Simulation-Based Inference for Parameter Identification in Mechanistic Models of Neural Dynamics

by

Nicolai Haug

Thesis

for the degree of

Master of Science

Department of Informatics

Faculty of Mathematics and Natural Sciences

University of Oslo

November 2021

This master's thesis is submitted under the master's program *Computational Science*, with program option *Imaging and Biomedical Computing*, at the Department of Informatics, University of Oslo. The scope of the thesis is 60 credits.

www.duo.uio.no

Print production: Reprosentralen, University of Oslo

Abstract

A central challenge in building a mechanistic model of neural dynamics is to identify the model parameters consistent with experimental data. Due to intractable likelihoods, traditional methods in the toolkit of statistical inference are inaccessible for many mechanistic models. To overcome intractable likelihoods, simulation-based inference provides a framework for performing rigorous Bayesian inference by just requiring forward simulations of the model. The objective of this thesis is to investigate the viability of simulation-based inference, in particular approximate Bayesian computation (ABC) and neural density estimation (NDE), for identifying parameters in mechanistic models of neural dynamics. Specifically, we infer the conductance parameters in the Hodgkin-Huxley model for initiation and propagation of action potentials and the synaptic weight parameters in the Brunel network model for activity dynamics in local cortical networks.

We develop the generic Python library pyLFI which uses ABC with quantile-based rejection sampling and local linear regression adjustment for estimating the posterior distributions of model parameters. As the curse of dimensionality forces ABC to require a compression of data into low-level summary statistics, we use expert-crafted statistics of spiking activity. The choice of summary statistics is crucial, and we carry out a correlation analysis to select and weight summary statistics. On synthetic data, pyLFI efficiently estimates posterior distributions and recovers ground truth parameters. We vary tuning parameters, and find that, with regression adjustment, we can accept more simulations without sacrificing substantial accuracy for the model parameters that are the most constrained by the summary statistics. The approach of pyLFI is compared to the recent NDE algorithm Sequential Neural Posterior Estimation (SNPE), which trains an artificial neural network to map features of observed data to posteriors over parameters by using adaptively proposed model simulations. Inference on the Brunel network model demonstrates the power and flexibility of SNPE; by training on simulations that includes two of the network states, SNPE is able to accurately predict posteriors that correspond to the network's state in the observed data.

In conclusion, we find that simulation-based inference is a powerful tool that can be applied to a wide range of computational investigations in neuroscience, which may help to both bridge the gap between mechanistic hypotheses and experimental neural data, and design better models of neural dynamics. However, there are challenges (and opportunities) ahead in scaling and automating simulation-based inference approaches, and the methods are simulation intensive.

Acknowledgements

The present work was conducted at the Centre for Integrative Neuroplasticity (CINPLA) and the Department of Informatics at the University of Oslo, under the supervision of professor Gaute T. Einevoll, associate professor Joakim Sundnes, professor Aslak Tveito and Dr. Alexander Stasik.

First of all, I want to thank my excellent supervisors. To Alex, thank you for providing valuable guidance and insight in the planning and development of this thesis. To Aslak, my sincere gratitude for all the help with the formalities. To Joakim, I am deeply grateful for the support and also the constructive suggestions on the code. To Gaute, I want to give a special thanks for both introducing me to the fascinating field of computational neuroscience and your great support. I appreciate the great amount of freedom you gave me to explore the aspects that interested me the most. Our regular meetings have been of invaluable help. Thank you for taking the time.

I want to extend a thanks to the friendly group of fellow students, especially Kristian Wold and Christer Dreierstad, for the teamwork, inspiring conversations and good times. I also want to thank the members of CINPLA for creating such a friendly environment and pleasant place to be.

I would like to thank my friends and family, especially my parents for their unconditional support and love in my years as a student in physics and computational science. I would also like to express my gratitude towards my partner Oda Hovet for her endless support throughout my degree.

Nicolai Haug
Oslo, November 2021

Abbreviations

ABC	Approximate Bayesian Computation
AI	Asynchronous Irregular
ANN	Artificial Neural Network
AP	Action Potential
HDI	Highest Density Interval
HH	Hodgkin-Huxley
IF	Integrate-and-Fire
iid	Independent and Identically Distributed
KDE	Kernel Density Estimation
LIF	Leaky Integrate-and-Fire
LFI	Likelihood-Free Inference
MAF	Masked Autoregressive Flow
MAP	Maximum a Posteriori Probability
MCMC	Markov Chain Monte Carlo
MDN	Mixture-Density Network
NDE	Neural Density Estimation
NF	Normalizing Flows
pdf	Probability Density Function
PPC	Posterior Predictive Check
RMSPE	Root-Mean-Square Percentage Error
SBI	Simulation-Based Inference (alias LFI)
SEM	Standard Error of the Mean
SI	Synchronous Irregular
SNL	Sequential Neural Likelihood
SNPE	Sequential Neural Posterior Estimation
SR	Synchronous Regular

Contents

1 Introduction

The human brain contains billions of neurons, and each interact, by exchanging electrical signals, with thousands of other neurons to create countless circuits that, together with the nerves throughout our bodies, form our nervous system [1]. To understand the complex mechanisms of the nervous system, and in particular the brain's behavior, computational neuroscientists construct and analyze computational models at many different levels [2]. In this thesis, we study the problem of inverse modelling, that is, the process of gathering information on a model and its parameters from measurements of what is being modelled. Inverse modelling is important because it tells us about parameters that we cannot directly observe.

1.1 Motivation

Mechanistic models in neuroscience aim to explain neural or behavioral phenomena in terms of causal mechanisms, and candidate models are validated by investigating whether proposed mechanisms can explain how experimental data manifests. The mechanistic modelling is generally through the use of differential equations, and these models often have non-measurable parameters. A central challenge in building a mechanistic model is to identify the parametrization of the system which achieves an agreement between the model and experimental data. Finding well-fitted parameters by inspection becomes more difficult as the complexity of both data and models increase, and automated identification of data-compatible parameters becomes necessary.

Statistical inference provides the mathematical means and procedures for automated parameter identification. Statistical inference uses the *likelihood function* to quantify the match between parameters and data by deriving estimators of the parameters from the data. In statistical inference, there are, broadly speaking, two paradigms for the analysis of sampled data: *frequentist* inference and *Bayesian* inference. In Bayesian inference, prior beliefs about parameters are updated according to *Bayes' theorem* upon observing data. Bayesian inference differs from the traditional frequentist inference by the fundamental interpretation of probability. In terms of parameter inference, the frequentist view is to regard the value of some parameter as fixed but unknown, whereas the Bayesian approach to inference is to regard the parameter as a random variable having a prior probability distribution. Consequently, a posteriori knowledge will also have a probability distribution, known as the posterior distribution. This is one of the most important features of Bayesian inference, as it allows for uncertainty quantification of predictions.

Many mechanistic models are defined through *simulators*, which describes how the process generates data and can be run forward to generate samples from the likelihood. Likelihoods can be derived for purely statistical models, but are generally intractable or computationally infeasible for simulator models. Hence are traditional methods in the toolkit of statistical inference inaccessible for many mechanistic models. To overcome intractable likelihoods, there have been devised a suite of methods that bypass the evaluation of the likelihood function, known as *simulation-based inference* (SBI), or *likelihood-free inference* (LFI), methods. These methods seek to directly estimate either the posterior or the likelihood, and require only the ability to generate data from the simulator to analyze the model in a fully Bayesian context.

Approximate Bayesian Computation (ABC) constitutes a class of computational algorithms rooted in Bayesian statistics that can be used to evaluate posterior distributions of model parameters without having to explicitly evaluate likelihoods. At its heart, the ABC approach is quite simple; evaluation of the likelihood is replaced by comparing simulated data (generated by the simulator model) to observed data, in order to assess how likely it is that the model could have produced the observed data. The curse of dimensionality forces ABC algorithms to measure the *discrepancy* between the simulated and observed data by using *summary statistics* of the data. Therefore, the success of an ABC algorithm largely depends on whether or not the summary statistics capture enough information from the data that are relevant for the parameters of interest. The original ABC algorithm, proposed by Tavaré et al. in [3] and later developed by Pritchard et al. in [4], is built around the standard *rejection sampling algorithm*; the model parameters of simulations that do not reproduce the summary statistics of the observed data within a distance specified by a tolerance are discarded, but those that do are accepted as posterior samples. The term *Approximate Bayesian Computation* was first established by Beaumont et al. in [5], who also extended the ABC approach by using Markov chain Monte Carlo (MCMC) sampling methods and post-sampling regression adjustment for correcting the posterior samples based on

distances between the corresponding simulated summary statistics and the observed ones.

Recently, there have been developed simulation-based inference machine learning algorithms using conditional *neural density estimators* (NDEs), that is, density estimators based on *artificial neural networks* (ANNs). In particular, the *Sequential Neural Posterior Estimation* (SNPE) algorithm targets parametrically learning the posteriors over model parameters by using adaptively proposed model simulations instead of likelihood calculations. More specifically, the algorithm trains a NDE, such as a mixture-density network (MDN) or normalizing flow (NF), to learn the association between data, or summary statistics of the data, and underlying parameters. Instead of filtering out simulations, as ABC algorithms do, SNPE uses *all* simulations to train the NDE to identify admissible parameters. Once trained, the network can then be applied to observed data to derive the posterior densities over the parameters of the simulator model. The strategy behind SNPE was first proposed by Papamakarios and Murray in [6] and further developed to the SNPE algorithm by Lueckmann et al. in [7]. SNPE was later refined by Greenberg et al. in [8]. In the literature, the authors refer to the variant by Papamakarios and Murray as SNPE-A, the variant by Lueckmann et al. as SNPE-B and the variant by Greenberg et al. as SNPE-C. Unless the distinction is made clear, SNPE will in the following refer to the variant by Greenberg et al.

1.1.1 Why Bayesian?

The frequentist approach to parameter identification yields a single best-fit. However, such a local point estimate:

(i) is potentially a poor representation of the true parameter;

(ii) hides the fact that many similar parameter values could be capable of describing the data equally well.

The Bayesian approach to parameter identification, on the other hand, yields a probability distribution (the posterior) of all data-compatible parameters. Thus, the Bayesian approach has the advantage that the uncertainty of an estimate can be quantified due to being encoded in a probability distribution. In addition, we can characterize the model by examining the posterior over model parameters, which can inform us about the ability of the data, or, as in the case of simulation-based inference, summary statistics of the data, to constrain the model. Being able to robustly characterize the model might also aid us in designing a better model to fit the data.

1.2 Objective of the Study

The overall objective of this thesis is to investigate the ability and utility of simulation-based inference for identifying parameters in mechanistic models of

neural dynamics. Specifically, we will investigate the performance of ABC using rejection sampling with post-sampling regression adjustment and the machine learning algorithm SNPE on two neuroscientific models; the Hodgkin-Huxley model [9] and the Brunel network model [10]. The primary focus of the study will be on ABC, and SNPE will be used mostly for comparison.

The seminal Hodgkin-Huxley model is a biophysically detailed description of the ionic mechanisms underlying the initiation and propagation of action potentials in squid giant axons. We will assess the identifiability of the potassium and sodium conductance parameters by examining the width and location of the resulting posterior estimates. As many biophysically detailed neuron models use the Hodgkin-Huxley formalism, the original Hodgkin-Huxley model becomes an ideal case for assessing and illustrating the application of simulation-based inference.

We also consider the Brunel network model for activity dynamics in local cortical networks. Much effort in computational neuroscience today concerns mechanistic models at the network level and the Brunel network is thoroughly analyzed in the literature. The Brunel network is a sparsely connected recurrent network consisting of one excitatory and one inhibitory population of leaky integrate-and-fire (LIF) neurons. The network may be in several different states of spiking activity, largely dependent on the values of the synaptic weight parameters. For the current investigation, we limit our analysis to two of these states; the synchronous regular (SR) state, where the neurons fire almost fully synchronized at high rates; and the asynchronous irregular (AI) state, where the neurons fire mostly independently at low rates. We will assess and compare the identifiability of the synaptic weight parameters with the network both in the SR and AI state.

The choice of summary statistics is crucial for the performance of simulation-based inference algorithms, in particular ABC, as they need to constrain the model well. Therefore, we will also investigate summary statistics of spiking activity in detail.

We divide the overall objective into six parts:

1. Implement simulators for both the Hodgkin-Huxley and Brunel network model in Python.
2. Implement a general ABC rejection sampler with post-sampling regression adjustment in Python.
3. Determine suitable summary statistics of the spiking activity using domain knowledge and develop or find methods for extracting them from the simulated neural data.
4. Assess how well the summary statistics constrain the model parameters by examining sensitivity through a correlation analysis. Based on the correlation analysis, implement an importance weighting procedure for the statistics.

5. Estimate the model parameter posteriors with both ABC and SNPE by using synthetic observed data generated by the simulators, and examine the performance of the simulation-based inference approach.
6. Compare the results obtained via ABC and SNPE and insights they might provide about the neuroscientific models.

1.3 Code

As part of the thesis, we developed the Python packages `pyLFI` and `NeuroModels`.

`pyLFI` uses likelihood-free inference (LFI), also known as simulation-based inference, methods for estimating the posterior distributions over model parameters. Specifically, we have implemented parallelized ABC rejection and Markov chain Monte Carlo (MCMC) samplers as well as procedures for linear and local linear regression adjustment. As a side note: We ended up not using the ABC MCMC sampler or the linear regression adjustment in the present work, as they did not provide any additional insights into the objective of the thesis. The package is made publicly available in the GitHub repository:

https://github.com/nicolossus/pylfi

We chose to implement our own ABC software for several reasons:

1. This being a thesis under a computational science master programme, programming and software development are central aspects.
2. Obtaining "under the hood" knowledge about a method might provide insights about its weaknesses and strengths, as well as a thorough understanding in general.
3. Flexibility. Other software might not facilitate the means for the particular analyses we want to carry out.

The `NeuroModels` toolbox provides a framework for the simulator models and methods for extracting summary statistics from the simulated neural data. The package is located in a separate repository:

https://github.com/nicolossus/neuromodels

Both `pyLFI` and `NeuroModels` are available via the Python Package Index (PyPI). Implementation details are given in Chapter 7.

The SNPE algorithm(s) is implemented in the `sbi` Python package [11]:

https://github.com/mackelab/sbi

All code used to carry out the present study is also made publicly available in the GitHub repository:

https://github.com/nicolossus/Master-thesis

1.4 Notation

In general, only standard and common mathematical notation will be used in this thesis, and special symbols will always be introduced where necessary. The notational convention in this thesis is based on a combination of the ones used in [12] and [13].

General Notation

As general notation, we let θ denote unobservable vector quantities or population *parameters* of interest, y denote the observed data, and $\tilde{y}$ denote unknown, but potentially observable, quantities. In general these symbols represent multivariate quantities. Observed or observable scalars and vectors are generally denoted by lower case Roman letters, and observed or observable matrices by upper case Roman letters. Parameters of models and distributions will mostly be denoted by Greek letters. However, due to notational conventions in the field, this will not be strictly followed when dealing with neuroscientific models. For example, a conductance of a neuroscientific model is usually indicated by g, although it may represent a model parameter from a statistical perspective. When using matrix notation, we consider vectors as column vectors throughout; for example, if u is a vector with n components, then $u^T u$ is a scalar and uu^T an $n \times n$ matrix. *Statistical estimates* of model parameters are indicated by a hat "ˆ", as standard in statistics, e.g., $\hat{\theta}$. Sometimes the hat symbol is also used to indicate a *predicted* value, as in $\hat{y}$.

Probability Notation

We let $p(\cdot \mid \cdot)$ denote a *conditional probability density* and $p(\cdot)$ a *marginal distribution*. The conditional probability $p(A \mid B)$ is the likelihood of event A occurring given that B is true, and the marginal probability $p(A)$ is the probability of observing A. The terms *distribution* and *density* are used interchangeably. For brevity, the term *probability density* will often be condensed into *density*. A *probability mass function*, which gives the probability that a discrete random variable is exactly equal to some value, is abbreviated *pmf*. Similarly, a *probability density function*, associated with continuous rather than discrete random variables, is abbreviated *pdf*. The same notation is used for continuous density functions and discrete probability mass functions. Furthermore, we refer to both pdf and pmf as pdf, when the nomenclature makes no difference.

In a statistical analysis, the assumption is usually that the n values y_i may be regarded as *exchangeable*, meaning that we express uncertainty as a joint density $p(y_1, ..., y_n)$ that is invariant to permutations of the indices. We commonly model data from an exchangeable distribution as *independently and identically distributed* (iid) given some parameter vector θ with distribution $p(\theta)$. The tilde symbol "$\sim$" is used in the sense of "distributed according to", as common in statistics. When using a standard distribution, we use notation based on the name of the distribution. For

example, if θ is distributed according to a normal distribution with mean μ and variance σ^2, we write $\theta \sim \mathrm{N}(\mu, \sigma^2)$ or $p(\theta) = \mathrm{N}(\mu, \sigma^2)$.

Bayesian Inference

In Bayesian inference we encounter conditional densities called *posterior* and *likelihood*. In order to make the distinction clear, we will denote the former by $\pi(\cdot \mid \cdot)$ and the latter by $p(\cdot \mid \cdot)$. We also encounter a marginal density called *prior*, which we will denote by $\pi(\cdot)$.

1.5 Structure of the Thesis

The thesis is organized into four parts.

In Part I we provide the theoretical background. First, we introduce Bayesian inference in general in Chapter 2 and then simulation-based inference along with the algorithms we will use in Chapter 3. Next, we give a brief introduction to neurobiology in Chapter 4 before presenting the neuroscientific models that will be used in the study in Chapter 5.

In Part II we discuss the methodologies we will use in the study. The specifics of the methodologies are given in Chapter 6 and the specifics of the computational approach in Chapter 7.

In Part III we present and discuss the results. Chapter 8 presents the results on the Hodgkin-Huxley model and Chapter 9 the results on the Brunel network model.

In Part IV we summarize our findings and conclusions. In Chapter 10 we put our findings in perspective before concluding in Chapter 11. In Chapter 12 we provide an outline of potential future research.

Part I

Theoretical Background

2

Bayesian Inference

> A decision was wise, even though it led to disastrous consequences, if the evidence at hand indicated it was the best one to make; and a decision was foolish, even though it led to the happiest possible consequences, if it was unreasonable to expect those consequences.
>
> Herodotus, around 500 BC

The aim of statistical inference is to learn about underlying properties of a population from observed data. In statistical inference, there are, broadly speaking, two paradigms for the analysis of observed data: *frequentist* inference and *Bayesian* inference. These often differ with each other on the fundamental interpretation of probability. In the frequentist view, the probabilities of events are defined as their relative frequencies in a repeatable objective process, and are thus ideally devoid of opinion. From a Bayesian perspective, probabilities are measures that quantifies the uncertainty level of statements based on the degree of belief about the state of the world. Probabilities can be assigned to any statement, even when a random process is not involved. Bayesian inference is the process of revising beliefs about the state of the world in the light of new evidence.

This chapter introduces the fundamentals of Bayesian inference, with a particular focus on parameter inference. The content of this chapter is mainly based on the material in the Bayesian textbooks [13], [14] and [15].

2.1 Bayes' Theorem

In terms of parameter inference, the Bayesian approach differs from the frequentist in that unknown parameters θ are treated as random variables rather than fixed quantities. In the Bayesian paradigm, all available information about an unknown parameter is incorporated in a *prior probability distribution*, expressing our beliefs before some evidence is taken into account. We usually have a prior probability density function (pdf), $\pi(\theta)$, since there will typically be a continuum of possible values of a parameter rather than just a discrete set. In the case of substantial prior knowledge about a parameter θ, the prior pdf is narrow and concentrated about some central value, whereas a lack of information yield a wider and relatively flat prior pdf as shown in Figure 2.1. The prior is often specified by a particular distribution among a set of well-known and tractable distributions, with the purpose of making evaluation of prior probabilities and random generation of θ values straightforward.

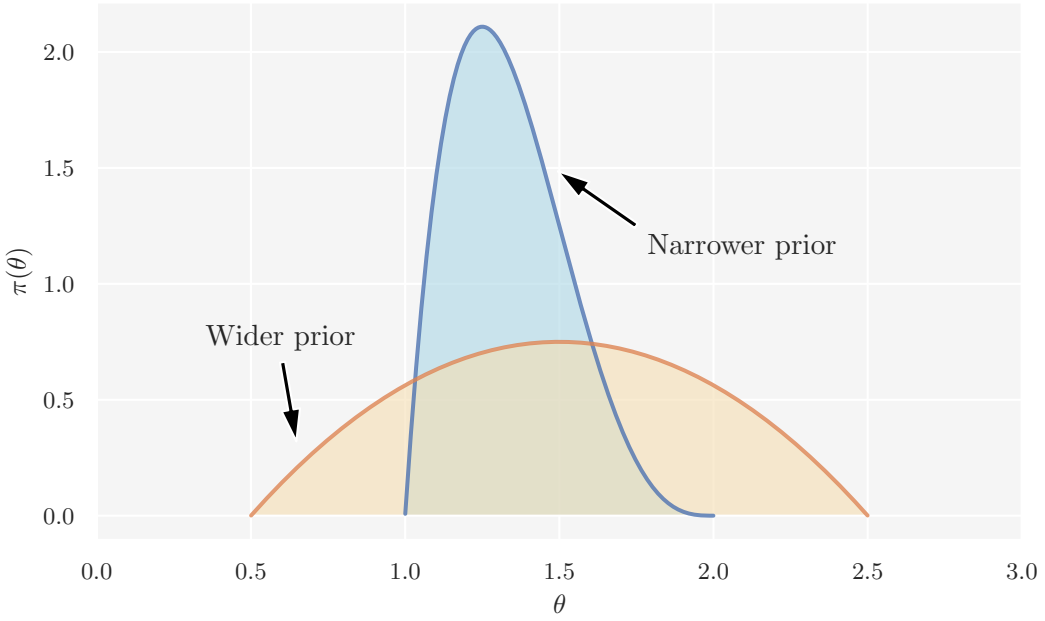


Figure 2.1: Two prior distributions $\pi(\theta)$. A narrow concentrated prior (more certainty) about some central value and a wider less informative prior (less certainty).

Our prior state of knowledge is modified by data y, obtained by performing experiments, through the conditional *sampling distribution* $p(y \mid \theta)$. When regarded as a function of θ, for fixed y, $p(y \mid \theta)$ is called the *likelihood function*. In order to make probability statements about θ given sample data y, a probabilistic model representing the joint probability distribution for θ and y must be provided. The joint pdf can be written as a product of the prior distribution $\pi(\theta)$ and the likelihood function $p(y \mid \theta)$:

$$p(\theta, y) = p(y \mid \theta)\pi(\theta).$$

At this point, Bayes' theorem is used to produce the *posterior distribution*, which represents our state of knowledge about θ in the light of y. A common incarnation of Bayes' theorem is:

$$\pi(\theta \mid y) = \frac{p(\theta, y)}{p(y)} = \frac{p(y \mid \theta)\pi(\theta)}{p(y)}, \tag{2.1}$$

where the marginal probability of the data $p(y) = \int p(y \mid \theta)\pi(\theta)\mathrm{d}\theta$ in the case of continuous parameters, or, in the case of a discrete set of parameters, $p(y) = \sum_\theta p(y \mid \theta)\pi(\theta)$, where the sum is over all possible values of θ.

$p(y)$ is the same for all possible θ, as it does not depend on θ. With fixed y, this factor can thus be omitted in parameter inference since it constitutes a normalizing constant and does not enter into determining the relative posterior probabilities of different values of θ. Omitting the factor $p(y)$ yields the unnormalized posterior distribution:

$$\pi(\theta \mid y) \propto p(\theta, y) = p(y \mid \theta)\pi(\theta). \tag{2.2}$$

In this formulation, $p(y \mid \theta)$ is taken as a function of θ and not y.

The core of Bayesian inference is encapsulated in Equation 2.1 and Equation 2.2. The principal task is to develop the joint probability model $p(\theta, y)$ and perform the computations to obtain the posterior $\pi(\theta \mid y)$.

2.2 Prior and Posterior Predictive Distributions

Before observing any data y, we simply have the chosen model, i.e., the likelihood, $p(y \mid \theta)$, and the prior distribution of θ, $\pi(\theta)$. To make predictive inference about the expected future data, $\hat{y}$, encoded in the prior assumptions, we calculate the marginal distribution of y, that is, the distribution of $y \mid \theta$ averaged over all possible values of θ:

$$p(\hat{y}) = \int p(y \mid \theta)\pi(\theta)\,\mathrm{d}\theta\,. \tag{2.3}$$

Equation 2.3 is called the *prior predictive distribution*.

Once we have a posterior, it is possible to generate predictions $\hat{y}$ following a similar logic. The *posterior predictive distribution* is calculated by marginalizing the distribution of $\hat{y} \mid \theta$ over the posterior distribution:

$$p(\hat{y} \mid y) = \int p(\hat{y} \mid \theta)\pi(\theta \mid y)\,\mathrm{d}\theta\,. \tag{2.4}$$

Thus, the posterior predictive distribution is an average of conditional predictions over the posterior distribution of θ.

2.3 Parameter Inference

The way in which Bayes' theorem operates is best seen through examples. In the following we discuss Bayesian inference in the context of a statistical model where the closed form is available.

2.3.1 The Beta-Binomial Model and the Effect of Priors

The beta-binomial model is one of the simplest Bayesian models, and useful for introducing important concepts and computational methods in Bayesian analysis. The model is often illustrated in the context of the classical coin-flipping problem, where only a single scalar parameter, the success probability θ, is to be estimated.

In the coin-flipping problem, we toss a coin n times and record the observations: either *heads* or *tails*. Based on this data, we try to answer questions such as *is the coin fair?* Or, more generally, *how biased is the coin?* In order to estimate the bias of a coin in a Bayesian setting, we need observed data, a probabilistic model of the data generating process, i.e., the likelihood, and a prior placed on the unknown model parameter. For this example, we assume that the data-gathering part is already done and we have recorded the number of heads after a number of coin flips. The bias of the coin is represented by the θ parameter, and we say that a coin with $\theta = 1$ will always land heads, one with $\theta = 0$ always tails and one with $\theta = 0.5$ has an equal chance of landing either heads or tails. Assuming that only two outcomes are possible, heads or tails, and the random variable *coin toss* is independent and identically distributed (iid), a candidate for the likelihood is the binomial distribution:

$$p(y \mid \theta) = \binom{n}{y}\theta^y(1-\theta)^{n-y}. \tag{2.5}$$

This is a discrete distribution returning the probability of getting y heads (or in general, successes) out of n coin tosses (or in general, trials or experiments) given a fixed value of θ (probability of success).

Figure 2.2 illustrates the binomial distribution for different θ. From the figure we see that θ indicates how likely it is to obtain a head when tossing a coin, making the binomial distribution a reasonable choice for the likelihood.

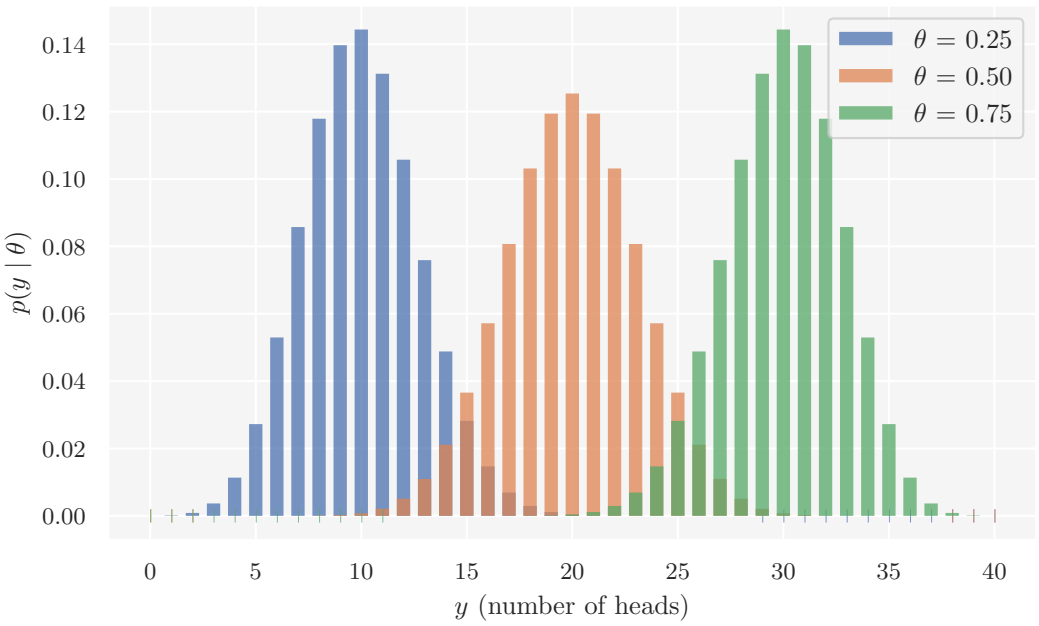


Figure 2.2: Binomial distributions with $n = 40$ coin flips and different success probabilities θ. The coin is biased towards tails when $\theta < 0.5$ (blue) and heads when $\theta > 0.5$ (green). For $\theta = 0.5$ (orange) the coin is unbiased (or fair). The legend indicates the values of the θ.

If the value of θ is known, the binomial distribution tells us the expected distribution of heads. However, θ is an unknown model parameter, and thus we need to place a prior on it. For mathematical convenience, we choose a family of prior densities that lead to simple posterior densities. Considered as a function of θ, Equation 2.5 is of the form:

$$p(y \mid \theta) \propto \theta^a (1-\theta)^b.$$

If the prior density is of the same form, with its own parameterization of a and b, then the posterior will also be of this form. Such a prior density can be parameterized as:

$$\pi(\theta) \propto \theta^{\alpha-1}(1-\theta)^{\beta-1},$$

which is the beta distribution with shape parameters $\alpha > 0$ and $\beta > 0$. The parameters of the prior distribution are often called *hyperparameters*. In order to ensure that the total probability is 1, the beta function,

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)},$$

where $\Gamma(z)$ is the gamma function, can be used as a normalizing constant:

$$\pi(\theta) = \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1}. \tag{2.6}$$

The beta distribution is defined on the interval $[0, 1]$. Figure 2.3 shows the beta distribution with different shape parameters. The figure displays the versatility of the beta distribution; the distribution adopts several shapes, determined by the shape parameters, including the uniform distribution with $\alpha = \beta = 1$. The uniform (blue) prior represents all the possible values for θ being equally likely a priori. The Gaussian-like (orange) prior is concentrated about $\theta = 0.5$, and reflects a belief that the coin is equally probable to land heads or tails. The reverse J-shaped (green) prior is skewed towards a tail-biased outcome.

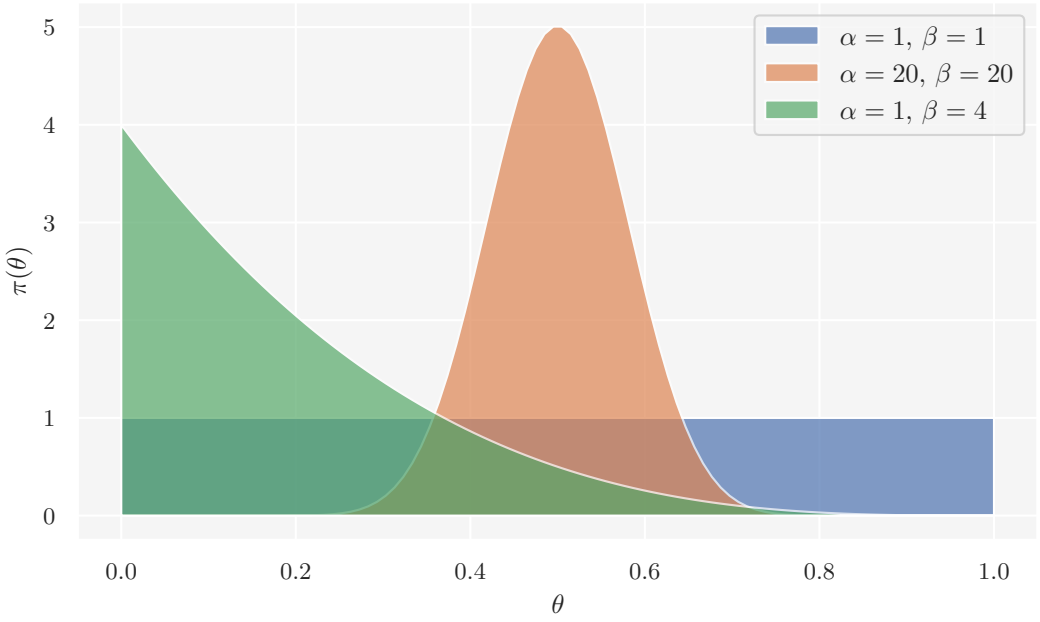


Figure 2.3: The beta prior probability distribution with different parameterizations by the two positive shape parameters. The beta distribution adopts several shapes controlled by the shape parameters; $\alpha = \beta = 1$ gives a uniform distribution (blue), $\alpha = \beta = 20$ gives a bell curve centered at $\theta = 0.5$ (orange) and finally $\alpha = 1$ and $\beta = 2$ gives a reverse J-shaped distribution with a right tail (green).

Bayes' theorem, Equation 2.2, states that the posterior is proportional to the product of the likelihood and the prior. Thus, for our problem the posterior density for θ is given as:

$$\pi(\theta \mid y) \propto \binom{n}{y} \theta^{y}(1-\theta)^{n-y} \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1}.$$

With fixed n and y, the factor $\binom{n}{y}$ does not depend on the unknown parameter θ, and neither does the beta function $B(\alpha, \beta)$. Thus can both be treated as constants when calculating the posterior distribution of θ:

$$\begin{aligned}\pi(\theta \mid y) &\propto \theta^y(1-\theta)^{n-y}\theta^{\alpha-1}(1-\theta)^{\beta-1} \\ &= \theta^{\alpha+y-1}(1-\theta)^{\beta+n-y-1},\end{aligned}$$

or, more concisely:

$$\pi(\theta \mid y) \propto \theta^{\alpha'-1}(1-\theta)^{\beta'-1}, \tag{2.7}$$

with $\alpha' = \alpha + y$ and $\beta' = \beta + n - y$. We recognize that the expression above has the same functional form as the unnormalized beta distribution. The property that the posterior distribution follows the same parametric form as the prior distribution is called *conjugacy*, and we say that the beta distribution is a *conjugate prior* for the binomial likelihood.

Figure 2.4 shows how the posteriors for the priors in Figure 2.3 evolve as more and more data become available. For easier comparison, they have all been scaled vertically to have the same maximum value. Figure 2.4 clearly reveals the effect of the different priors; when there are few data, the shape of the posteriors vary in detail; as the number of data increases, the shape and location of the posteriors tend to converge and they all become sharply peaked. Since the priors reflect the different information or assumptions before the results, and the posteriors the updated knowledge in the light of data, this seems quite reasonable. If the data only are the outcome of a few flips, the analysis of these data is dominated by the prior information. However, as the data increases, the posterior is dominated by the likelihood and we are eventually led to the same conclusions regardless of our initial beliefs. In the limit of infinite data, all priors will provide the same posterior. From a practical point of view, we could obtain nearly indistinguishable posteriors for a finite amount of data. Thus, the choice of the prior becomes largely irrelevant given a sufficiently large amount of data.

Priors are often categorized by the information they incorporate about parameters as either *noninformative*, *weakly informative* or *informative*. If the prior is noninformative, the posterior is data-driven, as illustrated by the uniform (blue) prior in Figure 2.4. On the other hand, if the prior is informative, as illustrated by the bell curve (orange) prior in Figure 2.4, the posterior is a mixture of the prior and the data. However, as mentioned and seen in the figure, the data will overwhelm the prior and dominate the posterior in the case of large amounts of data. Weakly informed priors are constructed to purposely include less information than we actually have. They can be useful if we want to let the data speak but not model complete ignorance.

