A Particle Filter Method of Inference for Stochastic Differential Equations

by

Pranav Subramani

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Data Science

Waterloo, Ontario, Canada, 2022

Abstract

Stochastic Differential Equations (SDE) serve as an extremely useful modelling tool in areas including ecology, finance, population dynamics, and physics. Yet, parameter inference for SDEs is notoriously difficult due to the intractability of the likelihood function. A common approach is to approximate the likelihood by way of data augmentation, then integrate over the latent variables using particle filtering techniques. In the Bayesian setting, the particle filter is typically combined with various Markov chain Monte Carlo (MCMC) techniques to sample from the parameter posterior. However, MCMC can be excessive when this posterior is well-approximated by a normal distribution, in which case estimating the posterior mean and variance by stochastic optimization presents a much faster alternative. This thesis explores this latter approach. Specifically, we use a particle filter tailored to SDE models and consider various methods for approximating the gradient and hessian of the parameter log-posterior. Empirical results for several SDE models are presented.

Acknowledgements

I would like to first thank my advisor, Prof. Martin Lysy for giving me the wonderful opportunity to work with him on exciting research and for mentoring me through this thesis. Next, I would like to thank my family for their constant support and help. I'd also like to take this opportunity to thank Prof. Pascal Poupart and Prof. Gautam Kamath for their mentorship during my undergraduate days for allowing me to pursue research with them and for developing a keen curiosity and passion for doing research. I'd like to thank all of my friends in Waterloo, Jimmy, Sanjay, Simran, Arnab, Dejen and many more for their constant companionship especially during this pandemic. Finally, I'd like to thank my research group comprising of Mohan, Jonathan, Michelle, Kanika who have helped answer questions, discuss different research problems and always served as a useful sounding board for the research we pursued.

Dedication

This is dedicated to my family.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Stochastic Differential Equations (SDE) have seen widespread adoption in a variety of academic fields ranging from physics and simulation of dynamics to ecology and finance [Särkkä and Solin, 2019]. As a result, being able to perform inference for high dimensional SDEs is a valuable research proposition. However, SDEs are notoriously difficult to perform inference for and have several pitfalls. Some of the pitfalls involve the complexity of the models [Picchini, 2014] which involves the number of parameters to infer. That being said, the most notorious of these difficulties lies in the intractability of the likelihood function. The pitfalls could also entail the choice of posterior distribution or the magnitude of the noise compared to the signal. Alternatively, another source of difficulty might arise from the lack of useful high-performance software for performing inference for SDEs. In the recent past, the advent of graphic-processing-units for high performance computation in machine learning can now potentially be leveraged for performing inference for SDEs as well. Moreover, some of the code that is used for high-performance computing for machine learning might utilize abstract programming constructs that can be readily adapted for inference for SDEs.

There have also been several contributions to the advancements of algorithms based on sequential Monte Carlo (SMC) in the recent past [Wigren et al., 2019, Kudlicka et al., 2020]. These improvements provide significantly better results (which might range from a reduction in the number of particles used to a reduction in the variance of estimated quantities) for a variety of models. Some of these advancements could also be utilized within the context of SDEs which has not been explored by the papers' respective authors. Given that this thesis mainly utilizes a variant of sequential Monte Carlo that does not deal with the parameters (i.e. the particle filter), advancements in SMC can potentially be ported directly to this setting. Furthermore, several programming advancements within the SMC framework are built on top of domain-specific languages [Murray and Schön, 2018] and there is a lack of software that utilizes SMC and is carefully finetuned for SDEs.

1.2 Contributions

In this section we outline the major contributions of this work:

- We combine state-of-the-art techniques in particle filtering and stochastic optimization to propose a method of inference tailors specifically to SDEs.
- We develop high performance software for our parameter estimator.
- We evaluate our methods and software improvements on several SDE models of interest.

1.3 Outline of Thesis

This thesis is divided into 6 chapters. The introduction deals with the motivation for inference for stochastic differential equations and the variety of instances they come up in, then we proceed to give a description of our major research contributions through this thesis and finally, we present an outline for the thesis. The second chapter dives into the background material required for this thesis. In particular, we explain several programming tools and frameworks as well as methods that have been applied for the experiments and the code that will be released. We also elaborate on the different proposals we can use for SDEs as well as an introduction to stochastic differential equations and sequential Monte Carlo. Chapter 3 dives into the methodology of the thesis itself, specifically, what are the major techniques we are using in this thesis. We talk about the implementation of the stochastic optimization algorithm within the particle filter. We also go into detail about the variable transformations required to use the stochastic optimization routines and finally, we discuss the programming constructs that we needed to utilize in order to obtain the results. Chapter 4 deals with the experimental section of the thesis where we outline several experiments that we would like to run in order to empirically validate our hypothesis regarding the effectiveness of some of the methods. To do this, we perform several experiments in high noise and low noise settings for different models to see how they would perform. In Chapter 5, we discuss some of these results and qualitatively evaluate their inference on some of our problems. We then proceed to discuss the issue of bias that is present within the gradients and notice that our method presents a biased estimate of the gradient, however, we show that the method is still effective at yielding high-precision parameter estimates. Finally, we conclude with what we have shown through this thesis and outline several future directions that are worth pursuing and are made possible as a consequence of some of the work we have done.

Chapter 2

Background Material

In this chapter we outline the different background material required to read this thesis. In particular, we note that the background includes both theoretical contributions arising from stochastic differential equations as well as practical techniques utilizing software advancements in recent years. It also outlines the workhorse algorithm for sequential Monte Carlo and some of the advancements in that domain.

2.1 Stochastic Differential Equations

To begin, we describe what class of problems we are dealing with. Let $\boldsymbol{X}(t) = (X_1(t), \dots, X_d(t))$ denote a d-dimensional stochastic process. Then $\boldsymbol{X}(t)$ is said to satisfy a stochastic differential equation (SDE) [Oksendal, 2013] if it can be written as

$$\mathrm{d}\boldsymbol{X}(t) = \boldsymbol{\Lambda}_{\boldsymbol{\theta}}(\boldsymbol{X}(t))\,\mathrm{d}t + \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{X}(t))^{1/2}\,\mathrm{d}\boldsymbol{B}(t), \tag{2.1}$$

where $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}(\boldsymbol{X})$ is a d-dimensional drift function, $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{X})$ is a $d \times d$ positive-definite diffusion matrix, and $\boldsymbol{B}(t)$ is d-dimensional Brownian motion. We shall provide a precise construction of the stochastic process satisfying (2.1) momentarily. For now, we just mention two fundamental properties of SDEs:

- $\boldsymbol{X}(t)$ has almost-surely continuous sample paths.
- $\boldsymbol{X}(t)$ satisfies the Markov property:

$$p(\boldsymbol{X}(t) \mid \{\boldsymbol{X}(u) : u \leq s < t\}, \boldsymbol{\theta}) = p(\boldsymbol{X}(t) \mid \boldsymbol{X}(s), \boldsymbol{\theta}). \tag{2.2}$$

The SDE inference problem is that of estimating the unknown model parameters $\boldsymbol{\theta}$ from noisy SDE observations, $\boldsymbol{Y}_{0:N} = (\boldsymbol{Y}_0, \ldots, \boldsymbol{Y}_N)$ with:

$$\boldsymbol{Y}_n \overset{\text{ind}}{\sim} g(\boldsymbol{Y}_n \mid \boldsymbol{X}_n, \boldsymbol{\theta}), \tag{2.3}$$

where $\boldsymbol{X}_n = \boldsymbol{X}(t_n)$, and for simplicity we assume hereafter that $t_n = n\Delta t$. We will also assume that the initial value of the SDE at time $t_0 = 0$ has the prior distribution

$$\boldsymbol{X}_0 \sim \pi(\boldsymbol{X}_0 \mid \boldsymbol{\theta}). \tag{2.4}$$

This will result in the following likelihood:

$$\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{Y}_{0:N}) = \int \left[\pi(\boldsymbol{X}_0 \mid \boldsymbol{\theta}) \cdot \prod_{n=0}^{N} g(\boldsymbol{Y}_n \mid \boldsymbol{X}_n, \boldsymbol{\theta}) \cdot \prod_{n=1}^{N} p(\boldsymbol{X}_n \mid \boldsymbol{X}_{n-1}, \boldsymbol{\theta}) \right] \mathrm{d}\boldsymbol{X}_{0:N}. \tag{2.5}$$

To utilize the above expression, we need the SDE transition density $p(\boldsymbol{X}_n \mid \boldsymbol{X}_{n-1}, \boldsymbol{\theta})$, which is seldom present in a closed form. To alleviate this, we approximate the likelihood function by the Euler (Euler-Maruyama) [Maruyama, 1955] discretization method. Namely, for $m \geq 1$, let $\boldsymbol{X}_n^{(m)} = \boldsymbol{X}(n\Delta t/m)$ be the value of the SDE at time $t = n\Delta t/m$, such that $\boldsymbol{X}_{mn}^{(m)} = \boldsymbol{X}_n = \boldsymbol{X}(t_n)$. As $\Delta t_m = \Delta t/m \to 0$, the normal approximation

$$\boldsymbol{X}_n^{(m)} \mid \boldsymbol{X}_{n-1}^{(m)} \sim \mathcal{N}(\boldsymbol{X}_{n-1}^{(m)} + \boldsymbol{\Lambda}_{\boldsymbol{\theta}}(\boldsymbol{X}_{n-1}^{(m)})\Delta t_m, \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{X}_{n-1}^{(m)})\Delta t_m) \tag{2.6}$$

becomes increasingly accurate. In this sense, the Euler approximation of resolution m to the likelihood function is

$$\hat{\mathcal{L}}_m(\boldsymbol{\theta} \mid \boldsymbol{Y}_{0:N}) = \int \left[\pi(\boldsymbol{X}_0^{(m)} \mid \boldsymbol{\theta}) \cdot \prod_{n=0}^{N} g(\boldsymbol{Y}_n \mid \boldsymbol{X}_{nm}^{(m)}, \boldsymbol{\theta}) \cdot \prod_{n=1}^{Nm} \varphi(\boldsymbol{X}_n^{(m)} \mid \boldsymbol{X}_{n-1}^{(m)}, \boldsymbol{\theta}) \right] \mathrm{d}\boldsymbol{X}_{0:Nm}^{(m)}, \tag{2.7}$$

where $\varphi(\boldsymbol{X}_n^{(m)} \mid \boldsymbol{X}_{n-1}^{(m)}, \boldsymbol{\theta})$ is the PDF of the normal distribution in (2.6), and we obtain $\hat{\mathcal{L}}_m(\boldsymbol{\theta} \mid \boldsymbol{Y}_{0:N}) \to \mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{Y}_{0:N})$ as $m \to \infty$ [Pedersen, 1995].

2.2 Particle Filters

The high dimensional integral (2.7) is at the heart of the computational challenge for SDE inference. A particularly effective method to stochastically approximate it is with a particle filter [Gordon et al., 1993]. In order to present the algorithm in a general context, we first set up some preliminaries with respect to the variables we will be using. Let $\boldsymbol{x}_t$ and $\boldsymbol{y}_t$ denote respectively the (latent) state and (observed) measurement variables at times $t = 0, \ldots T$, and consider a model of the form

$$\begin{aligned} \boldsymbol{x}_0 &\sim \pi(\boldsymbol{x}_0 \mid \boldsymbol{\theta}) \\ \boldsymbol{x}_t \mid \boldsymbol{x}_{t-1} &\sim f(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta}) \\ \boldsymbol{y}_t \mid \boldsymbol{x}_t &\sim g(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta}). \end{aligned} \tag{2.8}$$

Model (2.8) is called a state-space model. In the SDE context of (2.7), we have $\boldsymbol{x}_t = (\boldsymbol{X}^{(m)}_{(t-1)m+1}, \ldots, \boldsymbol{X}^{(m)}_{tm})$ and $\boldsymbol{y}_t = \boldsymbol{Y}_t$. The goal is to estimate the observed data likelihood

$$\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y}_{0:T}) = \int \pi(\boldsymbol{x}_0 \mid \boldsymbol{\theta}) \cdot \prod_{t=0}^{T} g(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{\theta}) \cdot \prod_{t=1}^{T} f(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{x}_{0:T}. \tag{2.9}$$

The particle filter requires one to specify proposal distributions $q_0(\boldsymbol{x}_0 \mid \boldsymbol{y}_0, \boldsymbol{\theta})$ and $q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{y}_t, \boldsymbol{\theta})$ for the initial and subsequent state variables, respectively, from which we can both sample from and evaluate the log-density of the distribution function. Algorithm 1 below presents a basic particle filter which both approximates $\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y}_{0:T})$ and draws a sample from the latent variable distribution $p(\boldsymbol{x}_{0:T} \mid \boldsymbol{\theta}, \boldsymbol{y}_{0:T})$.

Algorithm 1 Particle Filter with Multinomial Resampling

```
$\boldsymbol{x}_0^{1:N} \overset{\text{iid}}{\sim} q_0(\boldsymbol{x}_0 \mid \boldsymbol{y}_0, \boldsymbol{\theta})$ // sample first time point
$w_0^i = g(\boldsymbol{y}_0 \mid \boldsymbol{x}_0, \boldsymbol{\theta})\pi(\boldsymbol{x}_0 \mid \boldsymbol{\theta})/q_0(\boldsymbol{x}_0 \mid \boldsymbol{y}_0, \boldsymbol{\theta})$
for $t \geq 1$ do
    $v_{t-1}^i = w_{t-1}^i / \sum_{i=1}^N (\boldsymbol{w}_{t-1}^i)$
    $\boldsymbol{A}_t \sim \text{Multinomial}(N, \boldsymbol{v}_t)$
    $\boldsymbol{x}_t^i \overset{\text{ind}}{\sim} q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}^{A_i^t}, \boldsymbol{y}_t, \boldsymbol{\theta})$
    $w_t^i = g(\boldsymbol{y}_t \mid \boldsymbol{x}_t^i) f(\boldsymbol{x}_t^i \mid \boldsymbol{x}_{t-1}^{A_i^t}, \boldsymbol{\theta}) / q(\boldsymbol{x}_t^i \mid \boldsymbol{x}_{t-1}^{A_i^t}, \boldsymbol{y}_t, \boldsymbol{\theta})$
end
$\hat{\mathcal{L}}(\boldsymbol{\theta} \mid \boldsymbol{y}_{0:T}) = \prod_{t=0}^T \left( \frac{1}{N} \sum_{i=1}^N \boldsymbol{w}_t^i \right)$
$J_T \sim \text{Multinomial}(1, \boldsymbol{v}_T)$ // index of particle at time $T$
for $t = T \cdots 1$ do
    $J_{t-1} = A_t^{J_t}$ // index of same particle at time $t-1$
end
$\boldsymbol{x}_{0:T} = (\boldsymbol{x}_0^{J_0}, \ldots, \boldsymbol{x}_T^{J_T})$
Return $\hat{\mathcal{L}}(\boldsymbol{\theta} \mid \boldsymbol{y}_{0:T}), \boldsymbol{x}_{0:T}$
```

There has been a large body of work focused on different resampling schemes and comparing them[Douc and Cappé, 2005, Johansen and Doucet, 2008], but for this thesis we focus on the multinomial on the multinomial resampler. Similarly, we also have a wide variety of particle filters to choose from. For example, the auxiliary particle filter [Pitt and Shephard, 1999, Johansen and Doucet, 2008] uses a look-ahead strategy to improve sampling from $p(\boldsymbol{x}_{0:T} \mid \boldsymbol{\theta}, \boldsymbol{y}_{0:T})$. Another common approach to filtering is to utilize Markov-Chain Monte Carlo (MCMC) steps within the SMC algorithm [Doucet et al., 2009]. Particularly, we can use MCMC updates for the parameters of interest [Ronquist et al., 2021]. However, we do not pursue these extensions of Algorithm 1 here, focusing rather on the critical choice of proposal distribution which we discuss below.

2.2.1 Bridge Proposal

Perhaps the simplest of particle filter proposal distributions are are $q_0 = \pi$ and $q = f$. This is called a bootstrap filter [Gordon et al., 1993]. However, the bootstrap filter tends to perform poorly when the observed variables $\boldsymbol{y}_t$ have low noise about $\boldsymbol{x}_t$, in that the particle weights tend to collapse onto a single particle. This is known as the *particle degeneracy* problem [Doucet et al., 2009]. Fortunately, a much better proposal distribution has been developed for SDEs [Durham and Gallant, 2002, Stramer and Yan, 2007]. Before describing this so-called *bridge proposal*, consider the following proposition:

Proposition 1 *If*

$$\begin{aligned}\boldsymbol{W} &\sim \mathcal{N}(\boldsymbol{\mu}_W, \boldsymbol{\Sigma}_W)\\ \boldsymbol{X} \mid \boldsymbol{W} &\sim \mathcal{N}(\boldsymbol{W} + \boldsymbol{\mu}_{X|W}, \boldsymbol{\Sigma}_{X|W})\\ \boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{W} &\sim \mathcal{N}(\boldsymbol{A}\boldsymbol{X}, \boldsymbol{\Omega}),\end{aligned} \tag{2.10}$$

then

$$\begin{bmatrix}\boldsymbol{W}\\ \boldsymbol{Y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_W\\ \boldsymbol{\mu}_Y = \boldsymbol{A}[\boldsymbol{\mu}_W + \boldsymbol{\mu}_{X|W}]\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_W & \boldsymbol{\Sigma}_W\boldsymbol{A}'\\ \boldsymbol{A}\boldsymbol{\Sigma}_W & \boldsymbol{\Sigma}_Y = \boldsymbol{A}(\boldsymbol{\Sigma}_W + \boldsymbol{\Sigma}_{X|W})\boldsymbol{A}' + \boldsymbol{\Omega}\end{bmatrix}\right), \tag{2.11}$$

such that

$$\boldsymbol{W} \mid \boldsymbol{Y} \sim \mathcal{N}\left(\boldsymbol{\mu}_W + \boldsymbol{\Sigma}_W\boldsymbol{A}'\boldsymbol{\Sigma}_Y^{-1}(\boldsymbol{Y} - \boldsymbol{\mu}_Y), \boldsymbol{\Sigma}_W - \boldsymbol{\Sigma}_W\boldsymbol{A}'\boldsymbol{\Sigma}_Y^{-1}\boldsymbol{A}\boldsymbol{\Sigma}_W\right). \tag{2.12}$$

Proposition 1 is proved in the Appendix (A).

