Inferring the learning rule with particle Metropolis Hastings

Project thesis

Department of Mathematical Sciences
Norwegian University of Science and Technology (NTNU)

Maud Rødsmoen, Emil Alvar Myhre

December 21, 2020

Abstract

The brain is the command center for the nervous system for humans, as well as other species. *Neurons* are the fundamental cells of the brain, giving rise to the complex and powerful functioning brain that we possess by communicating through electrical and chemical signals. Learning and memory are often understood to be induced by new neural connections emerging in the brain. This evolution of the dynamics in the brain, with neural connections emerging or possibly disappearing, is often referred to as *synaptic plasticity*. Insight and understanding of how these mechanisms are driven, could be crucial within medical research, for recognising and understanding neurological disorders. In this work, we approach synaptic plasticity from a mathematical perspective, aiming to exploit mathematical tools to understand these dynamics.

This work extends on the framework presented by Linderman and coauthors [1] for studying synaptic plasticity, where the dynamics are believed to follow some underlying patterns, called *learning rules*. Specifically, utilising spike-timing-dependent plasticity (STDP) learning rules, as suggested in [2]. This work implements two Metropolis-Hastings procedures for inferring learning rule parameters in the Bayesian regime. Additionally, a relatively comprehensive analysis of the performance and robustness of the developed method is conducted based on simulated data. Furthermore, the method is applied to real neural data from mouse brains, collected by McKenzie and coauthors [3].

The results based on the applications on both simulated and real data are encouraging and promising. The nature of the learning rule appears to be captured rather confidently by performing statistical inference with the developed Metropolis-Hastings methods.

Preface

This work is our specialisation project - TMA4500 - in Applied Physics and Mathematics at the Norwegian University of Science and Technology (NTNU). The goal of this work is mainly to prepare for the upcoming Master's thesis, by studying a diversity of relevant literature as well as implementing the most relevant ideas, and thereby gaining a broad understanding of the desired theory.

This process was made both easier and more enjoyable by our supervisors on this project. We would therefore like to thank our supervisor Benjamin Dunn for his help and for letting us join his wonderful research group at the Department of Mathematical Sciences. Also, we would like to thank our co-supervisor Claudia Battistin for her passion for this project and for her willingness to help us whenever needed.

Maud Rødsmoen, Emil Alvar Myhre
Trondheim, Norway
December 2020

Contents

Abbreviations

GLM Generalised linear model

HMM Hidden Markov Model

MAP Maximum a Posteriori

MCMC Markov Chain Monte Carlo

MH Metropolis Hastings

MLE Maximum likelihood estimation

SMC Sequential Monte Carlo

STDP Spike-Timing-Dependent Plasticity

Chapter 1

Introduction

The study of dynamical and evolving networks is of great interest within various fields of study. In general, one could imagine a network, where internal components (or nodes) might interact and influence each other. At the same time, the network as a whole could be characterised by some state, which somehow describes the current network dynamics. These dynamics, however, might change over time due to either internal or external factors. Clearly, these are really flexible and complex systems that are applicable for many purposes, and that could describe many different dynamics that we face in real life.

Let's showcase this with a rather big and complex social example. Most countries in the world have a democracy of some sort [4]. Let us now consider this political system as a network, and briefly break down the dynamics. Well, we have a lot of different components in such a system: 1) politicians, 2) voters, 3) the press, 4) institutes/businesses, to mention a few. These are all allowed to interact, which leads to possible influence from each other, or voters could even be influenced by some external event in another country. If the dynamics within the population changes enough, the political system could also change. That is, some other party could come to power, and the "state" of the network could change.

Similar networks might be constructed for other time-dependent systems, for instance the stock market, the spread of a pandemic, social networks or infrastructural networks. However, in this work we will focus on a super interesting biological application of such networks: **the brain!**

The brain consists of interconnected neural cells that communicate with each other, causing our body to function as it does. How these neural cells are connected can be considered a network. Furthermore, we know that this

network develops over time, in that new connections between cells emerge, hence why we can learn new things or make new memories. Also, how the brain develops over time is believed to follow some underlying patterns. However, these patterns are most certainly different for people who suffer from brain diseases, like Alzheimer's disease, as their ability to learn and remember is severely altered. The aim of this work is to use statistical tools to gain an understanding of these underlying patterns. Such an understanding could be essential for detecting and possibly treating diseases like Alzheimer's.

We will work with a framework for analysing such neural dynamics, proposed by Linderman and coauthors [1]. We will develop this method and propose some modifications to it. Moreover, we will test and analyse the method to ensure it works before we apply it to real neural data. The data we are using is collected by McKenzie and coauthors [3]. It covers information about more than 400 000 neuron pairs from mice, and we are excited to see if our method will yield anything reasonable when applied to such interesting data.

In chapter 2 we will provide some basic insights in Neuroscience and introduce relevant terminology in order to make the network at hand more clear and understandable. In chapter 3 some of the relevant mathematical tools for this work will be presented. In chapter 4 we will present the specific model under investigation, followed by the concrete method used for inferring these underlying patterns in the brain. In chapter 5 we will present the results of our work, and in chapter 6 we will briefly comment and conclude on our findings, as well as suggest some further improvements or changes.

Chapter 2

Concepts in Neuroscience

In this section we will present some basic concepts within the field of neuroscience, giving insight into how the brain works on microscopic and macroscopic levels, which will be helpful for the reader in order to understand the work of this paper. Section 2.1 will consist of a biological presentation of the brain and the neural dynamics. In section 2.2 we will briefly discuss some ideas of how these dynamics can be reconstructed mathematically, which we will also apply later. At the end of the chapter, we will present these already mentioned underlying patterns in a more detailed manner, which is also the main focus of our upcoming work.

2.1 Neurons

A neural cell, from now on only referred to as a *neuron*, is the fundamental component of the brain. The human brain is believed to consist of ~ 100 billion neurons [5]. A single neuron consists of three main parts [6]:

- a cell body, also called *soma*
- *dendrites*
- an *axon*

As we already established, the neurons are interconnected and can communicate with each other. This happens through the neurons sending electrical signals to other neurons, and these signals are referred to as *action potentials*, *spikes* or the neuron *firing*. Exactly when an action potential occurs in a neuron, depends on the *membrane potential* of the neuron. The membrane potential is defined as the difference in electrical potential between

the inside and the outside of the cell, and is measured in voltage. If this voltage increases enough and hits a certain threshold, an action potential is triggered and the neuron spikes. Then this signal travels through the axon of the neuron, and connected neurons receive this signal through their dendrites. This connection between an axon of one neuron and a dendrite of another is called a *synapse*, and is illustrated in figure 2.1.

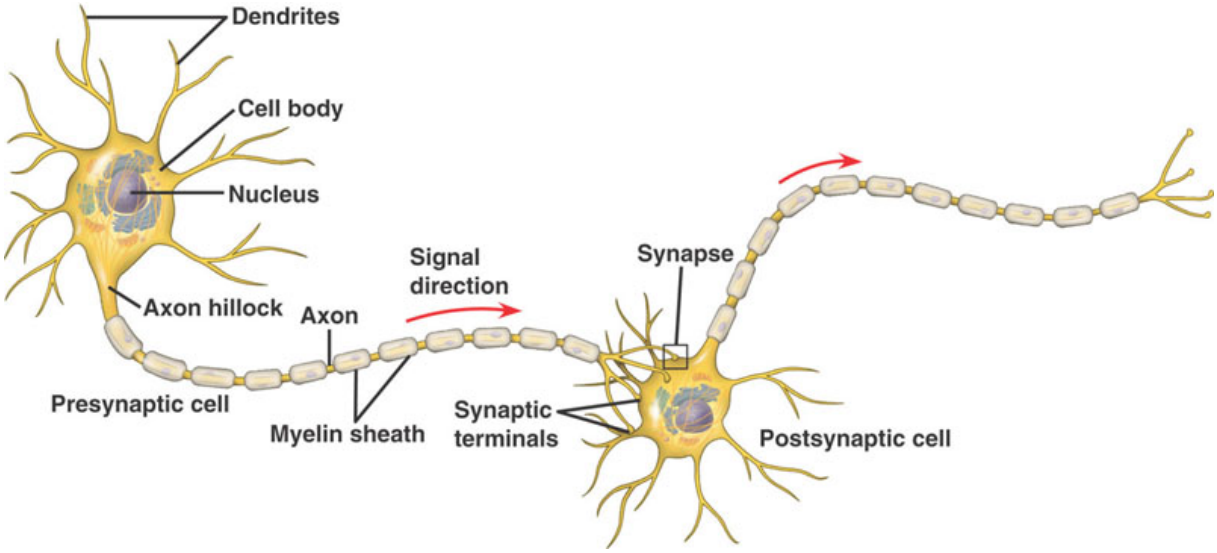

Figure 2.1: A picture showcasing two neurons and how they are connected through a synapse [7].

The neuron sending the signal is referred to as a *presynaptic neuron*, while the neuron receiving the signal is called a *postsynaptic neuron*. These synapses are one of the reasons for the constant evolution of the membrane potential in a neuron. An electrical signal following a spike from a presynaptic neuron affects the current voltage in the postsynaptic neuron. The membrane potential could either increase, in which case the synapse is called an *excitatory synapse*. However, the membrane potential could also decrease as a result of the spike, in which case the synapse is called an *inhibitory synapse*. Hence, if a neuron is exposed to a lot of excitatory inputs, the membrane potential will rise, which might result in an action potential if the voltage threshold is reached. Simultaneously, the neuron might be stimulated from other random sources, for instance other parts of the brain, which also could affect the membrane potential of the neuron. This is often referred to as *noise*. Figure 2.2 shows how the membrane potential of a neuron could change because of some stimulus, either from connected neurons or from noise, and when the threshold is reached, a spike is triggered.

Now, in order to analyse these dynamics, we would like to have some feasible mathematical frameworks, able to reconstruct and describe these dy-

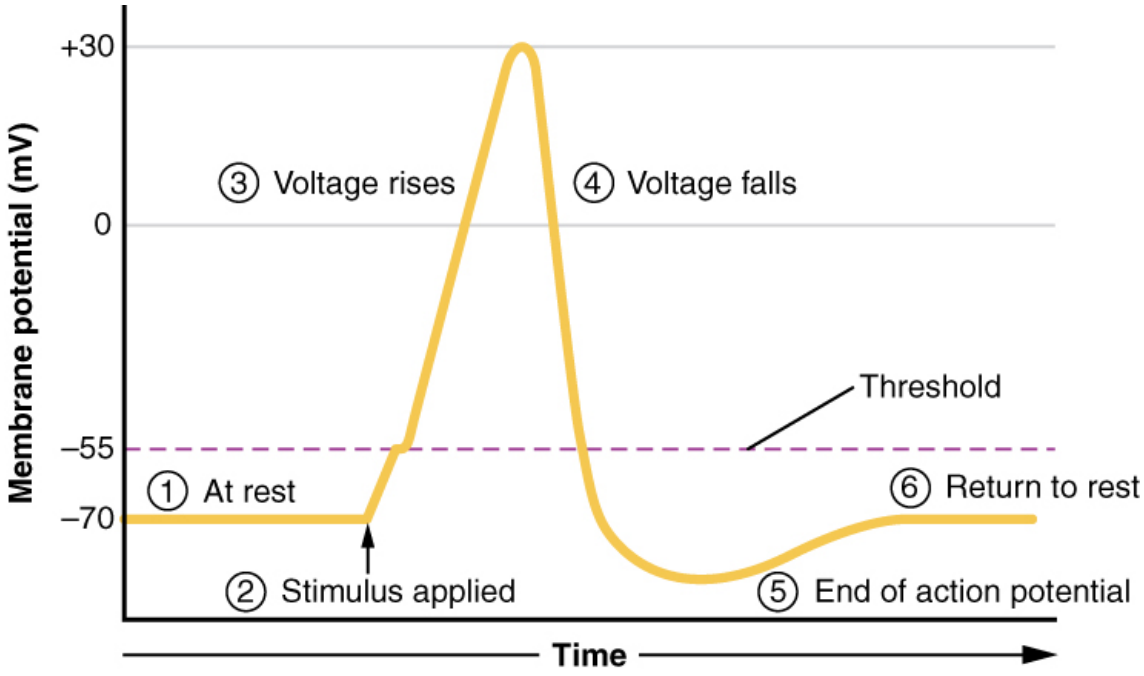

Figure 2.2: Dynamics of the membrane potential with a spike occuring [8].

namics as well as possible.

2.2 Mathematical frameworks

Our goal is first and foremost to be able to estimate the spiking events of the neurons, such that they depend on the input from other neurons and that the modelled neurons behave similarly to actual neurons in the brain. To provide more context and hopefully make the interpretation easier, we briefly present two different approaches for this purpose, even though only the latter one will be applied in our work.

2.2.1 A deterministic approach

The first option is to model the spiking events deterministically, which might be easier to relate to figure 2.2. In this case one usually models the evolution of the membrane potential directly, and as soon as this value exceeds some predetermined threshold, a spike is recorded, and the membrane potential is set to some reset value. These dynamics are often modelled through equations on the form

$$\tau_m \frac{dV}{dt} = -V(t) + RI(t), \quad \text{for } V(t) \leq V_t \tag{2.1}$$

for some constants τ_m, R. That is, the membrane potential, $V(t)$, evolves due to some (or several) input stimulus, $I(t)$, and when the voltage reaches a threshold, V_t, the voltage is reset back to some value, $V(t) = V_R$, before it starts to evolve again. This is called an *integrate-and-fire (IF) model* and usually yields a graph similar to the one shown in figure 2.3 [9].

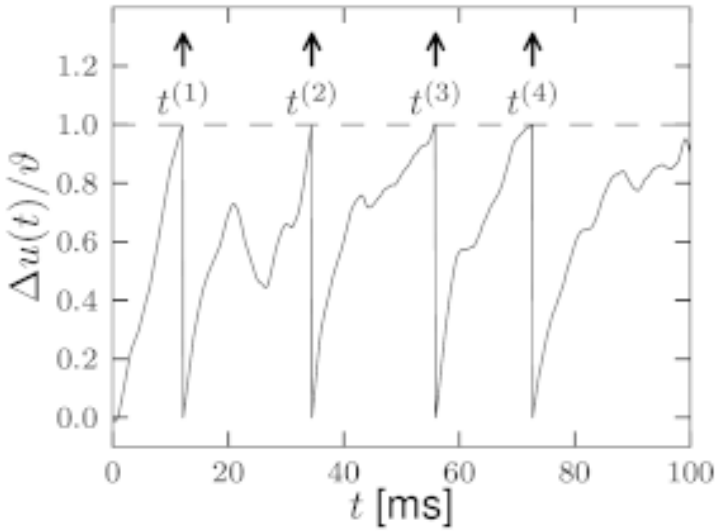

Figure 2.3: The evolution of the membrane potential with time-dependant input. Spike times $t^{(i)}$ recorded when $V(t)$ hits the threshold [9].

These paths before the spiking events are reconstructing the dynamics in figure 2.2, from the point of "Stimulus applied" until the purple threshold value is reached. These IF-models could also come in different forms, expanding on the general idea of equation (2.1).

2.2.2 A probabilistic approach

Even though the IF-models are simple, they also have limitations. We know that the brain is a really complex system with a lot of variation and noise, and an actual neuron probably does not have one defined threshold for spiking. Biologically this supports the idea of using a more irregular regime for modeling spikes, for example a stochastic framework.

In such models the event of spiking happens with a certain probability, depending on the current input stimulus of the neuron. Naturally, this probability of spiking would increase somewhat analogously to an increased stimulus. In general, an action potential is triggered with some probability,

$$0 < \lambda(I) < 1$$

where λ is dependent on the input stimulus, I. Another advantage of this approach is that it allows for other mathematical tools for interpreting and analysing neural activity, which we will exploit in this work. More on how our probabilistic model is defined and how we infer neural data will be presented in chapters 3 and 4.

2.3 Spike trains

The most interesting aspect for analysis purposes, is the actual spike times of the neurons. A functioning neuron will repeatedly fire, leading to a sequence of spikes, which is also illustrated in the mathematical representation in figure 2.3. If we consider a certain time interval, all the spike times recorded from a neuron form what we refer to as a *spike train*. Assuming a neuron spikes n times within the time interval $[0, T]$, the spike train for this neuron can be expressed compactly as

$$\{t^{(i)}\}_{i=1}^{n} = \{t^{(1)}, t^{(2)}, ..., t^{(n)}\}, \quad t^{(i)} < t^{(i+1)} \; \forall i, \quad t^{(i)} \leq T \; \forall i$$

where $t^{(i)}$ is the spiking time of spike number i of the neuron. A visualisation of such a spike train is provided in figure 2.4.

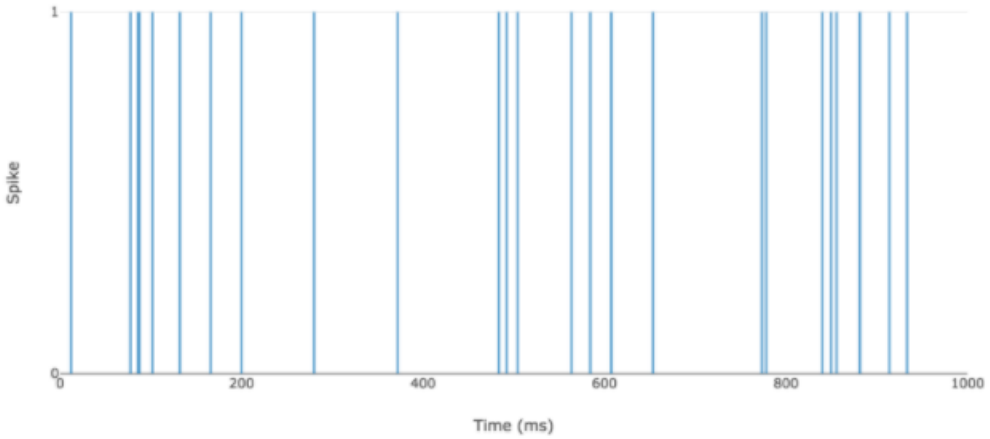

Figure 2.4: A spike train where the blue lines illustrates the spike times [10].

These spiking times will be essential for the work of this paper. Having access to spike trains from several connected neurons could be helpful to understand how they interact with each other.

2.4 Synaptic plasticity

Now that we know the basic dynamics of a single neuron, let us advance to the more interesting part: **a network of neurons**.

Initially in this paper, the brain was presented as a complex and dynamic network, consisting of interconnected neurons that communicate with each other. A deeper understanding of these dynamics is what we strive for. As we know, neurons might be connected through either excitatory or inhibitory synapses. Furthermore, the strength of these synapses might be of different magnitude, and they might change over time. These changes are referred to as *synaptic plasticity*. The stronger an excitatory synapse is, the more input a postsynaptic neuron receives from a presynaptic neuron spiking, and the greater the probability becomes of the postsynaptic neuron firing. Biologically this input can be interpreted as the increase of the membrane potential, and oppositely for inhibitory synapses.

We established that these changes probably follow some underlying patterns, these patterns will be referred to as *learning rules*. These rules are dictating how the brain develops. If we would be able to identify or characterise these learning rules, we could have a much deeper understanding of the synaptic plasticity. In order to gain such knowledge, spike train information from several neurons could be beneficial. Imagine that the spike trains of two neurons are almost identical, that is, the neurons spike at almost the same times. Then it would be reasonable to assume some correlation, and the connection between them should be strong. In this work we will consider *spike-timing-dependent plasticity* (STDP) learning rules [11]. These rules use exactly such information about the spike times of connected neurons to determine the evolution of the network dynamics. Furthermore, STDP in particular has been shown to serve for storing sequences [12] in artificial neural networks, so it is becoming a popular topic in the machine learning community as well [13].

2.5 Alzheimer's disease

A better understanding of synaptic plasticity could further be essential for the understanding of diseases like Alzheimer's. Synaptic plasticity is thought to be an essential mechanism for learning and for memory encoding [14] [15], meaning that the variations in connectivity strength between neurons might

be caused by Alzheimer's disease. The memory-related disorder progresses by causing brain cells to degenerate and die [16], which causes a decline in the ability to think and remember. Alzheimer's disease poses major challenges to both the diseased and society as a whole. As the disease is closely related to age, we can expect it to become increasingly common, as the life expectancy increases and is not even approaching a plateau yet [17].

There is currently no treatment that can stop the development of Alzheimer's completely. However, there are methods for slowing the worsening of the symptoms, thereby improving the quality of life for both patients and their relatives. It would therefore be a great advantage if one could identify the disease as early as possible. The challenges Alzheimer's disease poses have inspired a ton of research, aiming to both prevent and discover the disease early on.

An Alzheimer's patient will experience significant brain shrinkage, caused by the loss of synapses and neurons. The exact cause of this loss is still being researched extensively, but are suggested to be caused by an accumulation of amyloid plaques [18], which in turn impairs the neural activity, interactions and then plasticity. By studying the synaptic plasticity properties of brains, it might be possible to distinguish between a healthy brain and the brain of an Alzheimer's patient.

In research, these studies are often conducted on rats or mice, as their brains have sufficient similarities to human brains. More specifically, the most heavily damaged cortex (the entorhinal cortex) [19] in a brain with Alzheimer's is phylogenetically conserved across species [20], so studying this cortex in rat brains can be beneficial to humans as well.

Chapter 3

Theory

In this chapter we will provide some relevant mathematical ideas, all of them being applied in the inference in the upcoming chapters. Along the way we will also conceptually connect these mathematical ideas to the situation of interest, namely the synaptic plasticity in the brain.

In section 3.1, the idea of Markov chains will be presented, this part is based on the book *Introduction to probability models* [21]. More specifically, we will introduce a Hidden Markov Model (HMM) which will describe our network of neurons. In section 3.2 we present the concept of Generalised linear models (GLM) and how this looks like for the Bernoulli processes. This is the stochastic framework we will use for modeling the spiking events probabilistically, as mentioned in section 2.2. In section 3.3 we will present different methods for parameter estimation, which is essential for our end goal, namely to understand and identify the nature of the underlying learning rules. Lastly, we will in section 3.4 introduce a method for estimating the cross-correlation between different spike trains, which will be useful when working on the real data.

3.1 Markov Chains

Let X_n be a stochastic process in discrete time. Assume that the process X_n can take either finite, countable or continuous values. For now, let's assume a countable state space for simplicity. We therefore denote these possible values with indices $\{0,1,2,...\}$, and call them *states*. If $X_n = i$, the process is in state i at time n. Then, given that the process is in state i at time n, there is a certain probability that the process is in state j at time $n+1$. This probability

of transitioning from state i to j is called the *transition probability*, denoted by $P_{ij} \in [0,1]$. Transition probabilities exists for all set of state pairs i and j. The probability of remaining in the current state would be P_{ii}.

What characterises a Markov chain is the *Markov property*, which says that the next transition of the process only depends on the current state, regardless of the whole history of the process. That is

$$P\Big(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, ..., X_1 = i_1, X_0 = i_0\Big) = P\Big(X_{n+1} = j | X_n = i\Big) = P_{ij}.$$

Since all transition probabilities are defined to be non-negative and the process needs to make a transition, the sum of all transition probabilities from a given state i must be 1,

$$\sum_{j=0}^{\infty} P_{ij} = 1, \quad i = 0, 1, ...$$

All of these probabilities can be structured in a matrix $\boldsymbol{P}$, where the entry $\boldsymbol{P}[i, j]$ indicates the transition probability P_{ij}. An example of a 3-state Markov chain with transition probabilities is showcased in figure 3.1.

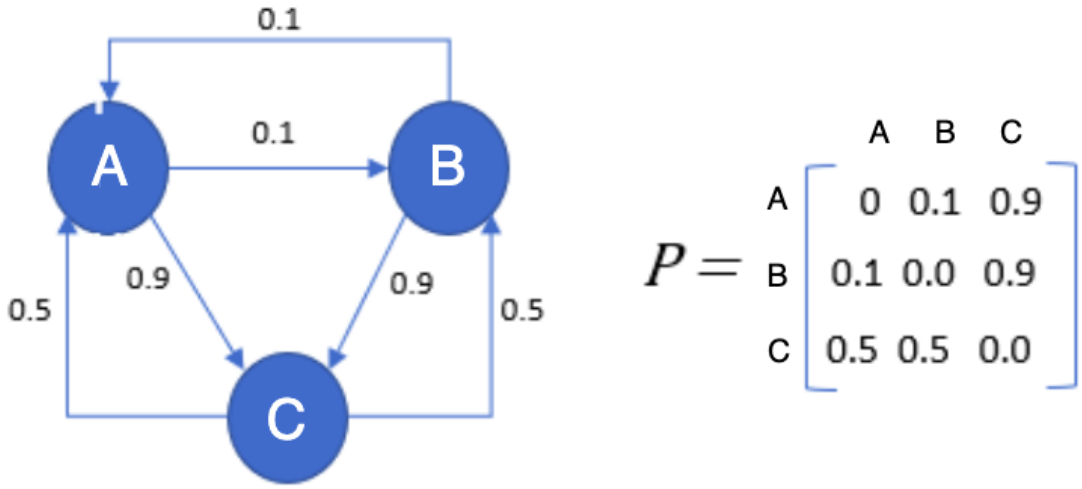

Figure 3.1: An example of a Markov chain with 3 possible states (left) and the corresponding transition probability matrix (right) [22].