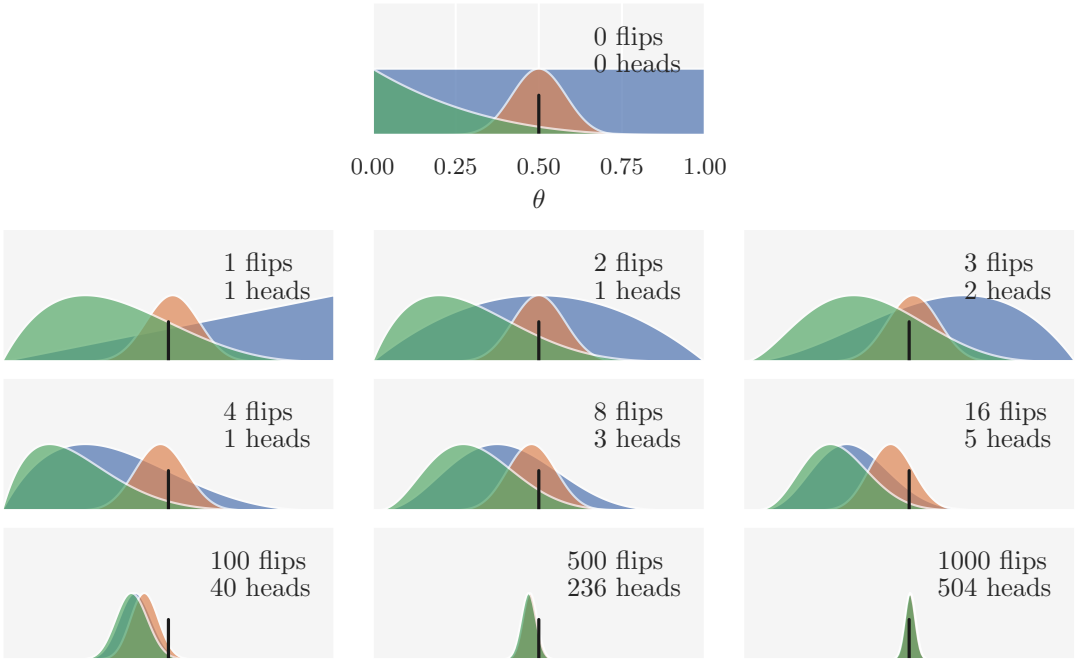


Figure 2.4: The effect of different priors on the posterior as the number of data available increases. To aid in the comparison, they have all been scaled vertically to have the same maximum value. The number of data analyzed is indicated at the upper right-hand corner of each panel. In the first panel there are zero flips, and thus the densities represent the priors from Figure 2.3. The ground truth, $\theta = 0.5$ (the coin is indeed fair), is indicated by the black vertical line.

2.4 Bayesian Computation

While conceptually simple, Bayesian analysis can be mathematically and numerically challenging. For a long time, Bayesians restricted their attention to conjugate families where posteriors can be computed analytically in closed form. However, realistic probabilistic models often lead to analytically intractable expressions. The arrival of the computational era and development of sampling-based numerical methods have transformed the Bayesian analysis practice. In this section we discuss estimating the posterior numerically using algorithms from the Markov Chain Monte Carlo (MCMC) family.

2.4.1 Markov Chain Monte Carlo

There have been devised a suite of methods for constructing and sampling from arbitrary posterior distributions, but Markov chain Monte Carlo (MCMC) methods have become the predominant computational strategy for Bayesian inference. The term *Monte Carlo* refers to methods that rely on the generation of random samples from a distribution of interest. In general, a *Markov chain* is a sequence of states

for which the probability of transitioning to the next state depends only on the present state. That the next state is only conditional on the present state and thus independent of the previous states is known as the Markov property. By providing a starting point, such a chain can perform a random walk between the states according to the transition probabilities. Hence, the main idea of MCMC is to draw samples of θ_t sequentially with the distribution of sampled draws depending on the previous sample θ_{t-1} to construct a Markov chain $\{\theta_t, t = 0, 1, 2, ...\}$. The key to the method's success is finding a Markov chain with transitions proportional to the target posterior distribution, $\pi(\theta \mid y)$. In other words, the success is determined by whether the chain is able to improve the sampling distribution at each step in the simulations, in the sense of converging to the posterior distribution.

2.4.2 The Metropolis Algorithm

The Metropolis algorithm is one of the most established MCMC sampling methods and was originally proposed in [16]. It was later generalized by Hastings in [17] into the Metropolis-Hastings algorithm. In its original form, the Metropolis algorithm is an adaptation of a random walk that explores the local neighborhood of the current value of the Markov chain. It uses an acceptance/rejection rule to converge to the specified target distribution. The algorithm proceeds as follows:

1. Initialize the Markov chain with a starting point θ_0 for which $\pi(\theta_0 \mid y) > 0$. Conceptually, it makes most sense to draw θ_0 from the prior $\pi(\theta)$, though it can be chosen by making an educated guess.
2. For each iteration of t, with $t = 1, 2, ...$:
   (a) Sample a *proposal* parameter θ^* from the *proposal distribution* (also called the *jumping distribution*) $q(\theta^* \mid \theta_{t-1})$ from which sampling is easily done. For the Metropolis algorithm (but not the Metropolis-Hastings algorithm), the proposal distribution must be symmetric, satisfying the condition $q(\theta^* = \theta_a \mid \theta_{t-1} = \theta_b) = q(\theta^* = \theta_b \mid \theta_{t-1} = \theta_a)$ for all θ_a, θ_b and t. Both the normal and uniform distributions are examples of symmetric distributions that satisfies this condition.
   (b) Evaluate the quality of the proposal θ^* by calculating the ratio of posterior densities:

$$r = \frac{\pi(\theta^* \mid y)}{\pi(\theta_{t-1} \mid y)} = \frac{p(y \mid \theta^*)\pi(\theta^*)}{p(y \mid \theta_{t-1})\pi(\theta_{t-1})}. \tag{2.8}$$

   Note that we do not actually use the posterior directly, but rather the proportionality given by Bayes' theorem (Equation 2.2). If the posterior density of θ^* is greater than that of θ_{t-1}, the ratio of the posterior densities will be greater than 1 and we will accept the proposal as the next state of the chain. If the posterior density is greater for θ_{t-1}, we

will not necessarily discard the proposal θ^*. Less probable parameter values are accepted probabilistically:

(c) Calculate the Metropolis acceptance criterion:

$$\alpha = \min(1, r), \tag{2.9}$$

and set

$$\theta_t = \begin{cases} \theta^* & \text{with probability } \alpha \\ \theta_{t-1} & \text{with probability } 1 - \alpha \end{cases}.$$

In this way, the Metropolis algorithm ensures that the chain tends to move towards the highest density regions of the posterior, but it can still move away from these high-density regions and towards the tails of the posterior. The chain being able to assume all possible states, given enough time, is called *ergodicity*, and is an important feature of the Metropolis algorithm.

To implement the algorithm in a computer program, step (c) requires, after computing α for θ^*, the generation of a uniform random number $u \sim \mathrm{U}(0, 1)$. If $u \leq \alpha$, we accept the proposal and set $\theta_t = \theta^*$. If $u > \alpha$, we reject the proposal and set $\theta_t = \theta_{t-1}$ instead. Note that when the proposal is not accepted, this still counts as an iteration of the algorithm.

The normal distribution, $\mathrm{N}(\mu, \sigma^2)$, is an example of a symmetric proposal distribution. Conditioning the normal distribution on the previous value θ_{t-1} of the chain means that the location parameter $\mu = \theta_{t-1}$. The scale parameter σ is a tuning parameter that we increase or decrease if the acceptance rate of the simulations is too high or low, respectively. According to [13], the optimal acceptance rate is 0.44 for single parameter inference problems and 0.23 for for inference problems concerning several parameters.

Algorithm 2.1 summarizes the Metropolis algorithm.

Algorithm 2.1: Metropolis sampling

Inputs :

- A target posterior density $\pi(\theta \mid y) \propto p(y \mid \theta)\pi(\theta)$ consisting of a prior $\pi(\theta)$ and likelihood $p(y \mid \theta)$.
- A symmetric Markov proposal density $q(\theta^* \mid \theta)$.
- An integer $N > 0$.

Initialize :

Sample $\theta_0 \sim \pi(\theta)$.

Sampling :

for $t = 1, ..., N$ **do**

  Generate proposal $\theta^* \sim q(\theta^* \mid \theta_{t-1})$.

  Calculate acceptance criterion $\alpha = \min\left(1, \frac{p(y \mid \theta^*)\pi(\theta^*)}{p(y \mid \theta_{t-1})\pi(\theta_{t-1})}\right)$.

  Sample $u \sim \mathrm{U}(0, 1)$.

  if $u \leq \alpha$ **then**

    $\theta_t = \theta^*$

  else

    $\theta_t = \theta_{t-1}$

2.5 Density Estimation

The probability density function (pdf) is a fundamental concept in statistics. When we estimate the posterior numerically, we obtain random samples of θ drawn from the posterior density $\pi(\theta \mid y)$ but not the posterior pdf itself. In this section, we briefly discuss *density estimation*, that is, methods for constructing an estimate of the pdf from sample data. The focus will be on *nonparametric* approaches to density estimation. It will be assumed that we are given a sample of n univariate observations $x_1, ..., x_n$ whose underlying density is to be estimated. The density estimators will be denoted by $\hat{p}$. The content of this section is based on the material in the statistical learning textbooks [18], [19] and [20].

2.5.1 Histograms

The most basic density estimator is the histogram. Standard histograms simply partition x into k bins of width h and then count the number of observations of x falling in each bin:

$$h(x) = \sum_{i=1}^{n} B(x - \tilde{x}_i; h),$$

where $\tilde{x}_i$ is the center of the bin in which observation x_i lies and

$$B(x;h) = \begin{cases} 1 & \text{if } x \in (-h/2, h/2) \\ 0 & \text{otherwise} \end{cases}.$$

The histogram estimator for the pdf is then:

$$\hat{p}(x) = \frac{h(x)}{nh} \tag{2.10}$$

This definition gives a density estimate $\hat{p}(x)$ that is constant over the width of each bin, and all bins have the same width h. The number of bins k can be assigned directly or calculated from a suggested bin width h as:

$$k = \left\lceil \frac{\max x - \min x}{h} \right\rceil,$$

where the braces indicate the ceiling function. The amount of smoothing inherent in the procedure is primarily controlled by the bin width. Too small bin widths can give a density model with structure not present in the underlying data-generating density, and too large bin widths a density model that is too smooth for capturing the nuances of the underlying density. There are no hard-and-fast rules concerning the bin width, but there are some rules of thumb for setting h based on n and the dimensionality of the problem, such as Scott's rule [21] and Freedman–Diaconis' rule [22].

In practice, the histogram density estimator can be useful for quickly visualizing data in one or two dimensions, but is unsuited for most density estimation applications. For higher dimensional data, histograms are likely to run into the curse of dimensionality as the number of bins have an exponential scaling with the dimension. Another major issue is that the density has discontinuities that are due to the bin edges. Figure 2.5 in the next section provides an example of the histogram density estimator which illustrates this. However, the two important principles of density estimation are encapsulated in the histogram approach. First, in order to estimate the density at a particular location we should smooth the local neighborhood of that point. Second, the smoothing parameter should be neither too small nor too large to obtain an accurate estimate. The concept of locality requires some form of distance measure, and we have here assumed the Euclidean distance.

2.5.2 Kernel Density Estimation

Kernel density estimates (KDEs) are closely related to histograms, but avoid some of the drawbacks. For instance, they have better scaling with dimensionality and can provide continuous density estimates. This is achieved by replacing the indicator function B of the histogram density estimator with a standard *smoothing kernel function*, defined by:

$$K_h(\|x - x_i\|) = \frac{1}{h} K\left(\frac{\|x - x_i\|}{h}\right), \tag{2.11}$$

where $h > 0$ is the smoothing parameter, $\|\cdot\|$ denotes the Euclidean distance and $K(u)$ the kernel of a pdf. A kernel with subscript h is usually referred to as a scaled kernel. The kernel estimator for the pdf is thus:

$$\hat{p}(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(\|x - x_i\|) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{\|x - x_i\|}{h}\right). \tag{2.12}$$

The smoothing parameter h is often called the *bandwidth* in the context of kernel smoothing, and corresponds to the scale of the kernel. There are also rules-of-thumb for setting the bandwidth, such as Silverman's rule [23]. Intuitively, the kernel estimator is a sum of "bumps" placed at the sample points. The kernel determines the shape of the bumps and the bandwidth their width. Figure 2.5 provides an illustration of this, and also compares the histogram and kernel density estimate using the same data. The smoothness of the KDE compared to the discreteness of the histogram illustrates how KDEs converge faster to the underlying pdf.

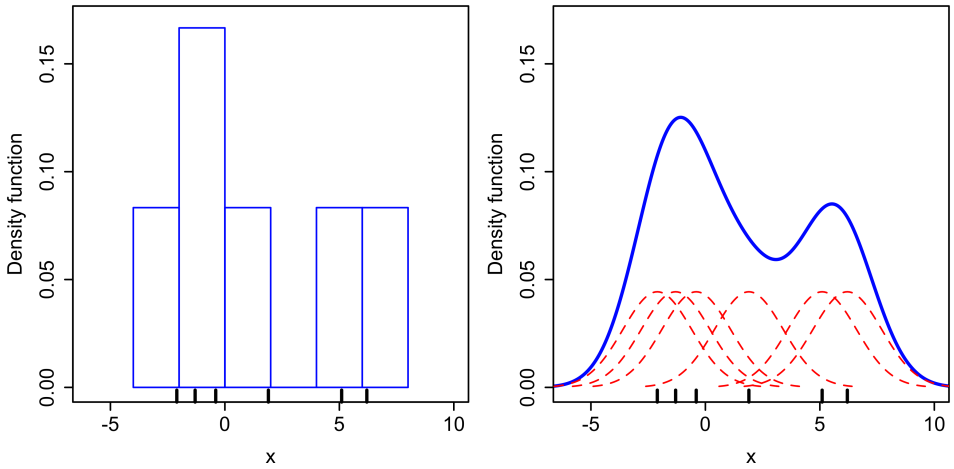


Figure 2.5: Comparison of the histogram (left) and kernel density estimate (right) constructed using the same six data samples. For the histogram, the data is partitioned into $k = 6$ bins, each of width $h = 2$. If more than one data point falls inside the same bin, the density (height) of the bin increases. Note the discontinuities that are due to the bin edges. For the kernel density estimate (KDE), the kernel is Gaussian with bandwidth $h = 2.25$. The kernel is placed on each of the six data samples (indicated by the red dashed curves), and the kernels are summed to make the KDE (solid blue curve). The data samples are shown as the rug plot on the horizontal axis.

Source: [24].

The kernel of a pdf is the form of the pdf in which any factors that are not functions of any of the variables in the domain are omitted. Kernels are symmetric functions such that $K(u) \geq 0$ for all u, $\int K(u)\,\mathrm{d}u = 1$ and $\int u^2 K(u)\,\mathrm{d}u < \infty$. Some common choices of kernels are the Gaussian kernel:

$$K(u) \propto \exp\left(-\frac{1}{2}u^2\right), \tag{2.13}$$

and the Epanechnikov kernel:

$$K(u) \propto \begin{cases} 1 - u^2 & \text{if } |u| \leq 1 \\ 0 & \text{otherwise} \end{cases}. \tag{2.14}$$

Figure 2.6 shows the Gaussian and Epanechnikov kernels. Kernels may or may not have finite support. Kernels with finite support are defined on a domain such as $[-1, 1]$, and kernels without on $(-\infty, \infty)$. The Gaussian kernel does not have finite support, while the Epanechnikov kernel has.

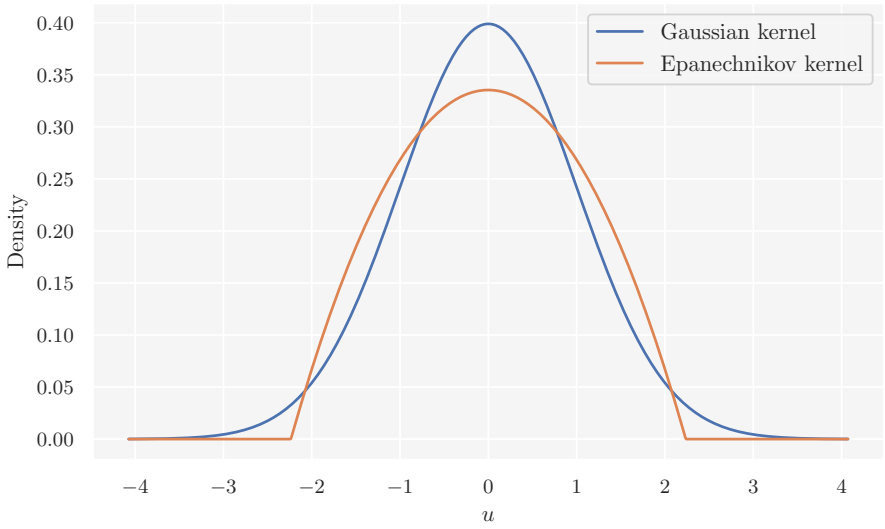


Figure 2.6: The Gaussian and Epanechnikov kernels defined by Equation 2.13 and Equation 2.14, respectively.

2.6 Summarizing the Posterior

The result of a Bayesian analysis is a posterior distribution which contains all the current information about the parameters θ. The focus of this section will be on how we can summarize the obtained posteriors.

2.6.1 Visualization

Visualizing the posterior is a useful first-step in assessing the results of an inference, as the shape of the posterior quickly tells us how well the procedure was. However, we generally obtain estimated posterior samples and not the posterior density itself. In order to examine the location and width of the posterior, we therefore have to generate a visual representation by using density estimation, in particular KDE. If θ is a one- or two dimensional vector, we can simply plot the full posterior, which will be a joint posterior in the latter case. If the parameter vector has more than two dimensions, we can plot the marginal posterior for each parameter.

However, it is useful to also derive summary statistics from the posterior, as numerical summaries often are easier to present and interpret than the full posterior.

2.6.2 Bayesian Point Estimates

Bayesian point estimates are properties of the posterior. The most common point estimates, $\hat{\theta}$, are the:

- Posterior mean:
$$\hat{\theta}_{\text{mean}} = \mathrm{E}[\theta \mid y] = \int \theta \pi(\theta \mid y)\, \mathrm{d}\theta\,, \tag{2.15}$$
which simply is the expected value of the posterior distribution.
- Posterior median:
$$\hat{\theta}_{\text{median}} = \mathrm{F}^{-1}_{\theta|y}(0.5), \tag{2.16}$$
where the cumulative posterior distribution $F_{\theta|y}\left(\tilde{\theta}_q\right) = \int_{\infty}^{\tilde{\theta}_q} \pi(\theta \mid y)\, \mathrm{d}\theta$. Here, $\tilde{\theta}_q$ is the value of θ at the q-quantile. For example, the median is the 0.5-quantile and the value of the cumulative posterior distribution is thus $F_{\theta|y}\left(\tilde{\theta}_{0.5}\right) = 0.5$. The median being the 0.5-quantile entails that it is the value of θ which divides the posterior in half.
- Posterior mode which is also called the maximum a posteriori probability (MAP) estimate:
$$\hat{\theta}_{\text{MAP}} = \max_{\theta} \pi(\theta \mid y), \tag{2.17}$$
and is the point at which the density is highest.

As seen by the definitions above, each of the location summaries has its own interpretation. In general, there is no particular summary statistic that is preferred, and there will be different reasons to use each summary statistic. The mean is often used because it is a simple and familiar concept, or the observable is truly believed to be the average of some process. However, the mean might be a misleading point estimate for more complex and skewed distributions, such that $\hat{\theta}$ ends up in a region of low posterior density. The median is more robust to outliers than the mean, and we might prefer an estimate in the middle of the distribution. However,

as with the mean, the median might also end up in regions of low density for more complex posteriors. The MAP estimate is closely related to the frequentist maximum likelihood estimate (MLE), and may be interpreted as the single "most likely" value. Providing an estimate of the MAP is more involved as we need to construct a KDE, which also may affect the quality of the estimate, based on the posterior samples and then find the mode of the KDE. The MAP estimate is most sensible to use for unimodal posteriors that have well-defined peaks. The MAP of a multimodal posterior might happen to be at one extreme and if the posterior is mostly flat the MAP might end up at an arbitrary location.

2.6.3 Posterior Uncertainty

In addition to posterior point estimates, it is important to report posterior uncertainty. The uncertainty can be characterized by the posterior standard deviation,

$$\text{sd}(\theta \mid y) = \sqrt{\text{var}(\theta \mid y)} = \sqrt{\int \left(\theta - \bar{\theta}\right)^2 \pi(\theta \mid y)\, \mathrm{d}\theta},$$

where $\text{var}(\theta \mid y)$ is the posterior variance and $\bar{\theta} = \text{E}[\theta \mid y]$ the posterior mean. The standard deviation works well for Gaussian-like distributions, but can be misleading for other, in particular skewed, distributions. An alternative is to use *credible intervals* to quantify the posterior uncertainty. In the Bayesian paradigm, a credible interval is an interval within which θ falls with a particular probability. They are the Bayesian analogue of a frequentist confidence interval, though the interpretation, as usual, is different. In the Bayesian setting, an interval having a posterior probability .95 gives a 95% probabilistic belief that the parameter is in that interval. A $100(1-\alpha)\%$ credible interval is a subset of θ such that

$$\int \pi(\theta \mid y)\, \mathrm{d}\theta = 1 - \alpha,$$

where α is the confidence level. If the confidence level is $\alpha = 0.05$, we have a 95% credible interval.

Credible intervals are not unique on a posterior. This means that we have to define a condition to construct a suitable interval. A common condition is to use the set of points for which the posterior density is higher than for any outside this set. This will be the narrowest interval on the posterior for some given confidence level, and is often referred to as the *highest posterior density interval* (HPDI) or just the *highest density interval* (HDI).

2.6.4 Posterior Predictive Checks

The generated predictions $\hat{y}$ from the posterior predictive distribution, Equation 2.4, can be used to validate and criticize the model by comparing them with the observed

data y. This is known as *posterior predictive checks* (PPCs). The main idea of a PPC is to check for auto-consistency. The predicated data are simulated by typical posterior parameter values. If the predictions do not fit the observed patterns of interest, this might inform us on potential limitations of the model.

3

Simulation-Based Inference

Computational neuroscientists have developed complex mechanistic models to describe neural phenomena of interest. Many mechanistic models are defined through simulators which describe how the process generates data. However, simulators are poorly suited for inference and lead to challenging inverse problems. Standard Bayesian inference is performed within the context of a statistical model from which the likelihood can be derived. Likelihoods are generally intractable or computationally infeasible for simulator models, which makes the typical approach to inference inaccessible.

In this chapter we discuss *simulation-based inference* (SBI), that is, algorithms that avoid explicit likelihood evaluations by instead using model simulations. SBI is perhaps best known under the moniker *likelihood-free inference* (LFI). Nevertheless, we prefer the term SBI to LFI as the latter indicates that the likelihood is not present at all, which we will see is not the case.

From this chapter and onwards, there will be a minor notational tweak. In order to avoid ambiguity, observed data will be denoted by y_{obs} and simulated data, generated by the simulator model, by y_{sim}.

3.1 Likelihood-Based vs. Simulation-Based

In this section, we detail the differences between likelihood-based and simulation-based inference. The content of this section is based on [25], [26] and [27].

Suppose a data-generating process is controlled by parameters θ, and the process generates data y when run forward. We assume that the process defines the

conditional likelihood function $p(y \mid \theta)$ for every setting of θ. Given an observation y_{obs}, the problem of interest is to infer parameter settings compatible with y_{obs}. That is, we want to compute the posterior density $\pi(\theta \mid y_{\text{obs}})$ obtained by Bayes' theorem (Equation 2.2). The choice of inference algorithm primarily depends on how the data-generating process is modelled.

A purely statistical model, or an *explicit model*, describes the likelihood $p(y_{\text{obs}} \mid \theta)$ of the process given values for y_{obs} and θ. With an explicit model, the posterior $\pi(\theta \mid y_{\text{obs}})$ is, in general, easily evaluated using Bayes' theorem. Samples from the posterior can be generated using Bayesian computation methods such as Markov chain Monte Carlo algorithms, as discussed in Section 2.4. Such methods are referred to as *likelihood-based inference* methods, as they explicitly evaluate the likelihood.

On the other hand, a *simulator model*, or an *implicit model*, describes how the process generates data. Many mechanistic models are defined through simulator models. For any parameter setting θ, a simulator model can be run forward to generate samples from the likelihood $p(y_{\text{obs}} \mid \theta)$. Unlike for explicit models, implicit simulator models generally have intractable or computationally infeasible likelihoods. The complexity or absence of the associated likelihood typically arises from it involving computationally expensive or intractable integrals, or that the simulator's internal states are unavailable. In order to perform inference on a simulator model, methods using simulations from the model rather than likelihood evaluations are needed. Such methods are referred to as *simulation-based inference* methods.

In general, simulation-based methods are less efficient than likelihood-based as the former can require lots of simulations to produce accurate results. One of the main topics of research in simulation-based inference is how to reduce the required number of simulations without sacrificing inference quality.

3.2 Approximate Bayesian Computation

Approximate Bayesian Computation (ABC) constitutes a class of computational algorithms rooted in Bayesian statistics that can be used to evaluate posterior distributions of model parameters without having to explicitly calculate likelihoods. In this section we discuss ABC and its two fundamental algorithms: the original *rejection ABC* algorithm and the more sophisticated *Markov chain Monte Carlo (MCMC) ABC* algorithm.

The content of this section is mainly based on material from [25].

3.2.1 The ABC of Approximate Bayesian Computation

Given observed data y_{obs}, a simulator model $\text{M}(\theta)$ with parameters θ having prior $\pi(\theta)$, we seek an algorithm to sample from the posterior $\pi(\theta \mid y_{\text{obs}}) \propto p(y_{\text{obs}} \mid \theta)\pi(\theta)$. This can be achieved by using the *rejection sampling algorithm*:

Algorithm 3.1: Rejection sampler

1 Sample $\theta \sim \pi(\theta)$.
2 Accept θ with probability proportional to the likelihood $p(y_{\text{obs}} \mid \theta)$.

Algorithm 3.1 can be made more general and avoid the need to explicitly compute probabilities by using the following, stochastically equivalent, formulation:

Algorithm 3.2: General rejection sampler

1 Sample $\theta \sim \pi(\theta)$.
2 Simulate data y_{sim} from $\text{M}(\theta)$.
3 Accept θ if $y_{\text{sim}} = y_{\text{obs}}$.

Algorithm 3.2 is due to Rubin [28]. The chance of the outcome $y_{\text{sim}} = y_{\text{obs}}$ will, however, be vanishingly small for most problems, and thus vastly time consuming to compute. Algorithm 3.2 will therefore, typically, not be an efficient algorithm. This is where the approximation to Bayesian computation comes into play. We define a discrepancy metric $\rho(\cdot, \cdot)$ to compare the simulated and observed data and a tolerance $\epsilon \geq 0$. The approximate Bayesian computation (ABC) algorithm is then:

Algorithm 3.3: Approximate rejection sampler

1 Sample $\theta \sim \pi(\theta)$.
2 Simulate data y_{sim} from $\text{M}(\theta)$.
3 Compute $\rho \equiv \rho(y_{\text{sim}}, y_{\text{obs}})$, and accept θ as an approximate draw from $\pi(\theta \mid y_{\text{obs}})$ if $\rho \leq \epsilon$.

Algorithm 3.3 is called the *rejection ABC algorithm* and was first proposed by Tavaré et al. in [3] and developed by Pritchard et al. in [4]. In the scheme of Pritchard et al., the simulated and observed data were compared through a choice of summary statistics. We will discuss the algorithm in more detail in the next section.

3.2.2 Rejection ABC

Rejection ABC, as outlined in Algorithm 3.3, is a rejection sampling algorithm for obtaining independent samples from the approximate posterior $\pi_{\text{ABC}}(\theta \mid \rho(y_{\text{sim}}, y_{\text{obs}}) \leq \epsilon)$, where $\rho(\cdot, \cdot)$ is a discrepancy metric, e.g. the Euclidean distance, and $\epsilon \geq 0$ a tolerance. The algorithm proceeds by first sampling a set of parameters θ from the prior, then generate simulated data y_{sim} under the simulator model $\text{M}(\theta)$ specified by the sampled θ, and finally accepting and retaining θ if the distance between

y_{sim} and y_{obs} is no more than ϵ. If $\rho(y_{\text{sim}}, y_{\text{obs}}) = 0$, then $y_{\text{sim}} = y_{\text{obs}}$, and the accepted θ is a sample from the true posterior. The tolerance parameter ϵ thus controls the trade-off between estimation accuracy and computational efficiency. With sufficiently small ϵ, the accepted samples follow the exact posterior more closely, though the algorithm accepts less often. On the other hand, the algorithm accepts more often with a large ϵ, but the accepted samples might yield a replica of the prior.

Stochastically matching y_{sim} and y_{obs} becomes increasingly difficult with increasing dimensionality of the data. In order to operate efficiently, ABC algorithms require a compression of the data into low-dimensional summary statistics $s = S(y)$. A summary statistic that contains the same amount of information about model parameters as the whole dataset, is referred to as being a sufficient statistic $\tilde{s}$. Thus,

$$\begin{aligned} \pi(\theta|y_{\text{obs}}) &= \lim_{\epsilon \to 0} \pi_{\text{ABC}}(\theta \mid \rho(y_{\text{sim}}, y_{\text{obs}}) \leq \epsilon) \\ &= \lim_{\epsilon \to 0} \pi_{\text{ABC}}(\theta \mid \rho(\tilde{s}_{\text{sim}}, \tilde{s}_{\text{obs}}) \leq \epsilon) \\ &\approx \lim_{\epsilon \to 0} \pi_{\text{ABC}}(\theta \mid \rho(s_{\text{sim}}, s_{\text{obs}}) \leq \epsilon) \end{aligned} \tag{3.1}$$

The choice of summary statistics is crucial for the performance of ABC algorithms, and the optimal choice is a minimal sufficient summary statistic. It is common to use the Fisher-Neyman factorization theorem to determine whether or not a summary statistic is sufficient. The theorem is based on being able to re-express the likelihood as a function of the sufficient summary statistic and data. Unfortunately, in the context of simulation-based inference, the likelihood is unavailable and we cannot examine a summary statistic to determine if the Fisher-Neyman factorization theorem holds. However, if powerful low-dimensional summary statistics are established, ABC algorithms can still offer a reasonable performance. The dimension of the vector of summary statistics s should be large enough so that it capture as many important features of the observed data as possible, but also low enough so the curse of dimensionality of matching s_{sim} and s_{obs} is avoided.

Figure 3.1 gives a conceptual overview of the rejection ABC algorithm in a two-dimensional summary statistic space. Having observed data y_{obs} provided by nature and reduced it to a summary statistic s_{obs} (the blue dot), we sample parameter values $\theta \sim \pi(\theta)$, generate simulated data y_{sim} through the simulator $\text{M}(\theta)$, compute the simulated summary statistic s_{sim} and compare with the observation. Here, the discrepancy metric is the Euclidean distance, and the acceptance region amounts to a circle (indicated by the shaded grey circle) with center according to the observation and radius determined by the tolerance ϵ. The θ that correspond to s_{sim} within this circle are accepted (the green dots), and those outside the circle are rejected (the red dots).

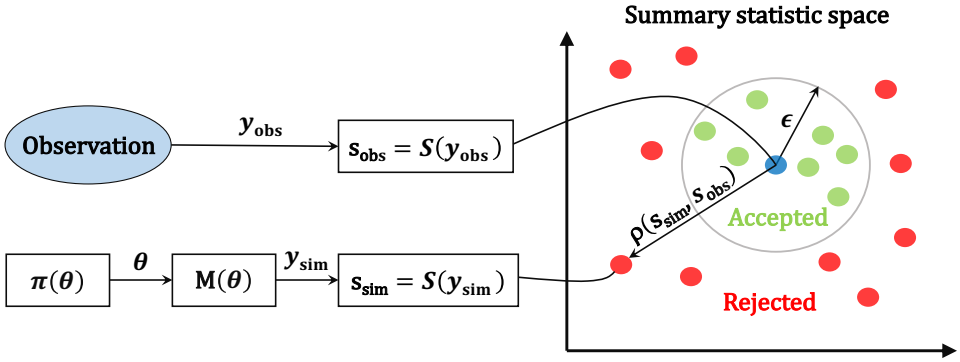


Figure 3.1: A conceptual overview of rejection ABC in 2D summary statistic space. The discrepancy metric is the Euclidean distance, which has a circular acceptance region with the observed summary statistic s_{obs} as center (blue dot) and tolerance ϵ as radius. The parameters θ with corresponding simulated summary statistics s_{sim} that fall within the acceptance region are accepted (green dots), and those outside are rejected (red dots). See text for additional details.

Source: Adapted from Fig. 3 in [29].

Algorithm 3.4 summarizes the full rejection ABC algorithm.

Algorithm 3.4: Rejection ABC

Inputs :

- An observation y_{obs}.
- A simulator model $\text{M}(\theta)$ with parameters θ having prior $\pi(\theta)$.
- A discrepancy metric $\rho(\cdot, \cdot)$ and threshold $\epsilon \geq 0$.
- A summary statistics function $S(y)$.
- An integer $N > 0$.

Sampling :

for $i = 1, ..., N$ **do**

1 Sample $\theta \sim \pi(\theta)$.

2 Simulate data y_{sim} from $M(\theta)$.

3 Calculate $\rho \equiv \rho(S(y_{\text{sim}}), S(y_{\text{obs}})) = \rho(s_{\text{sim}}, s_{\text{obs}})$.

if $\rho \leq \epsilon$ **then**

4 Accept θ.

else

5 Go to 1.

3.2.3 Markov Chain Monte Carlo ABC

In Rejection ABC, the proposal parameters are sampled from the prior. Although sampling from the prior is efficient, the efficiency of the overall procedure will be determined by whether the prior is chosen so that it is of a similar shape and location as the desired posterior. The acceptance rate will be low if the approximate posterior is significantly narrower than the prior, as is often the case. The rejection ABC algorithm will thus become more computationally inefficient as more of the expensive data generation becomes necessary. In this case, we might choose to develop a recursive strategy where the quality of previous candidates in our sampling algorithm can be used to guide the candidate generation process. Basing proposal samples off of previous states is precisely the strategy behind conventional Markov chain Monte Carlo (MCMC) methods, as we introduced in Section 2.4.

Beaumont et al. [5] were the first to both extend the ABC approach to use MCMC methods and use the term *Approximate Bayesian Computation*. Algorithm 3.5 summarizes the *MCMC ABC* algorithm with Metropolis sampling (also discussed in Section 2.4). To perform Metropolis sampling in the simulation-based context, we calculate the probability of accepting the proposal θ^* by evaluating:

$$\alpha = \begin{cases} \min\left(1, \frac{\pi(\theta^*)}{\pi(\theta_{t-1})}\right) & \text{if } \rho(s_{\text{sim}}, s_{\text{obs}}) \leq \epsilon \\ 0 & \text{if } \rho(s_{\text{sim}}, s_{\text{obs}}) > \epsilon \end{cases} \tag{3.2}$$

Similarly to rejection ABC, the acceptance probability of MCMC ABC decreases as ϵ becomes small. Moreover, the performance of MCMC ABC strongly depends on the selection of proposal and prior density.

Algorithm 3.5: Markov chain Monte Carlo ABC with Metropolis sampler

Inputs :

- An observation y_{obs}.
- A simulator model $\text{M}(\theta)$ with parameters θ having prior $\pi(\theta)$.
- A symmetric Markov proposal density $q(\theta^* \mid \theta)$.
- A discrepancy metric $\rho(\cdot, \cdot)$ and threshold $\epsilon \geq 0$.
- A summary statistics function $S(y)$.
- An integer $N > 0$.

Initialize :

Sample θ_0 by performing one iteration of rejection ABC (Algorithm 3.4).

Sampling :

for $t = 1, ..., N$ **do**

  Generate proposal $\theta^* \sim q(\theta^* \mid \theta_{t-1})$.

  Simulate data y_{sim} from $M(\theta^*)$.