Recall that for SDEs we have $\boldsymbol{x}_t = (\boldsymbol{X}^{(m)}_{(t-1)m+1}, \ldots, \boldsymbol{X}^{(m)}_{tm})$ and $\boldsymbol{y}_t = \boldsymbol{Y}_t$. Now suppose that the SDE measurements are multivariate normal:

$$\boldsymbol{Y}_t \overset{\text{ind}}{\sim} \mathcal{N}(\boldsymbol{A}\boldsymbol{X}^{(m)}_{tm}, \boldsymbol{\Omega}). \tag{2.13}$$

The bridge proposal for $\boldsymbol{x}_t$ then proceeds recursively as follows:

1. Without loss of generality, let $t = 1$, and assume that for fixed $0 \le n < m - 1$ the proposal value $\boldsymbol{X}^{(m)}_{(t-1)m+n} = \boldsymbol{X}^{(m)}_n$ is given. Now suppose we wish to specify the proposal distribution for $\boldsymbol{X}^{(m)}_{n+1}$.

2. Assume that $\boldsymbol{X}(t)$ is a Brownian motion with drift for $t \in (\frac{n}{m}\Delta t, \Delta t)$:

$$\mathrm{d}\boldsymbol{X}(t) = \boldsymbol{\Lambda}_n \,\mathrm{d}t + \boldsymbol{\Sigma}_n^{1/2}\,\mathrm{d}\boldsymbol{B}(t), \tag{2.14}$$

where $\boldsymbol{\Lambda}_n = \boldsymbol{\Lambda}_{\boldsymbol{\theta}}(\boldsymbol{X}^{(m)}_n)$ and $\boldsymbol{\Sigma}_n = \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{X}^{(m)}_n)$. This means that for any $\frac{n}{m}\Delta t \le s \le s + u \le \Delta t$ we have

$$\boldsymbol{X}(s+u) \mid \boldsymbol{X}(s) \sim \mathcal{N}(\boldsymbol{X}(s) + u\boldsymbol{\Lambda}_n, u\boldsymbol{\Sigma}_n), \tag{2.15}$$

To obtain the proposal for $\boldsymbol{X}^{(m)}_{n+1}$, we apply the formula to $(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}) = (\boldsymbol{X}^{(m)}_{n+1}, \boldsymbol{X}^{(m)}_m, \boldsymbol{Y}_1)$, for which we have

$$\begin{aligned} \boldsymbol{X}^{(m)}_{n+1} &\sim \mathcal{N}(\boldsymbol{X}^{(m)}_n + \Delta t_m \boldsymbol{\Lambda}_n, \Delta t_m \boldsymbol{\Sigma}_n) \\ \boldsymbol{X}^{(m)}_m \mid \boldsymbol{X}^{(m)}_{n+1} &\sim \mathcal{N}(\boldsymbol{X}^{(m)}_{n+1} + k\Delta t_m \boldsymbol{\Lambda}_n, k\Delta t_m \boldsymbol{\Sigma}_n) \\ \boldsymbol{Y}_1 \mid \boldsymbol{X}^{(m)}_m, \boldsymbol{X}^{(m)}_{n+1} &\sim \mathcal{N}(\boldsymbol{A}\boldsymbol{X}^{(m)}_m, \boldsymbol{\Omega}), \end{aligned} \tag{2.16}$$

where $k = m - n$, such that in the formula we have

$$\begin{aligned} \boldsymbol{\mu}_W &= \boldsymbol{X}^{(m)}_n + \boldsymbol{\Lambda}_n \Delta t_m, & \boldsymbol{\Sigma}_W &= \Delta t_m \boldsymbol{\Sigma}_n, \\ \boldsymbol{\mu}_Y &= \boldsymbol{A}[\boldsymbol{X}^{(m)}_n + k\Delta t_m \boldsymbol{\Lambda}_n], & \boldsymbol{\Sigma}_Y &= k\Delta t_m \boldsymbol{A}\boldsymbol{\Sigma}_n \boldsymbol{A}' + \boldsymbol{\Omega}. \end{aligned} \tag{2.17}$$

2.3 Automatic Differentiation

Automatic differentiation [Baydin et al., 2015] has seen much adoption in recent years due to advances in machine learning. We note that the primary algorithm for parameter learning of deep neural networks requires backpropagation [Rumelhart et al., 1986] which involves computing the partial derivatives of the loss function. Modern software [Abadi et al., 2015, Paszke et al., 2019] includes tools like traced arrays which allows for the efficient representation of the graph of computations that occur during program execution. This representation is then utilized to compute the partial derivatives for forward and reverse mode auto-differentiation.

Now, we describe a simple example of automatic differentiation.

$$y = f(g(\theta)) \tag{2.18}$$

Now, we utilize the substitution $a_0 = \theta$, $a_1 = g(a_0)$, $a_2 = f(a_1)$. Now, we have:

$$y = \frac{d}{da_1} f(a_1) \cdot \frac{d}{da_0} g(a_0) \tag{2.19}$$

This gives us the ability to compute the values as functions of the previous inputs to it in a computational graph. This naturally fits into the framework of GPUs and can also be utilized in CPUs. Furthermore, automatic differentiation is also a core tenet of several frameworks in the `Python` ecosystem for high performance computing. The efficient computing of gradients also allows for several improvements to algorithms. Particularly, if we wish to use gradient based moves for MCMC, we can make use of automatic differentiation for this. In the next section we elaborate on one such framework.

2.4 JAX

JAX[Frostig et al., 2018] is a high performance computation library built in `Python`. It combines automatic differentiation, Just In-Time Compilation (JIT) and vectorization to achieve extremely fast runtimes and low memory utilizations for machine learning and related areas. JAX possesses a JIT compiler that traces the expressions that are evaluated in a function and then constructs a jax expression (`jaxpr`) which is then utilized by the XLA [Sabne, 2020] (Accelerated Linear Algebra) compiler. These traced expressions are performed with the help of a traced array which is a JAX-specific object. JAX is built using the same API as `Numpy` [Harris et al., 2020] which makes it easy to use in place of existing libraries employing numerical computation in `Python`.

Vectorization is also an essential component to JAX whether it be parallelization across hardware or across dimensions in data objects. JAX provides two language primitives `vmap` and `pmap` which allow the user to parallelize across data dimensions and devices respectfully. These JAX primitives also integrate seamlessly with the JIT ecosystem (so long as the function being mapped is JIT compatible) which provide runtime and memory benefits[Subramani et al., 2021].

JIT compilation is not compatible with any `Python` code, particularly, control flow and loops are not immediately conducive to being JIT compiled. There are workarounds and constructs that exist within the JAX ecosystem that circumvent these issues, however, it is important to note that prototyping code in JAX is non-trivial especially if the code intends to be JIT compiled. Furthermore, JIT compilation in an interpreted language happens in the first iteration, therefore, one may expect that the first iteration of the function runs much slower than the other iterations. That being said, the runtime is significantly longer if the code is not JIT compiled versus what it would be if it were JIT compiled.

To provide the reader with an understanding of the efficacy of JIT compilation, we refer the reader to a model described in Chapter 4 for 100 particles and 100 observations for several iterations and deliver the mean running time with the standard deviation. The model itself is 3-dimensional. The non-JIT compiled code takes 1.31s $\pm$ 19.8ms per loop iteration whereas the code that is JIT compiled 1.22ms $\pm$ 2.76μs which is roughly 1000x faster. Note that this is specific to this particular example, the runtime can be longer or shorter depending on the code that is being statically compiled.

Chapter 3

Methodology

In this section we outline our primary algorithm for combining stochastic optimization and particle filtering. We also elaborate on some of the considerations that need to be taken with respect to the search space for optimization, particularly for variables that are constrained. Finally, we comment on the computational techniques that we employ to build these algorithms efficiently.

3.1 Bayesian Normal Approximation

In the Bayesian context, inference for the state-space model (2.8) (and for the SDE state-space model (2.7) in particular) in conducted via the posterior distribution

$$p(\boldsymbol{\theta} \mid \boldsymbol{y}_{0:T}) \propto \mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y}_{0:T})\pi(\boldsymbol{\theta}),$$

where $\pi(\boldsymbol{\theta})$ is the parameter prior. Often this parameter posterior is explored using MCMC sampling techniques. However, this can be somewhat excessive when the number of observations T is sufficiently large. This claim is justified by the so-called Bayesian normal approximation [Gelman et al., 1995], which roughly states that for the state-space model (2.8), under the same hypotheses and asymptotic conditions for which the maximum likelihood estimator (MLE) of $\boldsymbol{\theta}$ is approximately normal, the posterior distribution $p(\boldsymbol{\theta} \mid \boldsymbol{y}_{0:T})$ is also approximately normal,

$$\boldsymbol{\theta} \mid y_{0:T} \overset{\text{iid}}{\sim} \mathcal{N}(\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\Sigma}}), \tag{3.1}$$

where $\hat{\boldsymbol{\theta}} = \operatorname{argmax} \log p(\boldsymbol{\theta} \mid \boldsymbol{y}_{0:T})$, $\hat{\boldsymbol{\Sigma}} = [\frac{d^2}{d\boldsymbol{\theta}^2} \log p(\hat{\boldsymbol{\theta}} \mid \boldsymbol{y}_{0:T})]^{-1}$, and $\log p(\boldsymbol{\theta} \mid \boldsymbol{y}_{0:T}) = \ell(\boldsymbol{\theta} \mid \boldsymbol{y}_{0:T}) + \log \pi(\boldsymbol{\theta})$. When the Bayesian normal approximmation (3.1) is justified, instead of applying MCMC to conduct full Bayesian inference.

3.2 Stochastic Optimization with Particle Filtering

The algorithm that will be in the section below is quite simple. In a nutshell, we first run the particle filter and then take the gradient of the log-likelihood of the particle filter with respect to the parameters and step in that direction with some fixed ε. We can either repeat this procedure until a preset convergence tolerance has been obtained or for a fixed number of iterations. For the former, we note that the advantage is that we will always have arrived at a converged optima while in the latter, we may still have not converged. The advantage for the latter approach is that we now have a fixed computational budget that is quantifiable while in the former we do not. For the purposes of the experiments in this thesis, we restrict ourselves to a fixed computational budget.

Algorithm 2 Gradient Descent on the Log Posterior, θ

```
Result: $\hat{\boldsymbol{\theta}}$
Initialize $N \leftarrow$ number of particles
Initialize $y_{0:T} \leftarrow$ observations
Initialize $\varepsilon \leftarrow$ learning rate
Initialize $\hat{\boldsymbol{\theta}}$
for $i$ in $1:K$ do
    $\ell(\hat{\boldsymbol{\theta}}) \leftarrow$ particleFilter($y_{0:T}$, $N$, $\hat{\boldsymbol{\theta}}$)
    $\tilde{\boldsymbol{\theta}} \leftarrow \nabla_{\boldsymbol{\theta}}(\ell(\hat{\boldsymbol{\theta}}) + \pi(\hat{\boldsymbol{\theta}}))$
    $\hat{\boldsymbol{\theta}} \leftarrow \varepsilon \cdot \tilde{\boldsymbol{\theta}} - \hat{\boldsymbol{\theta}}$
end
```

Note that for the above algorithm we have K parameters, N iterations and M is the number of particles. We should also note that we are utilizing automatic differentiation to differentiate through the particle filter. ℓ is also the log-likelihood of the data in the algorithm above. This still optimizes for the posterior as discussed in the previous section. Additionally, this algorithm does not entail the computational techniques which are utilized to make this both memory and runtime efficient.

Notice that the algorithm did not utilize any of the advancements made to stochastic optimization which has been the workhorse algorithm for deep learning over the past decade. This may include improvements to stochastic optimization like ADAM [Kingma and Ba, 2014].

3.3 Variable Transformations for Stochastic Optimization

Several variables require specific consideration to ensure that we can perform stochastic optimization on an unconstrained scale. Note that, if we were to perform stochastic optimization directly on the parameter space with constraints, depending on the step size, we may potentially have an undefined likelihood. The primary variable transformations we use for a variable, α are:

- if $\alpha \in \mathbb{R}^+$, then, $\alpha \to \log(\alpha)$.
- If $\alpha < 0$, let $\alpha \to \log(-\alpha)$.
- If $\alpha \in [a, b]$, let $\alpha \to \log(\frac{\alpha - a}{b - \alpha})$. This is commonly referred to as the logit transform.

3.4 Gradient and Hessian Estimation

The first method of obtaining a gradient and hessian estimate for SDEs through particle filters is to differentiate through the particle filter. We may perform this with the help of automatic differentiation where we run the particle filter and obtain the gradient with respect to the parameters. This method is highlighted in Algorithm 3.2. While this method would yield an estimate of the gradient and hessian, these estimates are biased. Recent work [Corenflos et al., 2021] highlights a theoretical result which shows that this method is in fact biased.

In order to perform any form of inference for models, under the assumption that our parameters are normally distributed, we must obtain an estimate of the mean and the variance (additionally, the covariance between the parameters are important to obtain as well). In [Poyiadjis et al., 2011], a method is provided to infer estimates of the gradient and hessian for the particle filter where the resampling is performed with the multinomial resampling scheme. The method utilizes pointwise estimates that they compute for the gradient and the second derivative. They first use Fisher's identity for the score (first derivative), which is

$$\nabla \log p(\boldsymbol{y}_{0:T} \mid \boldsymbol{\theta}) = \int \nabla p(\boldsymbol{x}_{0:T}, \boldsymbol{y}_{0:T} \mid \boldsymbol{\theta}) \cdot p(\boldsymbol{x}_{0:T}|\boldsymbol{y}_{0:T}, \boldsymbol{\theta}) d\boldsymbol{x}_{0:T} \tag{3.2}$$

Additionally, they also present the following identity for the observed information matrix.

$$-\nabla^2 \log p(\boldsymbol{y}_{0:T} \mid \boldsymbol{\theta}) = \nabla \log p(\boldsymbol{y}_{0:T} \mid \boldsymbol{\theta}) \nabla \log p(\boldsymbol{y}_{0:T} \mid \boldsymbol{\theta})^T - \frac{\nabla^2 p(\boldsymbol{y}_{0:T} \mid \boldsymbol{\theta})}{p(\boldsymbol{y}_{0:T} \mid \boldsymbol{\theta})}, \tag{3.3}$$

with

$$\begin{aligned}\frac{\nabla^2 p(\boldsymbol{y}_{0:T}) \mid \boldsymbol{\theta}}{p(\boldsymbol{y}_{0:T} \mid \boldsymbol{\theta})} &= \int \nabla \log p(\boldsymbol{x}_{0:T}, \boldsymbol{y}_{0:T}, \mid \boldsymbol{\theta}) \nabla \log p(\boldsymbol{x}_{0:T}, \boldsymbol{y}_{0:T}, \mid \boldsymbol{\theta})^T \cdot p(\boldsymbol{x}_{0:T}|\boldsymbol{y}_{0:T}, \boldsymbol{\theta}) d\boldsymbol{x}_{0:T} \\ &+ \int \nabla^2 \log p(\boldsymbol{x}_{0:T}, \boldsymbol{y}_{0:T}, \boldsymbol{\theta}) \cdot p(\boldsymbol{x}_{0:T}|\boldsymbol{y}_{0:T}, \boldsymbol{\theta}) d\boldsymbol{x}_{0:T}.\end{aligned} \tag{3.4}$$

Since all terms can be computed with expectations with respect to $p(\boldsymbol{x}_{0:T} \mid \boldsymbol{\theta}, \boldsymbol{y}_{0:T})$, we can approximate these expectations by exhaustively calculating each of the N particle trajectories $\boldsymbol{x}_{0:T}$ in Algorithm 1. This however requires saving the entire history of the particle filter and also duplicates a number of computations. Algorithm 3 of [Poyiadjis et al., 2011] below performs the desired calculations while avoiding both of these pitfalls.