3.1.1 Hidden Markov Model

Now we will introduce a more sophisticated model, which will be useful for the network of neurons. Now let $X_n = \{X_1, X_2, ...\}$ be a Markov process as ex-

plained in section 3.1. In addition, lets now include another stochastic process, which is statistically dependent on the state of the Markov process X_n. Lets further assume, that we do not have any information about the states of the Markov process, hence the Markov Chain is called *hidden*. What we do have, however, is a set of observations $Y = \{Y_1, Y_2, ...\}$, from the second stochastic process. We also know, that they are dependent on X_n, so at time t, we consider the conditional probability for the observation, which follows the density

$$Y_t|X_t \sim P(Y_t|X_t). \tag{3.1}$$

We also know that X_n follows the Markov property, and is hence dependent on the previous state, that is

$$X_{t+1}|X_t \sim P(X_{t+1}|X_t). \tag{3.2}$$

Such a model, where the sequence of underlying Markov chain states are unknown, while we have a set of correlated observations, is called a *Hidden Markov Model* (HMM). How would this correspond to the brain?

Brain interpretation

Well, in the model we are studying, imagine the following interpretation, using the notation of this section

- X_t would be the state of the neural network at time t. That is, how the neurons are connected: in terms of which ones are connected and how strong the synapses are.
- Y would be the observed spiking times of the neurons.

The state of the network is unknown but we assume that it changes over time according to the Markov property, corresponding to (3.2). Furthermore, when the neurons spike also depends on the connections between them, corresponding to (3.1).

Building on this concept of a general HMM, in the brain we assume that the density $P(X_{t+1}|X_t)$ is determined by our specific learning rule, which will be mathematically defined in the next section. As previously mentioned, this learning rule depends on the whole spike history of the neurons, being Y. In addition, it features further parameters, θ, that have to be inferred and

determined. The state transition depends on the previous state, the spike times and the parameters, so the density can be expressed as

$$X_{t+1}|X_t \sim P(X_{t+1}|X_t, Y, \theta). \tag{3.3}$$

We will explain the model in more detail in the next section.

3.2 Generalised linear models

Now we will explore a generalisation of linear regression models, namely Generalised linear models, which we will refer to as GLMs. The next two sections are based on the book *Regression* [23]. Recall the standard linear regression for a response y and a vector of covariates $\boldsymbol{x}$:

$$y = \boldsymbol{x}^T \boldsymbol{\beta} + \epsilon$$

with $\boldsymbol{\beta}$ being the vector of regression coefficients and ϵ is assumed to be a normally distributed noise parameter with mean 0 and variance σ^2. Since the expected value of the y is the linear predictor $\boldsymbol{x}^T\boldsymbol{\beta}$, the response variable can be expressed through the normal distribution,

$$y \sim N(\boldsymbol{x}^T\boldsymbol{\beta}, \sigma^2). \tag{3.4}$$

However, if the response variable has some restrictions, the normality assumption for the response might be violated. The GLM framework is therefore useful, as it allows response variables to originate from different distributions than the normal one.

3.2.1 General GLM

The response variable y in the GLM framework can be drawn from a univariate exponential family with density function on the form

$$f(y|\theta) = \exp\left(\frac{y\theta - b(\theta)}{\phi} \cdot w + c(y, \phi, w)\right) \tag{3.5}$$

with $b(\theta)$ and $c(y, \phi, w)$ being known functions. θ is called the *canonical parameter*, ϕ is a so-called nuisance parameter and w is a weight function.

Several distributions can be rewritten such that their density functions are of the form of (3.5), for example: the normal distribution, the Poisson distribution, the Bernoulli distribution and the gamma distribution.

From (3.5) one can find the expected value and variance of the response variable y in the following way

$$\begin{aligned} \mathrm{E}[y] &= \mu = b'(\theta) \\ \mathrm{Var}[y] &= \sigma^2 = b''(\theta)\frac{\phi}{w}. \end{aligned} \tag{3.6}$$

For a normally distributed response, we see from (3.4), that the mean is equal to the linear predictor, defined as $\eta = \boldsymbol{x}^T\boldsymbol{\beta}$. However, in general we do not want to model the mean directly, but instead have a *link function*, $g(\mu)$, mapping the mean to the linear predictor, that is

$$g(\mu) = \eta. \tag{3.7}$$

Since for the normal distribution $\mu = \eta$, the link function is the identity function, $g(\mu) = \mu$. If the link function also equals the canonical parameter θ, that is

$$g(\mu) = \theta,$$

then $g(\mu)$ is called a *canonical link function*. So in general we have a response variable distributed according to some distribution ψ

$$y \sim \psi(\mu, \sigma^2), \tag{3.8}$$

where the mean μ is connected to the linear predictor η through a link function $g(\mu)$.

3.2.2 GLM for a Bernoulli process

When we model the spikes of neurons, the event of spiking at each timestep is a Bernoulli distributed response variable, receiving input stimulus from other neurons and background noise. This can be modelled as a GLM, with the input being modelled through a linear predictor.

A Bernoulli variable takes the values $\{0,1\}$ with probabilities $\{1-\mu, \mu\}$ respectively. The probability density function for a Bernoulli variable is

$$f(y|\mu) = \mu^y(1-\mu)^{1-y}, \tag{3.9}$$

which can be rewritten in the GLM framework given in (3.5) as follows

$$f(y|\mu) = \exp\left(y \cdot \ln\left(\frac{\mu}{1-\mu}\right) + \ln\left(1-\mu\right)\right). \tag{3.10}$$

Without proof, comparison with (3.5) gives that

$$\begin{aligned} \theta &= \ln\left(\frac{\mu}{1-\mu}\right) \\ b(\theta) &= \ln\left(1+\exp(\theta)\right), \end{aligned} \tag{3.11}$$

and $c(y,\phi,w) = 0$, $w = \phi = 1$. Recall the canonical link function, $g(\mu) = \theta$, which we can see from (3.11) has to be

$$g(\mu) = \ln\left(\frac{\mu}{1-\mu}\right), \tag{3.12}$$

which is called the *logit link funciton*. This also maps the expected value to the linear predictor, η,

$$\ln\left(\frac{\mu}{1-\mu}\right) = \eta = \boldsymbol{x}^T\boldsymbol{\beta}. \tag{3.13}$$

Solving for μ in (3.13), we obtain the expected value expressed through the linear predictor, with the *inverse logit function*

$$\mu = \frac{\exp(\eta)}{1+\exp(\eta)}. \tag{3.14}$$

3.3 Parameter estimation

An essential part of statistical analysis is the estimation of unknown model parameters given experimental data. Recall from section 3.1.1 that we are considering two simultaneously evolving processes, being 1) *the evolution of the network connectivity* and 2) *the spike history of the neurons*, respectively. Both of these processes are modelled with different parameters, that have to be inferred from only empirical neural data. In this work we will use two different approaches for estimating parameters.

3.3.1 Maximum likelihood estimation

Recall from section 3.2.2 that we model the neuronal spiking through a Bernoulli GLM. To estimate the parameters of this model, contained in the linear predictor, we use maximum likelihood estimation (MLE) [23]. Technicalities and a precise presentation of the model will be given in the next section, but first, we introduce the mathematical tools.

The likelihood function for independent Bernoulli random variables y_i is given as the product of the likelihoods for the independent observations

$$L(\mu) = \prod_{i=1}^{n} L_i(\mu_i) = \prod_{i=1}^{n} f(y_i|\mu_i) = \prod_{i=1}^{n} \mu_i^{y_i}(1-\mu_i)^{1-y_i} \tag{3.15}$$

where μ_i is the expected value of the Bernoulli variable y_i. Now however, we would like to express this in the GLM framework. That is, our mean μ_i actually depends on other parameters, $\boldsymbol{\beta}$ through a linear predictor and a link function, seen from equation (3.14). We therefor write

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \mu_i(\boldsymbol{\beta})^{y_i}(1-\mu_i(\boldsymbol{\beta}))^{1-y_i}. \tag{3.16}$$

The idea is now to estimate the parameters in $\boldsymbol{\beta}$, by choosing those that maximise the probability of our observations, given by equation (3.16). Often it is more tractable to work with the logarithm of $L(\boldsymbol{\beta})$, and since the logarithm is a monotonically increasing function, $\ln(L(\boldsymbol{\beta}))$ will have the same maximum likelihood estimates as $L(\boldsymbol{\beta})$. Applying the logarithm to (3.16) yields

$$l(\boldsymbol{\beta}) = \ln(L(\boldsymbol{\beta})) = \sum_{i=1}^{n} l_i(\boldsymbol{\beta}) = \sum_{i=1}^{n} y_i \ln\left(\frac{\mu_i(\boldsymbol{\beta})}{1-\mu_i(\boldsymbol{\beta}}\right) + \ln\left(1-\mu_i(\boldsymbol{\beta})\right) \tag{3.17}$$

as we derived in (3.14), the expected value, μ, can be expressed with $\boldsymbol{\beta}$ through the linear predictor. By exchanging this expression to (3.17), we obtain

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{n} l_i(\boldsymbol{\beta}) = \sum_{i=1}^{n} y_i \boldsymbol{x_i^T}\boldsymbol{\beta} - \ln\left(1+\exp(\boldsymbol{x_i^T}\boldsymbol{\beta})\right). \tag{3.18}$$

As mentioned, the goal is to find the $\boldsymbol{\beta}$'s maximising the log-likelihood. Since $l(\boldsymbol{\beta})$ is a strictly concave function [23], this can be done numerically with some gradient method. One of the most common methods, which we will use later, is the *Fisher scoring algorithm*. This makes use of the so-called *Score function* and *Fisher information matrix*, where the two provide information about the first and the second derivative of the log-likelihood, respectively.

For convenience, we now again write $\mu_i(\boldsymbol{\beta})$ instead of $\frac{\exp(\boldsymbol{x_i^T}\boldsymbol{\beta})}{1+\exp(\boldsymbol{x_i^T}\boldsymbol{\beta})}$. The score function is defined as

$$\boxed{s(\boldsymbol{\beta}) = \frac{\partial l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{n} s_i(\boldsymbol{\beta}) = \sum_{i=1}^{n} \boldsymbol{x_i}\left(y_i - \mu_i(\boldsymbol{\beta})\right)}. \tag{3.19}$$

The Fisher information matrix is defined as

$$F(\boldsymbol{\beta}) = \mathrm{E}\left(-\frac{\partial^2 l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} \right)$$

or equivalently

$$F(\boldsymbol{\beta}) = \mathrm{Cov}(s(\boldsymbol{\beta})) = \sum_{i=1}^{n} \mathrm{Cov}(s_i(\boldsymbol{\beta})),$$

where the sum is obtained by assuming independent observations. Note that $\mathrm{E}[s_i(\boldsymbol{\beta})] = 0$ as $\mathrm{E}[y_i] = \mu_i(\boldsymbol{\beta})$. Using this, and the definition of covariance we obtain

$$\sum_{i=1}^{n} \mathrm{Cov}(s_i(\boldsymbol{\beta})) = \sum_{i=1}^{n} \mathrm{E}\Big[\Big(s_i(\boldsymbol{\beta}) - \mathrm{E}[s_i(\boldsymbol{\beta})] \Big) \Big(s_i(\boldsymbol{\beta}) - \mathrm{E}[s_i(\boldsymbol{\beta})] \Big)^T \Big] = \sum_{i=1}^{n} \mathrm{E}\big[s_i(\boldsymbol{\beta}) s_i(\boldsymbol{\beta})^T \big].$$

Using the expression for $s_i(\boldsymbol{\beta})$ from earlier yields the following

$$F(\boldsymbol{\beta}) = \sum_{i=1}^{n} \mathrm{E}\big[s_i(\boldsymbol{\beta}) s_i(\boldsymbol{\beta})^T \big] = \sum_{i=1}^{n} \boldsymbol{x_i} \boldsymbol{x_i^T} \mathrm{E}\big[\big(y_i - \mu_i(\boldsymbol{\beta}) \big)^2 \big].$$

And finally, by exploiting that $\mathrm{E}\Big[\big(y_i - \mu_i(\boldsymbol{\beta}) \big)^2 \Big]$ is the variance of y_i, we arrive at our final expression

$$\boxed{F(\boldsymbol{\beta}) = \sum_{i=1}^{n} \boldsymbol{x_i} \boldsymbol{x_i^T} \mu_i(\boldsymbol{\beta}) \big(1 - \mu_i(\boldsymbol{\beta}) \big)}. \tag{3.20}$$

The Fisher scoring algorithm is closely related to Newton's method, where we iteratively search for $\boldsymbol{\beta}$ minimising the log-likelihood. The Fisher scoring algorithm is defined by

$$\boldsymbol{\beta}^{(i+1)} = \boldsymbol{\beta}^{(i)} + \big(F(\boldsymbol{\beta}^{(i)}) \big)^{-1} s(\boldsymbol{\beta}^{(i)}). \tag{3.21}$$

The algorithm will converge towards the solution $s(\boldsymbol{\beta}) = \frac{\partial l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0$, and because $l(\boldsymbol{\beta})$ is strictly concave, this will indeed converge towards the maximum likelihood estimate $\hat{\boldsymbol{\beta}}_{\mathrm{MLE}}$.

3.3.2 Bayesian estimation

Bayesian estimation is an estimation which aims to minimise the average expected value of a chosen loss function over the posterior distribution [24]. Using Bayes' rule, the posterior distribution of the parameter $\boldsymbol{\beta}$ given the data y can be written as

$$f(\boldsymbol{\beta}|y) = \frac{f(y|\boldsymbol{\beta})f(\boldsymbol{\beta})}{f(y)} \tag{3.22}$$

where the denominator $f(y) = \int f(y,\boldsymbol{\beta})d\boldsymbol{\beta}$ is a normalisation factor. The estimator for our parameter becomes

$$\hat{\boldsymbol{\beta}} = \arg\min_{\hat{\boldsymbol{\beta}}} \mathrm{E}\left[C(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta})\right] \tag{3.23}$$

where C is a non-negative cost function. Using that $\mathrm{E}(x) = \int xf(x)dx$, we can calculate

$$\hat{\boldsymbol{\beta}} = \arg\min_{\hat{\boldsymbol{\beta}}} \int_{-\infty}^{\infty} C(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta})f(\boldsymbol{\beta}|y)d\boldsymbol{\beta}, \tag{3.24}$$

and insert a desired choice of cost function to obtain an estimate. In our application we consider the absolute error, the mean squared error, and the 0-1 loss function, leading to three different estimators.

Absolute error

For the absolute error, the cost function is defined as

$$C_1(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}) = \left|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\right|. \tag{3.25}$$

Inserting this cost function into (3.24) yields

$$\begin{aligned}\hat{\boldsymbol{\beta}} &= \arg\min_{\hat{\boldsymbol{\beta}}} \int_{-\infty}^{\infty} \left|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\right| f(\boldsymbol{\beta}|y)d\boldsymbol{\beta} \\ &= \arg\min_{\hat{\boldsymbol{\beta}}} \left(\int_{-\infty}^{\hat{\boldsymbol{\beta}}} \left(\boldsymbol{\beta} - \hat{\boldsymbol{\beta}}\right) f(\boldsymbol{\beta}|y)d\boldsymbol{\beta} + \int_{\hat{\boldsymbol{\beta}}}^{\infty} \left(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\right) f(\boldsymbol{\beta}|y)d\boldsymbol{\beta} \right).\end{aligned}$$

To find the minimum, we take the derivative with respect to $\hat{\boldsymbol{\beta}}$,

$$\frac{d}{d\hat{\boldsymbol{\beta}}} \left(\int_{-\infty}^{\hat{\boldsymbol{\beta}}} \left(\boldsymbol{\beta} - \hat{\boldsymbol{\beta}}\right) f(\boldsymbol{\beta}|y)d\boldsymbol{\beta} + \int_{\hat{\boldsymbol{\beta}}}^{\infty} \left(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\right) f(\boldsymbol{\beta}|y)d\boldsymbol{\beta} \right) = 0. \tag{3.26}$$

We will also need to check that the obtained estimator indeed provides a minimum, which can be done by checking that the second derivative is positive. Using the fundamental theorem of calculus, and the product rule of deriva-

tives,

$$\begin{aligned}\frac{d}{d\hat{\boldsymbol{\beta}}}\int_{-\infty}^{\hat{\boldsymbol{\beta}}}\left(\boldsymbol{\beta}-\hat{\boldsymbol{\beta}}\right)f(\boldsymbol{\beta}|y)d\boldsymbol{\beta} &= \hat{\boldsymbol{\beta}}f(\hat{\boldsymbol{\beta}}|y) - \frac{d}{d\hat{\boldsymbol{\beta}}}\left(\hat{\boldsymbol{\beta}}\int_{-\infty}^{\hat{\boldsymbol{\beta}}}f(\boldsymbol{\beta}|y)d\boldsymbol{\beta}\right)\\ &= \hat{\boldsymbol{\beta}}f(\hat{\boldsymbol{\beta}}|y) - \left(\hat{\boldsymbol{\beta}}f(\hat{\boldsymbol{\beta}}|y) + \int_{-\infty}^{\hat{\boldsymbol{\beta}}}f(\boldsymbol{\beta}|y)d\boldsymbol{\beta}\right)\\ &= -\int_{-\infty}^{\hat{\boldsymbol{\beta}}}f(\boldsymbol{\beta}|y)d\boldsymbol{\beta},\end{aligned}$$

and similarly

$$\frac{d}{d\hat{\boldsymbol{\beta}}}\int_{\hat{\boldsymbol{\beta}}}^{\infty}\left(\hat{\boldsymbol{\beta}}-\boldsymbol{\beta}\right)f(\boldsymbol{\beta}|y)d\boldsymbol{\beta} = \int_{\hat{\boldsymbol{\beta}}}^{\infty}f(\boldsymbol{\beta}|y)d\boldsymbol{\beta},$$

which, inserted into (3.26) means that

$$\int_{-\infty}^{\hat{\boldsymbol{\beta}}}f(\boldsymbol{\beta}|y)d\boldsymbol{\beta} = \int_{\hat{\boldsymbol{\beta}}}^{\infty}f(\boldsymbol{\beta}|y)d\boldsymbol{\beta}.$$

Because we know that this posterior $f(\hat{\boldsymbol{\beta}}|y)$ is a probability distribution, it must sum to one over the entire plane,

$$\begin{aligned}\int_{-\infty}^{\infty}f(\boldsymbol{\beta}|y)d\boldsymbol{\beta} = 2\int_{-\infty}^{\hat{\boldsymbol{\beta}}}f(\boldsymbol{\beta}|y)d\boldsymbol{\beta} &= 1\\ \implies \int_{-\infty}^{\hat{\boldsymbol{\beta}}}f(\boldsymbol{\beta}|y)d\boldsymbol{\beta} &= \frac{1}{2}.\end{aligned}$$

Our estimator $\hat{\boldsymbol{\beta}}$ is therefore the median of the posterior.

Mean square error

For the mean square error, the cost function is defined as

$$C_2(\hat{\boldsymbol{\beta}}-\boldsymbol{\beta}) = \left|\hat{\boldsymbol{\beta}}-\boldsymbol{\beta}\right|^2. \tag{3.27}$$

As for the absolute error, we insert this cost function into (3.24),

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \arg\min_{\hat{\boldsymbol{\beta}}} \int_{-\infty}^{\infty} \left|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\right|^2 f(\boldsymbol{\beta}|y)\, d\boldsymbol{\beta} \\
&\implies \frac{d}{d\hat{\boldsymbol{\beta}}} \int_{-\infty}^{\infty} \left(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\right)^2 f(\boldsymbol{\beta}|y)\, d\boldsymbol{\beta} = 0 \\
&\implies 2\hat{\boldsymbol{\beta}} \int_{-\infty}^{\infty} f(\boldsymbol{\beta}|y)\, d\boldsymbol{\beta} = 2\int_{-\infty}^{\infty} \boldsymbol{\beta} f(\boldsymbol{\beta}|y)\, d\boldsymbol{\beta} \\
&\implies \hat{\boldsymbol{\beta}} = \int_{-\infty}^{\infty} \boldsymbol{\beta} f(\boldsymbol{\beta}|y)\, d\boldsymbol{\beta} = E(\boldsymbol{\beta}),
\end{aligned}$$

so the estimator is therefore the mean of the posterior distribution.

0-1 loss function

For the 0-1 loss function, the cost function is

$$C_3(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}) = \begin{cases} 0 & \text{if } |\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}| < \delta \\ 1 & \text{if } |\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}| > \delta \end{cases}, \tag{3.28}$$

where $\delta = 0$ in our case. Following the same procedure as for the previous estimators, we now obtain

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \arg\min_{\hat{\boldsymbol{\beta}}} \int_{-\infty}^{\infty} C_3(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}) f(\boldsymbol{\beta}|y)\, d\boldsymbol{\beta} \\
&= \arg\min_{\hat{\boldsymbol{\beta}}} \left(\int_{-\infty}^{\hat{\boldsymbol{\beta}}-\delta} 1 \cdot f(\boldsymbol{\beta}|y)\, d\boldsymbol{\beta} + \int_{\hat{\boldsymbol{\beta}}+\delta}^{\infty} 1 \cdot f(\boldsymbol{\beta}|y)\, d\boldsymbol{\beta} \right) \\
&= \arg\min_{\hat{\boldsymbol{\beta}}} \left(1 - \int_{\hat{\boldsymbol{\beta}}-\delta}^{\hat{\boldsymbol{\beta}}+\delta} f(\boldsymbol{\beta}|y)\, d\boldsymbol{\beta} \right) \\
&= \arg\max_{\hat{\boldsymbol{\beta}}} \int_{\hat{\boldsymbol{\beta}}-\delta}^{\hat{\boldsymbol{\beta}}+\delta} f(\boldsymbol{\beta}|y)\, d\boldsymbol{\beta},
\end{aligned}$$

which, for smooth and quasi-concave [25] posteriors $f(\boldsymbol{\beta}|y)$, as $\delta \to 0$, will give the estimator

$$\hat{\boldsymbol{\beta}} = \arg\max_{\boldsymbol{\beta}} f(\boldsymbol{\beta}|y).$$

The estimator is therefore the mode of the posterior, also known as *Maximum a Posteriori* (MAP).

3.4 Cross-correlation estimation

A *time series* can be defined as a sequence of random variables indexed according to the order they are obtained in time [26]. Hence, the spike train of a neuron can ultimately be considered as a time series. As mentioned, we will work with real data containing information about more than 400 000 neuron pairs. Our first task would be to identify neuron pairs that actually seem to be connected, before applying the full method. It could be wasteful to analyse each pair of neurons comprehensively, without even having an indication whether they might be connected or not. For rather quickly detecting possible connections, we will make use of cross-correlation estimation of different spike trains, where we apply the framework presented by Haugh [27].

The idea is to consider small time lags, denoted by k, for one of the series. If the processes somehow depend on each other, we would experience correlation for certain time lags, indicating the required time for one response to influence the other. This would in our case be the time a signal needs to travel from the presynaptic neuron to the postsynaptic one.

Denote two time series s_1 and s_2 for some time units $t = 0, 1, ..., N$ and let $\hat{r}_{s1,s2}(k)$ be the estimated correlation function for some time lag k, being a discrete amount of timesteps. To assure we do not calculate non existing events outside of the domain, we define it as

$$\hat{r}_{s1,s2}(k) = \begin{cases} \frac{\frac{1}{N}\sum_{t=0}^{N-k}(s_1(t+k)-\bar{s}_1)\cdot(s_2(t)-\bar{s}_2)}{\sigma_{s_1}\sigma_{s_2}}, & \text{for } k > 0 \\ \frac{\frac{1}{N}\sum_{t=0-k}^{N}(s_1(t+k)-\bar{s}_1)\cdot(s_2(t)-\bar{s}_2)}{\sigma_{s_1}\sigma_{s_2}}, & \text{for } k < 0 \end{cases} \tag{3.29}$$

where N is the number of timesteps, $\bar{s}_1$ and $\bar{s}_2$ are the means of the two time series, and σ_{s_1} and σ_{s_2} are the standard deviations of the s_1 and s_2.

From this we can obtain estimates for the correlation at different time lags. Furthermore, we would like to say something about the significance of the observed correlation, which we do in the following fashion.

Lets assume no autocorrelation for both time series, which basically means that the internal events of the individual series are independent of each other. Then we create a hypotheses test for the estimation for some time lag k, namely

$$H_0 : \ \hat{r}_{s1,s2}(k) = 0 \qquad \text{vs} \qquad H_1 : \ \hat{r}_{s1,s2}(k) \neq 0. \tag{3.30}$$

Under the assumption of H_0, it is shown in the article [27] that the estimator

asymptotically follows an approximate normal distribution

$$\hat{r}_{s1,s2}(k) \sim N\Big(0, \frac{1}{N-|K|}\Big). \tag{3.31}$$

Using this we can easily calculate a confidence interval for the estimator under the assumption of H_0, based on the normal distribution, for some desired significance value α. If the observed correlation lies outside this interval, we might reject the null hypothesis, and conclude that the correlation seems significant, that is, $\hat{r}_{s1,s2}(k) \neq 0$.

From this, one could also construct more sophisticated and accurate test-statistics, being for instance chi-squared distributed or F-distributed. We will however apply this more simple framework, as we consider this to be accurate enough for our purpose. For more developed methods, we can recommend the interested reader to the paper "*Tests for non-correlation of two multivariate time series: a nonparametric approach*" [28].

Chapter 4

Methods

This chapter will be dedicated to explaining our experimental setup, the detailed model we consider, as well as the procedures for inferring and estimated the relevant parameters in order to achieve an understanding of the learning rule and evolution of the brain.

In section 4.1 we will introduce the model framework and the concrete neural setting at hand. Here the learning rule will be presented, and we propose a modification of the learning rule which have been used for efficiency and biological purposes.

For parameter estimation in the Bayesian framework, Markov Chain Monte Carlo (MCMC) methods are a common choice within statistics. In section 4.2 we will introduce one such algorithm called Metropolis-Hastings sampling, from which we can draw empirical samples estimating the posterior distributions of the learning rule parameters, also being called our *target distribution*. We will present the algorithm, employing a *proposal distribution*, which again is important in order to replicate the target distribution. In this section we also propose a possible improvement of the algorithm, given the nature of our problem.

