Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Astrid Langsrud

Inferring the learning rule from spike train data with particle Metropolis-Hastings

Master's thesis in Master of Science in Physics and Mathematics
Supervisor: Benjamin Adric Dunn
June 2020

NTNU
Norwegian University of Science and Technology

Astrid Langsrud

Inferring the learning rule from spike train data with particle Metropolis-Hastings

Master's thesis in Master of Science in Physics and Mathematics
Supervisor: Benjamin Adric Dunn
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

The brain is a system of connected neurons that communicate by transmitting electrical signals to each other. Research has revealed that the way in which neural connections develop over time seem to follow some underlying patterns. These are known as *learning rules*, and are essential for the brain to learn and form memories. Statistical methods for inferring the learning rule from recordings of neural activity may thus give insights on basic computationally principles in different brain areas. Furthermore it has been hypothesized that the learning rule might be disturbed by memory related diseases, such as Alzheimer's. Therefore, being able to detect the underlying learning rule could shed light on the origin and workings of Alzheimer's disease and even have applications in medical research as well.

This thesis covers the implementation of particle Metropolis-Hastings for characterizing the learning rule in simulated neural spike data for one synapse, inspired by the method proposed in (Linderman et al., 2014). For our purpose we used the additive spike-timing-dependent plasticity (STDP) learning rule, and aimed at inferring its learning rule parameters. The neural spiking was modeled as a Bernoulli process in the Generalized Linear Model (GLM) framework. By numerical experiments it was demonstrated that with enough data and sufficiently low noise level, information of the learning rule parameters could be reconstructed from the spike data by using this method. The results indicate that it could be possible to distinguish between learning rules, by analysing spike train data with particle Metropolis-Hastings.

Preface

This paper is my assignment for the course TMA4900 - Industrial Mathematics, Master's Thesis, at the Department of Mathematical Sciences at the Norwegian University of Science and Technology (NTNU), and concludes my five years master program Master of Science in Applied Physics and Mathematics.

I would like to thank my supervisor, Benjamin Adric Dunn, who has guided and motivated me throughout this process, and who gave me the chance to write about a topic that I find very interesting. I would also like to thank my co-supervisor Claudia Battistin, who has also followed me closely, and helped me with understanding the statistics needed to complete this work. Finally, I want to thank my family and friends for support and motivation during my five years as a student at NTNU.

I am very grateful for all the new insight in statistics, and also in neuroscience, that the work on this thesis has given me.

Thank you!

Astrid Langsrud
Trondheim, Norway
June, 2020

Table of Contents

Chapter 1

Introduction

Networks come in many forms, such as transport networks, social networks or trading networks. These can be considered as systems of connected nodes that are allowed to interact with each other. Sometimes such connections can develop over time, perhaps according to some underlying rule. Knowledge of this rule would consequently give deep insight in the nature of the network.

Among networks, the system of communicating nerve cells in the brain is perhaps one of the more fascinating. Changes over time of such neural connections is known as *synaptic plasticity*, which has been hypothesized to follow some underlying patterns, known as *learning rules*. The object of this thesis is to investigate a statistical method aimed at inferring the underlying learning rule from neural activity data.

The background motivation for the study, is an ongoing collaboration on the Kavli institute at NTNU, where researchers are establishing a protocol for growing nerve cells cultures from brains of rats and rats adapted to develop Alzheimer's disease. Synaptic plasticity is an essential mechanism for learning and memory, and it is believed that it might be affected by memory related diseases, such as Alzheimer's. Whether this is the case is an open question for research, which we would start addressing by checking whether the presence of Alzheimer's disease could be detected by differences in the underlying learning rule. Testing the accuracy and spotting the limitations of a novel statistical procedure on synthetic data is a fundamental step to be able to analyse real data and interpret the results. The present thesis thus includes a simulation study, which precedes the application of the methodology to healthy and Alzheimer's nerve cells recordings, unfortunately not yet available.

The data are modelled in the Generalized Linear Model framework, and Bayesian inference of the learning rule performed using a particle Markov Chain Monte Carlo procedure. This approach for studying synaptic plasticity was originally suggested in (Linderman et al., 2014). In that paper the authors consider the Spike Timing Dependent Plasticity (STDP) learning rule, a form of Hebbian learning also adapted here. The essence of the method implemented in this work will be similar to Linderman's but differs on some small but potentially important points. One difference is in the modelling part, where they use a Poisson model, whereas we use a Bernoulli model. Another is in the Markov chain Monte Carlo sampler, as they use Gibbs sampling and we use Metropolis-Hastings sampling. Ultimately the present work expands on Linderman's by probing the robustness of the inference procedure against some experimentally relevant variables such as noise level, data length and the presence of stimulation. Our contribution adds to a relatively new field of statistical methods for learning the learning rule from neural activity ((Stevenson and Koerding, 2011),(Linderman

et al., 2014), (Robinson et al., 2014),(Costa et al., 2013) , (Ghanbari et al., 2017)).

The thesis is structured as follows. The relevant context from neuroscience, as well as a description of the background for the data is provided in chapter 2, to give context for the work and motivate a practical understanding. Chapter 3 and 4 respectively presents statistical theory of the Generalized Linear Model framework and the particle Metropolis-Hastings method, that was used in the numerical experiments. In chapter 5, the model under consideration is introduced, as well as some visualizations of the method and justification for choices made. The results are presented in chapter 6, and the conclusion in chapter 7.

Chapter 2

Neuroscience context

Before going into the statistics and modeling, it is useful to present some context for the work. The aim of this chapter is to describe the relevant concepts from neuroscience, explain the background for the data material and provide a practical understanding. Section 2.1.1 gives a brief description of signaling and connectivity in neural networks; the source used for this section is the book (Purves, 2011). The hallmarks for Alzheimer's disease in the brain are described in section 2.1.3, whose content is based on (Gomez-Isla, 1996) and (Witter, 2011). Section 2.2 provides an outline of the lab experiments where the data that is background for this project comes from. As mentioned in the introduction, only simulated data will be studied in this work. However, since the aim is to develop a method suitable for analysing real neural data, the concepts introduced in this chapter will be important for the mathematical setup.

2.1 Concepts from neuroscience

2.1.1 Neuron and connections

The basic computational unit in the brain is the nerve cell. It consists of a cell body (soma), an axon and dendrites, as illustrated in figure 2.1.

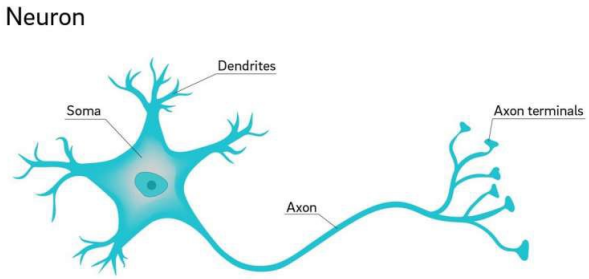


Figure 2.1: Illustration of a neuron. Source: `https://medicalxpress.com/news/2018-07-neuron-axons-spindly-theyre-optimizing.html`

The computational ability of a neuron relies on its electrochemical properties. When a neuron is at rest, there is a constant potential difference across the inside and the outside of its cell membrane. This is known as the *resting potential*. Ion channels embedded in the membrane allow ions to flow in and out, which can disturb the potential difference away from this equilibrium. If the voltage hits a certain threshold value, a rapid depolarization will be initiated. This phenomenon is known as an *action potential*, also referred to as neuron *firing* or *spiking*. Whenever the threshold potential is reached, the action potential will take place no matter what. In other words there is an all-or-non property, in its typical wave form (see figure 2.2). After being initiated in the soma the action potential will propagate along the axon, as illustrated to the right of figure 2.2.

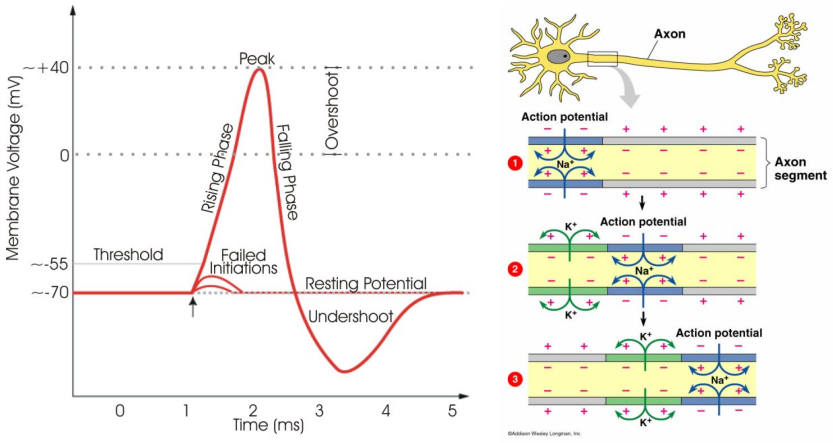


Figure 2.2: Graphical representation of an action potential (left) Source: `http://www.animalresearch.info/en/medical-advances/nobel-prizes/the-generation-of-action-potential-nerves/`. Illustration of action potential propagating along axon (right). Source: `https://www.toppr.com/ask/question/during-the-propagation-of-a-nerve-impulse-the-action-potential/`

The voltage increase that eventually leads to an action potential typically happens in response to stimuli from other neurons. Neurons in the brain are indeed connected to each other in a complex network. These connections are between an axon of one neuron and a dendrite of another, and are referred to as *synapses*. A synapse is in practice a short gap where chemical units, called neurotransmitters, are allowed to flow from the axon of a *presynaptic* neuron to the dendrite of a *postsynaptic* neuron. This neurotransmitter flow, the signal, happens when the presynaptic neuron undergoes an action potential and causes an alteration in the probability with which post-synaptic ion channels open and close. This input from the pre-synaptic neuron contributes to the membrane potential (together with on average 10^4 other neurons in the mammalian brain) which may develop into an action potential if the threshold is reached. Sometimes the electrical signal from the presynaptic neuron increases the likelihood of an action potential to also arise in the postsynaptic neuron. In this case we say that the synapse is excitatory. This property gives rise to the possibility for a signal to propagate through the neural network, and eventually end up for example in a muscle and cause a contraction. There are also synapses that decrease the chance that the postsynaptic neuron will fire when activated. These are called inhibitory synapses.

2.1.2 Synaptic plasticity

The strength of the neural connections is not fixed, but can change over time. Strength in this sense refers to the probability that the spiking in the postsynaptic neuron will be affected by an action potential in the presynaptic neuron. A frequent activation of a synapse can strengthen the synaptic connection over time. This phenomenon is called *long-term potentiation* (LTP) of a synapse. Other times activation of a synapse can weaken the connection over time. This is called *long-term depression* (LTD). These changes of connections are referred to as synaptic plasticity, which is one of the basic mechanisms underlying learning and memory.

Decades of experimental research have revealed properties in how the synaptic plasticity behaves. There are various suggested models, and common to all of them is that they rely on the Hebbian theory. In simple manners the theory says that if firing of neuron A is frequently followed by firing of neuron B, then the connection between neuron A and B will strengthen (Hebb, 1949). The functional expressions that describe this synaptic plasticity are referred to as *learning rules*.

Classical learning rules consider the instantaneous firing rates of the pre- and postsynaptic neurons. In this work the learning rule to be considered is a *spike-timing-dependent plasticity* (STDP) learning rule, for which the change in connectivity instead depends on single spikes of pre- and postsynaptic neurons at short time lags. The mathematical expression for this learning rule will be presented in section 5.1.3, were the relevant mathematical notation is defined.

2.1.3 Alzheimer's disease and the entorhinal cortex

If a patient suffers from Alzheimer's disease (AD), the brain will eventually shrink significantly. This is due to loss of synapses and neurons, which is one of the main characteristic of the disease. Figure 2.3 (right) illustrates how a brain can look like after having AD for many years. Exactly what causes these losses is still not known, but it is assumed that the accumulation of some protein aggregates called amyloid plaques and neurofirbillary tangles are involved. Another hallmark of AD is impaired neural activity, which may be related to dysfunctional plasticity mechanisms (Benedikt Zott, 2019). Therefore, it is interesting to investigate whether healthy brains and brains with AD can be distinguished due to their synaptic plasticity properties.

One target area in the brain for Alzheimer's research is the *entorhinal cortex*, which is associated with the earliest indications of AD. The entorhinal cortex is a brain region that is phylogenetically conserved across species, so research on how AD develops in rat's entorhinal cortex can give insights for humans as well. It is found in the medial temporal lobe and functions as a gateway between the neocortex and the hippocampus, which is known to be involved in declarative memory and learning. The position of the entorhinal cortex in the brain is shown in figure 2.3 (left). The entorhinal cortex is commonly subdivided into six layers, I-VI. Cells in layer II of the entorhinal cortex are shown to be affected in the initial stages of AD. Therefore layer II neurons are the subject of the experiments we will eventually get the data from to study how AD affects the synaptic plasticity.

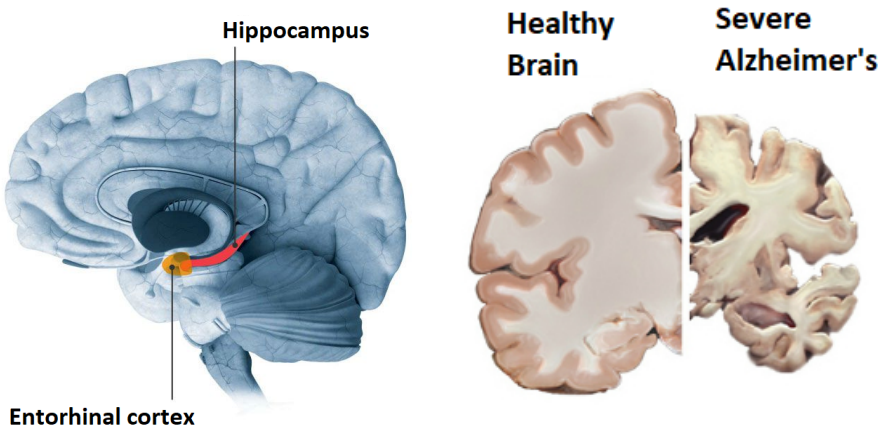


Figure 2.3: (Left) Illustration of brain showing location of the Entorhinal cortex. (Right) Visual comparison of healthy brain and brain with Alzheimer's disease. Source: `https://neurosciencenews.com/age-dementia-bmi-sleep-8989/`

2.2 Data material

The data material that is the background for this project is electric potential recordings from in-vitro cultured neural networks from rat brains. Rats do not get Alzheimer's naturally, so AD rats are designed with a genetic mutation that gives rise to amyloid plaque accumulation in their brains. It was shown that at an age of 8 months, mice with this mutation have learning impairments and behavioural differences from healthy mice (Radde R, 2006).

In short, tissue from layer II of the entorhinal cortex is gathered by microdissection from rat brains (Hanssen, 2019), and the embedded neurons are dissociated from their biological substrate to be plated into a dish. Next the seeded neurons are cultured in a medium, which allows them to survive and grow new connections. Once the network is mature, electrode arrays are then used to record the electrical activity of the neurons. Preferably each electrode should measure the activity of one neuron only. However, the recordings are extracellular, which means that the electrodes might pick up signals from several neurons. Therefore, a spike sorting procedure is performed to assign the recorded action potentials to single neurons.

It is the time points for the action potentials that are of interest, and not the actual voltage values. Hence, the relevant data material is a sequence of recorded time points for the action potentials for each neuron, in the time interval $[0, K]$. This can be written as,

$$\{\{a_i\}\}_{i=1}^{N} = \{a_{i1}, a_{i2}, ...\}_{i=1}^{N} \quad a_{ix} \in [0, K] \tag{2.1}$$

where a_{ix} is the recorded time for the $x'th$ action potential of neuron i, $a_{ix-1} < a_{ix}$, and $i = 1, 2, ..., N$ labels the neurons. Such a sequence of time stamps for a single firing neuron is called a *spike train*.

Chapter 3

Generalized linear models

In order to perform statistical analysis on the neural spike data (described in section 2.1.1), it is necessary to define a proper stochastic model for the activity. In a short time interval, referred to as a *time step*, the state of spiking or not spiking for one neuron can be considered as a Bernoulli random variable with probability parameter depending on spike history of the connected neurons and the respective connection strengths. This suggests to model the activity with help of the Generalized Linear Model (GLM) framework. In this chapter we introduce the relevant statistical theory on GLMs. The main source for this section is (Ludwig Fahrmeir, 2013).

3.1 General

Consider a system constituted by the variable Y, regarded as the response (dependent) variable and a set of variables $X_j, j = 1, ..., P$, regarded as explanatory (independent). Let $\{y, \mathbf{x}\}$ be a sample of the system. Then, in linear regression the relationship between the dependent and independent variables is modeled by the linear function

$$y = \boldsymbol{\beta}\mathbf{x} + \epsilon, \tag{3.1}$$

where $\boldsymbol{\beta}$ is a vector of regression coefficients, and ϵ is some random noise distributed as $N(0, \sigma^2)$, where σ^2 is a variance parameter.

Even though this model is useful for many situations, it has limitations. For example, if the range of the x-values is $(-\infty, \infty)$, letting an x approach infinity while everything else is kept constant makes also the corresponding y-value approach infinity (or minus infinity if β is negative). Hence, if the range of y should be restricted, the linear model is inappropriate.

Generalized linear models extends the framework of the general linear models, by allowing the response variable to come from several other distributions than the normal one. The response variable can now be distributed according to some exponential family, which are distributions on the form

$$f(y; \theta) = \exp\Big(\frac{y\theta - b(\theta)}{\phi} \cdot w + c(y, \phi, w)\Big), \tag{3.2}$$

where $b(\theta)$ and $c(y, \phi, w)$ are known functions, θ is the canonical parameter, ϕ is a nuisance parameter and w is a weight function. The expected value of the distribution, $\mathbf{E}[y] = \mu$ is related to the canonical parameter by,

$$\mu = b'(\theta). \tag{3.3}$$

An essential property of the GLM framework is that there is a specified functional relationship, g, between the linear predictor $\eta = \mathbf{x}^\mathbf{T}\boldsymbol{\beta}$ and this mean value.

The GLM framework can be summarized by the following three components:

- Response variable distributed as some member of the exponential family

$$y \sim f(y; \theta) \tag{3.4}$$

  with expected value, $\mathbf{E}[y] = \mu$.