Algorithm 3 Particle Filter with Score and Hessian Estimates

```
$\boldsymbol{x}_0^{1:N} \overset{\text{iid}}{\sim} q_0(\boldsymbol{x}_0 \mid \boldsymbol{y}_0, \boldsymbol{\theta})$ // sample first time point
$w_0^i = g(\boldsymbol{y}_0 \mid \boldsymbol{x}_0, \boldsymbol{\theta})\pi(\boldsymbol{x}_0 \mid \boldsymbol{\theta})/q_0(\boldsymbol{x}_0 \mid \boldsymbol{y}_0, \boldsymbol{\theta})$
$\alpha_0^{1:N} \leftarrow 0$
$\beta_0^{1:N} \leftarrow 0$
for $t \geq 1$ do
    $v_{t-1}^i = w_{t-1}^i / \sum_{i=1}^N (\boldsymbol{w}_{t-1}^i)$
    $\boldsymbol{A}_t \sim \text{Multinomial}(N, \boldsymbol{v}_t)$
    $\boldsymbol{x}_t^i \overset{\text{ind}}{\sim} q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}^{A_i^t}, \boldsymbol{y}_t, \boldsymbol{\theta})$
    $w_t^i = g(\boldsymbol{y}_t \mid \boldsymbol{x}_t^i) f(\boldsymbol{x}_t^i \mid \boldsymbol{x}_{t-1}^{A_i^t}, \boldsymbol{\theta}) / q(\boldsymbol{x}_t^i \mid \boldsymbol{x}_{t-1}^{A_i^t}, \boldsymbol{y}_t, \boldsymbol{\theta})$
    $\boldsymbol{\alpha}_n^i = \boldsymbol{\alpha}_{n-1}^i + \nabla \log g(\boldsymbol{y}_n | \boldsymbol{x}_n^i) + \nabla \log f(\boldsymbol{x}_n^i | \boldsymbol{x}_{n-1}^{A_i^t})$
    $\boldsymbol{\beta}_n^i = \boldsymbol{\beta}_{n-1}^i + \nabla^2 \log g(\boldsymbol{y}_n | \boldsymbol{x}_n^i) + \nabla^2 \log f(\boldsymbol{x}_n^i | \boldsymbol{x}_{n-1}^{A_i^t})$
end
$\hat{\mathcal{L}}(\boldsymbol{\theta} \mid \boldsymbol{y}_{0:T}) = \prod_{t=0}^T \left( \frac{1}{N} \sum_{i=1}^N \boldsymbol{w}_t^i \right)$
$\boldsymbol{S}_T = \sum_{i=0}^T W_T^i \boldsymbol{\alpha}_T^i$
$\boldsymbol{\Sigma}_T = \boldsymbol{S}_T \boldsymbol{S}_T^T - \sum_{i=0}^N W_T^i (\boldsymbol{\alpha}_T^i \boldsymbol{\alpha}_T^{(i)T} + \boldsymbol{\beta}_T^i)$
Return $\hat{\mathcal{L}}(\boldsymbol{\theta} \mid \boldsymbol{y}_{0:T}), \boldsymbol{S}_T, \boldsymbol{\Sigma}_T$
```

These values of the score and hessian can then be used in a stochastic optimization routine or for comparison with closed form methods. The authors also elucidate about how there is a path degeneracy[Doucet et al., 2009] problem for algorithm 3. They provide an improved version of the algorithm for which we refer the reader to the original paper[Poyiadjis et al., 2011].

Chapter 4

Experimental Evaluation

In this section we will comment about different experiments that we conduct in order to obtain inference results for our proposed method. We will begin by describing a model (Brownian Motion with Drift) that will serve as a benchmark for experiments involving identifying biased gradients and then we proceed to further experiments for different models.

4.1 Brownian Motion with Drift

The model is

$$\begin{aligned} x_0 &\sim \pi(x_0) \propto 1 \\ x_t &\sim \mathcal{N}(x_{t-1} + \mu\Delta t, \sigma^2 \Delta t) \\ y_t &\sim \mathcal{N}(x_t, \tau^2), \end{aligned} \tag{4.1}$$

with $\boldsymbol{\theta} = (\mu, \sigma, \tau)$. Note that with $\pi(x_0) \propto 1$, we may condition on y_0 and obtain $x_0 \mid y_0 \sim \mathcal{N}(y_0, \tau^2)$. In this case, the marginal likelihood is defined as

$$\mathcal{L}(\boldsymbol{\theta}) = p(y_{1:T} \mid y_0, \boldsymbol{\theta}) \tag{4.2}$$

rather than by $p(y_{0:T} \mid \boldsymbol{\theta})$. The reason is that the latter expression requires one to integrate over $x_0 \sim \pi(x_0)$, which can only be done when $\pi(x_0)$ is a proper prior. However, for our choice of $\pi(x_0) \propto 1$ this is not the case. On the other hand, $p(y_{1:T} \mid y_0, \boldsymbol{\theta})$ only requires us to integrate over $p(x_0 \mid y_0, \boldsymbol{\theta})$, which only requires this distribution to be proper, as is the case here.

Data and Latent Variable

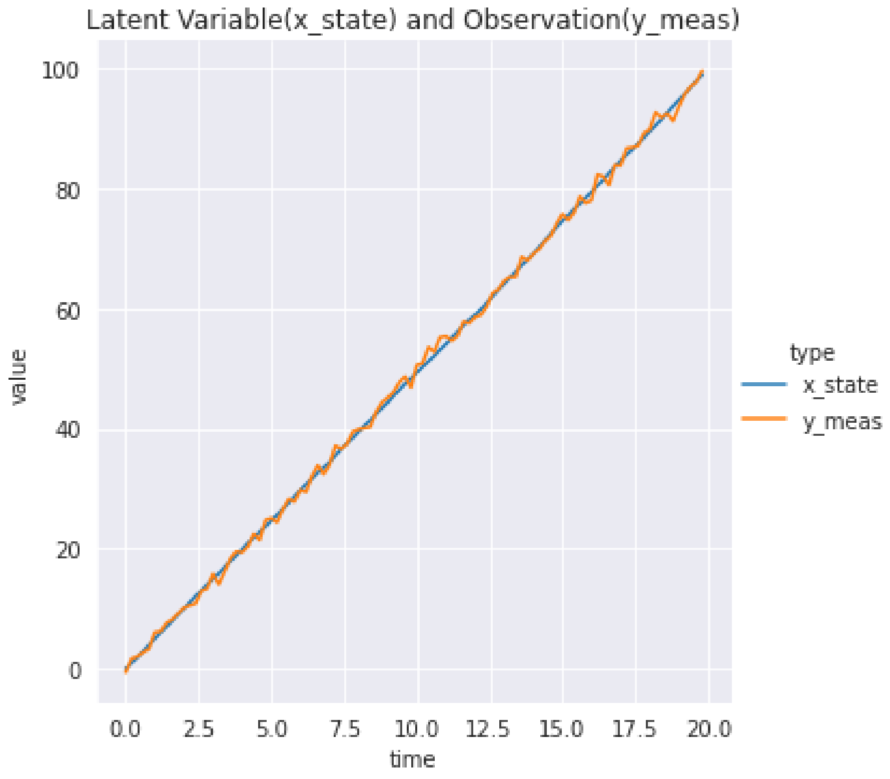

Figure 4.1: The x-axis is the time and the y-axis is the value of the variable at different points in time. We should note that the observations are made under noise.

Brownian motion with drift is a simple model that will serve as a benchmark for further experiments. Note that in this particular model, we have access to the closed form of $\mathcal{L}(\boldsymbol{\theta})$ (derived in Appendix A). Since our method is an approximation method, it will serve as a reasonable sanity check and debugging tool to ensure that our method is not only technically sound, but also empirically verifiable.

In this chapter we empirically evaluate the methods discussed in Chapter 3. However, before proceeding with the simulation studies, we briefly detour to evaluate the bias present in the two methods that we consider in Chapter 3.

4.1.1 Bias of Gradients

This section describes in detail the procedure we employ to determine the bias present in different methods we may use to compute gradients and hessians. We use multiple trials of the two methods, namely, taking the gradient of the particle filter using automatic differentiation versus the approximation of the score and hessian methods [Poyiadjis et al., 2011]. In order to do so, we compare three methods given below:

- Accumulator Method: Using the α, β terms which are used to estimate the score (first derivative) and the hessian (the second derivative) information.
- Auto Method: This method involves differentiating with respect to the particle filter and obtaining the results for the score and hessian.
- Analytic Method: This involves computing the gradient and hessian of the exact likelihood (4.2) via automatic differentiation.

We compare the scores and hessians derived from these three methods in order to see whether our estimators for the inferred parameters are unbiased. We note that the plots below confirm our hypothesis that they are in fact biased for the Auto method. Particularly, we also find out that the accumulator method is an unbiased estimate of the score. To generate the plots below, we run the experiments for either 10 or 100 observations and for $100, 1000, 10000$ particles.

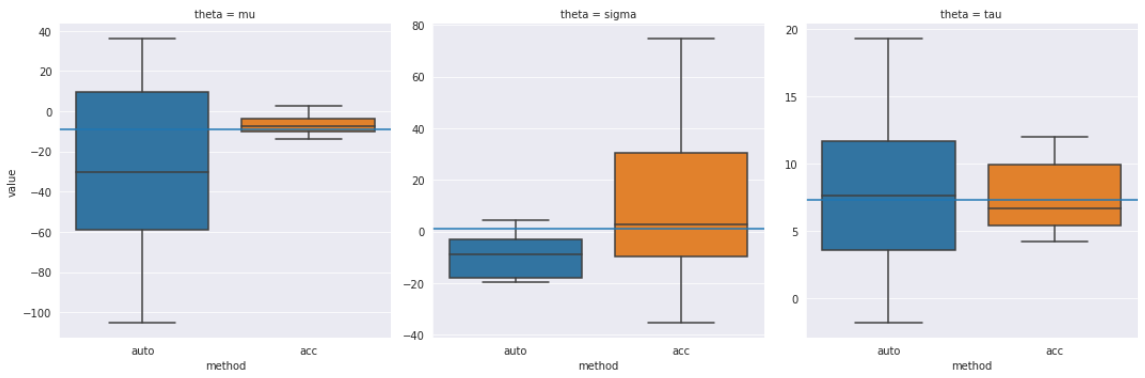

Figure 4.2: The estimates of the gradient for 10 observations and 100 particles. The line in red represents the analytic value of the score function.

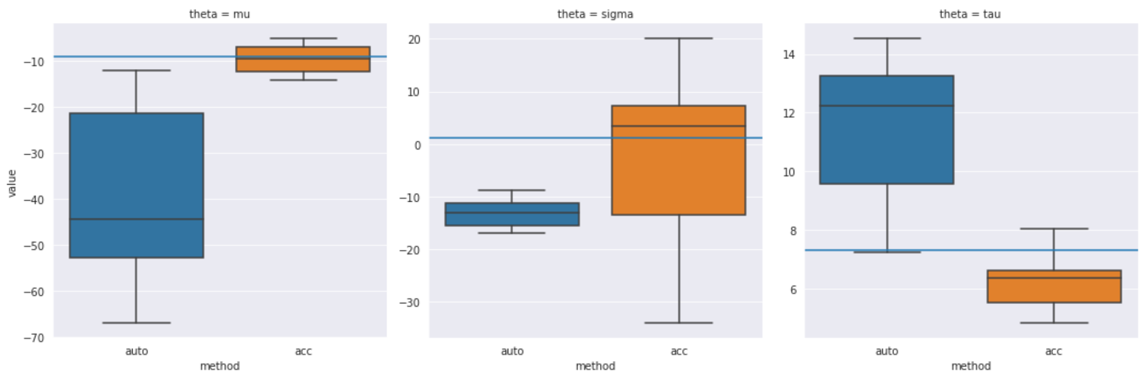

Figure 4.3: The estimates of the gradient for 10 observations and 1000 particles. The line in red represents the analytic value of the score function.

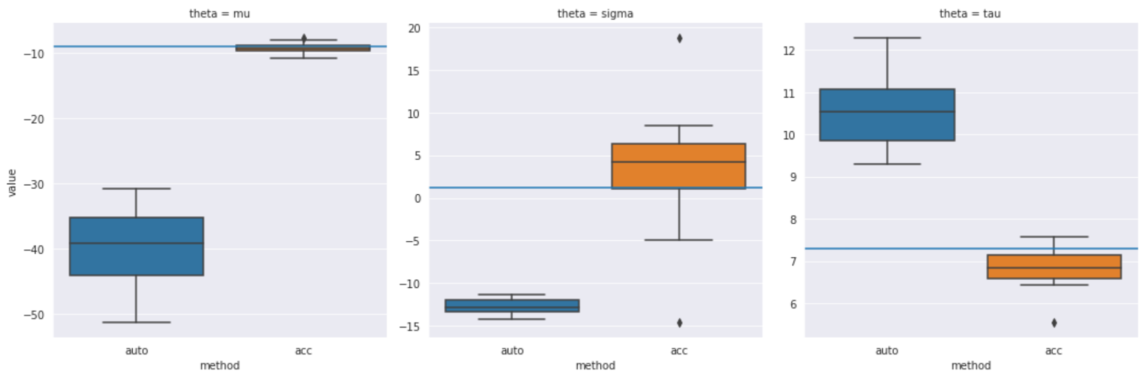

Figure 4.4: The estimates of the gradient for 10 observations and 10000 particles. The line in red represents the analytic value of the score function.

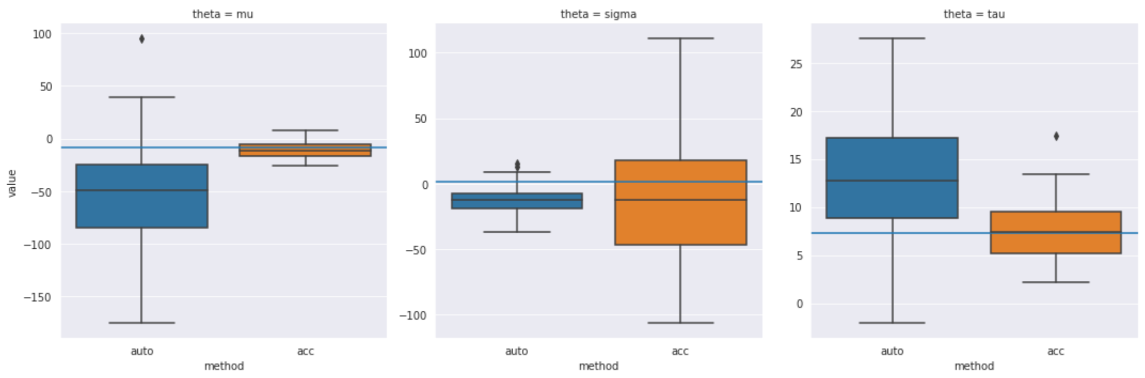

Figure 4.5: The estimates of the gradient for 100 observations and 100 particles. The line in red represents the analytic value of the score function.

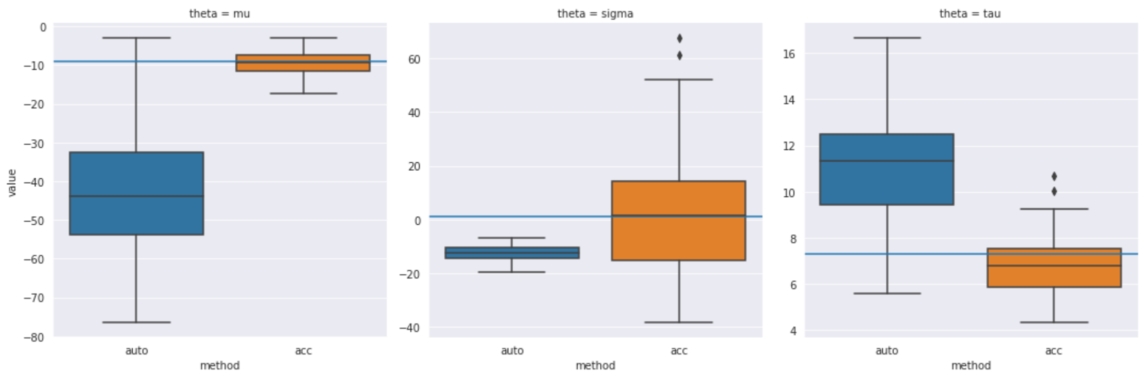

Figure 4.6: The estimates of the gradient for 100 observations and 1000 particles. The line in red represents the analytic value of the score function.

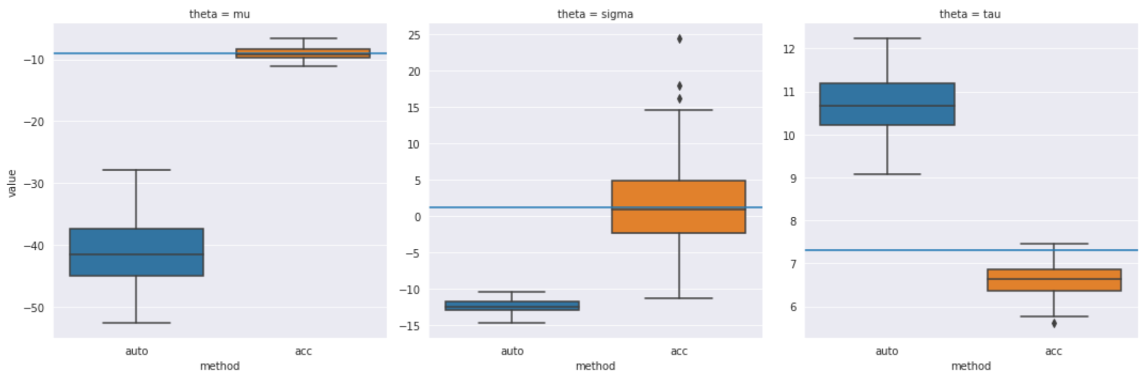

Figure 4.7: The estimates of the gradient for 100 observations and 10000 particles. The line in red represents the analytic value of the score function.

Now that we know that our score estimates derived from the accumulator method, we have some confirmation that our estimates of the score that we obtain using that method is unbiased. We benchmark these results of the score estimates obtained with the accumulator method against those obtained by the automatic differentiation method. Next, we compute the hessians of both the methods that we are benchmarking and compare it against the analytic solution.