In section 4.3 we extend the Metropolis-Hastings algorithm to a so-called *Particle Metropolis-Hastings algorithm*, which will help us calculate desired probabilities for our case, and make sure our Metropolis-Hastings algorithm hopefully yields good samples. How this *particle filtering* procedure looks like, will be presented both through theory and plots. The chapter will end with the full algorithm, which can be used for reproducing the results in chapter 5.

4.1 Model description

As already discussed, the brain consists of many different big networks of neurons, where several neurons are interconnected and interacting with each other. However, for simplicity we are so far only considering two neurons, and analysing the synapse between them, whose weight is evolving according to the spike trains of the neurons and the learning rule.

4.1.1 Biological setting

The spike trains of the neurons are defined according to section 2.3. We are considering a time domain of $[0, T]$ for $T = 120$ seconds, and more precisely we divide this into a grid of timebins of size δt. We will begin with a binsize of $\delta t = 5$ms, giving us 24000 timebins. However, other choices will be discussed later. The timebins will be numbered as $t \in \{0, 1, ..., T-1\}$. Hence time t refers to the time interval contained in timebin t. Let us first denote some of the essential notation for the biological system.

- w^t: the weight of the synapse, interpreted as the strength of the connection between neuron 1 to neuron 2 at time t.
- $w^{0:t}$: the weight trajectory up until time t.
- s_1^t, s_2^t: the spiking event at time t of neuron 1 and 2 respectively. s_1^t, s_2^t $\in \{0, 1\}$, denoting no spike or spike respectively.
- $s_1^{0:t}$, $s_2^{0:t}$: the spike train history of neuron 1 and 2 respectively, up until time t.
- b_1, b_2: the constant background input of neuron 1 and 2 respectively, from other neurons or brain areas which we do not explicitly consider.

We will only consider the synapse in one direction, so that neuron 2 can have input from neuron 1, but not oppositely. Figure 4.1 illustrates the considered model and the input both neurons receive.

Recall from 3.2.2, that the spiking events are modelled through a Bernoulli GLM, receiving input through the linear predictor η. That is, s_1^t and s_2^t are Bernoulli processes with some expected values μ_1^t and μ_2^t respectively, expressing the probability of the neuron spiking at time t. These are also referred to as *spike rates*. Also recall how these expected values are expressed

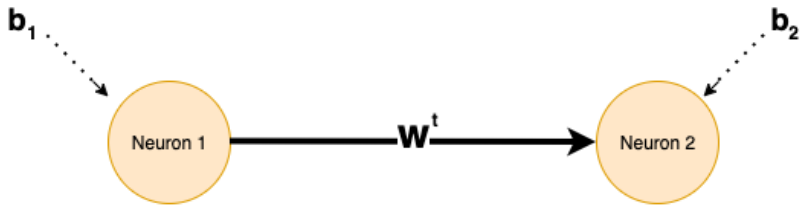

Figure 4.1: Our experimental setup with two neurons, and a connection from neuron 1 to neuron 2. Both neurons receive a constant background noise.

through the linear predictor η_i^t, representing the total input to neuron i at time t,

$$\mu_i^t = \frac{\exp(\eta_i^t)}{1+\exp(\eta_i^t)}. \tag{4.1}$$

As we see from figure 4.1, neuron 1 only has its constant background noise b_1 as input, hence

$$\eta_1^t = b_1.$$

For neuron 2 however, the input is time-dependent, as the input is a combination of the constant background noise b_2 and the input from neuron 1. We model this as

$$\eta_2^t = w^{t-1} \cdot s_1^{t-1} + b_2,$$

so if the neurons are connected $\left(w^{t-1} \neq 0\right)$, a spike of neuron 1 $\left(s_1^{t-1} = 1\right)$ can either inhibit $\left(w^{t-1} < 0\right)$ or excite $\left(w^{t-1} > 0\right)$ neuron 2 by decreasing or increasing its input with respect to the baseline b_2.

Hence the spiking of neuron 1 is modelled as a homogeneous Bernoulli process, while neuron 2 is driven by a non-homogeneous Bernoulli process

$$\begin{aligned} P\left(s_1^t = 1\right) &\sim \text{Ber}(\mu_1^t), \qquad & \mu_1^t &= \frac{\exp(b_1)}{1+\exp(b_1)}, \\ P\left(s_2^t = 1\right) &\sim \text{Ber}(\mu_2^t), \qquad & \mu_2^t &= \frac{\exp(w^{t-1} \cdot s_1^{t-1} + b_2)}{1+\exp(w^{t-1} \cdot s_1^{t-1} + b_2)}. \end{aligned} \tag{4.2}$$

4.1.2 Learning rule

Now, we will define the STDP learning rule, that we discussed in 2.4. This was first proposed by Abott and coauthors [2]. The learning rule considers the time between spikes, *interspike intervals*, of the neurons to determine the update of the synaptic weight.

The weight evolves according to the following relation

$$w^{t+1} = w^t + l(s_1^{0:t}, s_2^{0:t}, \theta) + \epsilon(\sigma) \tag{4.3}$$

where $\epsilon(\sigma)$ is a Gaussian white noise, while $l(s_1^{0:t}, s_2^{0:t}, \theta)$ is the STDP learning rule for parameters $\theta = \{A_+, A_-, \tau_+, \tau_-\}$, which is defined as

$$\begin{aligned} l(s_1^{0:t}, s_2^{0:t}, \theta) &= l_+(s_1^{0:t}, s_2^{0:t}, A_+, \tau_+) - l_-(s_1^{0:t}, s_2^{0:t}, A_-, \tau_-), \\ l_+(s_1^{0:t}, s_2^{0:t}, A_+, \tau_+) &= s_2^t \cdot \sum_{t'=0}^{t} s_1^{t'} A_+ \exp\left(\frac{t'-t}{\tau_+}\right), \\ l_-(s_1^{0:t}, s_2^{0:t}, A_-, \tau_-) &= s_1^t \cdot \sum_{t'=0}^{t} s_2^{t'} A_- \exp\left(\frac{t'-t}{\tau_-}\right). \end{aligned} \tag{4.4}$$

The parameters θ are the ones we are ultimately wishing to infer. They are a good indication of how the evolution in the brain looks like, given the activity of neurons, and could be essential for a further understanding of brain diseases.

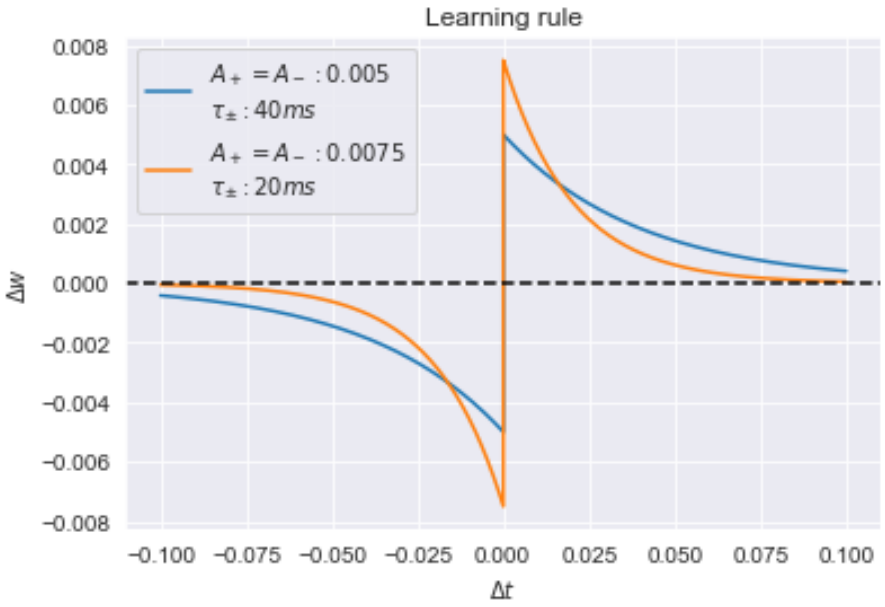

Figure 4.2: Two different learning rule shapes. A_+ and A_- are the maximum and minimum values. The lower value for $\tau_\pm$, the faster it converges to zero. Δt denotes the size of the interspike intervals.

The parameters A_+ and A_- can be interpreted as the scale of the weight update. For really small interspike intervals, the value of the learning rule approaches maximum values A_+ and A_- for either a positive or negative update. τ_+ and τ_- are describing the domain of relevant interspike intervals. The bigger these parameters are, the faster the contributions from bigger interspike intervals decays to zero. In figure 4.2 the shape of the learning rule is

illustrated, and the effects of these parameters become clear. From now, we will use values for θ proposed by Linderman and coauthors in [1], namely

$$\begin{aligned} A_+ &= 0.005, \\ A_- &= 1.05 \cdot A_+ = 0.00525, \\ \tau_+ &= \tau_- = 0.02\text{s}. \end{aligned} \tag{4.5}$$

Modified learning rule

We would want to propose a modification to the learning rule, which we believe would make the algorithm faster, which is biologically supported, and which does not really affect the mathematical calculations.

From the original definition of the STDP learning rule (4.4), we would have to sum over all spiking times of the neurons. However, from figure 4.3, we can see that the contributions for interspike intervals greater than $5\tau_\pm$ms are getting really small. Hence, for calculating the learning rule, we propose only to consider a time window of the last $10\tau_\pm$ms of the spike history. This would reduce the number of required calculations a lot, which speeds up the algorithm significantly. Also, we expect that a biological mechanism keeping track of pre/post-synaptic activity would be energetically expensive, in addition to being computationally inefficient, such that older spiking events than that should be independent of the current activity. Thirdly, these contributions for bigger than $10\tau_\pm$ms interspike intervals are so small that they can be neglected without affecting the network dynamics, which will be shown in the next section. The choice of $10\tau_\pm$ instead of for example $5\tau_\pm$ was made because the efficiency of the algorithm barely changed.

To assure that we do not consider negative times, we define at time t,

$$\gamma_t = \max\Big\{0, t - \frac{10\tau_\pm}{\delta t}\Big\},$$

where δt again is the binsize. So γ_t will be the number of timebins of the spike history we should consider in the learning rule. This leads to the modified learning rule being

$$\begin{aligned} l^m(s_1^{0:t}, s_2^{0:t}, \theta) &= l_+^m(s_1^{0:t}, s_2^{0:t}, A_+, \tau_+) - l_-^m(s_1^{0:t}, s_2^{0:t}, A_-, \tau_-), \\ l_+^m(s_1^{0:t}, s_2^{0:t}, A_+, \tau_+) &= s_2^t \cdot \sum_{t'=\gamma_t}^{t} s_1^{t'} A_+ \exp\Big(\frac{t'-t}{\tau_+}\Big), \\ l_-^m(s_1^{0:t}, s_2^{0:t}, A_-, \tau_-) &= s_1^t \cdot \sum_{t'=\gamma_t}^{t} s_2^{t'} A_- \exp\Big(\frac{t'-t}{\tau_-}\Big). \end{aligned} \tag{4.6}$$

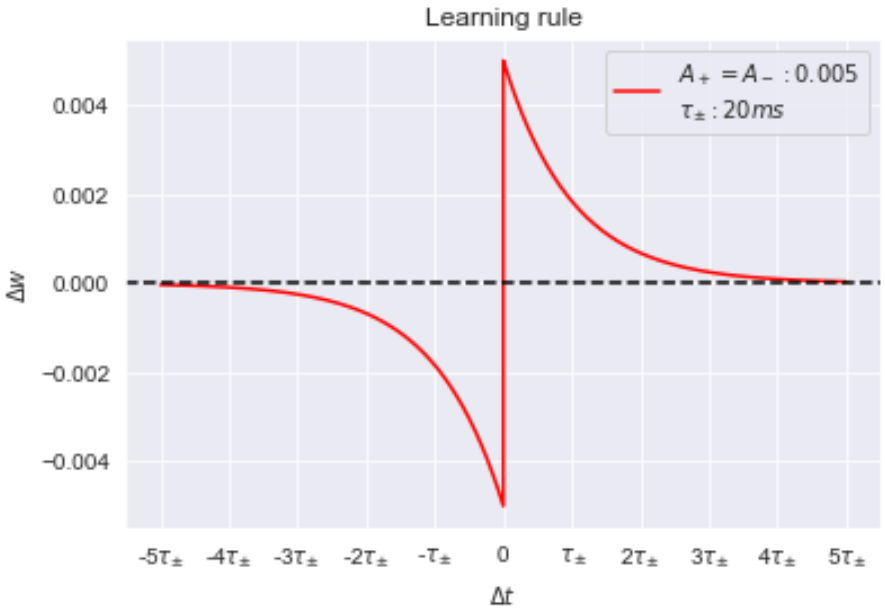

Figure 4.3: The learning rule for our specific parameters plotted in $\Delta t \in [-100\text{ms}, 100\text{ms}]$, showing the relation to $\tau_\pm$. Values for $t > 5\tau_\pm$ are becoming really small.

4.1.3 Simulating data sets

When developing the algorithm, and making sure that it works as expected, we are performing inference on a simulated data set, which is beneficial because we then know the underlying true parameter values.

In generating our spike train data set we use the parameter values of θ as explained in the previous section. Other parameters we need to set are the initial connectivity strength, w^0, as well as the background noises b_1 and b_2. Based on the work of Linderman [1], where a baseline firing rate of the neurons of $20s^{-1}$ is suggested, we decided to use the following values

$$b_1 = b_2 = -2,$$
$$w^0 = 1.$$

Also, we have the variance, σ^2, of the white noise, which we have experimented with, testing values ranging from 2% to 100% of the value of A_+.

We generate data sets using the mentioned constants and parameters, by iteratively drawing Bernoulli random variables for the two neurons according to equation (4.2). At the same time, the weight is updated based on the spike history. In figure 4.4 a set of generated weight trajectories are plotted, given our chosen parameters, according to the learning rule in (4.3). Clearly, these weights trajectory would generally be unobserved for neural data, and inference will be done only with the spike trains of the neurons.

From the figure we observe that there seems to be practically no difference between the original learning rule (4.4) and our proposed learning rule (4.6), which supports our claim that the mathematical differences are negligible. We also see that with our chosen parameters, the average weight between the neurons after 120 seconds seems to be around 4.5, which indicates a strong connection being developed between them, which again in a real brain could suggest that some memory or knowledge has been stored through communication of these neurons. Using equation (4.2), the spike rate of neuron 2 would actually be around 0.9 given that neuron 1 has spiked if the weight between them is 4.5.

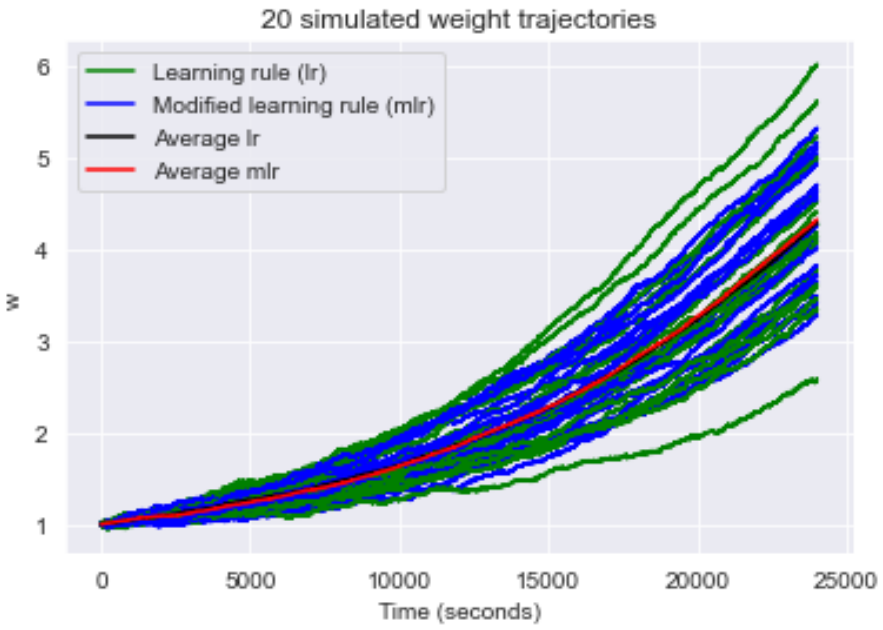

Figure 4.4: 20 randomly simulated weight trajectories, generated with a noise level of $\sigma = 0.0005$.

4.2 Metropolis-Hastings

Recall Bayesian estimation from section 3.3.2, which is the approach we adopt for inferring the learning rule parameters. In Bayesian estimation one is required to characterise the posterior of the parameters, which we here denote as

$$P(\theta|s_2^{0:T}) = \frac{P(\theta)P(s_2^{0:T}|\theta)}{\int_\Theta p(\theta')P(s_2^{0:T}|\theta')d\theta'} \tag{4.7}$$

where $s_2^{0:T}$ represents the whole spike train history of neuron 2. Notice that the dependence of the posterior and of the likelihood $P(s_2^{0:T}|\theta)$ on $s_1^{0:T}$ is here

omitted for the sake of a lighter notation. From the posterior we can then estimate the parameter values according to the three measures presented in section 3.3.2. One of the main difficulties with Bayesian estimation is that the denominator in (4.7) might be challenging or impossible to calculate. Markov Chain Monte Carlo (MCMC) methods are strong tools circumventing the calculation of the denominator. We will be using the Metropolis-Hastings algorithm, and the following subsection (4.2.1) is based on the work of Chib and Greenberg [29].

4.2.1 Algorithm

The idea of the Metropolis-Hastings algorithm is to iteratively construct a Markov chain which converges to an empirical sample with the same distribution as the target distribution. First, denote the parameter space of θ by $\mathcal{S}$. When the target distribution is the posterior (4.7), this is done by iteratively sampling new values in the parameters space, from a proposal distribution, dependent on the previous value of the parameters θ. The Markov chain can then either transition to this value, or stay at the current value. Define $T(\theta^*|\theta^{i-1})$ the transition probability for some value θ^* given the previous value θ^{i-1}. If the transition probability fulfills the *reversibility condition*, given by

$$P(\theta^*|s_2^{0:T})\,T(\theta^{i-1}|\theta^*) = P(\theta^{i-1}|s_2^{0:T})\,T(\theta^*|\theta^{i-1}). \tag{4.8}$$

It can be proven that $P(\theta|s_2^{0:T})$ is indeed an *invariant distribution* of the generated Markov chain. A detailed proof is provided in [29]. Now the question becomes, how do we construct this transition kernel $T(\theta^*|\theta^{i-1})$, in order to fulfill this reversibility condition?

Define the proposal distribution as

$$Q(\cdot|\theta^{i-1}) > 0, \qquad \int_{\mathcal{S}} Q(\theta'|\theta^{i-1})\,d\theta' = 1,$$

such that our proposal distribution is a probability density function over the parameter space, $\mathcal{S}$. In order for the Markov Chain to transition to some value θ^*, θ^* would have to be drawn from $Q(\cdot|\theta^{i-1})$ in the first place, so $T(\theta^*|\theta^{i-1})$ must be proportional to $Q(\theta^*|\theta^{i-1})$. Let us introduce a quantity $\alpha(\theta^*)$, denoting the probability of transitioning to θ^* given that θ^* was proposed. Then we clearly get

$$T(\theta^*|\theta^{i-1}) = \alpha(\theta^*) \cdot Q(\theta^*|\theta^{i-1}) \tag{4.9}$$

which we want to fulfill the equation

$$P(\theta^*|s_2^{0:T})Q(\theta^{i-1}|\theta^*)\cdot\alpha(\theta^{i-1}) = P(\theta^{i-1}|s_2^{0:T})Q(\theta^*|\theta^{i-1})\cdot\alpha(\theta^*). \tag{4.10}$$

Assume without loss of generality that the target distribution is biased for θ^{i-1}, hence we would like to transition to this value as often as possible, the fraction of how often we should transition to θ^* can then be found by setting $\alpha(\theta^{i-1}) = 1$ and solving (4.10) for $\alpha(\theta^*)$, yielding

$$\alpha(\theta^*) = \frac{P(\theta^*|s_2^{0:T})Q(\theta^{i-1}|\theta^*)}{P(\theta^{i-1}|s_2^{0:T})Q(\theta^*|\theta^{i-1})}. \tag{4.11}$$

The Markov Chain with transition probability (4.9) and a so-called acceptance ratio $\alpha(\cdot)$ as in (4.11) will then have $P(\cdot|s_2^{0:T})$ as an invariant distribution, which means we can use (4.9) to sample from it. Notice that with this scheme we have removed the problem of having to calculate the denominator in (4.7), since it cancels by the term $\frac{P(\theta^*|s_2^{0:T})}{P(\theta^{i-1}|s_2^{0:T})}$. We also know from (4.7) that the posterior is proportional to the prior distribution and the likelihood of the data, and since $\alpha(\theta^*) \in [0,1]$ we write

$$\alpha(\theta^*) = \min\left\{1, \frac{P(\theta^*)\cdot P(s_2^{0:T}|\theta^*)\cdot Q(\theta^{i-1}|\theta^*)}{P(\theta^{i-1})\cdot P(s_2^{0:T}|\theta^{i-1})Q(\theta^*|\theta^{i-1})}\right\}, \tag{4.12}$$

which leads to Algorithm 1.

Algorithm 1: Metropolis Hastings sampler

Initialise θ^0;
for *i = 1,2,...* **do**
  Draw $\theta^* \sim Q(\cdot|\theta^{i-1})$;
  Compute $\alpha(\theta^*) = \frac{P(\theta^*)\cdot P(s_2^{0:T}|\theta^*)\cdot Q(\theta^{i-1}|\theta^*)}{P(\theta^{i-1})\cdot P(s_2^{0:T}|\theta^{i-1})Q(\theta^*|\theta^{i-1})}$;
  Accept θ^* with probability $min\{1, \alpha(\theta^*)\}$;
  if θ^* *accepted* **then**
    $\theta^i = \theta^*$;
  else
    $\theta^i = \theta^{i-1}$;
  end
end

This procedure will be done over many iterations, and the resulting sequence is an empirical sample from the target distribution, from which we can find estimates for the learning rule parameters. In figure 4.5 we illustrate

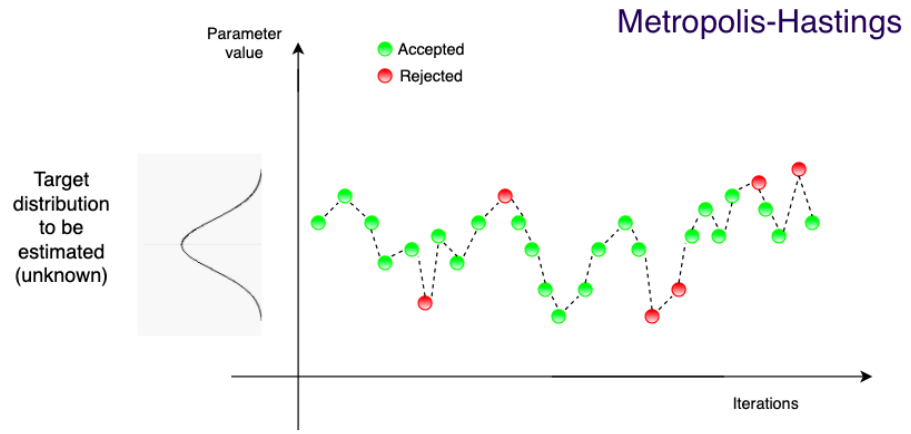

Figure 4.5: An illustration of how an arbitrary sampling process with the Metropolis-Hastings algorithm could look like for a one dimensional parameter.

how the Metropolis-Hastings sampling could behave for an arbitrary target distribution. To give the algorithm time to converge, we introduce a *burn-in time*, B, which is a number of iterations we dont include in our estimation, and the desired sample will be $\theta = \{\theta^{B+1}, \theta^{B+2}, ...\}$. Also, since we have defined the underlying constant relations for the learning rule parameters as (4.5), for the Metropolis-Hastings algorithm, our parameters for estimation are only $\theta = \{A_+, \tau_+\}$, and the two remaining parameters can be found by (4.5). From here on, τ refers to τ_+.

4.2.2 Proposal distribution

An important choice for the efficiency of the algorithm is the choice of proposal distribution $Q(\cdot|\theta^{t-1})$. Ideally we would want this to be as similar to the target distribution as possible, but clearly, the nature of the target distribution might often be unknown to us. In our case, we do not want the parameter space to include negative values, as we want both A_+, A_- and $\tau_\pm$ to be positive values. Hence we found it a reasonable choice with the gamma distribution, for both A_+ and τ. In each step, the two parameters are being proposed by their own separate gamma distributions, such that the proposal distribution becomes a product of two gammas. The gamma distribution has density function given by

$$f(x) = \frac{1}{\Gamma(\alpha)\beta^\alpha} x^{\alpha-1} e^{-\frac{x}{\beta}}, \qquad \text{for } x \in (0, \infty) \tag{4.13}$$

where α is called the *shape* parameter, and β is called a *scale* parameter. When the shape parameter gets sufficiently large, the gamma distribution

looks much like the standard distribution, which we consider being a reasonable assumption when nothing is known about the target distribution. The mean and variance of the gamma distribution are given by

$$\text{Mean} = \alpha\beta \qquad \text{Variance} = \alpha\beta^2. \tag{4.14}$$

The hyperparameters α and β for the proposal distribution are in every iteration determined according to the drawn sample. To construct a proposal distribution as effective as possible, we utilise an adaptive variance approach.

4.2.3 Adaptive variance

Ultimately, the proposal distribution can be seen as a random walk, as the means of the proposals are always set to the previous value, for A_+ and τ respectively. In general, we keep α fixed, and in every step β is found from (4.14) by setting the mean equal to the previous sample value. This is done separately for A_+ and τ to construct the two proposal distributions.