  Calculate $\rho \equiv \rho(S(y_{\text{sim}}), S(y_{\text{obs}})) = \rho(s_{\text{sim}}, s_{\text{obs}})$.

  Calculate acceptance criterion α (Equation 3.2).

  Sample $u \sim \text{U}(0, 1)$.

  if $u \leq \alpha$ *and* $\alpha \neq 0$ **then**

    $\theta_t = \theta^*$

  else

    $\theta_t = \theta_{t-1}$

3.3 Regression Adjustment

In this section, we discuss a post-sampling refinement called *regression adjustment*, first proposed by Beaumont et al. in [5]. The goal of the method is to improve the posterior approximation. Usually, we choose summary statistics with a systematic relationship to the model parameters. The idea behind regression adjustment is to correct the accepted posterior samples based on this relationship and the distance between the simulated and observed summary statistics. The content of this section is based on material from [5], [25] and [30].

3.3.1 Linear Regression Adjustment

Let $s_{\text{obs}} = (s_{\text{obs},1}, ..., s_{\text{obs},m})$ be the vector of observed summary statistics, where we let $s_{\text{obs},j}$ denote the jth summary statistic and assume appropriate scaling of the summary statistics, e.g., to equalize variances. An ABC algorithm generates a vector of parameters $\theta = (\theta_1, ..., \theta_l)$ by comparing the vectors of simulated summary statistics $s_{\text{sim}} = (s_{\text{sim},1}, ..., s_{\text{sim},m})$ with the observation. The posterior approximation of the kth model parameter, θ_k, is a vector consisting of n accepted samples, $\theta_k = \left(\theta_k^{(1)}, ..., \theta_k^{(n)}\right)$. In the following, we let $\theta^{(i)}$ indicate the ith accepted

sample of model parameters with $i = 1, ..., n$. Furthermore, assume we retain the simulated summary statistics associated with the accepted $\theta^{(i)}$ sample, such that we also have the vector $s_{\text{sim}}^{(i)}$, $i = 1, ..., n$. Thus, we have n pairs of $\left(\theta^{(i)}, s_{\text{sim}}^{(i)}\right)$.

It is convenient to start our discussion of regression adjustment based on standard linear regression to explain the main ideas. The first objective is to model the relationship between the m-dimensional vector of summary statistics and the l-dimensional vector of model parameters, where each vector component itself is an n-dimensional vector consisting of the associated accepted samples. A model for linearly regressing the summary statistics on the obtained posterior samples is:

$$\theta^{(i)} = \alpha + \left(s_{\text{sim}}^{(i)} - s_{\text{obs}}\right)^{\mathrm{T}} \beta + \xi^{(i)}, \qquad i = 1, ..., n \tag{3.3}$$

where α is the intercept, β the vector of regression coefficients and $\xi^{(j)}$ the residual assumed to be uncorrelated with mean zero and common variance. No other distributional assumptions are made about $\xi^{(i)}$ and hence $\theta^{(i)}$. When $s_{\text{sim}}^{(i)} = s_{\text{obs}}$, the $\theta^{(i)}$ are samples from the desired posterior with mean α. The least squares estimate of (α, β) is:

$$\left(\hat{\alpha}, \hat{\beta}\right) = \left(X^{\mathrm{T}} X\right)^{-1} X^{\mathrm{T}} \theta, \tag{3.4}$$

where X is the matrix of summary statistics augmented with a column of ones:

$$X = \begin{bmatrix} 1 & s_{\text{sim},1}^{(1)} - s_{\text{obs},1} & s_{\text{sim},2}^{(1)} - s_{\text{obs},2} & \cdots & s_{\text{sim},m}^{(1)} - s_{\text{obs},m} \\ 1 & s_{\text{sim},1}^{(2)} - s_{\text{obs},1} & s_{\text{sim},2}^{(2)} - s_{\text{obs},2} & \cdots & s_{\text{sim},m}^{(2)} - s_{\text{obs},m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & s_{\text{sim},1}^{(n)} - s_{\text{obs},1} & s_{\text{sim},2}^{(n)} - s_{\text{obs},2} & \cdots & s_{\text{sim},m}^{(n)} - s_{\text{obs},m} \end{bmatrix}.$$

The strategy is then to adjust the set of posterior samples θ to have mean α while simultaneously correcting for the trend in the relationship between parameters and summary statistics. It follows from Equation 3.3 that the corrected posterior sample $\tilde{\theta}^{(i)}$ defined by

$$\tilde{\theta}^{(i)} = \theta^{(i)} \left(s_{\text{sim}}^{(i)} - s_{\text{obs}}\right)^{\mathrm{T}} \hat{\beta} \tag{3.5}$$

form an approximate random sample from the posterior $\pi(\theta \mid s_{\text{obs}})$. If the relationship between the model parameters and the summary statistics is truly linear, which also implies that the summary statistics are sufficient, this will be exact (assuming that also the distributional assumptions about $\xi^{(i)}$ are met and the sample size is large so that $\left(\hat{\alpha}, \hat{\beta}\right) = (\alpha, \beta)$). It should be noted that $\hat{\alpha}$ can be interpreted as a point estimate of θ since it is an estimate of the posterior mean.

It may not be immediately evident how this correction helps to improve the posterior approximation, so here comes an explanation: A crucial limitation of the ABC algorithms is that they can only handle a small number of summary statistics. However, in order to constrain the model parameters accurately we might need several informative summary statistics. As a consequence, we usually have a trade-off between prohibitively low acceptance rates or large tolerances which can distort the approximation. This arises from the fact that s_{sim} are treated equally whenever $\rho(s_{\text{sim}}, s_{\text{obs}}) \leq \epsilon$, regardless of the proximity between s_{sim} and s_{obs}. The idea of regression adjustment is to learn a regression model and then correct the accepted simulations as if they were sampled from $\pi_{\text{ABC}}(\theta \mid \rho(s_{\text{sim}}, s_{\text{obs}}) \leq \epsilon)$ with $\epsilon = 0$. With this insensitivity to ϵ, we can also increase the number of summary statistics and thus potentially extract more information from the data. Figure 3.2 illustrates linear regression adjustment of a univariate parameter θ.

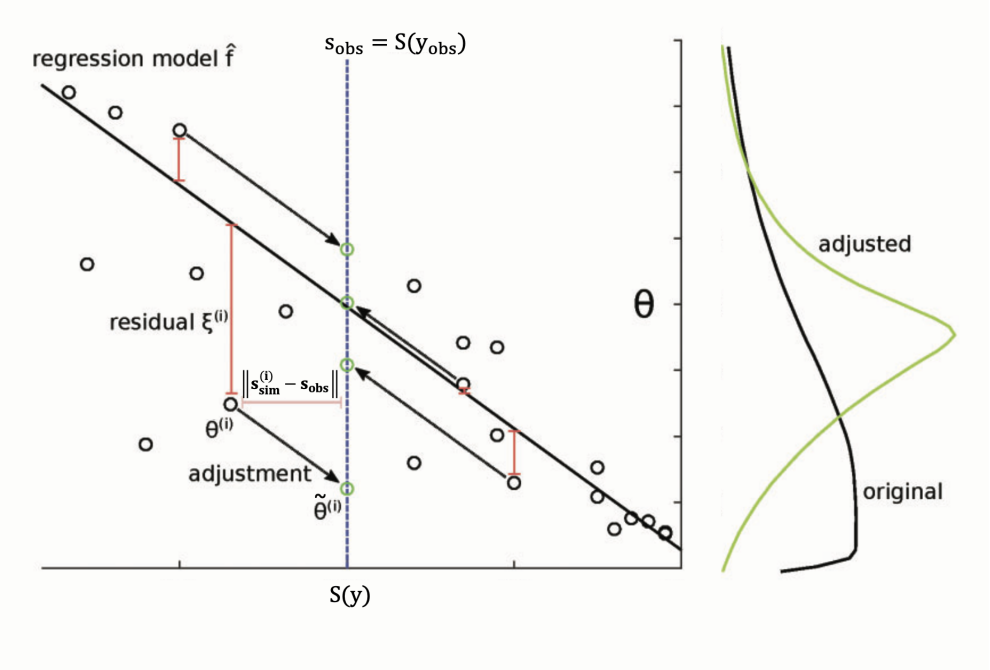


Figure 3.2: Illustration of linear regression adjustment of a univariate parameter θ. The vertical blue dashed line indicates the observed summary statistic, s_{obs}. First, a regression model $\hat{f}$ is learned and then, based on $\hat{f}$, all accepted posterior samples $\theta^{(i)}$ are corrected as if they were sampled from $\pi_{\text{ABC}}(\theta \mid \rho(s_{\text{sim}}, s_{\text{obs}}) \leq \epsilon)$ with $\epsilon = 0$. Note that the residuals $\xi^{(i)}$ are preserved. Corrected samples are denoted by $\tilde{\theta}^{(i)}$, and the shift in values are indicated by the black arrows pointing to the positions of the corrected values (green circles). The original posterior density is shown on the right (black curve) together with the adjusted posterior (green curve).

Source: Modified from Fig. 9 in [31].

3.3.2 Local Linear Regression Adjustment

Linear regression adjustment works well if the relationship between the parameters and the summary statistics is approximately linear, however this is rarely true in practice. Although, approximate linearity may be true in a localized region around s_{obs}. Thus, we can perform *local linear regression*, which applies kernel smoothed (see Section 2.5.2) weights to each $\theta^{(i)}$ based on the distance between $s_{\text{sim}}^{(i)}$ and s_{obs}. That is, we localize the regression problem by defining a kernel $K\left(\left\|s_{\text{sim}}^{(i)} - s_{\text{obs}}\right\|/\epsilon\right)$, where $\|\cdot\|$ is the Euclidean distance and the tolerance ϵ is the bandwidth of the kernel. This guarantees that $\theta^{(i)}$ associated with $s_{\text{sim}}^{(i)}$ close to s_{obs} are weighted more heavily. The kernel can assume many forms, such as the Gaussian kernel:

$$K\left(\frac{\|s_{\text{sim}} - s_{\text{obs}}\|}{\epsilon}\right) = \exp\left(-\frac{1}{2}\left(\frac{\|s_{\text{sim}} - s_{\text{obs}}\|}{\epsilon}\right)^2\right), \tag{3.6}$$

and the Epanechnikov kernel:

$$K\left(\frac{\|s_{\text{sim}} - s_{\text{obs}}\|}{\epsilon}\right) = \begin{cases} 1 - \left(\frac{\|s_{\text{sim}} - s_{\text{obs}}\|}{\epsilon}\right)^2 & \text{if } \|s_{\text{sim}} - s_{\text{obs}}\| \leq \epsilon \\ 0 & \text{otherwise} \end{cases}. \tag{3.7}$$

The weighted least squares estimate for (α, β) is:

$$\left(\hat{\alpha}, \hat{\beta}\right) = \left(X^{\mathrm{T}} W X\right)^{-1} X^{\mathrm{T}} W \theta, \tag{3.8}$$

where W is the $n \times n$ weight matrix whose ith diagonal element is $K\left(\left\|s_{\text{sim}}^{(i)} - s_{\text{obs}}\right\|/\epsilon\right)$ while all other elements are zero:

$$W = \begin{bmatrix} K\left(\frac{\left\|s_{\text{sim}}^{(1)} - s_{\text{obs}}\right\|}{\epsilon}\right) & 0 & \dots & 0 \\ 0 & K\left(\frac{\left\|s_{\text{sim}}^{(2)} - s_{\text{obs}}\right\|}{\epsilon}\right) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & K\left(\frac{\left\|s_{\text{sim}}^{(n)} - s_{\text{obs}}\right\|}{\epsilon}\right) \end{bmatrix}$$

The correction is then done according to Equation 3.5, but with the weighted estimate for β (Equation 3.8).

3.4 Neural Density Estimation

We reviewed some standard methods for *nonparametric* density estimation in Section 2.5. Here, we introduce *neural density estimation* (NDE), a *parametric* density estimation method where the density estimator is an *artificial neural network* (ANN). Since the posterior is a conditional density, we are in particular interested in *conditional neural density estimators*. One class of conditional NDEs is *normalizing flows* (NF), which is a transformation of a simple base distribution, e.g., the normal distribution, into a more complex distribution by a sequence of invertible transformations that are differentiable. NFs were popularized for density estimation by Dinh et al. in [32]. Another general framework for modelling arbitrary conditional densities is mixture density networks (MDNs), see e.g. Bishop [19], which combines a conventional feedforward ANN with a mixture density model of the target (the posterior in our case).

3.4.1 Sequential Neural Posterior Estimation

Sequential Neural Posterior Estimation (SNPE) is a novel algorithm for simulation-based inference based on NDE. Dissecting all of the intricacies of SNPE, and NDE in general, is beyond the scope of this thesis. We will in this section provide an overview of the algorithm(s). For full details, we refer the reader to the original articles [6], [7] and [8].

SNPE targets directly learning the posterior $\pi(\theta \mid y)$ by training a conditional NDE on adaptively proposed simulations of the simulator model. Using adaptive proposals means that the parameters are not drawn from the prior $\pi(\theta)$, but rather a proposal distribution $\tilde{\pi}(\theta)$ that is updated over a number of training rounds. In each round, the proposal is taken to be the approximate posterior distribution itself from the previous round. The idea behind this approach is to increase sample efficiency, as sampling parameters from the prior can lead to wasteful simulations. The use of a proposal $\tilde{\pi}(\theta)$ different from the prior $\pi(\theta)$ does, however, require a correction step, as samples drawn from $\tilde{\pi}(\theta)$ no longer yield the target posterior $\pi(\theta \mid y)$ but rather the *proposal posterior*:

$$\tilde{\pi}(\theta \mid y) = \pi(\theta \mid y)\frac{\tilde{\pi}(\theta)p(y)}{\pi(\theta)\tilde{p}(y)}, \tag{3.9}$$

where $\tilde{p}(y) = \int p(y \mid \theta)\tilde{\pi}(\theta)\,\mathrm{d}\theta$. Note that for $\tilde{\pi}(\theta) = \pi(\theta)$, it directly follows that $\tilde{\pi}(\theta \mid y) = \pi(\theta \mid y)$.

There have been developed three main approaches to the correction step so far, leading to three versions of SNPE; SNPE-A by Papamakarios and Murray [6], SNPE-B by Lueckmann et al. [7] and SNPE-C by Greenberg et al. [8]. All algorithms have in common that they train a neural network $F(y, \phi)$, i.e., a network with weights ϕ that takes as input data y, to learn the parameters of a family of densities q_ψ, where ψ are distribution parameters, to estimate the posterior.

They differ in what is targeted by q_ψ and which loss is used for F. With this notation, $q_{F(y,\phi)}(\theta)$ represents an estimate of $\pi(\theta \mid y)$, and $\tilde{q}_{F(y,\phi)}(\theta)$ an estimate of $\tilde{\pi}(\theta \mid y)$.

SNPE-A trains F, which is required to be a Gaussian MDN in this algorithm, to target the proposal posterior $\tilde{\pi}(\theta \mid y)$ by minimizing the log likelihood loss $\mathcal{L} = -\sum_n \log q_\psi(\theta_n \mid y_n)$. The learned proposal posterior $\tilde{q}_{F(y,\phi)}(\theta)$ is then corrected analytically by dividing it by $\tilde{\pi}(\theta)$ and multiplying it by $\pi(\theta)$. The analytical post-hoc step restricts $\tilde{\pi}(\theta)$ to be Gaussian and $\pi(\theta)$ to be either Gaussian or uniform.

SNPE-B also trains a MDN and deals with the correction step by adjusting the parameters θ_n by assigning them weights $w_n = \pi(\theta_n)/\tilde{\pi}(\theta_n)$. During training SNPE-B minimizes the importance weighted loss $\mathcal{L} = -\sum_n w_n \log q_\psi(\theta_n \mid y_n)$, which allows for direct recovery of $\pi(\theta \mid y)$. Since there is no need for post-hoc correction, SNPE-B lifts the restrictions on $\pi(\theta)$, $\tilde{\pi}(\theta)$ and q_ψ placed by SNPE-A. However, the weights can have high variance, which can lead to slow or inaccurate inference due to high-variance gradients and instability during training.

SNPE-C circumvents the issues of SNPE-A and SNPE-B by reparameterizing Equation 3.9 such that it is possible to automatically transform between estimates of $\pi(\theta \mid y)$ and $\tilde{\pi}(\theta \mid y)$. Since $\tilde{\pi}(\theta \mid y) \propto \pi(\theta \mid y)\tilde{\pi}(\theta)/\pi(\theta)$, SNPE-C defines:

$$\tilde{q}_{y,\phi}(\theta) = q_{F(y,\phi)}(\theta)\frac{\tilde{\pi}(\theta)}{\pi(\theta)}\frac{1}{Z(y,\phi)}, \tag{3.10}$$

where $Z(y,\phi) = \int q_{F(y,\phi)}(\theta)\frac{\tilde{\pi}(\theta)}{\pi(\theta)}\,\mathrm{d}\theta$ is a normalization constant. The network is trained by minimizing the loss $\mathcal{L} = -\sum_n \log \tilde{q}_{y,\phi}(\theta_n)$, such that both the target and proposal posteriors are recovered. SNPE-C supports a wide range of proposals and conditional NDEs, including MDNs and NFs. Its only requirement is a closed form solution of Equation 3.10 for $\tilde{q}_{F(y,\phi)}$, to be optimized during training and sampled from afterwards.

SNPE will in the following refer to the SNPE-C variant. Figure 3.3 gives a conceptual overview of the SNPE algorithm. Perhaps the two most notable features of SNPE is that it uses *all* model simulations to train the network and, once trained, it can be used to predict posteriors over model parameters on any empirical data with just one forward pass through the network.

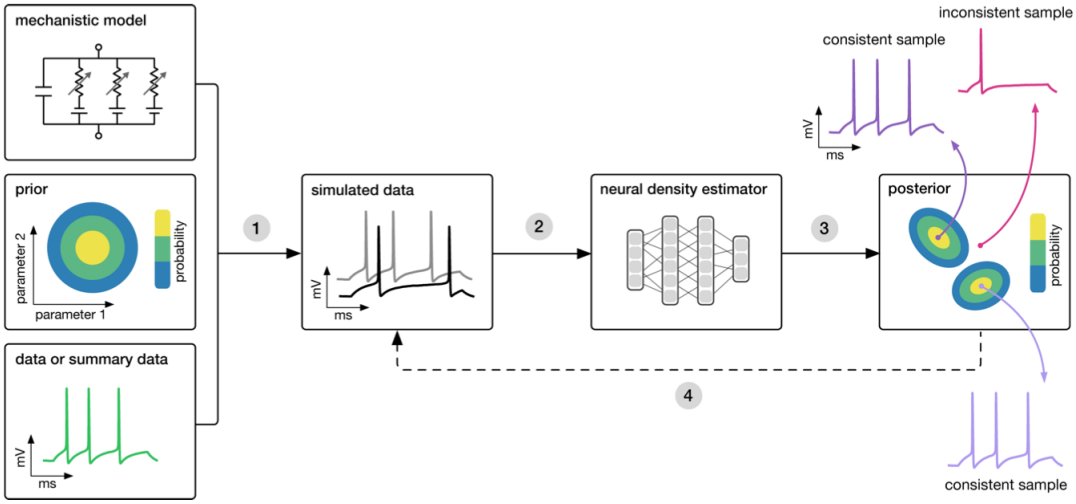


Figure 3.3: A conceptual overview of posterior estimation via the Sequential Neural Posterior Estimation (SNPE) algorithm. SNPE takes three inputs: a simulator of the mechanistic model, priors over model parameters and observed data, either as summary statistics of or the raw data itself. The algorithm then proceeds by: (1) Sample parameters from (i) the priors in the first rounds, and (ii) the proposal distribution in subsequent rounds, and feed them to the simulator to generate simulated data; (2) Train a conditional neural density estimator (a neural network) to learn the probabilistic association between data (or summary statistics of the data) and the underlying parameters; (3) Use the density estimation network to predict the posterior distributions over model parameters consistent with the observed data; (4) Under training, the approximate posterior on simulated data is used to adaptively update the proposal distribution so that more informative simulations become likely.

Source: [33].

4

Introduction to Neurobiology

The aim of this chapter is to give a brief introduction to neurobiology, with focus on the aspects useful for understanding the biological background of the models of neural dynamics presented in the next chapter. The content of this chapter is based on material from [1], [2] and [34].

4.1 Neural Circuits and Networks

The central organ of the human nervous system is the brain, which contains roughly 86 billion nerve cells, or neurons, organized in different brain regions specialized at analyzing different subsets of information encoded in nerve signals. Brain activity is made possible by the interconnections of neurons. A population of neurons interconnected to carry out a specific function is called a *neural circuit*. Neural circuits interconnect to one another to form elaborate *neural networks* that route signals through the different regions of the brain, analyzing and organizing different types of information within fractions of a second. Neurons are either *excitatory* or *inhibitory*. Excitatory neurons are predominant, comprising about 80% of the neurons in the brain, and send signals to neighboring neurons that increase activity. By contrast, inhibitory neurons, comprising about 20% of the neurons in the brain, send signals that suppress the activity of neighboring neurons. Every neural circuit contains both excitatory and inhibitory neurons that work together to regulate the activity in response to stimuli.

4.2 Neurons

The fundamental unit of neural circuits and networks is the neuron, which is a specialized cell that generates electrical signals in response to stimuli and transmits them to other cells. Although neurons come in a wide variety of shapes and sizes, they have common morphological specializations called *dendrites* and *axon*. Like other cells, a neuron consists of a cell body (soma) that contains the neuron's nucleus and molecular machinery critical to the cell's function, and the soma is confined by a cell membrane that consists of a lipid bilayer. The axon and dendrites are filaments that extrude from the soma, as illustrated by Figure 4.1. Dendrites receive incoming signals from other neurons and propagate them to the soma. The branching structure of the dendritic tree allows a neuron to receive signals from many other neurons. The soma processes the collected input signals and generates an output signal if the total input exceeds a certain threshold. The axon carries the output signal and branch out to deliver the signal to other cells through the axon terminals.

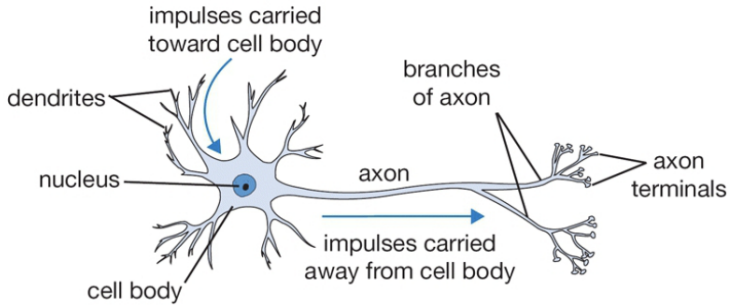


Figure 4.1: Schematic illustration of a neuron. A neuron can be divided into three functionally distinct parts called dendrites, the cell body (soma) and the axon. The dendrites receive signals from other neurons and transmit them to the cell body, which can generate an output signal if the total input exceeds a threshold. The output signal propagates down the axon which branch out and end in axon terminals that deliver the signal to other cells.

Source: [35].

4.3 Ion Channels and Action Potentials

The neuronal membrane is composed of a *lipid bilayer*, that is made up of two layers of lipids, which have their hydrophilic heads pointing outwards and their hydrophobic tails pointing inwards. It is virtually impermeable to water molecules and ions. *Ion channels* are protein pores in the neuronal membrane that allow ions, predominantly sodium (Na^+), potassium (K^+), calcium (Ca^{2+}) and chloride (Cl^-), to flow into and out of the neuron. Most ion channels are permeable only to

specific types of ions, and are either active or passive. Passive channels, also called leakage channels, are always permeable, whereas active channels have gates that can open and close the channel. Some active ion channels are voltage-gated, meaning that whether the channel is in an open or closed state is controlled by voltage, while others are chemically-gated, meaning they open and close by interactions with chemicals. The neuronal membrane also contains membrane-spanning protein structures that actively pump certain types of ions into and out of the neuron, called *ion pumps*. Ions tend to flow from regions of high concentration toward regions of low concentration, thus diminishing the concentration gradient. Ion pumps counteract this by pumping ions against the concentration gradient. For instance, the Na^+– K^+pump exchanges three $Na^+$from inside the cell with two $K^+$from outside using energy provided by hydrolysis of ATP into ADP and a phosphate ion. Figure 4.2 illustrates the constituents of the neuronal membrane.

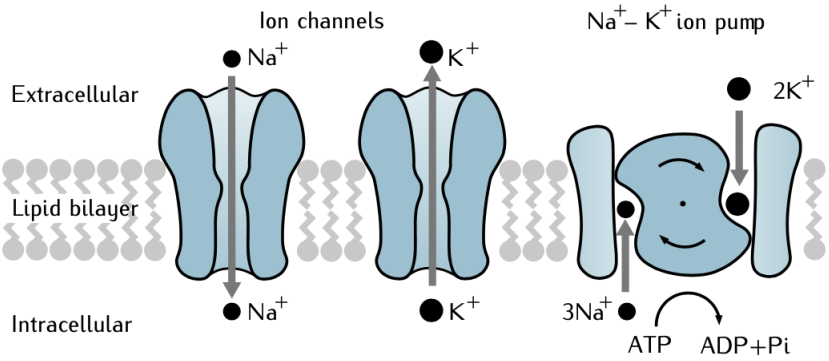


Figure 4.2: Constituents of the neuronal membrane. The lipid bilayer forms a virtually impermeable barrier for inorganic ions. Ion channels form pores in the neuronal membrane, allowing certain ions to flow into and out of the neuron. Ion pumps exchange certain ions across the neuronal membrane.

Source: [2].

Due to the ion channel and pump machinery, there is typically a greater concentration of Na^+in the extracellular space and K^+inside the neuron. Moreover, there are slightly more positive ions on the outside of the neuron and slightly more negative ions on the inside. Hence, there is a small electrical potential difference across the neuronal membrane, called *membrane potential*. The membrane potential of a resting neuron, known as the *resting membrane potential*, typically lies between -60 and $-70\,\mathrm{mV}$, because the potential is more negative inside the neuron than on its outer surface. The membrane potential is affected by signals received from other neurons in its circuit, which can make the membrane potential less negative (depolarized) or more negative (hyperpolarized). If a neuron is depolarized sufficiently to raise the membrane potential above the membrane's threshold voltage, a positive feedback process is initiated which triggers an electrical impulse called an

action potential (AP). The temporal evolution of APs can be divided into different phases, as shown by Figure 4.3. First, stimulus from other neurons connecting to

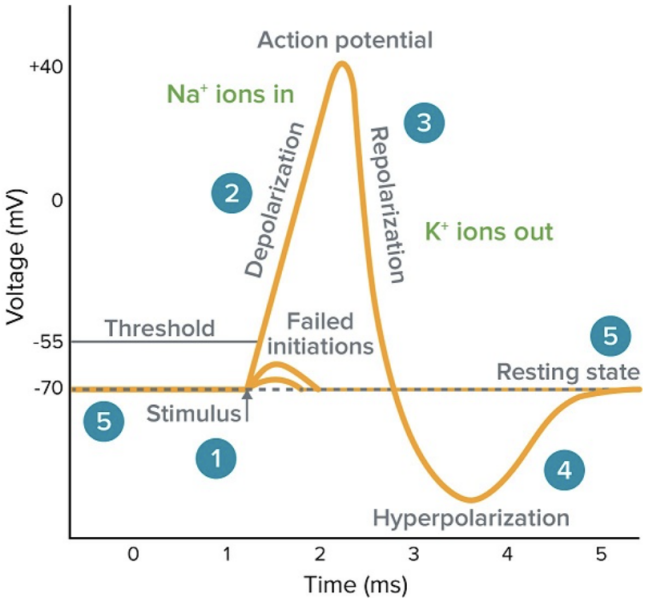


Figure 4.3: The temporal evolution of the membrane potential during a typical action potential. The membrane potential is initially in its resting state when (1) a stimulus pushes it over the threshold. (2) This results in a rapid rise (depolarization) in membrane potential caused by opening of voltage-gated sodium (Na^+) channels that give in an influx of Na^+ions. (3) This is followed by a sharp decrease (repolarization) in membrane potential caused by closing of sodium channels and opening of potassium (K^+) channels that give an efflux of K^+ions. (4) The membrane potential typically undershoots the resting potential (hyperpolarization) before it gradually recovers to the (5) resting state.

Source: [36].

the neuron depolarizes the soma, and increase the membrane potential towards the threshold potential. If the threshold potential is reached, voltage-gated sodium channels open and an influx of sodium ions takes place. This phase is called *depolarization*. During depolarization, the inside of the neuron becomes increasingly electropositive. Once the membrane potential becomes positive, the membrane potential continues to depolarize, commonly referred to as AP *overshoot*, until it reaches the electrochemical equilibrium, known as the *reversal potential*, and peaks. After the overshoot, sodium channels decrease the sodium permeability by closing. The overshoot also opens voltage-gated potassium channels, which cause a potassium efflux that decreases the neuron's electropositivity. This phase is called *repolarization*, and decreases the membrane potential towards the resting

state. Repolarization may be followed by *hyperpolarization*, in which the neuron's membrane potential falls below the resting potential before gradually recovering to the resting potential. The hyperpolarizing phase where the membrane falls below the resting potential is commonly referred to as *afterhyperpolarization* (AHP) or AP *undershoot*.

The resulting AP will propagate along the axon towards the next neurons in the circuit. APs are also known as *spikes*, and a temporal sequence of APs generated by a neuron is called a *spike train*. A neuron that emits an AP is often said to *fire*, and the frequency a neuron emits APs is therefore called the *firing rate* of the neuron. Generation of APs depends on the recent history of the neuron firing. In the brief *absolute refractory period* after an AP, it is virtually impossible to initiate another AP, regardless of the amplitude of the stimulus. For a longer interval known as the *relative refractory period*, the threshold for firing is higher than when the membrane is at rest, and APs initiated in this period have lower peak voltage.

4.4 Synapses

APs are passed from a neuron to next at junctions called *synapses*, which are usually formed between axon terminals on the sending neuron and the dendrites of the receiving neuron, with a space between the two called the *synaptic cleft*. It is common to refer to the sending neuron as the *presynaptic* neuron and the receiving as the *postsynaptic* neuron. The cleft of most synapses is too wide for APs to directly impact the postsynaptic neuron. Instead, chemical signals called *neurotransmitters* cross the synapse. Excitatory neurons release neurotransmitters to increase the activity of the postsynaptic neuron, making it more likely to generate an AP, while inhibitory neurons release neurotransmitters to suppress the activity, making the postsynaptic neuron less likely to generate an AP.

5

Models of Neural Dynamics

Understanding the complex mechanisms of the nervous system requires the construction and analysis of models of neural dynamics at different levels. In this chapter, we discuss the seminal Hodgkin-Huxley model [9] that is a biophysically detailed description of the ionic mechanisms underlying the initiation and propagation of action potentials in squid giant axons. We also consider the Brunel network model [10] for activity dynamics in local cortical networks. The content of this chapter is mainly based on [2], [34] and [37].

5.1 The Hodgkin-Huxley Model

The Hodgkin-Huxley (HH) model was the first quantitative model of active membrane properties, and gave a biophysically detailed description of the mechanisms that give rise to action potentials (APs) in neurons. The model was used to compute the shape of action potentials in the squid giant axon. The model, and the experimental work that led up to it, earned its authors a share of the 1963 Nobel Prize in Physiology or Medicine, establishing a new framework for thinking about the electrical activity of neurons. Before we present the model itself, we first briefly discuss the basics of mathematical modeling of neurons.

5.1.1 Electrical Properties of Neurons

The basis of electrical activity in neurons is the flow of ions into and out of the neuron through ion channels in the neuronal membrane. As ions are electrically charged, they exert forces on and experience forces from other ions. The force acting on an ion is proportional to the ion's charge, q. As we established in the

previous chapter, there is typically an excess negative charge on the inside surface of the neuron and a balancing positive charge on its outer surface. The lipid bilayer forms an insulating barrier between the outside and interior of the neuron. As such, the neuronal membrane behaves as a *capacitor*, described by:

$$q = C_m V, \tag{5.1}$$

where V is the membrane potential and the constant of proportionality C_m the membrane capacitance, which indicates how much charge can be stored on a particular capacitor for a given potential difference across it. All current passing through the membrane either charges or discharges the membrane capacitance, so the rate of change of charge, $\mathrm{d}q\,/\,\mathrm{d}t$, on the membrane is the same as the net current, I, flowing through the membrane: $I = \mathrm{d}q\,/\,\mathrm{d}t$. By differentiating Equation 5.1, we can use the membrane capacitance to determine how much current is required to change the membrane potential at a given rate:

$$C_m \frac{\mathrm{d}V}{\mathrm{d}t} = \frac{\mathrm{d}q}{\mathrm{d}t} = I. \tag{5.2}$$

Equation 5.2 is fundamental in the mathematical modeling of neurons, as it relates how the membrane potential evolves over time with the net flow of current through ion channels. It should be noted that in the context of modeling the entire neuron, C_m is technically the membrane capacitance per unit area. The current I_{X} per unit area through an ion channel of type X is modelled by the quasi-ohmic relation:

$$I_{\mathrm{X}} = g_{\mathrm{X}}(V - \mathrm{E}_{\mathrm{X}}), \tag{5.3}$$

where g_{X} is the *conductance* of the specific ion channel per unit area and E_{X} the reversal potential of ion X. $(V - \mathrm{E}_{\mathrm{X}})$ is called the *driving force*, and when the membrane potential is at the reversal potential for ion X, the driving force is zero. The conductance is a measure of the ease with which an electric current passes, and is the reciprocal of the resistance. The total current per unit area across the neuronal membrane, I, is the sum of the contributions from the different types of ion channels:

$$I = \sum_{\mathrm{X}} g_{\mathrm{X}}(V - \mathrm{E}_{\mathrm{X}}). \tag{5.4}$$

5.1.2 Biophysical Model of Ionic Mechanisms

From a biophysical point of view, APs are the result of ionic currents that pass through the neuronal membrane. In an extensive series of electrophysiology experiments on the squid giant axon, Hodgkin and Huxley succeeded to measure these currents and to describe their dynamics in terms of differential equations.

Hodgkin and Huxley treated the squid giant axon as an equivalent electrical circuit, see Figure 5.1, with the current across the membrane being carried by either a capacitor current, I_{C}, current of potassium ions, I_{K}, current of sodium ions, I_{Na} or a catch-all leakage current, I_{L}.

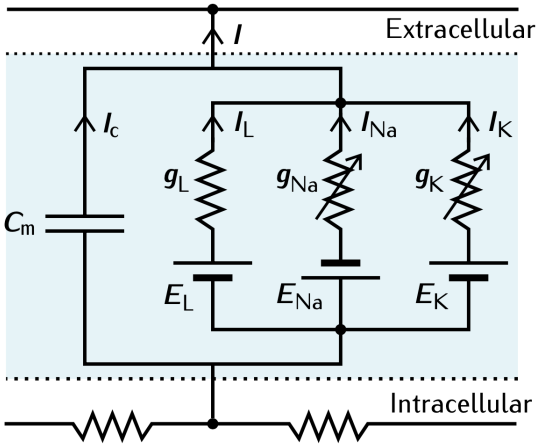


Figure 5.1: The HH equivalent circuit.

Source: [2].