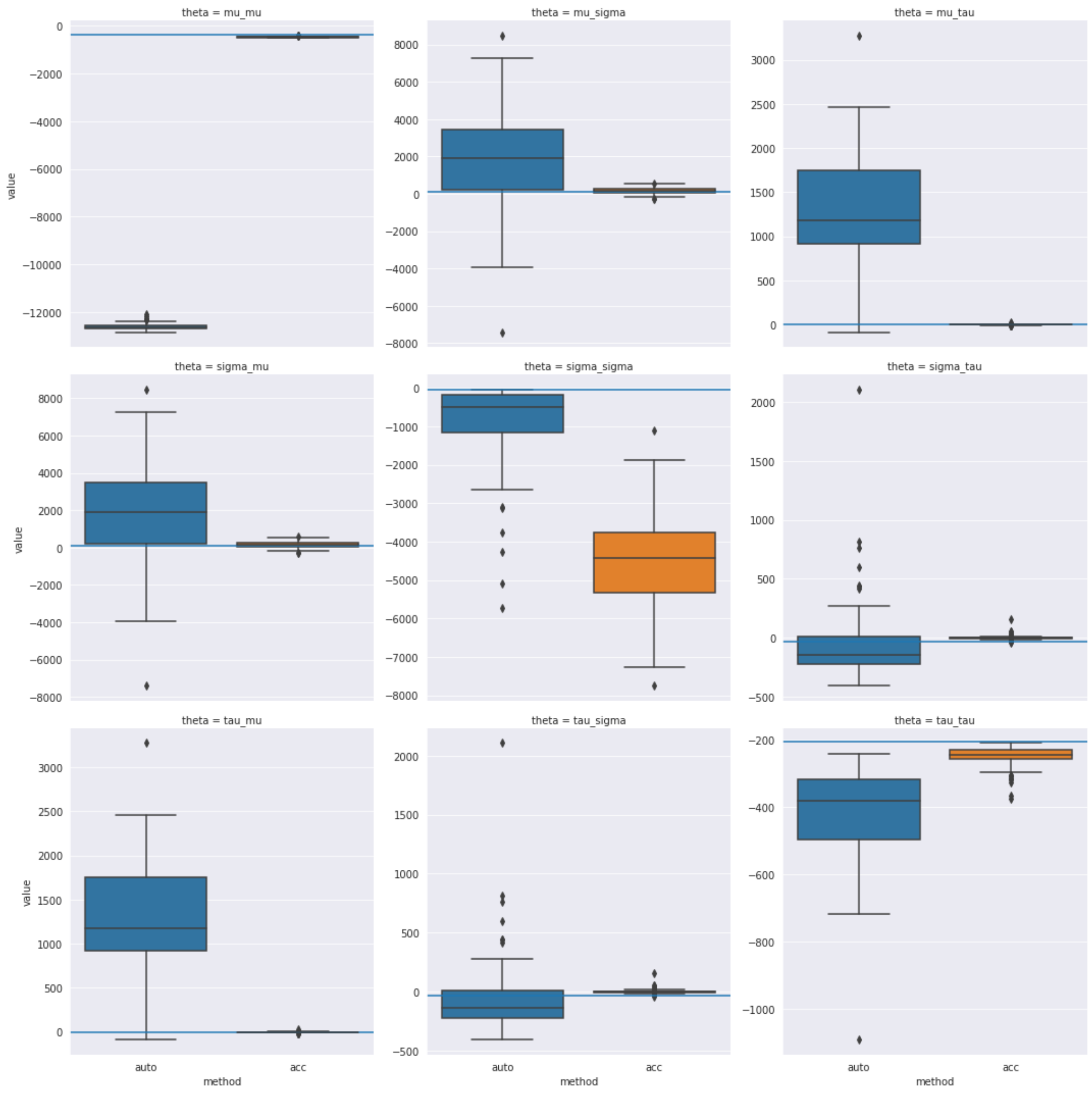

Figure 4.8: The estimates of the hessian for 10 observations and 100 particles. The line in red represents the analytic estimate of the hessian element.

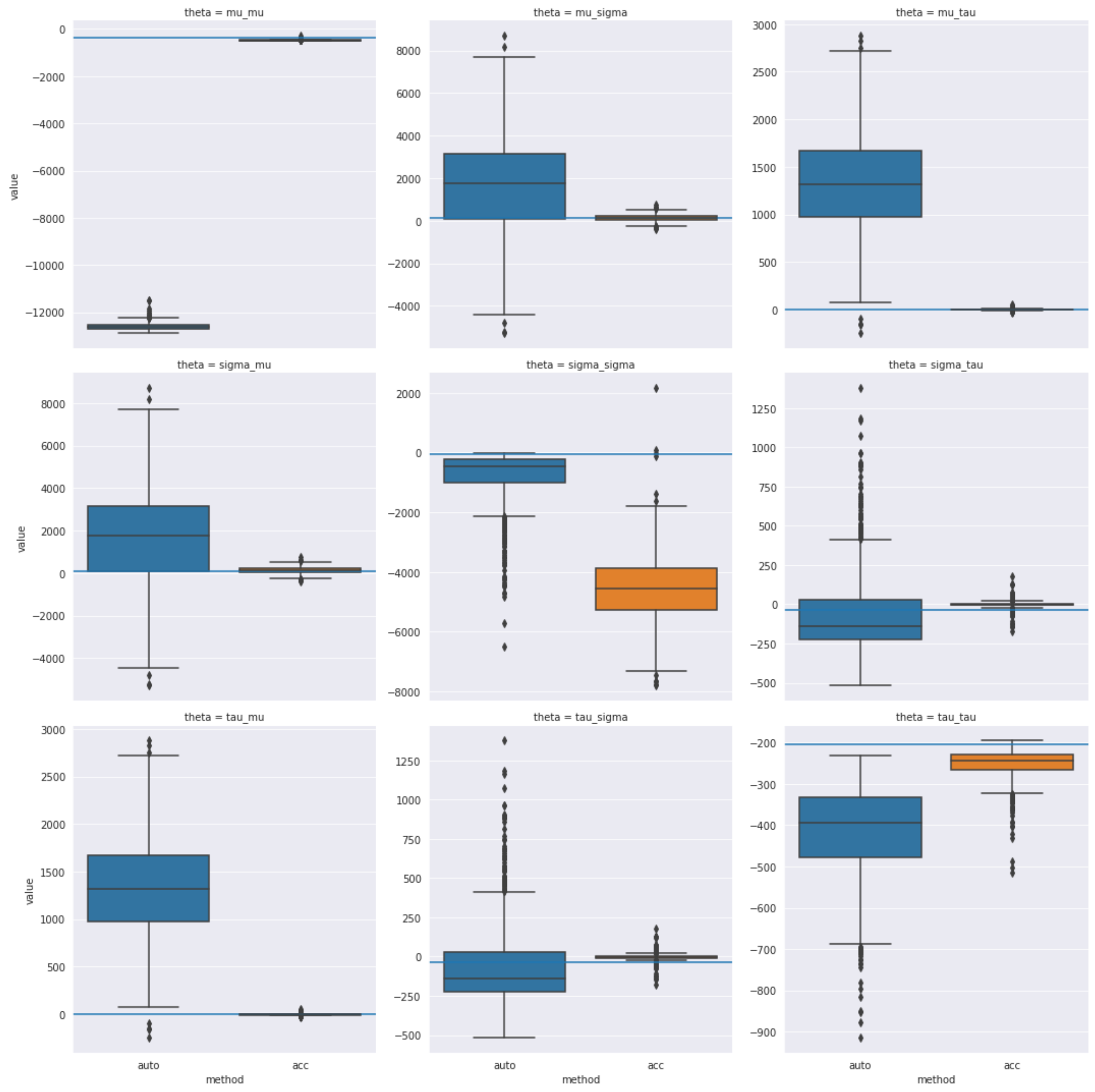

Figure 4.9: The estimates of the hessian for 10 observations and 1000 particles. The line in red represents the analytic estimate of the hessian element.

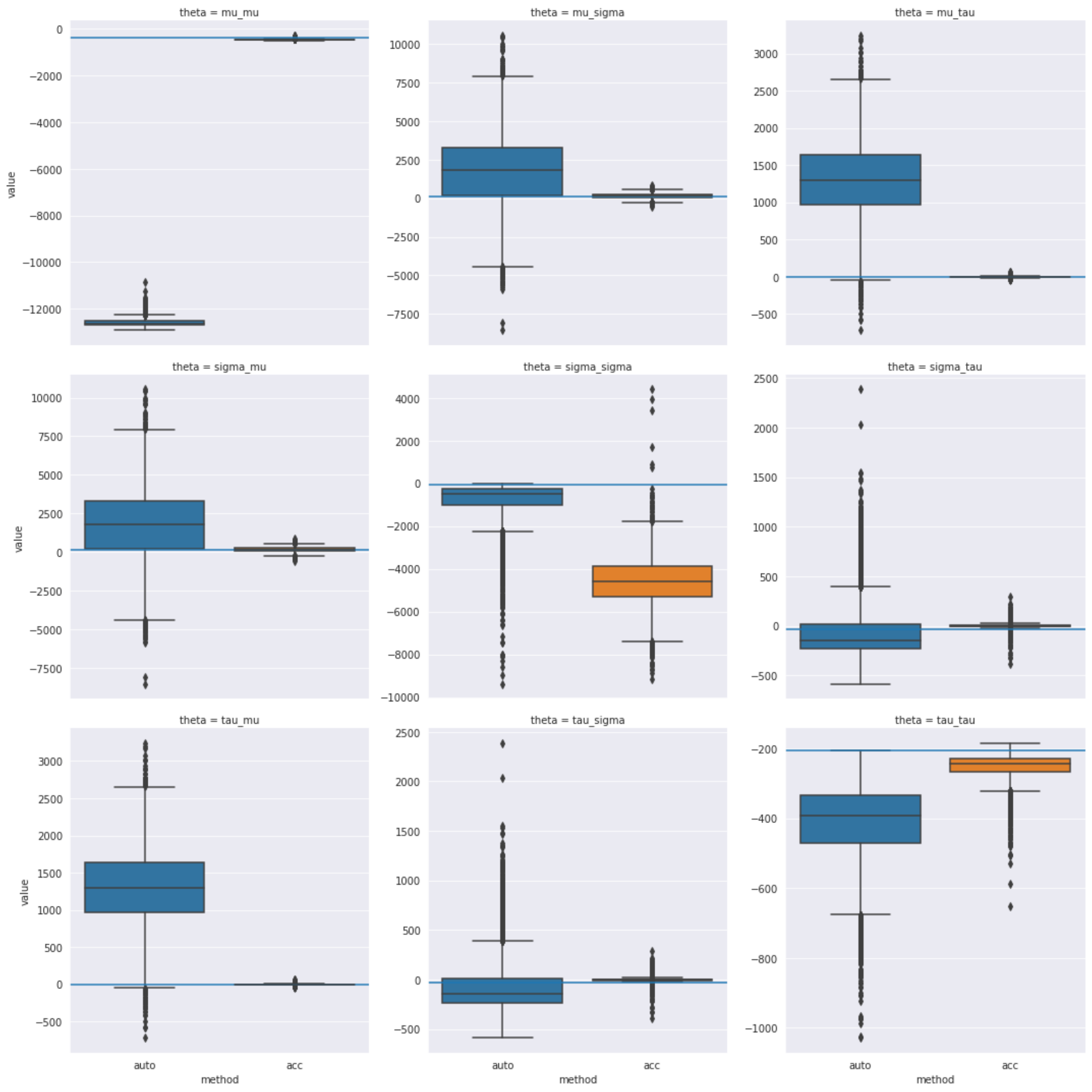

Figure 4.10: The estimates of the hessian for 10 observations and 10000 particles. The line in red represents the analytic estimate of the hessian element.

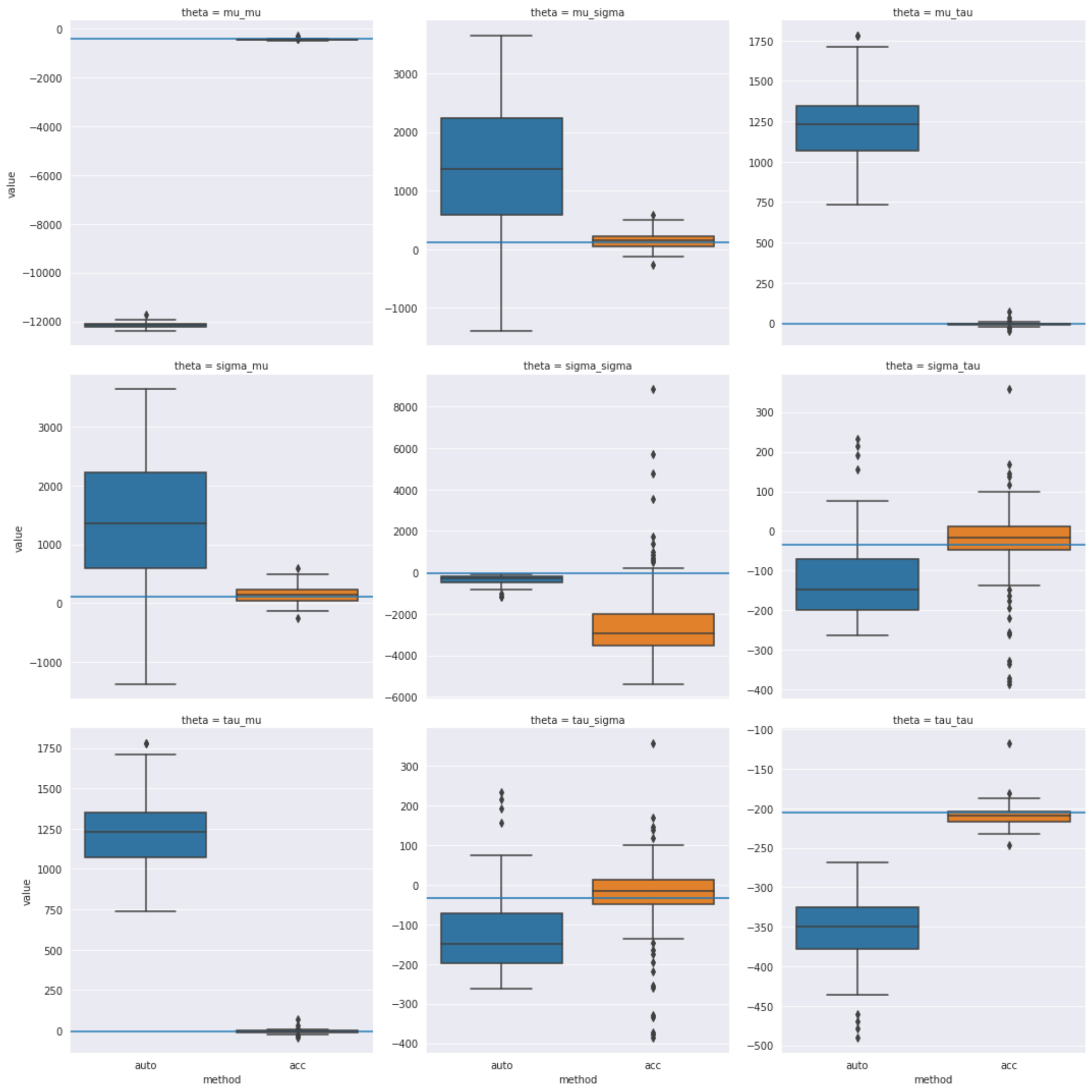

Figure 4.11: The estimates of the hessian for 100 observations and 100 particles. The line in red represents the analytic estimate of the hessian element.

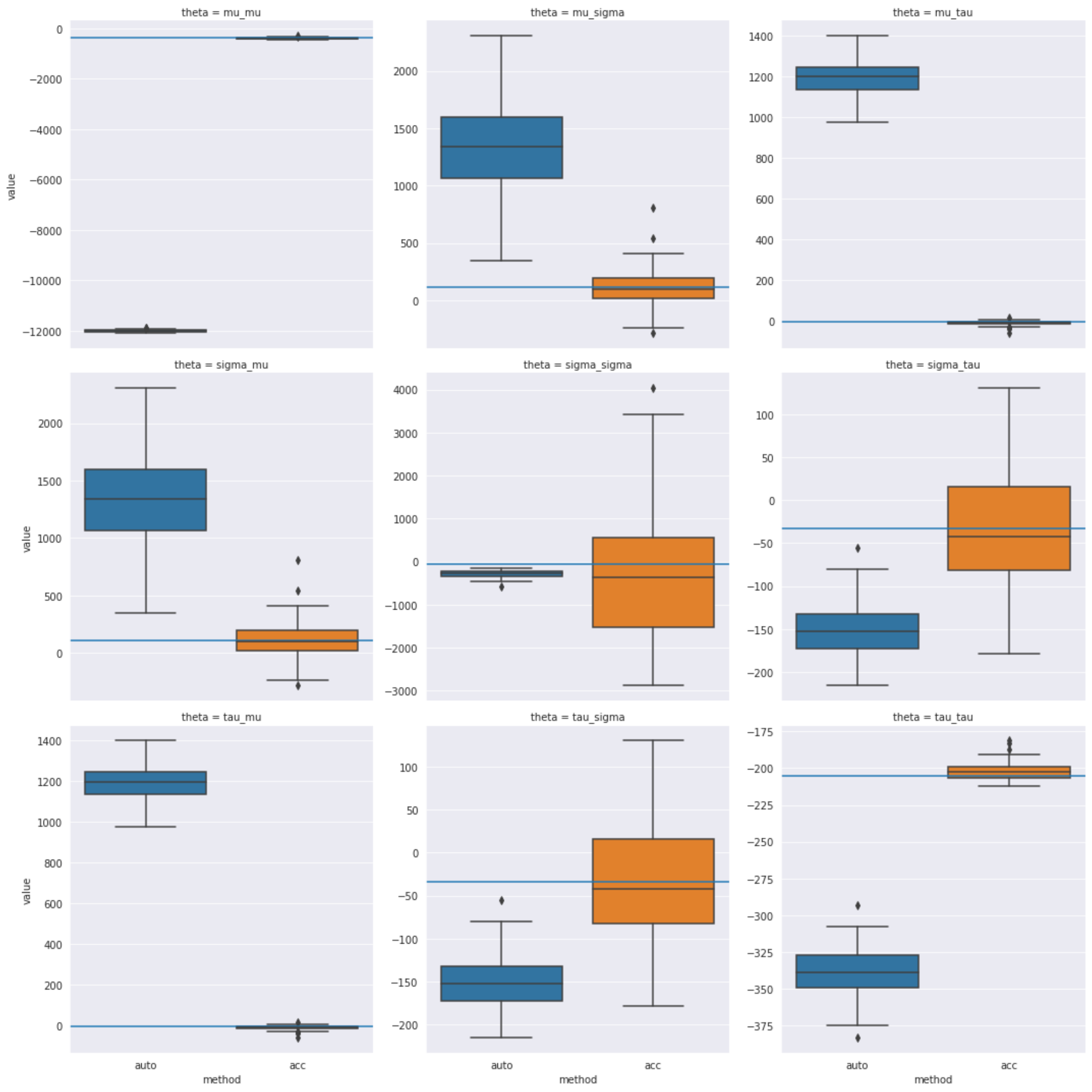

Figure 4.12: The estimates of the hessian for 100 observations and 1000 particles. The line in red represents the analytic estimate of the hessian element.

