samples per iteration against the training data size for the MMH. Figure 6.5b likewise illustrates the WD against the training data size for the MMH. The shaded regions are the standard deviations of the metrics. In both figures, the benchmark CMH algorithm metrics are also plotted.

6.3.2 Varying the Mahalanobis confidence levels

The MMH algorithm performs increasingly well when there is more training data available. However, in cases where there is insufficient data available, the neural network might make valid log-likelihood approximations only in small subsets of Ω_p.

We wish to assess how the Mahalanobis confidence level p can be utilised to vary the certainty of the neural network in its defined region. We set a small training size $K = 50$ and vary the confidence levels between $p = 0, 1 - 10^{-1}, \ldots, 1 - 10^{-40}$. In Figure 6.6 we present the ESS ratio and the WD as functions of the confidence levels.

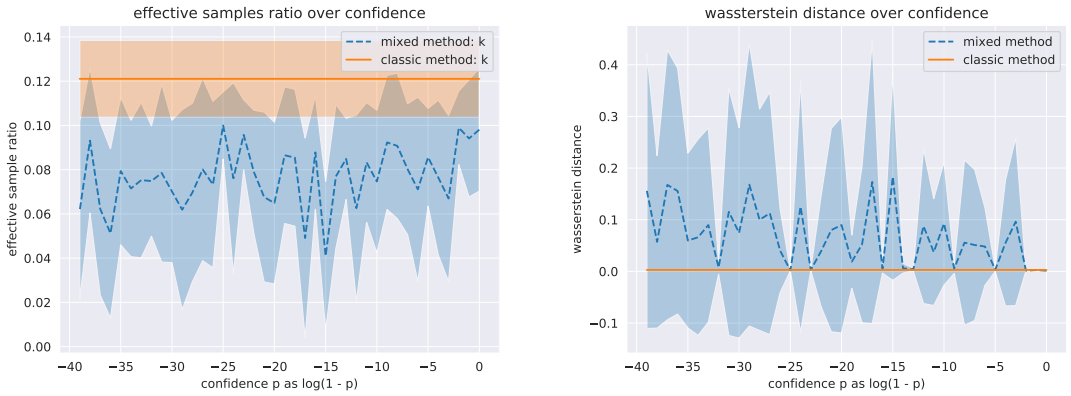
6.3.3 Varying the state-space dimensionality

We previously demonstrated that the MMH algorithm produces similar performance metrics as the CMH algorithm both with regards to accuracy and efficiency.

However, we wish to examine how well the performance is preserved when the state-space dimensionality is increased. We run the experiment described in Figure 5.4 while varying the state-space dimension as $N = 2, 4, \ldots, 20$. Figure 6.7a shows the ESS ratio as a function of the dimensionality, and Figure 6.7b shows the WD.

We also assess how the overall computational times differ based on the state-space dimensionality. In Figure 6.7c are the computational times of different parts of the program as a function of the dimensionality. We compare a full run of CMH with $T_C = 3000$ to a run of AAMH, which includes an initial CMH run with $T_C = 1000$, training of neural network and a final MMH run with $T_M = 2000$. The burn-in period is $t_b = 900$.

We see that the AAMH is more efficient than only running CMH for large runs. However, the AAMH algorithm is initially less efficient due to the required training



(a) **(b)**

Figure 6.6: The performance metrics when varying the confidence levels. In Figure 6.6a, we see the ESS ratio as a function of $\log(1-p)$. In Figure 6.6b, we see the WD against $\log(1-p)$. The confidence levels p span logarithmically from $p = 1 - 1^{-40}$ to $p = 0$.

Table 6.6: The parameter values which we use in the scheme where we vary the number of considered model parameters in the LSM-model.

k_1	k_2	k_3	k_4	c_1	c_2	c_3	c_4
5	7.5	10	12.5	0.1	0.2	0.3	0.4

period for the neural network. We consider the conditions under which the AAMH algorithm performs more efficiently than simply using the CMH algorithm. The computational times of the two methods are calculated as functions of ESS, for varying state-space dimensionalities. We denote the point at which the ESS per second is overtaken by the AAMH as the *point of indifference*. Again, we have that the CMH algorithm is run for $T_C = 3000$ iterations while the AAMH algorithm is run for $T_C = 1000$ and $T_M = 2000$ iterations, with $t_b = 900$.

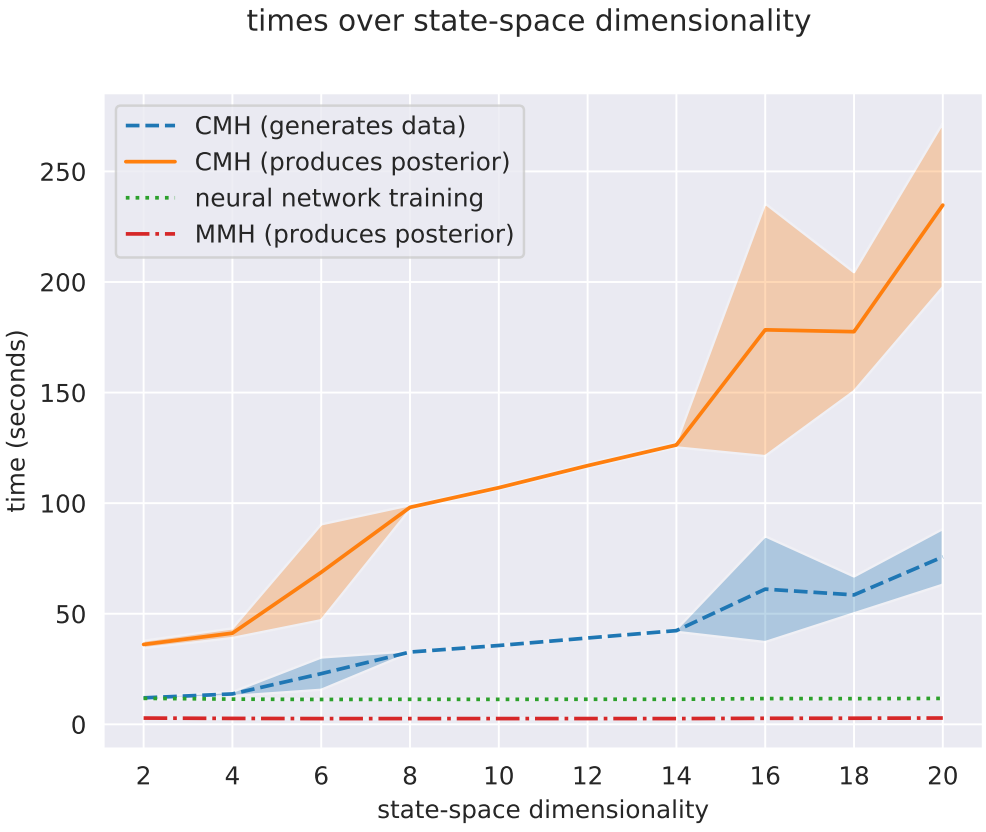
The test is performed 10 times per each considered state-space dimension and we present a selection of these graphs in Figures 6.8a and 6.8b. Furthemore, we plot the points of indifference against the state-space dimensionality in Figure 6.8c.