However, the variance of the proposal distribution is essential for how the algorithm behaves and performs. If the variance is too high, the proposals might make big jumps around in the parameter space, and hence many "bad" values will be proposed and many proposals will be rejected. On the other hand, if the variance is too small, the algorithm might struggle to move around and explore relevant areas of the parameter space, and might more easily get stuck.

Hence it is a good idea to also update α occasionally, in order to modify the variance, based on the behaviour of the algorithm. We define an update frequency U in terms of iterations, which is how often we update the variance. For now, we set this to $U = 100$.

It was proposed in the work of Haario and coauthors [30] to use information about the newly discovered sample variance to decide the new variance. Let $\boldsymbol{\theta}^U$ be the last U empirical samples. Then the proposed change in variance is found by solving the following set of equations

$$\text{Mean}(\boldsymbol{\theta}^U) = \alpha\beta \qquad \text{Variance}(\boldsymbol{\theta}^U) \cdot c_d^2 = \alpha\beta^2 \tag{4.15}$$

for α, and then we set the shape parameters to this value for the next U iterations, while β will remain to be calculated in every step as explained previously. Again, this procedure is done seperately for our learning rule parameters. c_d is a constant parameter depending on the dimension of our parameter space, for the one dimensional case, $c_d = 2.4$.

4.2.4 Prior distribution

For the prior distribution of the parameters for inference, it is much a matter of defining them, without really having enough information to say what such a reasonable prior for the brain would be. Linderman and coauthors [1] proposes the values $\boldsymbol{\alpha} = [1,1]$ and $\boldsymbol{\beta} = [50,100]$ for A_+ and τ respectively. We would however want to try some other priors, and propose the following values

$$\boldsymbol{\alpha} = [4,5],$$

$$\boldsymbol{\beta} = [50,100].$$

In figure 4.6 the corresponding prior distributions are visualised for a given range of parameter values.

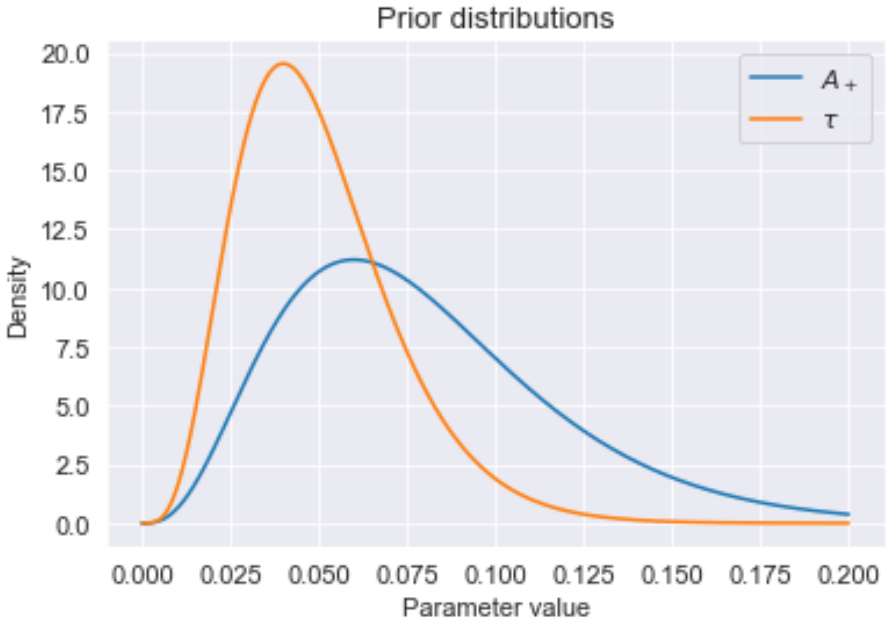

Figure 4.6: Prior distribution densities for the parameters A_+ and τ.

4.2.5 Alternating proposals

We would also like to propose an alternative approach for the Metropolis-Hastings algorithm. In Algorithm 1, recall that we iteratively obtain values for both A_+ and τ from two gamma distributions, and the acceptance ratio $\alpha(\theta^*)$ from (4.11) is computed as the product of the proposals. A possible consequence of this could for instance be that the proposal step of A_+ is a step towards a higher probability, but the proposal of τ is so bad that θ^* as

a whole is rejected. Because of this correlation between the parameters, we propose a method where we only propose steps for one of the values at a time, while the other one is kept constant as the previous sampled value. This process alternates, so that A_+ values are proposed on even-numbered iterations, and τ on odd ones, for example. We believe this could speed up the convergence of the algorithm and even produce better results.

For this algorithm, the update frequency of the variance, U would be changed to 200, and the variance of A_+ would be based on the previous 100 even iterations, similarly the variance of τ would be calculated from the previous 100 odd iterations.

4.3 Particle filtering

In section 4.2 we set up Metropolis-Hastings algorithm targeting $P(\theta|s_2^{0:T})$, which requires computing the acceptance ratio defined in (4.11). In order to do that we have to estimate the likelihood $P(s_2^{0:T}|\theta)$ first, for which we propose to use *particle filtering*. In essence, we use *particles*, sample sequences, combined with *importance weights*, to approximate this unknown distribution.

4.3.1 Importance weights

Importance weights are obtained through Importance sampling, a method that approximates our desired probability distribution, say $f(x)$ (that is difficult to sample from), by sampling from another distribution $g(x)$ that we can sample from. If we want an estimate of the mean of some other function $h(x)$, that is, $\mathrm{E}_f(h(x)) = \int h(x)f(x)dx$, we can rewrite this to be

$$\mathrm{E}_f(h(x)) = \int h(x)\frac{f(x)}{g(x)}g(x)dx,$$

meaning that we may try to estimate the mean with

$$\mathrm{E}_f(h(x)) = \mathrm{E}_g\left(\frac{h(x)f(x)}{g(x)}\right)$$

instead. A Monte Carlo estimate of the expectation of $h(x)$ with respect to the probability distribution $f(x)$, denoted $\hat{\mu}_{\mathrm{MC}}$, as described in [31], can be

obtained by drawing N samples from our distribution $f(x)$, thereby producing the sample average

$$\hat{\mu}_{\mathrm{MC}} = \frac{1}{N}\sum_{i=1}^{N} h(x_i).$$

In the case described above, where $f(x)$ is difficult to sample from directly, we use the rewritten version that has been multiplied and divided by $g(x)$, so that our estimator becomes

$$\hat{\mu}_{\mathrm{MC}} = \frac{1}{N}\sum_{i=1}^{N} h(x_i)\,v(x_i),$$

where these $v(x_i) = f(x_i)/g(x_i)$ are referred to as the (unnormalised) importance weights of the samples, and the normalised versions, $\tilde{v}$, are obtained by dividing all the other weights,

$$\tilde{v}(x_i) = \frac{v(x_i)}{\sum_{i=1}^{N} v(x_i)}. \tag{4.16}$$

In other words, importance sampling approximates $f(x)$ by using a discrete distribution with mass $v(x_i)$ for each of the observed points. In [32] it was shown that approximating using this sampling method will converge to the target distribution $f(x)$ as $N \to \infty$.

4.3.2 Sequential Monte Carlo

The observations, in the case of this work, the spike train history of neuron two, $s_2^{0:t}$ constitutes a single sample, and represents the entire history of the sequence up until time t.

Recall that we have an HMM with two stochastic processes depending on each other. The probability we want to estimate can be written as

$$P(s_2^{0:T}|\theta) = P(s_2^0|\theta)\prod_{t=1}^{T} P(s_2^t|s_2^{0:t-1},\theta), \tag{4.17}$$

and since this is also dependent on the underlying Markov process driving the connectivity of the synaptic weights, $w^{0:t}$, each factor can be expressed by marginalising over these

$$P(s_2^t|s_2^{0:t-1},\theta) = \int P(s_2^t|w^t,\theta)P(w^t|w^{t-1},s_2^{0:t},\theta)P(w^{0:t-1}|s_2^{0:t-1},\theta)\,dw^{0:t}. \tag{4.18}$$

Under the integral sign we recognise two distributions which we define in our model: the likelihood $P(s_2^t|w^t,\theta)$ and the learning rule $P(w^t|w^{t-1},s_2^{0:t},\theta)$, while $P(w^{0:t-1}|s_2^{0:t-1},\theta)$ is the posterior of the weights up to time $t-1$. Using Bayes' rule and the Markov property of the Markov process, observe that we can express $P(w^{0:t}|s_2^{0:t},\theta)$ as

$$P(w^{0:t}|s_2^{0:t},\theta) = P(w^t|w^{t-1},s_2^{0:t},\theta)\frac{P(s_2^t|w^{t-1},\theta)}{P(s_2^t|s_2^{0:t-1},\theta)}P(w^{0:t-1}|s_2^{0:t-1},\theta), \tag{4.19}$$

which expresses a recursive relation for the posterior of the weights, again involving the likelihood and the learning rule. Equation (4.19) and the similarity between its right-hand side and the integrand in equation (4.18), provide the possibility of estimating the factors $P(s_2^t|s_2^{0:t-1},\theta)$ if we have some estimate of the distribution $P(w^{0:t}|s_2^{0:t},\theta)$. For this purpose we can employ the importance sampling method, for which the desired distribution to approximate would be $P(w^{0:t}|s_2^{0:t},\theta)$. Since this is a sequence of samples, a so called *Sequential Monte Carlo* method, (SMC) (often referred to as sequential importance sampling) can be adopted. Here, instead of sampling the whole trajectory $w^{0:t}$, the idea is to iteratively append a sample w^t to the already sampled $w^{0:t-1}$ with a sampling distribution.

The SMC targeting the entire trajectory $P(w^{0:T}|s_2^{0:T},\theta)$ will yield P simulated sequences of the Markov processes, denoted by $\{w^{0:T}\}_{(p)}$, also referred to as *particles*. Together, they make up an estimated distribution of the target distribution with their normalised weights

$$\hat{P}(w^{0:T}|s_2^{0:T},\theta) = \sum_P \tilde{v}(w_{(p)}^{0:T})\delta_{w_{(p)}^{0:t}}(w^{0:T}). \tag{4.20}$$

where $\tilde{v}(w_{(p)}^{0:T})$ will be obtained from the unnormalised weights $v(w_{(p)}^{0:T})$ via equation (4.16). Now the goal is to define the unnormalised weights $v(w_{(p)}^{0:T})$ by choosing a sample distribution. In (4.19) we see that we have a term denoting the update for the process in every step, being $P(w^t|w^{t-1},s_2^{0:t},\theta)$, which is indeed just the learning rule defined in section 4.1.2. This will then be the natural choice as a sampling function, in every iteration generating the next step of the sequence particles. From (4.19), by choosing $P(w^t|w^{t-1},s_2^{0:t})$ as a sampling distribution, one gets the importance weights at time t:

$$v(w_{(p)}^{0:t}) = \frac{P(w_{(p)}^{0:t}|s_2^{0:t},\theta)}{P(w^t|w^{t-1},s_2^{0:t},\theta)} = \frac{P(s_2^t|w_{(p)}^{t-1},\theta)}{P(s_2^t|s_2^{0:t-1},\theta)}P(w_{(p)}^{0:t-1}|s_2^{0:t-1},\theta). \tag{4.21}$$

Notice that $P(s_2^t|s_2^{0:t-1},\theta)$ is just a constant independent of the generated particle sequences, which would cancel out for the normalised weights in (4.20).

Hence, the we can define the importance weights to be

$$v(w_{(p)}^{0:t}) = P(s_2^t | w_{(p)}^{t-1}, \theta) P(w_{(p)}^{0:t-1} | s_2^{0:t-1}, \theta) \tag{4.22}$$

as it would yield the exact same estimated distribution for our target distribution as (4.20). In each step we then exploit the previously generated particle sequences, and their approximated distributions analogous to (4.20), to obtain the final expression for the weight updates for particle p at time t

$$v(w_{(p)}^{0:t}) = P(s_2^t | w_{(p)}^{t-1}, \theta) \cdot \tilde{v}(w_{(p)}^{0:t-1}). \tag{4.23}$$

So our particles can be viewed as realisations of our underlying Markov process, given the proposed parameters for which we want to estimate $P(s_2^{0:T}|\theta)$. The final weights of the particles, $v(w_{(p)}^{0:T})$, will reflect the probability of this realisation generating the observed data, as in every step we update the weights using the probability of the observed spiking event for this step. After having sequentially generated these particles for the whole time domain, and calculated their weights, we can estimate the desired probability needed for the MH sampling, by using the weights of the particles at each time step,

$$P(s_2^{0:T}|\theta) = \prod_{t=0}^{T} \frac{1}{P} \sum_{p=1}^{P} v(w_{(p)}^{0:t}) \tag{4.24}$$

In figure 4.7 we have illustrated how these particle trajectories might look like for 20 realisations. Furthermore, the trajectories are plotted with transparency corresponding to their weights. We see that the strongest particles are the ones close to the underlying trajectory (black), as we expect from the theory, as these should be more likely to generate the observed data than the trajectories deviating more from the true one.

4.3.3 Resampling

As mentioned, the likelihoods for each of the particles are updated as they are propagated through time. As the synaptic weights are updated according to the stochastic learning rule, we will occasionally draw some unlikely values, resulting in a decrease of the particles' overall likelihood. One consequence of this could be that we will stumble into a situation where there is one dominating particle, and the rest are negligible in terms of importance weights. This sample will not be a good representation of our target distribution. Ideally, the particles' weights would be of similar magnitude [33].

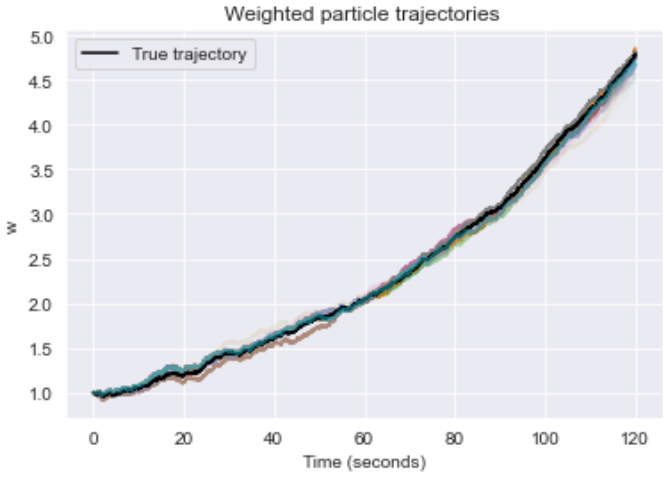

Figure 4.7: 20 particle trajectories plotted with transparency given by the final particle weights at $t = 120$s. Also featured is the underlying true trajectory (black line).

Implementing a resampling mechanism can aid the algorithm in avoiding this degeneration of the importance weights. This is done by drawing a bootstrap sample from the existing set of particles and using the likelihoods as the probabilities of drawing each particle. In essence, the likeliest particles are copied and replace the unlikely ones.

For this application, it is most suitable to use Multinomial Resampling [34]. At each time step, we have a set of particles and their respective weights. To resample we compute the normalised weights using (4.16), which, because they sum to one, work as probabilities for the corresponding particles to be resampled. We then draw a set of indexes (with replacement) using these probabilities, and obtain a new set of particles. After resampling, we assign all of the particles equal weights = $1/P$, where P is the number of particles.

Because we are replacing the unlikely particles with more likely ones, resampling causes less diversity in the synaptic weight trajectories and increases the variance. We therefore only want to resample if it is necessary. To evaluate whether or not we need to resample, we measure the effective sample size, which describes how big a sample (from the actual target distribution) our particle sample is worth. One way to measure the effective sample size is by measuring the *perplexity* of the sample, defined by the formula $\text{per}(v) = \exp(H(v))/P$ [35], where

$$H(v) = -\sum_{i=1}^{P} v^*(x_{(p)}^{1:t} \log\left(\tilde{v}(x_{(p)}^{1:t})\right) \tag{4.25}$$

is the Shannon Entropy of the sample, which captures the average level of uncertainty inherent in a variable's possible outcomes, being zero when all the importance weights' mass is concentrated in one particle and is maximised ($\text{per}(v) = \log(P)$) when all particles have identical importance weights [36]. Every time the calculated perplexity of the particle sample drops below a set threshold, we resample.

This threshold is set based on the work of Martino and coauthors [37] in which they suggest to adopt the mean of the perplexity under a uniform distribution over the P-dimensional simplex of $v * s$ (~ 0.66) as a threshold for resampling. In figure 4.8, the weight trajectories for 100 particles before and after resampling was implemented have been plotted in order to visualise the effect of resampling. The trajectories that drift too far from the actual weight trajectory have been reduced significantly, thereby improving the algorithm.

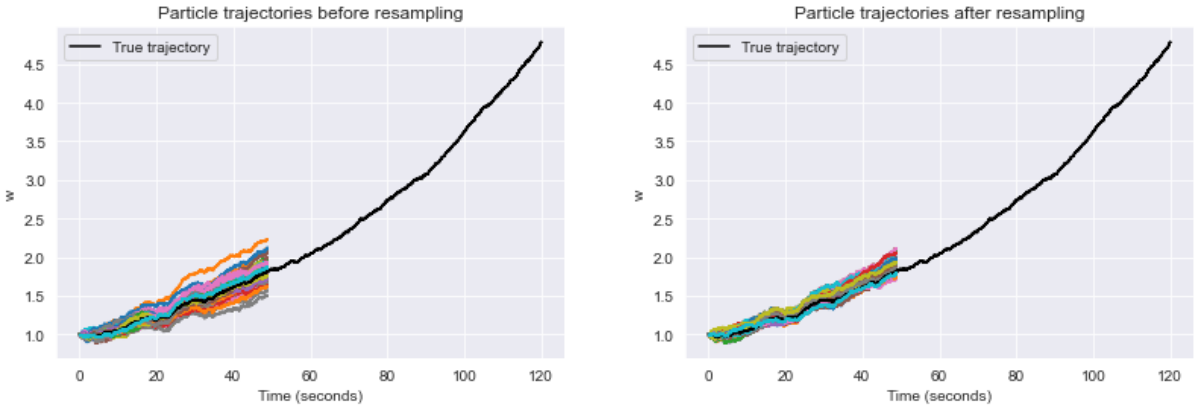

Figure 4.8: The weight trajectories for 100 particles before (left) and after (right) the resampling procedure is added to the algorithm. The black line depicts the actual true weight trajectory.

4.3.4 Algorithm/Posterior log likelihood

We now describe the algorithm that is implemented for the Sequential Monte Carlo method. The goal is to estimate the probability $P(s_2^{0:T}|\theta)$, given a fixed parameter value θ (our current proposal for the learning rule parameters). We initiate the process by drawing P values from $\pi(w^0)$, the prior distribution of the weights at $t = 0$, and these make up the set of particles for the first step. In our case, we here use a delta distribution with point mass at the value $\hat{w}^0$, being the maximum likelihood estimator of w^0. All the weights are then being initiated equal to 1. We propagate through time, appending samples to each particle, and evaluate the perplexity, checking that it is above the threshold, and if it is not, resampling is performed. After iterating through

all the time steps, a set of particles and their corresponding particle weights, are obtained, which make up a discrete approximation of the desired target distribution. The algorithm's pseudo-code is provided in Algorithm 2.

Algorithm 2: Sequential Monte Carlo

Sample $w^0_{(p)} \sim \delta(\hat{w}^0)$;
Set particle weights $v^0_{(p)} = 1$;
for $t = 1, 2, 3, \ldots$ **do**
    Sample $w^t_{(p)} \sim p(w^t_{(p)} | w^{t-1}_{(p)}, \theta)$;
    Compute particle weights $v^t_{(p)} = p(s^t_2 | w^t_{(p)}, \theta)\, \tilde{v}^{t-1}_{(p)}$;
    Compute perplexity, $\text{per}(v) = \exp(H(v))/P$;
    if *perplexity* ≤ 0.66 **then**
        Normalise the particle weights;
        Resample;
        Set weights $v^t_{(p)} = 1/P$
    end
end

4.3.5 Particle Metropolis Hasting algorithm

Now that the Sequential Monte Carlo algorithm and the use of particles has been explained in section 4.3.2, we are ready to incorporate these into our original Metropolis Hastings algorithm for inferring $P(\theta|s_2^{0:T})$. We run a Metropolis Hastings sampler, propose a θ at each iteration, and run a particle filter with this value, as explained in the pseudo-code in Algorithm 3.

Algorithm 3: Metropolis Hastings sampler with Particle filtering

Initialise θ^0;
Run particle filter, targeting $P(s_2^{0:T}|\theta^0)$;
for $i = 1, 2, \ldots$ **do**
  if $i \bmod U = 0$ **then**
    Adjust variance, as described in section 4.2.3;
  end
  Sample $\theta^* \sim Q(\cdot|\theta^{i-1})$;
  Run particle filter, targeting $P(s_2^{0:T}|\theta^*)$;
  Compute $\alpha(\theta^*) = \frac{P(\theta^*)\cdot P(s_2^{0:T}|\theta^*)\cdot Q(\theta^{i-1}|\theta^*)}{P(\theta^{i-1})\cdot P(s_2^{0:T}|\theta^{i-1})Q(\theta^*|\theta^{i-1})}$;
  Accept θ^* with probability $\min\{1, \alpha(\theta^*)\}$;
  if θ^* *accepted* **then**
    $\theta^i = \theta^*$;
  else
    $\theta^i = \theta^{i-1}$;
  end
end

Chapter 5

Inference

Whereas in Chapter 4 the method is developed and explained, in this chapter we will present our experimental results, evaluate how it performs, what challenges it experiences, and discuss further improvement of the method. If not explicitly specified, the choice for the number of particles in the SMC is set to be $P = 1000$, for the whole section.

In order to be able to estimate the learning rule parameters accurately, we must have estimates for the parameters driving the activity of the neurons, namely b_1, b_2 and w_0. Inference of these parameters will be discussed in section 5.1. We are furthermore going to evaluate the behaviour of the algorithm with respect to A_+ and τ separately, and investigate what kind of factors are affecting the estimations of the individual parameters. Sections 5.2 and 5.3 are dedicated to this.

Then, in sections 5.4 we are appraising the complete Metropolis Hastings sampling method for inferring both unknown parameters, and compare simultaneous updates of the learning rule parameters to our proposed alternating method.

After this, we prepare ourselves to actually apply these methods to real data recorded from a functioning brain. A problem which then would occur, is the presence of a Gaussian white noise of unknown magnitude, for the update of the synaptic weight. This is a parameter we would also ideally like to estimate, but which so far has been set to a constant. In section 5.5 we explore the possibility of inferring this unknown noise parameter σ, being the standard deviance of the noise.

Lastly, we can apply the method to real data and see how it performs in such a non-artificial regime. We are using neural data from the authors of

the paper "*Pyramidal Cell-Interneuron Circuit Architecture and Dynamics in Hippocampal Networks*" [3], which consists of recordings from several hundreds of thousand neuron pairs from mouse and rat brains. In section 5.7 we will discuss the data set, how we choose interesting neuron connections to analyse, and present the results of the analysis.

The computations are performed on resources provided by the IDUN/EPIC computing cluster [38].

5.1 Inference of GLM parameters

The constant input of the neurons, b_1 and b_2 are essential for the brain dynamics, and hence also for the inference of the underlying learning rule parameters. In addition, the initial connectivity between the neurons, w^0 needs to be estimated. All of these parameters are needed in the particle filtering procedure in order to estimate the likelihood of the observed data. For the estimation, we are using MLE for GLMs as explained in section 3.3.1.

For b_1 this is rather simple, as μ_1^t is constant and dependent on only b_1, recall equation (4.2). Hence we do not need to use the Fisher Scoring Algorithm, as the MLE of b_1 can be estimated directly from the observed constant firing rate, $\hat{\mu_1}$, as

$$\hat{b_1} = \frac{\hat{\mu_1}}{1-\hat{\mu_1}}. \tag{5.1}$$

For the parameters b_2 and w^0 however, we must find the estimates numerically according to the Fisher Scoring Algorithm (3.21). But, it is not straight forward, because from (4.2), our parameter w^{t-1}, which can be seen as the regression coefficient to the covariate s_1^{t-1}, is non-stationary. Recall figure 4.4, from which we observe that at the beginning of the time domain, we do not expect much learning to happen, and w^t is close to being stationary. Hence, we make this assumption for performing the Fisher Scoring Algorithm, namely that w^t is constant. The parameter w^0 is then estimated based on only the first 10 seconds of the history, where our assumption is closest to hold, and where this constant value would then be approximately w^0. b_2 can be estimated with the same assumption, but might be run over the whole domain.