- Linear predictor

$$\eta = \mathbf{x}^\mathbf{T}\boldsymbol{\beta} \tag{3.5}$$

- Link function

$$\eta = g(\mu) \tag{3.6}$$

If the link function maps also the mean of the response variable to the canonical parameter,

$$\theta = g(\mu), \tag{3.7}$$

it is referred to as the *canonical link function*. It then follows that $\theta = \eta$. The canonical link function is often chosen, as it comes with some advantageous properties for inferring the parameters of the GLM. So the pdf of a GLM with canonical link function can be written

$$f(y|\boldsymbol{\beta}) = \exp\Big(\frac{y\mathbf{x}^\mathbf{T}\boldsymbol{\beta} - b(\mathbf{x}^\mathbf{T}\boldsymbol{\beta})}{\phi} \cdot w + c(y, \phi, w)\Big). \tag{3.8}$$

The linear model, given by equation 3.1, is one special case of GLMs. It can be defined in the GLM framework, by specifying y as a normal distributed variable with mean μ, and having the identity link function, which is the canonical link function associated to the normal distribution. That is

$$\begin{aligned} y &\sim N(\mu, \sigma^2) \\ \mu &= \eta = \mathbf{x}^\mathbf{T}\boldsymbol{\beta}. \end{aligned} \tag{3.9}$$

3.1.1 Bernoulli GLM

In a Bernoulli process the response variable, y, takes the value 1 with a probability μ, and 0 with the probability $1 - \mu$. The corresponding probability density function is,

$$\begin{aligned} f(y|\mu) = Ber(\mu) &= \mu^y(1-\mu)^{1-y} \\ &= \exp(y \cdot \log\big(\frac{\mu}{1-\mu}\big) + \log(1-\mu)), \end{aligned} \tag{3.10}$$

where the bottom line shows that it corresponds to an exponential family. Given stochastic component set to be Bernoulli, a Bernoulli GLM is defined by the choice of a suitable link

function. As explained above, the link function relates the linear predictor η with the mean of the response variable y, which for the Bernoulli process in equation 3.10 is

$$\mathbf{E}[y] = \sum_{y=0,1} y \cdot \mu^y (1-\mu)^{1-y} = \mu \tag{3.11}$$

Since μ can only take values in [0,1], the inverse of the link function, the *response function*, have to be a mapping from the real line to $[0, 1]$. The most common is the *logit* link function, which is the canonical link in this case,

$$\eta = g(\mu) = \log(\frac{\mu}{1-\mu}) \Leftrightarrow \mu = \frac{\exp(\eta)}{1+\exp(\eta)} \tag{3.12}$$

3.2 Maximum likelihood inference via gradient descent

Given n samples of the explanatory and response variables $\{(y_i, \mathbf{x}_i)\}_{i=1,\dots,n}$, the linear predictor coefficients $\boldsymbol{\beta}$ can be estimated by maximizing the likelihood of the data

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} f(y_i|\boldsymbol{\beta}). \tag{3.13}$$

For a Bernoulli GLM this is,

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \mu_i(\boldsymbol{\beta})^{y_i} (1-\mu_i(\boldsymbol{\beta}))^{1-y_i}. \tag{3.14}$$

It is often convenient to work with the logarithm of the likelihood, which is maximized by the same $\boldsymbol{\beta}$-values as the likelihood. The loglikelihood for a Bernoulli GLM is

$$\begin{aligned} l(\boldsymbol{\beta}) = \log \prod_{i=1}^{n} (\mu_i(\boldsymbol{\beta})^{y_i} (1-\mu_i(\boldsymbol{\beta}))^{1-y_i}) &= \sum_{i=1}^{n} \log(\mu_i(\boldsymbol{\beta})^{y_i} (1-\mu_i(\boldsymbol{\beta}))^{1-y_i}) \\ &= \sum_{i=1}^{n} y_i \log(\frac{\mu_i(\boldsymbol{\beta})}{1-\mu_i(\boldsymbol{\beta})}) + \log(1-\mu_i(\boldsymbol{\beta})) \end{aligned} \tag{3.15}$$

Then, for the canonical link function, $\mu_i(\boldsymbol{\beta}) = \frac{\exp(\eta_i)}{1+\exp(\eta_i)} = \frac{\exp(\mathbf{x_i^T}\boldsymbol{\beta})}{1+\exp(\mathbf{x_i^T}\boldsymbol{\beta})}$, one arrives at

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{n} y_i \mathbf{x_i^T} \boldsymbol{\beta} - \log(1 + \exp(\mathbf{x_i^T}\boldsymbol{\beta})). \tag{3.16}$$

The goal is to find the parameters that maximizes the likelihood. For a convex problem, which is always the case for a GLM with a canonical link function, inference can be done using gradient based iterative methods. The idea of such optimization algorithms is to search in the parameter space in the direction of negative gradient to arrive at a mimima. Or equivalently, for a concave function one searches in the positive direction for the maximum. One famous such method is the Newton method. Generally, for a function $f(x)$ to be maximized in the variable x, one starts by choosing some initial guess $x^{(0)}$, and hence update the approximation in every iteration by

$$x^{(i+1)} = x^{(i)} - \frac{f'(x^{(i)})}{f''(x^{(i)})} \tag{3.17}$$

It's easy to see that the value of x at which this algorithm converges satisfies $f'(x) = 0$. The Newton method for maximum likelihood inference then entails computing the first and second derivatives of the loglikelihood called respectively score function and observed Fisher Information.

The score function is the vector of partial derivatives of the loglikelihood. In the Bernoulli case the score function can be derived as follows

$$\text{score}(\boldsymbol{\beta}) = \sum_{i=1}^{n} s_i(\boldsymbol{\beta}) = \sum_{i=1}^{n} \frac{\partial l_i(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{n} \mathbf{x_i}\Big(y_i - \frac{\exp(\mathbf{x_i^T}\boldsymbol{\beta})}{1 + \exp(\mathbf{x_i^T}\boldsymbol{\beta})}\Big) \tag{3.18}$$

The observed Fisher information matrix is defined as

$$H(\boldsymbol{\beta}) = -\frac{\partial^2 l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} = -\frac{\partial s(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}^T}, \tag{3.19}$$

which for the Bernoulli case corresponds to

$$H(\boldsymbol{\beta}) = \sum_{i=1}^{n} \mathbf{x_i} \mathbf{x_i^T} \pi_i(\boldsymbol{\beta})(1 - \pi_i(\boldsymbol{\beta})) \tag{3.20}$$

Hence, the Newton method for estimating β, is the iteration scheme

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

,

where $H(\boldsymbol{\beta}^{(i)})^{-1}$ is the matrix inverse of the observed Fisher information matrix.

Chapter 4

Particle Metropolis-Hastings

The system of interest consists of spiking neurons, with time varying connection strength developing according to a learning rule. In chapter 5 the details of this system will be explained, and the mathematical model will be defined. For now, note that this can be expressed as a state space model, where the time dependent connectivity works as a latent process essential for the inference. The connectivity can be treated as a high dimensional variable, with strong temporal correlations.

In the Bayesian paradigm, Markov chain Monte Carlo (MCMC) sampling is a class of powerful techniques for inference in multi-dimensional distributions. However, for a long sequence of highly correlated variables, a proposal distribution that mimics these correlations is required for the methods to be efficient. Particle Markov chain Monte Carlo methods are techniques able to deal with problems of this kind. The method was first introduced in 2010 by Christophe Andrieu and Arnaud Doucet in the paper *Particle Markov chain Monte Carlo methods*. The idea is to construct proposal distributions for the MCMC routine by performing a *particle filtering* method. Notice that *sequential Monte Carlo* essentially means the same as particle filtering, and that the two descriptions will be used interchangeably.

This chapter is dedicated to describe one variant of particle MCMC, the *Particle Metropolis-Hastings* method, including relevant statistical theory. In literature it is typical to use an ordinary hidden Markov model as example state space model to explain the method ((Andrieu et al., 2010), (Doucet et al., 2001), (Geof H. Givens, 2013), (Dahlin and Schön, 2015)). Therefore, this will also be used as in this chapter. The state space model to be used in this work will be introduced in chapter 5. This is slightly different, but the essence of the method is still the same. The few model specific considerations will therefore be presented in chapter 5.

4.1 A state space model

State space model refers to a representation of a stochastic dynamical system for some time dependent set of latent *states*, and a set of *observations* that have a probabilistic dependency on the latent states (Chen and Brown (2013)).

Consider the set of latent states $x^{1:T} \subset \chi$ and observations $y^{1:T} \subset \Upsilon$, and some static parameters θ. The notation $x^{1:T}$ is a short form for $\{x^1, x^2, ..., x^T\}$. Let the first latent state, x^1, come from an initial density $p_1(x^1|\theta)$, and let the following states follow a first order Markov process,

$$x^t|x^{t-1} \sim p_x(x^t|x^{t-1}, \theta). \tag{4.1}$$

The observations are dependent on the latent states through the density,

$$y_t|x_t \sim p_y(y_t|x_t, \theta). \tag{4.2}$$

The latent states cannot be observed directly, but have to be inferred from the observations. This state space model, in discrete time, is what we call a *hidden Markov model* (HMM).

4.2 Bayesian inference

The model parameters θ are unknown, and the aim is to characterize the distribution of the parameters given the observations, $p(\theta|y_{1:T})$. From beforehand we however have some knowledge on how the model parameters can be, given by the prior distribution $p(\theta)$.

The Bayesian framework serves this purpose. The goal of Bayesian parameter inference is to estimate the distribution of the model parameters based on the observations at hand. Via Bayes theorem one can obtain an expression for posterior distribution of the parameters given the observations and the prior knowledge. For the state space model presented above, this corresponds to the following expression

$$p(\theta|y_{1:T}) = \frac{p(\theta)p(y_{1:T}|\theta)}{p(y_{1:T})} = \frac{p(\theta)p(y_{1:T}|\theta)}{\int_\Theta p(\theta')p(y_{1:T}\ \theta')d\theta'}, \tag{4.3}$$

where $p(\theta)$ is the *prior distribution* for the model parameters and $p(y_{1:T}|\theta)$ is the *likelihood* of the observations given values for the model parameters. As typical in Bayesian statistics, the marginal likelihood of the data, in the numerator of equation 4.3, is hard to estimate. However, there are Monte Carlo techniques designed to overcome this problem by exploiting the proportionality of the posterior $p(\theta|y^{1:T})$ to the numerator of equation 4.3 at fixed data. Thus, $\int_\Theta p(\theta')p_{\theta'}(y_{1:T})d\theta'$ can be regarded as a normalizing constant, and we only have to consider

$$p(\theta|y_{1:T}) \propto p(\theta)p(y_{1:T}|\theta) \tag{4.4}$$

The relative amount that the likelihood and the prior contributes to the posterior depends on how wide their distributions are. If the prior knowledge is that the parameter exists in some narrow window, and this prior knowledge is very certain, the prior distribution will be very peaked. This makes the prior more dominating than if its variance was higher. As the size of the data set grows, the relative contribution of the likelihood function typically increases.

4.2.1 Metropolis-Hastings algorithm

Inference in the Bayesian framework is often performed using Markov Chain Monte Carlo methods (MCMC). MCMC is a class of algorithms for sampling from a probability distribution, which in Bayesian inference is the posterior distribution over the parameters. Markov chain Monte Carlo (MCMC) techniques utilize, as the name suggests, a combination of Markov chains with Monte Carlo sampling. A Markov chain is a sequence of events, typically time indexed, that satisfies the Markov property, which says that future events only depends on the present and not on the past. Monte Carlo techniques leverage random sampling from a distribution to make numerical estimates. However, this requires that we can sample from

the distribution, which is not always straightforward. So the idea of MCMC is to construct a Markov chain that has a limiting distribution equal to the one we want to sample from, and use the Markov chain to explore the state space accordingly. Metropolis-Hastings is one of several MCMCs. The following material is based on the source (Geof H. Givens).

The target distribution to be sampled from is the posterior, $p(\theta|y^{1:T})$, which cannot be sampled from directly. Also assume that $Q(\theta|\theta')$ is a conditional distribution of θ given θ' that is possible to draw direct samples from and that satisfies the following property

$$Q(\theta|\theta') > 0, \quad \forall\ (\theta; p(\theta|y^{1:T}) > 0),\ (\theta'; (p(\theta'|y^{1:T}) > 0) \tag{4.5}$$

Then the Metropolis-Hastings procedure for sampling from $p(\theta|y^{1:T})$ by using Q as proposal distribution is summarized in algorithm 1.

Algorithm 1

Set starting value θ^0
for $i = 0, 1, 2, \ldots$ **do**
Draw θ' from $Q(\theta'|\theta^i)$
Compute $\alpha = \frac{p(\theta'|y^{1:T})Q(\theta^i|y^{1:T})}{p(\theta^i|y^{1:T})Q(\theta'|y^{1:T})}$
$\theta^{i+1} = \begin{cases} \theta' \text{ with probability } min\{1, \alpha\} \\ \theta^i \text{ with probability } 1 - min\{1, \alpha\} \end{cases}$
end for

The resulting sequence $\{\theta^0, \theta^1, \theta^2, ...\}$ is then a Markov chain with transition probability

$$T(\theta^{i+1} = \theta|\theta^i) = \begin{cases} min\Big\{1, \frac{p(\theta'|y^{1:T})Q(\theta^i|y^{1:T})}{p(\theta^i|y^{1:T})Q(\theta'|y^{1:T})}\Big\}Q(\theta|\theta^i) & \text{if } \theta \neq \theta^i \\ 1 - \sum_{\theta \neq \theta^i} min\Big\{1, \frac{p(\theta'|y^{1:T})Q(\theta^i|y^{1:T})}{p(\theta^i|y^{1:T})Q(\theta'|y^{1:T})}\Big\}Q(\theta|\theta^i) & \text{if } \theta = \theta^i, \end{cases} \tag{4.6}$$

which can be shown to satisfy the reversibility condition,

$$p(\theta|y^{1:T})T(\theta'|\theta) = p(\theta'|y^{1:T})T(\theta|\theta'), \tag{4.7}$$

implying that $p(\theta|y^{1:T})$ is a stationary distribution of the Markov chain. Hence, for some burn in time n, we have that $\{\theta^n, \theta^{n+1}, \theta^{n+2}, ...\}$ is an approximate sample from $p(\theta|y^{1:T})$. The purpose of the burn in period is to avoid dependency on the starting value of the sampling.

4.2.2 Choice of proposal distribution

The choice of proposal distribution is essential for the efficiency of the Metropolis-Hastings algorithm. In theory, every distribution that covers the range of the target distribution could do the job for big enough number of iterations. However, it is advantageous if the proposal distribution does not differ too much from the target distribution (Geof H. Givens). Typically the proposal distribution is constructed as a random walk by conditioning on the current state (Robert and Casella, 2005), such that $(\theta^{i+1} - \theta^i) \sim N(0, \Sigma)$, where i labels the iterations and Σ is a fixed covariance matrix.

The variance of these proposals affect the result of the sampling. If the variance in the proposal distribution is too large, many proposed parameter values will be rejected. This means that it will take many iterations to obtain the wanted sample. Also, in finite time one may end up missing or oscillating around some maximum. On the other hand, if the variance

is too low, the algorithm will struggle to cover the relevant parameter space, and can more easily get stuck at local maximas in the likelihood. A way to avoid these scenarios is to adjust the variance along the way, according to the variation in the already sampled values.

4.2.2.1 Adaptive proposal

In (Roberts et al., 1997) the authors suggest to rescale the proposal variance in order to keep fixed the acceptance rate $\sim 25\%$. Fixing the acceptance rate corresponds to controlling the trade-off between exploration and exploitation of the sampling algorithm and therefore its efficiency. To this aim the authors in (Haario and Saksman, 1998) propose to exploit the variance of the collected samples, such that a normal proposal distribution centered at the current sample is given by:

$$Q(\cdot|\theta^{1:i}) \sim N(\theta^i, c_d^2 R_i) \tag{4.8}$$

where R_t denotes the empirical variance of the set of the H last sampled values, $\{\theta^{i-H+1}, \theta^{i-H+2}, ..., \theta^i\}$. We refer to H as a memory parameter. c_d is a scaling factor, dependent on the dimension of the target distribution. For the one-dimensional case, it takes the value 2.4.

It is not needed to undergo this variance adjustment in every iteration. Define a fixed number U, which is the *update frequency*. The variance can be recomputed as $c_d^2 R_i$ every U iteration.

4.3 Sequential Monte Carlo

Since in general for HHMs, the form of $p(\theta|y^{1:T})$ is unknown, it is useful to reformulate it as proportional to $p(\theta)p(y_{1:T}|\theta)$, for performing Metropolis-Hastings sampling. The prior, $p(\theta)$, can typically be evaluated directly, but the likelihood, $p(y^{1:T}|\theta)$, has to be approximated. It can be expressed with the joint distribution of products of conditional distributions as follows,

$$p(y_{1:T}|\theta) = p(y_1|\theta) \prod_{t=2}^{T} p(y_t|y_{1:t-1}, \theta). \tag{4.9}$$

Each factor can be obtained by integrating

$$p(y_t|y_{1:t-1}, \theta) = \int p_y(y_t|x_t, \theta) p_x(x_t|x_{t-1}, \theta) p(x_{1:t-1}|y_{1:t-1}, \theta) dx^{1:t}, \tag{4.10}$$

which can be approximated with a particle filtering routine. This section is dedicated to explain how the particle filtering method works.

4.3.1 Importance Sampling

Consider the probability distribution, $p(x)$, that we want to characterize, but are unable to sample from. Importance sampling is a method that makes use of a sampling function, $g(x)$, that we can sample from, to generate an approximate sample from $p(x)$.

Let's first assume we are able to sample from p(x) and are looking for an estimate of $\int h(x)p(x)dx$. A Monte Carlo estimate of the latter (Geof H. Givens, 2013) is

$$\frac{1}{K} \sum_{k=1}^{K} h(x_k), \tag{4.11}$$

where each x_k is sampled from $p(x)$. When $p(x)$ is impossible or inconvenient to sample from, *importance sampling* can be used. This relies on sampling x_k from another distribution $g(x)$, an *importance sampling function*, and then use the rewriting

$$\int h(x)p(x)dx = \int h(x)\frac{p(x)}{g(x)}g(x)dx. \tag{4.12}$$

to get an approximation of the desired integral on the left-hand size of equation 4.12 as

$$\frac{1}{K}\sum_{k=1}^{K} h(x_k)v(x_k), \tag{4.13}$$

where

$$v(x_k) = \frac{p(x_k)}{g(x_k)} \tag{4.14}$$

are unnormalized *importance weights*.

If $p(x)$ is only known up to a normalizing constant, one can standardize the importance weights, so that they add up to 1,

$$v^*(x_k) = \frac{v(x_k)}{\sum_{k'=1}^{K} v(x_{k'})}. \tag{4.15}$$

Based on the sample, one effectively gets an empirical approximate of the distribution $p(x)$,

$$p(dx) = \sum_{k=1}^{K} v^*(x_k)\delta_{x_k}(dx). \tag{4.16}$$

which can be used to generate an approximate sample from $p(x)$. Essentially, this means drawing a sample among the x's, each with a probability mass corresponding to the weight. It can be proven that as $K \Rightarrow \inf$ this distribution converges to $p(x)$ (Geof H. Givens, 2013).

4.3.2 Sequential Importance Sampling

For the state space model, the target density to be characterized is $p(x^{1:T}|y^{1:T}, \theta)$. The purpose is to obtain a sample of P sequences $\{x_p^{1:T}\}_{p=1}^{P}$ with corresponding importance weights, that together give an approximation for $p(x^{1:T}|y^{1:T}, \theta)$ of the form,

$$p(dx^{1:T}|\theta) = \sum_{p=1}^{P} v^*(x_p^{1:T})\delta_{x_p^{1:T}}(dx^{1:T}). \tag{4.17}$$

The sample sequences $\{x_p^{1:T}\}_{p=1}^{P}$ will be referred to as *particles*.

The importance sampling procedure described above can be applied for multidimensional densities, but as the number of dimensions increases it becomes less efficient. Sequential importance sampling takes a different approach, aimed at overcoming this problem. In sequential importance sampling, also called *particle filtering*, the idea is that, instead of sampling the whole trajectory $x^{1:t}$, one sample is generated for one time step $x^{t'}$ at a time, building on knowledge from the previously sampled time steps $x^{1:t'-1}$. The following equations will show why this is possible. For the rest of the section, conditioning on θ is assumed in all densities, and is omitted for simplicity.