6.3.4 Varying the number of model parameters

We have so far only considered a system with only two model parameters, namely k and c. However, we wish to see how the AAMH algorithm performs when we increase the number of considered parameters $\boldsymbol{\theta}$ to our posterior distribution $p(\boldsymbol{\theta}|\mathbf{y})$.

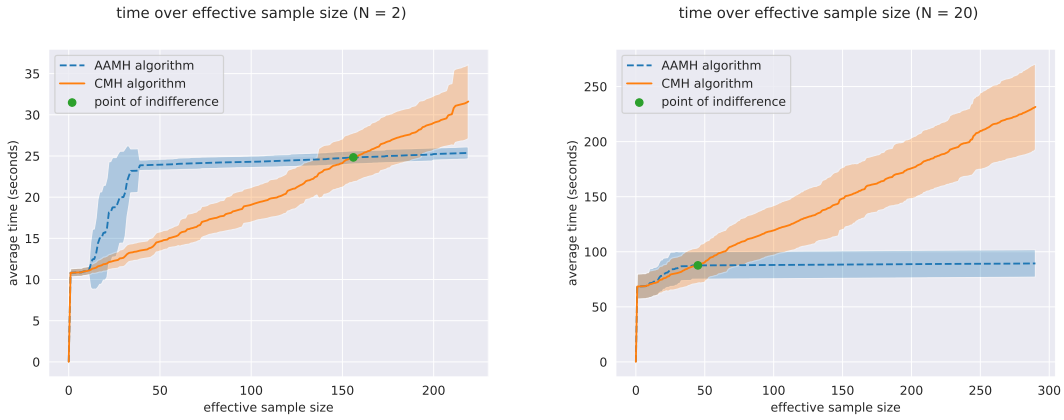
Begin by considering the linear model with three masses in total, which means that $\mathbf{y}_{1:T} \in \mathbb{R}^{6\times T}$. This implies that the system has stiffness constants k_1, k_2, k_3, k_4 and damping constants c_1, c_2, c_3, c_4. The number of considered parameters are varied between 2, 4, 6 and 8. See Table 6.7 for a scheme of how the considered parameters are used to define the system. The values considered are presented in Table 6.6.

(a) **(b)**

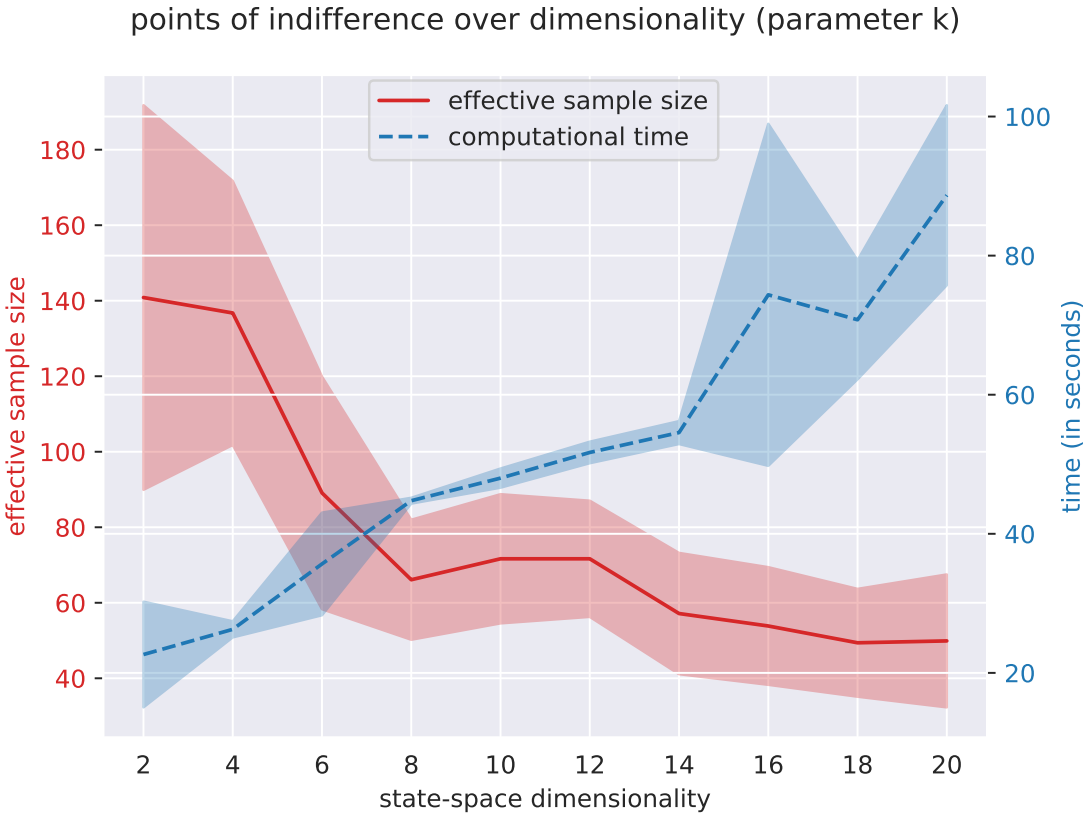


(c)

Figure 6.7: In Figure 6.7a, the ESS ratio over state-space dimensionality is plotted. In Figure 6.7b, the WDs between the posteriors using the CMH and MMH algorithms over state-space dimension are plotted. In Figure 6.7c, we see the total computational time for different parts against the dimensionality. The time for a full CMH run with $T_C = 3000$ is shown in dashed blue line. The time for generating training data via CMH with $T_C = 1000$ is shown in solid orange line. The time to train the neural network is shown in dotted green line and the time for running the final MMH with $T_M = 2000$ is shown in dash-dot line. To compare the full AAMH run to the full CMH run, add together the latter three graphs.