From figure 5.1 we observe that the estimation of the b_1 and b_2 are quite accurate, with the 95% credible interval centered at the true value and spanning a range amounting at the 5% of the true value. The mean of 10 000

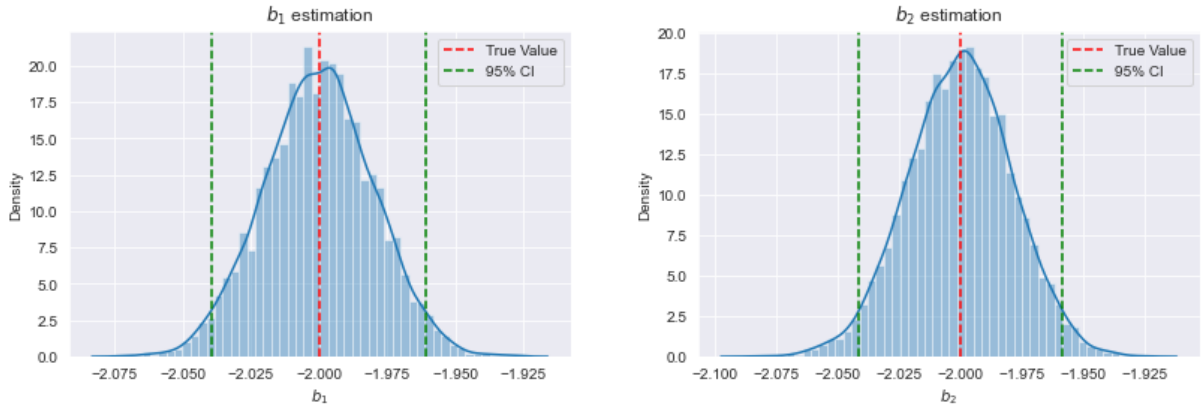

Figure 5.1: The distribution over 10000 estimations from different data sets of b_1 (left) and b_2 (right)

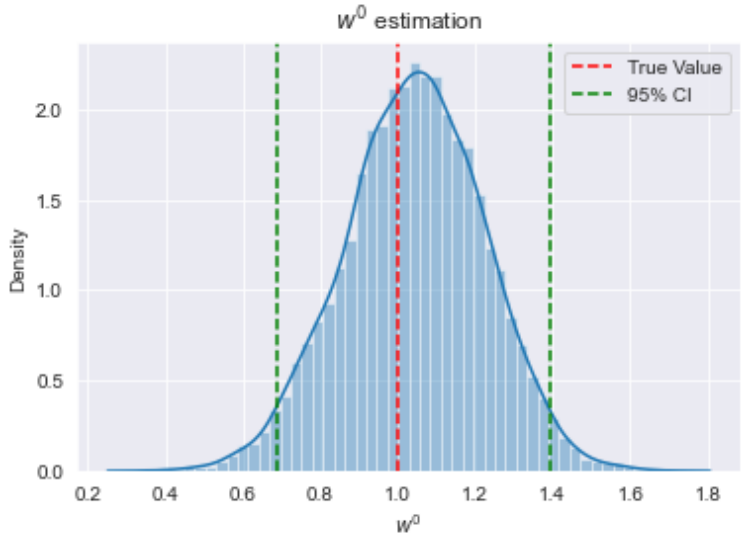

Figure 5.2: The empirical distribution over 10000 realisations of w^0 estimates.

estimations are peaked at the true value, and the variance is small, with tight credible intervals.

On the other hand, the estimation of w^0 is not as good, illustrated in figure 5.2, which indicates that the assumption we make does not hold. We observe that the mean of the estimations are shifted slightly, which makes sense if we assumed no learning, when there in reality actually exists slight learning. Also, the variance of our estimations are big, and from the credible intervals we see that values deviating up to 40% of the true one occurs relatively regularly. The variance is probably due to the limited data set we are using, which then is more sensitive to the uncertainty of the Bernoulli process.

5.2 Inference of A_+

Now we would like to investigate the ability of our algorithm to estimate A_+, when τ is being held constant at its true value. Recall from (4.6) that A_+ is the maximum magnitude of the synaptic weight update. For the most part, we experience good results, but with exceptions. We will provide an overview of the overall performance, and then discuss possible sources for sporadic inaccuracy.

5.2.1 General performance

Overall, the Metropolis Hastings algorithm performs really well when estimating only A_+. In figure 5.3 a good estimation of A_+ is presented. We see that the posterior is peaked around the true value, which is the case we observe most often. This sample is generated with noise level $\sigma = 0.0001$ which corresponds to a noise ratio of $\frac{0.0001}{0.005} = \frac{1}{50} = 2\%$ of the true value of the parameter. Hence also why the variance of the sample is so small.

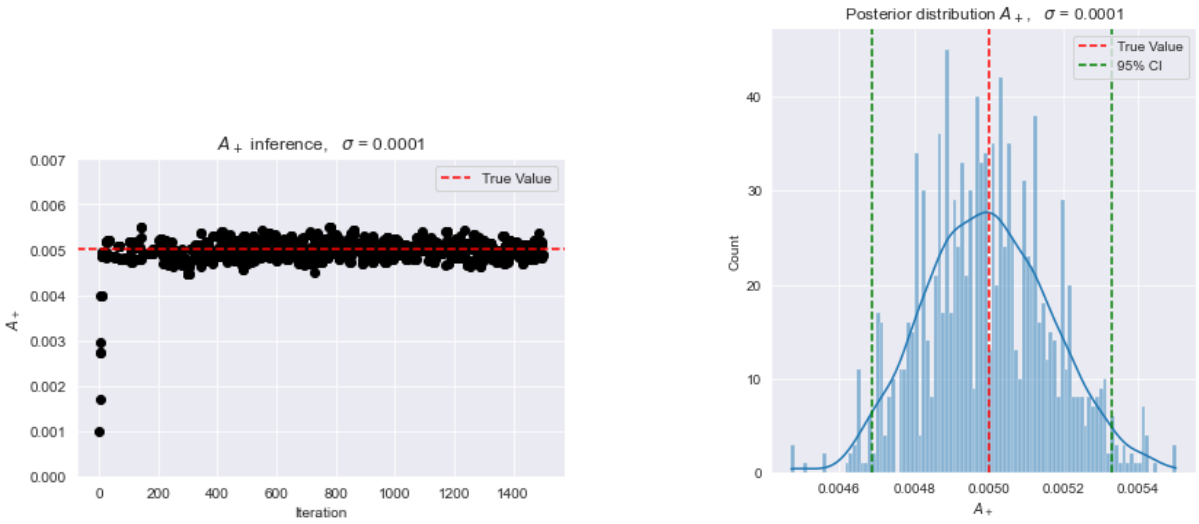

Figure 5.3: A good sample generated by the MH algorithm (left) and the corresponding posterior distribution of A_+ with noise level $\sigma = 0.0001$ (right). Green lines (right) show the 95% credible intervals.

If we increase the noise parameter, we would expect the variance to increase. However, sporadically we also observed samples not being centered exactly around the true mean, but where the posterior was shifted a little. An example of this is illustrated in figure 5.4. Here we had upped the noise to $\sigma = 0.005$, so that the noise to A_+ ratio was equal to one. As expected, we observe a much higher variance of the sample, but also, the posterior peaks between 0.006 and 0.007. These deviations from the true value also occurred

for small noise levels sporadically, hence why we investigated what factors could cause these undesired results.

We also observe from both figure 5.3 and 5.4, that the algorithm spends some iterations to converge to the high-density area. Based on these figures, we consider a reasonable burn-in time, as explained in 4.2.1, to be $B = 300$. This will be the choice for all of the results.

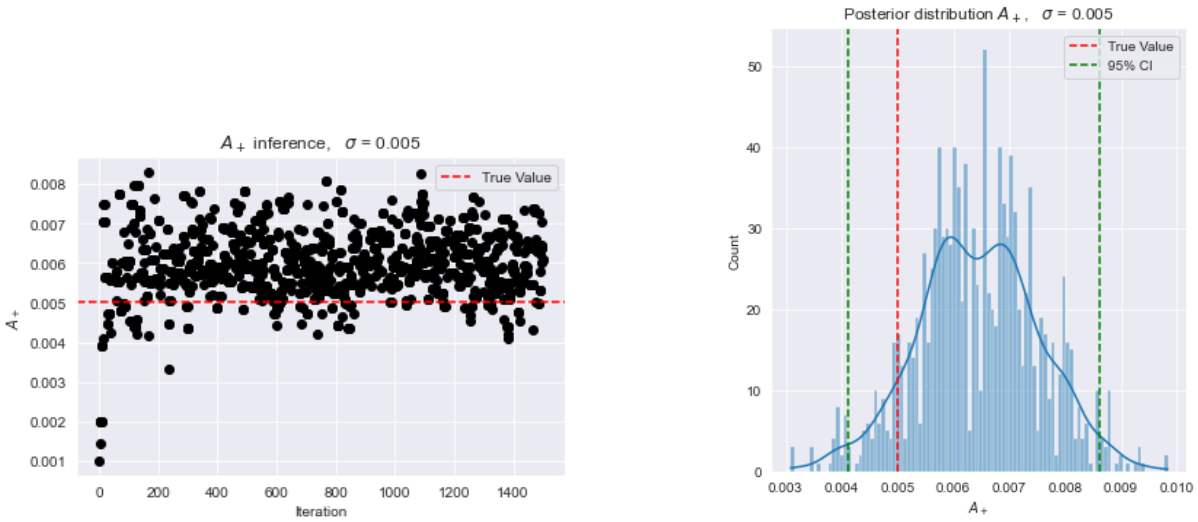

Figure 5.4: A not so good sample generated by the MH algorithm (left) and the corresponding posterior distribution of A_+ with noise level $\sigma = 0.005$ (right). Green lines (right) show the 95% credible intervals.

5.2.2 Sensitivity to w^0 estimation

An interesting consideration, is whether the fact that our estimation of w^0 from figure 5.2 is inconsistent, might cause problems for the inference of A_+. Recall from section 4.3, that the SMC tries to estimate the probability of the data given some proposed parameters, $P(s_2^{0:T}|\theta)$. The closer the particle trajectories $w^{0:T}$ are to the true (unobserved) trajectory, the higher the probability will be. Therefore, a good initialisation of the particles at w^0, might play an important role. Also recall that the posterior of some parameter is proportional to this likelihood of the data

$$P(\theta|s_2^{0:T}) \propto P(\theta)P(s_2^{0:T}|\theta).$$

For the moment we consider only $\theta = A_+$. Hence, for the algorithm to work efficiently, we want this value $P(s_2^{0:T}|\theta)$ to be as peaked as possible, and peaked around the true value. For the parameter A_+, the likelihood indeed seems pretty peaked, illustrated in figure 5.5. However, for which value of A_+ the

peak occurs, seems to depend heavily on the estimation of w^0. In figure 5.6, this relation is showcased. Here we have calculated $P(s_2^{0:T}|A_+)$ for a range of A_+ values, repeatedly for many data sets, and plotted which A_+ maximised the likelihood as a function of the estimation of w^0 for the given data set.

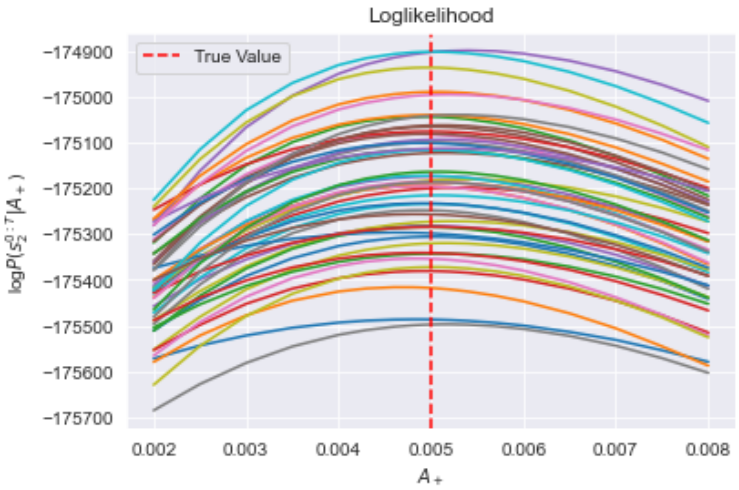

Figure 5.5: The likelihood $\log P(s_2^{0:T}|A_+)$ as a function of A_+ for 50 unique data sets. Data sets were generated with noise level $\sigma = 0.0001$.

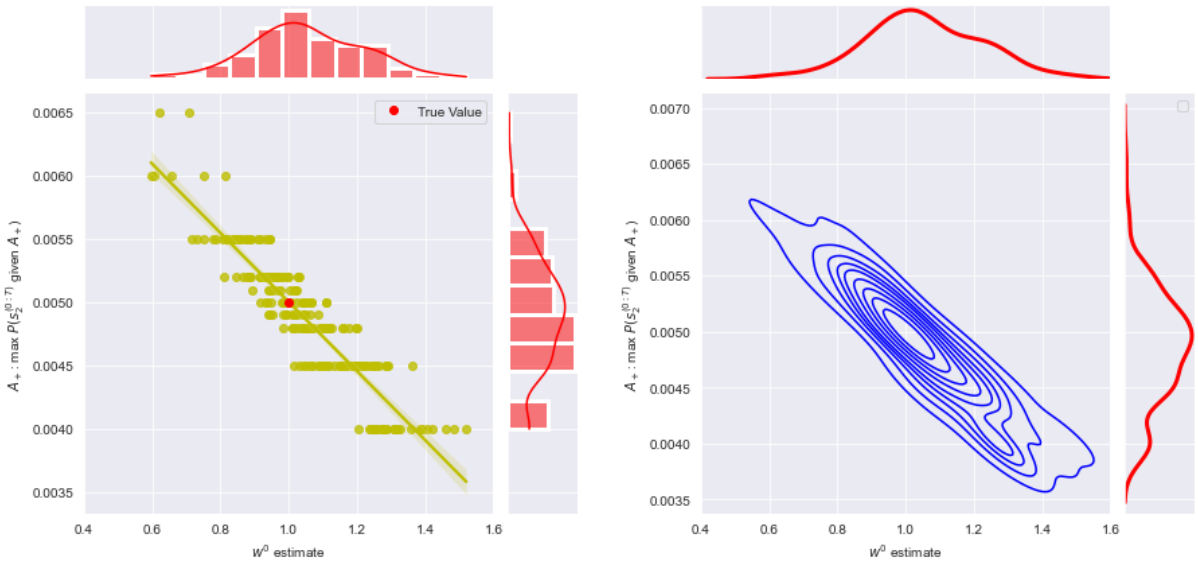

Figure 5.6: The value of A_+ maximising $P(s_2^{0:T}|A_+)$ as a function of the w^0 estimation, with the same data being illustrated through a regression plot (left) and a contour plot (right).

We see that the highest density areas are around the true values, which would in turn cause more accurate samples by the MH algorithm. However, the plot clearly shows a linear relationship, where a decrease of the w^0 estimation leads to higher values of A_+ being more likely.

This makes sense, because when τ is held constant, and w^0 is initiated too

small compared to the true value in the particle filter, the particle trajectories for higher values of A_+ will increase faster, and then come closer to the true trajectory, and generate higher likelihoods for the data. And oppositely a similar argumentation leads to a shift in the other direction. This is one possible reason for why the sample sporadically deviates from the true value.

For bigger time domains, we imagine that this initialisation would have less of an effect, and that the algorithm would be less sensitive to the w^0 estimation. We also tried different distributions for initialising the w^0 parameter in the particle filter procedure, without being able to solve the problem.

5.2.3 Sensitivity to noise

We also wished to do a more in-depth analysis of the impact of noise, that is, the Gaussian white noise term in (4.3). More specifically, how the size of the noise used in the SMC, affects the inference of A_+. For this purpose we first generated a simulated data set with low noise level, namely $\sigma = 0.0001$. In all of the following calculations, we made sure to use data sets where w_0 was estimated accurately, to eliminate the possible effect of this parameter, as discussed in section 5.2.2. We then applied the algorithm for different levels of noise (still with τ being held constant) on this exact same data set. The resulting means and standard deviations of the samples are presented in figure 5.7.

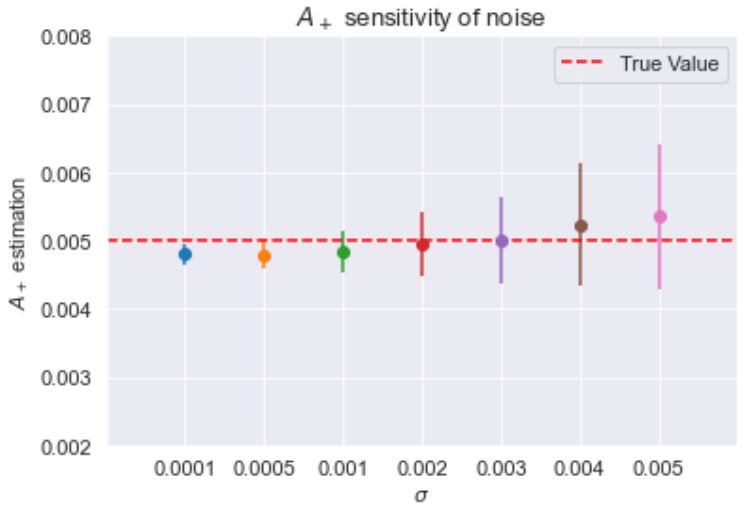

Figure 5.7: The means of A_+ from the MH algorithm samples on the same data set estimated with different noise levels used for learning, with corresponding errorbars of the standard deviations of the samples. The generative noise level here is $\sigma = 0.0001$.

This figure illustrates what we already expected, that the variance of the

samples increases with increasing noise. However, we also observe that even though the data sets are identical, the means of the samples seem to shift towards higher values of A_+ with increasing noise. To check if this is a systematic error which the algorithm makes, we generated 20 different data sets, again with $\sigma = 0.0001$, and ran the algorithm in similar fashion for different noise levels on all data sets. We furthermore calculated the mean and the MAP of the posteriors. Then we found the mean of these different measures over all 20 data sets for the 7 different noise levels, and the corresponding standard deviation of these measures. They all resulted in similar plots, and in figure 5.8 the results for the mean and the MAP are presented.

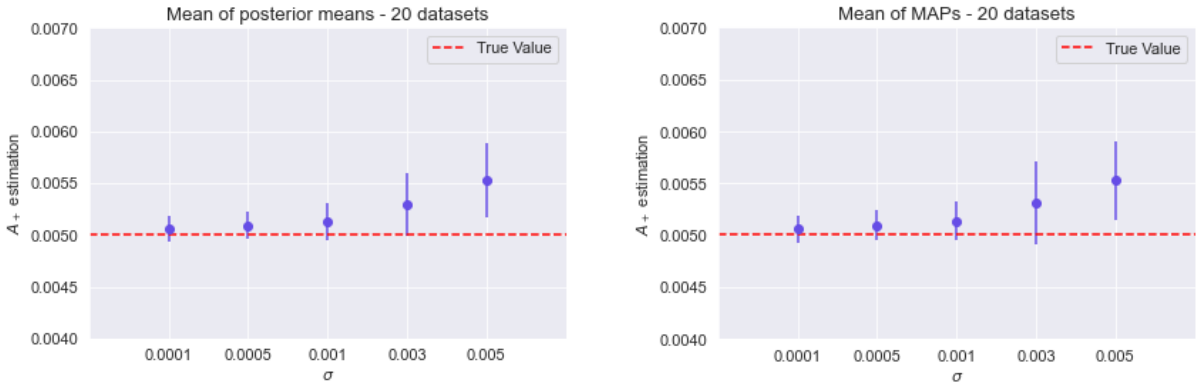

Figure 5.8: The means of the 20 estimates calculated with the mean of the posterior (left) and the MAP (right), respectively. The errorbars correspond to the standard deviation of the calculated estimators across the 20 data sets. Generative noise for the data sets was $\sigma = 0.0001$.

This is a strong indication, that the algorithm actually deviates towards higher values of A_+, when the noise is increased. The fact that the estimate of A_+ has a positive bias when the inference noise is larger than the generative one might be explained by the algorithm trying to overcompensate for small synaptic weight updates.

What we also found interesting and relevant, is to see how this deviation looks like when the generative noise is bigger than the inference one. When working with real data, the underlying unknown noise might be reasonably big, hence we would like to know how the inference is affected by this when we try to infer with a lower noise than the true one. To do this, the same procedure as for figure 5.8 was implemented, but now the data sets were generated by a noise level of $\sigma = 0.005$. The results are showcased in figure 5.9. In this case, we have unfortunately so far only been able to run 10 different data sets. We observe that the standard deviation is increased a lot compared to figure 5.8. This could be understood in that the nature of the data might vary more, as they are generated with 50x noise levels as previously. In addition,

the standard deviation might seem artificially big because we only ran for 10 unique data sets. But interestingly, there seems to be a similar bias present, where the values of A_+ are positively shifted from the true value. The variance is also bigger for smaller noise levels, so the bias appears to be compensating to some degree for the variety across the data sets, leading to less variance of the estimators. However, samples from more than 10 data sets would be desired to gain a better understanding.

Overall, the results in figure 5.8 and 5.9 seem to indicate that a choice of a small noise level for the inference might be the safest one, regardless of the generative one.

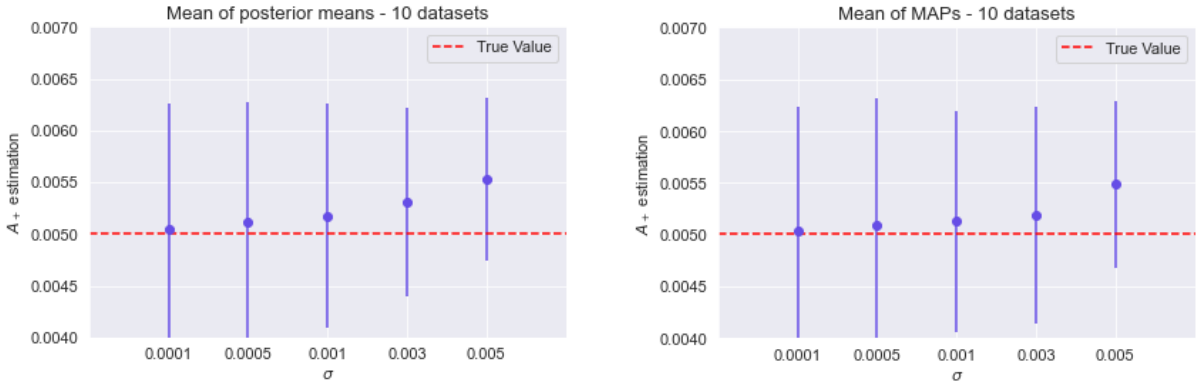

Figure 5.9: The means of the 10 estimates calculated with the mean of the posterior (left) and the MAP (right), respectively. All the data sets were generated with noise level $\sigma = 0.005$. The errorbars correspond to the standard deviation of the calculated estimators across the 10 data sets.

5.2.4 Sensitivity to the number of particles

The accuracy in the estimates of A_+ (and τ) may depend on the number of particles used in the particle filtering procedures, and we need to determine the appropriate number of particles to use. Increasing the number of particles will make the calculations more computationally demanding, and therefore slower, but it may help us assess how the number of particles plays a role in the accuracy of our estimates.

To investigate this, we first analysed the values of A_+ qualitatively (while holding $\tau = 0.02$ constant) for the same number of iterations but varying numbers of particles.

So far, we have done a brief analysis based on limited data samples. We generated 10 data sets, with $b_1 = b_2 = -2$, binsize = 5ms, noise level $\sigma =$ 0.0001, and then ran our Metropolis Hastings Particle procedure on each of

the same sets of 10 data sets for the different candidates for the particle numbers = {10, 50, 100, 500, 1000, 5000} for 1500 iterations. We averaged the A_+ values within each data set, and then looked at the ten means and medians for each of the candidate particle numbers. The results are plotted in figure 5.10.

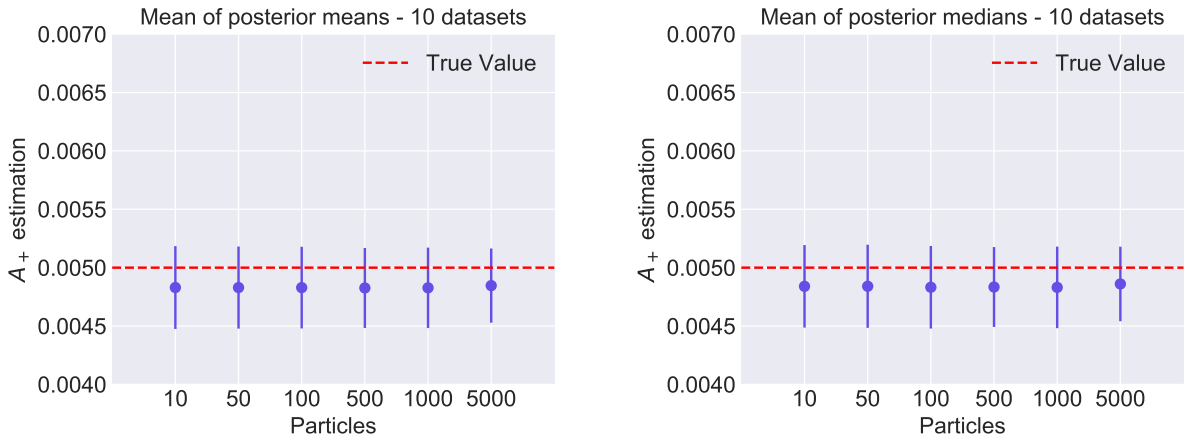

Figure 5.10: The mean of the posterior means (left) and medians (right), using ten data sets for each candidate particle value. Errorbars correspond to the standard deviation of the estimations accross data sets. The other parameters were kept constant accross the data sets, and the data sets were generated with noise level $\sigma = 0.0001$.

As seen in figure 5.10, the means of the posterior means are almost identical for the different particle numbers, up until 5000, for which there is a slight improvement of about $1 \cdot 10^{-5}$, which is not a large enough improvement that justifies the enormous increase in computation time. A similar observation can be made in the plot of the mean of the medians.

Furthermore, notice that the estimations are slightly biased, which can be connected to a slight bias in the inferred initial weight $\hat{w}_0$ in these data sets. Their average value was calculated to be $\hat{w}_0 = 1.066$, whilst the data was generated with $w_0 = 1$. Based on section 5.2.2 we would then also expect the estimates for A_+ to be negatively biased.

The standard deviations of the estimations from figure 5.10 are also pretty uniform. However, it seems to be a small improvement of with increasing parameters, in terms of lowering the variance of the estimations. From 1000 particles to 5000, the standard deviation decreases by $\sim 2.7 \cdot 10^{-4}$, being around 5% of the value of A_+. Again, we do not deem this change significant considering the increase of computations.