Thus, the fundamental total current equation is:

$$\begin{aligned} I &= I_{\mathrm{C}} + I_{\mathrm{K}} + I_{\mathrm{Na}} + I_{\mathrm{L}} \\ &= C_m \frac{\mathrm{d}V}{\mathrm{d}t} + g_{\mathrm{K}}(V - E_{\mathrm{K}}) + g_{\mathrm{Na}}(V - E_{\mathrm{Na}}) + g_{\mathrm{L}}(V - E_{\mathrm{L}}) \end{aligned} \tag{5.5}$$

In voltage-gated ion channels, the channel conductance g_{X} is a function of both voltage and time, $g_{\mathrm{X}}(V, t)$. We can write the time-dependent conductances like:

$$g_{\mathrm{X}}(V, t) = \bar{g}_{\mathrm{X}} p_{\mathrm{X}}(V, t),$$

where $\bar{g}_{\mathrm{X}}$ is the total conductance when all channels of type X are fully open, i.e., the *maximum conductance*, and $p_{\mathrm{X}}(V, t)$ is the fraction of channels of type X that are open, i.e., the *channel density* (a number between 0 and 1). Hodgkin and Huxley suggested that the opening and closing of the voltage-gated ion channels were controlled by one or more *gating particles*. Using a series of voltage clamp experiments, i.e., where the membrane potential is held at a level determined by the experimenter, and clever ionic substitutions, they were able to isolate the voltage-gated conductances of potassium and sodium. They obtained rate constants for the opening, closing and inactivation of the conductances by analyzing the voltage-dependence using first-order kinetics, and reduced these rate constants to a set of four ordinary differential equations:

$$C_m \frac{\mathrm{d}V}{\mathrm{d}t} = -\bar{g}_{\mathrm{K}} n^4 (V - E_{\mathrm{K}}) - \bar{g}_{\mathrm{Na}} m^3 h (V - E_{\mathrm{Na}}) - \bar{g}_{\mathrm{L}} (V - E_{\mathrm{L}}) + I \tag{5.6a}$$

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \alpha_n(V)(1 - n) - \beta_n(V) n \tag{5.6b}$$

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \alpha_m(V)(1 - m) - \beta_m(V) m \tag{5.6c}$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \alpha_h(V)(1 - h) - \beta_h(V) h \tag{5.6d}$$

where the gating variables n, m and h are dimensionless quantities that are associated with potassium channel activation, sodium channel activation and sodium channel inactivation, respectively. The gating variables n and m range from 0 (not activated) to 1 (fully activated) and h ranges from 0 (fully inactivated) to 1 (no inactivation). The voltage-dependent rate constants $\alpha_x(V)$ and $\beta_x(V)$, where $x \in \{n, m, h\}$, represent the activation and inactivation rates, respectively, for gate x. Equation 5.6 describes how the membrane potential V across a membrane with capacitance C_m responds to an input current I. Note that Equation 5.6a has been slightly reorganized from the formulation of the total current equation (Equation 5.5) to better reflect this. The potassium current, I_{K}, is controlled by four identical gating particles, whereas the sodium current, I_{Na}, is controlled by three identical and one distinct gating particle. The leak current, I_{L}, is not voltage-dependent, and no gating particles are therefore associated with its conductance.

The forms of the α_x and β_x functions were empirically proposed by the authors to fit the experimental recordings, yielding the following equations for the rate constants associated with the potassium activation gating variable;

$$\alpha_n(V) = 0.01 \frac{V + 55}{1 - \exp\left(-(V + 55)/10\right)} \tag{5.7a}$$

$$\beta_n(V) = 0.125 \exp\left(-(V + 55)/10\right) \tag{5.7b}$$

and for the sodium activation and inactivation gating variables:

$$\alpha_m(V) = 0.1 \frac{V + 40}{1 - \exp\left(-(V + 40)/10\right)} \tag{5.8a}$$

$$\beta_m(V) = 4 \exp\left(-(V + 65)/18\right) \tag{5.8b}$$

$$\alpha_h(V) = 0.07 \exp\left(-(V + 65)/20\right) \tag{5.8c}$$

$$\beta_h(V) = \frac{1}{\exp\left(-(V + 35)/10\right) + 1} \tag{5.8d}$$

Here, we use a formulation of the model where the membrane voltage has been reversed in polarity from the original HH convention and shifted to reflect a resting

potential of $-65\,\text{mV}$. An example of how to arrive at the alternative formulation is provided in Appendix B. The original model parameter values are summarized in Table 5.1.

Table 5.1: The original parametrization of the HH model.

Parameter	Value	Description
C_m	$1.0\,\mu\text{F}\,\text{cm}^{-2}$	Membrane capacitance
$\bar{g}_\text{K}$	$36.0\,\text{mS/cm}^2$	Maximum potassium channel conductance
$\bar{g}_\text{Na}$	$120.0\,\text{mS/cm}^2$	Maximum sodium channel conductance
$\bar{g}_\text{L}$	$0.3\,\text{mS/cm}^2$	Maximum leakage channel conductance
E_K	$-77.0\,\text{mV}$	Potassium reversal potential
E_Na	$50.0\,\text{mV}$	Sodium reversal potential
E_L	$-54.4\,\text{mV}$	Leak reversal potential

5.1.3 Simulation of Action Potentials

Figure 5.2 shows the numerical solutions of a simulation of the HH model. As seen in the top panel, the shape of the simulated action potentials match the description of an action potential (see Section 4.3) well. Besides reproducing action potentials, the HH model offers insights into the mechanisms underlying them. The middle panel shows how the gating variables change during the temporal evolution of the action potentials. At stimulus onset (starting at $t = 10\,\text{ms}$), the initial depolarization of the membrane potential is due to the input current. When the depolarization is above the threshold (at about $-55\,\text{mV}$), the sodium current activates, as reflected in the increase in m. As the sodium reversal potential is far higher than the resting membrane potential, the driving force of the sodium current pushes the membrane potential to sharply increase. The slower potassium conductance, reflected by the gating variable n, activates after the sharp rise in membrane potential and allows potassium ions to flow out of the neuron because of the low potassium reversal potential. In addition, the repolarization of the membrane potential is also assisted by the inactivating sodium gating variable, h, which shuts off the sodium current. This drives the membrane potential quickly back down towards its resting state, but undershoots somewhat, due to the slow de-inactivation of the sodium current, to hyperpolarize the neuron. The final recovery involves a rapid deactivation of the sodium current and a slower deactivation of the potassium current. Eventually all the state variables and the membrane potential return to their resting states. The HH model also explains the refractory period. Relative to the duration of an action potential, the gating variables recover to their resting states slowly. During this period, it is harder to generate an action potential. In the initial recovery phase, an increasing voltage will not increase the sodium conductance, and hence the membrane potential, considerably due to the ongoing inactivation of the sodium current and the prolonged activation of the potassium

current. As the state variables advances in their recovery toward the resting states, an action potential can be initiated but will have a lower peak voltage.

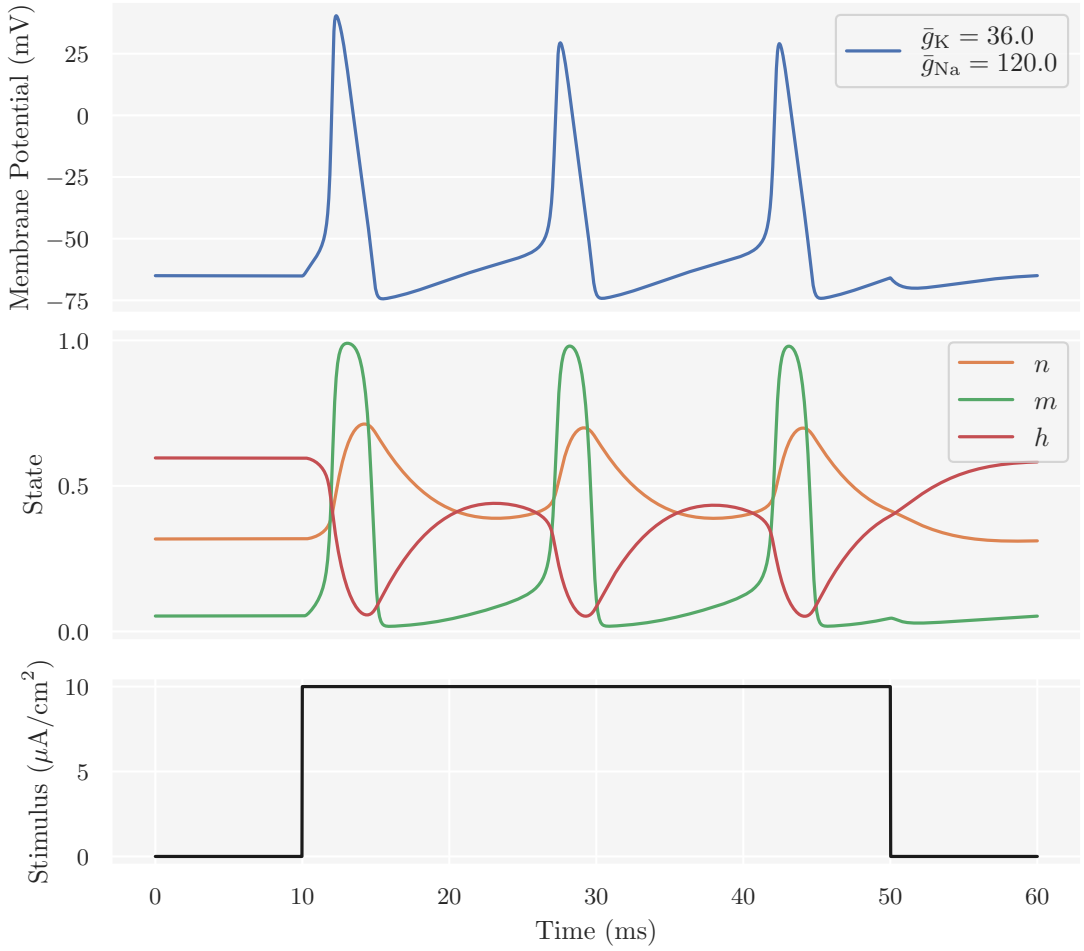


Figure 5.2: Simulated dynamics of V, n, m and h in the HH model during the firing of action potentials in the squid giant axon. The top panel shows the numerical solution of Equation 5.6a and the middle panel the numerical solutions of Equations (5.6b) to (5.6d). The system is simulated for $T = 60\,\text{ms}$ with time resolution $\Delta t = 0.025\,\text{ms}$. The input stimulus received by the neuron (bottom panel) is a step current with amplitude $I = 10\,\mu\text{A/cm}^2$, and onset and offset at 10 ms and 50 ms, respectively. The parametrization of the model is given by Table 5.1.

5.2 The Brunel Network Model

Many neural networks of interest consist of thousands or millions of neurons, and, generally, it is infeasible to include all in the model. Neural network models are usually scaled down according to the ratio of different neuron populations in the

network that the model tries to mimic. Furthermore, they often use simplified neuron models to reduce computational cost. Still, neural network models may exhibit a high diversity of spiking dynamics. In this section, we present one such model that is thoroughly analyzed in the literature; the Brunel network model [10]. Before we present the network model itself, we first discuss the central building block of the network: the leaky integrate-and-fire (LIF) neuron model.

5.2.1 Integrate-And-Fire Neurons

While the ionic mechanisms behind APs are quite complicated, the conditions for AP generation are often quite straightforward: When the membrane potential reaches a specific threshold, a spike is generated and the membrane potential returns to the background state. Simulations of APs can be accelerated significantly by not explicitly modeling the responsible biophysical mechanisms. *Integrate-and-fire* (IF) models are simplified neuron models with a spike generation and reset mechanism. In these models, whenever the membrane potential of a neuron reaches a threshold value θ, a spike is generated and the membrane potential is reset to a value V_{reset} below the threshold potential, $V_{\text{reset}} < \theta$. In the simplest IF model, all active membrane conductances are ignored and the entire membrane conductance is modeled as a single passive leakage conductance:

$$I_{\text{L}} = \bar{g}_{\text{L}}(V - E_m),$$

where E_m is the resting membrane potential. Since the conductance is the reciprocal of the resistance, the above equation can be equivalently formulated as:

$$I_{\text{L}} = \frac{V - E_m}{R_m},$$

where R_m is the membrane resistance. Furthermore, we assume that the model neuron behaves like an electric circuit consisting of a resistor and a capacitor in parallel, i.e., an RC circuit, driven by a current I. In addition, the circuit needs a switch, representing the reset mechanism, which is open until the membrane potential reaches θ and then closes to short-circuit the membrane resistance, bringing the membrane potential back to rest. The switch opens again after a refractory period τ_{rp}, allowing the membrane to charge. This neuron model is often called the *leaky integrate-and-fire* (LIF) model. The circuit diagram and the RC circuit response to input stimulus is shown in Figure 5.3. When the membrane potential is below the threshold, its value is determined by the equation for an RC circuit:

$$C_m \frac{\mathrm{d}V}{\mathrm{d}t} = -\frac{V - E_m}{R_m} + I.$$

The above equation is usually written in terms of the membrane time constant, $\tau_m = C_m R_m$:

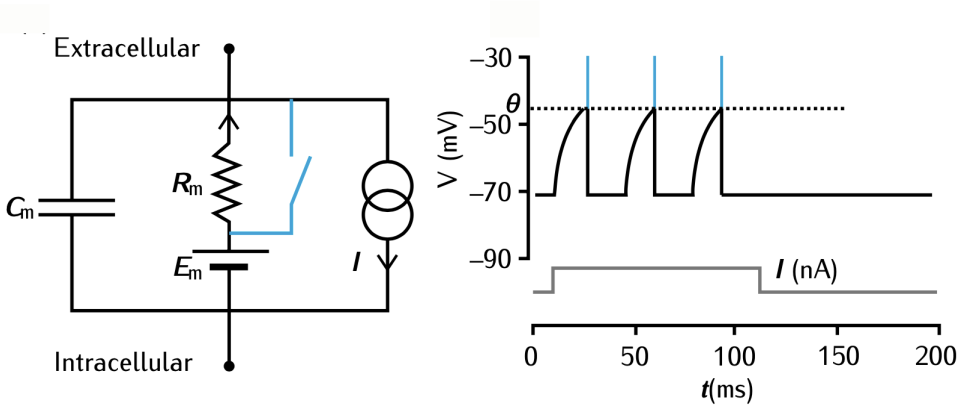


Figure 5.3: The LIF neuron model as an RC circuit diagram (left) with a switch (in blue on the diagram) and the response of the LIF neuron to input stimulus (right). When the switch is open, the membrane can charge. When the membrane potential reaches the threshold θ (indicated by the dashed line in the voltage trace), the neuron fires a spike (indicated by the vertical blue line in the voltage trace) and the switch closes. This short-circuits the membrane resistance, bringing the membrane potential back to its resting state. After a refractory period, the switch opens, allowing the membrane to charge again.

Source: [2].

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -V + E_m + R_m + I.$$

Furthermore, the leak battery is often omitted from the circuit, with the only effect of making the resting membrane potential 0 mV instead of E_m:

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -V + R_m + I. \tag{5.9}$$

The simple LIF neuron model only captures the timing of each spike, but is fast to simulate compared to biophysically detailed neuron models. This makes it especially useful for simulating large networks, as these often contain thousands of neurons.

5.2.2 A Sparsely Connected Recurrent Network

The local cortical network consists of a population of excitatory neurons and a population of inhibitory neurons, with a ratio of about 80% excitation and 20% inhibition. The Brunel model characterizes the local cortical network as a network of N identical LIF neurons, from which N_E are excitatory and $N_I = N_E/4$ inhibitory.

Each neuron, be it excitatory or inhibitory, receives C randomly connections from other neurons in the network, from which $C_E = \epsilon N_E$ are from the excitatory population and $C_I = \epsilon N_I$ from the inhibitory population. Here, ϵ denotes the fraction of incoming connections, and we consider a sparsely connected network with $\epsilon = C_E/N_E = C_I/N_I << 1$. In addition to the sparse recurrent inputs from within the local network, each neuron receives excitatory synaptic input from a population of C_E randomly firing neurons outside the network with activation governed by identical, independent Poisson processes (PGs) with fixed-rate ν_{ext}. The randomly firing population mimics input from the rest of cortex. An illustration of the network is shown in Figure 5.4.

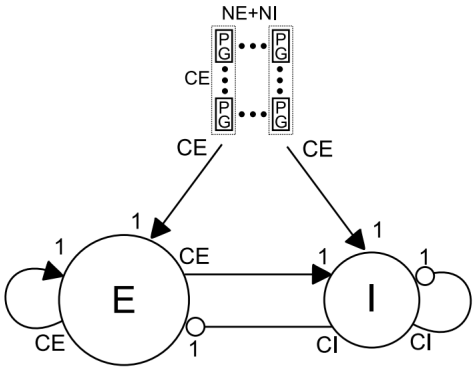


Figure 5.4: Illustration of the Brunel network model. The network consists of two local populations, one with N_E excitatory neurons (circle labeled E) and one with N_I inhibitory neurons (circle labeled I), and one external population of identical, independent Poisson processes (PGs). The connections between network nodes are indicated by arrows, where triangular arrow-heads represent excitatory and round arrow-heads inhibitory connections. The numbers at the start and end of each arrow indicate the multiplicity of the connection.

Source: [38].

The subthreshold dynamics of LIF neuron i in the network ($i = 1, ..., N$) evolves in time according to:

$$\tau_m \frac{\mathrm{d}V_i(t)}{\mathrm{d}t} = -V_i(t) + R_m I_i(t), \tag{5.10}$$

where I_i are the synaptic inputs arriving at the soma. These synaptic inputs are the sum of spike contributions from both local and external synapses, and are modeled as δ-current inputs, i.e., discontinuous voltage jumps:

$$R_m I_i(t) = \tau_m \sum_j J_{ij} \sum_k \delta\Big(t - t_j^k - D\Big), \tag{5.11}$$

where the first sum is over all the presynaptic neurons j with postsynaptic potential amplitude (voltage jump) J_{ij}, while the second sum is over the spike times of those neurons. Here, D is the synaptic delay, $\delta(x)$ the Dirac δ function, with $\delta(x) = 0$ for $x \neq 0$ and $\int_{-\infty}^{\infty} \delta(x)\,\mathrm{d}x = 1$, and t_j^k represents the emission time of the kth spike of presynaptic neuron j. For simplicity, we assume the synaptic connection strengths are constant for each population. We let $J_{ij} = J > 0$, and for excitatory neurons and external input $J_E = J$, while for inhibitory neurons $J_I = -gJ_E$, where g determines the relative strength of the inhibitory synapses compared to the excitatory synapses. The amount of input the local neurons receive from the external population is determined by the parameter $\eta = \nu_{\text{ext}}/\nu_{\text{thr}}$, where $\nu_{\text{thr}} = \theta/(J_E C_E \tau_m)$ is the minimum constant rate input needed for a neuron to reach threshold in absence of feedback. Thus, the external input rate is given by $\nu_{\text{ext}} = (\eta\theta)/(J_E C_E \tau_m)$.

When the membrane potential $V_i(t)$ of LIF neuron i reaches the firing threshold θ, the neuron fires a spike, the synapses onto all its postsynaptic neurons are activated after a time delay D and the neuron's membrane potential is clamped to the reset potential V_{reset} for a refractory period τ_{rp}.

5.2.3 States of Spiking Activity

The Brunel network may be in several different states of spiking activity, largely dependent on the values of the synaptic weight parameters. In the context of a biological neural network, synaptic weight parameters refer to parameters that determines the influence the firing of one neuron has on another. With particular fixed values for J and D and varying values for η and g, Brunel [10] characterized the spiking activity of the network by phase diagrams. An example of these is shown in Figure 5.5. The spiking activity can be in a state of synchronous regular (SR), asynchronous irregular (AI) or synchronous irregular (SI), with either fast or slow oscillations, activity. It can be seen from the phase diagram that $g = 4$ corresponds to balance between excitation and inhibition, while $\eta = 1$ corresponds to external input, in the absence of recurrent input from the network, just sufficient to reach the firing threshold. Stability of the AI state breaks down at the dashed or solid lines and can lead to SR or SI (either fast or slow) activity.

Figure 5.6 illustrates an example of each of the states of the Brunel network. For each state, the figure shows the firing times (rasters) of 20 randomly chosen excitatory neurons, the temporal evolution of the activity in the network (time resolved firing rate computed in bins of 10 ms) together with the mean firing rate (horizontal axis line), and the pairwise Pearson's correlation coefficient matrix (described in the next chapter) of the recorded neurons. The particular values for η and g are stated in the subplot titles. In the SR state, the network is almost fully synchronized and the neurons fire at high rates. It is characterized by fast periodic oscillations of the spiking activity. In the AI state, neurons fire mostly independently at low rates. It is characterized by that neurons in the population fire at different times (*asynchronous firing*) and at irregular intervals. In the SI states,

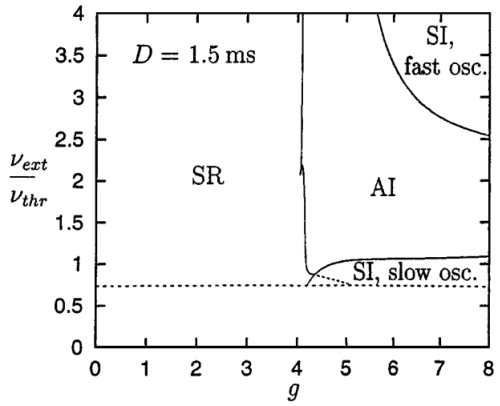


Figure 5.5: Phase diagram of different network states which arise depending on the parameters $\eta = \nu_{\text{ext}}/\nu_{\text{thr}}$ and g. In the present example, a fixed synaptic delay $D = 1.5\,\text{ms}$ and voltage jump (amplitude of excitatory synaptic input currents) $J_E = 0.1\,\text{mV}$ is used. The simulation is of a network consisting of $N_{\text{E}} = 10,000$ excitatory and $N_{\text{I}} = 2,500$ inhibitory neurons with connection probability $\epsilon = 0.1$, which corresponds to a network where each neuron has $C_{\text{E}} = 1,000$ and $C_{\text{I}} = 250$ randomly selected connections to excitatory and inhibitory neurons, respectively. The phase diagram shows four states of spiking activity: synchronous regular (SR), asynchronous irregular (AI) and of synchronous irregular (SI), fast and slow.

Source: [10].

the spiking activity of the network is characterized by either fast or slow synchrony, but individual neurons fire irregularly. It can be traced back to an instability of the AI firing regime towards oscillatory activity. The pairwise Pearson's correlation coefficient measures the correlation between spike trains of two neurons in the network, and can be used to examine how synchronous the spiking of a network is. We see that in the SR state, the correlation coefficients are much higher than in the AI state. The SI state with fast oscillations, however, is only scarcely more synchronous than the AI state. The SI state with slow oscillations, on the other hand, has increased synchrony compared to the AI state.

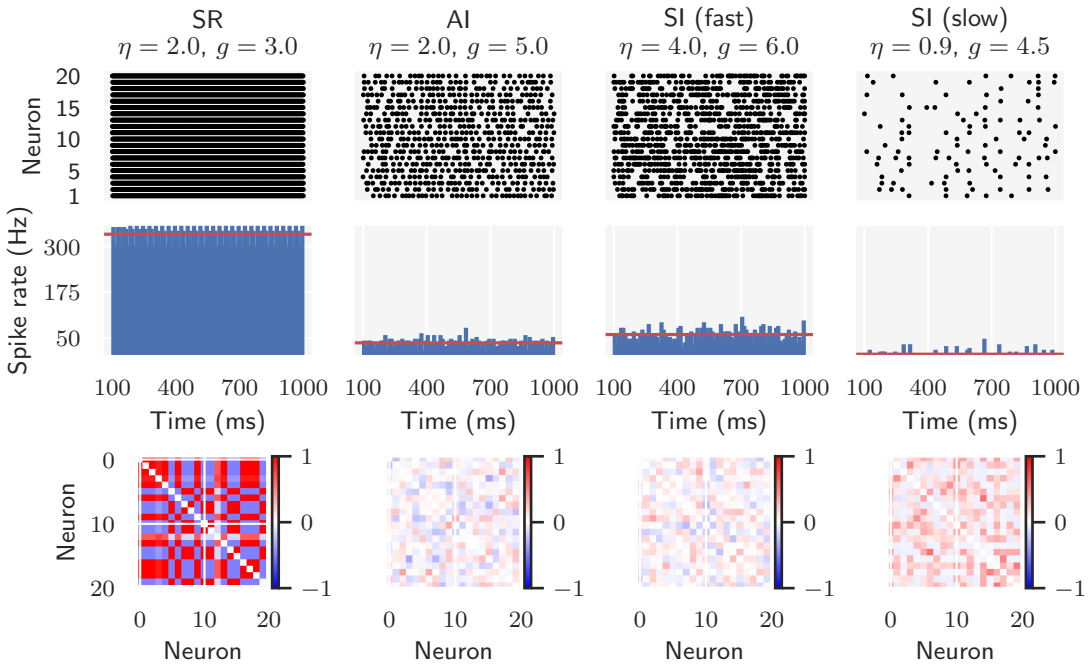


Figure 5.6: Simulation of the network specified by Table 5.2, with the values of η and g stated in each subplot's title along with the network state. The network is simulated for $T_{\text{sim}} = 1,000$ ms, and we record the output from $N_{\text{rec}} = 20$ excitatory neurons. To avoid transient effects, we start recording after $T_{\text{transient}} = 100$ ms. The top row shows the firing times (raster) of the recorded neurons. Each point in the raster corresponds to the firing of a neuron. The second row shows the network activity as a time resolved firing rate computed in bins of 10 ms. The mean firing rate is indicated by the horizontal (red) axis line. The third row shows the pairwise Pearson's correlation coefficient matrix of the recorded neurons.

Table 5.2 summarizes the parametrization of the Brunel model used in the above simulations.

Table 5.2: The parametrization of the Brunel model. Specific values are not provided for the parameters derived from the varying η and g.

Parameter	**Value**	**Description**
N	$12,500$	Total number of neurons
N_E	$10,000$	Number of excitatory neurons
N_I	$2,500$	Number of inhibitory neurons
ϵ	0.1	Connection probability
C_E	$1,000$	Excitatory synapses per neuron
C_I	200	Inhibitory synapses per neuron
E_m	$0\,\mathrm{mV}$	Resting membrane potential
C_m	$1\,\mathrm{pF}$	Membrane capacitance
τ_m	$20\,\mathrm{ms}$	Membrane time constant
θ	$20\,\mathrm{mV}$	Firing threshold
V_{reset}	$10\,\mathrm{mV}$	Reset membrane potential
τ_{rp}	$2\,\mathrm{ms}$	Refractory period
D	$1.5\,\mathrm{ms}$	Synaptic delay
J_E	$0.1\,\mathrm{mV}$	Excitatory synapse strength
J_I	$-gJ_E$	Inhibitory synapse strength
ν_{ext}	$(\eta\theta)/(J_E C_E \tau_m)$	External firing rate

Part II

Methodology & Computational Approach

6 Methodology

6.1 Outline of Analyses

Here, we provide an outline of the analyses that will be carried out, in order to motivate the following methodologies and give the reader an overview of what is to come. We keep this brief, as we will reiterate the objectives and expand on details as we go along.

The overall objective of this thesis is to investigate the ability and utility of simulation-based inference (SBI) for identifying parameters in mechanistic models of neural dynamics. Specifically, we will investigate the performance of approximate Bayesian computation (ABC) using rejection sampling with post-sampling regression adjustment and the neural density estimation (NDE) algorithm SNPE (see Chapter 3). The primary focus will be on ABC, and SNPE will only be used for comparison.

For the assessment of the strengths and weaknesses of SBI, we mainly use the original Hodgkin-Huxley (HH) model for the potassium (K^+), sodium (Na^+) and leakage channels found in the squid giant axon membrane (see Section 5.1). The objective of the inferential task on the HH model is to identify the maximum conductance of the K^+channel, $\bar{g}_{\mathrm{K}}$, and the maximum conductance of the Na^+channel, $\bar{g}_{\mathrm{Na}}$. As such, the remaining parametrization will be kept fixed according to their original values, as tabulated in Table 5.1. The observed data will primarily be a synthetically generated voltage trace recording, free of any noise, in order to not have the results overshadowed by noisy data. We will, however, also investigate the impact a noisy recording has on the inference, as real-world neural data are quite noisy.

Much effort in computational neuroscience today concerns mechanistic models at the network level. We also consider the Brunel network model for activity dynamics in local cortical networks (see Section 5.2). Here, the inferential task will be to identify the synaptic weight parameters η and g. The remaining parametrization of the Brunel model will be as given in Table 5.2. For the current study, we primarily limit our analysis to infer the parameters in the asynchronous irregular (AI) state. However, we will try to utilize the flexibility of SNPE by training on simulations from both the AI and synchronous regular (SR) state, to investigate whether the predicted posteriors, when using observed data from one of these states, match the expected parameter ranges from the phase diagram (Figure 5.5).

We broadly divide the analyses in two parts, one part concerning the inferential task on the HH model and the other the Brunel network model. In the introduction, we divided the overall objective into six parts:

1. Implement simulators for both the Hodgkin-Huxley and Brunel network model in Python.
2. Implement a general ABC rejection sampler with post-sampling regression adjustment in Python.
3. Determine suitable summary statistics of the spiking activity using domain knowledge and develop or find methods for extracting them from the simulated neural data.
4. Assess how well the summary statistics constrain the model parameters by examining sensitivity through a correlation analysis. Based on the correlation analysis, implement an importance weighting procedure for the statistics.
5. Estimate the model parameter posteriors with both ABC and SNPE by using synthetic observed data generated by the simulators, and examine the performance of the simulation-based inference approach.
6. Compare the results obtained via ABC and SNPE and insights they might provide about the neuroscientific models.

6.2 Summary Statistics of Spiking Activity

The choice of summary statistics is vital in determining the outcome of the inverse modelling with SBI, particularly with ABC algorithms. In the following, we present summary statistics of spiking activity based on domain knowledge.

6.2.1 Spike Statistics

In order to characterize the voltage response of a HH model neuron to depolarizing stimulus, we project the response to low-dimensional summary statistics related to action potential (AP) shape and firing behavior. We will use the spike statistics suggested by Druckmann et al. [39]:

(i) **Spike rate** – calculated as the number of spikes divided by the duration of the stimulus;

(ii) **Average AP overshoot** – calculated by averaging the absolute peak voltage of all APs;

(iii) **Average AP width** – calculated by averaging the width of every AP at the midpoint between its onset and its peak;

(iv) **Average AHP depth** – calculated by averaging all minima voltage throughs, i.e., afterhyperpolarization (AHP) depths, between two consecutive APs;

(v) **Latency to first spike** – calculated as the time between stimulus onset and first AP peak;

(vi) **Accommodation index** – calculated as the normalized difference in length of two consecutive interspike intervals (ISIs), i.e., the time between subsequent APs:

$$A = \frac{1}{N-k-1} \sum_{i=k}^{N} \frac{\mathrm{ISI}_i - \mathrm{ISI}_{i-1}}{\mathrm{ISI}_i + \mathrm{ISI}_{i-1}}, \tag{6.1}$$

where N is the number of spikes and k determines the number of ISIs that will be disregarded to protect against possible transient behavior. The value for $k = \min(4, N_{\mathrm{ISI}}/5)$, where N_{ISI} is the total number of ISIs.

6.2.2 Spike Train Statistics

From the Brunel network, we record the spike trains from multiple neurons. Spikes are events characterized by their firing time t^k, where $k = 1, 2, ...$ labels a spike by the spike count. We define the spike train of a neuron i as the sequence of firing times:

$$S_i(t) = \sum_k \delta\left(t - t_i^k\right), \tag{6.2}$$

where $\delta(x)$ is the Dirac δ function (defined in Section 5.2.2). Thus, spikes are reduced to points in time. In a population of N neurons, we calculate the proportion of active neurons by counting the number of spikes in a small time interval Δt and dividing by N. Further division by Δt yields the population activity:

$$\nu(t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \frac{\int_t^{t+\Delta t} \sum_i \sum_k \delta\left(t - t_i^k\right) \mathrm{d}t}{N} \tag{6.3}$$

In practice, we usually have to bin the spike trains in bins of Δt to obtain the time resolved firing rate.

To characterize the network activity, we will use three summary statistics that aim to capture different aspects of the activity:

(i) **Mean firing rate**. We characterize the mean firing rate of the network as a time and population averaged firing rate. This is calculated by first determining the average firing rate of each single spike train by counting its spikes and dividing by a time window, then the population average is found by averaging over all the recorded neurons.

(ii) **Mean CV**. The regularity of spike trains can be summarized by the spike interval statistic *coefficient of variation* (CV), defined as the standard deviation of the ISIs divided by their mean. The mean CV is calculated by averaging the CV of each neuron's ISIs over all recorded neurons. A regularly spiking neuron would have CV of 0, since there is no variance in the ISIs, whereas a Poisson process has a CV of 1.

(iii) **Fano factor**. The Fano factor is a statistic across spike trains that measures the variability. It is defined as the variance-to-mean ratio of spike counts in a time window. The Fano factor is typically computed for spike trains representing the activity of the same neuron over different trials. However, since all neurons in the network have identical properties, we can think of the activity of each recorded neuron as a single trial.

6.3 Correlation Analysis & Importance Weights

Some summary statistics may carry more information about a model parameter than others. As the ABC methodology is based on comparing simulated and observed summary statistics, the most informative summary statistics will typically be those with higher variability relative to movement of model parameter values. If we weight the summary statistics in accordance with the variability they exhibit relative to changes in model parameter values, the inferential algorithm might be able to constrain the model parameter better. The notion of weighting the summary statistics in this manner can be said to be a form of *importance weighting*. We would then give larger weights to the summary statistics that are most sensitive to changes in model parameter value and smaller weights to those less sensitive.

There are numerous robust approaches to base the construction of importance weights on, for instance parameter sensitivity analysis [40] or analysis of curvature of an objective function [39]. We will, however, develop a rather simplistic approach where the importance weights are constructed based on correlation analysis. We landed on this approach simply because of the limited time available for carrying out a master project, and this aspect is of lesser importance than others in the study. Nevertheless, the idea behind basing the importance weights on correlation analysis follows.

Correlation analysis is a method to measure the strength of the linear relationship between the relative movements of two variables X and Y. A common measure is

the pairwise Pearson's correlation coefficient r, which is defined as the ratio between the covariance of the variables and the product of their standard deviations:

$$r = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}. \tag{6.4}$$

As r essentially is a normalized measurement of the covariance, the magnitude of the correlation will be a value between -1 and 1, where the sign indicates the direction of the relationship. A high correlation, i.e., when r is close to 1 or -1, indicates a strong relationship, while $r = 0$ points to no relation between the variables.

Correlation analysis, in particular examination of the pairwise Pearson's correlation coefficient matrix, can be used to assess sensitivity by examining which model parameters contribute the most variability to the summary statistics. Thus, correlation analysis may indicate which of the summary statistics might constrain the model parameters the best, though one should keep the assumption of linearity in mind. In practice, assessing the sensitivity via a correlation analysis requires us to perform a pilot study where we sample parameters from the prior predictive distribution, given by Equation 2.3, and generate the corresponding summary statistics with the simulator model.

Furthermore, we can use the results from the correlation analysis to construct importance weights for the summary statistics. Given a vector of summary statistics $s = (s_1, ..., s_m)$ and a vector of model parameters $\theta = (\theta_1, ..., \theta_l)$, the following procedure will generate a vector of importance weights $w = (w_1, ..., w_m)$:

1. Given paired samples $\left\{\left(\theta_k^{(1)}, s_i^{(1)}\right), ..., \left(\theta_k^{(n)}, s_i^{(n)}\right)\right\}$ consisting of n pairs where we let $\left(\theta_k^{(j)}, s_i^{(j)}\right)$ indicate the jth sample, we first compute the Pearson correlation coefficient as:

$$r_{i,k} = \frac{\sum_{j=1}^{n}\left(\theta_k^{(j)} - \bar{\theta}_k\right)\left(s_i^{(j)} - \bar{s}_i\right)}{\left[\sum_{j=1}^{n}\left(\theta_k^{(j)} - \bar{\theta}_k\right)^2 \sum_{j=1}^{n}\left(s_i^{(j)} - \bar{s}_i\right)^2\right]^{1/2}},$$

   where $\bar{\theta}_k$ and $\bar{s}_i$ are the sample means of the kth model parameter and the ith summary statistic, respectively.