(a) (b)

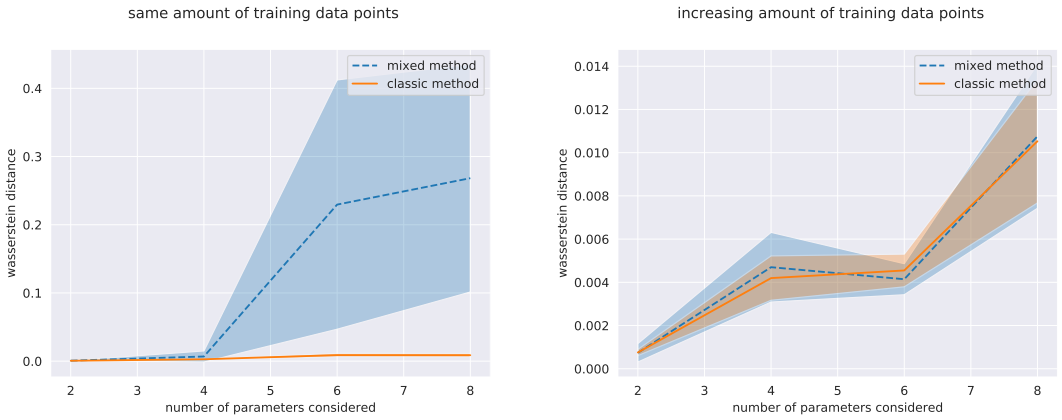


(c)

Figure 6.8: In Figures 6.8a and 6.8b, we see the points of indifference plotted for varying state-space dimensions. The CMH algorithm spends some time in the burn-in phase (producing no effective samples) and then increases linearly. The AAMH algorithm has a training phase after the burn-in phase and then has a linear increase in time taken per effective sample, but with a lower slope. In Figure 6.8c we see the computational times and ESS at the points of indifference are plotted for varying state-space dimensionality.

Table 6.7: The different parameter setups when increasing the number of considered parameters. Note that we wish to preserve as much symmetry as possible.

position	1	2	3	4
2 parameters	k_1, c_1	k_1, c_1	k_1, c_1	k_1, c_1
4 parameters	k_1, c_1	k_1, c_2	k_2, c_1	k_2, c_2
6 parameters	k_1, c_1	k_2, c_3	k_3, c_2	k_1, c_1
8 parameters	k_1, c_1	k_2, c_2	k_3, c_3	k_4, c_4



(a) **(b)**

Figure 6.9: The results of the AAMH algorithm with an increase in the considered model parameters for the posterior distribution. In Figure 6.9a, the neural network uses 100 training data points. In Figure 6.9b, the neural network uses an increasing number of points to train on, namely 100, 500, 5000 and 9000 training data points.

First, we run the experiment with $T_C = 1000$, $t_b = 900$ and $T_M = 2000$ while training on the same amount of 100 data points. The resulting WDs are presented in Figure 6.9a, where it is visible that the algorithms performs worse for more considered parameters. Since the domain of the log-likelihood function grows in dimension by the number of considered parameters, the neural network will require more data points to approximate it.

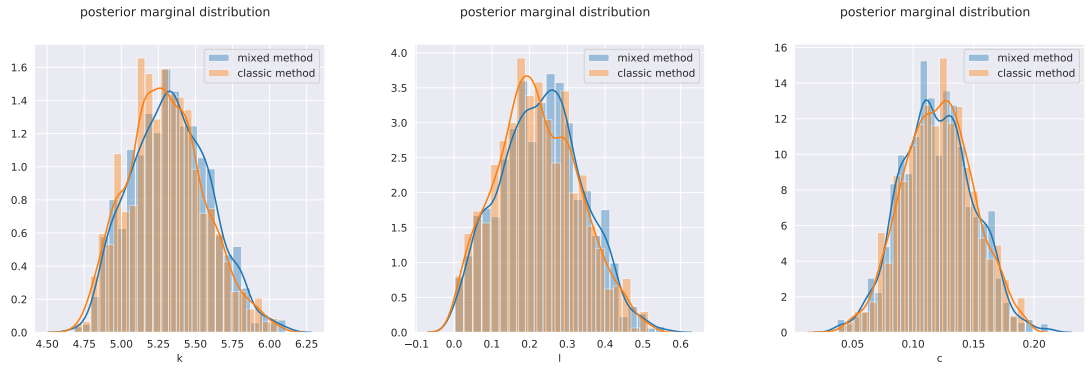
Thus, we proceed by running the experiment for $T_C = 10.000$ while considering $t_b = 9900, 9500, 5000, 1000$. This means that the neural network trains on $K = 100, 500, 5000, 9000$ points respectively for 2, 4, 6 and 8 parameters. The results are presented in Figure 6.9b.

6.4 Cubic spring-mass model

We wish to see how the AAMH algorithm performs when considering a nonlinear model. Consider the cubic model presented in Section 3.4. The log-likelihood must be estimated using either the EKF or the BF. We begin by using the EKF and assess the performance when increasing the state-space dimension. Thereafter, we proceed by doing the same test while considering the BF.

Table 6.8: The model parameters considered for the CSM-model. The model parameters are k, l and c. The parameter m is the mass. The parameters T and Δt are the step length and step size respectively. The parameters σ_1 and σ_2 are the system noise exerted on the positions and velocities, respectively. The parameter R is the measurement noise and N is the state-space dimensionality.

parameters	k	l	c	m	T	Δt	σ_1	σ_2	R	N
values	5	0.1	0.1	1	100	0.01	0	0.01	1	6



(a) **(b)** **(c)**

Figure 6.10: The results from the AAMH model on the CSM-model with the EKF. In Figures 6.10a, 6.10b and 6.10c, we see the comparative posterior distributions of both methods. Both methods produce very similar looking posterior distributions.

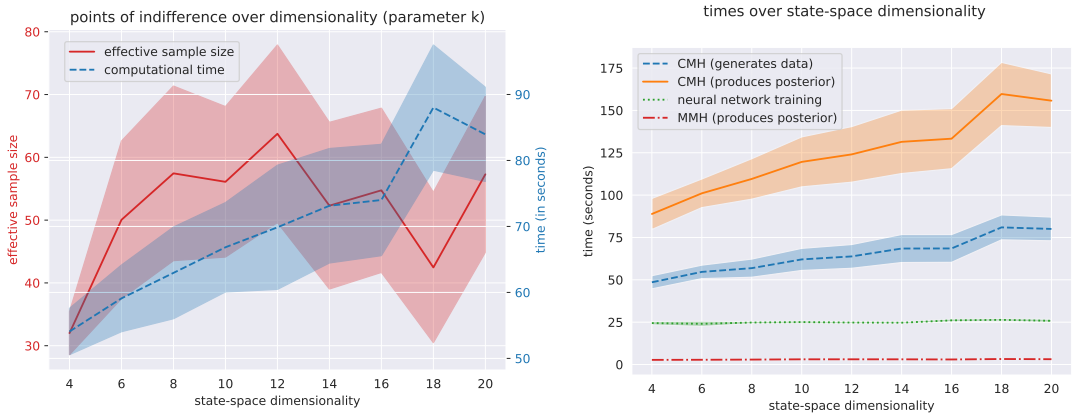
6.4.1 Using the extended Kalman filter

In light of the same issues as in the LSM, we consider a constant $m = 1$ for all masses. Consider the model parameters $\tilde{\boldsymbol{\theta}} = (\tilde{k}, \tilde{l}, \tilde{c})$ presented in Table 6.8. We run the whole AAMH procedure with $T_C = 1000$, $t_b = 900$ and thus $K = 100$ along with $T_M = 2000$, presenting the results in Figure 6.10. We see that the posterior distributions align well, while noting that there is an inherent deviation from the true parameter value of $\tilde{l} = 0.1$.