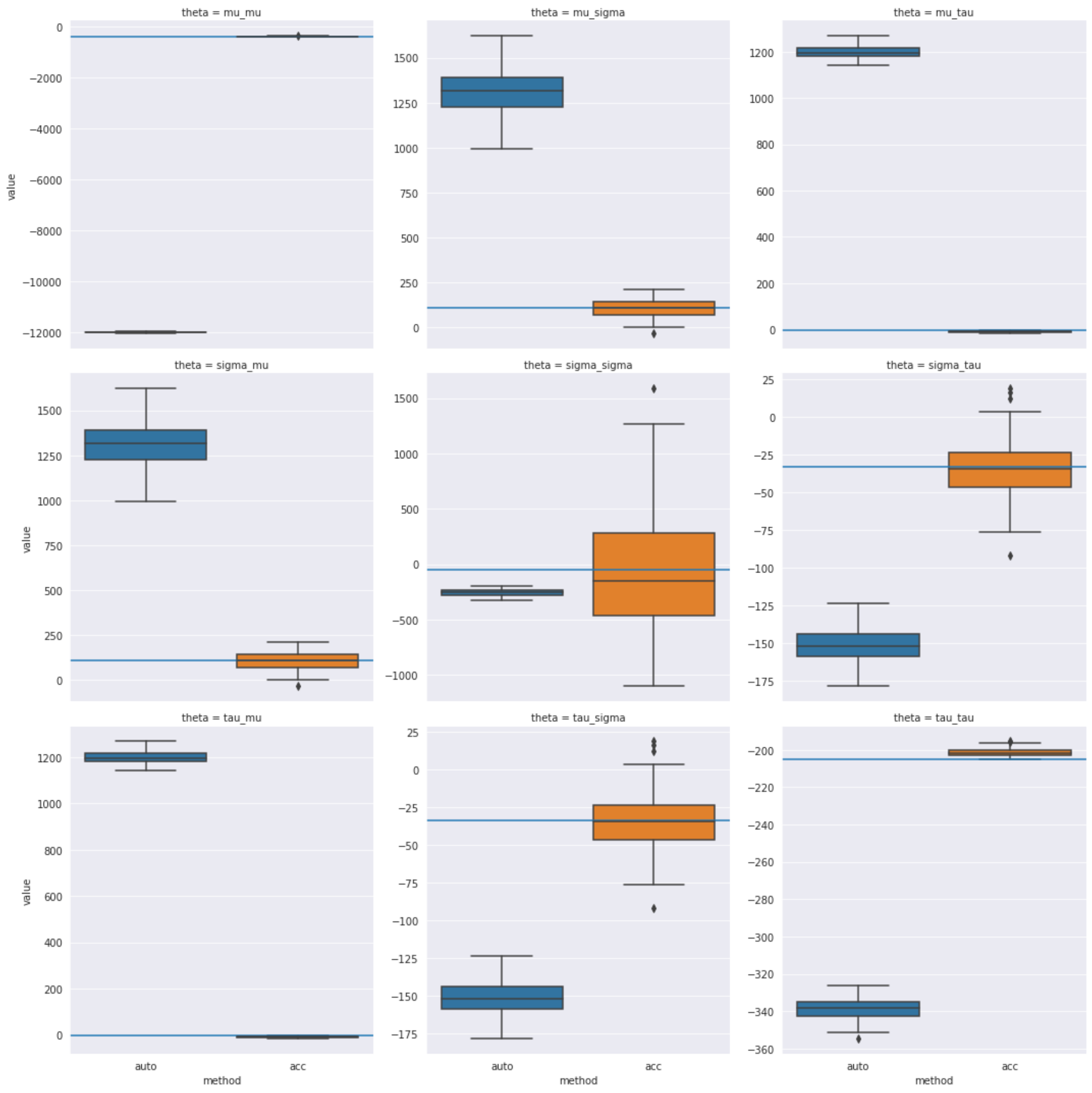

Figure 4.13: The estimates of the hessian for 100 observations and 10000 particles. The line in red represents the analytic estimate of the hessian element.

We observe that the accumulator method significantly outperforms the auto method in terms of bias of the score and hessian and therefore, we use it in our results. We also note that the hessian being symmetric implies that the lower triangular elements (excluding the diagonal itself) is the same as its symmetric counterpart. Any differences seen between them can be attributed to a different key being used for them. Thus, for the remainder of this section, we utilize the accumulator method to obtain our results since it is unbiased.

We now outline the projection plots of the data. The projection plots are obtained by holding all other variables constant at the value it was initialized to in the simulation and varying the single variable of interest (for instance, μ). The projection plots are demonstrable convex when all other variables are held constant, however, one would expect that as we optimize over the multidimensional vector $\boldsymbol{\theta}$, the problem would no longer be convex. The y-axis on 4.14 is the exact likelihood, which we have access to for this particular model (since it is contains a closed form likelihood). There is also no randomness that is present in the generation of 4.14.

Projection Plot BM Model

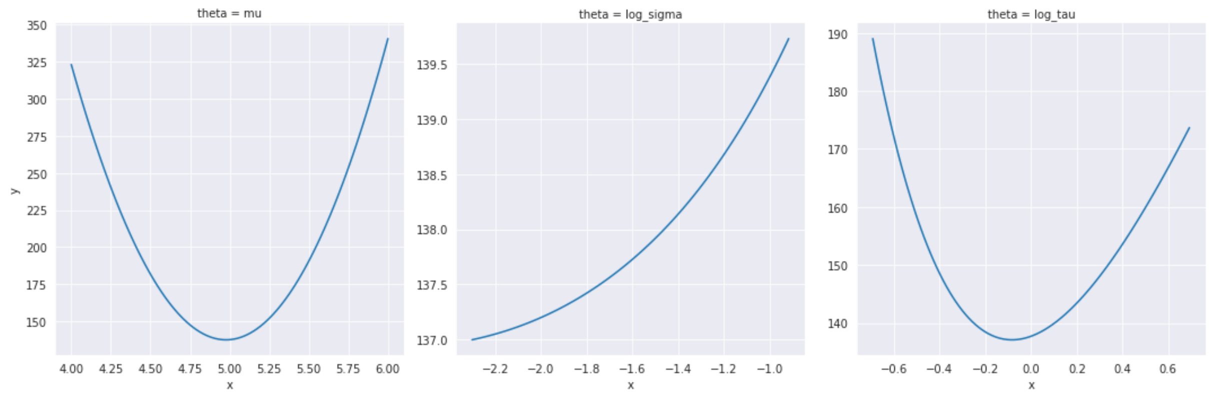

Figure 4.14: The y-axis is the exact likelihood and the x-axis is the different values for the particular parameter. It is important to note that each of these plots is a 1-dimensional representation of the parameter, holding other parameters constant.

Figure 4.15 is obtained by instead using the particle filter to obtain an estimate of the log-likelihood instead of utilizing the exact likelihood. Thus, we obtain a stochastic estimate of the 1-dimensional projection plots where the stochasticity is governed by the generation of the random keys. It is important to note that as the number of particles approaches infinity, we would observe that Figure 4.15 converges to 4.14. As we can see from both Figure 4.14 and 4.15, while the extremities of the plots are not entirely the same, the models of the plot are located in roughly the same region. It is also important to note that viewing a subsection of the plot under a more fine lens would reveal that there is a great deal of local non-convexity.

Projection Plot BM Model with the Particle Filter

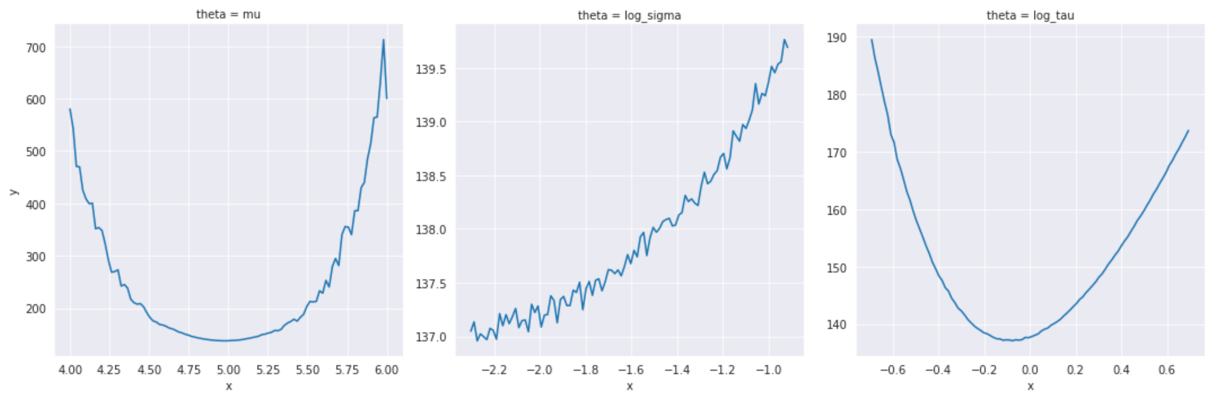

Figure 4.15: The y-axis is the approximate log likelihood obtained using the particle filter and the x-axis is the same as Figure 4.14. We note that certain parameters are more non-convex (that is to say that they have several regions of non-convexity) compared to others.

4.1.2 Inference for Brownmian Motion with Drift

Now, we estimate the parameters for the model and perform inference for the same. We run the stochastic optimization algorithm for 1000 iterations with 500 particles with a learning rate of 0.01 and obtain our converged estimates for the score and information. We use a starting value of $[1.0, 1.0, 1.0]$ for the parameters.

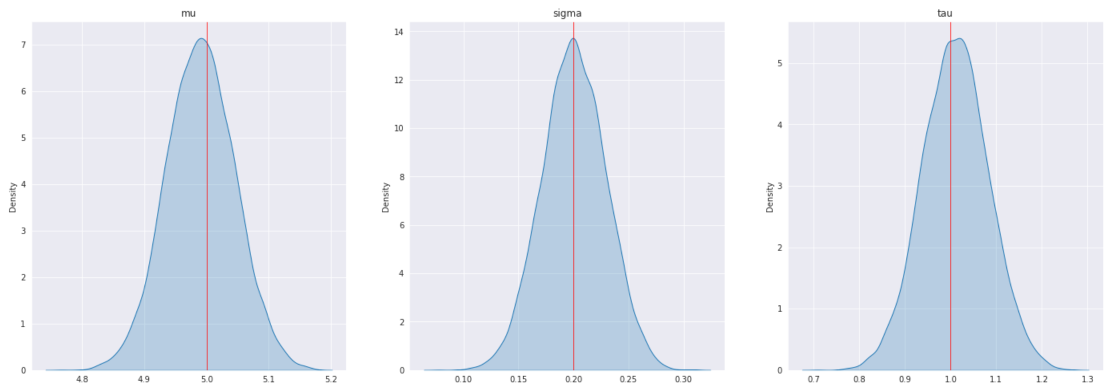

Figure 4.16: Inference for the brownian motion with drift model. The red line corresponds to the true parameter values that we use for the simulation.

4.2 Lotka-Volterra with Additive Noise

The Lotka-Volterra model [Wangersky, 1978, Lotka, 1925, Lotka, 1920] is a set of ordinary differential equations that is supposed to model the growth of predator and prey in a given population. The equations typically model the change in the prey and predator populations as a function of the interaction between the two. In particular, if the predator and the prey interact, the total number of prey will reduce. There has been a plethora of work done on the model which takes into consideration some of the assumptions present in the Lotka-Volterra models. Additionally, if left alone, the predators will become extinct, and similarly if left alone, the prey subpopulation will continue to grow.

In this example, we are interested in the ODE version of the Lotka-Volterra model, which in terms of the prey variable x_H and the predator variable x_L is given by

$$\begin{aligned} \frac{dx_H}{dt} &= \alpha x_H - \beta x_H x_L, \\ \frac{dx_L}{dt} &= \delta x_H x_L - \gamma x_L. \end{aligned} \tag{4.3}$$

Since we have $x_L, x_H > 0$, it is convenient to rewrite the ODE in terms of $X_H = \log(x_H)$ and $x_L = \log(x_L)$:

$$\begin{aligned} \frac{dX_H}{dt} &= \alpha - \beta \exp(X_L), \\ \frac{dX_L}{dt} &= \delta \exp(x_H) - \gamma. \end{aligned} \tag{4.4}$$

The measurement model is on $\boldsymbol{Y}_n = (Y_n^H, Y_n^L)$ and is given by

$$\begin{aligned} Y_n^H &\overset{\text{ind}}{\sim} \mathcal{N}(\exp(X_H(n\Delta t)), \tau_H^2) \\ Y_n^L &\overset{\text{ind}}{\sim} \mathcal{N}(\exp(X_L(n\Delta t)), \tau_L^2). \end{aligned} \tag{4.5}$$

The parameters of the model are thus $\boldsymbol{\theta} = (\alpha, \beta, \gamma, \delta, \sigma_H, \sigma_L, \tau_H, \tau_L)$.

Inference here can be conducted by approximately solving the ODE using numerical methods. Typically this is done using deterministic solvers. Here however we wish to explore a simple stochastic solver which transforms the ODE in to the SDE model on $\boldsymbol{X}(t) = (X_H(t), X_L(t))$ given by

$$\begin{aligned} \mathrm{d}X_L(t) &= (\alpha - \beta \exp(X_L(t)))\, \mathrm{d}t + \sigma_H \, \mathrm{d}B_H(t) \\ \mathrm{d}X_H(t) &= (-\gamma + \delta \exp(X_H(t)))\, \mathrm{d}t + \sigma_L \, \mathrm{d}B_L(t). \end{aligned} \tag{4.6}$$

4.2.1 Prior Distribution

Let $Z_i = \exp(X_0^i)$, $i = H, L$, and $\boldsymbol{Z}_0 = (Z_H, Z_L)$. Then for the choice of prior

$$\boldsymbol{Z}_0 \sim \pi(\boldsymbol{Z}_0) \propto 1, \tag{4.7}$$

we have

$$Z_i \mid \boldsymbol{Y}_0, \boldsymbol{\theta} \overset{\text{ind}}{\sim} \text{TruncatedNormal}(Y_0^i, \tau_i^2)), \tag{4.8}$$

where the truncation is for $Z_i > 0$. Thus, we can sample implement a perfect importance sampler by sampling directly from $p(\boldsymbol{x}_0 \mid \boldsymbol{y}_0, \boldsymbol{\theta})$. While the corresponding log-weights are constant, they are not equal to zero. That is, the weight of particle $\boldsymbol{x}_0$ is given by

$$w(\boldsymbol{x}_0) = \frac{\pi(\boldsymbol{x}_0 \mid \boldsymbol{\theta}) p(\boldsymbol{y}_0 \mid \boldsymbol{x}_0, \boldsymbol{\theta})}{q(\boldsymbol{x}_0 \mid \boldsymbol{y}_0, \boldsymbol{\theta})}, \tag{4.9}$$

where for the perfect importance sampler we have the proposal distribution

$$q(\boldsymbol{x}_0 \mid \boldsymbol{y}_0, \boldsymbol{\theta}) = p(\boldsymbol{x}_0 \mid \boldsymbol{y}_0, \boldsymbol{\theta}) = \frac{\pi(\boldsymbol{x}_0 \mid \boldsymbol{\theta}) p(\boldsymbol{y}_0 \mid \boldsymbol{x}_0, \boldsymbol{\theta})}{\int \pi(\boldsymbol{x}_0 \mid \boldsymbol{\theta}) p(\boldsymbol{y}_0 \mid \boldsymbol{x}_0, \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{x}_0}, \tag{4.10}$$

such that $w(\boldsymbol{x}_0) = \int \pi(\boldsymbol{x}_0 \mid \boldsymbol{\theta}) p(\boldsymbol{y}_0 \mid \boldsymbol{x}_0, \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{x}_0$. In this particular case, this boils down to

$$w(\boldsymbol{x}_0) = \prod_{i=H,L} \int_0^\infty \phi((Z_i - Y_0^i)/\tau_i)/\tau_i \, \mathrm{d}Z_i = \prod_{i=H,L} \Phi(Y_0^i/\tau_i), \tag{4.11}$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ are the PDF and CDF of $\mathcal{N}(0, 1)$.