2. The squared Pearson correlation coefficient, $r_{i,k}^2$, indicates the proportion of variance in s_i that is accounted for by (or shared with) θ_k. By definition, $r_{i,k}^2$ will be a number between 0 and 1 that can be used to weight the summary statistics. The summary statistics most sensitive to model parameter movements, will in this way be given a larger weight. Thus, we set the importance weight for ith summary statistic for the kth model parameter as:

$$w_{i,k} = r_{i,k}^2.$$

3. Since the ABC algorithms do not facilitate comparison of summary statistics for individual model parameters, we need to average over all model parameters to obtain the importance weight of the ith summary statistic:

$$w_i = \frac{1}{l} \sum_{k=1}^{l} w_{i,k}.$$

4. After obtaining all weights, we ensure that $\sum_{i=1}^{m} w_i = 1$ by setting each $w_i = w_i / \sum_{i=1}^{m} w_i$.

6.4 Configuration of ABC Algorithm

Configuring an ABC algorithm for inference requires some choices. In particular, for the rejection ABC algorithm we need to set priors over the model parameters, select a discrepancy metric and a tolerance. In this section we discuss the particular choices we will make.

6.4.1 Choice of Priors

In practice, the choice of priors over unknown model parameters is a study in and of itself. Mimicking such a choice is beyond the scope of this thesis. For the present inferential tasks, the choice of priors will regardless be artificial since we actually know the ground truths. For most inferences, we will therefore use noninformative priors to demonstrate the accuracy of the methods based on data alone. For the HH model parameters, we will use priors with about $\pm 10\%$ range around the ground truth parameters. We will, however, also investigate the effect slightly more informative priors have on the convergence of the posterior with the HH model. For the Brunel model, we will only use noninformative priors. For a given observed state, the prior ranges will be set according to the corresponding ranges in the phase diagram of network states (Figure 5.5).

6.4.2 Discrepancy Metric

In ABC algorithms, each simulation is converted to a vector of summary statistics $s_{\text{sim}} = \left(s_{\text{sim}}^{(1)}, s_{\text{sim}}^{(2)}, ..., s_{\text{sim}}^{(m)}\right)$. We need to define a discrepancy metric that compares each of the simulated statistics in s_{sim} to the corresponding ones in the vector of observed summary statistics s_{obs}. As discrepancy metric, we will use the Euclidean distance:

$$\rho(s_{\text{sim}}, s_{\text{obs}}) = \|s_{\text{sim}} - s_{\text{obs}}\|_2 = \left[\sum_{i=1}^{m} \left(s_{\text{sim}}^{(i)} - s_{\text{obs}}^{(i)}\right)^2\right]^{1/2}.$$

As illustrated by Figure 3.1, the Euclidean distance amounts to a circular acceptance region, which implies identical scales of the summary statistics. However, the

summary statistics of spiking activity tend to be on quite different scales, and we will thus be in danger of comparing apples with oranges. The summary statistics with largest scales can dominate any distance calculation unless we normalize the summary statistics so that they vary roughly over the same scale. Scaling the summary statistics in such a manner can be achieved by using a weighted Euclidean distance:

$$\rho(s_{\text{sim}}, s_{\text{obs}}) = \left[\sum_{i=1}^{m} \left(\frac{s_{\text{sim}}^{(i)} - s_{\text{obs}}^{(i)}}{\sigma^{(i)}} \right)^2 \right]^{1/2},$$

where $\sigma^{(i)}$ is an estimator of the ith summary statistic scale. In practice, we need to sample parameters from the prior predictive, feed the parameters to the simulator model and then calculate the empirical scale from the resulting summary statistics. We will scale the summary statistics according to their standard deviation (SD) estimated from the prior predictive samples. We could have chosen e.g. the median absolute deviation (MAD) as scale instead, which is a more robust estimator of scale than SD. However, if more than 50% of the prior predictive samples for a particular summary statistic have identical values, MAD will equal zero. We therefore opt for the more reliable SD as scale.

To extend the above distance metric to include the importance weighting of the summary statistics, the distance metric will be on the form:

$$\rho(s_{\text{sim}}, s_{\text{obs}}) = \left[\sum_{i=1}^{m} w^{(i)} \left(\frac{s_{\text{sim}}^{(i)} - s_{\text{obs}}^{(i)}}{\sigma^{(i)}} \right)^2 \right]^{1/2}, \tag{6.5}$$

where $w^{(i)}$ is the importance weight of the ith summary statistic. If all $w^{(i)} = 1$, then the summary statistics are equally weighted. We will use the Euclidean distance on the particular form given by Equation 6.5.

6.4.3 Semi-Automatic Tolerance Selection

Determining the tolerance parameter ϵ in the ABC algorithms can be quite finicky, as we usually do not know in advance exactly what a reasonable cutoff might be. Conceptually, it is easier to require the algorithm to accept some small proportion of the simulations rather than setting ϵ by hand. If we define the tolerance as the q-quantile of the distances from n simulations, we avoid manually setting ϵ. With this quantile-based rejection scheme, defining the tolerance as the 0.5-quantile of the distances amounts to accepting 50% of the simulations, the 0.3-quantile 30% of the simulations etc.

6.5 Performance Metrics

Choice of suitable performance metrics are central to any analysis. We discussed methods for summarizing posteriors in Section 2.6. Using KDE (see Section 2.5.2) to create a visual representation of obtained posterior samples is a useful first-step, as KDE plots quickly inform us about the shapes and locations of the posteriors. In addition, we should use numerical summaries of the posterior. For each posterior over a model parameter we will both indicate in the KDE plot and provide the value(s) of the MAP estimate and the 95%-HDI. We will also perform posterior predictive checks (PPCs).

While all of these summaries enable us to assess the goodness of fit, we will also take advantage of the fact that we have access to the ground truth parameters. Just comparing a point estimate with a ground truth is not a particularly good error measure, since this will not account for the width of the posterior. That is, a wide posterior and a narrow posterior might have nearly identical point estimates close to the ground truth, but the narrow posterior will then clearly be more accurate. We will therefore define an error measure that takes the width of the posterior into account. By definition, there will be more posterior samples in the regions of high density in the KDE representation of the posterior. Thus, by averaging over all the posterior samples, we obtain an implicitly weighted error estimate where the width is accounted for. We will use the *root-mean-square percentage error* (RMSPE) as performance metric, defined as:

$$\text{RMSPE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\theta_{\text{true}} - \hat{\theta}_i}{\theta_{\text{true}}}\right)^2} \cdot 100 \tag{6.6}$$

where the sum is over all n posterior samples, θ_{true} is the ground truth and $\hat{\theta}_i$ is the ith posterior sample. The RMSPE thus evaluates the accuracy of a posterior in terms of the percentage difference between the ground truth parameter and the weighted posterior estimate.

We aim to investigate different settings of tuning parameters, and will generate several posteriors for the same settings in order to assess variability. In these analyses, we take the RMSPE to be the mean of the RMSPEs of all posteriors for a particular setting. The variability can then be assessed through the *standard error of the mean* (SEM), which is the expected value of the standard deviation of means of several samples. This is estimated from a single sample as:

$$\text{SEM} = \frac{s}{\sqrt{k}}, \tag{6.7}$$

where s is standard deviation of the sample mean and k is the sample size.

7

Computational Approach

7.1 Computational Strategies

In this section, we present an assortment of computational strategies we will use.

7.1.1 Log Densities

Whenever possible, we compute with logarithmic densities in order to avoid computational overflows and underflows. Exponentiation is performed only when necessary. The Metropolis algorithm (Algorithm 2.1) is an example of where log densities should be used, as it requires evaluation of two densities in the calculation of the ratio. With log densities, the ratio is actually computed as the exponential of the difference of the log densities.

7.1.2 Parameter Transformations

Before regression adjustment, positive parameters should be log transformed. This will both stabilize the variance of the regression model and make it more homoscedastic. In addition, the log transform guarantees that the adjusted parameter values lie in the range of the prior distribution [41]. The final adjusted values are obtained by exponentiation of the r.h.s. of Equation 3.5.

7.1.3 Sample from the Prior and Posterior Predictive

When we want to do posterior predictive checks, we need to sample from the posterior predictive distribution defined by Equation 2.4. This involves an integral which can be completely avoided by the following iterative two-step process:

1. Sample a value of θ from the posterior $\pi(\theta \mid y)$.
2. Generate a prediction $\hat{y}$ by feeding the value of θ to:
   (a) The likelihood $p(y \mid \theta)$ in the case of likelihood-based inference;
   (b) The simulator model $\mathrm{M}(\theta)$ in the case of simulation-based inference.

The result of one iteration will be one sample from the posterior predictive distribution.

Following a similar logic, we can sample from the prior predictive distribution defined by Equation 2.3 via:

1. Sample a value of θ from the prior $\pi(\theta)$.
2. Generate a prediction $\hat{y}$ by feeding the value of θ to:
   (a) The likelihood $p(y \mid \theta)$ in the case of likelihood-based inference;
   (b) The simulator model $\mathrm{M}(\theta)$ in the case of simulation-based inference.

7.1.4 vtrap

The HH model contains rate equations that are equivalent to expressions on the form:

$$\text{rate} = \frac{x}{\exp(x/y) - 1}.$$

However, such expressions are prone to computational overflow. If $x/y = 0$ or close to zero, then the denominator is zero or really small which leads to infinite or extremely large output. From Taylor series approximation, we can find that the above expression is approximated by:

$$\text{rate} = y\left(1 - \frac{x}{2y}\right)$$

if $x/y << 1$. See Appendix B for the derivation. This expression is similar to how the NEURON simulator [42] handles indeterminate cases for HH style rate equations, and is called *vtrap* in their software. We will also refer to this approximation as vtrap. The HH model rate equations on this form will thus be computed by:

$$\text{vtrap} = \begin{cases} y\left(1 - \frac{x}{2y}\right) & \text{if } \frac{x}{y} << 1 \\ \frac{x}{\exp(x/y) - 1} & \text{otherwise} \end{cases} \tag{7.1}$$

7.1.5 A More Efficient Metropolis Sampler

Due to complex simulator models, it is not uncommon for the data generation step in ABC algorithms to be expensive and thereby dominate the computational overheads of the algorithms. In the MCMC ABC algorithm (Algorithm 3.5), there is actually no need to run forward the simulator model if the proposal parameter is rejected by the Metropolis acceptance criterion, which typically will be less expensive to evaluate. Therefore, the MCMC ABC algorithm can, in general, be formulated in a more efficient manner by changing the order of the required computations, as shown in Algorithm 7.1.

Algorithm 7.1: Efficient MCMC ABC with Metropolis sampler

Initialize :

Sample θ_0 by performing one iteration of rejection ABC (Algorithm 3.4).

Sampling :

for $t = 1, ..., N$ **do**

- Generate proposal $\theta^* \sim q(\theta^* \mid \theta_{t-1})$.
- Calculate acceptance criterion $\alpha = \min\left(1, \frac{\pi(\theta^*)}{\pi(\theta_{t-1})}\right)$.
- Sample $u \sim \mathrm{U}(0, 1)$.
- **if** $u \leq \alpha$ **then**
  - Simulate data y_{sim} from $M(\theta^*)$.
  - Calculate $\rho \equiv \rho(S(y_{\text{sim}}), S(y_{\text{obs}})) = \rho(s_{\text{sim}}, s_{\text{obs}})$.
  - **if** $\rho \leq \epsilon$ **then**
    - $\theta_t = \theta^*$
  - **else**
    - $\theta_t = \theta_{t-1}$
- **else**
  - $\theta_t = \theta_{t-1}$

7.1.6 Parallelization

The ABC algorithms are so-called *embarrassingly parallelizable*, which means there is little to no effort to divide the workload as the computations are independent. We will therefore parallelize the ABC samplers we implement ourselves. In the case of rejection ABC, we sample independent posterior samples. Hence, the prescribed number of posterior samples the sampler has to generate can be fairly divided between all available workers. In the case of MCMC ABC, the posterior samples are not independent due to the dependent Markov chains. Parallelizing a single chain is

not a straightforward task, but we could let multiple chains sample independently in parallel. As each chain needs sufficient time to converge towards the stationary distribution, each will need a sufficient number of posterior samples to generate in its workload.

7.2 Software Development

As part of the thesis, we developed two Python packages; `pyLFI`[1] for the implementation of ABC algorithms and regression adjustment, and `NeuroModels`[2] for the implementation of the neural simulator models and methods for summary statistic extraction. Both packages are available via the Python Package Index (PyPI)[3]. Besides personal preference, Python was chosen as programming language for several reasons; it is open source, allows for easy, flexible coding and have a plethora of available packages for analysis and visualizations. Moreover, for heavier computations, many Python packages interface with procedures written in faster languages like C. Most of the code is written using either Python's standard library or the standard scientific libraries `NumPy` [43], `SciPy` [44], `Matplotlib` [45], `pandas` [46] and `seaborn` [47]. For parallelization we used `Pathos` [48], which is a convenient wrapper of the `multiprocessing` package in the standard library. For implementations regarding the Brunel network model, we used `NEST` [49], `Neo` [50] and `Elephant` [51]. In addition, SNPE is implemented by its creators in a Python package called `sbi` [11].

We will not dive into too much detail regarding the implementation of `pyLFI` and `NeuroModels`, as the code itself is fairly documented and available for those interested. In the following, we instead focus on a broader overview of the implementation and brief demonstrations of usage.

7.2.1 NeuroModels

`NeuroModels` is toolbox for simulating neuroscientific models, post-simulation analysis and feature extraction. Here, we provide some implementation details and examples of usage of the different modules.

HH Solver

We made a general and flexible solver class of the HH model. The coupled differential equations are solved by using `scipy.integrate.solve_ivp`. The numerical integration method can be selected by the user, but in the present study we use the default explicit Runge-Kutta method of order 5(4) (RK45) [52]. Listing 7.1 shows an example of usage.

[1] https://github.com/nicolossus/pylfi
[2] https://github.com/nicolossus/neuromodels
[3] https://pypi.org

Listing 7.1: Example usage of the HH solver.

```
import neuromodels as nm

# The simulation parameters needed are the simulation time,
# time step and input stimulus:
T = 50.      # Simulation time [ms]
dt = 0.01    # Time step

# Stimulus can be provided as either a scalar, array or callable
stimulus = nm.stimulus.Constant(I_amp=10,
                                t_stim_on=10,
                                t_stim_off=40
                                )

# Initialize the Hodgkin-Huxley system; model parameters can either
# be set in the constructor or accessed as class attributes:
hh = nm.solvers.HodgkinHuxleySolver(V_rest=-65)
hh.gbarK = 36.0

# The system is solved by calling the class method `solve`:
hh.solve(stimulus, T, dt, method='RK45')

# The solutions can be accessed as class attributes:
t = hh.t
V = hh.V
n = hh.n
m = hh.m
h = hh.h
```

HH Simulator

The HH solver is wrapped into a HH simulator class with a call method that takes the conductance parameters as arguments and returns the simulated data. The simulator class also has methods for post-simulation analysis. Listing 7.2 shows an example of usage.

Listing 7.2: Example usage of the HH simulator.

```
import matplotlib.pyplot as plt
import neuromodels as nm

# The simulation parameters needed are the simulation time,
# time step and input stimulus:
T = 50.      # Simulation time [ms]
dt = 0.01    # Time step
stimulus = nm.stimulus.Constant(I_amp=10,
                                t_stim_on=10,
                                t_stim_off=40
                                )

# Initialize the Hodgkin-Huxley simulator; simulation and fixed
# model parameters are passed to the constructor:
hh = nm.models.HodgkinHuxley(stimulus,
                             T,
                             dt,
                             method='RK45',     # integration method
                             pdict={},          # dict of model params
                             solver_options={}  # dict of solver opts
                             )

# Calling the instance solves the HH system for the passed values
# of the active conductances, and the voltage trace is returned:
V, t = hh(gbar_K=36., gbar_Na=120.)
```

```
# The simulator class has methods for post-simulation analysis, e.g.:
hh.plot_voltage_trace(with_stim=True)
plt.show()
```

Spike Statistics

The output voltage trace from the HH simulator needs to be reduced to a set of low-dimensional summary statistics. We implemented procedures for extracting the summary statistics outlined in Section 6.2.1. The extraction of the summaries are based on using `scipy.signal.find_peaks` for identifying spikes that are above a specified firing threshold and separated by a distance greater than the refractory period. Having found the spike positions, most summary statistics can be derived by clever indexing of the voltage and time arrays. Average AP width is extracted via `scipy.signal.peak_widths`, and average AHP depth by passing the negative voltage array (i.e., the flipped voltage trace) to `scipy.signal.find_peaks`. The spike statistics extractor class has a call method that takes the voltage trace as argument and returns the summary statistics specified in the constructor. Listing 7.3 shows an example of usage.

Listing 7.3: Example usage of spike statistics extraction class.

```
import neuromodels as nm

# The simulation parameters:
T = 50.          # Simulation time [ms]
dt = 0.01        # Time step
t_stim_on = 10   # Stimulus onset
t_stim_off = 40  # Stimulus offset
stimulus = nm.stimulus.Constant(I_amp=10,
                                t_stim_on=t_stim_on,
                                t_stim_off=t_stim_off
                                )

# Initialize the Hodgkin-Huxley simulator:
hh = nm.models.HodgkinHuxley(stimulus, T, dt)

# Call simulator to solve system for passed conductances:
V, t = hh(gbar_K=36., gbar_Na=120.)

# Create a list of summary statistics to extract:
stats = ["average_AP_overshoot",
         "spike_rate",
         "average_AP_width",
         "average_AHP_depth",
         "latency_to_first_spike",
         "accommodation_index"]

# Initialize spike statistics extraction class; stimulus onset
# and offset as well as statistics to extract must be passed
# to the constructor:
sps = nm.statistics.SpikeStats(t_stim_on=t_stim_on,
                               t_stim_off=t_stim_off,
                               stats=stats,
                               threshold=0  # find only spikes above 0 mV
                               )

# The SpikeStats instance is callable; the voltage trace must be
# passed as argument. The extracted summary statistics are returned:
sum_stats = sps(V, t)
```

Brunel Solver

We implemented a flexible solver for the Brunel network model using `NEST` inside a Python class. The output of the network is returned as `neo.SpikeTrain` objects, which in turn are specified as `Quantity` objects; these are essentially arrays (or numbers) with a unit of measurement attached. The solver is parallelized through `NEST`. Listing 7.4 shows an example of usage.

Listing 7.4: Example usage of the Brunel network model solver.

```
import neuromodels as nm

# Initialize the Brunel network; the `order` parameter determines
# the number of neurons and connections in the network. Model
# parameters can either be set in the constructor or accessed as
# class attributes:
bnet = nm.solvers.BrunelNetworkSolver(order=2500, J=0.35)
bnet.eta = 2.0
bnet.g = 4.5

# The system is solved by calling the class method `simulate`.
# Simulation parameters have default values, but can also be set:
bnet.simulate(T=1000,       # Simulation time [ms]
              dt=0.1,       # Time step
              N_rec=20,     # Number of neurons to record from
              threads=8,    # Number of threads
              )

# The output of the network is returned as `neo.SpikeTrain` objects.
# Whether to return spike trains from excitatory ('exc', default) or
# inhibitory ('inh') neurons is controlled by the `n_type` keyword:
spiketrains = bnet.spiketrains(n_type="exc")

# The `summary` method gives a simple summary of the simulation:
bnet.summary()
```

Brunel Simulator

The Brunel simulator is similar in construction to the HH simulator. It wraps the Brunel solver class and is callable. The call method takes the synaptic weight parameters η and g as arguments. The simulator also has methods for post-simulation analysis. Listing 7.5 shows an example of usage.

Listing 7.5: Example usage of the Brunel simulator.

```
import matplotlib.pyplot as plt
import neuromodels as nm

# Model parameters
order = 2500    # -> NE=10,000 ; NI=2500 ; N_tot=12,500 ; CE=1000 ; CI=250
epsilon = 0.1   # Connection probability
T = 1000        # Simulation time [ms]
N_rec = 20      # Record output from N_rec neurons
n_type = 'exc'  # Record excitatory spike trains
D = 1.5         # Synaptic delay [ms]
J = 0.1         # Excitatory synapse weight [mV]

# NEST settings
threads = 16        # Number of threads to use in simulation
print_time = False  # Print simulated time or not

# Simulator model class constructor:
```

```
bnet = nm.models.BrunelNet(order=order,
                           epsilon=epsilon,
                           T=T,
                           N_rec=N_rec,
                           n_type=n_type,
                           D=D,
                           J=J,
                           threads=threads,
                           print_time=print_time,
                           )

# The call method takes the synaptic weight parameters `eta` and `g`
# as arguments and returns the output spike trains:
spiketrains = bnet(eta=2.0, g=4.5)

# The simulator class has methods for post-simulation analysis, e.g.:
bnet.rasterplot_rates()
plt.show()
```

Spike Train Statistics

The output spike trains from the Brunel simulator also need to be reduced to a set of low-dimensional summary statistics. We used `Elephant` to extract the summary statistics outlined in Section 6.2.2. The usage of the spike train statistics extractor class is similar to its counterpart for the HH simulator, as shown in Listing 7.6.

Listing 7.6: Example usage of the spike train statistics extraction class.

```
import neuromodels as nm
import quantities as pq

# Simulator model class constructor:
bnet = nm.models.BrunelNet(order=2500,
                           epsilon=0.1,
                           T=1000,
                           N_rec=20,
                           n_type='exc',
                           D=1.5,
                           J=0.1,
                           threads=16,
                           )

# Simulator call method:
spiketrains = bnet(eta=2.0, g=4.5)

# Create a list of summary statistics to extract:
stats = ["mean_firing_rate",  # rate estimation
         "mean_cv",           # spike interval statistic
         "fanofactor"         # statistic across spike trains
         ]

# Define start and end time as `Quantity` objects:
t_start = 100. * pq.ms  # Cutoff to avoid transient effects
t_stop = 1000 * pq.ms   # End time

# Initialize spike train statistics extraction class;
# start and end time as well as statistics to extract
# must be passed to the constructor:
sts = nm.statistics.SpikeTrainStats(t_start=t_start,
                                    t_stop=t_stop,
                                    stats=stats
```

```
                                    )

# The SpikeTrainStats instance is callable; the spike trains must be
# passed as argument. The extracted summary statistics are returned:
sum_stats = sts(spiketrains)
```

7.2.2 pyLFI

`pyLFI` is a Python toolbox using likelihood-free inference (LFI) (also known as simulation-based inference) for estimating the posterior distributions over model parameters. We have implemented both rejection ABC (Algorithm 3.4) and MCMC ABC (Algorithm 7.1), as well as post-sampling regression adjustment. We made the software general and flexible, so that it can accommodate other algorithms as well. The price to pay for the generality and flexibility is that the simulation of data and calculation of summary statistics are left entirely to the user. To perform parameter identification with `pyLFI`, there are generally four inputs that need to be specified:

1. A simulator model. The mechanistic model needs to be specified through a simulator model that can generate simulated data y_{sim} for any parameters θ.
2. A summary statistics calculator. The ABC algorithms require the use of low-dimensional summary statistics $s = S(y)$ calculated from the raw data y.
3. Observed data y_{obs}. This must be on the same form as y_{sim}.
4. A prior $\pi(\theta)$ for each unknown parameter that describes the range of possible parameter values.

For each problem, the objective is to estimate the posterior distribution $\pi(\theta \mid y_{\text{obs}})$. In general, setting up the inference procedure requires three design choices:

1. A distance metric.
2. Tuning parameters. The number of tuning parameters depend on which ABC algorithm is being used. The central tuning parameter for all algorithms is the threshold ϵ, for which we introduced the alternative quantile-based approach. For MCMC algorithms, there are additional tuning parameters like proposal density scale, burn-in iterations etc.
3. Either a simulation budget, i.e. a prescribed number of simulations to run, or a prescribed number of posterior samples the ABC sampler must obtain. Running the simulator is generally the costliest step of the procedure, and many simulator runs might be needed to accurately produce the posterior.

Example Usage

In the following, we demonstrate `pyLFI` on a toy example. We will infer the mean μ and standard deviation σ of a Gaussian distribution with likelihood $p(y_{\text{obs}} \mid \mu, \sigma) = \text{N}(\mu = 163, \sigma = 15)$, The observed data are sampled from the likelihood:

```
mu_true = 163
sigma_true = 15
likelihood = scipy.stats.norm(loc=mu_true, scale=sigma_true)
obs_data = likelihood.rvs(size=1000)
```

We assume that the likelihood is unknown, and formulate a model to describe the observed data. The model needs to be implemented as a Python callable, i.e., a function or a call method in a class, that is parametrized by the unknown model parameters we aim to infer, here μ and σ:

```
def simulator(mu, sigma, size=1000):
    y_sim = stats.norm(loc=mu, scale=sigma).rvs(size=size)
    return y_sim
```

Next, we need to reduce the raw data into low-dimensional summary statistics. The summary statistics calculator also needs to be implemented as a Python callable. The function must return the summary statistics as a Python `list` or `numpy.ndarray`. Here, we take the mean and standard deviation to be summary statistics of the data (these are actually *sufficient* summary statistics):

```
def stat_calc(y):
    sum_stats = [numpy.mean(y), numpy.std(y)]
    return sum_stats
```

We then place priors over the unknown model parameters using the `pylfi.Prior` class. The signature of the class constructor is:

```
pylfi.Prior(distr_name, *params, name=None, tex=None, **kwargs)
```

Here, `distr_name` is passed as a string that correspond to any distribution from `scipy.stats`. `params` are the parameters of the distribution, typically these would be `shape` parameters or `loc` and `scale` passed as positional arguments. `kwargs` are keyword arguments passed to the `SciPy` distribution methods. The `name` keyword argument is required, and is used by `pyLFI` to keep track of the parameter. The `tex` keyword argument is optional, and can be used to provide the LaTeX typesetting for the parameter name. `pyLFI` includes procedures for automatically plotting priors and posteriors, and will use the `tex` name of the parameter as axis labels if provided. The `name` string is used to access the parameters further down the pipeline and is also used as axis labels if `tex` is not provided.

In the present example, we define the priors:

```
mu_prior = pylfi.Prior('norm',
                       loc=165,
                       scale=2,
                       name='mu',
                       tex='$\mu$'
                       )

sigma_prior = pylfi.Prior('uniform',
                          loc=12,
                          scale=7,
                          name='sigma',
                          tex='$\sigma$'
                          )

priors = [mu_prior, sigma_prior]
```

Note that the priors must be collected in a Python `list`. By providing points x to evaluate the prior pdf at, the `pylfi.Prior` class also lets us visualize the priors:

```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(5, 2), tight_layout=True)

x = np.linspace(159, 171, 1000)
mu_prior.plot_prior(x, ax=axes[0])

x = np.linspace(11, 20, 1000)
sigma_prior.plot_prior(x, ax=axes[1])

plt.show()
```

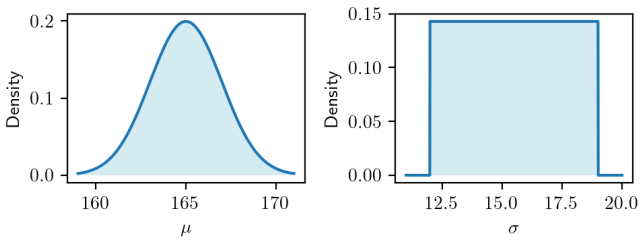


Figure 7.1: Priors over μ and σ in the toy example with a Gaussian model.

Now, all components needed for inference with `pyLFI` are in place, and we can initialize the ABC sampler:

```
sampler = pylfi.RejABC(obs_data,    # observed data
                       simulator,   # simulator model
                       stat_calc,   # sum stat calculator
                       priors,      # priors over params
                       log=True.    # display logger or not
                       )
```

The constructors of the ABC samplers are identical, as each sampler inherit most methods from a general parent class (`pylfi.ABCBase`).

As mentioned above, there are two approaches for configuring the ABC sampler; we either set a simulation budget or the number of posterior samples to obtain. The latter approach requires a pilot study if we want to estimate ϵ and the summary statistic scales from the prior predictive distribution. The signature of the pilot study method is:

```
pylfi.ABCBase.pilot_study(nsims,
                          quantile=None,
                          stat_scale=None,
                          stat_weight=1.,
                          n_jobs=-1,
                          seed=None,
                          )
```

The pilot study runs the simulator `nsims` times with parameters sampled from the prior predictive distributions, and sets the tolerance ϵ automatically as the

quantile, specified by the `quantile` keyword, of the simulated distances. If the `stat_scale` keyword is passed as one of the strings `sd` or `mad`, the pilot study also provides an estimate of each summary statistic scales, which are used in the weighted Euclidean distance. `stat_scale='sd'` scales the summary statistics according to their standard deviation (SD) estimated from the prior predictive samples, and `stat_scale='mad'` according to their median absolute deviation (MAD). The `stat_weight` keyword can be used to provide importance weights to the summary statistics. The computational demanding ABC sampler methods are parallelized, and the `n_jobs` keyword can be used to set the number of workers. The default, `n_jobs=-1`, sets the number of workers automatically as the number of found CPUs in the system. Furthermore, the ABC sampler can be seeded via the `seed` keyword. The seed is just provided as an integer. `pyLFI` has procedures for parallel random number generation (PRNG) based on the provided seed, and ensures correct advancement of the underlying PRNG states. This means that the ABC sampler themselves are reproducible, but a caveat of PRNG is that exact reproducibility only is possible when using the same seed and same number of workers.

For our toy example, performing a pilot study is done as follows:

```
nsims = 1000
sampler.pilot_study(nsims,
                    quantile=0.2,
                    stat_scale="sd",
                    stat_weight=1,
                    n_jobs=4,
                    seed=4
                    )
```

To sample from the posterior, we must call the `sample` method, which is specific for each algorithm. For the rejection ABC sampler when a pilot study has been performed, the call becomes:

```
nsamples = 3000
journal = sampler.sample(nsamples,
                         use_pilot=True,
                         n_jobs=4,
                         seed=42,
                         return_journal=True
                         )
```

Here, the first positional argument is the number of posterior samples we want. By setting `use_pilot=True`, the sampler will use the tolerance and summary statistic scales found in the pilot study automatically. The results of an inference are stored as a `pylfi.Journal` object, which can be returned by setting the `return_journal=True`. The journal can also be accessed through the method:

```
nsamples = 3000
sampler.sample(nsamples,
               use_pilot=True,
               n_jobs=4,
               seed=42
               )
journal = sampler.journal()
```

`pylfi.Journal` objects can be both saved to and loaded from disk:

```
# Save journal
filename = 'my_journal.jnl'
journal.save(filename)

# Load journal
journal = pylfi.Journal.load(filename)
```

Post-sampling regression adjustment can be performed via a base ABC class method with signature:

```
pylfi.ABCBase.reg_adjust(method="loclinear",
                         transform=True,
                         kernel='epkov',
                         return_journal=False
                         )
```

The `method` keyword selects the regression method, either linear (`'linear'`) or local linear (`'loclinear'`), the `transform` keyword determines whether or not to take the log transform of the target (the log transform usually gives better regression results) and the `kernel` keyword selects the smoothing kernel, either the Gaussian kernel (`'gaussian'`) or the Epanechnikov kernel (`'epkov'`).

Thus, performing a local linear regression for our toy example with Epanechnikov kernel and log transform of the the target is coded as:

```
journal = sampler.reg_adjust(method="loclinear",
                             transform=True,
                             kernel="epkov",
                             return_journal=True
                             )
```

The obtained posterior samples can be retrieved as a `pandas.DataFrame` from the `pylfi.Journal` class:

```
df = journal.df  # pandas DataFrame with posterior samples
```

Furthermore, posteriors can be plotted with the `pylfi.Journal.plot_posterior` method. The first positional argument is required and expects a string with the name of a parameter, corresponding to the `name` keyword argument passed to the `pylfi.Prior` constructor. Other options are `hdi_prob` which can be used to set the probability $(1 - \alpha)$ of the HDI, `point_estimate` which point estimate to use, either `'map'`, `'mean'` or `'median'`, and `theta_true` to set the ground truth (if available). If `theta_true` is set, then the RMSPE will also be included in the plot. For our present example:

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3), tight_layout=True)
journal.plot_posterior('mu',
                       hdi_prob=0.95,
                       point_estimate='map',
                       theta_true=mu_true,
                       ax=axes[0]
                       )

journal.plot_posterior('sigma',
                       hdi_prob=0.95,
                       point_estimate='map',
                       theta_true=sigma_true,
```

```
                        ax=axes[1]
                        )

plt.show()
```

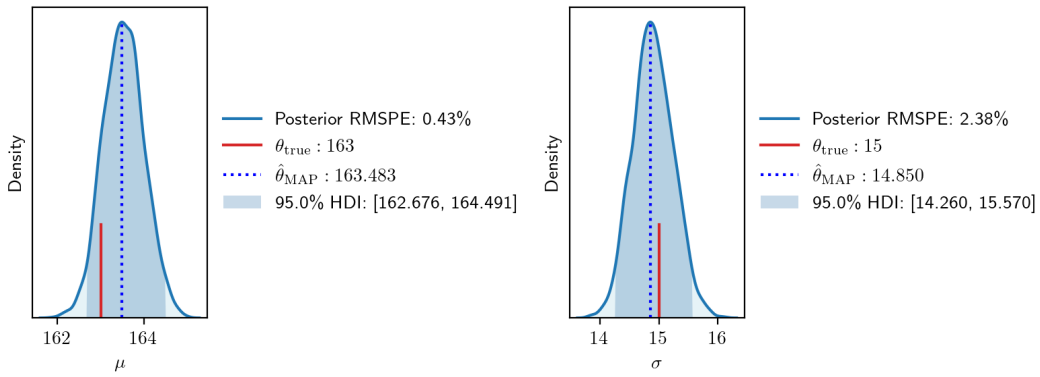


Figure 7.2: Posteriors over μ and σ in the toy example with a Gaussian model.

Listing 7.7 lists a complete script for the usage of `pyLFI` with rejection ABC on the Gaussian toy example:

Listing 7.7: Example usage of the pyLFI on a Gaussian toy model.