Notice that for the HHM defined in section 4.1, $p(x^{1:t}|y^{1:t})$ follows a recursive relation (Geof H. Givens, 2013),

$$p^t(x^{1:t}|y^{1:t}) = p^{t-1}(x^{1:t-1}|y^{1:t-1}) \cdot p_x(x^t|x^{t-1}) \cdot p_y(y^t|x^t). \tag{4.18}$$

Let the importance sampling function be defined on the form

$$\begin{cases} g^1(x^1) \\ g^t(x^t|x^{1:t-1}, y^{1:t-1}). \end{cases} \tag{4.19}$$

This structure allows importance sampling at individual time steps.

The procedure begins by drawing P samples from $g^1(x^1)$, and weighting the samples according to

$$v^1(x_p^1) = \frac{p^1(x_p^1|y^1)}{g^1(x_p^1)} \propto \frac{p_x(x_p^1) \cdot p_y(y^1|x_p^1)}{g^1(x_p^1)}. \tag{4.20}$$

Then, an approximation of $p^1(x^1|y^1)$ is obtained as

$$\hat{f}^1(x^1|y^1) = \sum_{p=1}^{P} v^*(x_p^1)\delta_{x_p^1}(dx) \tag{4.21}$$

where $v^*(x_p^1)$ are the weights normalized as in equation 4.15.

The particle filtering proceeds by iteratively drawing P samples $x_p^t \sim g^t(x^t|x_p^{t-1}, y^{1:t-1})$ and combining that with the ancestor particle,$\{x_p^{1:t-1}\}$,to get $\{x_p^{1:t}\}$. According to the recursive relation in equation 4.18 the weight at time t, given the atomic approximation of $p(x^{1:t-1}|y^{1:t-1})$ is

$$v^t(x_p^{1:t}) = \frac{p_x(x^t|x^{t-1}) \cdot p_y(y^t|x^t)}{g^t(x_p^t|x_p^{1:t-1})} \cdot v^{t-1}(x_p^{1:t-1}). \tag{4.22}$$

One common choice for the sampling distribution to equal p_x, namely

$$g^t(x_p^t|x_p^{1:t-1}) = p_x(x^t|x^{t-1}). \tag{4.23}$$

In this case, the particle weights are updated as

$$v^t(x_p^{1:t}) = p_y(y^t|x^t) \cdot v^{t-1}(x_p^{1:t-1}). \tag{4.24}$$

In the end this method produces a collection of P particles and particle weights, $\{x_{(p)}^{1:T}, v^T(x_{(p)}^{1:T})\}$, making up the distribution of equation 4.17.

4.3.3 Resampling

As the particles propagate in time, their respective likelihoods are updated. Since the particle updates contain a random element, very unlikely values will occasionally be added, resulting in a permanent decrease of the particles' likelihood. For a long time sequences, this will often lead to a situation where one particle dominates in terms of weights, while the rest will have negligible weights in comparison. Such a sample of particles will be a poor representation of the target distribution (Doucet et al., 2001), whereas a more effective representation would be given by samples whose particles' weights are comparable in magnitude.

This problem can be avoided by including a resampling mechanism in some of the iterations. The idea is to draw a bootstrap sample from the set of particles, and then use this

sample as particles further in the process. In practice this means that, on average, the likeliest particles will be copied, replacing the unlikely ones.

The method proceeds as follows. At time step t of the particle filtering, one has a set $\{x_{(p)}^{1:t}, v(x_{(p)}^{1:t})\}_{p=1}^{P}$ of particles and particle weights. To perform resampling, it is necessary to compute the normalized weights using equation 4.16. Since the normalized weights add up to one, they can function as probabilities for the corresponding particle to become resampled. Resampling involves drawing a set $\{i_1, i_2, ..., i_P\}$ of P indexes with replacement from $\{1, 2, ..., P\}$, with probabilities $\{v^*(x_{(1)}^{1:t}), v^*(x_{(2)}^{1:t}), ..., v^*(x_{(P)}^{1:t})\}$, and obtain the new set of particles $\{x_{(i_1)}^{1:t}, x_{(i_2)}^{1:t}, ..., x_{(i_P)}^{1:t}\}$. This way of resampling is called *multinomial resampling* since the number of replicates of a particle is distributed multinomially (Naesseth et al., 2019). After resampling, all particles are assigned identical particle weights, equal to $\frac{1}{P}$.

4.3.3.1 When to resample

Resampling causes a decrease of diversity of particle samples, resulting in a increased variance of the Monte Carlo estimators (Martino et al., 2017). Therefore, it is advantageous to only resample when it is necessary. To determine when this is the case, it is relevant to assess the *effective sample size* (ESS) of the set of particles. The ESS is a measure of how big sample from the actual target distribution that our particle sample is worth (Geof H. Givens (2013)). In the best case scenario, the particle sample corresponds to a sample from the target distribution, with all normalized particle weights equal to $\frac{1}{P}$. In that case the ESS is P. Worst case is that one of the normalized particle weights equalt 1, while the rest equals zero. This corresponds to an ESS of 1.

There are various measures for ESS. In (Cappé et al., 2008) they present a measure, *perplexity*, defined as

$$\exp(H(\mathbf{v})/P), \tag{4.25}$$

where

$$H(\mathbf{v}) = -\sum_{i=1}^{P} v^*(x_{(p)}^{1:t}) \log(v^*(x_{(p)}^{1:t})), \tag{4.26}$$

is the Shannon entropy of the sample. Notice that this differs from the (approximate) entropy of the posterior distribution, since the sum runs over the particles, which are not necessarily distinct states of the system.

The idea is then to keep track of the perplexity of the particle sample, and resample whenever the perplexity becomes lower than some threshold value.

In (Martino et al. (2017)) it is suggested a way to set this threshold. In section 6 of the paper they study ESS-values for sets of uniformly distributed vectors in the unit simplex, and compute the distribution of these for different ESS measures. Subsequently they suggest to set a threshold for resampling equal to the mean of these distributions. For perplexity this mean is 0.66.

4.3.4 Sequential Importance Sampling Resampling Algorithm

To wrap up this section we present a short summary of the particle filtering procedure, the way it will be implemented in this work. Now the parameter θ is included in the notation to remind the reader that this is an essential component of the model, even though kept constant while running the algorithm..

Given a specified parameter value θ, and a set of observations $y^{1:T}$, the aim is to characterize the posterior distribution $p(x^{1:T}|y^{1:T},\theta)$ of the sequence of latent variables $x^{1:T}$. The procedure is initiated by drawing P values from $\pi(x^1)$, that makes up the set of particles for the first time step. Then, we move sequentially through the time line, each step appending samples to the particles, drawn from $g^t(x_p^t|x_p^{1:t-1})$. Before moving to the next time step, the perplexity in the set of particles is evaluated. If that goes below some threshold value, resamplig is performed. Finally a set of particles and particle weights $\{x_{1:P}^{1:T}, v_{1:P}^T\}$ are obtained, making up a discrete approximate of the target distribution.

The pseudocode for this procedure is given in algorithm 2.

Algorithm 2 Sequential Monte Carlo

Sample $x_{1:P}^1 \sim \pi(x^1)$
Compute the particle weights as $v_{(p)}^1 = p(y^1|x_{(p)}^1,\theta)$
for $t = 2, 3 \ldots$ **do**
    Sample $x_{1:P}^t \sim p_x(x_{1:P}^t|x_{1:P}^{1:t-1},\theta)$
    Compute particle weights $v_{(p)}^t = p(y^t|x_{(p)}^t,\theta) \cdot v_{(p)}^{t-1}$
    if Perplexity of particles < 0.66 **then**
        Normalize particle weights
        Resample
        Reset weights $v_{(p)}^t = \frac{1}{P}$
    end if
end for

4.4 Particle marginal Metropolis-Hastings procedure

Now, the material in this chapter can be combined into a particle Metropolis-Hastings procedure for inferring $p(\theta|y^{1:T})$.

For a given θ-value, the distribution $p(x_{1:T}|y_{1:T},\theta)$ is approximated by equation 4.17 obtained from the particle filtering. In combination with equation 4.9 and 4.10 we see that we can approximate $p(y^{1:T}|\theta)$ as

$$\hat{p}(y^{1:T}|\theta) = \prod_{t=1}^{T} \frac{1}{N} \sum_{p=1}^{P} v_{(p)}^t \tag{4.27}$$

Thus, we can run a Metropolis-Hastings sampler, each iteration proposing a θ-value and run a particle filter with this value. The complete procedure is summarized in algorithm 3.

Algorithm 3 Particle marginal Metropolis-Hastings sampler

Set starting value θ^0
Run particle filter targeting $\hat{p}(x_{1:T}|y_{1:T}, \theta^0)$
Calculate $\hat{p}_(y_{1:T}|\theta^0)$
for $i = 1, 2 \ldots$ **do**
  if (i mod U) $= 0$ **then**
    Adjust variance of proposal density q
  end if
  Sample $\theta^* \sim q(\cdot|\theta^{i-1})$
  Run particle filter targeting $\hat{p}(x_{1:T}|y_{1:T}, \theta^*)$
  Calculate $\hat{p}_{\theta^*}(y_{1:T})$
  $r = \frac{p(y_{1:T}|\theta^*)p(\theta^*)q(\theta^{i-1}|\theta^*)}{p(y_{1:T}|\theta^{i-1})p(\theta^{i-1})q(\theta^*|\theta^{i-1})}$
  $\theta^i = \begin{cases} \theta^* \text{ with probability } min\{1, r\} \\ \theta^{i-1} \text{ with probability } 1 - min\{1, r\} \end{cases}$
end for

Chapter 5

Experimental setup

Now we have presented the context for the work as well as the relevant mathematical concepts needed for the inference procedure. The approach is to construct a model for neural activity and plasticity, simulate spike data accordingly and test the particle Metropolis-Hastings procedure for inference of the learning rule parameters. This chapter presents the components of the model to be investigated, as well as the experimental procedure for inference. Material from chapter 3 and 4 is assumed known, so the content here is in the context of the neural model.

The chapter is divided into three sections. In section 5.1 the mathematical notation is defined and the system under consideration is presented. To begin with, the model is presented in context of several connected neurons. Even though the experimental method in this work only targets a single synapse, this is included to give the reader an indication that the method can be expanded to a bigger network of neurons. Today we have equipment for measuring activity of several connected neurons in the lab, so this scenario is relevant for real data. Section 5.2 deals with some models specific considerations for applying the particle Metropolis-Hastings method. Finally, section 5.3 gives a thorough description of the steps implemented, including justifications of choices made and figures visualizing the process. This is included to increase the reproducibility of the work, and as a natural link to the following chapters, which present numerical tests of the method introduced here.

5.1 Model

5.1.1 Framework

As described in section 2.2, neural activity measured in the lab comes in the format of time points of action potentials in time interval $[0, K]$ for the N neurons, as given by equation 2.1. Let the time line be divided into T equally sized bins, and number these bins,

$$t \in \{1, 2, ..., T\}. \tag{5.1}$$

Also, let $s_i^t \in \{0, 1\}$ be a binary variable taking value 1 if neuron i fires at least once in the time bin t, and 0 otherwise. This is illustrated in figure 5.1, where the top array is a spike train, and the bottom array is the corresponding binary values for the time bins.

We model the variable s_i^t as a non-homogeneous Bernoulli process with expected value μ^t, also referred to as *spike rate*, that can be understood as the probability that the neuron will

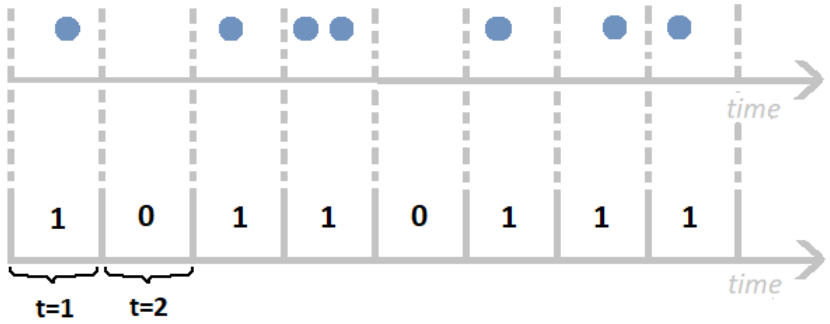


Figure 5.1: Upper time line illustrates the time points, a_x, for action potentials. Bottom time line illustrates the corresponding binary value for the defined time bins.

fire in the time bin t. Within the Bernoulli GLM framework, introduced in section 3.1.1, the spike rate is calculated from some linear predictor, η_i^t through a logit link function.

$$\begin{aligned} P(s_i^t|\mu_i^t) &= \text{Ber}(\mu_i^t) = \mu_i^{t\,s_i^t}(1-\mu_i^t)^{1-s_i^t}, \\ \mu_i^t &= h(\eta_i^t) = \frac{\exp(\eta_i^t)}{1+\exp(\eta_i^t)}. \end{aligned} \tag{5.2}$$

In our neural network model the linear predictor is a linear combination of the states of the neurons at the previous time step and a background noise, b_i. This is expressed as,

$$\eta_i^t = \Big(\sum_{j=0}^{N} w_{ji}^t s_j^{t-1}\Big) + b_i. \tag{5.3}$$

Here $w_{ji}^t \in \mathbb{R}$ is a *weight* between neuron j and neuron i at time step t, and represents the strength of the connection between the two neurons. The contribution to the linear predictor for μ_i^t from neuron j is $w_{ji}^t s_j^{t-1}$. A positive value for w_{ji}^t corresponds to an excitatory synaptic connection, whereas a negative weight represents an inhibitory one. A weight with value zero, means that there is no connection.

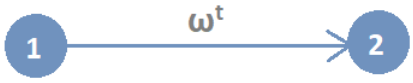
Normally these weights are considered stationary when neural activity is modeled. This makes things simpler and reduces computational power needed. In this work we aim to study synaptic plasticity, so the weights are set to vary with time. The connectivity of the whole network of neurons at each time step t can thus be summarized by a time dependent $N \times N$ weight matrix, W^t, which for three neurons would look like,

$$W^t = \begin{pmatrix} w_{11}^t & w_{12}^t & w_{13}^t \\ w_{21}^t & w_{22}^t & w_{23}^t \\ w_{31}^t & w_{32}^t & w_{33}^t \end{pmatrix}.$$

Considering the whole time line, for each neural connection there is a sequence of T weights, $w_{ji}^{1:T}$. This will be referred to as a *weight trajectory*. The way in which this trajectory develops in time is assumed to follow a parametric learning rule, that we aim to infer. Therefore, the weight trajectory serves as a latent process in this context.

5.1.2 System for investigation

The system to be considered in this work consists of two neurons, neuron 1 and neuron 2, with one directed synaptic weight, ω^t



Neuron 1 has a constant probability for spiking, according to a background parameter b_1. Neuron 2 has spiking probability that depends on the spiking of neuron 1 in the previous time step through the linear predictor $b_2 + \omega^t \cdot s_1^{t-1}$. The actual spiking rate is related to the linear predictor through a logit link. The distributions of the stochastic variables s_i^t, representing the spiking in this system, are given by equations

$$s_1^t \sim \text{Ber}(\mu_1) \qquad \mu_1 = \text{logit}^{-1}(b_1) \tag{5.4}$$

$$s_2^t | s_1^{t-1}, \omega^t \sim \text{Ber}(\mu_2^t) \qquad \mu_2^t = \text{logit}^{-1}(\omega^t \cdot s_1^{t-1} + b_2) \tag{5.5}$$

equivalent to

$$p(s_1^t) = \mu_1^{s_1^t}(1-\mu_1)^{1-s_1^t} \tag{5.6}$$

$$p(s_2^t | s_1^{t-1}, \omega^t) = (\mu_2^t)^{s_2^t}(1-\mu_2^t)^{1-s_2^t}. \tag{5.7}$$

5.1.3 Spike timing dependent plasticity

The way in which the weight trajectory develops over time is given by,

$$p(\omega^{t+1} | \omega^t, s_1^{1:t}, s_2^{1:t}, \theta) = \omega^t + l(s_1^{1:t}, s_2^{1:t}, \theta) + \epsilon(\sigma), \tag{5.8}$$

where l is a *learning rule*, and $\epsilon(\sigma)$ is a noise term. Here we apply the STDP learning rule, which takes the following form

$$l(s_i^{1:t}, s_j^{1:t}, \theta) = l_+(s_i^{1:t}, s_j^{1:t}, A_+, \tau_+) - l_-(s_i^{1:t}, s_j^{1:t}, A_-, \tau_-)$$

$$l_+(s_i^{1:t}, s_j^{1:t}, A_+, \tau_+) = s_j^t \sum_{t'=1}^{t} s_i^{t'} A_+ e^{(t-t')/\tau_+}$$

$$l_-(s_i^{1:t}, s_j^{1:t}, A_-, \tau_-) = s_i^t \sum_{t'=1}^{t} s_j^{t'} A_- e^{(t-t')/\tau_-},$$

where, $\theta = \{A_+, A_-, \tau_+, \tau_-\}$ are learning rule parameters, and is the object for inference. The parameters τ_+ and τ_- control the scale of lags in which the firing contributes to connection updates. Decreasing the τ value corresponds to shrinking the window for where firing has significant impact on the plasticity. The parameters A_+ and A_- scale the size of the updates, and correspond to the maximum value of connectivity updates when $\Delta t = |t - t'|$ is small. Figure 5.2 illustrates the learning rule, for two different combinations of learning rule parameters.

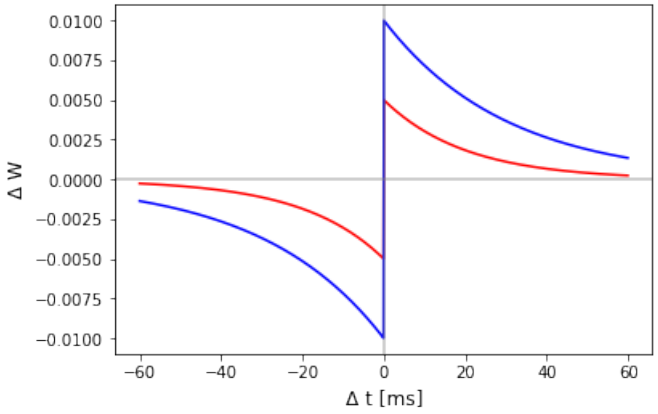


Figure 5.2: Shape of the additive STDP learning rule for $\theta = \{0.01, 0.01, 30ms, 30ms\}$ (blue) and $\theta = \{0.005, 0.005, 20ms, 20ms\}$ (red), as a function of the time lag Δt between a spike of neuron 2 and a spike of neuron 1.

5.2 Particle Metropolis-Hastings for spike data

The objective is to characterize the posterior of the learning rule parameters given this spike data,

$$p(\theta|s_2^{1:T}). \tag{5.9}$$

The dependency on the spikes of neuron 1 is assumed in all conditional distributions in this section, but omitted for simplicity. Both the spikes of neuron 2 and the weight trajectory are strongly dependent on the spikes of neuron 1. However, neuron 1 is only dependent on the static parameter, b_1, and will therefore be considered as a fixed vector of parameters.