4.2.2 Results

In this section, we provide all of the results for the simulation studies involving the Lotka-Volterra model which is derived from an underlying ODE.

Data for Lotka-Volterra Model with Particle Filter

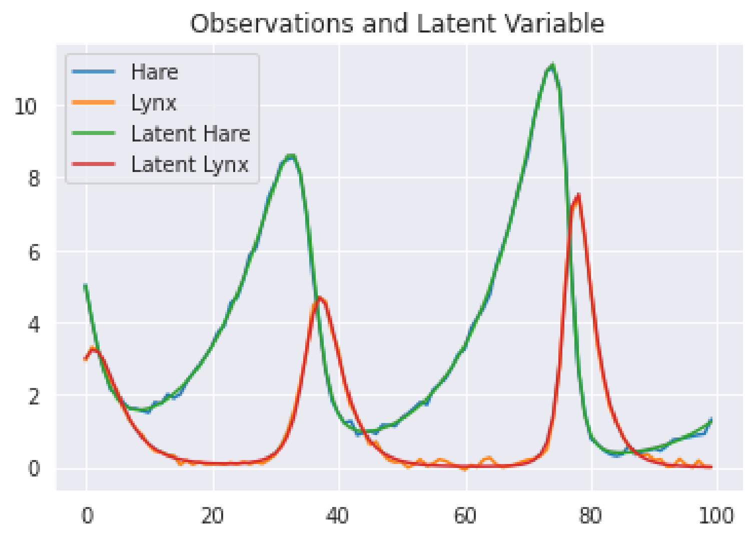

Figure 4.17: This figure consists of the observations and the latent variable that are sampled from the model.

We then proceed to display the projection plots for the different parameters. For these plots we set the bounds for each of the variables to be: $[0.1, 2.0]$, $[0.1, 2.0]$, $[3.0, 5.0]$, $[0.1, 2.0]$, $[0.01, 4.]$, $[0.01, 4.]$, $[0.01, 2.]$, $[0.01, 2.]$ for $\Theta = [\alpha, \beta, \gamma, \delta, \sigma_h, \sigma_l, \tau_h, \tau_l]$ and we plot 100 points in that range for each parameter. It should be noted that much like the previous projection plots, while plotting θ_i, we hold all θ_j such that $j \neq i$ constant at its true parameter value. For the projection plots, use a resolution number of 2. It should be noted that the higher the resolution number the more fine-grained the approximation is and as a result, we expect our stochastic optimization to perform better.

Projection Plot Lotka-Volterra Model with Particle Filter

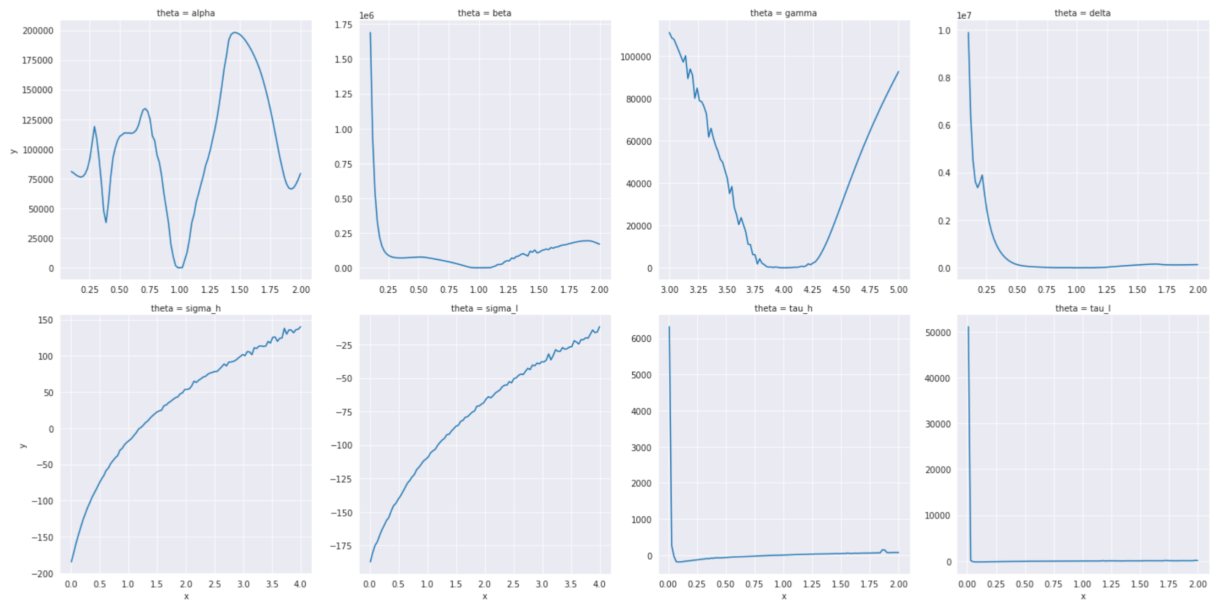

Figure 4.18: This figure consists of the projection plot for all parameters in the Lotka-Volterra model in the low-noise setting. The resolution number for this particular plot is 2.

The plots seem amenable to the accumulator method in general. We should note that for this particular example, while we have σ_h and σ_l as parameters within our stochastic optimization routine, we do not remove them at the time of parameter estimation.This set of experiments with the Lotka-Volterra Model has $n_{\text{res}} = [2, 4, 8, 16]$, the number of observations is 100, $dt = 0.1$ and $\Theta = [\alpha, \beta, \gamma, \delta, \sigma_h, \sigma_l, \tau_h, \tau_l] = [1.0, 1.0, 4.0, 1.0, 0.0001, 0.0001, 0.1, 0.1]$. This is the low-noise setting for the Lotka-Volterra model. For the inference results, we use the Adam [Kingma and Ba, 2014] optimizer with a learning rate of 0.01 and it is initialized to be all $[1.0, 1.0, 4.0, 1.0, 0.0001, 0.0001, 0.1, 0.1]$ on the log scale. We run these experiments for 100 iterations. Theoretically, we must note that the σ_l and σ_h parameters are almost exactly zero in our setting. However, for the purposes of simulation, we set it to be an extremely small value and compute the results. For this particular problem, we initialize the problem at the true values used for simulation.

To obtain our results for stochastic optimization, we run the accumulator algorithm 3 with 500 particles for 1000 iterations and with a step size of 0.01. This yields the point estimates for our parameters. To obtain the inference plots, we take the estimate of the hessian that we derive. We also assume for simplicity that our parameters are distributed as a $\mathcal{N}(\hat{\theta}, \hat{\Sigma})$ where $\hat{\theta}$ is the estimate of our stochastic optimization procedure. In totality, the stochastic estimates from the optimization procedure take around 33s to run for 1000 iterations.

Density Lotka-Volterra Model with Particle Filter

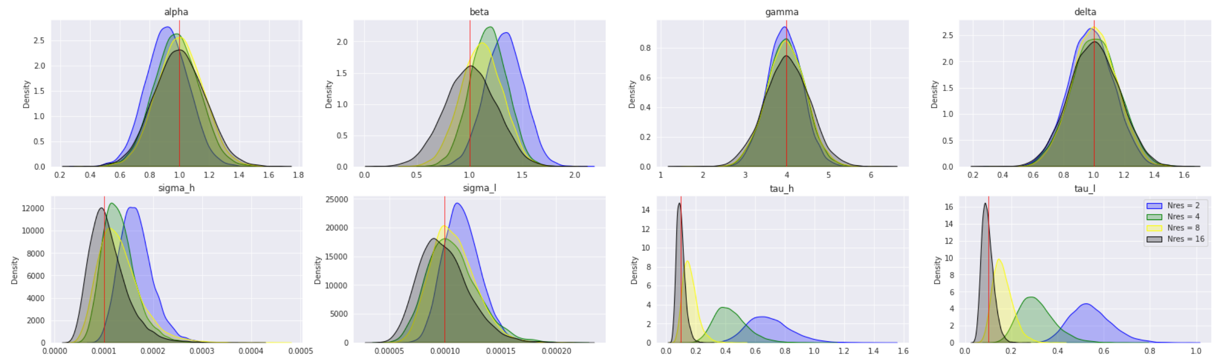

Figure 4.19: The true value of the parameters which is obtained from the simulation is the line in red in the plots and the density is obtained from our samples. We also transform the variables to their appropriate scale. The lines in red correspond to the true values of the parameters used to simulate the data in the low-noise setting.

Next, we analyze the same problem but for $\tau_l = 0.25$ and $\tau_h = 0.25$ which is termed as the high noise setting for the problem.

Lotka-Volterra Model Data - High Noise Setting

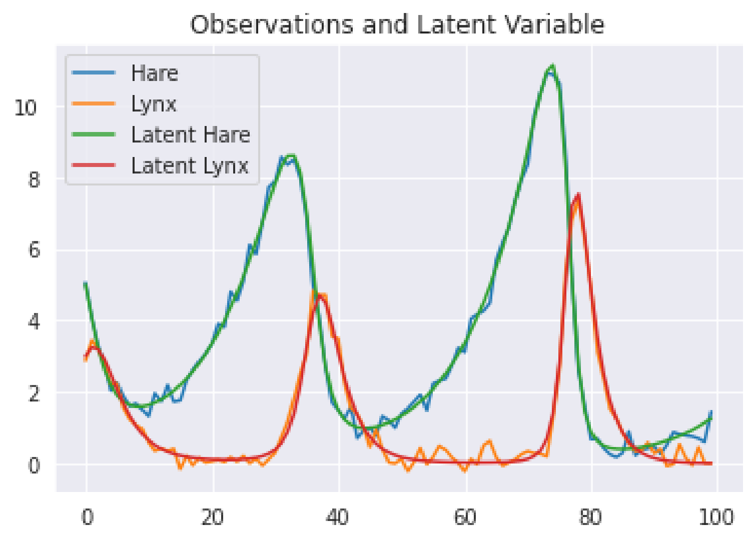

Figure 4.20: This figure consists of the observations and the latent variable that are sampled from the model.

As we see in figure 4.20, the data has more noise in it compared to the low noise setting.

Projection Plot Lotka-Volterra Model with Particle Filter

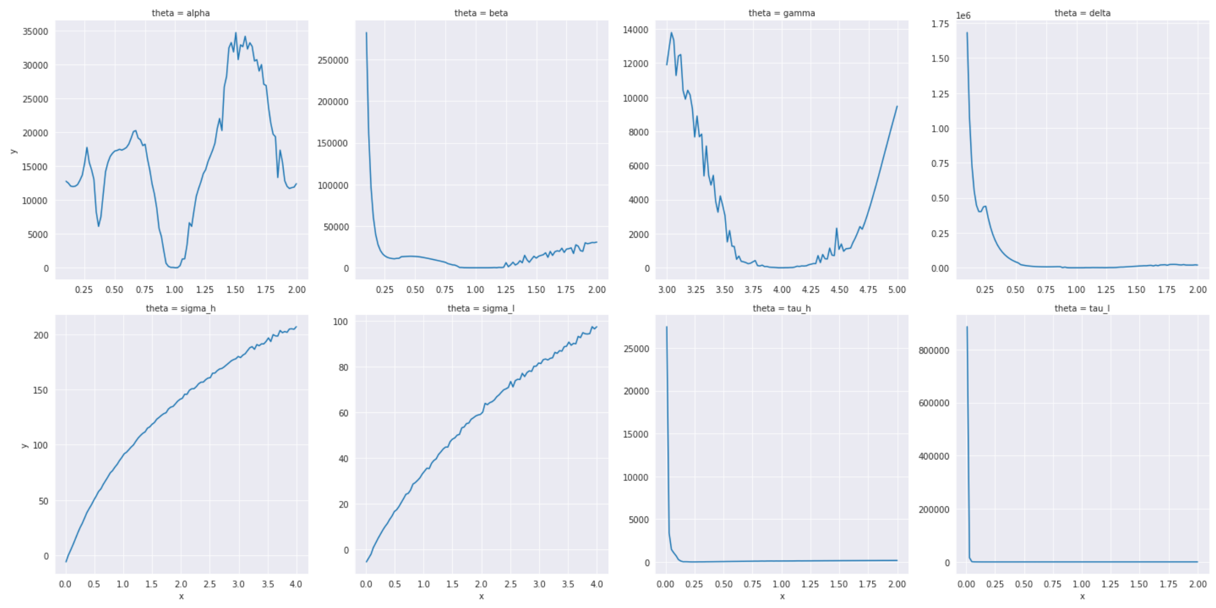

Figure 4.21: This figure consists of the projection plot for all parameters in the Lotka-Volterra model in the high-noise setting. The resolution number for this plot is 2.

In this case as well, the projection plots seem reasonable as we would expect. We do notice some slight deviation in the values of γ and β that are noticeable.

Density Plot Lotka-Volterra Model for parameters

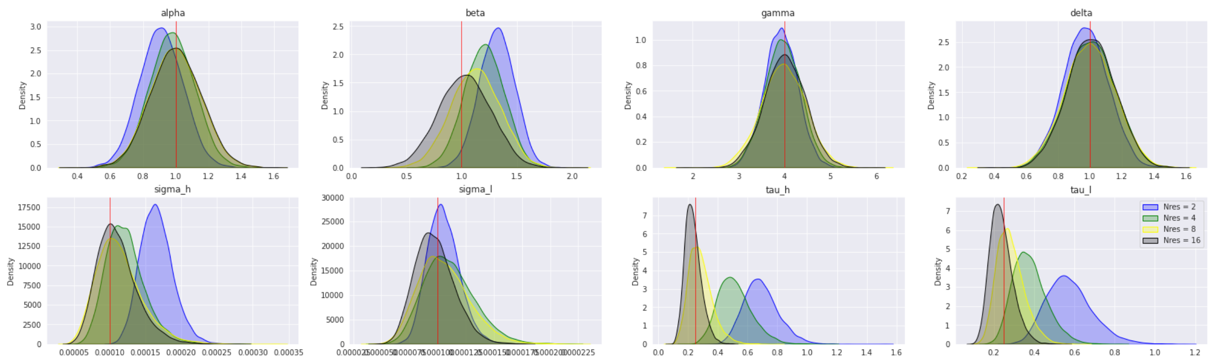

Figure 4.22: The true value of the parameters which is obtained from the simulation is the line in red in the plots and the density is obtained from our samples. We also transform the variables to their appropriate scale. The lines in red correspond to the true values of the parameters used to simulate the data in the high-noise setting. The density for resolution number 2 is blue, 4 is green, 8 is yellow and 16 is black

Now that we have obtained our results for the Lotka-Volterra model in both the high and low-noise settings, we proceed to sampling data from the underlying ordinary differential equation that we use to obtain the stochastic differential equation. Once again, we produce the projection plots and the inference results. Once again, we categorize it into the high and low noise settings. We first sample 21 observations in the low noise setting and use it as the data for our model. That is to say that in this setting, we discard the simulated y_i and rely on the estimates of the state variables.

Projection Plot Lotka-Volterra Model with Particle Filter

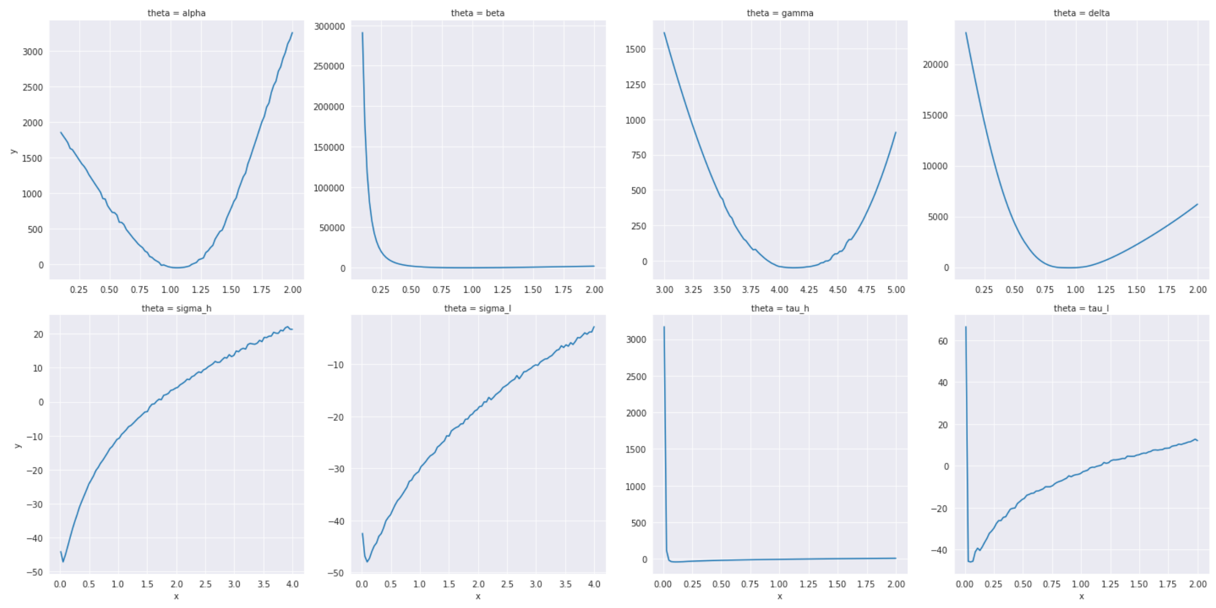

Figure 4.23: This figure consists of the projection plot for all parameters in the Lotka-Volterra model in the low-noise setting with data simulated from the underlying ODE. The resolution number used for this plot is 2.