Consider the tests presented in Section 6.3.3. We run the AAMH algorithm while varying the state-space dimension from $N = 4, 6, \ldots, 18, 20$ and calculate the points of indifference at which it outperforms only using CMH. As previously mentioned, we use $T_C = 1000$, $t_b = 900$ and $T_M = 2000$ for the AAMH. The points of indifference as a function of the dimensionality is plotted in Figure 6.11, as well as the total computational times of the different parts of AAMH. Figure 6.12a presents the WD against the dimensionality using the EKF for the CMH.

6.4.2 Using the bootstrap filter

We proceed to consider the BF for estimating the log-likelihood values in the CSM model. We consider the same setup as in the previous section, but we need a plan for increasing the number of particles used as a function of the state-space dimension.



(a) **(b)**

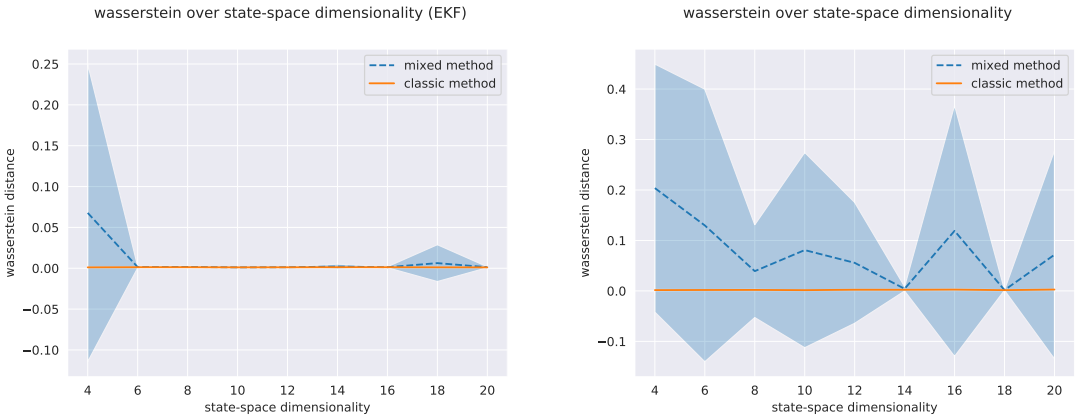
Figure 6.11: The efficiency results of the AAMH algorithm with the EKF for the CSM model. In Figure 6.11a we see the computational times and ESS at the points of indifference for varying state-space dimensionalities. In Figure 6.11b, we see the computational times of the CMH algorithm compared to the parts of the MMH algorithm.

Table 6.9: For 20 number of y and 20 number of evaluations of $\ell_p(\tilde{\boldsymbol{\theta}}|\mathbf{y})$ for the standard deviation estimation using particle filter with p number of particles. An optimal number of particles is accepted when $\hat{\sigma} < 1$.

dimension	4	6	8	10	12	14	16	18	20
particles	2	3	6	12	19	23	33	38	45

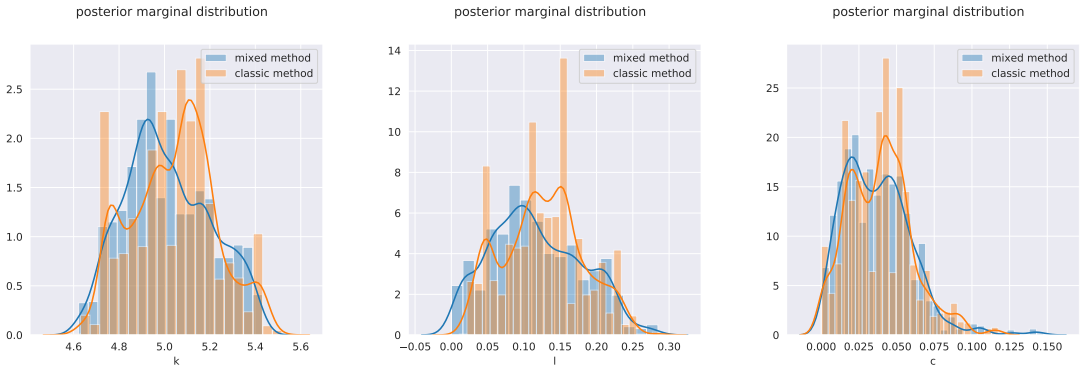
In Section 2.3.5 we present a scheme for selecting the number of particles by checking for which number the standard deviation of the estimated log-likelihood is less than at least 1.5. For our test, we choose the number of particles so that the standard deviation is less than 1. The standard deviation is computed over 20 log-likelihood evaluation of the same observation. This is done for the state-space dimenionalities $N = 4, 6, \ldots, 18, 20$. This procedure is in turn done for 20 different simulated observations $\mathbf{y}_{1:T}$ from the CSM model, to get an average number of particles needed. The estimated number of particles needed for each dimensionality is presented in Table 6.9.

We run the same tests as for the EKF, using the BF instead. The same model parameters as the ones presented in Table 6.8 are used. In Figure 6.13, we present the posterior distribution for the case $N = 20$, both with the classic method and the mixed method. In Figure 6.14 we present the points of indifference, and the total times for the AAMH algorithm. In Figure 6.12b the WD is plotted.



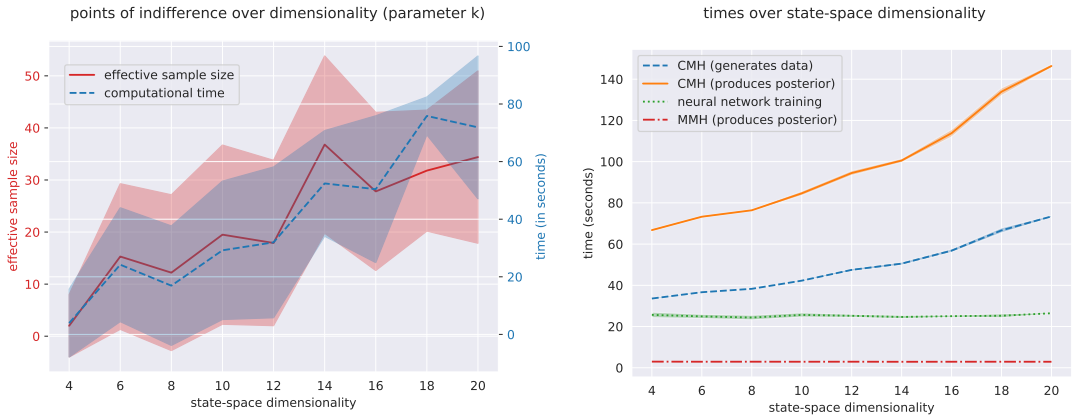
(a) **(b)**

Figure 6.12: The WD for the EKF and the BF over state-space dimensionality, plotted in figures 6.12a and 6.12b respectively.