The distribution in question is characterized by implementing a particle Metropolis-Hastings sampler. Since its form is not known explicitly, we rewrite with help of the Bayesian theorem as in equation 4.4,

$$p(\theta|s_2^{1:T}) \propto p(s_2^{1:T}|\theta) \cdot p(\theta), \tag{5.10}$$

where $p(s_2^{1:T}|\theta)$ is the likelihood of the spike data given the learning rule parameters, and $p(\theta)$ denotes the prior distribution of the parameters. The posterior in equation 5.10 will be characterized using Metropolis-Hastings sampling, including a particle filtering step to approximate $p(s_2^{1:T}|\theta)$. As explained in section 4.4, the target distribution for the particle filtering is that of the latent states, in this case

$$p(\omega^{1:T}|s_2^{1:T}, \theta), \tag{5.11}$$

leading to the approximated distribution

$$\hat{p}(\omega_{(p)}^{1:T}|s_2^{1:T}, \theta) = \sum_{p=1}^{P} = \delta_{\omega^{1:T}} \omega_{(p)}^{1:T} v^*(\omega_{(p)}^{1:T}) \tag{5.12}$$

The sampling procedure will be implemented as presented in chapter 4. However, notice that the distribution for the time updates of the latent process, equation 5.8, differs from that in the ordinary HMM, as it is conditioned also on the observations. Despite this modification,

as it will be explained in section 5.2.1, our model is still suitable for inference with particle Metropolis-Hastings. The rest of this section is dedicated to present some model specific considerations for the procedure.

5.2.1 Choice of importance sampling function

For the particle filtering procedure it is necessary to specify which importance sampling function to use. Our intuition tells us that the closer the sampling distribution to the target, the fewer samples are required to provide a good estimate.

By employing Bayes rule, our target distribution $p(\omega^{1:T}|s_2^{1:T},\theta)$ can be rewritten as follows,

$$p(\omega^{1:T}|s_2^{1:T},\theta) = p(\omega^T|\omega^{1:T-1},s_2^{1:T},\theta)\cdot p(\omega^{1:T-1}|s_2^{1:T},\theta) \tag{5.13}$$

$$= p(\omega^T|\omega^{T-1},s_2^{1:T},\theta)\cdot \frac{p(\omega^{1:T-1},s_2^T|s_2^{1:T-1},\theta)}{p(s_2^T|s_2^{1:T-1},\theta)} \tag{5.14}$$

$$= p(\omega^T|\omega^{T-1},s_2^{1:T},\theta)\cdot \frac{p(s_2^T|\omega^{T-1},\theta)}{p(s_2^T|s_2^{1:T-1},\theta)}\cdot p(\omega^{1:T-1}|s_2^{1:T-1},\theta), \tag{5.15}$$

where in equation 5.14 and 5.15 the Markov property of the learning rule and of the Bernoulli GLM was used respectively. The first factor corresponds to the weight updates from the learning rule, equation 5.8, whereas the numerator of the second factor corresponds to the GLM, equation 5.7. Equation 5.15 is a recursive relation in time for the posterior of the weights trajectory, which mirrors equation 4.18. This suggests to use particle filtering to sample from this target distribution and adopt the learning rule $p(\omega^{t+1}|\omega^t,s_1^{1:t},s_2^{1:t},\theta)$ from equation 5.8 as the sampling distribution.

5.2.2 Importance weights

The importance weights are defined in equation 4.14 as the ratio between the target distribution to the sampling distribution (section 4.3.1). This gives the importance weights,

$$v(\omega_{(p)}^{1:T}) = \frac{p(\omega_{(p)}^{1:T}|s_2^{1:T-1},\theta)}{p(\omega^T|\omega^{T-1},s_2^{T-1})} \tag{5.16}$$

$$= \frac{p(s_2^T|\omega_{(p)}^{T-1},s_2^{T-1},\theta)p(\omega_{(p)}^{1:T-1}|s_2^{1:T-1},\theta)}{p(s_2^T|s_2^{1:T-1},\theta)}. \tag{5.17}$$

This would require having an estimate of $p(\omega_{(p)}^{1:T-1}|s_2^{1:T-1},\theta)$, but also computing the normalization $p(s_2^T|s_2^{1:T-1},\theta)$, which could be cumbersome.

Notice that when we normalize the weights, the latter simplifies:

$$v^*(\omega_{(p)}^{1:T}) = \frac{v(\omega_{(p)}^{1:T})}{\sum_{p'=1}^P v(\omega_{(p')}^{1:T})} \tag{5.18}$$

$$= \frac{p(s_2^T|\omega_{(p)}^{T-1},s_2^{T-1},\theta)p(\omega_{(p)}^{1:T-1}|s_2^{1:T-1},\theta)}{\sum_{p'=1}^P p(s_2^T|\omega_{(p')}^{T-1},s_2^{T-1},\theta)p(\omega_{(p')}^{1:T-1}|s_2^{1:T-1},\theta)} \tag{5.19}$$

So instead we define our unormalized weights as

$$v(\omega_{(p)}^{1:T}) = p(s_2^T|\omega_{(p)}^{T-1}, s_2^{T-1}, \theta)p(\omega_{(p)}^{1:T-1}|s_2^{1:T-1}, \theta), \tag{5.20}$$

Since $p(\omega_{(p)}^{1:T-1}|s_2^{1:T-1}, \theta)$ is approximated according to equation 5.12, the system in sum leads to the iterative procedure of sampling from 5.8 and weight as

$$p(s_2^T|\omega_{(p)}^{T-1}, s_1^{T-1}, \theta) \cdot v^*(\omega_{(p)}^{1:T}). \tag{5.21}$$

5.2.3 Prior of learning rule parameters

As a prior distribution for the learning rule parameters, we use a gamma distribution as in (Linderman et al. (2014)). Since the parameters cannot be negative, gamma is a natural choice. The gamma distribution function is

$$f(x) = \frac{\beta^\alpha}{\Gamma(k)} x^{\alpha-1} e^{-\beta x}, \tag{5.22}$$

where α and β is the *shape* and *rate* parameter, respectively. Linderman suggests to use shape parameter equal to 1, and rate parameters $\{50, 150, 100, 100\}$ for A_+, A_-, τ_+ and τ_- respectively. Figure 5.3 shows an illustration of the corresponding prior distribution for A_+.

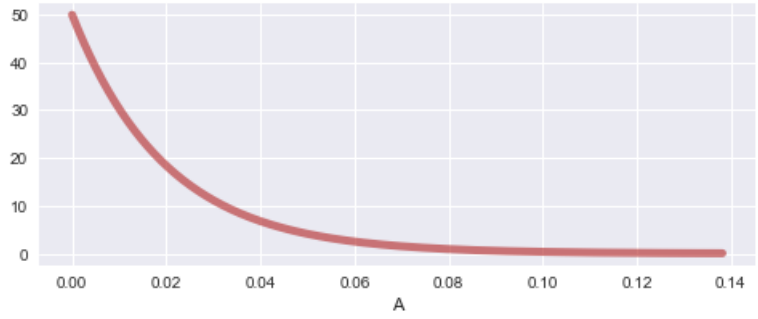


Figure 5.3: Prior distribution for A_+

5.3 Method and analysis

In this work we assess the capability of Particle Metropolis-Hastings to infer the learning rule parameters on synthetic data, which gives us control over the generative model and lets us compare the results with the ground truth. While the setup introduced in the previous sections allows for several focuses for inference, there is still a number of degrees of freedom to be pinned when it comes to its implementation and testing on synthetic data. For instance which parameters to use in the generative model and how to infer the background noise parameters b_1 and b_2, are choices to be made. Now we present a description of how the involved parts were carried out in this work. The aim of the section is to guide the reader through the method implemented, including visualisations, considerations of choices made on the way and some preliminary results, with the intention that the results in chapter 6 would be possible to reproduce.

5.3.1 Simulating the generative model

Spike data for two neurons with one directed synaptic connection and a trajectory of weights updating according to the STDP rule was generated by numerical simulations, with a 5ms time resolution (time bin size). Since the weight update in each time step is dependent on the previous spiking activity, and the activity is in turn dependent on the weight, it was necessary to generate the two in parallel. The spike data was simulated according to the Bernoulli processes in equation 5.4 and 5.5, and the weight trajectory was evolved with the STDP learning rule as in equation 5.8. The noise term in the learning rule was set to be a gaussian distribution centered at 0, with a variance parameter σ^2. Various sizes for the noise level were used in the experiments to compare the outcomes.

The value of the learning rule parameters is a property of the involved synapses, which in this case has to be specified in the simulations. In some of the experiments, comparisons of different sets of parameters will be made. For the remaining experiments it is practical to stick to one fixed set of parameters. These typical values for the learning rule parameters in this generative model that was used in this work is presented in table 5.1. These values are chosen as they are the same as those used in (Sen Song, 2000).

Parameter	Value
A_+	0.005
A_-	0.00525
τ_+	20
τ_-	20

Table 5.1: Values for learning rule parameters used in simulations of the generative model, unless specified differently

5.3.1.1 Some evaluations of the data simulation

Since the study is based on simulated data, it was necessary to make sure that the simulations produced reasonable data. In addition to the learning rule parameters, there are a few other parameters, $\{b_1, b_2, \omega^0\}$ essential for the simulations. Different combinations of these generate data with different properties, which may not be representative of the neural system of interest. For simplicity and time limitations it was preferable to chose some values for these, which then could be held fixed. The values were inspired by those used in (Linderman et al., 2014), but also based on a qualitative pre-analysis, ensuring that the chosen combinations lead to realistic data in terms of firing rates of neurons. Since the spike data and weight trajectory are generated together and closely related, the simulations were evaluated by visual inspection of the weight trajectories produced.

In this work, a weight trajectory was considered to be appropriate if it demonstrated learning over time, without getting too big or too small. For the system and learning rule under consideration, the process behaves so that if the weight reaches a certain threshold in absolute value, it will develop monotonically upwards or downwards from there. This means that the weight learns over time, and is a shape we are looking for. However, since the learning rule is unbounded, the weight will eventually reach a point where there is no longer in practice a change in spike rate. This is a consequence of the nature of the logit link function. Therefore, in addition to set appropriate values for $\{b_1, b_2, \omega^0\}$, it was also necessary to specify an upper bound for the data length.

In Linderman's simulations, a baseline firing rate with mean $20s^{-1}$ is used. Therefore, it was chosen to test values for background parameters that gave a spike rate of comparable

size to this for modeling a scenario of natural neural activity. In the lab, one can drive the spike rate of a neuron by stimulation. So in addition, we wanted to make some data sets for a situation where the activity of neuron 1 is highly driven by the external stimulus, by setting higher values for the parameter b_1 and therefore increasing its average firing rate.

Based on visual inspections of plots of weight trajectories and spike rates, some parameters for the simulated data were chosen. These are summarized in table 5.2.

Parameter	Value
b_1	-2 or 1
b_2	-2
W_0	1
Time (s)	120 s

Table 5.2: Parameter values to be used in the experiments, unless specified differently

The value $b_1 = -2$ gives a Bernoulli spike rate parameter of $\mu_1 = 0.12$. For the scenario where neuron 1 is stimulated, $b_1 = 1$ is used. This corresponds to $\mu_1 = 0.73$.

Figure 5.4 shows example trajectories of the chosen combinations of parameters, with low and high spike rate of neuron 1 respectively.

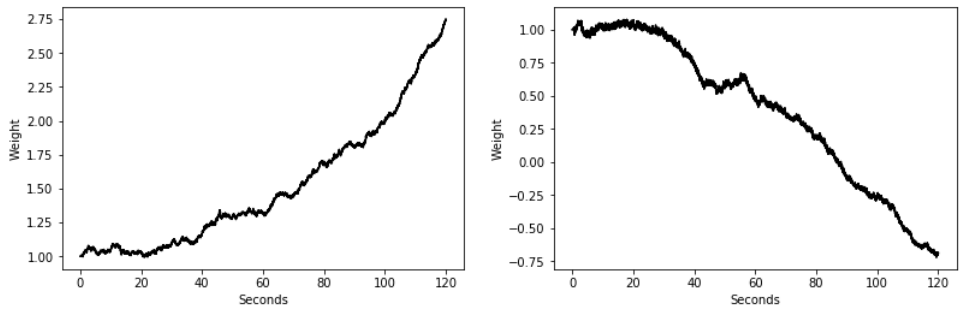


Figure 5.4: Example weight trajectories for the chosen parameter values specified in table 5.2, with $b_1 = -2$ (left) and $b_2 = 1$ (right).

Figure 5.5 shows two example cases with other parameter combinations, that produced less interesting weight trajectories. The top row shows the weight trajectory for the parameter values $b_1 = -1$, $b_2 = -1$ and $\omega^0 = 1$ (to the right), and the corresponding spike rate, $\mu_2^t | s_1^{t-1} = 1$, for neuron 2 in the case where neuron 1 fired in the previous time step (to the left). Here the weight gets so high that the increase in spike rate flattens out. The second row shows a case with $b_1 = -3$, $b_2 = -3$ and $\omega^0 = 0$. Now the weight change over time is too little to be any interesting for the purpose of this work.

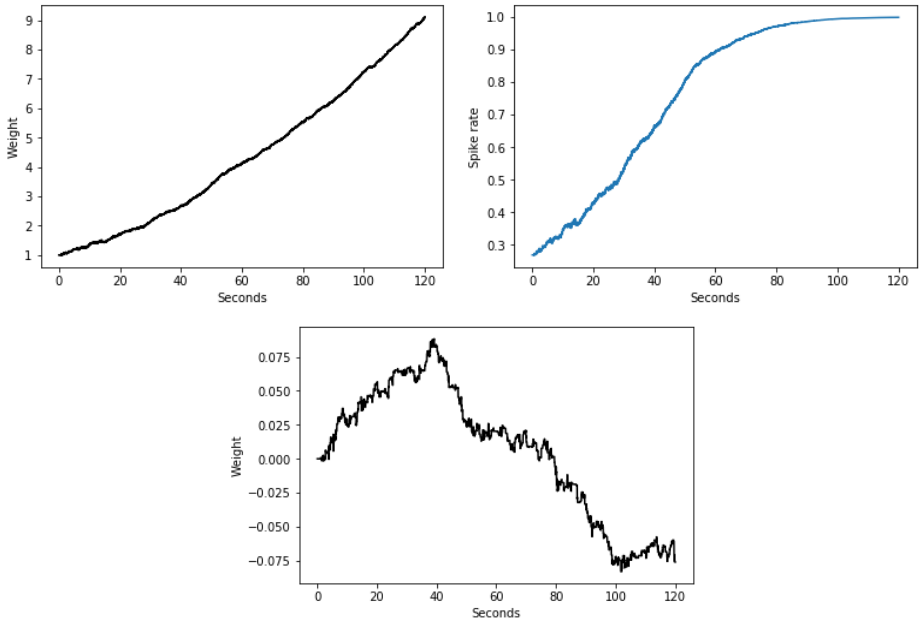


Figure 5.5: Black trajectories : Example weight trajectories for some combinations of parameters $\{b_1, b_2, \omega^0\}$ that do not appear reasonable. Top: $\{-1, -1, 1\}$. Bottom: $\{-3, -3, 0\}$. Blue trajectory: Spike rate, $\mu_2^t | s_1^{t-1} = 1$, for parameters $\{-1, -1, 1\}$.

5.3.2 Particle Metropolis-Hastings for inferring learning rule parameters

Having the simulated data, the aim is to test if the particle Metropolis-Hastings method can be used to infer the learning rule parameters in the generative model. For real spike measurements the only information available is the time points for spiking of the neurons. This means that in addition to learning rule parameters and the weight trajectories, also the background rates are unknown and therefore have to be inferred from the data.

5.3.2.1 Inferring learning rule parameters

The learning rule parameters are the main target in the procedure, and are sampled with the particle Metropolis-Hastings sampler. One approach could be to sample all the parameters $\{A_+, A_-, \tau_+, \tau_-\}$, and characterize the four individual distributions. However, the A_+ and A_- are typically closely correlated, and so are the values τ_+ and τ_-. Therefore, it was used fixed relationships $A_+ = 1.05 \cdot A_-$ and $\tau_+ = \tau_-$ in this work. These are the same relations as used in (Sen Song, 2000). So, in the implemented sampling routine, only values for A_+ and τ_+ was sampled, and the values for A_- and τ_- was adjusted accordingly. For the rest of the report, the notation A and τ without subscripts, refers to A_+ and τ_+ respectively.

In the experiments we run some cases where only one of the two learning rule parameters is inferred while the other is fixed to its true value, and some experiments where both are considered unknown and are inferred in the procedure. In the latter case, proposal values A and τ are drawn independently from their respective proposal distribution (see paragraph 5.3.2.3), and then the pair of proposed values is either accepted or rejected via the usual M-H procedure.

5.3.2.2 Inferring b_1, b_2 and ω^0

The likelihood of the spikes given the parameters, which is included in the posterior distribution, is dependent on the background rates b_1 and b_2. This is calculated in the particle filtering step, and produces an approximate distribution of the weight trajectory, $p(\omega^{1:T}|s_2^{1:T}, \theta)$. The particle trajectory is generated based on the learning rule and the spiking activity, with an exception of the first step at $t = 0$, which is drawn from the initial distribution $\pi(\omega^0)$. This is chosen to be a gaussian distribution around some pre-estimate $\widehat{\omega^0}$. Even though the values b_1, b_2 and ω^0 are not so interesting in themselves, they are essential for the inference. Therefore, they also have to be approximated from the spike data.

The spike rate of neuron 1 is only dependent on the value of the b_1 parameter. This means that the maximum likelihood estimate of b_1 can be computed directly from the spike data of neuron 1 as

$$\widehat{b_1} = \log\Big(\frac{\hat{\mu_1}}{1-\hat{\mu_1}}\Big), \tag{5.23}$$

with $\hat{\mu_1} = \frac{\sum_{t=1}^{T} s_1^t}{T}$.

The spike rate of neuron 2 is dependent on the linear predictor $\omega^t \cdot s_1^{t-1} + b_2$ at time t. In other words it depends on b_2, but also on the whole unknown trajectory $\omega^{1:T}$. For approximating b_2 this linear predictor was simplified with a stationary weight parameter, which would reflect the average weight along the time line. For this system, a Newton iteration routine can be performed, as described in section 3.2, targeting b_2 and this stationary weight parameter. Running this for 1000 replicates of data simulations, with parameters, gives the distribution of estimated b_2-values shown in figure 5.6. The empirical 95% credible interval for a symmetrical sample is calculated as the 2.5 and 97.5 percentiles of the sample (Geof H. Givens). For the sample of b_2 values this gave the credible interval $[-2.04, -1.96]$

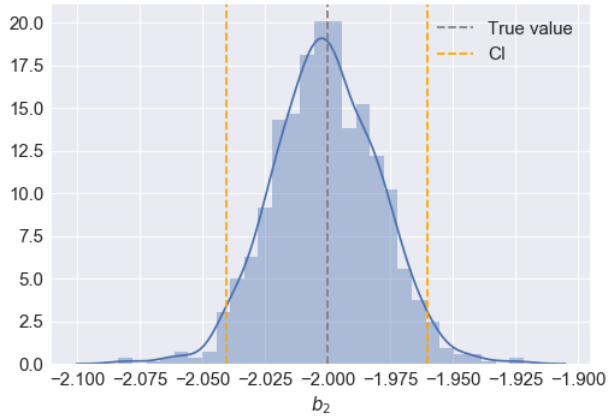


Figure 5.6: Distribution of b_2 values calculated from 1000 data simulations

These simulations also give estimates for the stationary weight parameter. However, since the weight changes across the data set, a better approximation of ω^0 can be achieved by running the procedure for only the first ten seconds of data. Figure 5.7 shows the distribution of this estimate based on 1000 simulations.