Next, we plot the density for the parameters that we wish to infer and obtain the variance by adding a small additive constant to the diagonal to ensure positive definiteness.

Density Plot Lotka-Volterra Model for parameters

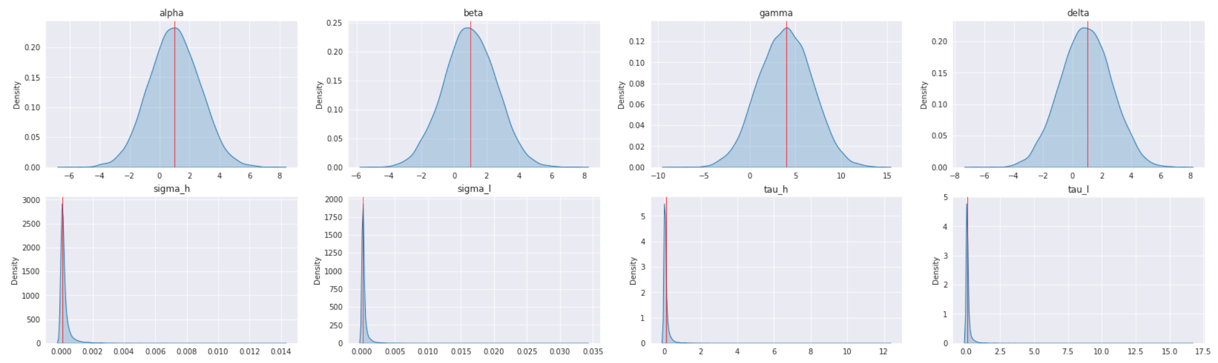

Figure 4.24: The true value of the parameters which is obtained from the simulation is the line in red in the plots and the density is obtained from our samples. We also transform the variables to their appropriate scale. The lines in red correspond to the true parameter values we have for the experiment. The resolution number for this plot is 2.

Finally, we evaluate the Lotka-Volterra model with the underlying ode data in the high noise setting.

Projection Plot for Lotka-Volterra Model with Particle Filter

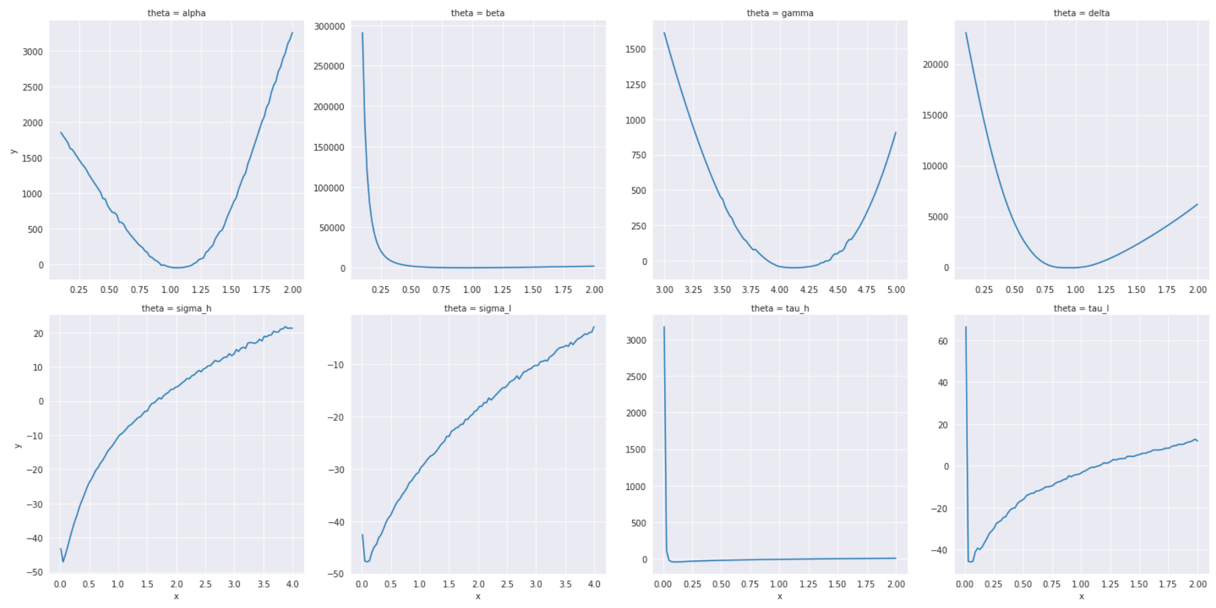

Figure 4.25: This figure consists of the projection plot for all parameters in the Lotka-Volterra model in the high-noise setting with data from the underlying ODE. This plot was generated with resolution number 2.

Next, we infer the parameters as we did in the previous settings.

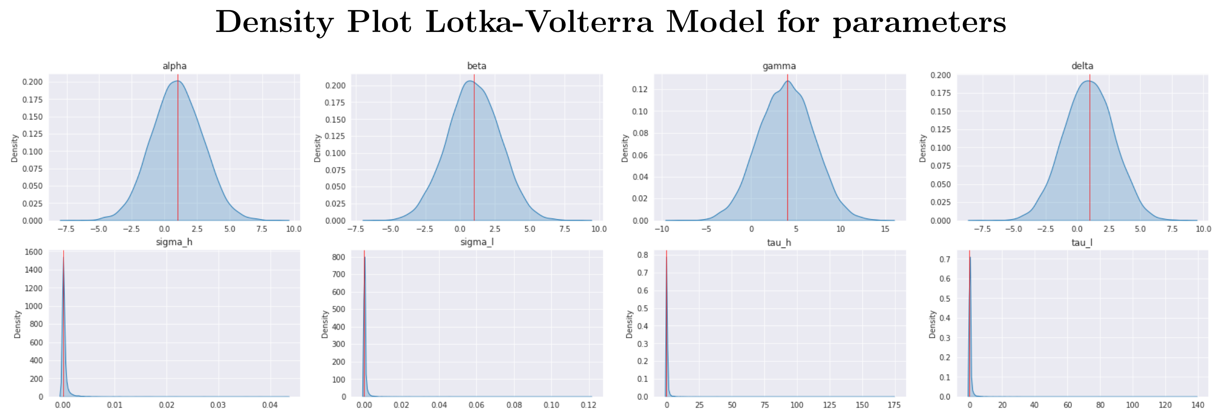

Figure 4.26: The true value of the parameters which is obtained from the simulation is the line in red in the plots and the density is obtained from our samples. We also transform the variables to their appropriate scale. The lines in red correspond to the true parameter values we have for the experiment. The resolution number used to obtain these results is 2.

4.3 Lotka Volterra with Multiplicative Noise

This model is the SDE variant of the previous model, but does not possess an underlying ODE. There has also been a large body of work [Ryder et al., 2018] that focuses on this particular variant of the model with different techniques for bayesian inference such as variational inference. This is out of the scope of this particular thesis, however, we should note that our results are obtained significantly faster than the neural-net based counterpart. The SDE is given by $\boldsymbol{X}_t = (U_t, V_t)$

$$\begin{bmatrix} \mathrm{d}U_t \\ \mathrm{d}V_t \end{bmatrix} = \begin{bmatrix} \alpha U_t - \beta U_t V_t \\ \beta U_t V_t - \gamma V_t \end{bmatrix} \mathrm{d}t + \begin{bmatrix} \alpha U_t + \beta U_t V_t & -\beta U_t V_t \\ -\beta U_t V_t & \gamma V_t + \beta U_t V_t \end{bmatrix}^{1/2} \mathrm{d}\boldsymbol{B}_t. \tag{4.12}$$

However, the SDE is actually approximated on the log scale, $\boldsymbol{Z}_t = \log \boldsymbol{X}_t$. The noisy observations are given by

$$\boldsymbol{y}_n \overset{\text{ind}}{\sim} \mathcal{N}(\exp(\boldsymbol{Z}_n), \operatorname{diag}(\boldsymbol{\tau}^2)). \tag{4.13}$$

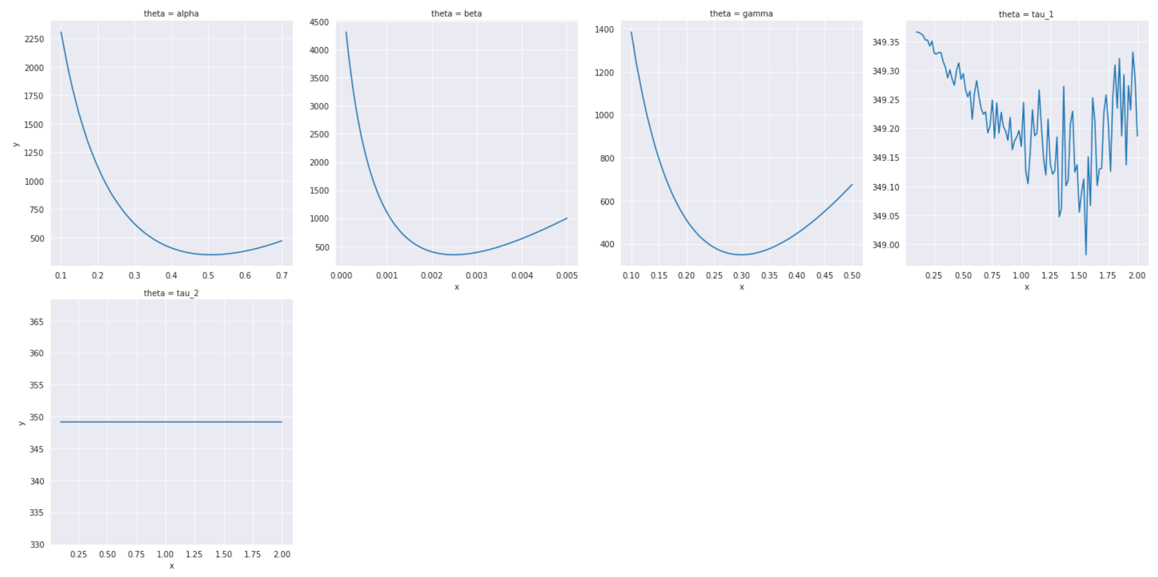

Figure 4.27: The projection plots for the different parameters in the Lotka-Volterra model with multiplicative noise. We are interested in the inference of α, β, γ. The resolution number used to obtain this plot is 10.

In this particular setting we are only concerned with α, β and γ. The projection plots indicate that the variables seem amenable to stochastic optimization (except for τ, but we are not concerned at the moment with inferring τ). Next, we run our stochastic optimization algorithm over the particle filter for 1000 iterations, with a learning rate (step size) or 0.01 and we obtain the converged estimates of $\boldsymbol{\theta}$. For this particular experiment, the values of $\boldsymbol{\theta}$ that we use are $[0.5, 0.0025, 0.3]$, we run the Adam optimizer with a learning rate of 0.01 from the initial value of $[1.0, 1.0, 1.0]$. Finally, we used 50 observations for this experiment.

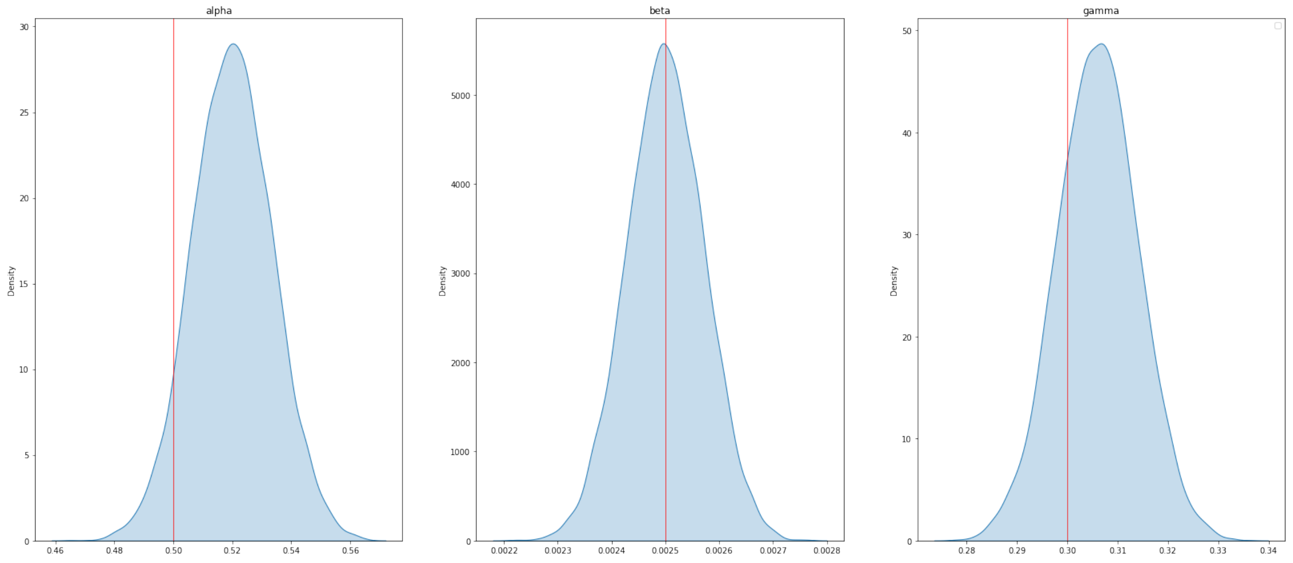

Figure 4.28: The densities are given for the Lotka-Volterra model with the parameter value used for simulation given as the red like and the parameters are the titles of the subplot. The resolution number used to obtain this plot is 10.

Chapter 5

Discussion of Experimental Results

In this chapter we go into detail about interpreting the results from Chapter 4. We first summarize some of the timing improvements and caveats to them, followed by a qualitative evaluation of the simulated inference results, then an evaluation of the results using the real data and understanding the issues present with the variance.

5.1 Inference for Simulation

The inference that we obtained from the experiments in Chapter 4 through the simulated data are very promising for our proposed approach. In particular we note that the mode that we find for the parameters are very close to if not exactly at the true value that we used for the model.The challenges in the variance is likely as a consequence of the fact that we are simulating the likelihood at every step, which is a function of the key we are using. Since the key changes iteration to iteration, our likelihood also changes which makes finding a stable hessian challenging. We also note that the local optima that we arrive at is also stable, this is to say that when we vary the learning rate, we still observe that we are converged at the local optima. This serves as a useful sanity check, because if we consider a projection plot and visualize a subdomain within the plot, we notice quickly that the likelihood is very non-convex within that subdomain. We provide one such plot below:

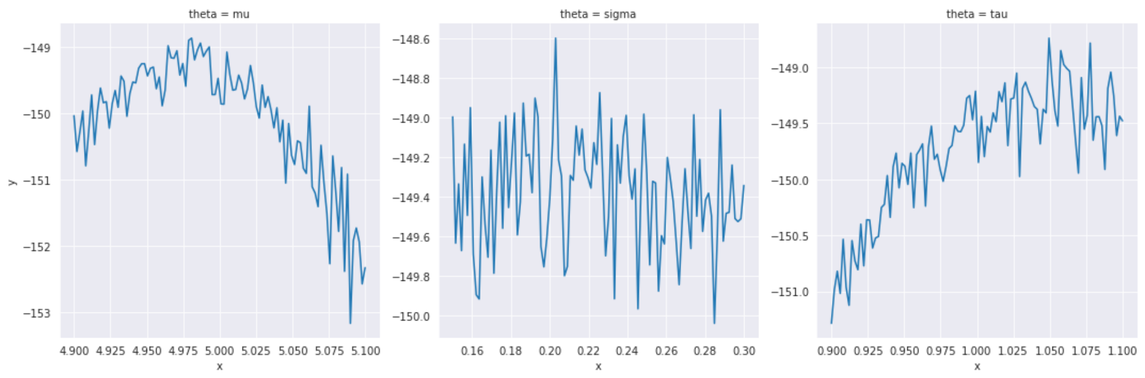

Figure 5.1: The plot reduces the x values that we search over and as a result, we obtain a magnified view of the likelihood in a very small domain.

5.2 Timing and JIT Compilation

We first note that all of the results presented in the experiments section of the paper include the timing for JIT compilation as well as the runtime of the algorithm itself. As we mentioned in Chapter 2, it is important to note that the intial time of the JIT compile step is a fixed one-time cost and is never incurred again for the entirety of the execution. As a consequence, we note that some of our results are impressive since we would only have to JIT it once and then we can run it on multiple datasets. This is to say that our timings for the experiments are slightly more than what we would expect and the amortized cost of the JIT compilation step is negligible if we run it on several datasets. Additionally, we note that it is not particularly easy to ensure that the code for the particle filter or custom resampling steps is JIT compatible. This incurs significant overhead when developing code for a particular applications. That being said, it is important to weight the amount of time it would take to create code that is JIT compatible versus the additional runtime that would be incurred by not carefully crafting the code to adhere to JIT standards.

5.3 Bias in Automatic Differentiation

One of the experimental results that was observed was the fact that there was bias present in the Auto method. This bias can be attributed to the multinomial resampling step. Multinomial resampling is compatible with automatic differentiation but when we differentiate the particle filter with respect to it, it imposes some bias on the estimated gradient. This is highlighted in theoretical work [Corenflos et al., 2021] where they prove that the gradient estimated through the multinomial resampling step is in fact biased. Due to the recent nature of this work, our implementation and verification of the bias present in the gradients occurred simultaneously to this work. As mentioned earlier, the multinomial resampling step is a non-continuous operation and as a result, gradients cannot propagate through it.