(a) **(b)** **(c)**

Figure 6.13: The results from the AAMH model on the CSM-model with the BF. In Figures 6.13a, 6.13b and 6.13c, we see the comparative posterior distributions of both methods. Both methods produce similar looking posterior distributions, but of lesser quality than those produced by the EKF in Figure 6.10.



(a) **(b)**

Figure 6.14: The point of indifference results for the AAMH algorithm with BF for the CSM model. In Figure 6.14a, the computational times and ESS at the points of indifference are plotted for varying state-space dimensionalities. In Figure 6.11b, we see the computational times for the full CMH run and for the parts of the AAMH run.

7
Discussion

In this chapter we discuss our results. First, we briefly mention the results concerning the Ornstein-Uhlenbeck model. Then we discuss the tests on the LSM model, where we test both the applicability and limits of our method. Lastly, we proceed with discussing the results of the CSM model, where we utilise both the EKF and BF.

7.1 Ornstein-Uhlenbeck model

In Figure 6.3, we illustrate the preliminary results where a neural network driven surrogate log-likelihood can be used to produce a satisfactory posterior distribution on a simple one-dimensional model without considering the efficiency. It is clear from this figure that the empirical posterior produced by the MMH is qualitatively very similar to that of the CMH.

In Table 6.4 we see that the MMH algorithm performs the log-likelihood calculations almost as slowly as the CMH algorithm. This is because the KF is very fast for one-dimensional systems. Thus, the neural network can't compete.

7.2 Linear spring-mass model

We then proceed to the case of the LSM model, which allows for any even state-space dimensionality. First, we show, in Section 6.3.1, that the quality of the sampling is sensitive to the quantity of training data to the neural network. However, Figure 6.5 shows that that once a sufficient amount of data is obtained, the algorithm works in a satisfactory manner. The amount of data required to get good results is surprisingly small. The method then preserves both the ESS ratio, as well as produces a posterior distribution within a reasonable Wasserstein distance to those produced by simply using the KF.

We also explore, in Section 6.3.2, the case when the amount of training data is scarce, and we want some measure of how good the neural network is at estimating the log-likelihood. In Figure 6.6 we show that the resulting posterior distributions become increasingly better the more we let the KF calculate the log-likelihood. This suggest that the less data we have, the more conservative we need to be with the choice of the confidence level p in order to maintain accurate results. This leads to a more computationally heavy MMH, but in return the initial CMH for generating the training data is shorter.

Figure 6.4 shows qualitatively that the method works well for the LSM model. Quantitaviely, we show in Section 6.3.3 that the posterior produced by the AAMH

has comparable performance metrics to that of just running CMH. Additionally, this accuracy is consistent as the dimensionality is increased.

Moreover, it is clear from Figure 6.7c that the AAMH produces a large posterior sample at a much shorter time than only running the CMH. We see that the bulk of the computational time of the AAMH consist of generating the training data. This is also the only part that has a non-neglible increase in computational time with increasing dimensionality. It is not clear at first how the computational time associated with generating the training data would affect the overall efficiency of the method. Therefore, we test how the efficiency is affected by the state-space dimensionality. The result presented in Figure 6.8 gives the indication that the computational time until the point of indifference is reached increases with the state-space dimensionality. This is not surprising since the computational time taken by the initial CMH has a polynomial increase (since it utilises the KF). However, the results suggests that the point of indifference occurs at gradually lower number of effective samples as the state-space dimensionality increases.

Finally, in Section 6.3.4, we see that the amount of data required is dramatically increased by the number of considered parameters for the posterior distribution, as shown by Figure 6.9. This is reasonable, since in order to span any space with an increasing dimensionality, while preserving the same frequency of points, the amount of points required increases exponentially.

7.3 Cubic spring-mass model

For the nonlinear CSM model, the same tests are done with both the EKF and the BF. Comparing Figures 6.10 and 6.13, we see that the AAMH produces good results using both the EKF and the BF. It should be noted that the dimensionality in Figure 6.10 is 6, whereas it is 20 in Figure 6.13, which might affect the difference in performance. Moreover, Figure 6.12a show that AAMH with the EKF has very good performance metrics.

Looking at Figures 6.11a and 6.14a we see similar trends. Both the time and ESS of the indifference points increase with the dimensionality. This differs from the LSM model, where the ESS of the indifference point decreased. Whether we use the EKF or BF, it is clear from Figures 6.11b and 6.14b that the AAMH is preferable to only the CMH for large runs.

Important to note is that the mixed method which is trained on data generated by the BF as opposed to the EKF produces systematically worse posterior distributions (compare Figures 6.12a and 6.12b). This could be due to a number of factors, but the number of particles considered for each state-space dimensionality might not have been high enough.

In Table 6.9 we present the estimated optimal number of particles to be used by the BF. The number of needed particles for each dimensionality is surprisingly low. However, the number of calculations done to estimate the standard deviation may be too few, thus causing us to underestimate the number of required particles. It is still interesting to note that our method produces serviceable results using the BF, even when using very few particles, as the log-likelihoods used for the training are stochastic in this case.

8

Conclusion

Our goal of with thesis is to develop a deep-learning-accelerated Metropolis-Hastings algorithm which works for Bayesian estimation for state-space models. The purpose is to combat the increase in time complexity brought by an increase in state-space dimensionality. Traditional log-likelihood estimators such as the KF, the EKF and the BF all scale badly with the dimensionality. It is also important that the output of our method maintains the same quality as that of the traditional method.

Our results show that both the effective sample size and the shape of the posterior distribution can be preserved with our proposed algorithm, the AAMH. We have demonstrated this for both a linear and a non-linear model. Moreover, the performance is consistent for varying state-space dimensionality. Most importantly, our method is much more computationally efficient than the traditional method when considering larger sample sizes. This gain in computational time only increases with increasing state-space dimensionality.

The number of experiments and the size of experiments were limited by time constraints. Given more time, we would have liked to push our method even further into higher dimensions. The trend says that the initial runs would take longer, but the gain in efficiency once the neural network has been trained would increase.

In future work, to fortify our results it is imperative to check how much our method actually improves on the information contained in the training data. Moreover, it would be interesting to try to recreate our results for more complex models, specifically models that require particle filters. The CSM model could namely be filtered using only the EKF. It would also be of interest to investigate models where the posterior distribution is inherently multimodal, which means that the algorithm would visit many different regions. To solve for the problem of the algorithm falling outside of Ω as it tries to visit another mode, the neural network could perhaps be retrained on a set of new proposals. This could potentially increase the performance even further. Another aspect of this idea is that it could help if the original training set is not particularly coincident with the stationary distribution, which can happen if the burn-in or the number of iterations in the initial run is too short.

Finally, the network we use considers only one observation. If a network could be trained to predict the log-likelihood of an arbitrary observation and parameter choice, that would be very beneficial. In that case the input to the network would consist of long sequential data, possibly with variable length, and parameters. This would necessitate the use of other network architectures, such as a recurrent neural network, that could utilize the temporal nature of the data. As the networks would need to be larger, more training data would be needed. Hopefully, this would be offset by the fact that the networks would generalize to new observations.