```
import matplotlib.pyplot as plt
import numpy as np
import pylfi
import scipy.stats as stats

# Observed data
mu_true = 163
sigma_true = 15
likelihood = stats.norm(loc=mu_true, scale=sigma_true)
obs_data = likelihood.rvs(size=1000)


# Simulator model
def simulator(mu, sigma, size=1000):
    y_sim = stats.norm(loc=mu, scale=sigma).rvs(size=size)
    return y_sim


# Summary statistics calculator
def stat_calc(y):
    sum_stat = [np.mean(y), np.std(y)]
    return sum_stat


# Priors
mu_prior = pylfi.Prior('norm',
                       loc=165,
                       scale=2,
                       name='mu',
                       tex='$\mu$'
                       )

sigma_prior = pylfi.Prior('uniform',
```

```
                            loc=12,
                            scale=7,
                            name='sigma',
                            tex='$\sigma$'
                            )


priors = [mu_prior, sigma_prior]

# Initialize sampler
sampler = pylfi.RejABC(obs_data,
                       simulator,
                       stat_calc,
                       priors,
                       log=True
                       )

# Pilot study
nsims = 1000
sampler.pilot_study(nsims,
                    quantile=0.2,
                    stat_scale="sd",
                    stat_weight=1,
                    n_jobs=4,
                    seed=4
                    )

# Sample posterior
nsamples = 3000
sampler.sample(nsamples,
               use_pilot=True,
               n_jobs=4,
               seed=42
               )

# Local linear regression adjustment
journal = sampler.reg_adjust(method="loclinear",
                             kernel="epkov",
                             transform=True,
                             return_journal=True
                             )


# Plot posteriors
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 3), tight_layout=True)
journal.plot_posterior('mu',
                       hdi_prob=0.95,
                       point_estimate='map',
                       theta_true=mu_true,
                       ax=axes[0]
                       )

journal.plot_posterior('sigma',
                       hdi_prob=0.95,
                       point_estimate='map',
                       theta_true=sigma_true,
                       ax=axes[1]
                       )