5.4 Constraint Satisfied Stochastic Gradient Descent

In the experiments we have seen, most, if not all the constraints were satisfied with a log-exp transformation scheme. This coupled with Ito's lemma allowed us to efficiently obtain an unconstrained scale for the parameters and the latent variables which in turn provided us with stochastic gradient estimates. However, we should note that for more complicated constraints, for example $\theta_i \in [0, 10] \cap \mathbb{Z}$, we would need a more sophisticated approach. The first possibility is to utilize a form a constrained optimization that would allow us to effectively optimize over the particle filter without worrying about undefined likelihoods. The second possibility is to potentially rederive a version of Ito's lemma that might suit the particular constraint. The final possibility is to take the floor or ceiling of the converged value to obtain an integer, however, this deals with the issue of biases present in our obtained values which we discuss in detail below.

Chapter 6

Conclusion

In this chapter we summarize our contributions through this thesis. Firstly, we show that our method performs well for the simulation studies that we look at. In particular, we obtain precise estimates of our parameters of inference for the different examples. We also combine several existing methods that work on diverse problems and craft a solution that is specific to SDEs.

We also showcase several methods for SDE inference and make use of the latest technological advances in the space of Just In Time compilation and parallelization to ensure that our methods have low runtime and memory utilization.

We present a high-performance library that integrates all of these methods using JAX. We document all of our code and ensure that our methods are tested, robust and scalable. This allows us to port inference to high dimensional settings, multi-cpu settings as well as utilize automatic computation of gradients present in JAX's automatic differentiation framework. Furthermore, the open-sourcing of this library will also allow other researchers to make use of the methods we have developed and ported for SDE inference.

6.1 Future Research Directions

- We present several algorithms that are utilized for SDE inference, but several advancements in the literature of Sequential Monte Carlo is yet to be implemented. In particular, techniques like SMC^2 [Chopin et al., 2013] , Auxiliary particle filters or SMC with MCMC moves [Chopin et al., 2020] are potential directions for future research which might be fruitful.

- Fully differentiable particle filters [Jonschkowski et al., 2018, Corenflos et al., 2021] are an exciting avenue for further exploration. In our work, the resampling step is not differentiable and we elaborate on the bias induced by this in the discussion section. It is possible to construct fully differentiable particle filters which might circumvent such issues and also yield theoretically favourable results like unbiasedness.

- There are several additional score and information methods that build on the original implementation of Doucet. Some of these algorithms have a runtime that is quadratic in the number of particles and is computationally more expensive than the approach we employ in this thesis. However, their estimates are lower variance and might be interesting to investigate.

References

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Andrieu et al., 2010] Andrieu, C., Doucet, A., and Holenstein, R. (2010). Particle markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342.

[Baydin et al., 2015] Baydin, A., Pearlmutter, B., Radul, A., and Siskind, J. (2015). Automatic differentiation in machine learning: A survey. arxiv preprint arxiv: 150205767.

[Chopin et al., 2013] Chopin, N., Jacob, P. E., and Papaspiliopoulos, O. (2013). Smc2: an efficient algorithm for sequential analysis of state space models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(3):397–426.

[Chopin et al., 2020] Chopin, N., Papaspiliopoulos, O., et al. (2020). *An introduction to sequential Monte Carlo*. Springer.

[Corenflos et al., 2021] Corenflos, A., Thornton, J., Deligiannidis, G., and Doucet, A. (2021). Differentiable particle filtering via entropy-regularized optimal transport. In *International Conference on Machine Learning*, pages 2100–2111. PMLR.

[Del Moral et al., 2006] Del Moral, P., Doucet, A., and Jasra, A. (2006). Sequential monte carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436.

[Douc and Cappé, 2005] Douc, R. and Cappé, O. (2005). Comparison of resampling schemes for particle filtering. In *ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005.*, pages 64–69. IEEE.

[Doucet et al., 2009] Doucet, A., Johansen, A. M., et al. (2009). A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3.

[Durham and Gallant, 2002] Durham, G. B. and Gallant, A. R. (2002). Numerical techniques for maximum likelihood estimation of continuous-time diffusion processes. *Journal of Business & Economic Statistics*, 20(3):297–338.

[Frostig et al., 2018] Frostig, R., Johnson, M. J., and Leary, C. (2018). Compiling machine learning programs via high-level tracing. *Systems for Machine Learning*, pages 23–24.

[Gelman et al., 1995] Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (1995). *Bayesian data analysis*. Chapman and Hall/CRC.

[Gordon et al., 1993] Gordon, N. J., Salmond, D. J., and Smith, A. F. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F-radar and signal processing*, volume 140, pages 107–113. IET.

[Harris et al., 2020] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

[Johansen and Doucet, 2008] Johansen, A. M. and Doucet, A. (2008). A note on auxiliary particle filters. *Statistics & Probability Letters*, 78(12):1498–1504.

[Jonschkowski et al., 2018] Jonschkowski, R., Rastogi, D., and Brock, O. (2018). Differentiable particle filters: End-to-end learning with algorithmic priors. *arXiv preprint arXiv:1805.11122*.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kudlicka et al., 2020] Kudlicka, J., Murray, L. M., Ronquist, F., and Schön, T. B. (2020). Probabilistic programming for birth-death models of evolution using an alive particle filter with delayed sampling. In *Uncertainty in Artificial Intelligence*, pages 679–689. PMLR.

[Lotka, 1920] Lotka, A. J. (1920). Analytical note on certain rhythmic relations in organic systems. *Proceedings of the National Academy of Sciences*, 6(7):410–415.

[Lotka, 1925] Lotka, A. J. (1925). *Elements of physical biology*. Williams & Wilkins.

[Lysy and Tong, 2017] Lysy, M. and Tong, J. (2017). msde: Bayesian inference for multivariate stochastic differential equations. *R package version*, 1(2).

[Maruyama, 1955] Maruyama, G. (1955). Continuous markov processes and stochastic equations. *Rendiconti del Circolo Matematico di Palermo*, 4(1):48–90.

[Murray, 2013] Murray, L. M. (2013). Bayesian state-space modelling on high-performance hardware using libbi. *arXiv preprint arXiv:1306.3277*.

[Murray and Schön, 2018] Murray, L. M. and Schön, T. B. (2018). Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control*, 46:29–43.

[Oksendal, 2013] Oksendal, B. (2013). *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Pedersen, 1995] Pedersen, A. R. (1995). A New Approach to Maximum Likelihood Estimation for Stochastic Differential Equations Based on Discrete Observations. *Scandinavian Journal of Statistics*, 22(1):55–71.

[Picchini, 2014] Picchini, U. (2014). Inference for sde models via approximate bayesian computation. *Journal of Computational and Graphical Statistics*, 23(4):1080–1100.

[Pitt and Shephard, 1999] Pitt, M. K. and Shephard, N. (1999). Filtering via Simulation: Auxiliary Particle Filters. *Journal of the American Statistical Association*, 94(446):590–599. tex.ids= pitt.shephard22 publisher: [American Statistical Association, Taylor & Francis, Ltd.].

[Poyiadjis et al., 2011] Poyiadjis, G., Doucet, A., and Singh, S. S. (2011). Particle approximations of the score and observed information matrix in state space models with application to parameter estimation. *Biometrika*, 98(1):65–80.

[Ronquist et al., 2021] Ronquist, F., Kudlicka, J., Senderov, V., Borgström, J., Lartillot, N., Lundén, D., Murray, L., Schön, T. B., and Broman, D. (2021). Universal probabilistic programming offers a powerful approach to statistical phylogenetics. *Communications biology*, 4(1):1–10.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

[Ryder et al., 2018] Ryder, T., Golightly, A., McGough, A. S., and Prangle, D. (2018). Black-box variational inference for stochastic differential equations. In *International Conference on Machine Learning*, pages 4423–4432. PMLR.

[Sabne, 2020] Sabne, A. (2020). Xla : Compiling machine learning for peak performance.

[Särkkä and Solin, 2019] Särkkä, S. and Solin, A. (2019). *Applied stochastic differential equations*, volume 10. Cambridge University Press.

[Stramer and Yan, 2007] Stramer, O. and Yan, J. (2007). On simulated likelihood of discretely observed diffusion processes and comparison to closed-form approximation. *Journal of Computational and Graphical Statistics*, 16(3):672–691.

[Subramani et al., 2021] Subramani, P., Vadivelu, N., and Kamath, G. (2021). Enabling fast differentially private sgd via just-in-time compilation and vectorization. *Advances in Neural Information Processing Systems*, 34.

[Svedin et al., 2021] Svedin, M., Chien, S. W., Chikafa, G., Jansson, N., and Podobas, A. (2021). Benchmarking the nvidia gpu lineage: From early k80 to modern a100 with asynchronous memory transfers. In *Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, pages 1–6.

[Wangersky, 1978] Wangersky, P. J. (1978). Lotka-volterra population models. *Annual Review of Ecology and Systematics*, 9(1):189–218.

[Wigren et al., 2019] Wigren, A., Risuleo, R. S., Murray, L., and Lindsten, F. (2019). Parameter elimination in particle gibbs sampling. *Advances in Neural Information Processing Systems*, 32.

Appendix A

A.1 Proof of 2.12

In order to prove the above statement, we first have to observe that $p(W, X, Y)$ is a multivariate normal and thus we must obtain the mean and variance. Then, let $\boldsymbol{Z}_W, \boldsymbol{Z}_X, \boldsymbol{Z}_Y$ be independent vectors of independent and identically distributed normal random variables of size corresponding to the dimensions of $\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}$ then we have

$$\begin{aligned}
\boldsymbol{W} &= \boldsymbol{\mu}_W + \boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W \\
\boldsymbol{X} &= \boldsymbol{W} + \boldsymbol{\mu}_{X|W} + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X \\
&= \boldsymbol{\mu}_W + \boldsymbol{\mu}_{X|W} + \boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X \\
\boldsymbol{Y} &= \boldsymbol{A}\boldsymbol{X} + \boldsymbol{\Sigma}_Y^{1/2}\boldsymbol{Z}_Y \\
&= \boldsymbol{A}[\boldsymbol{\mu}_W + \boldsymbol{\mu}_{X|W} + \boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X] + \boldsymbol{\Omega}^{1/2}\boldsymbol{Z}_Y \\
&= \boldsymbol{A}[\boldsymbol{\mu}_W + \boldsymbol{\mu}_{X|W}] + \boldsymbol{A}[\boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X] + \boldsymbol{\Omega}^{1/2}\boldsymbol{Z}_Y.
\end{aligned} \tag{A.1}$$

The following steps will be computing the covariance between each of the variables using the equations derived above

$$\begin{aligned}
\mathrm{cov}(\boldsymbol{W},\boldsymbol{X}) &= \mathrm{cov}(\boldsymbol{\mu}_W + \boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W, \boldsymbol{\mu}_W + \boldsymbol{\mu}_{X|W} + \boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X) \\
&= \mathrm{cov}(\boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W, \boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X) \\
&= \boldsymbol{\Sigma}_W \\
\mathrm{cov}(\boldsymbol{W},\boldsymbol{Y}) &= \mathrm{cov}\left(\boldsymbol{\mu}_W + \boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W, \boldsymbol{A}[\boldsymbol{\mu}_W + \boldsymbol{\mu}_{X|W}] + \boldsymbol{A}[\boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X] + \boldsymbol{\Omega}^{1/2}\boldsymbol{Z}_Y\right) \\
&= \mathrm{cov}\left(\boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W, \boldsymbol{A}[\boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X] + \boldsymbol{\Omega}^{1/2}\boldsymbol{Z}_Y\right) \\
&= \boldsymbol{\Sigma}_W\boldsymbol{A}' \\
\mathrm{cov}(\boldsymbol{X},\boldsymbol{Y}) &= \mathrm{cov}\left(\boldsymbol{\mu}_W + \boldsymbol{\mu}_{X|W} + \boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X, \boldsymbol{A}[\boldsymbol{\mu}_W + \boldsymbol{\mu}_{X|W}]\right. \\
&\quad \left. + \boldsymbol{A}[\boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X] + \boldsymbol{\Omega}^{1/2}\boldsymbol{Z}_Y\right) \\
&= \mathrm{cov}\left(\boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X, \boldsymbol{A}[\boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X] + \boldsymbol{\Omega}^{1/2}\boldsymbol{Z}_Y\right) \\
&= (\boldsymbol{\Sigma}_W + \boldsymbol{\Sigma}_{X|W})\boldsymbol{A}'.
\end{aligned} \tag{A.2}$$

Similarly, we have the variance of each variable

$$\begin{aligned}
\mathrm{var}(\boldsymbol{W}) &= \mathrm{var}(\boldsymbol{\mu}_W + \boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W) \\
&= \boldsymbol{\Sigma}_W \\
\mathrm{var}(\boldsymbol{X}) &= \mathrm{var}(\boldsymbol{\mu}_W + \boldsymbol{\mu}_{X|W} + \boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X) \\
&= \boldsymbol{\Sigma}_W + \boldsymbol{\Sigma}_{X|W} \\
\mathrm{var}(\boldsymbol{Y}) &= \mathrm{var}(\boldsymbol{A}[\boldsymbol{\mu}_W + \boldsymbol{\mu}_{X|W}] + \boldsymbol{A}[\boldsymbol{\Sigma}_W^{1/2}\boldsymbol{Z}_W + \boldsymbol{\Sigma}_{X|W}^{1/2}\boldsymbol{Z}_X] + \boldsymbol{\Omega}^{1/2}\boldsymbol{Z}_Y) \\
&= \boldsymbol{A}(\boldsymbol{\Sigma}_W + \boldsymbol{\Sigma}_{X|W})\boldsymbol{A}' + \boldsymbol{\Omega}.
\end{aligned} \tag{A.3}$$

Now, we move on to a proof for (3) Given the joint distribution from (2), we can directly apply the formula for the conditional distribution of a multivariate normal to obtain $p(W|Y) \sim \mathcal{N}(\boldsymbol{\mu}_{W|Y}, \boldsymbol{\Sigma}_{W|Y})$. In this context we have

$$\begin{aligned}
\boldsymbol{\mu}_{W|Y} &= \boldsymbol{\mu}_W + \boldsymbol{\Sigma}_W\boldsymbol{A}'\boldsymbol{\Sigma}_Y^{-1}(\boldsymbol{Y} - \boldsymbol{\mu}_Y) \\
\boldsymbol{\Sigma}_{W|Y} &= \boldsymbol{\Sigma}_W - \boldsymbol{\Sigma}_W\boldsymbol{A}'\boldsymbol{\Sigma}_Y^{-1}\boldsymbol{A}\boldsymbol{\Sigma}_W
\end{aligned} \tag{A.4}$$

which completes the final formulation.

A.2 Brownian Motion Posterior Derivation

The exact likelihood is defined as $p(\boldsymbol{y}_{1:T} \mid y_0, \boldsymbol{\theta})$ rather than $p(\boldsymbol{y}_{0:T} \mid \boldsymbol{\theta})$. The reason is that the latter expression requires one to integrate over $x_0 \sim \pi(x_0)$, which can only be done when $\pi(x_0)$ is a proper prior. However, for our choice of $\pi(x_0) \propto 1$ this is not the case. On the other hand, $p(\boldsymbol{y}_{1:T} \mid y_0, \boldsymbol{\theta})$ only requires us to integrate over $p(x_0 \mid y_0, \boldsymbol{\theta})$, which only requires the posterior to be proper (which is always the case for valid Bayesian inference).

Conditioned on x_0 and $\boldsymbol{\theta}$, the Brownian latent variables $\boldsymbol{x}_{1:T}$ are multivariate normal with

$$\begin{aligned} E[x_t \mid x_0, \boldsymbol{\theta}] &= x_0 + \mu t, \\ \mathrm{Cov}(x_s, x_t \mid x_0, \boldsymbol{\theta}) &= \sigma^2 \min(s,t). \end{aligned} \tag{A.5}$$

Conditioned on $\boldsymbol{x}_{0:T}$ and $\boldsymbol{\theta}$, the measurement variables $\boldsymbol{y}_{1:T}$ are multivariate normal with

$$\begin{aligned} E[y_t \mid \boldsymbol{x}_{0:T}, \boldsymbol{\theta}] &= \boldsymbol{x}_{1:T}, \\ \mathrm{Cov}(y_s, y_t \mid \boldsymbol{x}_{0:T}, \boldsymbol{\theta}) &= \tau^2 \delta_{st}. \end{aligned} \tag{A.6}$$

Therefore, the marginal distribution of $\boldsymbol{y}_{1:T}$ is multivariate normal with

$$\begin{aligned} E[y_t \mid x_0, \boldsymbol{\theta}] &= x_0 + \mu t, \\ \mathrm{Cov}(y_s, y_t \mid x_0, \boldsymbol{\theta}) &= \sigma^2 \min(s,t) + \tau^2 \delta_{st}. \end{aligned} \tag{A.7}$$

For the given choice of prior, we have $x_0 \mid y_0 \sim \mathcal{N}(y_0, \tau^2)$ for the initial observation y_0. Integrating over x_0, the marginal distribution of $\boldsymbol{y}_{1:T}$ is MVN with

$$\begin{aligned} E[y_t \mid y_0, \boldsymbol{\theta}] &= y_0 + \mu t, \\ \mathrm{Cov}(y_s, y_t \mid y_0, \boldsymbol{\theta}) &= \sigma^2 \min(s,t) + \tau^2 (\delta_{st} + 1). \end{aligned} \tag{A.8}$$

This concludes our derivation.