However, we need to emphasize that these are just indications based on a small sample size. One should therefore be careful to draw any immediate

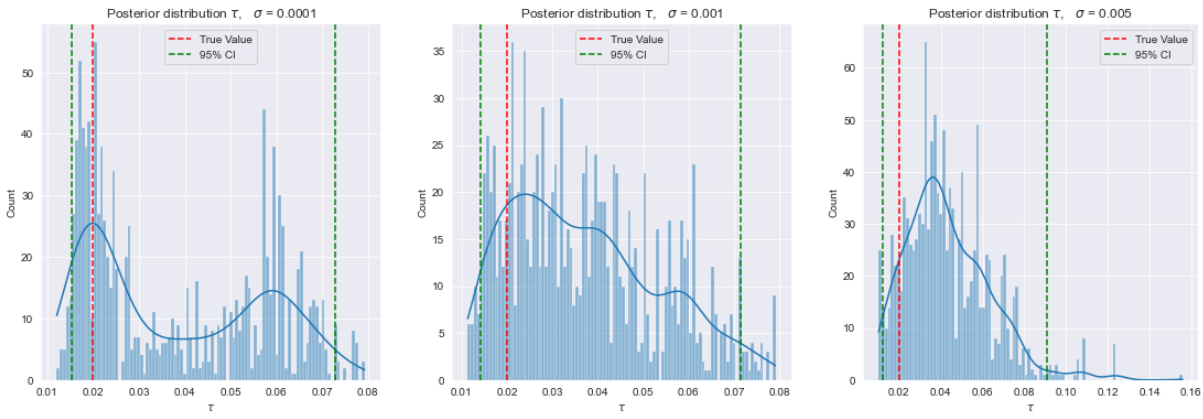

Figure 5.11: Posterior distributions from three different samples of τ for noise levels $\sigma = 0.0001$ (left), $\sigma = 0.001$ (middle), $\sigma = 0.005$ (right). Green lines show the 95% credible intervals.

conclusions from these initial explorations, before determining whether or not the number of particles affects the variance of the posterior. This will be investigated further by using more data sets, and preferably ensuring that $\hat{w}_0$ is not biased as in this case.

5.3 Inference of τ

Now, let us move to the other parameter of interest, τ, which we recall from (4.6) as being the decay time of the learning rule value. Again, we first aim to investigate how well the algorithm manages to characterise the posterior distribution of τ when A_+ is being held constant, and what problems it faces.

5.3.1 General performance

Overall, the inference of τ appears to be more difficult and the results are more varying than for A_+. We experience a lot more variance in the posteriors, also for really small noise levels, as well as the peak not necessarily matching the generative value. Additionally, we sometimes observe a multimodal posterior distribution for τ. A selection of samples illustrating these different challenges is shown in figure 5.11. We also observe that the level of noise does not seem to play such a big role. The variances of the samples are relatively big for both small and high noise regimes. Hence we have tried to identify possible sources for these inconsistent results.

5.3.2 Sensitivity to w^0 estimation

Again, we would be interested in knowing how the SMC estimates the likelihood of the data for different parameters of τ, $P(s_2^{0:T}|\tau)$, and the more peaked this distribution is, the more accurate we expect our Bayesian estimates of the parameter τ to be.

Also, recall from section 5.2.2 that for different values of A_+, the likelihood was sensitive to the estimator $\hat{w}^0$. Therefore we find it interesting to investigate this for τ as well.

To investigate how the shape of this likelihood looks like, we generated many data sets, again making sure they yielded similar estimates for w^0, and calculated $P(s_2^{0:T}|\tau)$ for a range of τ values for every data set. For the case when w^0 was estimated accurately, we observed that the likelihood actually is not heavily peaked around the true value, which explains the high variance of our samples and that the peak could easily get shifted to another value. To illustrate how the shape looks like, we calculated the mean curve over these data sets, which is showcased in figure 5.12.

As for cases when w^0 was estimated inaccurately, we mainly observed an effect for the case when w^0 was estimated at a too high value. For this case, we made a similar plot based on 100 data sets, which is also presented in figure 5.12.

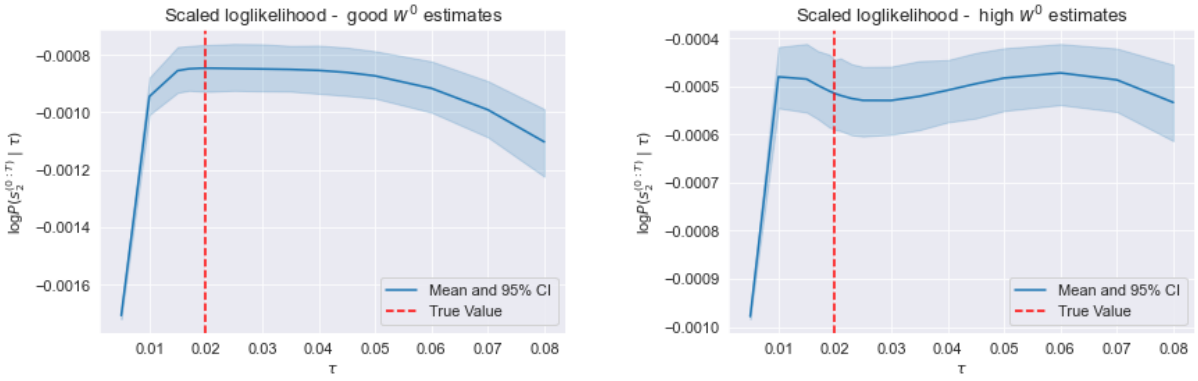

Figure 5.12: The mean curve over 100 data sets of the scaled $\log P(s_2^{0:T}|\tau)$ as a function of τ, with corresponding confidence interval, for cases when the w^0 estimate is good (left) and too high (right).

In the plot on the right of figure 5.12 we observe that the likelihood has two peaks, which can explain why we sometimes observe a bimodal posterior distribution, as the first sample in figure 5.11. Depending on how bad the w^0 estimate is, the location of these peaks may be shifted, but the general shape is consistent. Different distributions for the initialisation of w^0 in the SMC

did not seem to fix the problem in this case either.

In order to deal with these issues, both the non-peaked log-likelihood in the case of good w^0 estimates, and the double-peaked log-likelihood in the case of high w^0 estimates, we experimented with lowering the binsize of the time domain, hoping that the characteristics of τ would be better captured as a result of this.

5.3.3 Reducing the binsize

We lowered the binsize from 5ms to 2ms and 1ms, and the results were promising regarding the estimation of τ. In order to simulate data sets of these binsizes, and to maintain the stationary firing rate of 20s^{-1}, the probability of spiking in each bin must also be smaller. Hence we adjusted the background inputs b_1 and b_2 in both cases such that the baseline firing rate still was as desired.

When studying the likelihood $P(s_2^{0:T}|\tau)$ in the same manner as previously, it seems that the likelihood is more peaked on average and the problem with multiple peaks for high w^0 estimates seems to disappear. This is showcased for the case of 1ms binsize in figure 5.13. The first plot, shows that for high w^0 estimates, the shape is now more peaked and even around the true value, compared with the observations for 5ms. Hence, the likelihood no longer seems to be too sensitive to w^0. In the second plot we show the expected shape over arbitrary data sets with arbitrary w^0 estimates, again based on 100 data sets. We see that the likelihood is in general more peaked than for the 5ms case, and also peaked more or less around the true value.

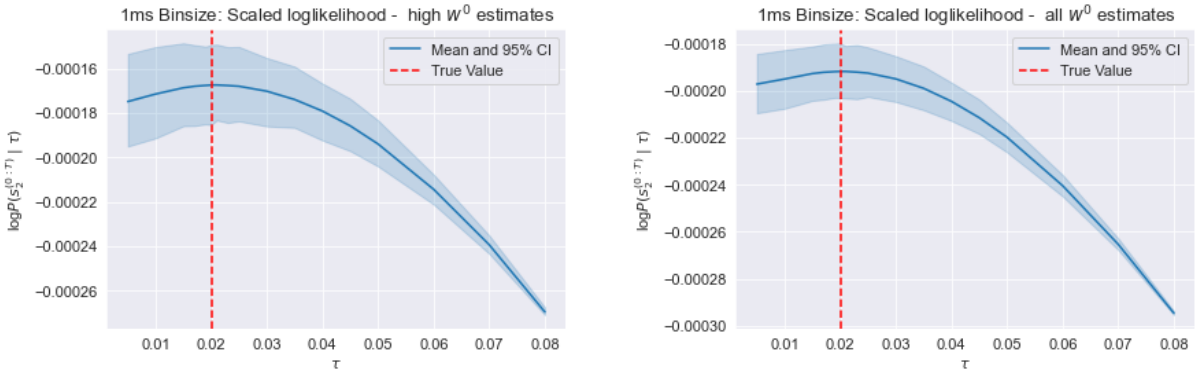

Figure 5.13: The mean curve over 100 data sets of $\log P(s_2^{0:T}|\tau)$ as a function of τ, with binsize 1ms and confidence intervals are added shaded. The plots show high estimates of w^0 (left), and arbitrary w^0 values (right).

When estimating the posterior distribution of τ, it seems to work bet-

ter, especially for small noise levels. The samples seem to be more peaked around the true value and the variance seems to be decreased significantly. In figure 5.14, posterior distributions for noise two different noise levels are showcased, estimated with 1ms and 2ms respectively.

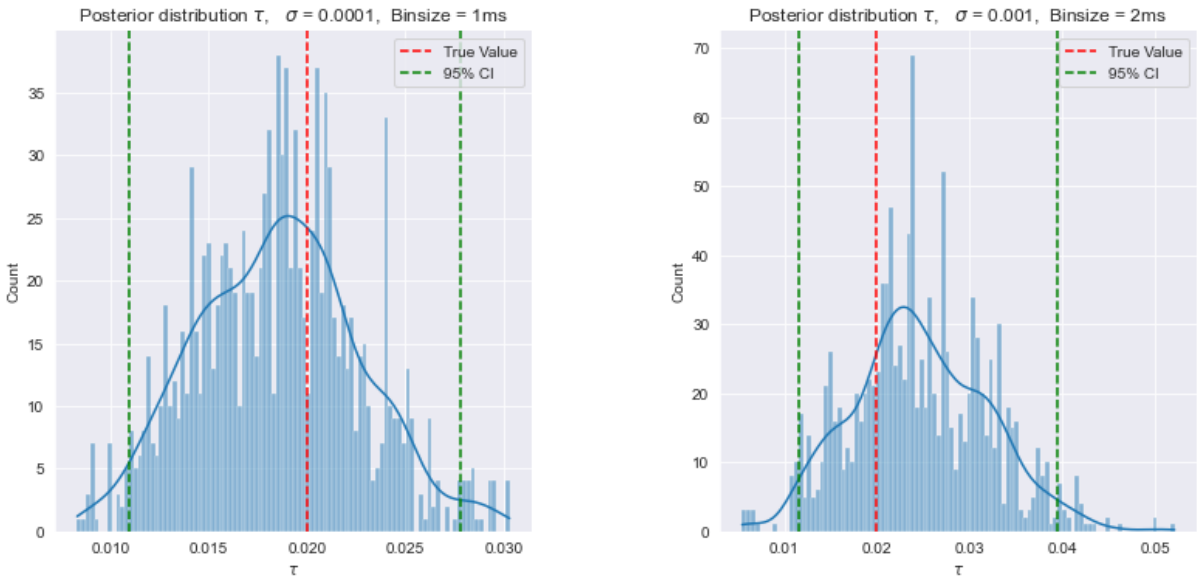

Figure 5.14: Estimated posterior distributions of τ for binsizes 1ms (left) and 2ms (right), with noise levels $\sigma = 0.0001$ and $\sigma = 0.001$ respectively. Green lines show the 95% credible intervals.

5.3.4 Sensitivity to noise

We want to investigate more comprehensively how the noise used for inference affects the Bayesian estimates of τ, and how the effect of noise possibly changes when we reduce the binsize. Hence we have done similar analysis as for A_+, where we first generated a data set with small noise, being $\sigma = 0.0001$, and tried to infer τ from the same data set, but for different noise levels. This was done for the three investigated binsizes, namely 5ms, 2ms and 1ms. The results are showcased in figure 5.15. For the case of 5ms, we observe that the variance of the samples are pretty large regardless of the noise level, which is reasonable given the non-peaked likelihood we experienced earlier for this binsize. However, when we lower the binsize, we observed more peaked likelihoods, which is here again reflected in the cases of 2ms and 1ms. At least for small noise levels, we see that the samples have small variance, and are centered around the true value. Moreover, when the noise increases, we seem to experience similar deviations as for A_+, for all binsizes.

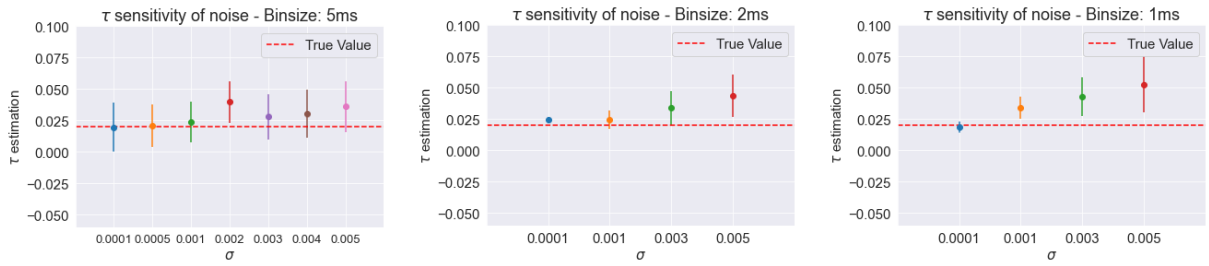

Figure 5.15: Estimated parameter values $\hat{\tau}$ for different noise levels based on MAP with errorbars representing the standard deviation of the samples, done with 5ms binsize (left), 2ms binsize (middle) and 1ms binsize (right). The generative noise for simulating datesets were $\sigma = 0.0001$.

5.4 Simultaneous Inference

Now that we know that the algorithm seems to be working decently for both parameters separately, and what factors might cause problems, we are ready to test the method in the context where it ideally should be applied, and where it hopefully should work, namely when both parameters are unknown. For this purpose we are testing two different versions of the algorithm, as explained in section 4.2. For all of the samples in this section, a binsize of 5ms is used.

5.4.1 Standard MH method

The first method is the standard Metropolis Hastings algorithm, Algorithm 3, where at each step, new values for both A_+ and τ are proposed simultaneously. A consequence of this could be that the algorithm makes some weird steps in either of the parameter spaces, or refuses to do relevant steps for one of the parameters, because the proposal for the other parameter might be significantly better or significantly worse. However, the method seems to perform relatively well. Figure 5.16 illustrates two samples from different data sets of the posterior distribution of A_+ for noise levels $\sigma = 0.0001$ and $\sigma = 0.002$.

We observe that the posteriors still seem to be centered around the true value, which is exactly what we are hoping for. However, as expected, we observe more variance compared to when the parameters were estimated separately, as well as several outliers, since A_+ and τ are being inferred simultaneously. In figure 5.17, we have plotted the corresponding posterior distributions for τ.

As we discussed previously, τ seems to be more difficult to estimate with

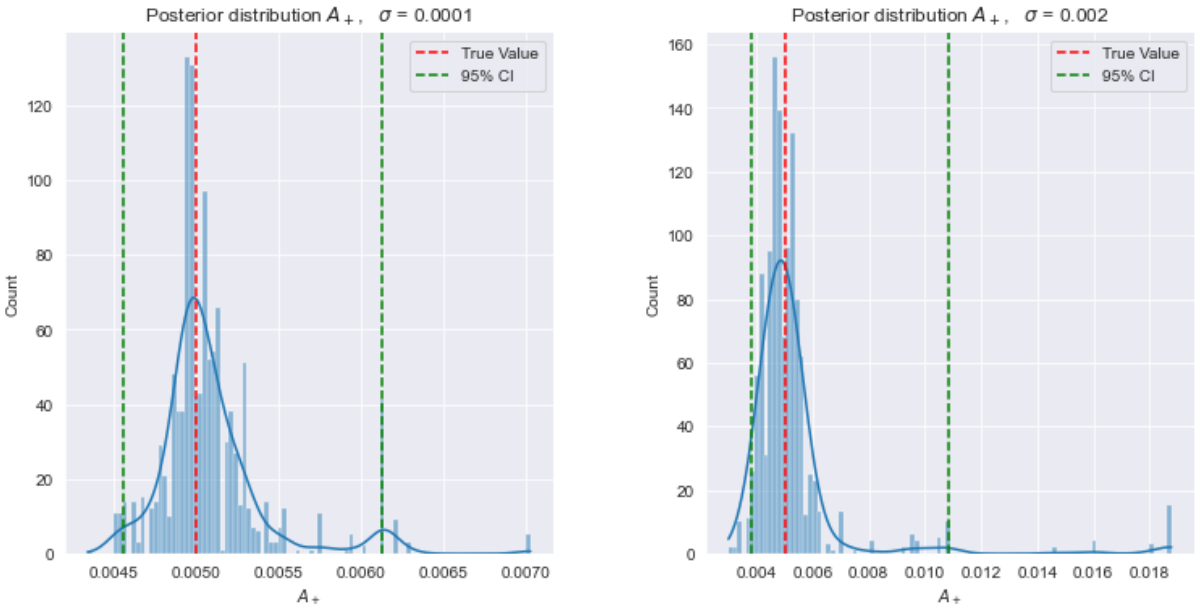

Figure 5.16: Estimated posterior distributions for A_+ when applying the standard MH algorithm, for noise values $\sigma = 0.0001$ (left) and $\sigma = 0.002$ (right) respectively. Green lines show the 95% credible intervals.

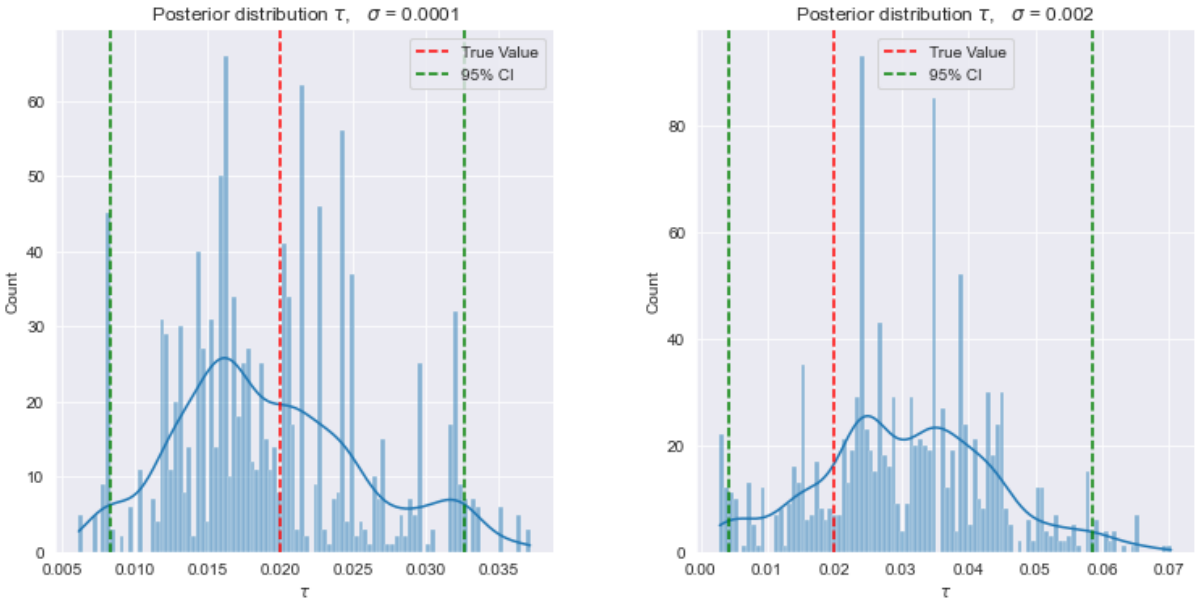

Figure 5.17: Estimated posterior distributions for τ when applying the standard MH algorithm, for noise values $\sigma = 0.0001$ (left) and $\sigma = 0.002$ (right) respectively. Green lines show the 95% credible intervals.

the same accuracy, which one could see from the figures. The variance of the estimations are still pretty big, and the posterior is not centered around the true value in these cases. However, the results are not horrible, and they are pretty much what we could expect with this binsize, based on the analysis in the previous section.

5.4.2 Alternating MH method

The second method was explained in section 4.2.5. The only real difference between the MH adopted in this section and the one studied in the previous one, is that this algorithm only proposes an update for either A_+ or τ at each iteration, while keeping the other parameter fixed, and alternates between which parameter is proposed. We implemented this variation over the standard Metropolis-Hastings hoping to avoid what we observed for the standard method, namely outliers and unreasonable steps for either parameter. Again, we simulated different data sets and ran the algorithm with different noise levels to get an impression of the behaviour.

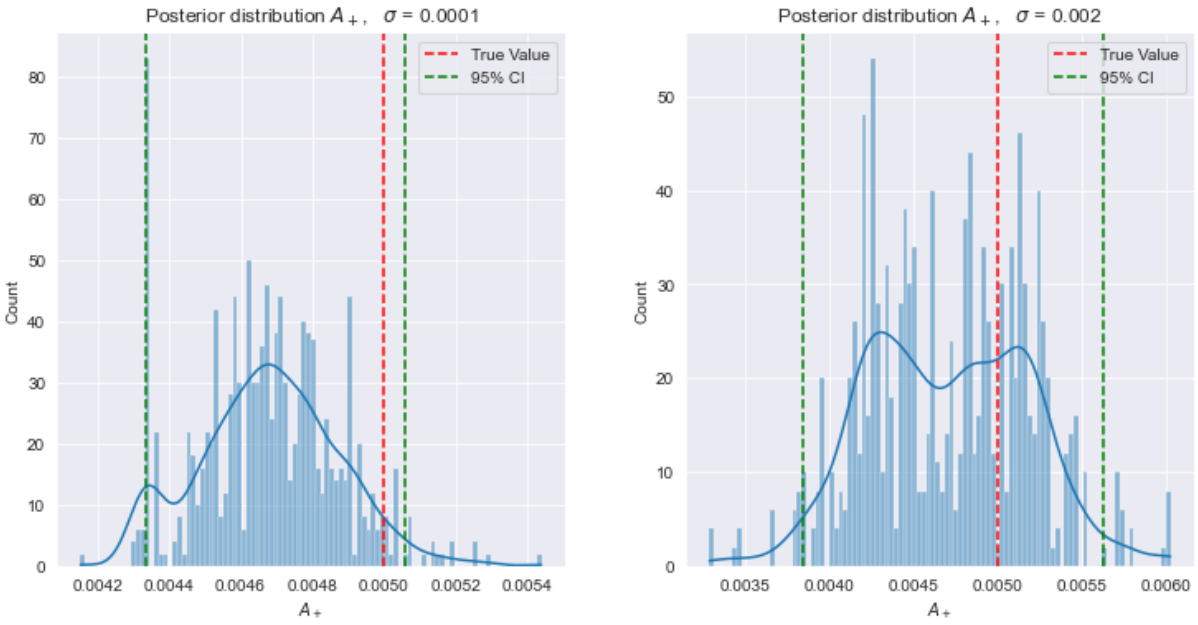

Figure 5.18: Estimated posterior distributions for A_+ when applying the alternating MH algorithm, for noise values $\sigma = 0.0001$ (left) and $\sigma = 0.002$ (right). Green lines show the 95% credible intervals.

In figure 5.18, two samples from different data sets are presented, with the inference being done with the same noise levels as for the standard method, $\sigma = 0.0001$ and $\sigma = 0.002$. Be aware that the data sets used for estimation here are not identical to the ones used for the standard MH in the previous section. For the smallest noise regime, the distribution seems to be shifted slightly towards lower values, which could indicate a too high estimation for w^0 for this data. However, the distribution is pretty tight with small variance, note the range of the x-axis in comparison to the left plot in figure 5.16. Also, compared to the results of the standard method in 5.16, the outliers we previously observed are now non-existent. When increasing the noise, again the

variance naturally increases, but still we do not really observe outliers and the estimation is relatively good considering the size of the noise.

The corresponding posteriors for τ are presented in figure 5.19, from which it seems that the alternating method did not improve the estimation of τ, indeed rather conversely, based on these samples. Especially for the posterior for $\sigma = 0.002$ (right) from figure 5.19, we observe a shifted mode compared to the true value, as well as outliers. This could also partly be due to other previously discussed factors.

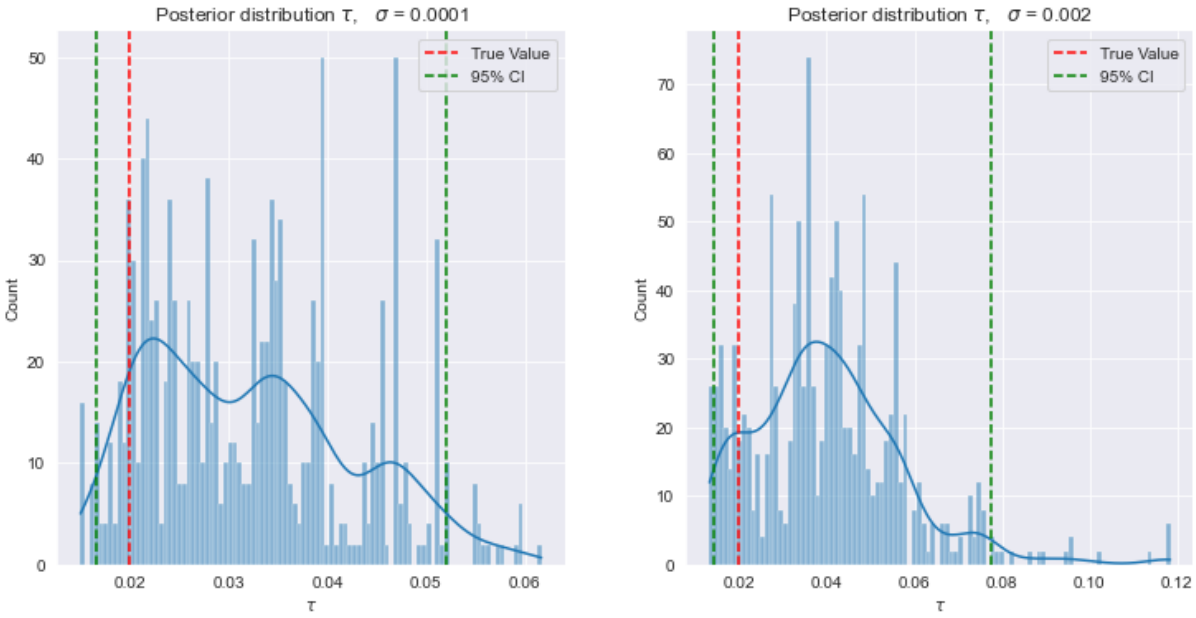

Figure 5.19: Estimated posterior distributions for τ when applying the alternating MH algorithm, for noise values $\sigma = 0.0001$ (left) and $\sigma = 0.002$ (right). Green lines show the 95% credible intervals.