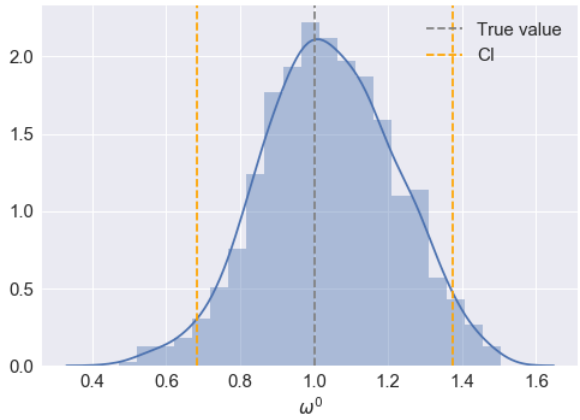


Figure 5.7: Distribution of ω_0 values calculated from 1000 data simulations

5.3.2.3 Proposal distributions

Every iteration in the particle Metropolis-Hastings samples begins with proposing values for the learning rule parameters. In analogy to the prior distribution over the learning rule parameters (see section 5.2.3), a gamma distribution was chosen also for the proposal, to ensure that negative values are not proposed. For a gamma distribution, the mean and variance are given by

$$\text{Mean} = \frac{\alpha}{\beta} \qquad \text{Variance} = \frac{\alpha}{\beta^2}, \tag{5.24}$$

where α and β are the shape and rate parameters, respectively.

We opted for a local proposal distribution, in the form of a random walk, so the mean value was set to the current parameter value in each iteration by adjusting the β-value at a fixed α-value. The variance in the proposal was updated every 100 iterations, according to the empirical variance in the 100 latest sampled values, as described in section 4.2.2.1. The choice of 100 was inspired by suggested values for the memory parameter and update frequency in (Haario and Saksman, 1998). At these updates, both α and β was adjusted to satisfy the equations.

5.3.3 Performing particle filtering

At every iteration of the Metropolis Hastings procedure, the posterior distribution of the weights given the current values of the learning rule parameters was characterized using particle filtering. A set of 100 particles was initiated by drawing values according to the initial distribution $\pi(\omega^0)$. The particles were evolved sequentially according to the importance sampling distribution including gaussian noise of a specified size, producing weight trajectories growing in time. Like in the generative model, the size of each time step is set to 5 ms also here. Figure 5.8 is a visualization of the sample of particles after propagating for 20, 60 and 120 seconds, along with the trajectory in the generative model in black. This is from a particle filtering with hyperparameters set to the same as in the generative model, and without any resampling.

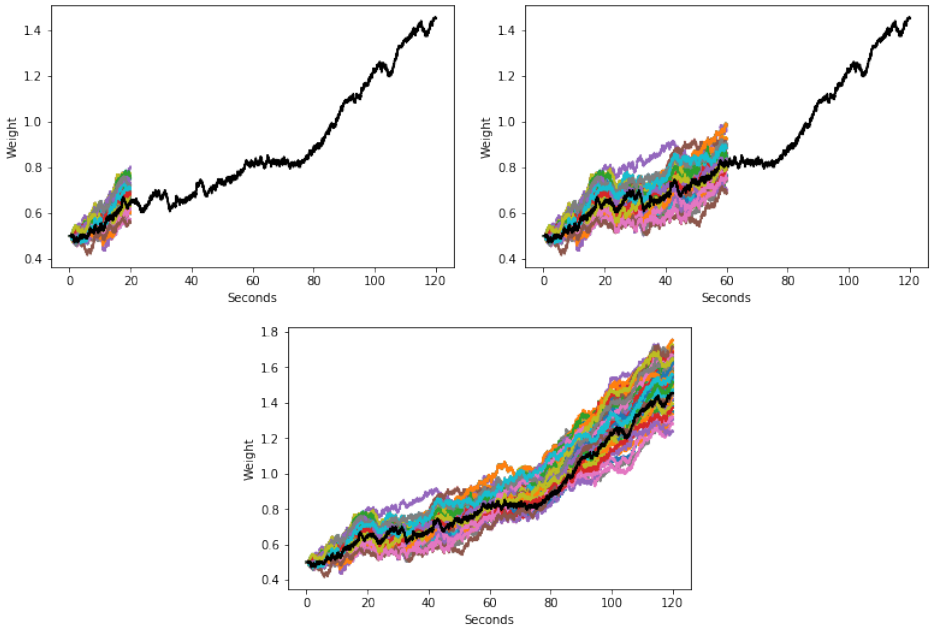


Figure 5.8: Visualization of particle propagation at times 20s, 60s and 120s, and weight trajectory in generative model (black)

The particle weights are reflecting the relative probability of the particles given the observed spike data. In figure 5.9 the same particle distribution as in figure 5.8 are shown with transparency corresponding to their relative particle weights.

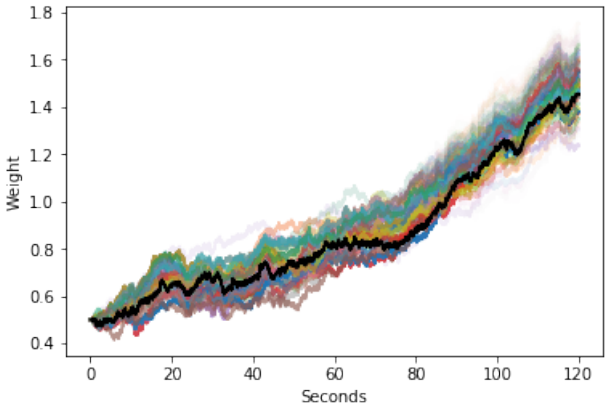


Figure 5.9: Particle distribution with transparency corresponding to the particle weights, and weight trajectory in generative model (black)

As expected, the particle trajectories positioned furthest away from the generative trajectory have lower particle weights than the ones close to the generative trajectory.

For every time step, the preplexity in the particle distribution is computed. If the preplexity goes below 66, the resampling mechanism is performed, as explained in section 4.3.3. Figure 5.10 shows an example of how the particle distribution can look before and after a

resamplig.

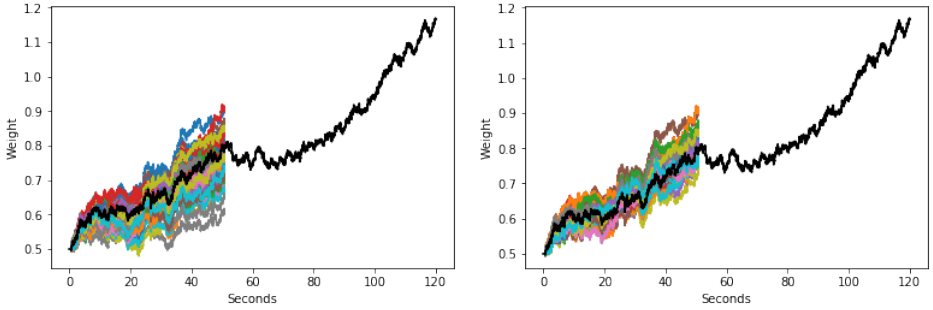


Figure 5.10: Particle distribution before (left) and after (right) resampling

Figure 5.11 shows an example of how the generated particle distribution looks after setting the true value $A_+ = 0.005$ compared to setting the wrong value $A_+ = 0.002$.

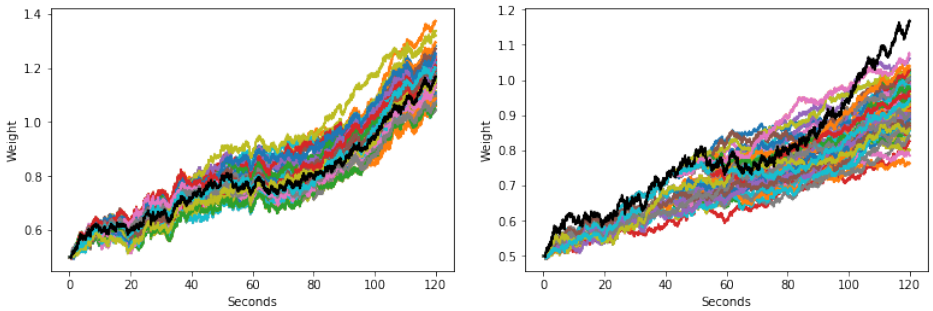


Figure 5.11: Particle distribution generated with parameters equal to those in the generative model (left), and with $A_+ = 0.002$ (right)

Chapter 6

Results

In order to have full control over the inference algorithm, the method presented in the previous chapters was implemented in Python, with no use of any packages for particle MCMC or particle filtering (see Python script in appendix) In this chapter we present the results from this procedure. The main focuses are to assess whether the pMCMC method is appropriate for inference on spike data, and to investigate properties of the learning rule for the system under consideration. Section 6.1 shows the posterior values across the range of A and τ separately, as calculated by the particle filtering. This is to visualize the output from running particle filtering, and give an indication of how the posterior behaves. The next two sections include results from running the particle Metropolis-Hastings. In section 6.2, A and τ are inferred individually, keeping the other fixed to its true value, with the aim of evaluating the performance of the method itself, and to understand the effects of the two parameters separately. Section 6.3 shows the results of treating both learning rule parameters as unknown, which reflects the situation for real data, and will therefore give indications on the performance of the algorithm on the data from future lab experiments, assuming that the models is close enough to the reality.

Before presenting the results, we want to make a point clear for the reader. One of the main targets for investigation will be to compare the accuracy of our inferences for different noise levels. The noise level is given as the value of the standard deviation in the gaussian noise (in equation 5.8), and will take the values $\sigma = \{0.0001, 0.0005, 0.001, 0.002, 0.005\}$. To get an intuition on the size of these noise levels, notice that a characteristic size of the learning rule steps is the value of the A-parameter. Since $A = 0.005$ is the typical value used in these experiments, the noise level $\sigma = 0.005$ corresponds to a signal to noise ratio of 1.

6.1 Performance of particle filtering

How easily the particle Metropolis-Hastings method can characterize the learning rule parameters depend on the shape of the posterior distribution. For example it is interesting to know if the posterior is convex, and how peaked it is. To get an idea of its nature, the posterior over a range of values of the learning rule parameters was characterized. 250 simulated data sets with equal parameter values, including the values $A = 0.005$ and $\tau = 20ms$, were generated. Then the particle filtering procedure was ran for each of these data sets, after setting the parameters A and τ to values within the prescribed range. Particles and weights for each combination of pair of parameters and data set, allowed to compute the likelihood of the

pair given the data set. Figure 6.1 shows the computed log likelihood for A-values ranging from 0.001 to 0.01 for each of the 250 simulated data sets. τ was fixed to its true value in the particle filtering. The figure shows that there is a convex trend across the range of A-values, with a maximum in the area around the true value for every data set.

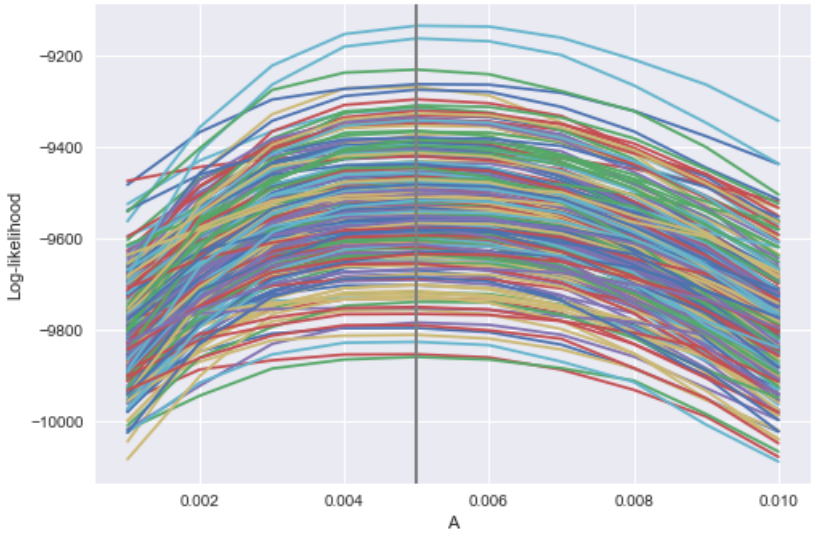


Figure 6.1: The log likelihood computed with particle filtering, for a range of A-values for 250 individual simulated data sets of 120 seconds spike data, with $b_1 = b_2 = -2$, $\omega^0 = 1$ and $\sigma = 0.0005$. Grey line corresponds to the true A-value.

In the Metropolis-Hastings procedure, the sampled A- and τ-values are evaluated according to the posterior, not only the likelihood. The following plots show the mean and standard deviation of the log posterior over the 250 replicates. The prior distribution, $p(\theta)$ was set as described in section (section 5.2.3) and the likelihoods, $p(s_2^{1:T}|\theta)$ are computed with particle filtering. In order to get the posterior distribution $p(\theta|s_2^{1:T})$ for each data set, the normalization constant, $p(s_2^{1:T})$, was approximated by Laplace approximation of the integral over θ of the joint distribution $p(s_2^{1:T}, \theta) = p(s_2^{1:T}|\theta) * p(\theta)$ (MacKay (2002)). Figure 6.2 display the average across data sets of the resulting estimate of the posterior for noise levels $\sigma = 0.0001$, $\sigma = 0.0005$ and $\sigma = 0.005$. All curves attain a maximum at the true value $A = 0.005$. For the two lower noise levels, the curves are clearly peaked, and the variance on the top is very low. This indicates that most replicates had maximum values at $A = 0.005$. For $\sigma = 0.005$ the curve is clearly flatter. Notice that the scale on the y-axis varies among the plots.

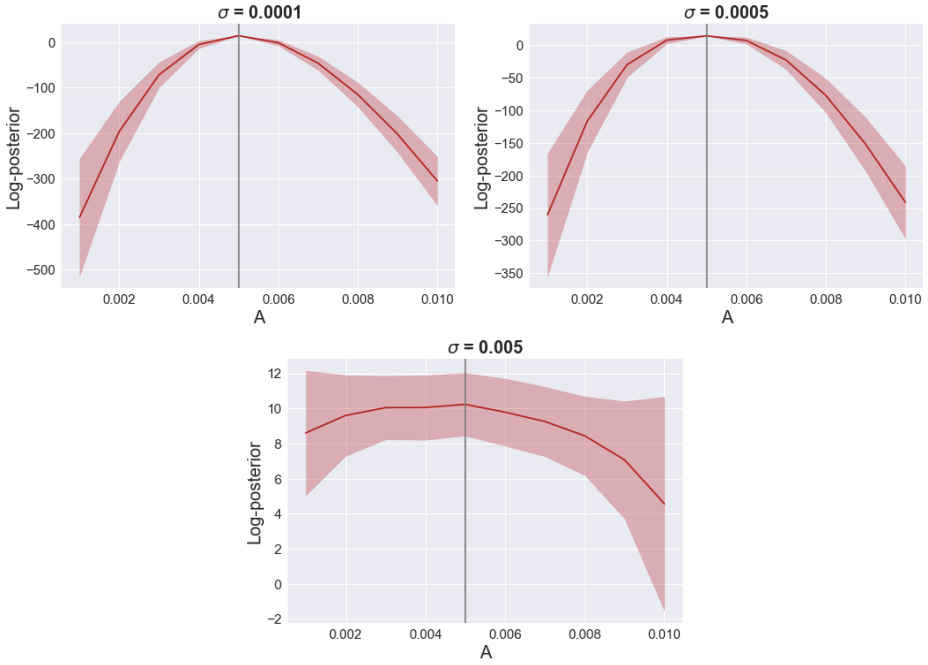


Figure 6.2: Log-posterior value for a range of A-values for the noise levels $\sigma = 0.0001$ (top left), $\sigma = 0.0005$ (top right) and $\sigma = 0.005$ (bottom). Grey line corresponds to the true A-value.

Figure 6.3 shows the same curve for varying τ-values, with noise levels $\sigma = 0.0001$ and $\sigma = 0.001$. In this case A was kept fixed to 0.005 in the particle filtering. Figure 6.4 shows the same, but with parameter $b_1 = 1$. A larger b1 correspond to an higher average firing rate for neuron 1, which for real data can be achieved via stimulation. Noticeably, the presence of stimulation strongly affects the accuracy of our inference algorithm: without stimulation, all values from $10ms$ to $60ms$ are relatively similar in terms of posterior density, whereas with stimulation there is a clear peak at $\tau = 20ms$ for the noise levels $\sigma = 0.0001$ and $\sigma = 0.0005$.

Another observation is that the exact value of A seems more critical than that of τ. In all plots with A, we observe that there is a definite maximum point at $A = 0.005$. For τ in the stimulation case, the curve is flat on the top already for a noise level of $\sigma = 0.0005$.

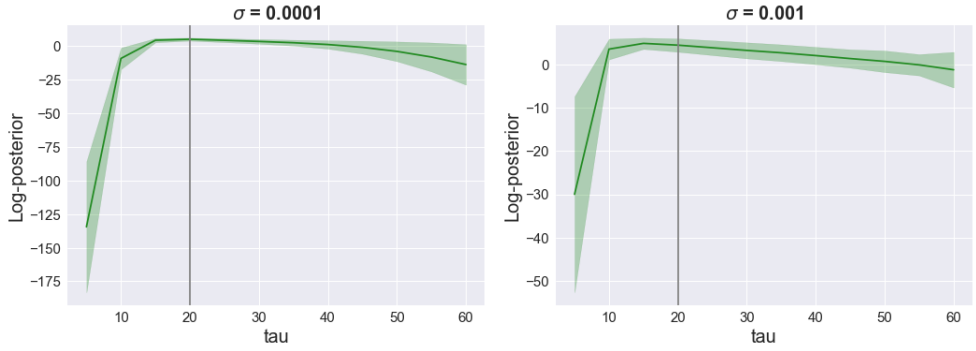


Figure 6.3: Log-posterior value over a range of τ-values for the noise levels $\sigma = 0.0001$ (left) and $\sigma = 0.001$ (right), for simulations of 120 seconds spike data, with $b_1 = b_2 = -2$ and $\omega^0 = 1$. Grey line corresponds to the true τ-value.

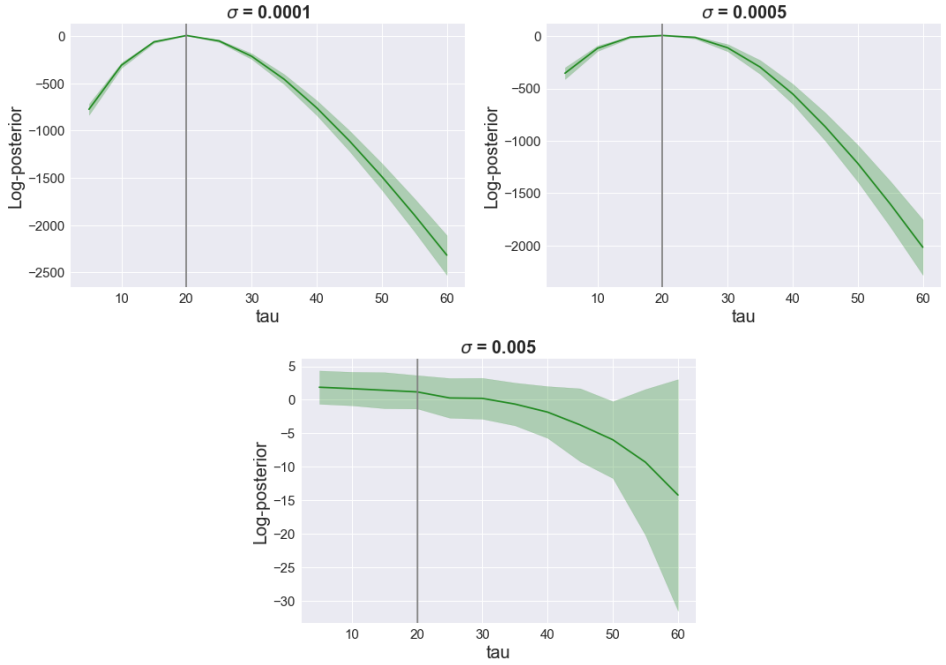


Figure 6.4: Log-posterior value for a range of τ-values for the noise levels $\sigma = 0.0001$ (top left), $\sigma = 0.0005$ (top right) and $\sigma = 0.005$ (bottom), for simulations of 120 seconds spike data, with $b_2 = -2$, $\omega^0 = 1$, and stimulation $b_1 = 1$. Grey line corresponds to the true τ-value.