Bibliography

[1] Simo Särkkä. *Bayesian Filtering and Smoothing*. USA: Cambridge University Press, 2013.

[2] Yaakov Bar-Shalom, X.-Rong Li, and Thia Kirubarajan. *Estimation with Applications to Tracking and Navigation: Theory, Algorithms and Software*. Jan. 2004.

[3] Andrew Golightly and Darren Wilkinson. "Bayesian parameter inference for stochastic biochemical network models using particle Markov chain Monte Carlo". In: *Interface focus* 1 (Dec. 2011), pp. 807–20.

[4] Yong Zeng and Shu Wu. *State-space models: Applications in economics and finance*. Jan. 2013, pp. 1–347.

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

[6] Christopher C. Drovandi, Matthew T. Moores, and Richard J. Boys. "Accelerating pseudo-marginal MCMC using Gaussian processes". In: *Computational Statistics & Data Analysis* 118 (2018), pp. 1–17.

[7] Samuel Wiqvist, Umberto Picchini, Julie Lyng Forman, Kresten Lindorff-Larsen, and Wouter Boomsma. "Accelerating delayed-acceptance Markov chain Monte Carlo algorithms". In: *arXiv preprint arXiv:1806.05982* (2018).

[8] Stefan T. Radev, Ulf K. Mertens, Andreass Voss, Lynton Ardizzone, and Ullrich Köthe. "BayesFlow: Learning complex stochastic models with invertible neural networks". In: *arXiv preprint arXiv:2003.06281* (2020).

[9] George Papamakarios, David Sterratt, and Iain Murray. "Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows". In: *Proceedings of Machine Learning Research*. Ed. by Kamalika Chaudhuri and Masashi Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, Apr. 2019, pp. 837–848.

[10] Howard Raïffa and Robert Schlaifer. *Applied statistical decision theory*. Studies in managerial economics. Division of Research, Graduate School of Business Adminitration, Harvard University, 1961.

[11] Francisco J. Samaniego. *A Comparison of the Bayesian and Frequentist Approaches to Estimation*. Springer Series in Statistics. Springer New York, 2010.

[12] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. CRC press, 2011.

[13] Wilfred K. Hastings. "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57.1 (Apr. 1970), pp. 97–109.

[14] Heikki Haario, Eero Saksman, and Johanna Tamminen. "An adaptive Metropolis algorithm". In: *Bernoulli* 7.2 (Apr. 2001), pp. 223–242.

[15] Corey Montella. “The Kalman Filter and Related Algorithms: A Literature Review”. In: (May 2011).

[16] Branko Ristic, Sanjeev Arulampalam, and Neil Gordon. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, 2003.

[17] Michael K. Pitt, Ralph dos Santos Silva, Paolo Giordani, and Robert Kohn. “On some properties of Markov chain Monte Carlo simulation methods based on the particle filter”. In: *Journal of Econometrics* 171.2 (2012). Bayesian Models, Methods and Applications, pp. 134–151.

[18] Bernt Øksendal. *Stochastic Differential Equations: An Introduction with Applications*. Hochschultext / Universitext. Springer, 2003.

[19] Desmond Higham. “An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations”. In: *SIAM Review* 43 (Sept. 2001), pp. 525–546.

[20] R. Girshick. “Fast R-CNN”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448.

[21] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

[22] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. “Algorithms for Hyper-Parameter Optimization”. In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS’11. Granada, Spain: Curran Associates Inc., 2011, pp. 2546–2554.

[23] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D. Cox. “Hyperopt: a Python library for model selection and hyperparameter optimization”. In: *Computational Science & Discovery* 8.1 (July 2015), p. 014008.

[24] Roy De Maesschalck, Delphine Jouan-Rimbaud, and Désiré L. Massart. “The Mahalanobis distance”. In: *Chemometrics and Intelligent Laboratory Systems* 50.1 (2000), pp. 1–18.

[25] Cédric Villani. *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2008.

[26] Nicolas Bonneel, Michiel Panne, Sylvain Paris, and Wolfgang Heidrich. “Displacement Interpolation Using Lagrangian Mass Transport”. In: *ACM Trans. Graph.* 30 (Dec. 2011), p. 158.

A

Appendix 1

A.1 Lemmas concerning Gaussian distributions

Presented here are two important lemmas concerning multivariate Gaussian distributions. They are, among other things, used to derive KF and EKF [1].

Lemma 1. *Let* $\mathbf{x}, \mathbf{y}$ *be two random variables with distributions*

$$\begin{aligned} \mathbf{x} &\sim \mathcal{N}(\mathbf{m}, \mathbf{Q}) \\ \mathbf{y}|\mathbf{x} &\sim \mathcal{N}(\mathbf{H}\mathbf{x}, \mathbf{R}) \end{aligned}$$

then their joint distribution is

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mathbf{m} \\ \mathbf{H}\mathbf{m} \end{pmatrix}, \begin{pmatrix} \mathbf{Q} & \mathbf{Q}\mathbf{H}^T \\ \mathbf{H}\mathbf{Q} & \mathbf{H}\mathbf{Q}\mathbf{H}^T + \mathbf{R} \end{pmatrix} \right).$$

Lemma 2. *Let* $\mathbf{x}, \mathbf{y}$ *be two random variables with joint distribution*

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix}, \begin{pmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^T & \mathbf{B} \end{pmatrix} \right)$$

then we have the following conditional distribution

$$\mathbf{x}|\mathbf{y} \sim \mathcal{N}(\mathbf{a} + \mathbf{C}\mathbf{B}^{-1}(\mathbf{y} - \mathbf{b}), \mathbf{A} - \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^T).$$

Finally, we also present a Gaussian approximation used to derive the EKF.

Approximation 1. *Let* $\mathbf{x}, \mathbf{y}$ *be two random variables with distributions*

$$\begin{aligned} \mathbf{x} &\sim \mathcal{N}(\mathbf{m}, \mathbf{Q}), \\ \mathbf{y}|\mathbf{x} &\sim \mathcal{N}(\mathbf{g}(\mathbf{x}), \mathbf{R}), \end{aligned}$$

where $\mathbf{g}$ *is a non-linear function. The linear approximation based Gaussian approximation to the joint distribution of* $\mathbf{x}$ *and* $\mathbf{y}$ *is*

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mathbf{m} \\ \mathbf{g}(\mathbf{m}) \end{pmatrix}, \begin{pmatrix} \mathbf{Q} & \mathbf{Q}\mathbf{G}_{\mathbf{x}}^T(\mathbf{m}) \\ \mathbf{G}_{\mathbf{x}}(\mathbf{m})\mathbf{Q} & \mathbf{G}_{\mathbf{x}}(\mathbf{m})\mathbf{Q}\mathbf{G}_{\mathbf{x}}^T(\mathbf{m}) + \mathbf{R} \end{pmatrix} \right),$$

where $\mathbf{G}_{\mathbf{x}}$ *is the Jacobian of* $\mathbf{g}$.