Based on these samples, it is difficult to conclude which method is better. The posteriors of A_+ could support the alternating method, whereas the results for τ might suggest otherwise. A more in-depth comparison between the two is desired for the future.

Furthermore, occasionally we observe weird behaviour for the alternating method. It appears to be some correlation between the learning rule parameters A_+ and τ, in that when one of them is deviating away from the true value, the other parameter might also get shifted away from the true value. Hence, we sporadically observe samples from the alternating MH method were both parameters are making some weird movements in the parameter space. An example of this is presented in figure 5.20.

From the plots we observe that when A_+ moves towards higher values (left plot), τ simultaneously is moving towards smaller values (middle plot). When

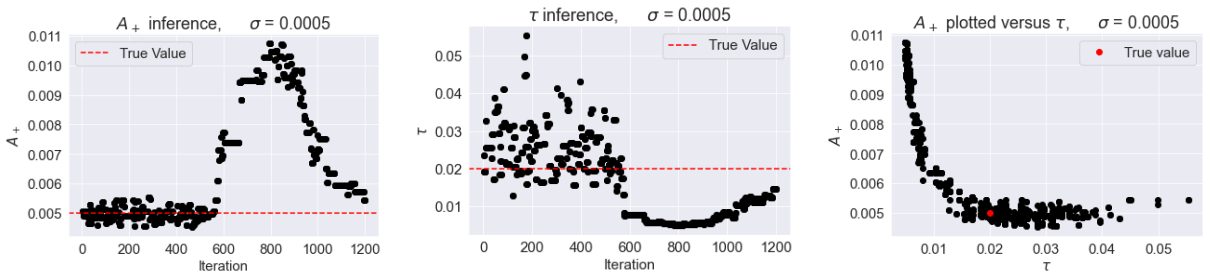

Figure 5.20: A weird sample of A_+ (left) and τ (middle) drawn by the alternating MH method. Indication of correlation between the two parameters, when plotted against each other (right).

plotting τ versus A_+ (right plot), we observe a correlation curve somewhat having a similar shape as the learning rule we are considering in our framework, recall for instance figure 4.3. We find this interesting, and something worth looking deeper into in the future. So far however, we have not been able to conduct any further analysis on this.

5.5 Inferring the noise

Ideally, we would also like to say something about the level of noise in the brain, based on real neural data. So far, we have fixed the noise at different values, and seen how this affects our estimations. In sections 5.2.3 and 5.3.4 the robustness of the algorithm was studied. Based on this, low noise levels, preferably $\sigma < 0.001$, are preferred. Hence, being able to quantitatively say something about the order of magnitude of the noise could be adequate. In this section we therefore study the ability of our algorithm to estimate the parameter σ, being the standard deviation of our Gaussian white noise.

For this purpose, we first considered three different levels of noise, namely $\sigma = 0.0005$, $\sigma = 0.001$ and $\sigma = 0.003$. For each of them, we simulated 100 data sets, and for each of the data sets we calculated the likelihood of the data, $P(s_2^{0:T})$, estimated by the SMC, given a range of σ values. As discussed previously, ideally we would hope that this likelihood is peaked somewhat around the underlying true noise level, for the Metropolis Hastings to be more accurate. The results are showcased in figure 5.21.

From the figure we observe that the SMC is not really able to distinguish between the underlying true noise levels, in that the peak seems to be in the region $\sigma \in [0.002, 0.004]$, regardless of what noise level generated the data. Also, we observe that the likelihood seems to be more peaked, the smaller the underlying noise level is, but again not around the true value. This suggests

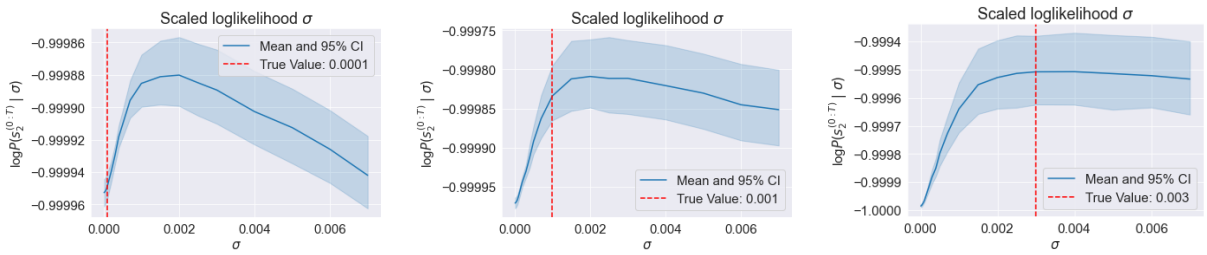

Figure 5.21: Scaled log-likelihoods, $\log P(s_2^{0:T}|\sigma)$ based on 100 simulated data sets, with corresponding 95% confidence interval. The data sets are generated by noise levels 0.0001 (left), 0.001 (middle) and 0.003 (right).

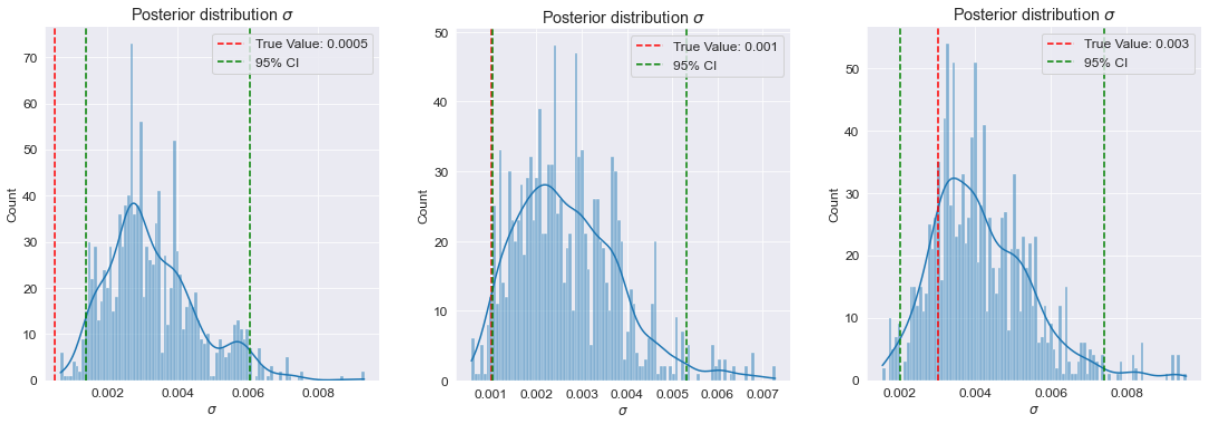

Figure 5.22: Posterior distributions from three different samples of σ, with the true noise levels generating the data being 0.0005 (left), 0.001 (middle) and 0.003 (right). Green lines show the 95% credible intervals.

that it will be difficult to obtain a meaningful posterior distribution from the Metropolis Hastings.

To confirm this, we ran the Metropolis Hastings algorithm, targeting the parameter σ, whilst both A_+ and τ were fixed at their true values. For the choice of prior distributions, we experimented with different options. Our aim was that the mean of the prior would be somewhere around the true value of A_+, and that the variance would not be unreasonably big. We tried two different gamma priors with the discussed properties, one of them with mode at 0, while the other one with mode around the mean. The latter of the two seemed to yield the most reasonable results. The chosen prior was therefore a gamma distribution with parameters

$$\alpha = 5, \qquad \beta = 800 \tag{5.2}$$

The resulting posterior distributions from the MH sampling, for the same underlying noise levels that we previously introduced, are presented in figure 5.22.

From the figure we observe what we would expect based on the behaviour of the likelihood, namely that all posteriors also are peaked in the region of $\sigma = [0.002, 0.004]$, regardless of what value of σ actually generated the data. Hence, for the upcoming section we will omit the inference of noise, but this is something we strongly want to develop a solution for in the future.

5.6 Application to real data

In this section we will finally explore how the method performs on real neural data. Recall that the data is from the authors of the mentioned article [3]. According to them, the mouse was at different times exposed to some stimulation, causing some of the neurons to fire more rapidly. This could either be a current injection or some optogenetic (light-based) stimulation. The data is then gathered as the spiking times, or spike trains, of the recorded neurons. For now, we only aim to estimate the learning rule parameters and to study whether the resulting learning rule seems to be interesting and useful. The noise will therefore be set for the inference, at $\sigma = 0.0001$, until we have developed a good way to estimate this, see section 5.5.

5.6.1 Identifying a unidirectional connection

We have studied the data from one session of the whole data collection process. This session includes recordings from 34 different neurons over a span of approximately two hours recording time. Due to the sporadic stimulations, the data is rather heteroscedastic. In some time intervals, the neurons might barely fire, while at other times they might spike a lot, also known as burstiness, which is emphasised in [3]. Firstly, we are trying to identify a monosynaptic connection, that is, a neuron pair where only one of the neurons receives signals directly from the other. Such a connection would fit into the framework we have developed for inferring the learning rule. When searching for such a connection, we are looking for a time interval where the neurons seem to be stimulated, which might cause some sort of learning.

To do this, we visualised the spike trains of the neurons over different time intervals, and looked for possible stimulation events by observing increased firing rates. In figure 5.23 the spike trains of six selected and indexed neurons from the recordings are illustrated, for the range of approximately three minutes.

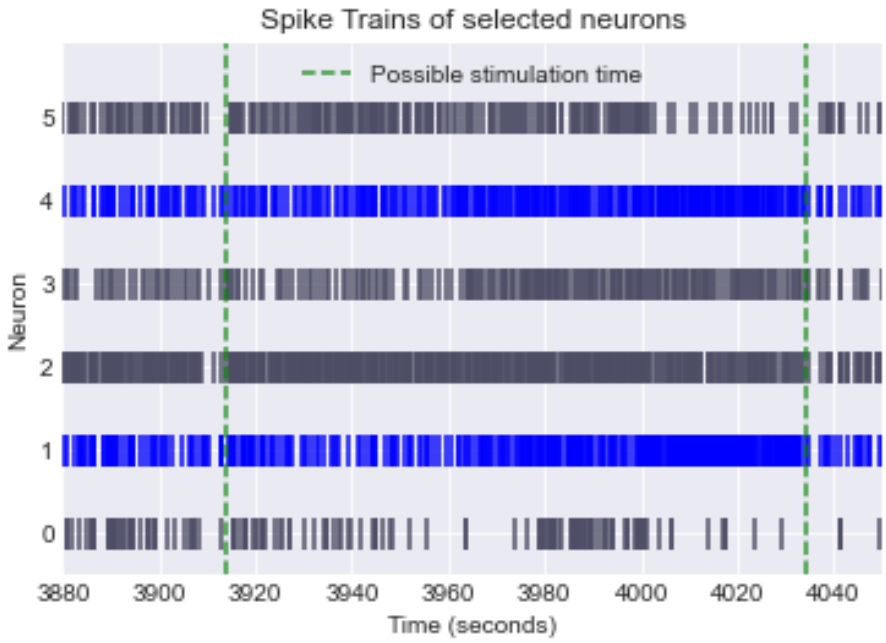

Figure 5.23: The spike trains of six selected neurons within a subset of the experimental time domain. The blue ones are the desired neurons for further analysis.

From this figure, we observed one time interval indicating some sort of stimulation of the neurons. This interval is illustrated with two green dotted lines in figure 5.23. In particular two neurons, plotted with blue spike trains, seem to be stimulated and to experience a similar evolution in terms of activity. Before the indicated start of stimulation, they seem to have somewhat stationary firing rates, but then they appear to increase, until the end of the selected interval, when the spike rates seem to decrease again. Also other neurons, for instance neuron 3, seem to be affected by the stimulation to some extent.

We defined our time domain of interest to be

$$T \in [3900, 4035], \qquad [T] = s.$$

To look for connections within this interval, we first discretised our time domain into bins of size $\delta = 1$ms. All bins were then assigned binary values 1 or 0, if the neuron did or did not spike within that 1ms time interval, respectively. This was done separately for all considered neurons, to express their spiking activity as time series. These time series were then used to plot the cross-correlations for all pairs of neurons, as explained in section 3.4. The results for two neuron pairs are presented in figure 5.24.

To measure the significance of the correlations, we do a hypothesis test as explained in the theory in section 3.4. Due to us wanting the test to be reason-

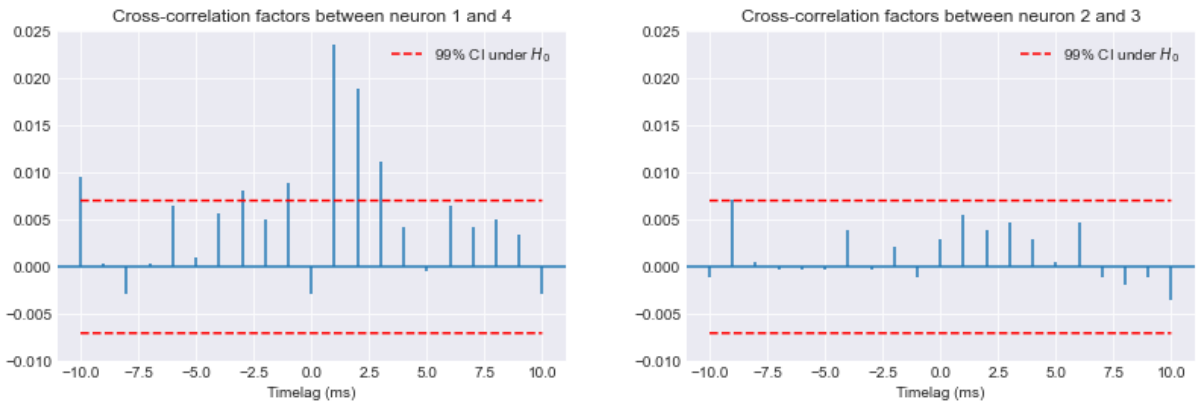

Figure 5.24: Cross correlograms for two different pairs of neurons. The first one being a possible good connection (left), while the second one does not indicate a significant connection (right).

ably trustworthy, we chose significance level $\alpha = 99\%$ for our hypotheses test. Thus we have calculated 99% confidence intervals for the correlation under the null hypotheses of no correlation. In the first plot, the correlation between the two neurons with blue spike trains is plotted for different time lags. We observe clear peaks at time lags 1ms and 2ms, also well outside the confidence interval, strongly indicating that these correlations are significant. For comparison, we show in the second plot how the correlations looked qualitatively for several other pairs of neurons, where there are no clear peaks, and none of the correlations seem to be significant according to our hypothesis test.

Hence, we have indication to believe that neuron 1 and 4 might be connected, and thus we choose this pair of neurons for the upcoming application. From our definition of the cross-correlation function (3.29), neuron 1 in figure 5.24 is the one exposed to lag. The correlation is significant for positive lags of neuron 1, indicating that this would be the neuron being affected, hence the postsynaptic neuron. We will stick with the notation of neuron 4 and neuron 1 for our pre- and postsynaptic neurons.

5.6.2 Applying the method

Now that we have identified a candidate monosynaptic connection (peak of the cross-correlation at 1-2ms lag) between neurons 1 and 4 from figure 5.24, we can start discussing the inference of the learning rule parameters with the framework presented in chapter 4. First, recall that the estimation of the initial weight parameter, w^0, might be influential for the inference, first and foremost regarding the parameter A_+, which in turn could also affect the estimation of τ, see section 5.2. Hence, we would like this estimation to be as good as possible. Therefore, we would like to know whether our assumption of no learning and a stationary w at the start of the domain holds. In figure 5.25 we have plotted the correlation in similar fashion as earlier, for the two chosen neurons, but only for the first 10 seconds of the chosen time domain.

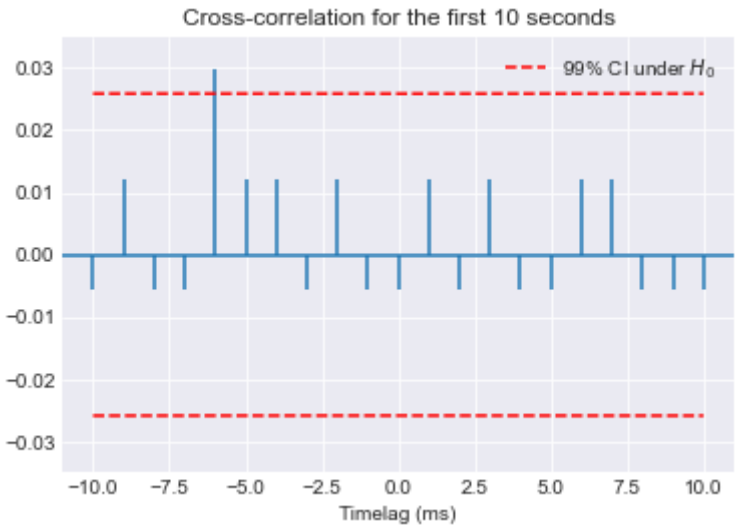

Figure 5.25: The cross-correlation between the chosen neurons for inference, based on the first 10 seconds of history of the time domain of interest.

From the figure we see that in the beginning, in general it does not seem to be any significant connection between them. There is one correlation factor, for -6ms time lag, which is here deemed significant, based on our hypotheses test. However, based on seemingly little correlation, we believe that the underlying assumption for the estimation of w^0 is not too heavily violated in this case, which hopefully means that our upcoming estimations will be accurate.

For the inference, we ran the two discussed variants of the Metropolis Hastings algorithm with binsize in time of $\delta = 1$ms. Let b_1 be the baseline

input of the presynaptic neuron, being neuron 4. Similarly, b_2 is the baseline input for the postsynaptic neuron 1. First, the GLM parameters w^0, b_1 and b_2 were estimated, which yielded the results presented in table 5.1. From this we can obtain an estimated baseline spike rate of the neurons within this time interval, being $\approx 8.8s^{-1}$ and $\approx 10.4s^{-1}$ for neuron 4 and 1, respectively. In addition, we observe from $\hat{w^0}$, that the neurons seem to be weakly connected before the potential stimulation.

	Estimate
$\hat{w^0}$	1.137
$\hat{b_1}$	-4.728
$\hat{b_2}$	-4.551

Table 5.1: Estimates for the GLM parameters for the chosen pair of neurons from the data set [3].

Furthermore, we present the sampled posterior distributions for the learning rule parameters A_+ and τ. Together with the posteriors of the parameters we have reported the three Bayesian estimators for A_+ and τ introduced in section 3.3.2, namely the mean, MAP and median of the posteriors. In figure 5.26 and 5.27 the results from the two different methods are presented.

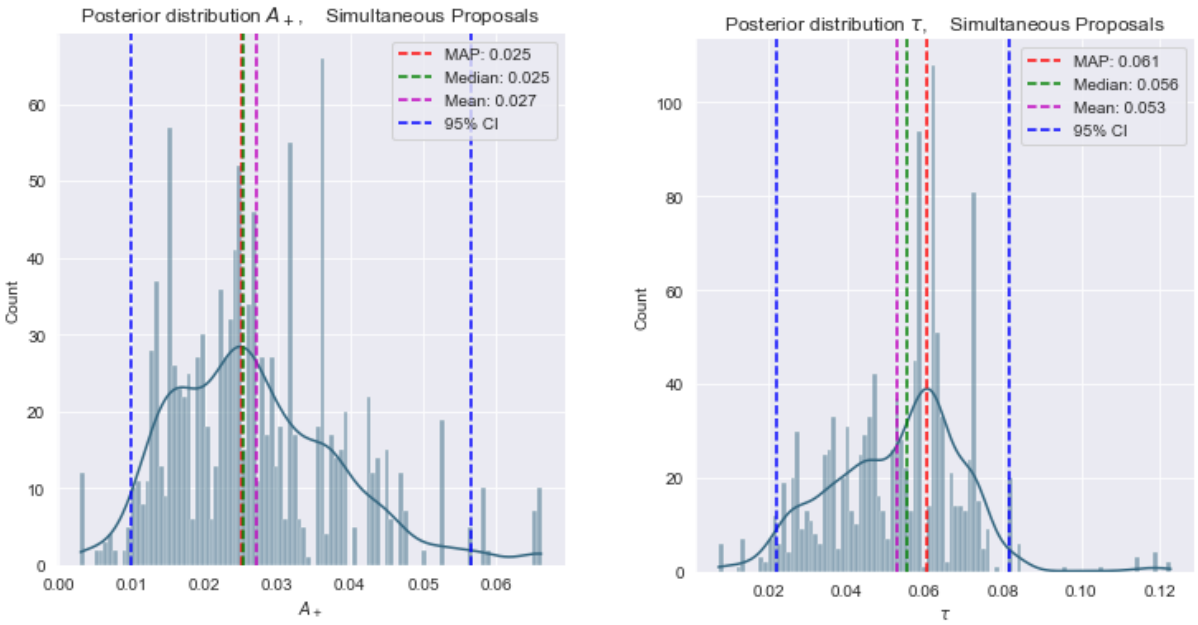

Figure 5.26: Posterior distributions for A_+ (left) and τ (right) from the standard MH method with simultaneous proposals, with the MAP, mean and median illustrated in the figures. Blue lines show the 95% credible intervals.

We observe that the estimates for both methods and across the different measures are pretty similar. Using the pre-determined relations given by (4.5)

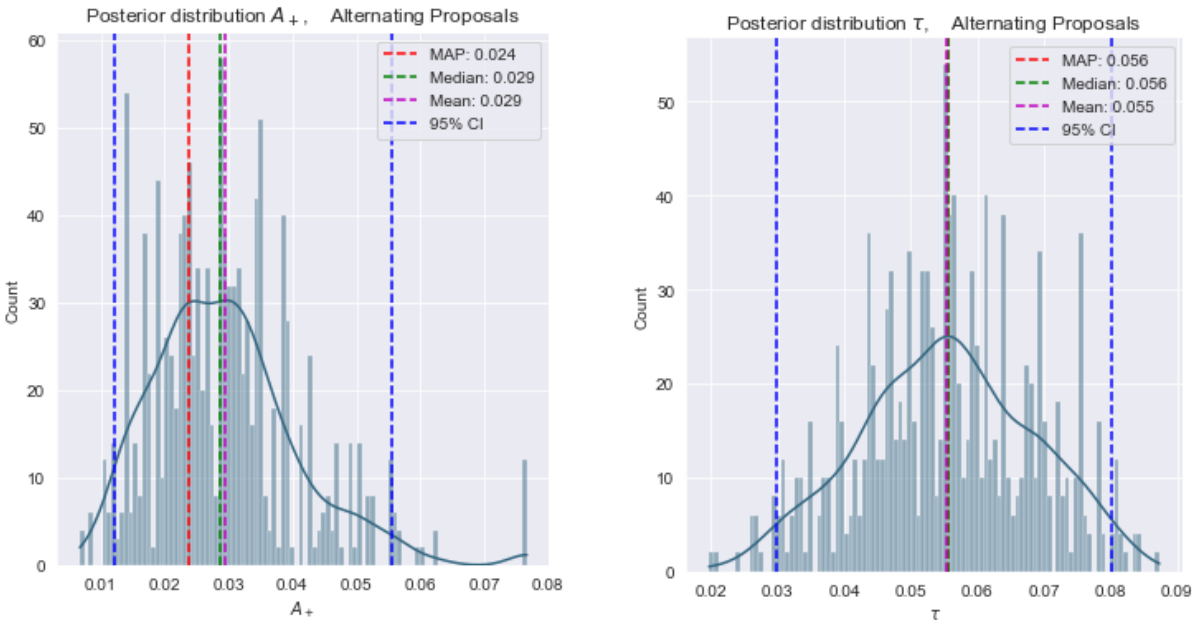

Figure 5.27: Posterior distributions for A_+ (left) and τ (right) from the alternating MH method, with the MAP, mean and median illustrated in the figures. Blue lines show the 95% credible intervals.

we can find the estimates for all learning rule parameters $\hat{\theta} = \{\hat{A}_+, \hat{A}_-, \hat{\tau}_+, \hat{\tau}_-\}$, summarised in table 5.2.

$\hat{\theta}$	**Standard MH**	**Alternating MH**
$\hat{A}_+$	*MAP*: 0.025	*MAP*: 0.0238
	Mean: 0.02704	*Mean*: 0.02941
	Median: 0.02522	*Median*: 0.0287
$\hat{A}_-$	*MAP*: 0.02625	*MAP*: 0.02499
	Mean: 0.0284	*Mean*: 0.03088
	Median: 0.02648	*Median*: 0.03013
$\hat{\tau}_+$	*MAP*: 0.06061	*MAP*: 0.0556
	Mean: 0.05284	*Mean*: 0.05534
	Median: 0.05551	*Median*: 0.05553
$\hat{\tau}_-$	*MAP*: 0.06061	*MAP*: 0.0556
	Mean: 0.05284	*Mean*: 0.05534
	Median: 0.05551	*Median*: 0.05553

Table 5.2: Estimated values for the learning rule parameters.