6.2 Distributions of individual parameters PMCMC

Now we investigate the results of particle Metropolis-Hastings inference, targeting A and τ individually. In this section we demonstrate that the method converges from wrong starting value towards a distribution around the generative one, we show the effect of the noise parameter and length of the spike data set, and we present visualizations of the posterior distribution of the two learning rule parameter.

6.2.1 Inference of A paramter

By initializing the Metropolis Hastings algorithm with a value of A lying outside the typical set of its distribution, the Markov chain should manage to navigate the parameter space towards the typical set and, after a certain number of iterations, be independent of the starting position. Figure 6.5 shows two examples of the accepted A-values in 1500 iterations of the algorithm, starting at $A = 0.002$, for noise levels $\sigma = 0.0001$ and $\sigma = 0.001$ respectively. Notice that in both cases the algorithm manages to move away from the wrong starting value to approach a neighbourhood of the generative value $A = 0.005$. We observed that for all the data sets we simulated, and the different initializations that we explored, after 300 iterations the samples seemed to be independent of the starting value. Therefore, the burn-in period was set to 300 iterations.

Figure 6.5: Accepted samples A-values over 1500 iterations, starting from A=0.002, for noise levels $\sigma = 0.0001$ and $\sigma = 0.001$.

How well the unknown parameters can be inferred from data will typically depend on the size of the data set. A bigger data set indeed typically contains more information on the parameters, and will intuitively give a more peaked posterior distribution. To compare the width of the posterior distribution for different data lengths, pMCMC was performed on five replicates of simulated data sets for each of the lengths $\{30s, 60s, ..., 180s\}$, and synaptic plasticity noise levels of $\sigma = 0.0005$. Figure 6.6 shows the mean and the mean of the sample standard deviation of the posterior distribution across the 5 instances. The latter clearly decreases with data length, supporting the idea that the posterior gets more peaked with more data.

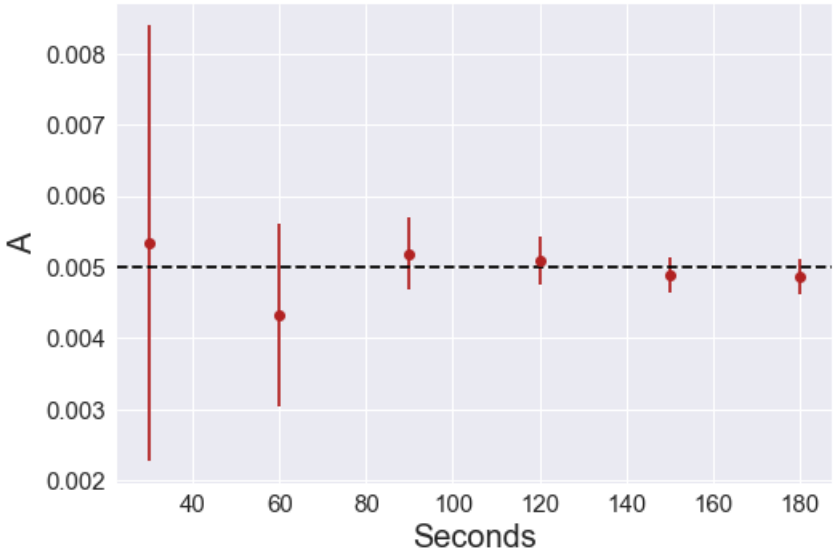


Figure 6.6: Mean of the sample means of the posterior distribution of A in five replicates, with error bars corresponding to mean of the sample standard deviations, for different data lengths. Black line indicate the generative value.

Also, it is interesting to see how the shape of the posterior is affected by the noise level. For this purpose we start by presenting some visualizations of the posterior distribution as sampled with pMCMC. Figure 6.7 shows the histograms of the sampled A-values, after the burn-in period, for noise levels $\sigma = \{0.0001, 0.0005, 0.001, 0.002, 0.005\}$. Each plot corresponds to one individual data simulation, and a particle MCMC run with 6000 iterations. Since the problem is stochastic, different runs with identical parameters would produce slightly different results. However, even though the plots are based on single runs, they capture the essence in how the distributions appear, and how they change with noise level.

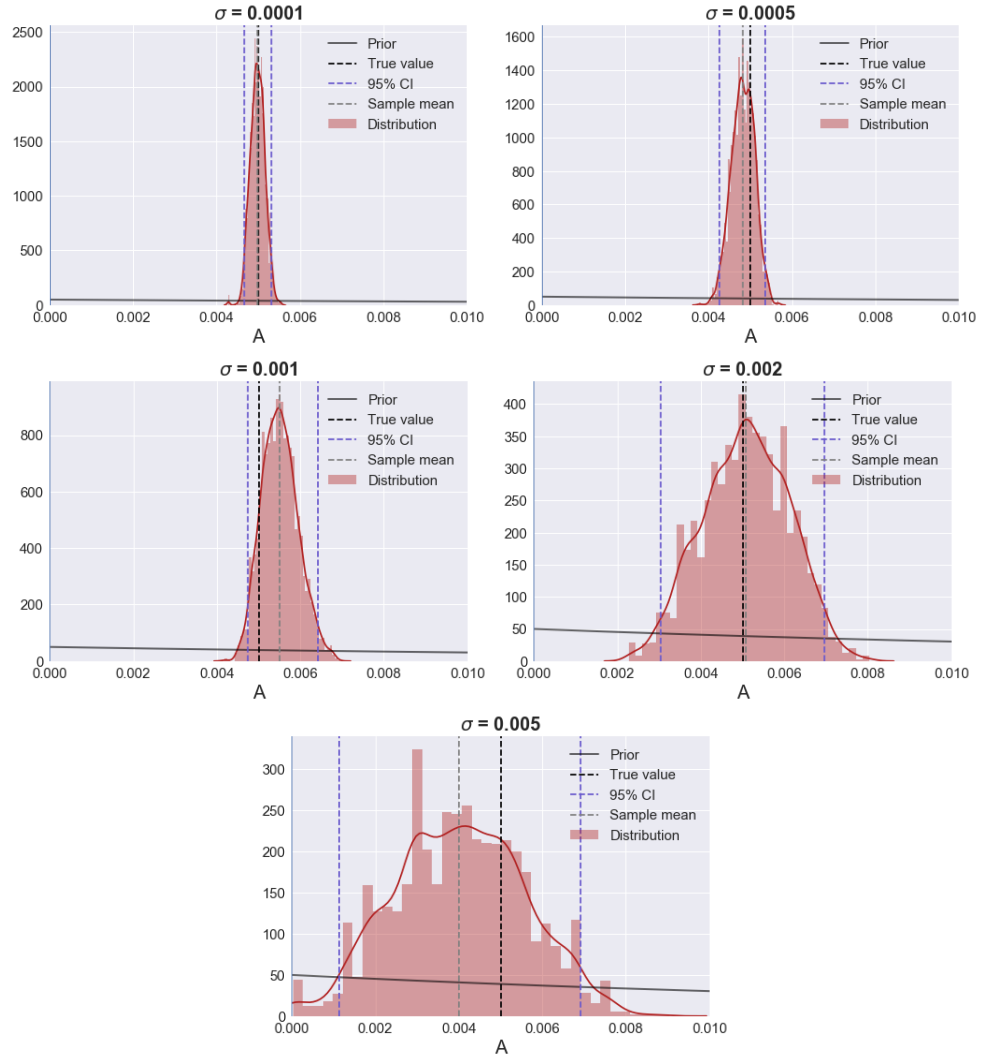


Figure 6.7: Posterior density of A as sampled by particle Metropolis-Hastings, for various noise levels; $\sigma = 0.0001$ (top left), $\sigma = 0.0005$ (top right), $\sigma = 0.001$ (middle left), $\sigma = 0.002$ (middle right) and $\sigma = 0.005$ (bottom). Included in the plots are the sample mean (grey dashed), true value (black dashed) boundaries of the 95% empirical credible intervals (purple dashed), and the prior density function (black).

From these plots we observe that the generative value falls within the credible interval in all cases, and that the prior seems to play a minor role in shaping the posterior. In accordance with the results of section 6.1, the width of the distribution increases with increasing noise level. For $\sigma = 0.0001$ the credible interval is $[0.0047, 0.0053]$, meaning that the learning rule can be characterized quite precisely, whereas for $\sigma = 0.005$ the credible interval is $[0.0031, 0.0070]$.

Figure 6.8 shows the trend of increasing posterior width for increasing noise more systematically, averaging over ten replicates of the procedure. As in figure 6.6, it displays the mean of the sample means, along with errorbars corresponding to the mean of the sample standard deviations.

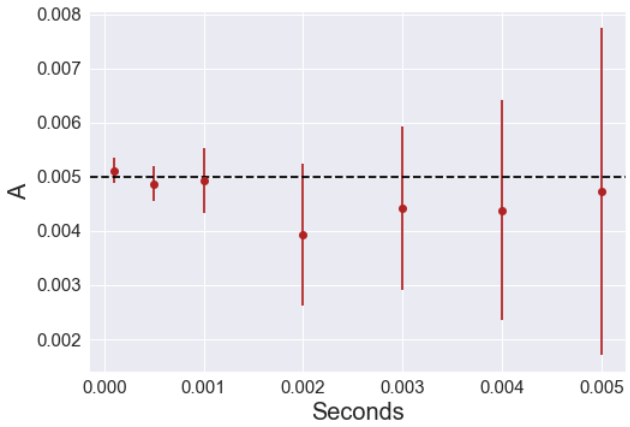


Figure 6.8: Mean of the sample means in ten replicates, with error bars corresponding to mean of the sample standard deviations, for different noise levels. Black line indicate the generative value.

6.2.2 Inference of τ-parameter

As for the parameter A in section 6.2.1, we conducted an analysis of the performance of pMCMC for, while A was kept fixed. As with A, we checked that the method converges when staring from a wrong input value (results not reported here). As already noted, figure 6.3 and 6.4 indicate that the posterior for τ behaves differently when neuron 1 was stimulated compared to when it was not. Therefore, it is interesting to consider the result from particle MCMC for both settings.

Figure 6.9 plots show the posterior distribution for τ for noise level $\sigma = 0.0005$ and $\sigma = 0.005$, for the case without stimulation.

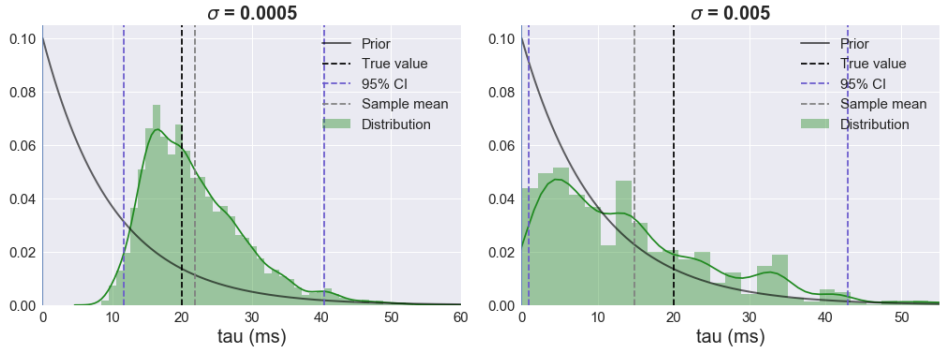


Figure 6.9: Posterior density of τ as sampled by particle Metropolis-Hastings, for $b_1 = -2$ and noise levels $\sigma = 0.0005$ and $\sigma = 0.005$.

Notice that for both high and low noise level the 95% credible interval includes the generative value $\tau = 20ms$. With $\sigma = 0.0005$ the empirical 95% credible interval is $[11.6, 40.4]$, which means that the generative value is hard to determine precisely, but that the sample distribution is still affected by the data. For $\sigma = 0.005$ it seems like the inference starts to collapse, and that the prior starts to dominate.

Analogously to figure 6.9, figure 6.10 shows the sample distribution of τ at different noise levels, here in the case with stimulation ($b_1 = 1$).

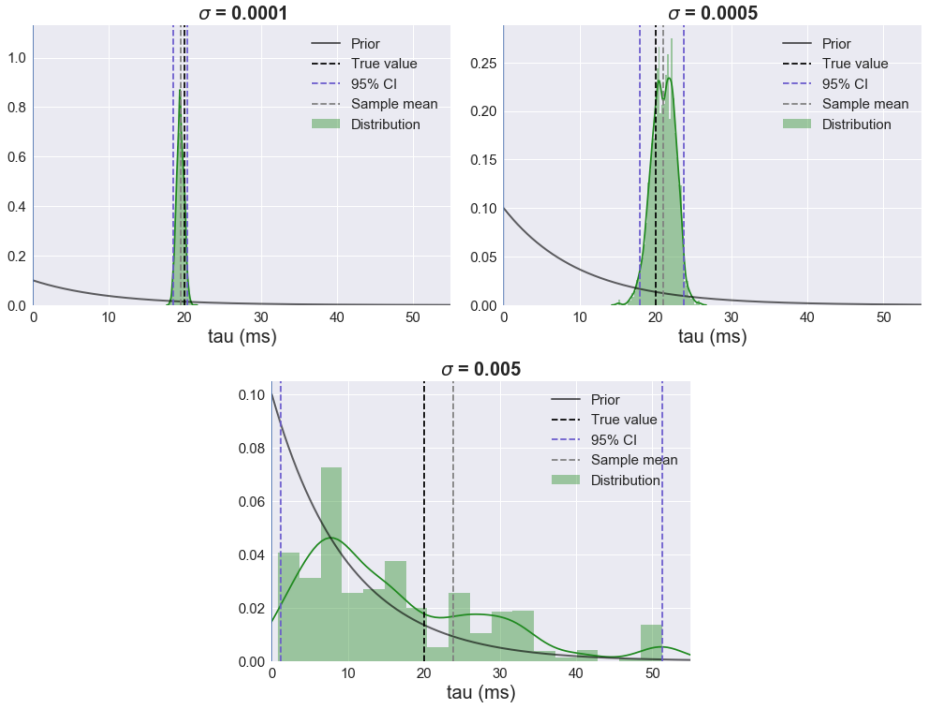


Figure 6.10: Posterior density of τ as sampled by particle Metropolis-Hastings, for $b_1 = 1$ and noise levels $\sigma = 0.0001$, $\sigma = 0.0005$ and $\sigma = 0.005$.

Also in the presence of stimulation the method collapses for $\sigma = 0.005$, which corresponds to an high signal to noise ratio. This agrees with the finding for the same noise level in figure 6.4. However, for noise level $\sigma = 0.0001$ and $\sigma = 0.0005$ the sample distribution is relatively narrow around the true parameter value.

A more systematic assessment of the role of noise in the pMCMC inference accuracy was performed by running ten replicates with different noise level, for the case with (right) and without (left) stimulation. The result is displayed in figure 6.11. As in figure 6.8 the figure shows the mean of the sample means, with errorbars corresponding to the mean of the sample standard deviations.

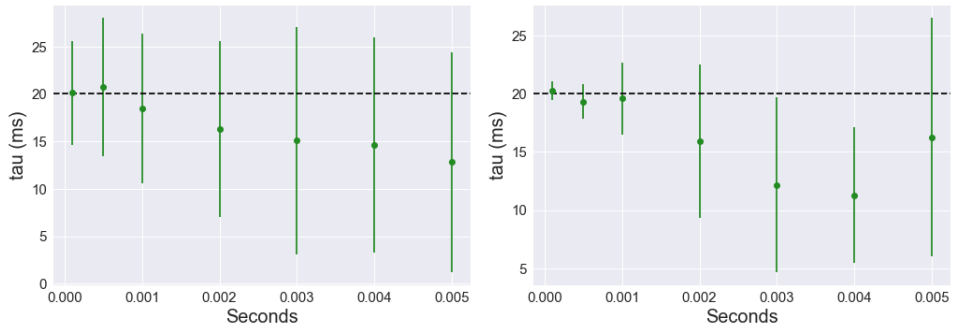


Figure 6.11: Mean of the sample means in ten replicates, with error bars corresponding to mean of the sample standard deviations, for different noise levels, for $b_1 = -2$ (left) and $b_1 = 1$ (right). Black line indicate the generative value.

This clearly shows that with stimulation, the distribution in narrower for small noise levels, whereas without stimulation, the distribution is relatively wide for all noise levels.

Analogously to figure 6.6, figure 6.12 and 6.12 display the mean and standard deviation of the sample posterior for different lengths of simulated data sets, without and with stimulation respectively. At each data length it is shown the average over five replicates, with noise level equal to $\sigma = 0.0005$. Notice that the scale on the y-axis is different in the two, meaning that the general sample standard deviation is smaller in the case with stimulation. In both figures, the sample standard deviation is larger at the shortest data set, corresponding to $30s$. Despite this observation, there is little variations across the time line. So we only have a slight evidence of time dependency of the standard deviation. This is different from what we would expect, so one continuation of the project would be to investigate this more thoroughly.

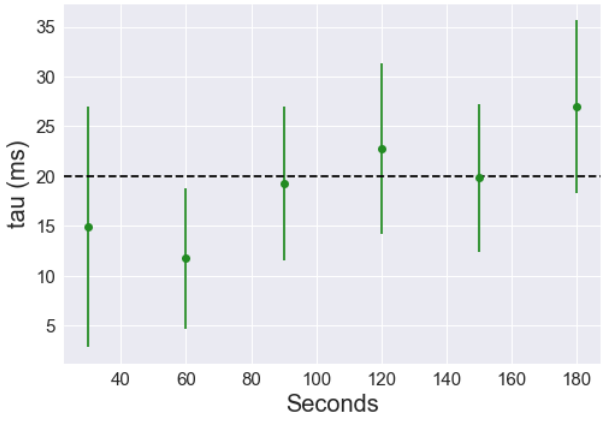


Figure 6.12: Mean of the sample means of the posterior distribution of τ without stimulation in five replicates, with error bars corresponding to mean of the sample standard deviations, for different data lengths. Black line indicate the generative value.