A.2 Optimal importance distribution

In order to use *particle filters* we need to sample from an *importance distribution* $q(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{y}_{1:k})$. This distribution can be chosen differently, but the *optimal importance distribution* with regards to variance of the importance weights is

$$q(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{y}_{1:k}) = p(\mathbf{x}_k|\mathbf{x}_{k-1}, y_k).$$

We want to derive this distribution for a linear dynamical system.
Theorem 1. *For a linear dynamical system the optimal importance distribution is*

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{y}_k) = \mathcal{N}(\mathbf{x}_k|\mathbf{m}_k, \mathbf{P}_k), \tag{A.1}$$

where

$$\begin{aligned}\mathbf{m}_k &= \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{Q}_{k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{Q}_{k-1}\mathbf{H}_k^T + \mathbf{R}_k)^{-1}(\mathbf{y}_k - \mathbf{H}_k\mathbf{A}_{k-1}\mathbf{x}_{k-1}),\\ \mathbf{P}_k &= \mathbf{Q}_{k-1} - \mathbf{Q}_{k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{Q}_{k-1}\mathbf{H}_k^T + \mathbf{R}_k)^{-1}\mathbf{H}_k\mathbf{Q}_{k-1}.\end{aligned}$$

Proof. We have that $\mathbf{x}_k|\mathbf{x}_{k-1} \sim \mathcal{N}(\mathbf{A}_{k-1}\mathbf{x}_{k-1}, \mathbf{Q}_{k-1})$ and $\mathbf{y}_k|\mathbf{x}_k \sim \mathcal{N}(\mathbf{H}_k\mathbf{x}_k, \mathbf{R}_k)$. Moreover, $\mathbf{y}_k|\mathbf{x}_k = \mathbf{y}_k|\mathbf{x}_k, \mathbf{x}_{k-1}$. Thus, by Lemma 1 we get

$$\begin{pmatrix}\mathbf{x}_k\\ \mathbf{y}_k\end{pmatrix}|\mathbf{x}_{k-1} \sim \mathcal{N}\left(\begin{pmatrix}\mathbf{A}_{k-1}\mathbf{x}_{k-1}\\ \mathbf{H}_k\mathbf{A}_{k-1}\mathbf{x}_{k-1}\end{pmatrix}, \begin{pmatrix}\mathbf{Q}_{k-1} & \mathbf{Q}_{k-1}\mathbf{H}_k^T\\ \mathbf{H}_k\mathbf{Q}_{k-1}^T & \mathbf{H}_k\mathbf{Q}_{k-1}\mathbf{H}_k^T + \mathbf{R}_k\end{pmatrix}\right).$$

Furthermore, Lemma 2 give us that

$$\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{y}_k \sim \mathcal{N}(\mathbf{m}_k, \mathbf{P}_k),$$

where

$$\begin{aligned}\mathbf{m}_k &= \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{Q}_{k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{Q}_{k-1}\mathbf{H}_k^T + \mathbf{R}_k)^{-1}(\mathbf{y}_k - \mathbf{H}_k\mathbf{A}_{k-1}\mathbf{x}_{k-1}),\\ \mathbf{P}_k &= \mathbf{Q}_{k-1} - \mathbf{Q}_{k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{Q}_{k-1}\mathbf{H}_k^T + \mathbf{R}_k)^{-1}\mathbf{H}_k\mathbf{Q}_{k-1}.\end{aligned}$$

□

We can compare this result to KF. The mean $\mathbf{m}_k$ and variance $\mathbf{P}_k$ can be computed in a similar fashion to the prediction and update steps of KF

$$\begin{aligned}\text{Predict}\quad & \tilde{\mathbf{m}}_k = \mathbf{A}_{k-1}\mathbf{x}_{k-1},\\ & \tilde{\mathbf{P}}_k = \mathbf{Q}_{k-1},\\ \text{Update}\quad & \mathbf{v}_k = \mathbf{y}_k - \mathbf{H}_k\tilde{\mathbf{m}}_k,\\ & \mathbf{S}_k = \mathbf{H}_k\tilde{\mathbf{P}}_k\mathbf{H}_k^T + \mathbf{R}_k,\\ & \mathbf{K}_k = \tilde{\mathbf{P}}_k\mathbf{H}_k^T\mathbf{S}_k^{-1},\\ & \mathbf{m}_k = \tilde{\mathbf{m}}_k + \mathbf{K}_k\mathbf{v}_k,\\ & \mathbf{P}_k = \tilde{\mathbf{P}}_k - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T.\end{aligned}$$

The difference lies in the prediction step. As we assume that the previous value is known the equations are not recursive and we do not need to bring with us the variance of the previous estimated distribution.

If we have a nonlinear dynamical system, we can use local linearization techniques similar to how the EKF is derived. This gives us an approximation of the optimal importance distribution on the same form as in (A.1), but with $\mathbf{m}_k$ and $\mathbf{P}_k$ given by

$$
\begin{aligned}
\text{Predict} \quad & \tilde{\mathbf{m}}_k = \mathbf{f}(\mathbf{x}_{k-1}), \\
& \tilde{\mathbf{P}}_k = \mathbf{Q}_{k-1}, \\
\text{Update} \quad & \mathbf{v}_k = \mathbf{y}_k - \mathbf{h}(\tilde{\mathbf{m}}_k), \\
& \mathbf{S}_k = \mathbf{H}_{\mathbf{x}}(\tilde{\mathbf{m}}_k)\tilde{\mathbf{P}}_k\mathbf{H}_{\mathbf{x}}^T(\tilde{\mathbf{m}}_k) + \mathbf{R}_k, \\
& \mathbf{K}_k = \tilde{\mathbf{P}}_k\mathbf{H}_{\mathbf{x}}^T(\tilde{\mathbf{m}}_k)\mathbf{S}_k^{-1}, \\
& \mathbf{m}_k = \tilde{\mathbf{m}}_k + \mathbf{K}_k\mathbf{v}_k, \\
& \mathbf{P}_k = \tilde{\mathbf{P}}_k - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T.
\end{aligned}
$$

A.3 Description of the log-likelihood function

We simulate an observation $y_{1:T}$ from the OU-model given the model parameters $(\tilde{\gamma}, \tilde{\beta}, \tilde{\sigma}) := (2,\, 0.1,\, 0.01)$. Let S be a grid of values γ, β and σ, defined within the open set $(-2, 6) \times (0, 0.2) \times (0, 0.1)$. The log-likelihood values $\ell(y_{1:T}|\gamma, \beta, \sigma)$ are then estimated using the Kalman filter for each parameter triplet in the grid $(\gamma,\, \beta,\, \sigma) \in S$.

Figure A.1 shows the surface plots of the log-likelihood function over the parameters $(\gamma,\, \beta,\, \sigma) \in S$, while keeping one parameter equal to the model parameter value which gave rise to y. We see a tendency of the log-likelihood function to quickly diverge to negative infinity when leaving the region closest to the model parameters $(\tilde{\gamma}, \tilde{\beta}, \tilde{\sigma})$.