We consider it really promising and encouraging, that the posterior distributions seem to be so peaked, especially the ones sampled by the alternating MH method.

From now we will proceed by using the MAP estimates of $\hat{\theta}$ based on the Alternating MH method. Based on these we calculated the estimated learning rule for the considered neural connection. Using this we can estimate,

based on the neural activity and this learning rule, how the connection actually evolves in this time interval, based on our framework, recall (4.3). This is interesting to see whether our results actually yield anything reasonable and whether some learning seems to happen within this time interval. These results are presented in figure 5.28.

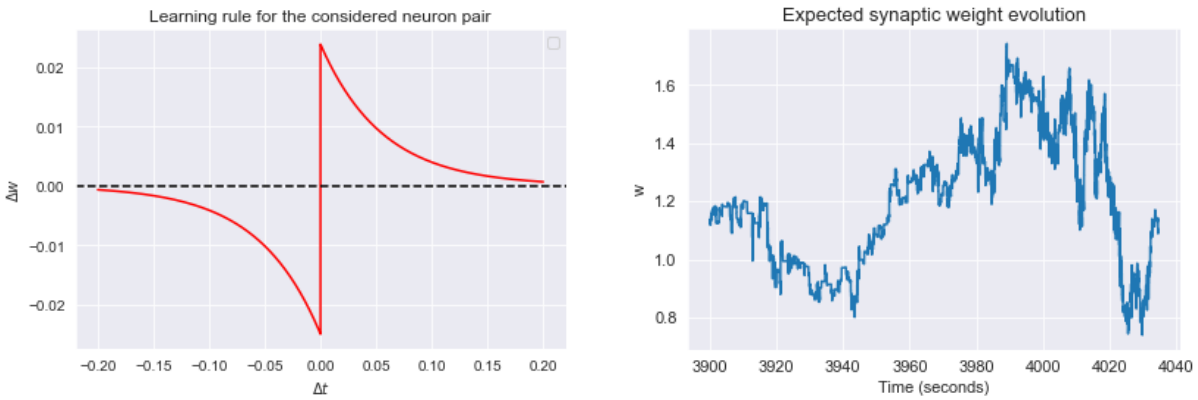

Figure 5.28: The resulting learning rule for the neuron connection based on our parameter estimations (left). The expected weight trajectory, given the spike data and the estimated learning rule (right). That is, a realisation without noise, $\sigma = 0$.

We observe that some sort of learning seems to occur, or at least some evolution of the synaptic strength, in the middle of the domain where the spike rates heavily increased. At the end however, the connectivity strength is again descending.

Chapter 6

Final remarks

As our work is still very much at the preliminary stage, we are not ready to draw any conclusions at this point. However, we would like to make some final comments and discuss the work so far, as well as reflect on future plans and prospects.

First, we proposed a possible modification to the STDP learning rule, which we argued provided several advantages compared to the original one, especially in terms of computational efficiency, recall section 4.1.2. Our results indicate that this could be implemented unproblematically, which we consider a success.

Furthermore, our aim was to be able to estimate these parameters associated with the learning rule, given neural data, adapting a particle Metropolis-Hastings algorithm. For this purpose, we also proposed an alternative to the standard MH method, which alternates the proposals of A_+ and τ, see section 4.2.1. We investigated the performance and robustness of these methods, to identify what challenges should be handled in order to optimise the method. Our findings in chapter 5 indicate that especially three factors seem to worsen the performance of the algorithm, being

- **The estimation of w^0**. An inaccurate estimate seems to bias the estimates of A_+, whereas the posterior of τ appears to become multimodal.
- **The noise level, σ, used for inference**. A too high noise level for the inference adds a lot of variance to the samples and the estimates become biased.
- **Binsize**. An overlarge binsize seems to cause problems for the estimation of τ.

However, we would overall suggest that the results are encouraging. When lowering the binsize, using a small noise level and having a decent estimate $\hat{w}^0$, the posteriors for both A_+ and τ are promisingly peaked, and for the most part unbiased, when estimated separately. Moreover, estimating the parameters simultaneously also seems to yield positive results, in that the nature of the learning rule is often well recovered from the inference, as discussed in section 5.4. The standard MH method appears to be somewhat vulnerable to outliers, whereas the alternating MH method occasionally captures some correlation between the parameters. However, both methods are indicating decent average performance, and we are so far not able to conclude with a superior method.

Both methods were applied to real neural data collected from mouse and rat brains, see section 5.6. Based on only these results, it is difficult to make overall evaluations about whether or not the estimated learning rule and the resulting synaptic weight evolution are reasonable for the mouse brain. However, the fact that the method behaved so nicely, in terms of the posterior distributions being so peaked and the estimates so consistent, we consider really exciting.

For the imminent future, we consider this project to have many interesting prospects. In our upcoming work, we consider the following things interesting to assess

- Conduct a more complete analysis of the effect of the number of particles, both in terms of estimation of A_+ and τ.
- Develop a method for the inference of noise
- Further experimentation with real data sets
- Possibly apply different learning rules and perform model selection based on these
- Develop an optimal stimulation paradigm

We see this as appropriate next steps in developing this framework with the vision of eventually contributing to the understanding of learning in the normal and diseased brain, how networks learn in general, as well as for the advancement of neural network models in machine learning.

References

[1] S. W. Linderman, C. H. Stock, and R. P. Adams. A framework for studying synaptic plasticity with neural spike train data, 2014.

[2] S. Song, K. D. Miller, and L. F. Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity:919–926, 2000. URL: https://doi.org/10.1038/78829.

[3] D. F. English, S. McKenzie, T. Evans, K. Kim, E. Yoon, and G. Buzsaki. Pyramidal cell-interneuron circuit architecture and dynamics in hippocampal networks, 2017.

[4] D. Desilver. Despite global concerns about democracy, more than half of countries are democratic, 2019.

[5] S. Herculano-Houzel. The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in Human Neuroscience*, 3:31, 2009. ISSN: 1662-5161. DOI: 10.3389/neuro.09.031.2009. URL: https://www.frontiersin.org/article/10.3389/neuro.09.031.2009.

[6] A. Woodruff. What is a neuron? Aug. 2019. URL: https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron.

[7] Neurons, nerve tissues, the nervous system. URL: http://biomedicalengineering.yolasite.com/neurons.php.

[8] The action potential. URL: https://opentextbc.ca/anatomyandphysiology/chapter/12-4-the-action-potential/.

[9] W. Gerstner, W. Kistler, R. Naud, and L. Paninski. *Neuronal dynamics*. Cambridge university press, 2014.

[10] Y. Kaddar. Spike trains. Apr. 2017. URL: https://younesse.net/Neuromodeling/ProblemSet3/.

[11] B. Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *J. Neurophysiol*, 76:3460, 1996.

[12] N. Yusoff and A. Grüning. Biologically inspired temporal sequence learning. *Procedia Engineering*, 41:319–325, 2012. ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2012.07.179. URL: http://www.sciencedirect.com/science/article/pii/S1877705812025659. International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012).

[13] A. Vigneron and J. Martinet. A critical survey of stdp in spiking neural networks for pattern recognition. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2020.

[14] E. I. Moser, K. A. Krobert, M.-B. Moser, and R. G. Morris. Impaired spatial learning after saturation of long-term potentiation. *Science*, 281(5385):2038–2042, 1998.

[15] J. R. Whitlock, A. J. Heynen, M. G. Shuler, and M. F. Bear. Learning induces long-term potentiation in the hippocampus. *science*, 313(5790):1093–1097, 2006.

[16] B. A. Yankner. Mechanisms of neuronal degeneration in alzheimer's disease. *Neuron*, 16(5):921–932, 1996.

[17] J. W. Vaupel. Biodemography of human ageing. *Nature*, 464(7288):536–542, 2010.

[18] J. A. Hardy and G. A. Higgins. Alzheimer's disease: the amyloid cascade hypothesis. *Science*, 256(5054):184–186, 1992.

[19] G. W. Van Hoesen, B. T. Hyman, and A. R. Damasio. Entorhinal cortex pathology in alzheimer's disease. *Hippocampus*, 1(1):1–8, 1991.

[20] A. Kobro-Flatmoen and M. P. Witter. Neuronal chemo-architecture of the entorhinal cortex: a comparative review. *European Journal of Neuroscience*, 50(10):3627–3662, 2019.

[21] S. Ross. *Introduction to probability models*. Elsevier, pp.191-195, 269-275, 2010.

[22] H. Scheepers. Markov chain analysis and simulation using python. Nov. 2019. URL: https://towardsdatascience.com/markov-chain-analysis-and-simulation-using-python-4507cee0b06e.

[23] L. Ross, T. Kneib, S. Land, and B. Marx. *Regression*. Springer, pp.269-285, 2013.

[24] S. V. Vaseghi. *Advanced Digital Signal Processing and Noise Reduction, Second Edition*. John Wiley and Sons, pp.105-109, 2008.

[25] R. Bassett and J. Deride. Maximum a posteriori estimators as a limit of bayes estimators. *Mathematical Programming*, 174(1-2):129–144, Jan. 2018. ISSN: 1436-4646. DOI: 10.1007/s10107-018-1241-0. URL: http://dx.doi.org/10.1007/s10107-018-1241-0.

[26] R. Shumway and D. Stoffer. *Time Series Analysis and Its Applications With R Examples*, volume 9. Jan. 2011, page 10. ISBN: 978-1-4419-7864-6. DOI: 10.1007/978-1-4419-7865-3.

[27] L. D. Haugh. Checking the independence of two covariance-stationary time series: a univariate residual cross-correlation approach. *J. Amer. Statist. Assoc.*, 7(354):378–385, 1976. ISSN: 0162-1459. URL: http://links.jstor.org/sici?sici=0162-1459(197606)71:354%3C378:CTIOTC%3E2.0.CO;2-Q&origin=MSN.

[28] K. E. Himdi and R. Roy. Tests for non-correlation of two multivariate time series: a nonparametric approach, 2003.

[29] S. Chib and E. Greenberg. Understanding the metropolis-hastings algorithm, 1995.

[30] H. Haario, E. Saksman, and J. Tamminen. Adaptive proposal distribution for random walk metropolis algorithm, 1999.

[31] G. Givens and J. Hoeting. Simulation and monte carlo integration. In 2013.

[32] D. Rubín and D. Rubin. Using the sir algorithm to simulate posterior distributions. In 1988.

[33] A. Doucet, N. de Freitas, and N. Gordon. *An Introduction to Sequential Monte Carlo Methods*. Springer, 2001.

[34] C. A. Naesseth, F. Lindsten, and T. B. Schön. *Elements of Sequential Monte Carlo*. 2019. URL: https://arxiv.org/abs/1903.04797.

[35] O. Cappé, R. Douc, A. Guillin, J.-M. Marin, and C. P. Robert. *Adaptive Importance Sampling in General Mixture Classes*. 2008. URL: https://arxiv.org/abs/0710.4242.

[36] C. Shannon. *A Mathematical Theory of Communication*. Bell System Technical Journal. 27 (3): 379-423., 1948.

[37] L. Martino, V. Elvira, and F. Louzada. *Effective Sample Size for Importance Sampling based on discrepancy measures*. Signal Processing 131, 386–401, 2017.

[38] M. Själander, M. Jahre, G. Tufte, and N. Reissmann. EPIC: an energy-efficient, high-performance GPGPU computing research infrastructure, 2019. arXiv: 1912.05848 [cs.DC].

Appendix A

Source code in Python

```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from scipy.stats import gamma
from numba import njit
@njit

def learning_rule(s1,s2,Ap,Am,taup,taum,t,i,binsize):
    '''
    s1,s2 : binary values for the different time bins for neuron 1 and 2 respectively, 1:spike, 0:no spike
    i : current iteration/timebin for the numerical approximation
    '''
    l = i - np.int(np.ceil(10*taup / binsize))
    return s2[i-1]*np.sum(s1[max([l,0]):i]*Ap*np.exp((t[max([l,0]):i]-max(t))/taup)) - s1[i-1]*np.sum(s2[max([l,0]):i]*Am*np.exp((t[max([l,0]):i]-max(t))/taum))

def logit(x):
    return np.log(x/(1-x))

def inverse_logit(x):
    return np.exp(x)/(1+np.exp(x))


class SimulatedData():
    '''
    Ap, Am, tau : learning rule parameters
    b1,b2 : background noise constants for neuron 1 and neuron 2, determnining their baseline firing rate
    w0 : start value for synapse strength between neuron 1 and 2.
    '''
    sec = 120
    binsize = 1/200.0
    def __init__(self,Ap=0.005, tau=0.02, std=0.001,b1=-2.0, b2=-2.0, w0=1.0):
        self.Ap = Ap
        self.tau = tau
        self.std = std
```

```
        self.Am = 1.05*self.Ap
        self.b1 = b1
        self.b2 = b2
        self.w0 = w0

    def set_Ap(self,Ap):
        self.Ap = Ap
    def set_tau(self,tau):
        self.tau = tau
    def set_std(self,std):
        self.std = std
    def set_b1(self,b1):
        self.b1 = b1
    def set_b2(self,b2):
        self.b2 = b2
    def set_w0(self,w0):
        self.w0 = w0

    def get_Ap(self):
        return self.Ap
    def get_tau(self):
        return self.tau
    def get_std(self):
        return self.std
    def get_b1(self):
        return self.b1
    def get_b2(self):
        return self.b2
    def get_w0(self):
        return self.w0

    def create_data(self):
        iterations = np.int(self.sec/self.binsize)
        t,W,s1,s2 = np.zeros(iterations),np.zeros(iterations),np.zeros(
iterations),np.zeros(iterations)
        W[0] = self.w0
        s1[0] = np.random.binomial(1,inverse_logit(self.b1))
        for i in tqdm(range(1,iterations)):
            lr = learning_rule(s1,s2,self.Ap,self.Am,self.tau,self.tau,t
,i,self.binsize)
            W[i] = W[i-1] + lr + np.random.normal(0,self.std)
            s2[i] = np.random.binomial(1,inverse_logit(W[i]*s1[i-1]+self
.b2))
            s1[i] = np.random.binomial(1,inverse_logit(self.b1))
            t[i] = self.binsize*i
        self.s1 = s1
        self.s2 = s2
        self.t = t
        self.W = W

    def get_data(self):
        return self.s1,self.s2,self.t,self.W

    def plot_weight_trajectory(self):
        plt.figure()
        plt.title('Weight trajectory')
        plt.plot(self.t,self.W)
```

```
        plt.xlabel('Time')
        plt.ylabel('Weight')
        plt.show()

class ParameterInference():
    '''
    Class for estimating b1,b2,w0,Ap,Am,tau from SimulatedData, given data s1,s2.
    '''
    sec = 120
    binsize = 1/200.0
    def __init__(self,s1,s2,P = 100, Usim = 100, Ualt = 200,it = 1500, std=0.0001, N = 2\
                 , shapes_prior = np.array([4,5]), rates_prior = np.array([50,100])):
        self.s1 = s1
        self.s2 = s2
        self.std = std
        self.P = P
        self.Usim = Usim
        self.Ualt = Ualt
        self.it = it
        self.N = N
        self.shapes_prior = shapes_prior
        self.rates_prior = rates_prior

    def b1_estimation(self):
        self.b1est = logit(np.sum(self.s1)/len(self.s1))
        return self.b1est

    def normalize(self,vp):
        return vp/np.sum(vp)

    def perplexity_func(self,vp_normalized):
        h = -np.sum(vp_normalized*np.log(vp_normalized))
        return np.exp(h)/self.P

    def resampling(self,vp_normalized,wp):
        wp_new = np.copy(wp)
        indexes = np.linspace(0,self.P-1,self.P)
        resampling_indexes = np.random.choice(indexes,self.P,p=vp_normalized)
        for i in range(self.P):
            wp_new[i] = np.copy(wp[resampling_indexes.astype(int)[i]])
        return wp_new

    def likelihood_step(self,s1prev,s2next,wcurr):
        return inverse_logit(wcurr*s1prev + self.b2est)**(s2next) * (1-inverse_logit(wcurr*s1prev + self.b2est))**(1-s2next)

    def parameter_priors(self):
        return np.array([(np.random.gamma(self.shapes_prior[i],1/self.rates_prior[i])) for i in range(self.N)])

    def proposal_step(self,shapes,theta):
        return np.array([(np.random.gamma(shapes[i],theta[i]/shapes[i]))
    for i in range(self.N)])
```

```
139
140     def adjust_variance(self,theta,shapes):
141         means = theta[-self.Usim:].mean(0)
142         var_new = np.array([0,0])
143         u_temp = self.Usim
144         while (any(i == 0 for i in var_new)):
145             var_new = theta[-u_temp:].var(0)*(2.4**2)
146             u_temp += 50
147             if u_temp > self.it:
148                 return shapes, np.array([(np.random.gamma(shapes[i],
    theta[-1][i]/shapes[i])) for i in range(self.N)])
149         new_shapes = np.array([((means[i]**2) / var_new[i]) for i in
    range(self.N)])
150         proposal = np.array([(np.random.gamma(new_shapes[i],theta[-1][i
    ]/new_shapes[i])) for i in range(self.N)])
151         return new_shapes,proposal
152
153     def ratio(self,prob_old,prob_next,shapes,theta_next,theta_prior):
154         spike_prob_ratio = prob_next / prob_old
155         prior_ratio, proposal_ratio = 1,1
156         for i in range(self.N):
157             prior_ratio *= gamma.pdf(theta_next[i],a=self.shapes_prior[i
    ],scale=1/self.rates_prior[i])/\
158             gamma.pdf(theta_prior[i],a=self.shapes_prior[i],scale=1/self
    .rates_prior[i])
159             proposal_ratio *= gamma.pdf(theta_prior[i],a=shapes[i],scale
    =theta_next[i]/shapes[i])/\
160             gamma.pdf(theta_next[i],a=shapes[i],scale=theta_prior[i]/
    shapes[i])
161         return spike_prob_ratio * prior_ratio * proposal_ratio
162
163
164     def scaled2_spike_prob(self,old,new):
165         return np.exp(old - min(old,new)),np.exp(new - min(old,new))
166
167     def b2_w0_estimation(self):
168         '''
169         Fisher scoring algorithm
170         Two in parallell, since w0 is estimated with a subset of the
    data
171         '''
172         s1short,s2short = self.s1[:2000],self.s2[:2000]
173         beta,beta2 = np.array([0,0]),np.array([0,0])
174         x,x2 = np.array([np.ones(len(self.s1)-1),self.s1[:-1]]),np.array
    ([np.ones(len(s1short)-1),s1short[:-1]])
175         i = 0
176         score,score2 = np.array([np.inf,np.inf]), np.array([np.inf,np.
    inf])
177         while(i < 1000 and any(abs(i) > 1e-10 for i in score) and any(
    abs(j) > 1e-10 for j in score2)):
178             eta,eta2 = np.matmul(beta,x),np.matmul(beta2,x2) #linear
    predictor
179             mu,mu2 = inverse_logit(eta),inverse_logit(eta2)
180             score,score2 = np.matmul(x,self.s2[1:] - mu),np.matmul(x2,
    s2short[1:] - mu2)
181             hessian_u,hessian_u2 = mu * (1-mu), mu2 *(1-mu2)
182             hessian,hessian2 = np.matmul(x*hessian_u,np.transpose(x)),np
```

```
    .matmul(x2*hessian_u2,np.transpose(x2))
183             delta,delta2 = np.matmul(np.linalg.inv(hessian),score),np.
    matmul(np.linalg.inv(hessian2),score2)
184             beta,beta2 = beta + delta, beta2 + delta2
185             i += 1
186         self.b2est = beta[0]
187         self.w0est = beta2[1]
188         return self.b2est,self.w0est
189
190
191     def particle_filter(self,theta):
192         '''
193         Particle filtering, (doesnt quite work yet, smth with weights vp
    )
194         Possible to speed it up?
195         How to initiate w0 and vp?
196         '''
197         timesteps = np.int(self.sec/self.binsize)
198         t = np.zeros(timesteps)
199         wp = np.full((self.P,timesteps),np.float(self.w0est))
200         vp = np.ones(self.P)
201         log_posterior = 0
202         for i in range(1,timesteps):
203             v_normalized = self.normalize(vp)
204             perplexity = self.perplexity_func(v_normalized)
205             if perplexity < 0.66:
206                 wp = self.resampling(v_normalized,wp)
207                 vp = np.full(self.P,1/self.P)
208                 v_normalized = self.normalize(vp)
209             lr = learning_rule(self.s1,self.s2,theta[0],theta[0]*1.05,
    theta[1],theta[1],t,i,self.binsize)
210             ls = self.likelihood_step(self.s1[i-1],self.s2[i],wp[:,i-1])
211             vp = ls*v_normalized
212             wp[:,i] = wp[:,i-1] + lr + np.random.normal(0,self.std,size
    = self.P)
213             t[i] = i*self.binsize
214             log_posterior += np.log(np.sum(vp)/self.P)
215         return wp,t,log_posterior
216
217
218     def standardMH(self):
219         '''
220         Monte Carlo sampling with particle filtering, Metropolis
    Hastings algorithm
221         '''
222         theta_prior = self.parameter_priors()
223         theta = np.array([theta_prior])
224         shapes = np.copy(self.shapes_prior)
225         _,_,old_log_post = self.particle_filter(theta_prior)
226         for i in tqdm(range(1,self.it)):
227             if (i % self.Usim == 0):
228                 shapes, theta_next = self.adjust_variance(theta,shapes)
229             else:
230                 theta_next = self.proposal_step(shapes,theta_prior)
231             _,_,new_log_post = self.particle_filter(theta_next)
232             prob_old,prob_next = self.scaled2_spike_prob(old_log_post,
    new_log_post)
```

```
            r = self.ratio(prob_old,prob_next,shapes,theta_next,
theta_prior)
            choice = np.int(np.random.choice([1,0], 1, p=[min(1,r),1-min
(1,r)]))
            theta_choice = [np.copy(theta_prior),np.copy(theta_next)][
choice == 1]
            theta = np.vstack((theta, theta_choice))
            theta_prior = np.copy(theta_choice)
            old_log_post = [np.copy(old_log_post),np.copy(new_log_post)
][choice == 1]
        return theta


    def adjust_variance_alternating(self,theta,par_ind,shapes):
        mean = np.zeros(self.N)
        var_new = np.zeros(self.N)
        theta_new = np.copy(theta[-1])
        u_temp = self.Ualt
        while (any(i == 0 for i in var_new)):
            for i in range(self.N):
                if i == 0:
                    mean[i] = theta[-u_temp+1::2].mean(0)[i]
                    var_new[i] = theta[-u_temp+1::2].var(0)[i]*(2.4**2)
                elif i == 1:
                    mean[i] = theta[-u_temp::2].mean(0)[i]
                    var_new[i] = theta[-u_temp::2].var(0)[i]*(2.4**2)
                else:
                    mean[i] = theta[-u_temp:].mean(0)[i]
                    var_new[i] = theta[-u_temp:].var(0)[i]*(2.4**2)
            u_temp+= 100
            if u_temp > self.it:
                return shapes, np.array([(np.random.gamma(shapes[i],
theta[-1][i]/shapes[i])) for i in range(self.N)])
        new_shapes = np.array([((mean[i]**2) / var_new[i]) for i in
range(self.N)])
        for i in par_ind:
            theta_new[i] = np.random.gamma(new_shapes[i],theta[-1][i]/
new_shapes[i])
        return new_shapes,theta_new


    def proposal_step_alternating(self,shapes,theta,par_ind):
        theta_new = np.copy(theta)
        for i in par_ind:
            theta_new[i] = np.random.gamma(shapes[i],theta[i]/shapes[i])
        return theta_new

    def alternatingMH(self):
        '''
        Alternating MH sampling
        '''
        theta_prior = self.parameter_priors()
        theta = np.array([theta_prior])
        shapes = np.copy(self.shapes_prior)
        par_ind = np.linspace(0,self.N-1,self.N).astype(int)
        _,_,old_log_post = self.particle_filter(theta_prior)
        for i in tqdm(range(1,self.it)):
```

```
283             ex = [1,0][i % 2 == 0]
284             par_ind_temp = np.delete(par_ind,ex)
285             if (i % self.Ualt == 0):
286                 shapes, theta_next = self.adjust_variance_alternating(
    theta,par_ind_temp,shapes)
287             else:
288                 theta_next = self.proposal_step_alternating(shapes,
    theta_prior,par_ind_temp)
289             _,_,new_log_post = self.particle_filter(theta_next)
290             prob_old,prob_next = self.scaled2_spike_prob(old_log_post,
    new_log_post)
291             r = self.ratio(prob_old,prob_next,shapes,theta_next,
    theta_prior)
292             choice = np.int(np.random.choice([1,0], 1, p=[min(1,r),1-min
    (1,r)]))
293             theta_choice = [np.copy(theta_prior),np.copy(theta_next)][
    choice == 1]
294             theta = np.vstack((theta, theta_choice))
295             theta_prior = np.copy(theta_choice)
296             old_log_post = [np.copy(old_log_post),np.copy(new_log_post)
    ][choice == 1]
297         return theta
298
299 data = SimulatedData(Ap=0.005, tau=0.02, std=0.001,b1=-2.0, b2=-2.0, w0
    =1.0)
300 data.create_data()
301 s1,s2,t,W = data.get_data()
302
303
304 inference = ParameterInference(s1, s2,P = 1000, Usim = 100, Ualt = 200,
    it = 1500, std=0.0001, N = 2\
305                                ,shapes_prior = np.array([4,5]),
    rates_prior = np.array([50,100]))
306 b1est = inference.b1_estimation()
307 b2est,w0est = inference.b2_w0_estimation()
308
309 theta_sim = inference.standardMH()
310 theta_alt = inference.alternatingMH()
```

Listing A.1: Python implementation that infers the learning rule parameters using Metropolis-Hastings and Particle filtering.