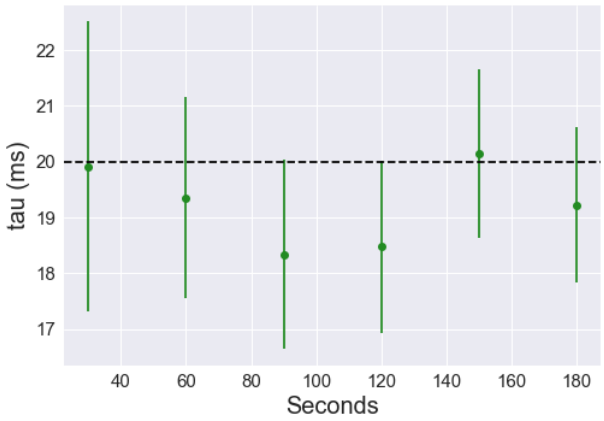


Figure 6.13: Mean of the sample means of the posterior distribution of τ with stimulation in five replicates, with error bars corresponding to mean of the sample standard deviations, for different data lengths. Black line indicate the generative value.

6.3 Simultaneous inference

In this section we present the results of inferring both A and τ from the data. Now both A and τ are drawn according to their respective proposal distributions, and either both are accepted or both are rejected in each iteration. It turns out that the marginal sample distributions obtained do not give much information of the generative parameter values in this case. Figure 6.14 shows an example of how these marginal distributions can look.

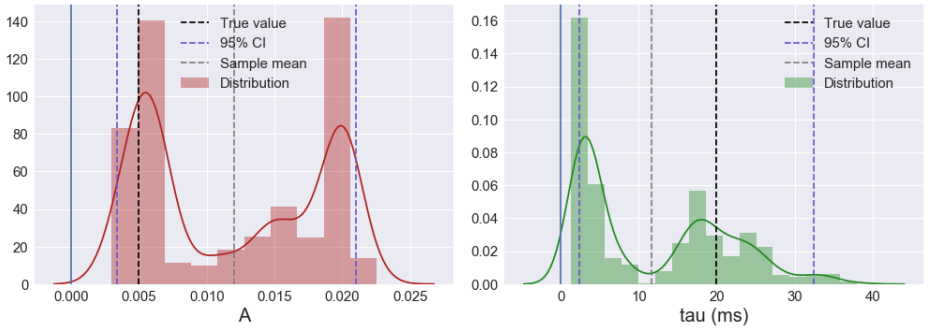


Figure 6.14: Marginal sample densities of A (left) and τ (right), for joint sampling with particle Metropolis-Hastings, for noise level $\sigma = 0.0005$.

Notice that the shape of the learning rule is determined by the combination of A and τ. So even though the marginal distributions do not provide so much information, it is interesting to see the joint structure of the sample values. Figure 6.15 shows the same sampled values as in figure 6.14, displayed in the A, τ-plane. Interestingly, the four modes of the marginals in Figure 6.12 corresponded to two modes of the join distribution of the parameters A and τ. The sampled parameters are not only correlated at the modes though.

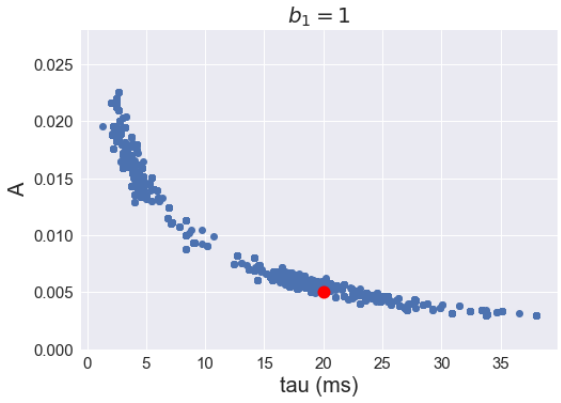


Figure 6.15: $\{A, \tau\}$ sample points obtained from particle Metropolis-Hastings, for $b_1 = 1$ and noise level $\sigma = 0.0005$. The red dot correspond to the generative parameters.

This shows that the sample has a characteristic shape, allowing only specific combinations of the two parameters. Figure 6.16 shows the same plot for a case where there is no stimulation of neuron, with $b_1 = -2$.

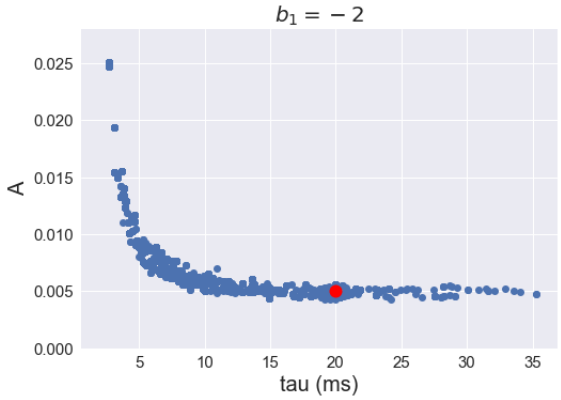


Figure 6.16: $\{A, \tau\}$ sample points obtained from particle Metropolis-Hastings, for $b_1 = -2$ and noise level $\sigma = 0.0005$. The red dot correspond to the generative parameters.

In the absence of stimulation the curve delineated by the sample is much flatter in τ in a neighbourhood of the generative value $A = 0.005$ with respect to the case with stimulation.. These characteristics agrees with what was observed for the τ-parameter in figure 6.3 and 6.4.

Figure 6.17 shows how the sample looks for varying values of the noise parameter. The thickness of the typical set increases with the noise level, in line with the results of section 6.1 and 6.2, further hindering our inference capability in this setting.

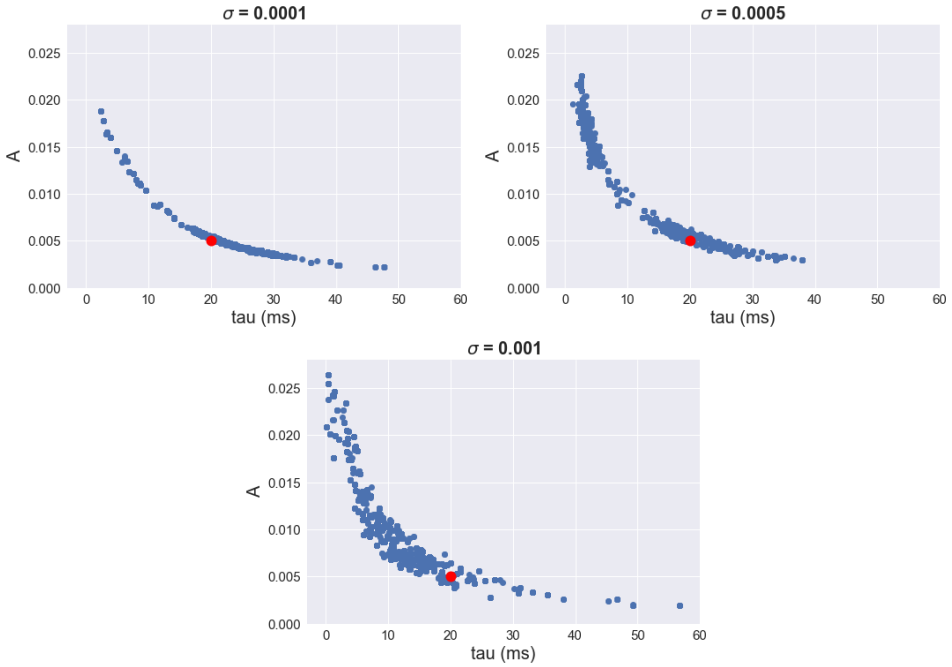


Figure 6.17: $\{A, \tau\}$ sample points obtained from particle Metropolis-Hastings, for $b_1 = 1$ and noise levels $\sigma = 0.0001$ (top left), $\sigma = 0.0005$ (top right) and $\sigma = 0.001$ (bottom). The red dot correspond to the generative parameters.

It is also interesting to see whether there are differences in terms of posterior value across the sample, that may help resolve the generative parameters. In figure 6.18 the sample points are color coded according to their relative log posterior values. The normalization constant would be the same for all sample points, and is not taken into account here.

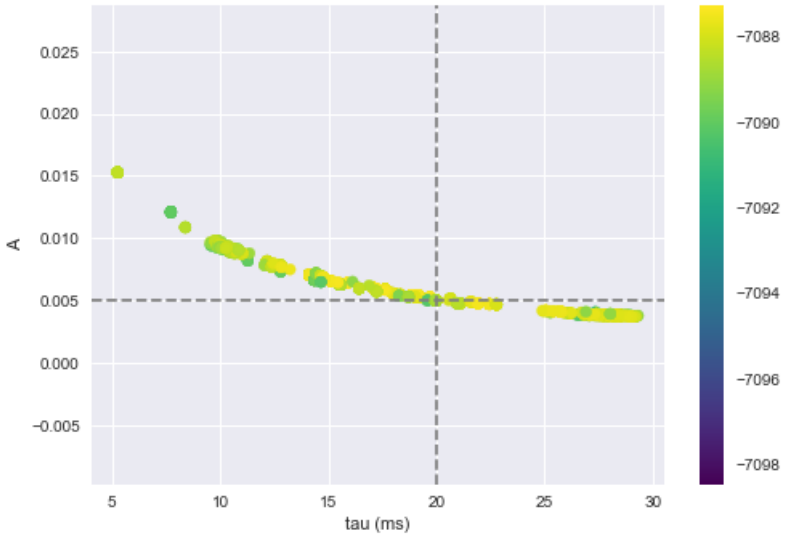


Figure 6.18: $\{A, \tau\}$ sample points obtained from particle Metropolis-Hastings, for $b_1 = 1$ and noise level $\sigma = 0.0001$, color coded according to the relative (not normalized) log posterior values.

Unfortunately, it seems like there is no trend in where the posterior is higher or lower. Thus, the posterior cannot be used for determining the generative values of the learning rule parameters more precisely. However, there are clearly many combinations of parameters that are not sampled. So it could still be possible to distinguish between different shapes of learning rules from this method. To test this hypothesis, data sets with other combinations of learning rule parameters was also generated. The combinations tried were $\{A = 0.004, \tau = 10ms\}$ and $\{A = 0.007, \tau = 30ms\}$, along with the parameter values used earlier, $\{A = 0.005, \tau = 20ms\}$. The left plot in figure 6.19 shows the resulting samples for these three different parameter combinations look. The right plot in figure 6.19 shows the shape of the learning rule for four different sample values from each of the clusters. These are chosen to be one from each end of each cluster, and two in between.

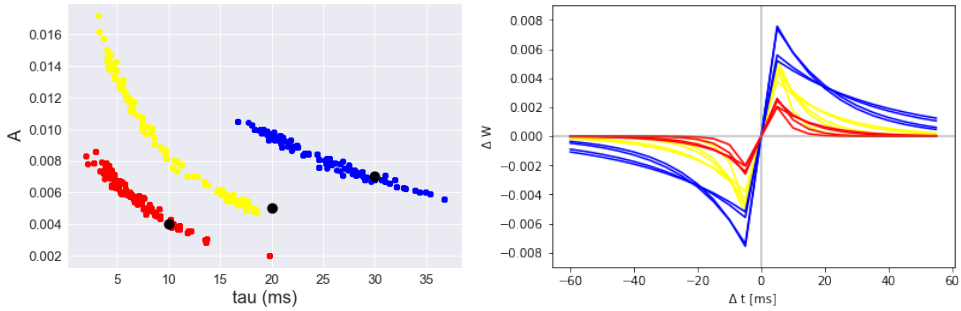


Figure 6.19: Left plot: $\{A, \tau\}$ sample points obtained from particle Metropolis-Hastings, for $b_1 = 1$ and noise levels $\sigma = 0.0001$, for generative values $\{A = 0.004, \tau = 10ms\}$ (red), $\{A = 0.005, \tau = 20ms\}$ (yellow), and $\{A = 0.005, \tau = 20ms\}$. The black dots indicate the generative parameter for the nearest cluster of samples. Right plot: Shape of deterministic part of the learning rule for four different combinations of parameters from the different clusters, colored according to the corresponding cluster. The learning rules are downsampled at every $\Delta t = 5ms$ to match the time scale of the neural activity.

This shows that combinations of learning rule parameters belonging to the same sample, corresponds to learning rules of similar shapes. Ultimately this suggests that the nature of the learning rule for the model in this work does not allow us to determine which specific learning rule parameter was used, but could give an indication of what kind of underlying learning rule that is present. Notice that despite the similarity of the learning rules within the clusters of Figure 6.19, these are not identical, not even when binned at the time bin size of the neural activity. It follows that the inability of the algorithm to infer the generative parameters can't be simply attributed to an identifiability issue of our learning rule model. It's rather likely an interplay between learning rule parametrization, neural activity and synaptic plasticity time scales, and finite sampling which prevent our algorithm to resolve from specific combinations of parameters.

Chapter 7

Conclusion and further work

This thesis has covered an exploration of the particle Metropolis-Hastings method, for making inference of the STDP learning rule influencing the activity of two connected neurons. This is an approach for studying synaptic plasticity that was suggested in (Linderman et al. (2014)). In contrast to the model used by Linderman, which was a Poisson spiking model, we modeled the neural spiking as a Bernoulli process for a binned time line. Spike data was simulated according to this model, for various specifications of model parameters, data lengths and noise levels, in order to test the abilities and limitations of the inference method.

The main target of the experimental tests was to characterize the posterior distribution of the two learning rule parameters A and τ. In chapter 6 the results of these tests were presented. The experiments show that the method manages to retrieve information on the generative learning rule parameters from the spike data, given enough data and sufficiently low noise level. Less noise and more data gave a sharper sample posterior distribution of both learning rule parameters. For the τ-parameter (keeping A fixed) it was shown that the spike rate of the presynaptic neuron affects the performance of inference. At the base spike rate, the posterior value was approximately flat for a range of τ-values around the generative one. However, by increasing the spike rate of the presynaptic neuron, to mimic a case where the spiking is driven by high frequency external stimuli, the behavior was different. This gave a posterior with bell-shaped structure around the generative value, for low enough noise levels. Ultimately the presence of stimulation, particle Metropolis Hastings could accurately infer each model parameter at signal to noise ratios below 0.4, on $120ms$ recordings of the activity of pre- and post-synaptic neurons. Starting from this anecdotal report it would be interesting to systematically investigate the role of stimulation on pMCMC inferences, which could potentially instruct optimal stimulation protocols when it comes to real data recordings.

When sampling A and τ simultaneously, the sample marginal posterior densities gave limited information of the underlying parameter values. However, it was demonstrated that the sampled $\{A, \tau\}$-values are highly correlated, tracing a characteristic curve in the $\{A, \tau\}$-space. It was shown that the location of this curve was different for sufficiently different learning rules. This supports the idea that some information of the underlying learning rule can be detected with the method, regardless of the specific identity of the learning rule parameters. As a continuation of the work in this thesis, it would be interesting to compare how well various learning rules fit to a set of spike data, by implementing Bayesian model selection (MacKay, 2002). This could for example be applied to the spike data from healthy rats and rats with Alzheimer's, to investigate whether a difference in plasticity effects can be detected.

For continuation of this work there are several considerations and modifications that can be made, in order to optimise the method and to make the model more biologically plausible. In the following we discuss some of these. One important methodical consideration that was not included here, was an evaluation of the number of particles in the particle filtering. More particles would give a better approximate of target distribution, but on the other hand make the problem more computationally demanding. This trade-off should be investigated. In addition, it would be interesting to test the sensitivity of the results to the used prior and to the bin size of the time line.

In order to improve inference, a bigger data set would be beneficial. As discussed in section 5.3.1, the experimental setup used in this work only allowed for spike data of lengths up to a couple of minutes, primarily due to the unboundedness of the learning rule. One could imagine that the neurons can be stimulated in a way such that the weight trajectory would change direction now and then avoiding a monotone increase or decrease. In that way, longer data sets could be produced, perhaps giving a better inference. Also, the amount of data could be increased by inclusion of spike data for more neurons, assuming that the synapses share the same learning rule parameters. However, an optimisation of the code could be needed, since the number of synapses scales as the square of the number of neurons, meaning computationally cost of the problem increases quickly.

There are also some expansions to the model that would be interesting. The model considered in this work was a simplified system, that would potentially miss some crucial aspects of synaptic transmission, plasticity and neural physiology in general of real neurons. For example, just after firing an action potential, a neuron undergoes a refractory period, corresponding to membrane hyperpolarization and therefore reduced likeliness of another spike. Also, when a neuron is exposed to constant stimuli over time, the responsiveness of the neuron will typically decrease, a phenomenon known as *adaptation*. These effects can by modeled by including a self coupling of the neurons, such that their activity would also depend on their own spike history.

Another model component that would be interesting to investigate further, is the learning rule itself. The additive STDP learning rule used in this work is simple and monotone, and well suited for the approach in this thesis. However, there exist other, more flexible and complex learning rules, that has other benefits. For example, in (Ghanbari et al., 2017) it is considered the Tsodyks-Markram model, which includes parameters for neural facilitation and depression. Finally an interesting avenue for investigation is the effect of common input, which often arises when working with real data. By common input we mean that the activity of the observed neurons is affected by activity of unobserved neurons and as such our inferences on the underlying observed system. E.g in (Dunn and Battistin (2017)) is was shows that if the weights in the system are too strong, it would be problematic to reconstruct even stationary connectivity of observed neurons if the connectivity in unobserved neurons is unknown.

Overall there are many possible expansions to the model, and several aspects yet to be investigated in order to optimise the method for its purpose. Despite this fact, findings in this work supports the idea of using particle Metropolis-Hastings when aiming for inference of the underlying learning rule in neural spike data.

Bibliography

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

Benedikt Zott, Manuel M. Simon, A.K., 2019. A vicious cycle of amyloid-dependent neuronal hyperactivation.

Cappé, O., Douc, R., Guillin, A., Marin, J.M., Robert, C.P., 2008. Adaptive importance sampling in general mixture classes. Statistics and Computing 18, 447–459.

Chen, Z., Brown, E.N., 2013. State space model. Scholarpedia 8, 30868. doi:10.4249/scholarpedia.30868. revision #189565.

Costa, R., Sjostrom, P.J., van Rossum, M., 2013. Probabilistic inference of short-term synaptic plasticity in neocortical microcircuits. Frontiers in Computational Neuroscience 7, 75.

Dahlin, J., Schön, T.B., 2015. Getting started with particle metropolis-hastings for inference in nonlinear dynamical models. arXiv:1511.01707.

Doucet, A., de Freitas, N., Gordon, N., 2001. An Introduction to Sequential Monte Carlo Methods. Springer New York, New York, NY. pp. 3–14.

Doucet, A., Johansen, A., 2009. A tutorial on particle filtering and smoothing: Fifteen years later. Handbook of Nonlinear Filtering 12.

Dunn, B., Battistin, C., 2017. The appropriateness of ignorance in the inverse kinetic ising model. Journal of Physics A: Mathematical and Theoretical 50, 124002. doi:10.1088/1751-8121/aa59dc.

Geof H. Givens, J.A.H., . John Wiley Sons, Ltd.

Geof H. Givens, J.A.H., 2013. Simulation and Monte Carlo Integration. John Wiley Sons, Ltd. chapter 6. pp. 151–199.

Ghanbari, A., Malyshev, A., Volgushev, M., Stevenson, I., 2017. Estimating short-term synaptic plasticity from pre- and postsynaptic spiking. PLOS Computational Biology 13, e1005738.

Gomez-Isla, T., 1996. Profound loss of layer ii entorhinal cortex neurons occurs in very mild alzheimer's disease. The Journal of Neuroscience 16, 4491–4500.