plt.show()
```

Part III

Results & Discussion

8

Inference on the HH Model

In this chapter, we present the results from simulation-based inference on the Hodgkin-Huxley (HH) model's conductance parameters $\bar{g}_{\mathrm{K}}$ and $\bar{g}_{\mathrm{Na}}$.

8.1 Observation and Feature Extraction

Let us assume we current-clamped a neuron and recorded the voltage trace in Figure 8.1. This voltage trace was not actually measured experimentally but synthetically generated by simulating the HH model through the HH simulator in `NeuroModels`. The model was simulated for $T = 120\,\mathrm{ms}$ with step size $\Delta t = 10\,\mathrm{ms}$ and stimulus $I = 10\,\mu\mathrm{A/cm}^2$ turned on at 10 ms and off at 110 ms. The conductance parameters were set as $\bar{g}_{\mathrm{K}} = 36\,\mathrm{mS/cm}^2$ and $\bar{g}_{\mathrm{Na}} = 120\,\mathrm{mS/cm}^2$. The rest of the HH model's parametrization is given in Table 5.1. The idealized voltage trace recording, free of any noise, will be used as the observed data in our first analyses. Hopefully, we can then more easily assess strengths and weaknesses of the algorithms themselves, and not have the results overshadowed by noisy data. Furthermore, the present trace allows us to verify whether the computational implementations are accurate. The ground truth parameters will therefore be the particular values of $\bar{g}_{\mathrm{K}}$ and $\bar{g}_{\mathrm{Na}}$ used in the simulation.

By visual inspection of the trace in Figure 8.1, the expected shape of an action potential (AP) is reproduced by the numerical solution, which indicates that the implementation of the simulator is accurate. Moreover, since the voltage trace does not display any unexpected abrupt behavior, the time resolution of $\Delta t = 0.025\,\mathrm{ms}$ seems to be sufficient.

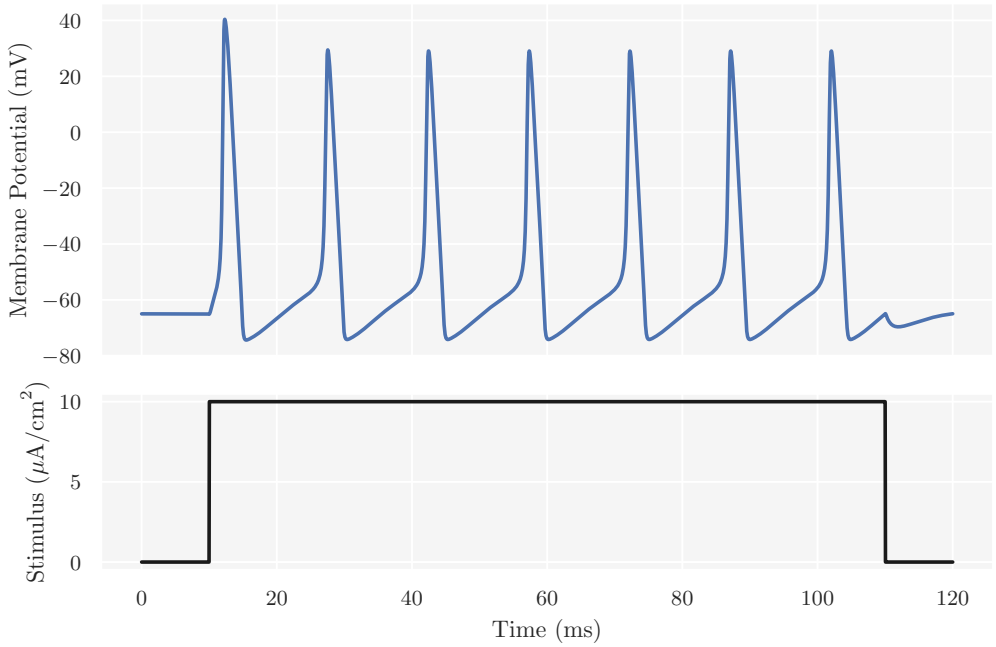


Figure 8.1: Observed voltage trace of a current clamped neuron synthetically generated by the HH simulator simulated for $T = 120\,\text{ms}$ with time resolution $\Delta t = 0.025\,\text{ms}$. The stimulus is a step current $I = 10\,\mu\text{A/cm}^2$ with onset and offset at 10 ms and 110 ms, respectively. Here, the conductance parameters $\bar{g}_\text{K} = 36\,\text{mS/cm}^2$ and $\bar{g}_\text{Na} = 120\,\text{mS/cm}^2$. The present voltage trace being the observation, these conductance parameters are thus the ground truths for the subsequent analyses.

From the voltage trace, we extract spike statistics by the computational algorithms outlined in Section 7.2. Figure 8.2 shows the locations in the voltage trace that form the basis of the spike statistic calculations. In fact, the annotations on the voltage trace are set automatically according to the positions found by the extraction algorithms. By the definitions of the different summary statistics provided in Section 6.2.1, we see that the extraction locations are placed correctly on the voltage trace. Consequently, the extraction algorithms seem to function as intended.

Table 8.1 summarizes the calculated summary statistics from the observed voltage trace; (i) *spike rate*, calculated as the number of spikes divided by the duration of the stimulus; (ii) *average AP overshoot*, calculated by averaging the absolute peak voltage of all APs; (iii) *average AP width*, calculated by averaging the width of every AP at the midpoint between its onset and its peak; (iv) *average AHP depth*, calculated by averaging all minima voltage throughs between two consecutive APs; (v) *latency to first spike*, calculated as the time between stimulus onset and first AP peak; (vi) *accommodation index*, which measures the local variance in ISIs and is calculated by Equation (6.1). Comparing the values of the tabulated summary

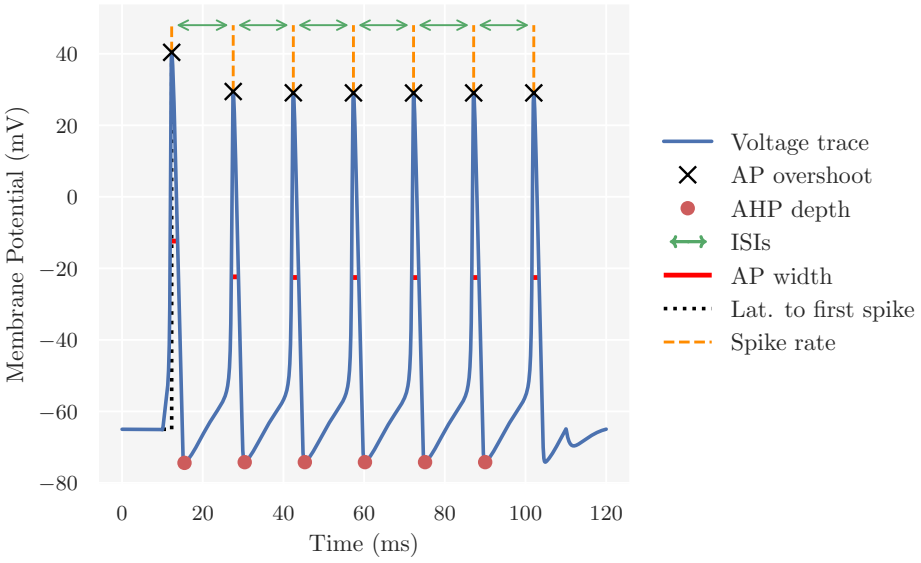


Figure 8.2: Locations found by the feature extraction algorithms for spike statistic calculations on the observed voltage trace. The locations are annotated by different markers, with labels stated in the legend, which indicate the particular summary statistic calculation they are affiliated with.

statistics with the information in Figure 8.2, we find agreement. There are 7 spikes over the course of the stimulus duration of 100 ms, so the spike rate must be 0.07 mHz. Furthermore, the value of the average AP overshoot and width, as well as the average AHP depth, seem reasonable when compared with the voltage values at the extracted locations. A latency to first spike of about 2 ms also matches what is seen in the voltage trace. There is practically no difference in length between two consecutive ISIs in the voltage trace, and the accommodation index should therefore reflect, as it does, the lack of variability. All in all, this indicates that also the summary statistic calculations are implemented correctly.

Table 8.1: Observed voltage trace reduced to a set of summary statistics. See text for details on the statistics.

Summary statistic	**Observed value**
Spike rate	0.0700 mHz
Average AP overshoot	30.7316 mV
Average AP width	2.0501 mV
Average AHP depth	-74.2234 mV
Latency to first spike	2.3000 ms
Accommodation index	$2 \cdot 10^{-17}$

8.1.1 Correlation Analysis & Importance Weights

Next, we carry out the correlation analysis outlined in Section 6.3. The objective of the analysis is to characterize the effects of parameter variability on the output of the model in terms of the summary statistics. The analysis is done by sampling from the prior predictive distribution, and the priors for $\bar{g}_{\mathrm{K}}$ and $\bar{g}_{\mathrm{Na}}$ are shown in Figure 8.3. For each parameter, we use a noninformative prior (orange density)

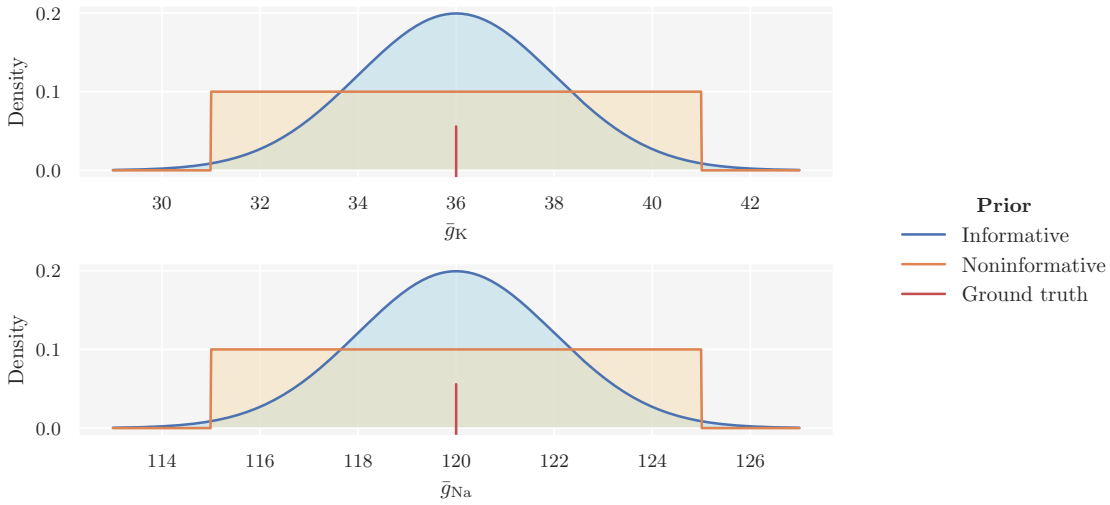


Figure 8.3: Priors over $\bar{g}_{\mathrm{K}}$ (top) and $\bar{g}_{\mathrm{Na}}$ (bottom). We use both an informative (blue) and noninformative (orange) centered about the ground truth parameter value (red line) for each parameter.

with about $\pm 10\%$ range around the ground truth parameter and a slightly more informative prior (blue density). While technically the informative priors could be classified as weakly-informative (as per the definition given in Section 2.3.1), we will refer to them as informative.

For each category of priors, we sampled 2000 parameter pairs, fed them to the HH simulator model and calculated the summary statistics from the simulated data for each pair. The spike statistics are only well-defined in the presence of spikes, and accommodation index and average AHP depth need at least two and three spikes, respectively, to be defined. As such, we need to remove samples if they contain ill-defined statistics.

With Informative Priors

Of the 2000 summary statistics samples simulated under the informative priors, 1881 were well-defined. Scatter plots for a subset of these are shown in Figure 8.4, where the summary statistics are shown as functions of the pairs of parameter values. Thus, the scatter plots enable us to see the variability the different summary statistics exhibit relative to change in model parameter values. Each point indicates

the relative magnitude of the statistic by its size and color, with a reference table stated in the legend along with the name of the particular statistic.

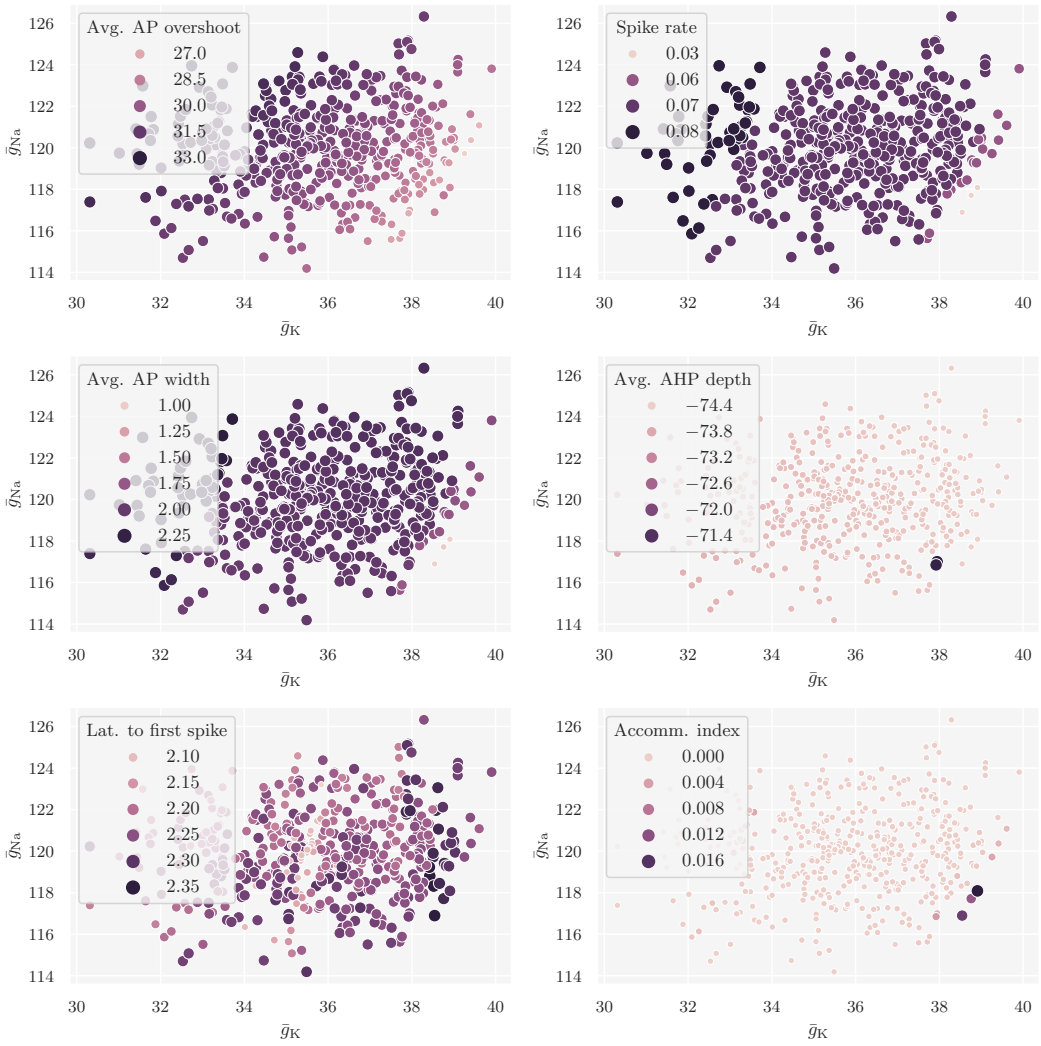
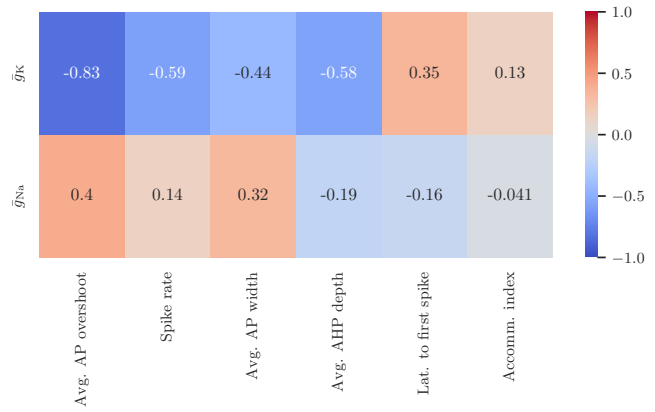


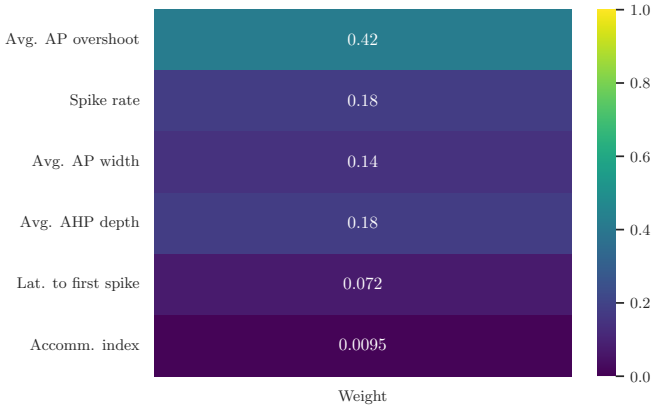
Figure 8.4: Scatter plots of summary statistics simulated with different pairs of model parameter values. The summary statistics were simulated under the joint informative prior predictive distribution. Of 2000 generated samples, 1881 were well-defined. Here, a subset of 470 samples are shown. Each summary statistic is assigned to its own panel, with the particular statistic stated in the legend. Each point represents the value of a summary statistic for a pair of parameter values, $(\bar{g}_{\mathrm{K}}, \bar{g}_{\mathrm{Na}})$. The color of a point indicates the relative magnitude of the statistic, for which bright colors represent small and dark colors large values, also indicated in each subplot legend. The scatter plots thus indicate the variability of summary statistics relative to movement of the pairs of model parameter values.

In addition to displaying the variability, the scatter plots also indicate if there is a systematic relationship between a parameter and summary statistic. By fixing the value of one of the parameters and following its line-of-sight, we can assess whether the other parameter systematically increments or decrements a summary statistic. Average AP overshoot exhibits a steady variability and seems to follow an approximately linear trend. This also applies to the average AP width, though to a lesser extent. The approximate linear relationship is most apparent for $\bar{g}_{\mathrm{K}}$ in both cases. In terms of the underlying biophysical mechanisms, we would expect average AP overshoot to be most sensitive to $\bar{g}_{\mathrm{Na}}$ and average AP width to $\bar{g}_{\mathrm{K}}$. The role of the Na^+channel in AP generation is well-established, and AP overshoot is tied to the fast Na^+channel dynamics. Likewise, given the role of the K^+channel in repolarizing the neuron after an AP, the average AP width is tied to the duration of the recovery period. While these expectations are not clearly present in the scatter plots, we should keep in mind that we only explore a fairly limited region of the parameter space. We should therefore be cautious in our interpretation of how well the scatter plots indicate sensitivity. Accommodation index has almost no variation, which might be unsurprising since it measures the local variance in ISIs. The constant current protocol does not facilitate much variation in spike trains. Thus, for the observed voltage trace at hand, accommodation index is not an informative summary statistic. However, it could be useful in characterizing spike trains generated under more complex current protocols. Average AHP depth also exhibits little variation, whereas latency to first spike an ample amount. Though, for latency to first spike there is no apparent systematic relationship, as it increments and decrements right and left. This might not be optimal for constraining the model parameters. Finally, the spike rate, though it does not vary much, seem to have a more systematic and stable relationship with the model parameters, predominantly $\bar{g}_{\mathrm{K}}$. It behaves more like a step function, where it retains a particular value for a prolonged range of $\bar{g}_{\mathrm{K}}$ values. In conclusion, how well this set of summary statistics will constrain the model parameters needs to be investigated further down the line.

In order to not rely solely on visual inspection of the sensitivity, Figure 8.5 provides the pairwise Pearson's correlation coefficients of each model parameter and summary statistics, as well as the importance weights derived from the correlation coefficients. As was indicated by the scatter plots, $\bar{g}_{\mathrm{K}}$ has a stronger (approximately) linear relationship with the summary statistics than $\bar{g}_{\mathrm{Na}}$. The interpretation of these results are that the summary statistics related to the shape of an AP encode the most information, with average AP overshoot being the most dominant. Moreover, the results indicate that $\bar{g}_{\mathrm{K}}$ will be constrained better than $\bar{g}_{\mathrm{Na}}$ by these summaries when performing regression adjustment. Since the relationship between $\bar{g}_{\mathrm{K}}$ and the summary statistics is much stronger than for $\bar{g}_{\mathrm{Na}}$, the weighting scheme also becomes biased towards $\bar{g}_{\mathrm{K}}$. The weighting is therefore a bit unfair for $\bar{g}_{\mathrm{Na}}$. For instance, even though the spike rate is weakly correlated with $\bar{g}_{\mathrm{Na}}$, it receives a heavy weight because it is strongly correlated with $\bar{g}_{\mathrm{K}}$. The weight of the spike rate even surpasses that of average AP width, which shows a decent amount of correlation with both



(a)



(b)

Figure 8.5: **(a)** The pairwise Pearson's correlation coefficients for the model parameters and summary statistics. **(b)** The importance weights calculated from the correlation coefficients, see Section 6.3 for details. Note that the weights sum to one.

$\bar{g}_{\text{Na}}$ and $\bar{g}_{\text{K}}$. The pairwise Pearson's correlation coefficients strong assumption about linearity again necessitates the need to be wary of the interpretation of which summary statistics are the most informative. As the weighting scheme prefers $\bar{g}_{\text{K}}$, we should also investigate the effects importance weights have on the inference further down the line.

With Noninformative Priors

The findings with noninformative priors are similar to the ones discussed above. This is perhaps unsurprising, since a prior does not alter the intrinsic relationship between a parameter and summary statistic, just how the samples are distributed in the parameter space. The corresponding figures with samples from the joint noninformative prior distribution can be found in Appendix A.

8.2 Study of ABC Settings

We now turn to a study concerning the tuning parameters in the rejection ABC algorithm. As the generation of one posterior amounts to a single stochastic trial, we will generate several posteriors for the same settings in order to assess potential variability in the results. Here, we use the rejection sampler in `pyLFI` to infer the conductance parameters in the HH model. The performance metrics RMSPE and SEM, defined in Section 6.5, will be used to assess the accuracy and variability, respectively, of a particular inference setting.

In our first tuning parameter analysis, we study the effect of the tolerance parameter ϵ in terms of the p_ϵ-quantile of the distances. For each quantile, we generate 10 posterior with 1000 posterior samples in each. As discussed in Section 7.2, this means that the tolerance will be set via a pilot study. For each quantile, the pilot study performs 2000 simulations to estimate both the tolerance and the scale of the summary statistics. In this analysis, the summary statistics are equally weighted. Having obtained a posterior, we perform local linear regression adjustment with the Epanechnikov kernel and log transformation of the parameters to obtain a corresponding adjusted posterior. We do the above using both the informative and noninformative priors. Figure 8.6 shows the results. RMSPE measures the percentage difference between the ground truth and a weighted estimate that accounts for the width of the posterior. Increasing p_ϵ-quantiles amount to accepting simulated data that are increasingly further away from the observed data. As such, we expect the error to increase with p_ϵ, due to more distorted approximations. This is also the general trend, though the differences in error between posteriors are generally small. Here, the posterior error estimates of $\bar{g}_\mathrm{K}$ and $\bar{g}_\mathrm{Na}$ differ. Focusing on the estimates of error in the original posterior samples, i.e., the samples obtained solely with the rejection ABC algorithm, $\bar{g}_\mathrm{K}$ has larger errors than $\bar{g}_\mathrm{Na}$. The errors of $\bar{g}_\mathrm{Na}$ remain almost constant as p_ϵ increases, and are actually slightly larger for the lowest p_ϵ. The reason for this may be intricate, but most likely it has to do with correlation between the posterior samples of $\bar{g}_\mathrm{Na}$ and $\bar{g}_\mathrm{K}$ and that the simulated summary statistics accepted under a strict tolerance happen to shift the $\bar{g}_\mathrm{Na}$ estimate for the worse (by a little amount) and $\bar{g}_\mathrm{K}$ for the better. In terms of variation, the SEM of all the mean RMSPE shows that the inferred posteriors for the same settings are practically indistinguishable. We also see that estimates with informative priors converge better than those with noninformative priors, as expected. The RMSPE of the regression adjusted posterior estimates are significantly more accurate than the original posterior estimates. The improvement is most prominent for $\bar{g}_\mathrm{K}$, and aligns with the expectation obtained from the correlation analysis; since $\bar{g}_\mathrm{K}$ has a stronger linear relationship with the summary statistics than $\bar{g}_\mathrm{Na}$, the local linear regression model will give better adjustment of the $\bar{g}_\mathrm{K}$ posterior samples. The difference in error when p_ϵ increases are tiny for $\bar{g}_\mathrm{K}$, which suggests that the regression approach manages to correct the $\bar{g}_\mathrm{K}$ samples as if they were sampled from $\pi_\mathrm{ABC}(\bar{g}_\mathrm{K} \mid \rho(s_\mathrm{sim}, s_\mathrm{obs}) \leq \epsilon)$ with $\epsilon = 0$. For $\bar{g}_\mathrm{Na}$, however, this breaks down and the error increases with p_ϵ, with a particular jump between the

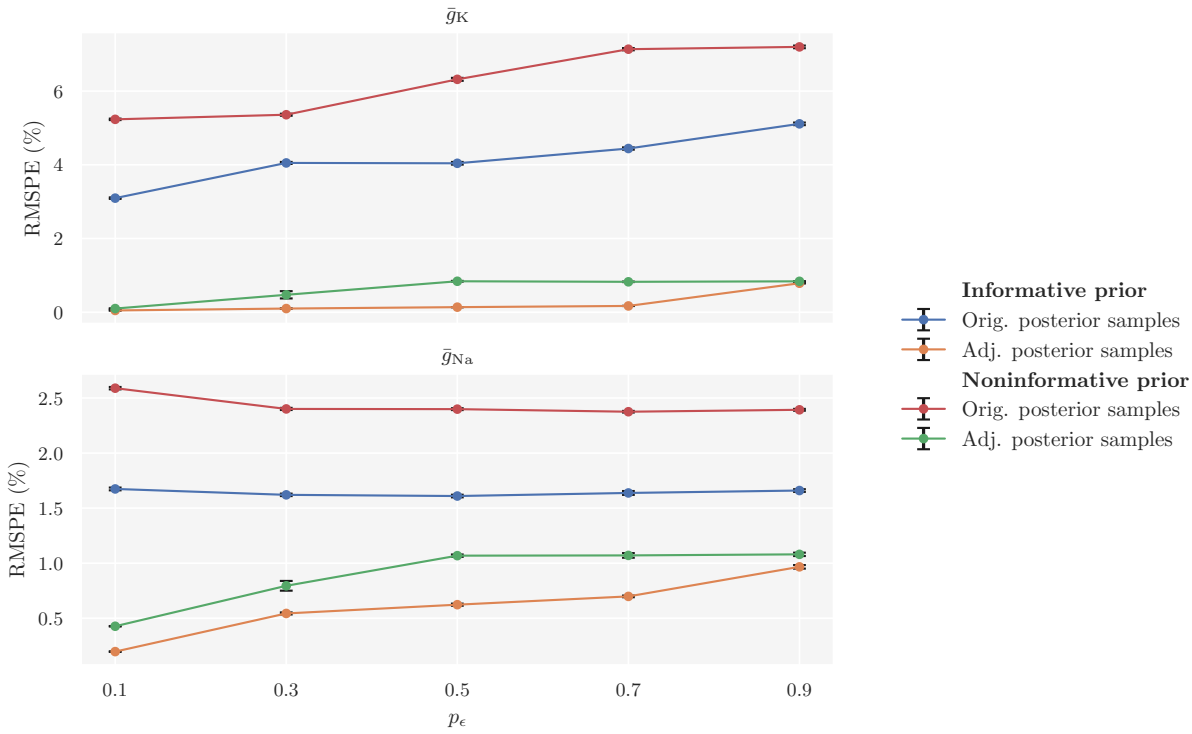


Figure 8.6: The RMSPE in posteriors over $\bar{g}_{\mathrm{K}}$ (top) and $\bar{g}_{\mathrm{Na}}$ (bottom) against the p_ϵ-quantile as a measure of tolerance. Each point is the mean RMSPE over 10 posteriors, each consisting of 1000 posterior samples, and the SEM is shown as a vertical bar. The posteriors were generated by the rejection ABC algorithm and then adjusted with local linear regression adjustment. Whether an estimate of error had informative/noninformative priors or is from the original/adjusted posterior is color coded (see legend).

0.1 and 0.5-quantiles. Nevertheless, with regression adjustment, more simulations can be accepted without sacrificing substantial accuracy, especially for the model parameters that are the most constrained by the summary statistics. In addition, the difference in error between using informative and noninformative priors is also reduced when performing regression adjustment.

Next, we investigate error in the estimates against the number of summary statistics used to constrain the model parameters. We start with only a single statistic, average AP overshoot, and increment by one more according to the following order; spike rate, average AP width, average AHP depth, latency to first spike and accommodation index. Again, we use the quantile-based rejection ABC algorithm with local linear regression adjustment using the Epanechnikov kernel to estimate the posteriors. We use the 0.4-quantile as a compromise between accuracy and run time (see Figure A.4). From the preceding result (Figure 8.6) we found that the posteriors generated for the same settings are practically identical. Here, we therefore only

generate a single posterior for each number of summary statistics. This is, however, done for both the informative and noninformative priors. In addition, we generate posteriors for both cases of weighting of the summary statistics; either equally or importance weighted (we ensure the importance weights sum to one). The results are shown in Figure 8.7. The figure shows that the HH model is more tightly

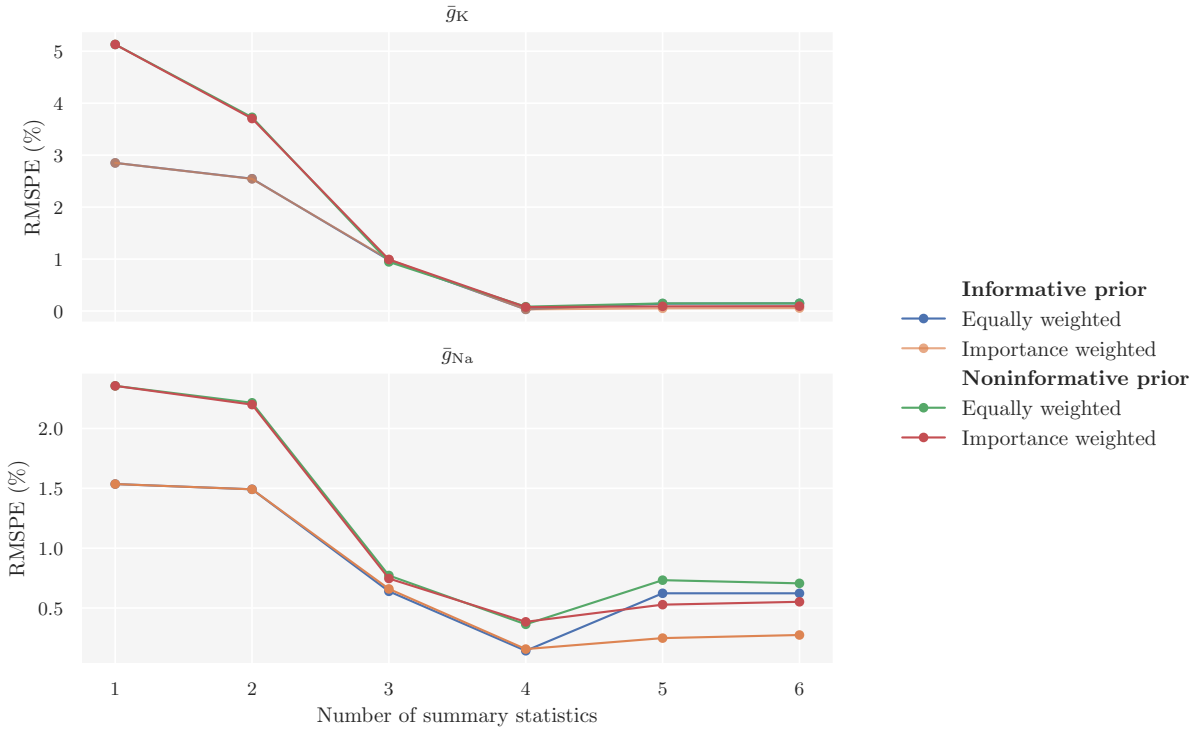


Figure 8.7: The RMSPE in posteriors over $\bar{g}_K$ (top) and $\bar{g}_{Na}$ (bottom) against the number of summary statistics. The first, single statistic is (i) average AP overshoot, and then the number of statistics is incremented by one more according to the following order; (ii) spike rate, (iii) average AP width, (iv) average AHP depth, (v) latency to first spike and (vi) accommodation index. Each point is the RMSPE in a regression adjusted posterior consisting of 1000 posterior samples. Whether an estimate of error had informative/noninformative priors or equally/importance weighted the summary statistics is color coded (see legend).

constrained by increasing the number of summary statistics, particularly $\bar{g}_K$ as we have already discussed. Though, if we use summary statistics that do not capture relevant information for the parameters, it might lead to worse inference. The set of summary statistics that gives the most accurate posteriors consists of: (i) average AP overshoot, (ii) spike rate, (iii) average AP width and (iv) average AHP depth. For $\bar{g}_K$, inclusion of the remaining two statistics, (v) latency to first spike and (vi) accommodation index, does not lead to a noticeable difference in accuracy. Neither does using equally or importance weighted summary statistics. For $\bar{g}_{Na}$, on

the other hand, the error becomes moderately worse by including (v) latency to first spike and (vi) accommodation index. Here, using importance weights actually helps to constrain $\bar{g}_{\text{Na}}$ and improves the error in the posterior.

8.3 Summarizing Posteriors

We can assess the identifiability of the HH model's active conductance parameters by examining the locations and widths of the resulting posterior estimates. A wide, flat posterior on a parameter indicates a large number of equally optimal values, which suggests that the parameter may be unidentifiable. As outlined in Section 6.5, the goodness of fit of the inferred posteriors will be considered through the MAP estimate, 95% highest density interval (HDI) and, since we have access to the ground truths, RMSPE. We will also use posterior predictive checks (PPCs) to check for auto-consistency. To reiterate the settings of the rejection ABC sampler; we generate the following posteriors using the 0.4-quantile to determine the tolerance and the set of importance weighted summary statistics (i) average AP overshoot, (ii) spike rate, (iii) average AP width and (iv) average AHP depth.

Informed by the preceding findings, going forward we will use the set of summary statistics labelled (i)-(iv) above and also keep the inclusion of importance weights.

8.3.1 Posteriors from Informative Priors

Figure 8.8 shows the original posteriors over $\bar{g}_{\text{K}}$ and $\bar{g}_{\text{Na}}$, with summarizing metrics stated in the legends. Compared with the informative priors over the model parameters (see Figure 8.3), the updated state of knowledge represented by the posteriors is more constrained. The MAP estimates are centered close on the ground truth parameters, which, by the definition of MAP, means that the ground truth parameters are in regions of high posterior density. Compared to the $\bar{g}_{\text{Na}}$ posterior, the RMSPE of the $\bar{g}_{\text{K}}$ posterior is slightly larger, even though the 95% HDI of $\bar{g}_{\text{K}}$ is narrower than that of $\bar{g}_{\text{Na}}$. This might be a bit surprising, but can be explained by noticing the sharpness of the posterior peaks. The peak of the $\bar{g}_{\text{K}}$ posterior is more flat compared to the peak of the $\bar{g}_{\text{Na}}$ posterior, which is quite sharp about the ground truth, and this is reflected in the RMSPE measure.

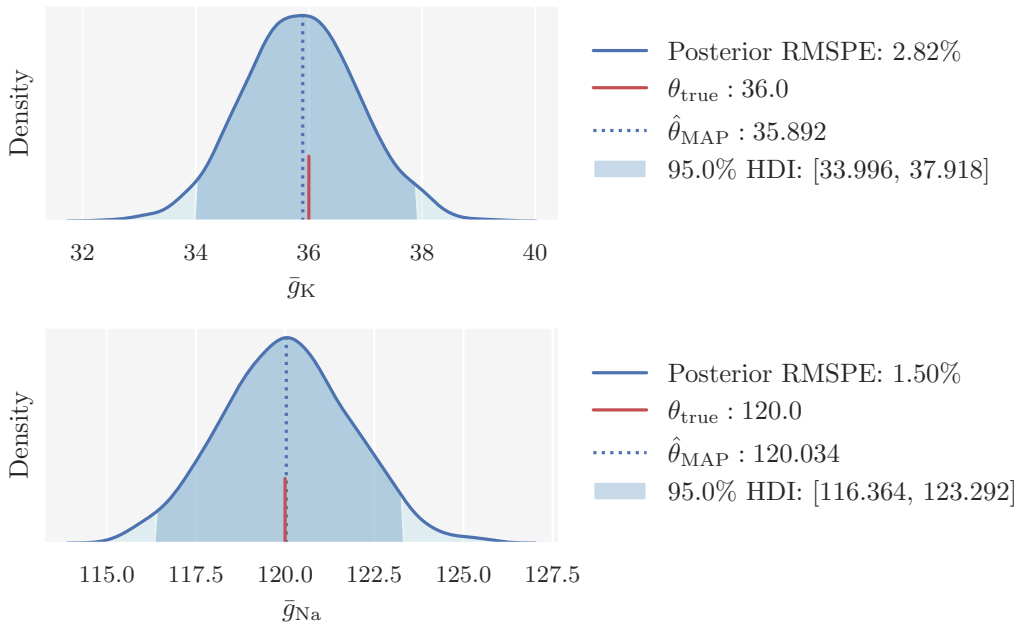
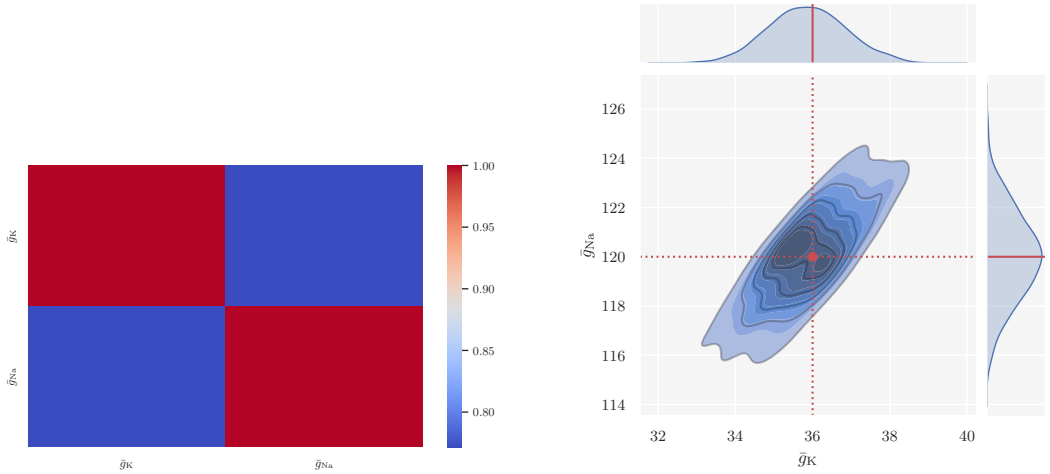


Figure 8.8: Original rejection ABC posteriors over the Hodgkin-Huxley model parameters $\bar{g}_{\mathrm{K}}$ (top) and $\bar{g}_{\mathrm{Na}}$ (bottom). Here, the parameter proposals were sampled from the joint informative prior distribution. The dark shaded region indicates the 95% HDI and the dotted line the MAP estimate. The ground truth is indicated by the red line. The legend states the numerical values for each of these, in addition to the RMSPE in the posterior.

Figure 8.9 maps the correlation, in terms of pairwise Pearson's correlation coefficients, between the posterior samples of $\bar{g}_{\mathrm{K}}$ and $\bar{g}_{\mathrm{Na}}$. It shows that the parameter samples are indeed strongly correlated. Consequently, for predictive sampling from the posteriors, we need to sample from the *joint posterior*, also shown in the figure.

Figure 8.10 shows the regression adjusted posterior. Here, $\bar{g}_{\mathrm{K}}$ is constrained to the interval $[35.986, 36.029]$ with 95% probability, and $\bar{g}_{\mathrm{Na}}$ to the $[120.001, 120.423]$ with 95% probability. The RMSPE in both posteriors are consequently reduced significantly compared with RMSPE in the original posteriors (Figure 8.8). Again, we see that the corrected $\bar{g}_{\mathrm{K}}$ samples become more constrained than the $\bar{g}_{\mathrm{Na}}$ samples, due to the stronger linear relationship $\bar{g}_{\mathrm{K}}$ has with the summary statistics. The regression adjustment overshoots the ground truth of $\bar{g}_{\mathrm{Na}}$ by a tiny amount, but the estimated parameter range could be equally capable of describing the observed data. The regression adjusted posteriors are able to identify the conductance parameters remarkably well, which is promising for using ABC algorithms to identify parameters in other conductance-based neural models based on the HH formalism as well.



(a) **(b)**

Figure 8.9: **(a)** The pairwise Pearson's correlation coefficients for the posterior samples of $\bar{g}_{\mathrm{K}}$ and $\bar{g}_{\mathrm{Na}}$. **(b)** The joint posterior distribution of $\bar{g}_{\mathrm{K}}$ and $\bar{g}_{\mathrm{Na}}$. Darker regions correspond to higher density, and the ground truth is indicated by the red marker and axis lines. Since the marginal posteriors over model parameters (shown on the marginal axes) are highly correlated, predictive posterior samples need to be sampled from the joint posterior.

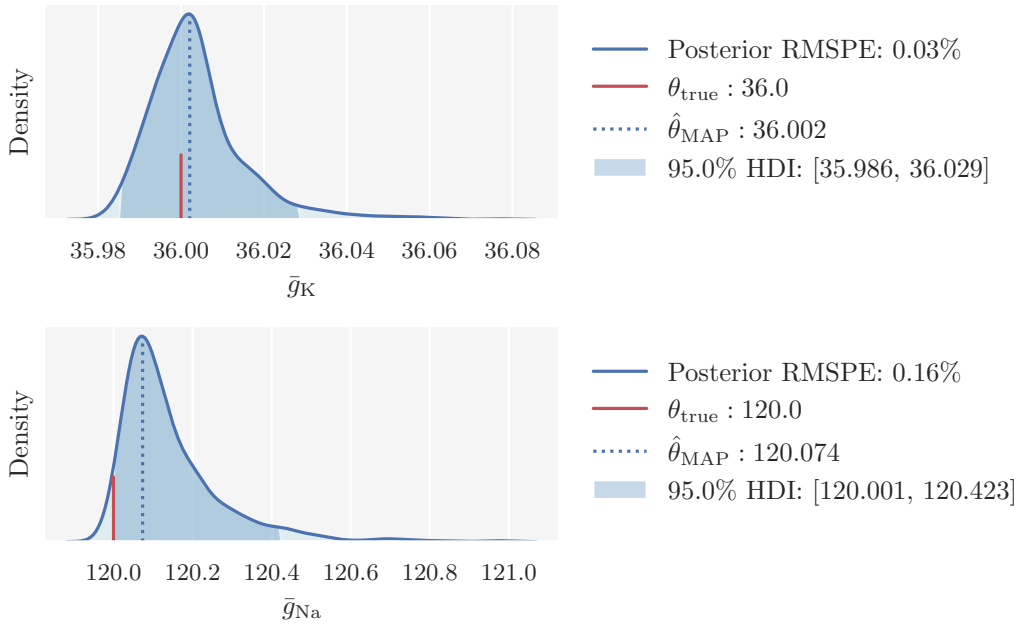


Figure 8.10: Regression adjusted rejection ABC posteriors over the Hodgkin-Huxley model parameters $\bar{g}_{\mathrm{K}}$ (top) and $\bar{g}_{\mathrm{Na}}$ (bottom). Here, the parameter proposals were sampled from the joint informative prior distribution. The dark shaded region indicates the 95% HDI and the dotted line the MAP estimate. The ground truth is indicated by the red line. The legend states the numerical values for each of these, in addition to the RMSPE in the posterior.

To verify that the parameter ranges in the regression adjusted posteriors are able to describe the observed data well, we perform a PPC. We draw 100 samples from the joint posterior predictive distribution and feed the parameters to the HH simulator. We then take the average of the simulated voltage traces. The result can be seen in Figure 8.11, where we plot the predicted simulations and their mean together with the observed voltage trace. As can be seen from the figure, the samples from

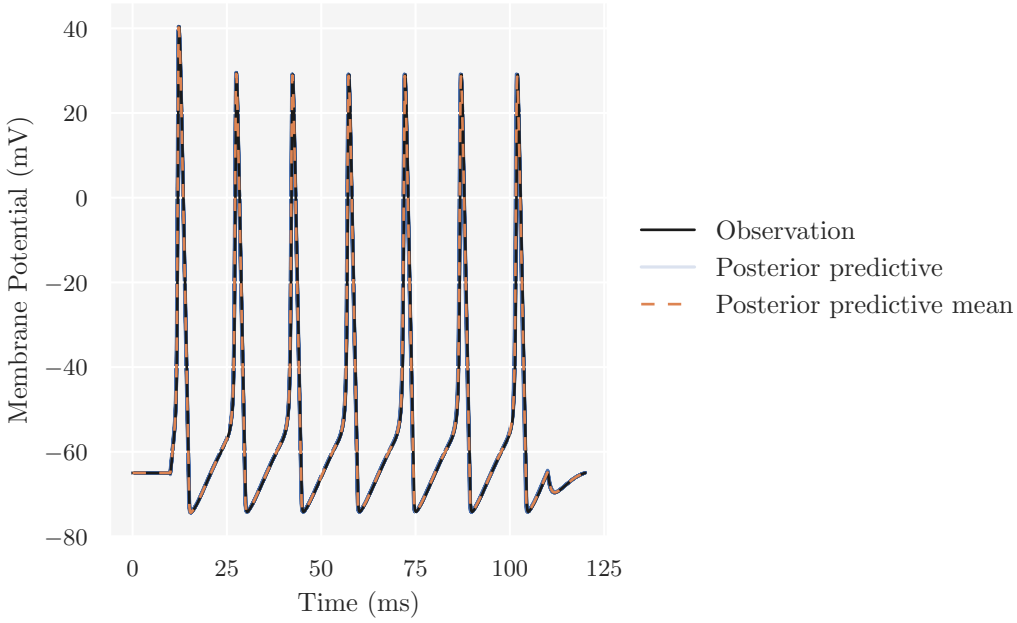


Figure 8.11: Graphical posterior predictive check comparing the observed voltage trace to simulated data predicted by the Hodgkin-Huxley model under the regression adjusted joint posterior predictive distribution.

the inferred joint posterior lead to simulations that are virtually identical to the observed data, confirming that the procedure succeeds at capturing the observed data and identifying the underlying parameters.

8.3.2 Posteriors from Noninformative Priors

By now, it has been demonstrated that regression adjustment of the posteriors is crucial for improving the accuracy of estimates from the rejection ABC sampler. Figure 8.12 shows the regression adjusted posteriors over $\bar{g}_{\mathrm{K}}$ and $\bar{g}_{\mathrm{Na}}$ when we use noninformative priors. Compared with the regression adjusted posteriors where we used informative priors, the present posteriors are only marginally less accurate. This means that, even with data-driven ABC inference, the HH conductance parameters can be accurately identified.

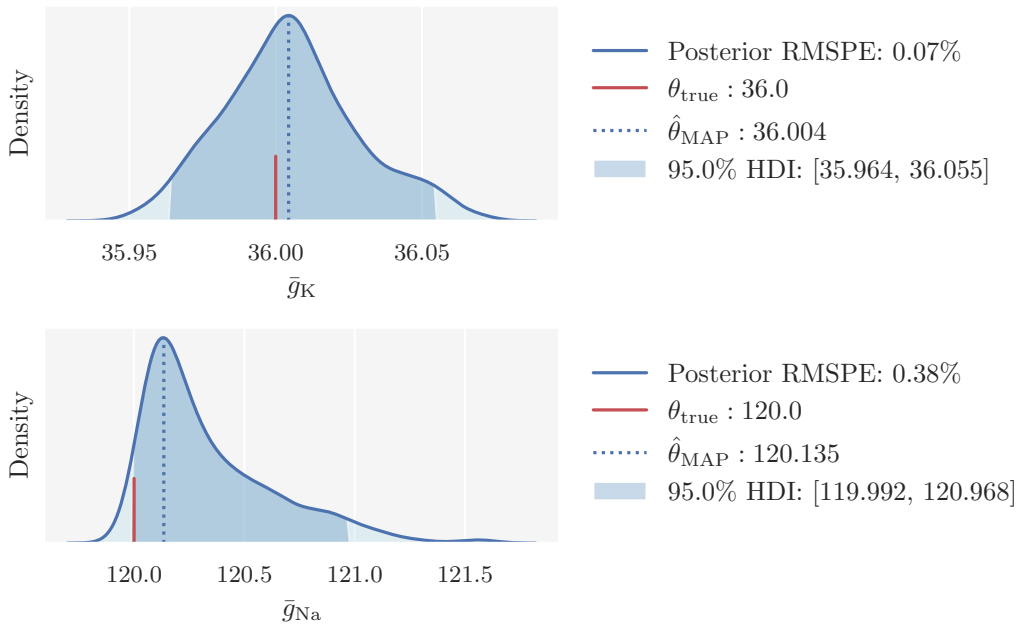


Figure 8.12: Regression adjusted rejection ABC posteriors over the Hodgkin-Huxley model parameters $\bar{g}_{\mathrm{K}}$ (top) and $\bar{g}_{\mathrm{Na}}$ (bottom). Here, the parameter proposals were sampled from the joint noninformative prior distribution. The dark shaded region indicates the 95% HDI and the dotted line the MAP estimate. The ground truth is indicated by the red line. The legend states the numerical values for each of these, in addition to the RMSPE in the posterior.

8.4 SNPE Posteriors