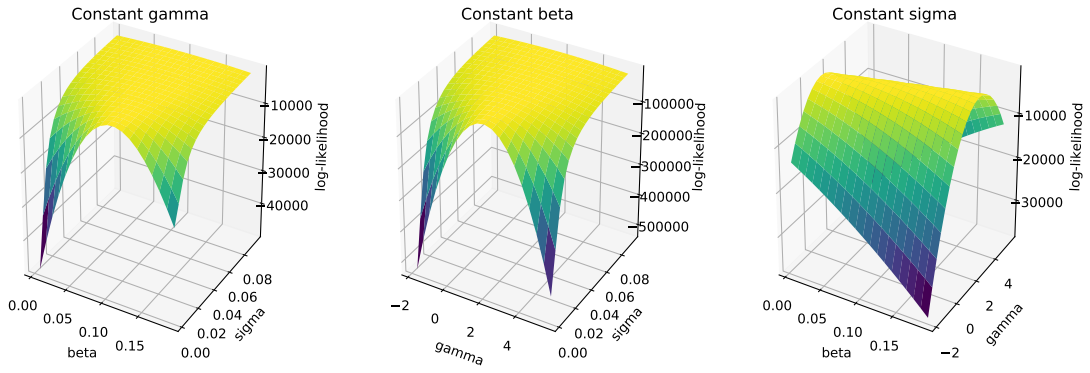


Figure A.1: Surface plots of log-likelihood $\ell(y_{1_T}|\gamma, \beta, \sigma)$ for $(\gamma, \beta, \sigma) \in S$, while keeping one of the parameters constant. The constant parameter is set to the true value of the parameter, in other words setting exclusively $\gamma := \tilde{\gamma}$, $\beta := \tilde{\beta}$, or $\sigma := \tilde{\sigma}$).

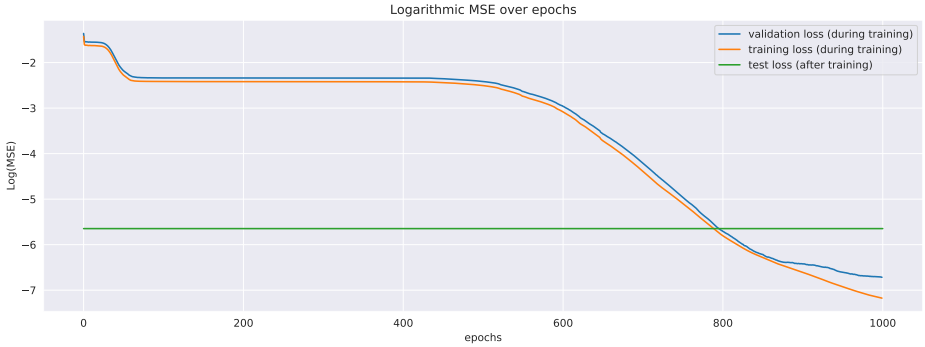


Figure A.2: The logarithmic loss over every 100th epoch for the OU-model. The training data is split up into 70% training data, 15% validation data, and 15% test data. We see that the Adam optimiser manages to bypass two major saddle points.

A.4 Description of the training loss

The loss function from training the neural network on the data generated by the Ornstein-Uhlenbeck model is presented in Figure A.2. The data is presented in Section 6.1. We see that the loss function bypasses two saddle points, the latter of which stretches for a long time before being improved. We also see that there is very little indication of over-fitting since the validation loss remains stable.

A.5 Hyperparameter optimisation results

For the hyperparameter optimisation in Section 6.2, the convergence results are presented in Figure A.3. We see that the convergence plots focus on very clear regions, suggesting that $\beta_1 > 0.9$ and $\beta_2 > 0.99$ is preferable for the Adam Optimiser, see Section 4.5. The learning rate η also focuses on values around $\log \eta \approx -9$.

There is a tendency for the training to get stuck in local minimum, which by increasing β_1 above the default 0.9 improves the performance of the training. See Appendix A.4 for a presentation of how the loss function in the OU-model case. Note that the loss reaches saddle points which are later resolved by the adaptive nature of the Adam optimiser. In general, higher values of β_1 and β_2 are associated with more focus on getting out of saddle points and other difficult regions.

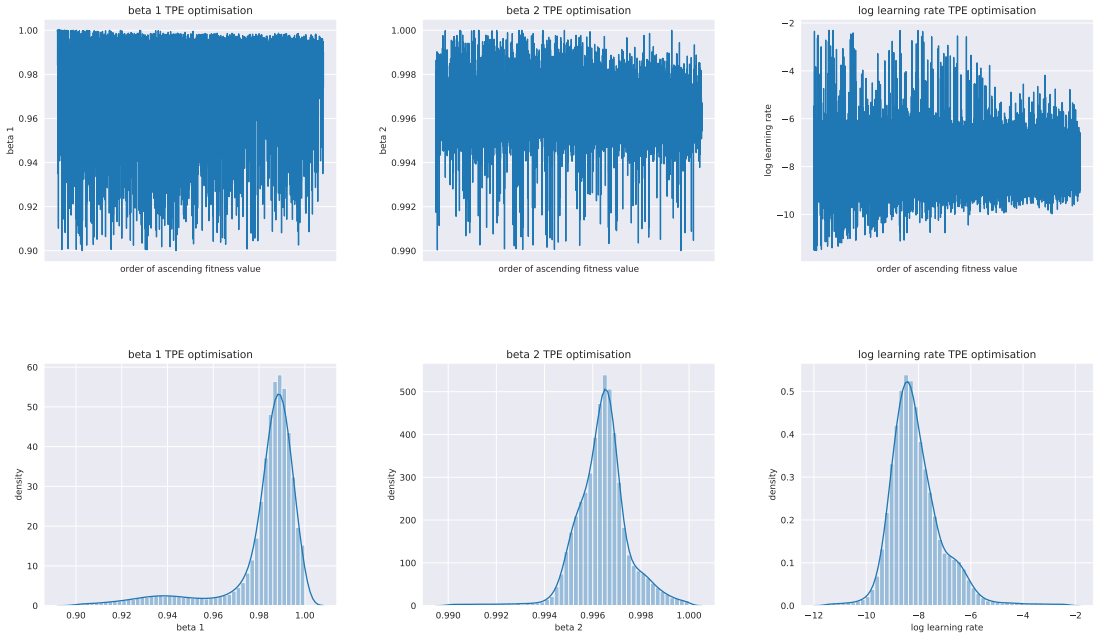


Figure A.3: The result of the TPE hyperparameter optimisation algorithm. Figures A.3a, A.3b and A.3c present the convergence plots of the parameter values in ascending order with regards to their associated fitness value. Figures A.3d, A.3e and A.3f present the marginal distributions of the samples parameter choices.