Haario, H., Saksman, E., 1998. Adaptive proposal distribution for random walk metropolis algorithm. Computational Statistics 14.

Hanssen, K.S., 2019. Establishing lateral entorhinal cortex layer II-neuron cultures as a tool for the investigation of early Alzheimer's disease related changes. Master's thesis. NTNU.

Hebb, D., 1949. Organization of Behavior. Wiley Sons.

Linderman, S., Stock, C., Adams, R., 2014. A framework for studying synaptic plasticity with neural spike train data. Advances in Neural Information Processing Systems 3.

Ludwig Fahrmeir, Thomas Kneib, S.L.B.M., 2013. Regression (Models, Methods and Applications). Springer, Berlin, Heidelberg.

MacKay, D.J.C., 2002. Information Theory, Inference Learning Algorithms. Cambridge University Press, USA.

Martino, L., Elvira, V., Louzada, F., 2017. Effective sample size for importance sampling based on discrepancy measures. Signal Processing 131, 386–401.

Naesseth, C.A., Lindsten, F., Schön, T.B., 2019. Elements of sequential monte carlo. arXiv:1903.04797.

Purves, D., 2011. Neuroscience, 5th edition. Sinauer Associates Inc.,U.S.

Radde R, Bolmont T, K.S., 2006. Abeta42-driven cerebral amyloidosis in transgenic mice reveals early and robust pathology. EMBO Reports 7, 940–946.

Robert, C.P., Casella, G., 2005. Monte Carlo Statistical Methods (Springer Texts in Statistics). Springer-Verlag, Berlin, Heidelberg.

Roberts, G.O., Gelman, A., Gilks, W.R., 1997. Weak convergence and optimal scaling of random walk metropolis algorithms. The Annals of Applied Probability 7, 110–120.

Robinson, B.S., Song, D., Berger, T.W., 2014. Generalized volterra kernel model identification of spike-timing-dependent plasticity from simulated spiking activity, in: 2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp. 6585–6588.

Sen Song, Kenneth D. Miller, L.F.A., 2000. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. Nature neuroscience , 919–926.

Stevenson, I., Koerding, K., 2011. Inferring spike-timing-dependent plasticity from spike train data, in: Shawe-Taylor, J., Zemel, R.S., Bartlett, P.L., Pereira, F., Weinberger, K.Q. (Eds.), Advances in Neural Information Processing Systems 24. Curran Associates, Inc., pp. 2582–2590.

Witter, M., 2011. Entorhinal cortex. Scholarpedia 6, 4380. doi:10.4249/scholarpedia.4380. revision #137315.

Appendix

Python script for particle Metropolis-Hastings

```
import sys, os
import numpy.random
import numpy as np
import random
import time
from scipy import *
import scipy.stats
from scipy.stats import gamma
from scipy.stats import norm
import csv

def learning_rule(t,S1_t, S2_t,S1,S2, A_p, A_m, tau_p,tau_m):
#STDP learning rule
    l_p = np.sum(A_p*np.exp(-((t-S1_t)*1000)/(tau_p*t_per_second)))
    l_m = np.sum(A_m*np.exp(-((t-S2_t)*1000)/(tau_m*t_per_second)))
    return(S2[t]*l_p-S1[t]*l_m)

def generate(T, b1, b2, A_p_true,A_m_true,tau_true, W_0_true, std):
#Generate spikes and weights
    W_T = np.zeros(T)

    S1 = np.zeros(T)
    S2 = np.zeros(T)
    S1_t_init = 1
    S2_t_init = 1

    S2_t = np.array([0])

    lambda1 = 1/(1+np.exp(-b1))
    S1[0] = np.random.binomial(1,lambda1,1)
    if S1[0] == 1:
        S1_t = np.array([0])
        S1_t_init = 0
    W_T[0] = W_0_true

    for t in range(1,T):
        lambda2 = 1/(1+np.exp(-(b2 + (W_T[t-1]*S1[t-1]))))

        S1[t] = np.random.binomial(1,lambda1,1)
        S2[t] = np.random.binomial(1,lambda2,1)

        if S1[t] == 1:
            if S1_t_init:
                S1_t = np.array([t])
                S1_t_init = 0
            else:
                S1_t = np.append(S1_t, t)
        if S2[t] == 1:
            if S2_t_init:
                S2_t = np.array([t])
                S2_t_init = 0
            else:
                S2_t = np.append(S2_t, t)

        if t<t_constant:
            W_T[t] = W_T[t-1]

        else:
            W_T[t] = W_T[t-1] + (learning_rule(t,S1_t, S2_t,S1,S2,A_p_true,A_m_true,tau_true,tau_true)) +
np.random.normal(0,std)

    return(W_T, S1, S2, S1_t, S2_t)

def learning_rule_steps(S1, S2, S1_t, S2_t, t_per_second, A, tau, T):
#Compute deterministic learning rule steps given data and hyperparameters
```

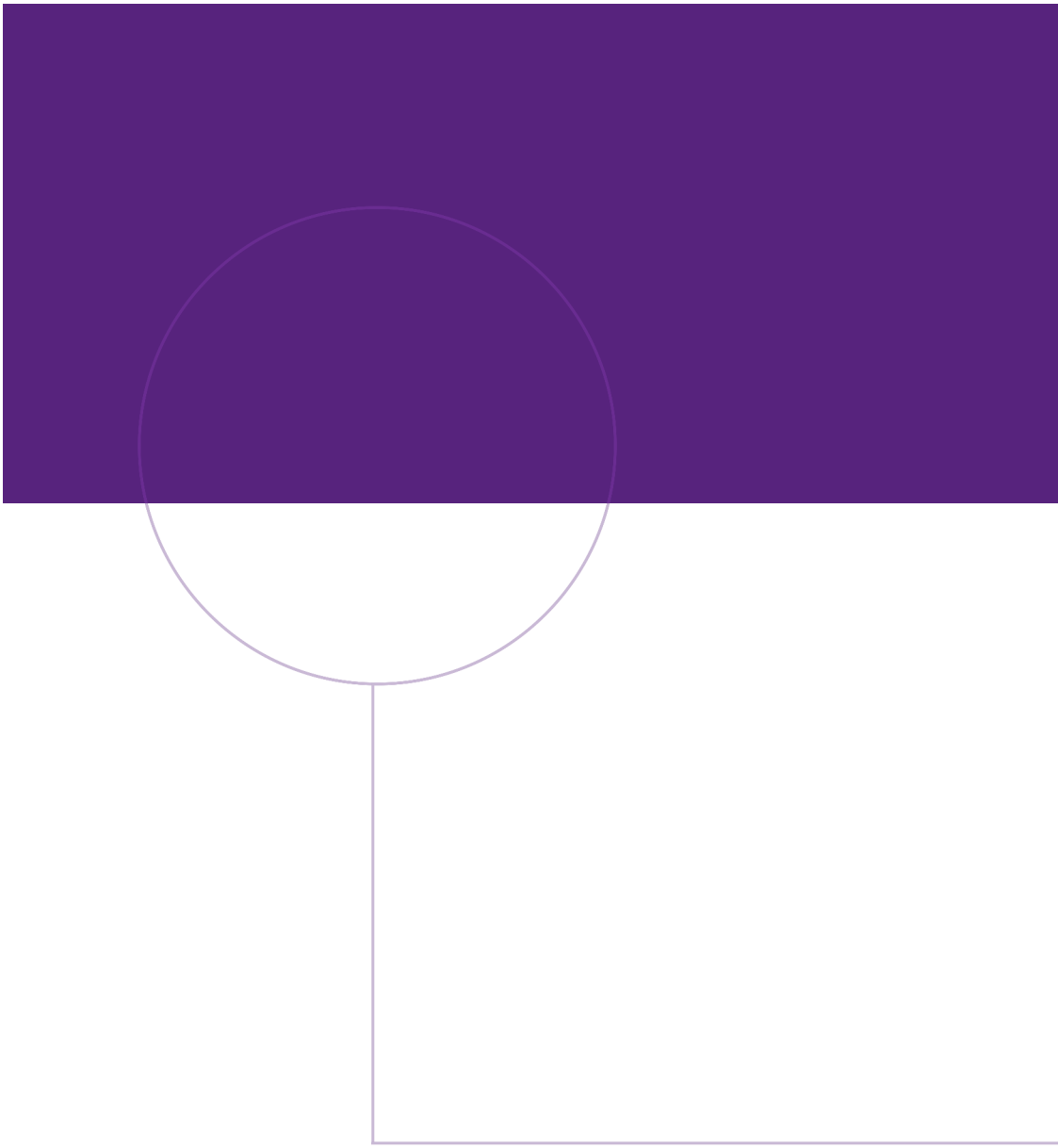
```
    t = t_constant
    step_list = [learning_rule(t,S1_t[0:int(sum(S1[0:t+1]))], S2_t[0:int(sum(S2[0:t+1]))],S1,S2,A,A*1.05,tau,tau)]
    while (t+1 < T):
        t = t+1
        step_list.append(learning_rule(t,S1_t[0:int(sum(S1[0:t+1]))], S2_t[0:int(sum(S2[0:t+1]))],S1,S2,A,A*1.05,tau,tau))
    return(step_list)



#Functions for particle filtering
def sample_weights(w_p, P, learning_step, std):
    w_p_t = np.zeros(P)
    for p in range(P):
        sample_point = np.random.normal(w_p[p][-1]+learning_step,std)
        w_p[p].append(sample_point)
        w_p_t[p] = sample_point
    return(w_p, w_p_t)

def particle_log_likelihoods(S1, S2, w_p_t, t, P, b1, b2):
    log_alpha_t = S2[t+1]*(b2+(w_p_t*S1[t])) - np.log(1+np.exp(b2+(w_p_t*S1[t])))
    return(log_alpha_t)

def update_particle_weights(v_p, P, log_alpha_t):
    alpha_t_scaled = exp(log_alpha_t-max(log_alpha_t))
    v_p = v_p*alpha_t_scaled
    return(v_p)

def resample_weights(w_p,v_p, P):
    xk = np.arange(P)
    v_sum = np.sum(v_p)
    pk = v_p/v_sum
    custm = scipy.stats.rv_discrete(name='custm', values=(xk, pk))
    p_r = list(custm.rvs(size=P))
    init = 1
    for it in p_r:
        if init==1:
            w_p_r = [w_p[it].copy()]
            init = 0
        w_p_r.append(w_p[it].copy())
    v_p = np.repeat(1/P, P)
    return(w_p_r, v_p)

def N_eff(v_p):
    v_sum = np.sum(v_p)
    v_norm = v_p/v_sum
    squared_sum = 0
    for v in v_norm:
        squared_sum = squared_sum + v**2
    return(1/squared_sum)

def perplexity(v_p):
    v_sum = np.sum(v_p)
    v_norm = v_p/v_sum
    v_norm_log = np.log(v_norm)
    perplexity = exp(- sum(v_norm*v_norm_log))
    return(perplexity)

def log_lik_total(w_p, S1, S2,P, b1, b2):
    log_lik = np.zeros(P)
    for p in range(P):
        log_lik[p] = np.sum(S2[1:]*(b2+(w_p[p][:]*S1[:-1])) - np.log(1+np.exp(b2+(w_p[p][:]*S1[:-1]))))
    return(log_lik)



def SMC(P, T, W_0, std_initial, S1, S2, learning_step, b1, b2):
```

```
#Particle filtering
    resampling_list = [0]

    w_p = [[W_0]]
    for p in range(P):
        w_p.append([W_0])

    v_p = np.ones(P)

    it = 0

    for t in range(1,T-1):
        if t<t_constant:
            for p in range(P):
                w_p[p].append(w_p[p][-1])
        else:
            w_p, w_p_t = sample_weights(w_p, P, learning_step[it], std)
            alpha_t = particle_log_likelihoods(S1,S2,w_p_t,t,P, b1, b2)
            v_p = update_particle_weights(v_p, P, alpha_t)
            it = it+1
            if perplexity(v_p)<N_threshold:
                w_p, v_p = resample_weights(w_p,v_p,P)
                resampling_list.append(t)
    return(w_p, v_p)

def estimate_W_and_b2(S1, S2, w_guess, b2_guess, iterations):
#Newton iterations

    par = [b2_guess,w_guess]
    par_list = [par]
    for j in range(iterations):
        lin_pred = par[0] + (par[1]*S1)


        lambda_it = 1/(1+exp(-(lin_pred[:-1])))

        arr_score = [np.ones(len(S1)-1),S1[:-1]]*(S2[1:] - (exp(lin_pred[:-1])/(1 + exp(lin_pred[:-1]))))

        score = arr_score.sum(axis=1)
        score = np.matrix(score)

        for t in range(len(S1)-1):
            if t == 0:
                hessian = np.matrix([[1,S1[t]],[S1[t],S1[t]]])*lambda_it[t]*(1-lambda_it[t])
            else:
                hessian = hessian + np.matrix([[1,S1[t]],[S1[t],S1[t]]])*lambda_it[t]*(1-lambda_it[t])


        m = numpy.linalg.inv(hessian)*(numpy.matrix.transpose(score))
        m = np.array(m)
        m = np.ndarray.flatten(m)

        par = par + m

        par_list.append(par)
    return(par_list[-1])

def estimate_b1(S1):
    lambda1_estim = sum(S1)/(len(S1))
    b1_estim = np.log(lambda1_estim/(1-lambda1_estim))
    return(b1_estim)

def adaptive_alpha(H, A_p_list):
    A_slice = A_p_list[len(A_p_list)-H:]
    m = mean(A_slice)
    v = var(A_slice)*(2.4**2)
```

```
    new_alpha = (m**2)/v
    return (new_alpha)

def adaptive_alpha_tau(H, tau_list):
    tau_slice = tau_list[len(tau_list)-H:]
    tau_slice_2 = [x/1000 for x in tau_slice]
    m = mean(tau_slice_2)
    v = var(tau_slice_2)*(2.4**2)
    new_alpha = (m**2)/v
    return (new_alpha)

def proposal_A_p(A_p, alpha):
    return(np.random.gamma(alpha, A_p/alpha))

def proposal_tau(tau, alpha):
    return(1000*np.random.gamma(alpha, (tau/1000)/alpha))

def A_p_prior(A_p, A_p_rate):
    return(gamma.pdf(A_p, a = 1, scale = 1/A_p_rate))

def tau_prior(tau, tau_rate):
    return(gamma.pdf(tau/1000, a = 1, scale = 1/tau_rate))


def p_spike_train_fraction(w_p, w_p_new, S1, S2, P, b1, b2):
    init = 1
    for p in range(P):
        log_lik_temp_new = np.sum(S2[1:]*(b2+(w_p_new[p][:]*S1[:-1])) - np.log(1+np.exp(b2+(w_p_new[p][:]*S1[:-1]))))
        log_lik_temp = np.sum(S2[1:]*(b2+(w_p[p][:]*S1[:-1])) - np.log(1+np.exp(b2+(w_p[p][:]*S1[:-1]))))
        if init:
            init = 0
            log_lik_list = [log_lik_temp]
            log_lik_list_new = [log_lik_temp_new]
        else:
            log_lik_list.append(log_lik_temp)
            log_lik_list_new.append(log_lik_temp_new)

    min1 = min(log_lik_list)
    min2 = min(log_lik_list_new)
    min_tot = min(min1,min2)

    log_lik_list_scaled = [x-min_tot for x in log_lik_list]
    log_lik_list_new_scaled = [y - min_tot for y in log_lik_list_new]

    p_old = sum([exp(a) for a in log_lik_list_scaled])
    p_new = sum([exp(b) for b in log_lik_list_new_scaled])

    return(p_new/p_old)

def particle_marginal_Metropolis_Hastings_both(A_p_start,tau_start, W_0, b1, b2, iterations,alpha, alpha_tau):

    alpha_new = alpha
    alpha_tau_new = alpha_tau
    tau = tau_start
    A_p = A_p_start
    learning_step = learning_rule_steps(S1, S2, S1_t, S2_t, t_per_second, A_p, tau, T)
    w_p, v_p = SMC(P, T, W_0, std_initial, S1, S2, learning_step, b1, b2)

    A_list = [A_p]
    tau_list = [tau]
    c = 0

    for i in range(iterations):

        if (i%100)==0:
            if i>150:
```

```
            alpha_new = adaptive_alpha(H, A_list)

        if (i%100)==0:
            if i>150:
                alpha_tau_new = adaptive_alpha_tau(H, tau_list)


        A_p_new = proposal_A_p(A_p, alpha_new)
        tau_new = proposal_tau(tau, alpha_tau_new)
        learning_step_new = learning_rule_steps(S1, S2, S1_t, S2_t, t_per_second, A_p_new, tau_new, T)
        w_p_new, v_p_new = SMC(P, T, W_0, std_initial, S1, S2, learning_step_new, b1, b2)

        w_p_new, v_p_new = resample_weights(w_p_new,v_p_new,P)

        p_s_ratio = p_spike_train_fraction(w_p, w_p_new, S1, S2, P, b1, b2)
        prior_new_A = A_p_prior(A_p_new, A_p_rate)
        prior_old_A = A_p_prior(A_p, A_p_rate)
        prior_ratio_A = prior_new_A/prior_old_A

        prior_new_tau = tau_prior(tau_new, tau_rate)
        prior_old_tau = tau_prior(tau, tau_rate)
        prior_ratio_tau = prior_new_tau/prior_old_tau

        ratio = p_s_ratio*prior_ratio_A*prior_ratio_tau*(gamma.pdf(A_p, a = 1, scale = A_p_new)/gamma.pdf(A_p_new, a = 1,
scale = A_p))*(gamma.pdf(tau/1000, a = 1, scale = tau_new/1000)/gamma.pdf(tau_new/1000, a = 1, scale = tau/1000))

        if (np.random.uniform(0,1)<ratio):
            c = c+1
            tau = tau_new
            A_p = A_p_new
            w_p = w_p_new
            v_p = v_p_new
            learning_step = learning_step_new

        print(A_p, tau)

        A_list.append(A_p)
        tau_list.append(tau)

    return(A_list,tau_list, c)

#Parameters
W_0_true = 1

b1_true = 2
b2_true = -2

A_p_true= 0.005
A_m_true = A_p_true*1.05
tau_true = 20

n_seconds = 120
t_per_second = 200
T = n_seconds * t_per_second
t_constant = 100


std = 0.0001  #Noise level

#Start shape parameters for gamma proposal
alpha = 5
alpha_tau = 4

#Rate parameters for gamma proposal
A_p_rate = 50
tau_rate = 100
```

```
H = 100

P = 100 #Number of particles

N_threshold = P*0.66 #perplexity threshold

iterations = 1500

A_p_start = A_p_true
tau_start = tau_true

std_initial = 0.001
w_guess = W_0_true
b2_guess = b2_true

#Main

#Generate spikes and weight trajectory
W_T, S1, S2, S1_t, S2_t = generate(T, b1_true, b2_true, A_p_true,A_m_true,tau_true, W_0_true, std)

#Estimate b2
b2 = estimate_W_and_b2(S1, S2, w_guess, b2_guess, 40)[0]

#Estimate b1
b1 = estimate_b1(S1)

#Estimate W0
W_0 = estimate_W_and_b2(S1[:2000], S2[:2000], w_guess, b2, 40)[1]

#Particle Metropolis-Hastings
(A_list,tau_list,c)  = particle_marginal_Metropolis_Hastings_both(A_p_start,tau_start, W_0, b1, b2, iterations,alpha, alpha_tau)
```



NTNU

Norwegian University of
Science and Technology