Rejection ABC is one of simplest simulation-based inference algorithms. We have added certain refinements to the standard rejection ABC sampling procedure, such as regression adjustment, weighted Euclidean distance and quantile-based rejection. Now, we will compare the posteriors obtained through our implementation of ABC in `pyLFI` to one of the more recent advancements in the field; neural density estimation (NDE). In particular, the NDE algorithm SNPE introduced in Section 3.4. The objective here is not to perform an exhaustive analysis of SNPE and its capabilities, as this is demonstrated in the original papers [6], [7] and [8]. Here, and in subsequent analyses with SNPE, the aim is to compare how its estimated posteriors compare with the ABC posterior under similar settings. This means that we will train the neural density estimator to learn the association between summary statistics of the data and the underlying parameters. As neural density estimator we use a particular normalizing flow (NF) called *masked autoregressive flow* (MAF) that is developed by Papamakarios et al. [53]. SNPE is given a modified HH simulator, which returns the same set of summary statistics we used in the preceding analyses; (i) average AP overshoot, (ii) spike rate, (iii) average AP width and (iv) average AHP depth. Moreover, we use the same noninformative priors as in Figure 8.3, and

train the network on 1000 simulations. The resulting posteriors over $\bar{g}_{\mathrm{K}}$ and $\bar{g}_{\mathrm{Na}}$ are shown in Figure 8.13.

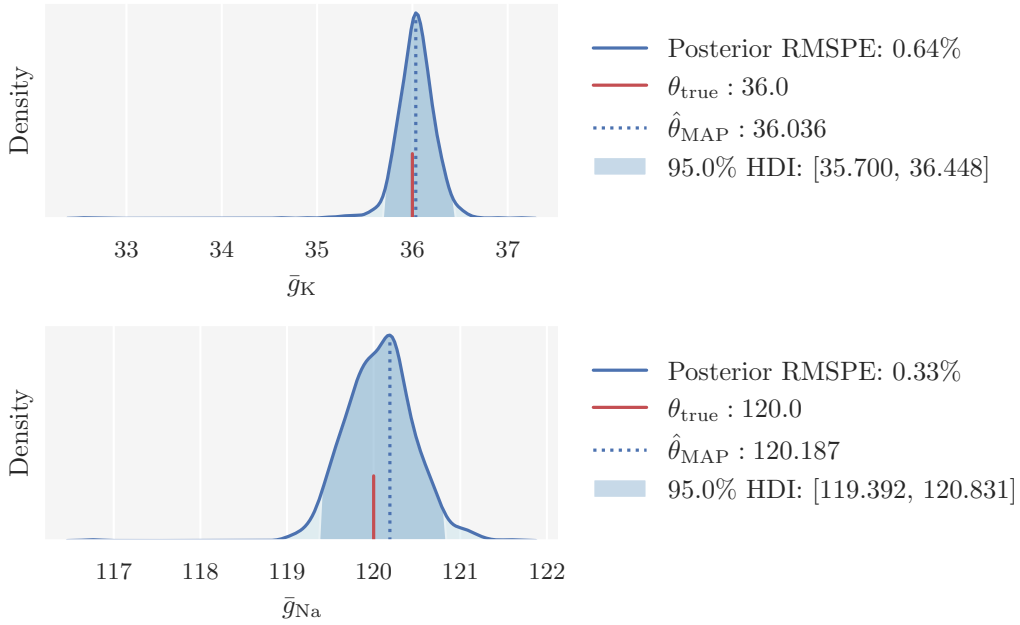


Figure 8.13: SNPE posteriors over the Hodgkin-Huxley model parameters $\bar{g}_{\mathrm{K}}$ (top) and $\bar{g}_{\mathrm{Na}}$ (bottom). The dark shaded region indicates the 95% HDI and the dotted line the MAP estimate. The ground truth is indicated by the red line. The legend states the numerical values for each of these, in addition to the RMSPE in the posterior.

Both posteriors we obtain are narrow and sharply peaked about the ground truth parameters, meaning that SNPE is able to identify admissible parameters as well. The summaries of the estimated posteriors are similar to the ones for the regression adjusted ABC posteriors. Though it is a close competition, the ABC posteriors are actually slightly more narrow than the SNPE posteriors. Moreover, the outliers observed in the tails of the SNPE posterior are not present in the ABC posteriors. This comparison might not be entirely fair to SNPE, as it is difficult to pin-point exactly how many training simulations that measure up to be "under similar settings". By training the network on even more simulations, the SNPE posteriors could perhaps be made even more narrow. Furthermore, it is likely that the parameter ranges in the SNPE posteriors all are compatible with the observed data, which will be illuminated in more detail in the next section.

8.5 Noisy Observation

So far we have only used an idealized voltage trace, free of any noise, as the observed data. However, real-world neural data are quite noisy. Here, we will examine the

impact a noisy observed recording has on the inference with the rejection ABC sampler. There are several sources to noise in cellular dynamics, and, in extension, ways to introduce noise to the Hodgkin-Huxley equations, see e.g. [54] for a review. We will introduce noise to the observed voltage trace by using a stochastic version of the HH model that incorporates current noise as a Gaussian white noise process. Besides the inclusion of current noise, we use the same settings for the HH simulator as earlier and record the voltage trace seen in Figure 8.14. The corresponding

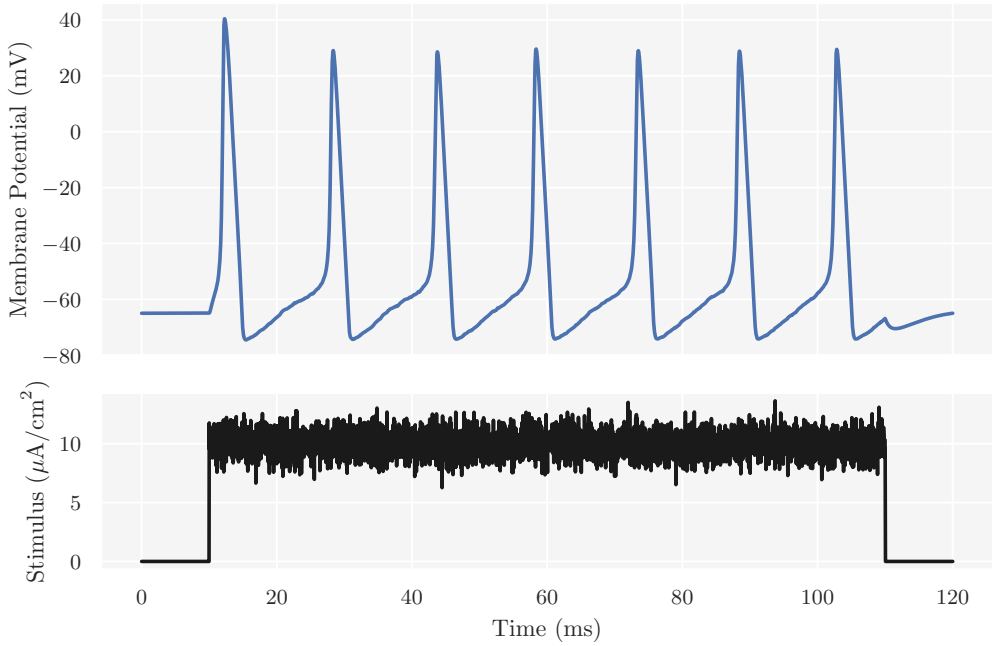


Figure 8.14: Noisy voltage trace generated with the HH simulator by introducing Gaussian white noise to the input stimulus. The parametrization of the HH model and simulation parameters are identical to the ones used in preceding noise-free voltage trace (Figure 8.1).

summary statistics are tabulated in Table 8.2.

Table 8.2: Summary statistics extracted from the noisy observed voltage trace.

Summary statistic	Observed value
Number of spikes	7
Spike rate	0.0700 mHz
Average AP overshoot	30.7223 mV
Average AP width	2.0679 mV
Average AHP depth	-74.3394 mV
Latency to first spike	2.2750 ms
Accommodation index	-0.0067

We then use the rejection ABC sampler in `pyLFI` and a HH simulator that generates noise-free simulations to infer the underlying parameters in the observed voltage trace. Though our observation is not a particularly noisy recording, it still distorts the regression adjusted ABC posteriors significantly compared to when we used noise-free observed data, as seen in Figure 8.15. As can be seen, the ground truth

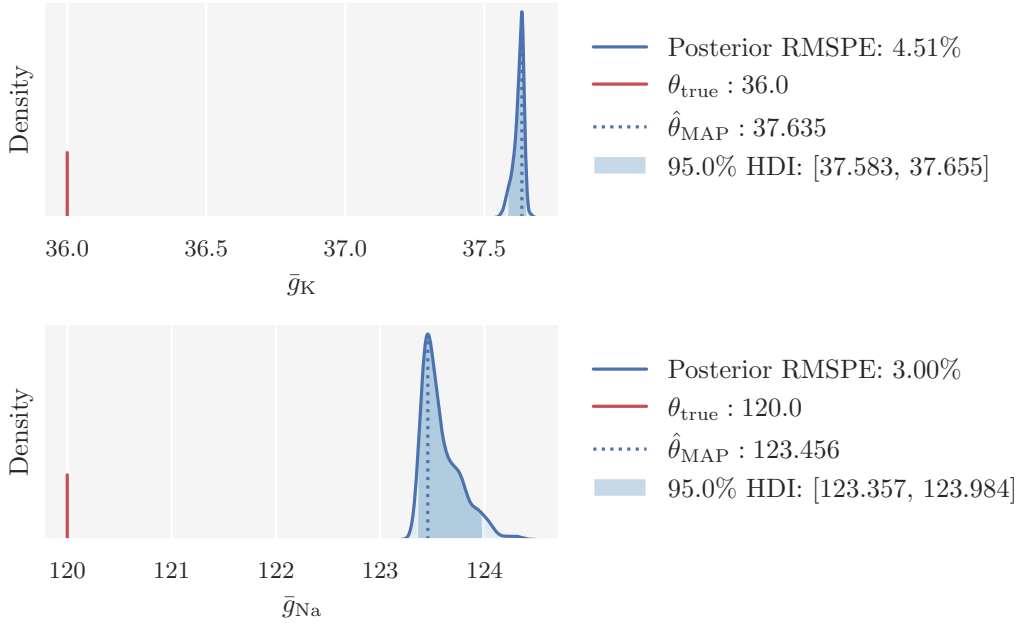


Figure 8.15: Regression adjusted rejection ABC posteriors over the Hodgkin-Huxley model parameters $\bar{g}_{\mathrm{K}}$ (top) and $\bar{g}_{\mathrm{Na}}$ (bottom) with noisy observed voltage trace. Here, the parameter proposals were sampled from the joint informative prior distribution. The dark shaded region indicates the 95% HDI and the dotted line the MAP estimate. The ground truth is indicated by the red line. The legend states the numerical values for each of these, in addition to the RMSPE in the posterior.

parameters are no longer in regions of high posterior density – they are not actually included in the posteriors at all. Though both the $\bar{g}_{\mathrm{K}}$ and $\bar{g}_{\mathrm{Na}}$ posteriors are narrow and sharp, their locations in parameter space are shifted toward a completely different set of parameter values, especially $\bar{g}_{\mathrm{Na}}$, than we found with the noise-free observation. However, if we do a graphical PPC, as seen in Figure 8.16, we find that the simulations predicted by the joint posterior match the observed voltage trace surprisingly well. This example bolster the motivation for why the Bayesian approach to inference should be considered; there might be multiple parameter settings that are consistent with the observed data.

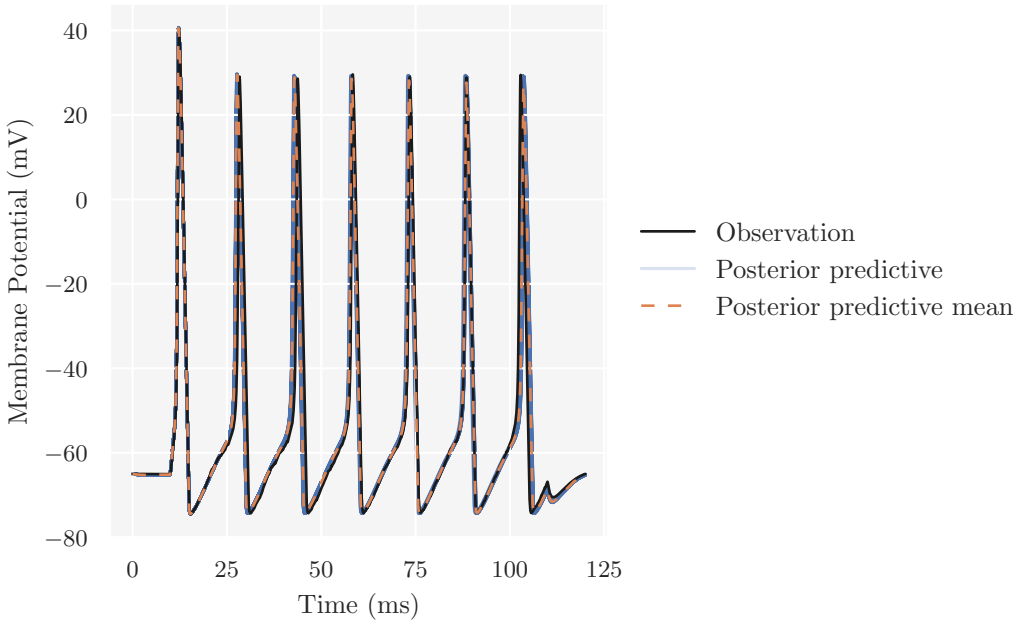


Figure 8.16: Graphical posterior predictive check comparing the noisy observed voltage trace to simulated data predicted by the Hodgkin-Huxley model under the regression adjusted joint posterior predictive distribution. The posterior predictive mean is the average of 100 predicted simulations.

9

Inference on the Brunel Model

In this chapter, we present the results from simulation-based inference on the Brunel network model's synaptic weight parameters η and g.

We will first try to identify the synaptic weight parameters with the quantile-based rejection ABC sampler in `pyLFI` and post-sampling local linear regression adjustment with the Epanechnikov kernel. The observed data will be from the network's AI state, synthetically generated from the Brunel simulator in `NeuroModels`. The parametrization of the Brunel network will be as given in Table 5.2.

We will then try to identify the parameters with SNPE. Here, we will try to utilize the flexibility of SNPE by training on simulations from both the AI and SR states, and see what posteriors it will predict when presented with observed data from one of these states.

9.1 Inference with ABC

9.1.1 Observation from AI State

We create a Brunel network with $10,000$ excitatory and $2,500$ inhibitory neurons. Each neuron is randomly connected with 1000 excitatory and 250 inhibitory neurons. The synaptic weight parameters are set as $\eta = 2$ and $g = 5$, which according to the phase diagram of the network (Figure 5.5) corresponds to the AI state. We simulate the network for $1,000$ ms and record the output spike trains from 20 excitatory neurons. We start recording after 100 ms to avoid transient effects. The observed network activity is shown in Figure 9.1. As seen in the figure, neurons in the AI state are weakly correlated and fire irregularly at low rates. The observed

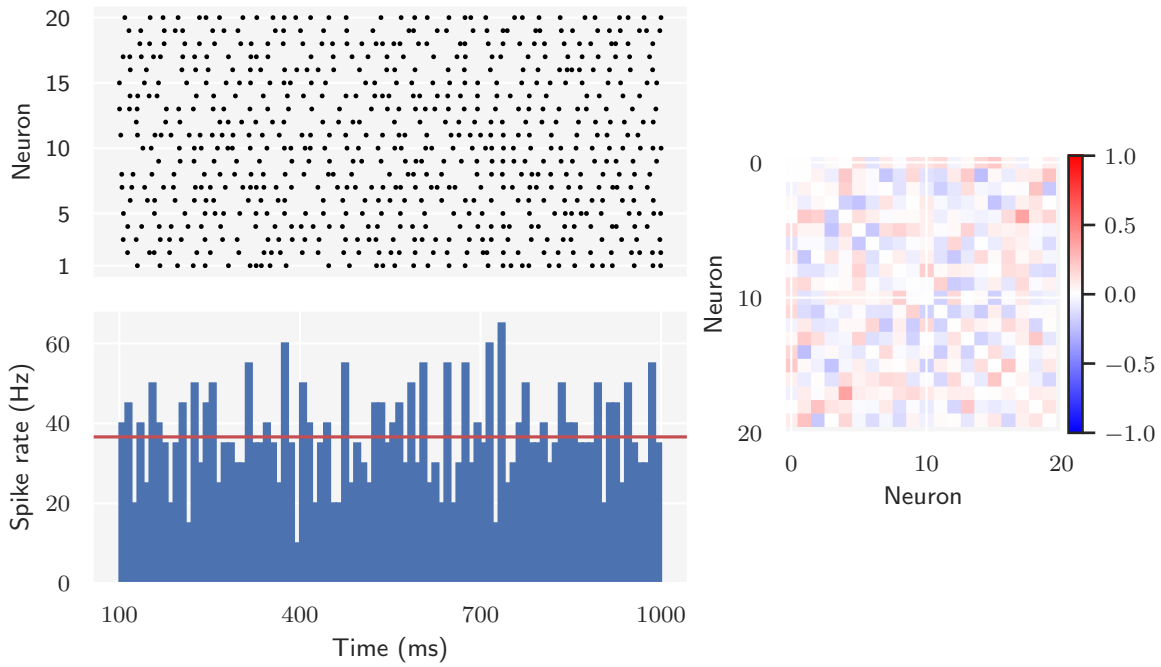


Figure 9.1: Observed network activity recorded from 20 excitatory neurons in the Brunel network's AI state. The synaptic weight parameters are set as $\eta = 2$ and $g = 5$, and the remaining parameters according to Table 5.2. The network is simulated for $1,000$ ms and recording start after 100 ms. The top left panel shows the firing times (raster) of the recorded neurons, and the bottom left panel the network activity as a time resolved firing rate computed in bins of 10 ms. The mean firing rate is indicated by the horizontal (red) axis line. The right panel shows the pairwise Pearson's correlation coefficient matrix of the recorded neurons, which is a measure of how synchronous the spiking of the network is.

activity is reduced to the set of low-dimensional summary statistics outlined in Section 6.2.2; (i) mean firing rate; (ii) mean CV; (iii) Fano factor. Table 9.1 tabulates the calculated summary statistics from the observed AI activity. Each

Table 9.1: Observed summary statistics in the AI state.

Summary statistic	Observed value
Mean firing rate	0.0366 kHz
Mean CV	0.4250
Fano factor	0.2341

statistic captures different aspects of the activity. The mean firing rate is a direct

measure of the population's spiking activity, mean CV measures the regularity of spike trains and Fano factor the variability across spike trains.

9.1.2 Correlation Analysis & Importance Weights

As we did for the inference on the HH model, we first assess how sensitive the summary statistics are to movement in parameter values, and then measure the relationship between them with a correlation analysis. This analysis is done by sampling from the prior predictive distribution, and the priors for η and g are shown in Figure 9.2. We have here chosen noninformative priors, with parameter

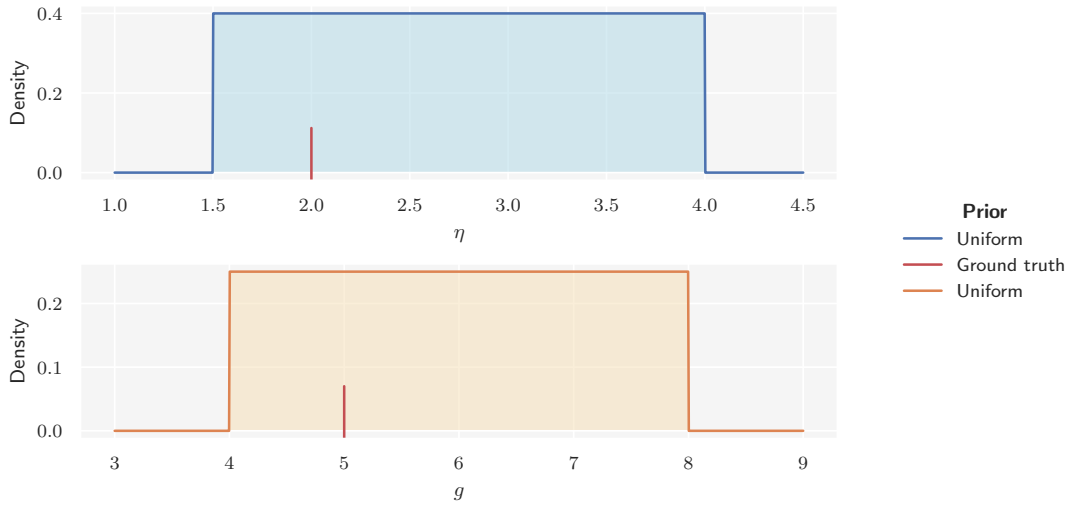


Figure 9.2: Priors over η (top) and g (bottom). We use noninformative priors with ranges corresponding to the AI state. The ground truth parameters are indicated by the red lines.

ranges that correspond to the AI state, as seen in the phase diagram over the Brunel network's states (Figure 5.5). We then draw 2000 samples from the joint prior predictive distribution, feed each parameter pair to the Brunel simulator and calculate the resulting summary statistics from each simulation. Figure 9.3 shows a subset of the generated samples as scatter plots, where each point is the simulated statistic for a given pair of parameter values. The relative magnitude of a statistic is indicated by its size and color, with a reference table stated in the legend along with the name of the particular statistic. The scatter plots show that all the summary statistics exhibit a steady variability. In particular, for the model parameter g, which controls the amount of inhibition in the network, the relationship with each of the statistics seems to follow an approximately linear trend. However, for η, which determines the strength of the external drive, a distinct systematic relationship with the statistics is less pronounced. From a biophysical point of view, this is not entirely surprising. AI activity is a hallmark of recurrent networks, and arise from that excitation is balanced by inhibition. It can persist even in the absence of external input. As such, the dynamics of the network tend to be more dependent

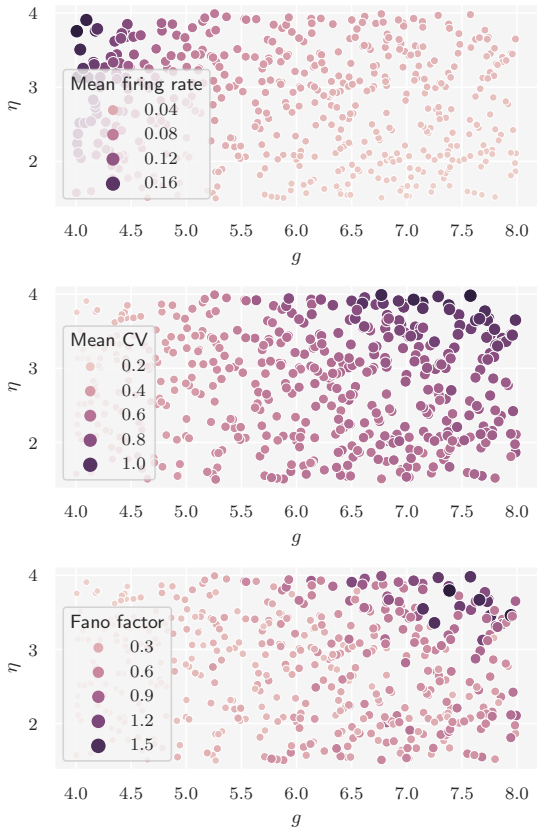
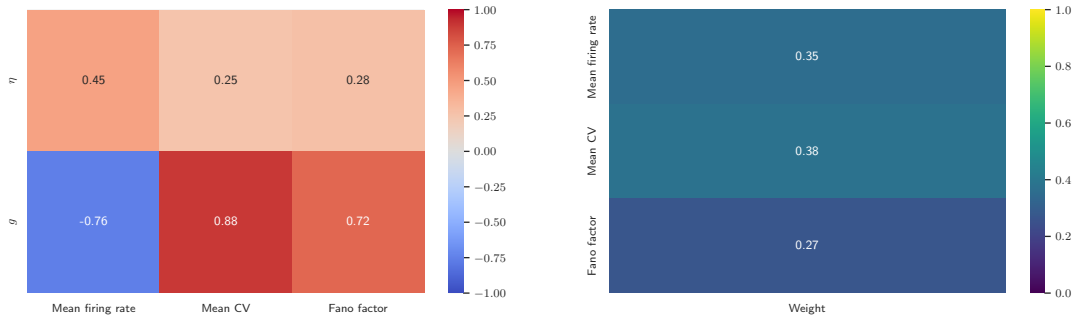


Figure 9.3: Scatter plots of summary statistics simulated with different pairs of model parameter values. The summary statistics were simulated under the joint prior predictive distribution. Here a subset of 500 samples out of 2000 is shown. Each summary statistic is assigned to its own panel, with the particular statistic stated in the legend. Each point represents the value of a summary statistic for a pair of parameter values, (g, η). The color of a point indicates the relative magnitude of the statistic, for which bright colors represent small and dark colors large values, also indicated in each subplot legend. The scatter plots thus indicate the variability of summary statistics relative to movement of the pairs of model parameter values.

on g than η, and we can expect, at least with this set of summary statistics, that g will be constrained better than η.

The pairwise Pearson's correlation coefficients in the left panel of Figure 9.4 confirm the observation of a stronger linear relationship between the summary statistics and g than with η. Furthermore, as seen in the right panel, the importance weights for the summary statistics calculated from the correlation coefficients have roughly the same magnitude. This implies that all the summary statistics encode useful information about the activity for constraining the model parameters, and that the information they encode is nearly equally important.



(a) (b)

Figure 9.4: **(a)** The pairwise Pearson's correlation coefficients for the model parameters and summary statistics. **(b)** The importance weights calculated from the correlation coefficients, see Section 6.3 for details. Note that the weights sum to one.

9.1.3 ABC Settings

The Brunel simulator has costly simulations. We will therefore use the approach where we set a *simulation budget*, discussed in Section 7.2. This means that the simulator only will run the specified amount of times and the number of posterior samples we retain depend on which quantile is set. This is in contrast to the approach we used for the HH model, where a pilot study was used to estimate the tolerance based on the provided quantile and the ABC sampler used this tolerance for obtaining the specified amount of posterior samples. To examine the impact the choice of quantile has on the accuracy of the posteriors, we use a simulation budget of $2,000$ model simulations and compute the RMSPE for different choices of quantile. The result is shown in Figure 9.5. As expected, increasing the p_ϵ-quantile gives an increase in the RMSPE, since simulations further away from the observed data are accepted. Moreover, we see once more that regression adjustment of the posteriors is necessary to obtain accurate estimates. We also see that g is more constrained by the summary statistics than η, which falls in line with our expectations from the previous analysis. However, regression adjustment significantly helps to constrain η as well.

In the subsequent analyses, we keep the simulation budget of $2,000$ simulations. As a compromise between accuracy and the number of samples in the posterior, we will use the 0.3-quantile as a measure of tolerance.

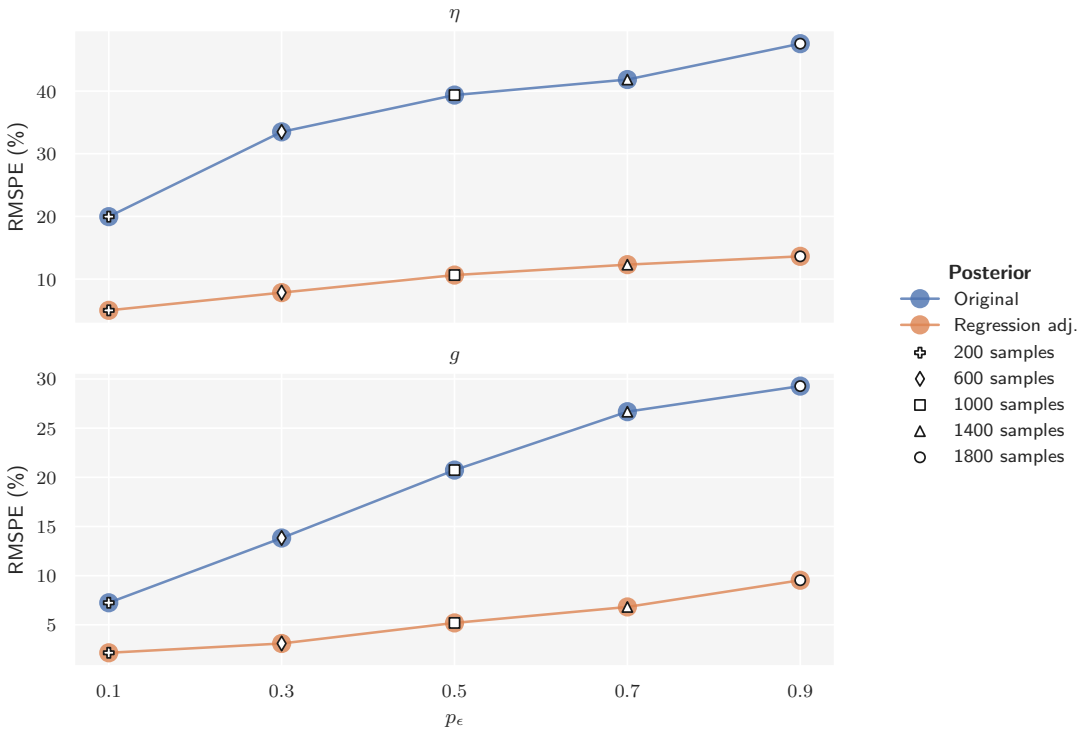


Figure 9.5: The RMSPE in posteriors over η (top) and g (bottom) against the p_ϵ-quantile as a measure of tolerance. A simulation budget of $2,000$ was used, and the number of posterior samples that are retained for a particular p_ϵ-quantile is indicated by the marker inside the point. The corresponding label in the legend states the number of posterior samples retained for that particular quantile. The posteriors were generated by the rejection ABC algorithm and then adjusted with local linear regression adjustment. Whether an estimate of error is from the original/adjusted posterior is color coded (see legend).

9.1.4 Summarizing Posteriors

The regression adjusted posteriors over η and g are shown in Figure 9.6. In the updated state of knowledge, the prior range of $\eta \in [1.5, 4.0]$ has been constrained to $\sim \eta \in [1.6, 2.1]$ with 95% probability, whereas the prior range of $g \in [4.0, 8.0]$ has been constrained to $\sim g \in [4.7, 5.3]$ with 95% probability. Even though both ground truth parameters lie in regions of high posterior density, it is only g that is identified moderately accurate. The posterior over g is sharply peaked, though it is not particularly narrow. The posterior over η, on the other hand, has a relatively flat peak compared to g, and the identification of η is therefore less successful as a wide and flat posterior indicates a large number of equally optimal values. Although this aligns with our biophysical expectations, these results illustrate the limitation of the rejection ABC approach where proposal parameters are only sampled from the prior. By using a sampler which updates the proposals recursively, like the MCMC

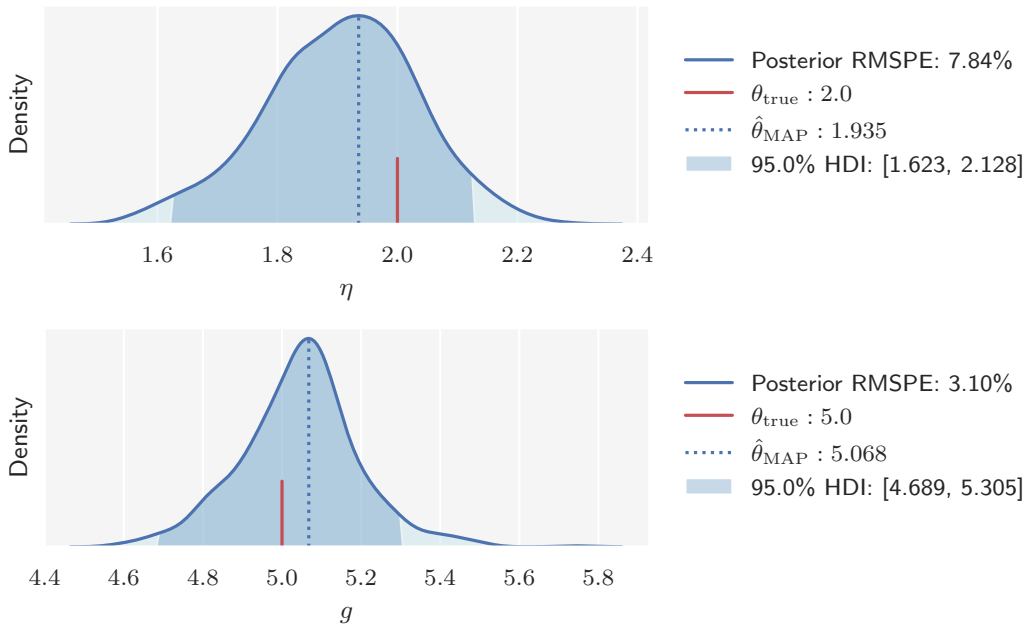


Figure 9.6: Regression adjusted rejection ABC posteriors over the Brunel network model parameters η (top) and g (bottom) with observed data from the AI state. The dark shaded region indicates the 95% HDI and the dotted line the MAP estimate. The ground truth is indicated by the red line. The legend states the numerical values for each of these, in addition to the RMSPE in the posterior.

ABC algorithm discussed extensively in this thesis (see e.g. Section 3.2), more efficient sampling of parameters from high density regions can be achieved.

Figure 9.7 illustrates the joint posterior over η and g. It can be seen that the joint ground truth (red marker) lies in a region of high posterior density.

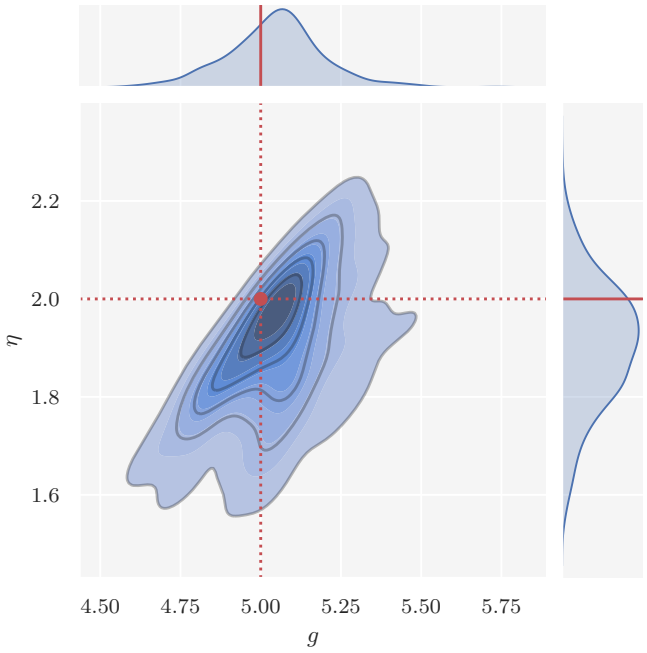


Figure 9.7: The regression adjusted joint posterior distribution over η and g in the AI state. Darker regions correspond to higher density, and the ground truth is indicated by the red marker and axis lines. Since the marginal posteriors over model parameters (shown on the marginal axes) are highly correlated, predictive posterior samples need to be sampled from the joint posterior.

9.1.5 Posterior Predictive Checks

To see whether particular summaries of the network activity are mapped accurately by the posterior predictions, we perform a graphical PPC. We draw 50 samples from the regression adjusted joint posterior predictive distribution and feed the parameters to the Brunel simulator. We then calculate the summary statistics, i.e., (i) mean firing rate; (ii) mean CV; (iii) Fano factor, for each of the output spike trains. The simulated summary statistics and their average together with the corresponding observed summary statistic are then plotted in summary statistic space. The result can be seen in Figure 9.8. From the figure it can be seen that most of the predicted summary statistics are in the neighborhood of the observation, but they can also be relatively far away. This is not unexpected though. Due to the stochastic nature of spike generation in networks, activity simulated under the exact same settings will differ among themselves to a significant degree. This intrinsic variability may have an important functional role in biological neural networks, but pose a challenge for fitting models to data. Since the target observations themselves are variable, the approach used here, where we fit to just a single observation, might not be the best method. Even though we are able to constrain the model parameters to some extent, the intrinsic variability of the observation, and hence

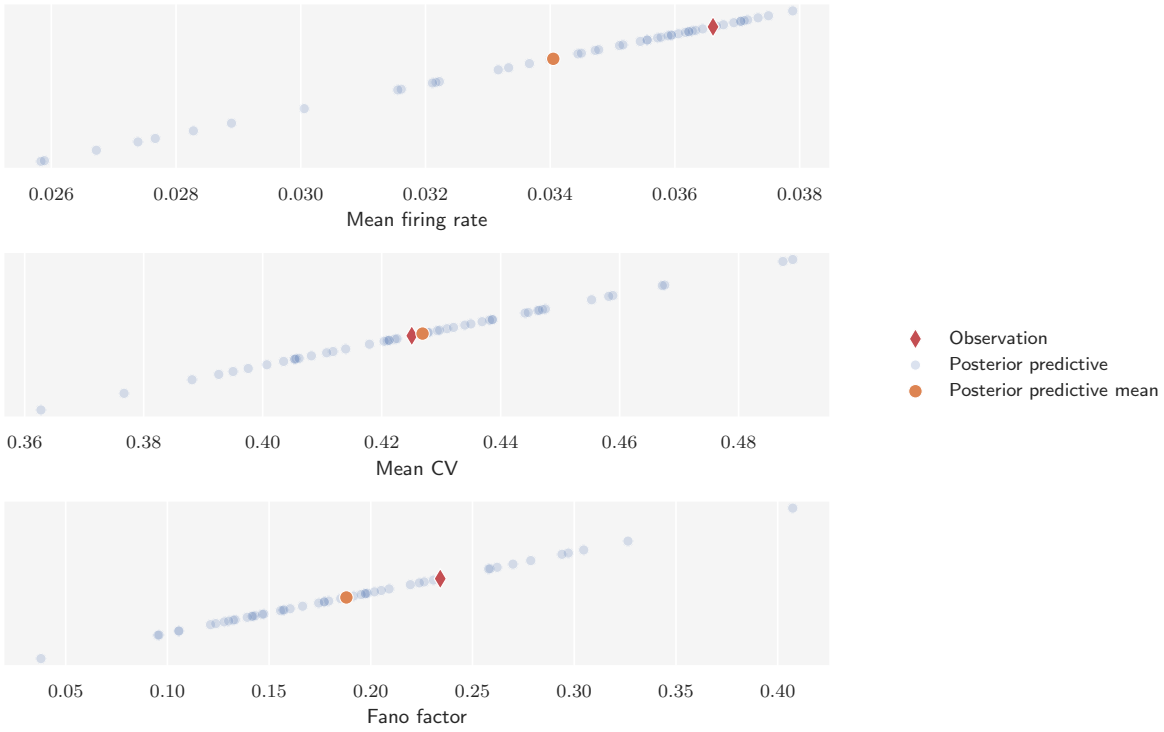


Figure 9.8: Graphical posterior predictive check comparing the observed summary statistics with simulated summary statistics predicted by the Brunel model under the regression adjusted joint posterior predictive distribution. The posterior predictive mean is the average of 50 predicted simulations.

the underlying data generating process, should be taken into account before any general insights can be proclaimed.

9.2 Inference with SNPE

9.2.1 Training the Neural Density Estimator

Next, we will infer the Brunel network model's synaptic weight parameters with SNPE. We use the same configuration as for the HH model, i.e., MAF as neural density estimator and training on 1000 model simulations. The objective of the neural density estimator is to map an association between the simulated summary statistics, (i) mean firing rate; (ii) mean CV; (iii) Fano factor, and the underlying parameters η and g. As priors we use $\pi(\eta) = \mathrm{U}(1.5, 4)$ and $\pi(g) = \mathrm{U}(1.5, 8)$, where the latter corresponds to network dynamics in both the AI and SR states (see Figure 5.5). As mentioned before, the goal is to examine whether SNPE will be able to predict posteriors that fall within the parameter ranges of the state it is presented with as observed data.

9.2.2 Inference in the AI State

Figure 9.9 shows the posteriors over η and g when the SNPE density estimator is presented with the same observed activity from the AI state as earlier. The

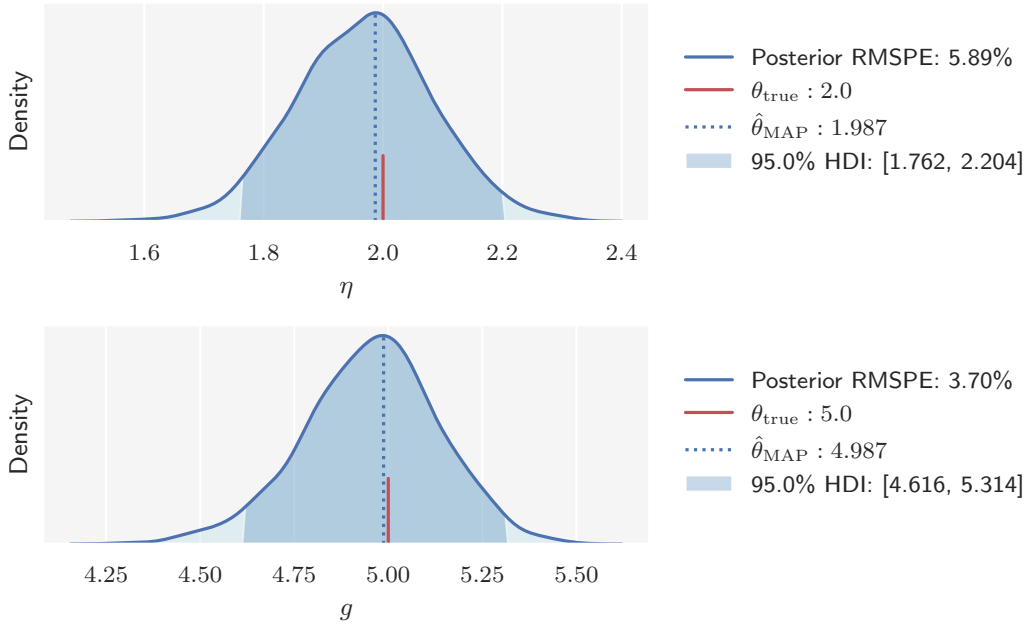


Figure 9.9: SNPE posteriors over the Brunel network model parameters η (top) and g (bottom) with observed data from the AI state. The dark shaded region indicates the 95% HDI and the dotted line the MAP estimate. The ground truth is indicated by the red line. The legend states the numerical values for each of these, in addition to the RMSPE in the posterior.

SNPE and regression adjusted ABC posteriors for the Brunel model parameters are

almost identical, as we also found with the HH model. A minor difference is that the peaks of the SNPE posteriors are slightly more centered about the ground truth parameters, which means that the ground truth parameters are in regions of slightly higher posterior density. The minimal differences between the posteriors of SNPE and ABC are not our primary interest here. Rather, we take notice of the fact that SNPE is able to accurately place the posteriors in a range corresponding to the AI state. Figure 9.10 shows the PPC for the joint SNPE posterior. Compared with the PPC for the joint ABC posterior, the mean SNPE predictions are closer to the observations. However, the same variability due to the stochastic nature of spike generation is also seen here.

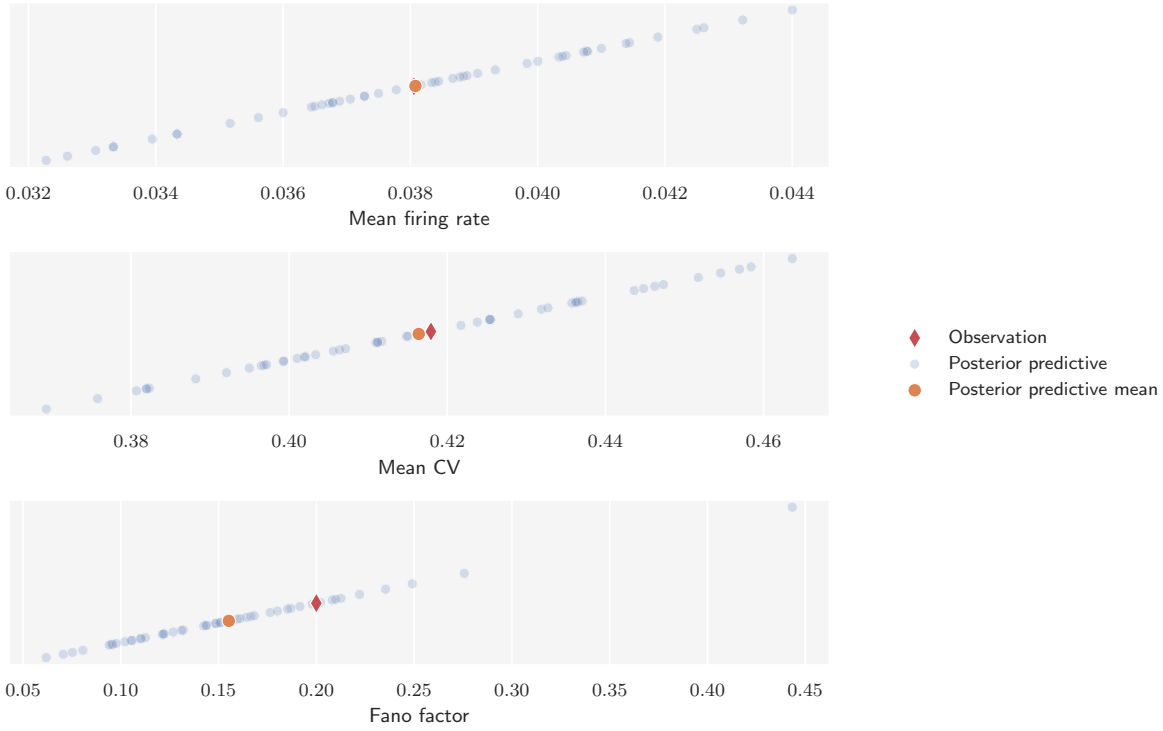


Figure 9.10: Graphical posterior predictive check comparing the observed summary statistics with simulated summary statistics predicted by the Brunel model under the regression adjusted joint posterior predictive distribution. The posterior predictive mean is the average of 50 predicted simulations.

9.2.3 Inference in the SR State

We create another Brunel network with the same number of neurons and connections as earlier, but this time we set the synaptic weight parameters as $\eta = 2$ and $g = 3$, which according to the phase diagram of the network (Figure 5.5) corresponds to the SR state. We again simulate the network for $1,000$ ms and start to record the output spike trains from 20 excitatory neurons after 100 ms. The observed network

activity is shown in Figure 9.11. As seen in the figure, neurons in the SR state are

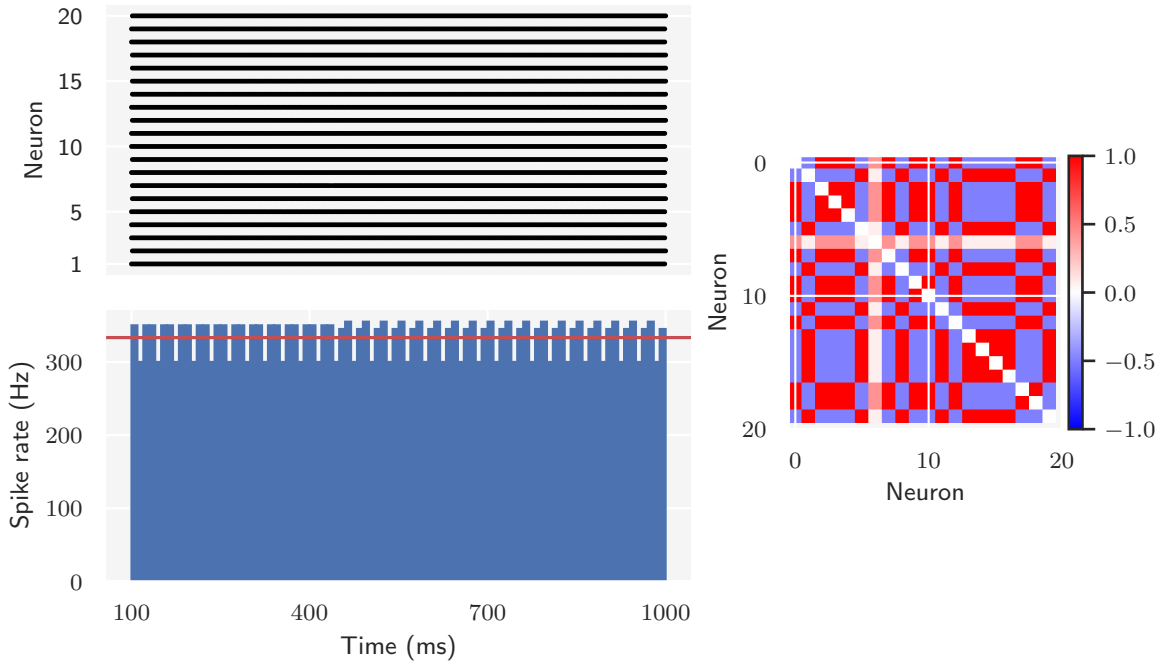


Figure 9.11: Observed network activity recorded from 20 excitatory neurons in the Brunel network's SR state. The synaptic weight parameters are set as $\eta = 2$ and $g = 3$, and the remaining parameters according to Table 5.2. The network is simulated for $1,000$ ms and recording start after 100 ms. The top left panel shows the firing times (raster) of the recorded neurons, and the bottom left panel the network activity as a time resolved firing rate computed in bins of 10 ms. The mean firing rate is indicated by the horizontal (red) axis line. The right panel shows the pairwise Pearson's correlation coefficient matrix of the recorded neurons, which is a measure of how synchronous the spiking of the network is.

strongly correlated and fire regularly at high rates. The observed activity is reduced to the same set of low-dimensional summary statistics as earlier; (i) mean firing rate; (ii) mean CV; (iii) Fano factor. Table 9.2 tabulates the calculated summary statistics from the observed SR activity.

Table 9.2: Observed summary statistics in the SR state.

Summary statistic	**Observed value**
Mean firing rate	0.3333 kHz
Mean CV	0.0121
Fano factor	0.007

Figure 9.12 shows the posteriors over η and g when the SNPE density estimator is presented with the observed activity from the SR state. We see that the SNPE

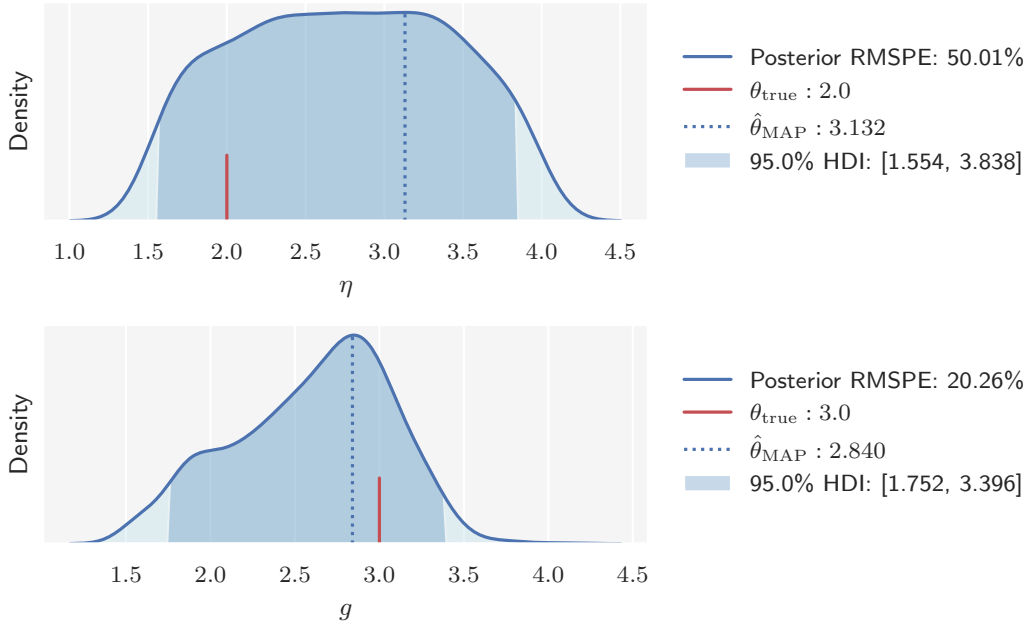


Figure 9.12: SNPE posteriors over the Brunel network model parameters η (top) and g (bottom) with observed data from the SR state. The dark shaded region indicates the 95% HDI and the dotted line the MAP estimate. The ground truth is indicated by the red line. The legend states the numerical values for each of these, in addition to the RMSPE in the posterior.

posterior over g again matches the parameter range of the observed activity state. Though the posterior is wide, it has a well-defined peak centered about the ground truth parameter. The SNPE posterior over η, on the other hand, is outstandingly flat and wide, which means that SNPE is unable to identify η in the SR state. There is, however, a biophysical explanation to this unidentifiability. In the SR state, the recurrent input from the network is high due to the low inhibition, and can become self-sustaining. In this state of activity, the role of external inputs, which are described by η and typically of a much lower magnitude, virtually vanishes. If a model parameter is not a primary driver for the observed dynamics, we will not be able to identify it accurately either. Figure 9.13 shows the PPC for the joint SNPE posterior. Here we see that the predictions are close to the observations, even though neither posterior is particularly constrained. This indicates that there are other model parameters than η and g that are the primary drivers for the dynamics observed in the SR state.

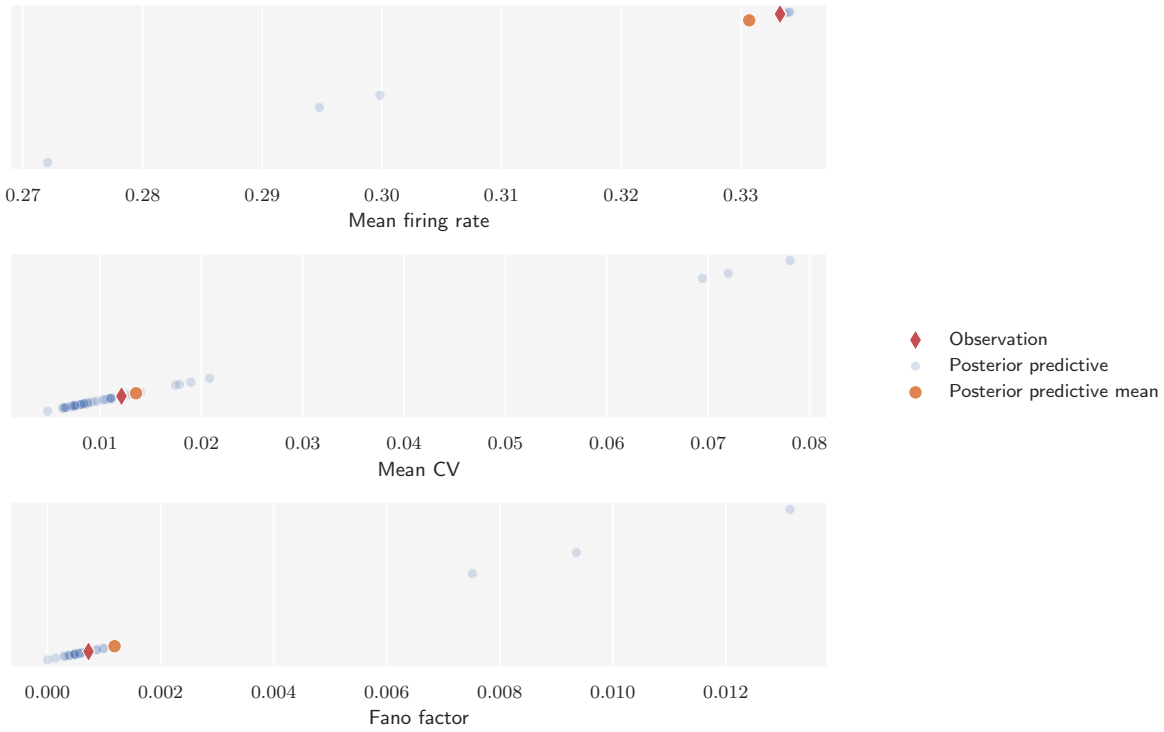


Figure 9.13: Graphical posterior predictive check comparing the observed summary statistics with simulated summary statistics predicted by the Brunel model under the regression adjusted joint posterior predictive distribution. The posterior predictive mean is the average of 50 predicted simulations.

Part IV

Summary & Conclusions

10

Summary

Mechanistic models of neural dynamics are poorly suited for inference and lead to challenging inverse problems. The objective of this thesis is to investigate the ability and utility of simulation-based inference for identifying parameters in mechanistic models of neural dynamics. We have used two simulation-based inference algorithms; (i) *Approximate Bayesian Computation* (ABC) with quantile-based rejection sampling and post-sampling regression adjustment; (ii) *Sequential Neural Posterior Estimation* (SNPE), a neural density estimation algorithm; to infer model parameters in the Hodgkin-Huxley (HH) model [9] and the Brunel network model [10]. The primary focus of the study was on ABC, and SNPE results are mostly for comparison. In the following, we summarize our findings.

As part of this thesis, we implemented a framework for simulation-based inference using rejection ABC with post-hoc corrections through regression adjustment in the Python package `pyLFI`. We characterized the properties of this method when applied to models typically used in computational neuroscience; a biophysically detailed neuron model (the HH model) and a recurrent network model of point neurons (the Brunel network model). Examination of the rejection ABC sampler's estimated posteriors revealed that model parameters can be identified accurately. The level of accuracy largely depends on choosing a set of nearly sufficient low-dimensional summary statistics. We were particularly successful in identifying the active conductance parameters in the HH model, which suggests that simulation-based inference may be a valuable tool for computational investigations concerning the many models that use the HH formalism. Daly et al. [55] carried out a similar study where they used ABC to identify the rate parameters in the HH model. Identification of the synaptic weight parameters in the Brunel network model posed a more challenging inferential problem. Due to the stochastic nature of spike

generation in neural networks, there is an intrinsic variability in both observed and simulated data. We only fitted the model to a single observation, and did thus not account for the intrinsic variability. However, whether one should aim to fit the model to a specific recording or a recording with averaged behavior depends on the objective of the investigation. Our aim was to assess identifiability of model parameters, which was achievable by only using a single recording. For a parameter to be identified, we found that it must be a primary driver of the underlying data generating process, such that it plays a role in determining the summaries extracted from the data. Thus, if we are unable to identify a parameter it does not necessarily imply that the inference procedure failed, as it can imply a structural unidentifiability of the parameter instead. Simulation-based inference can therefore be used to characterize the function of biophysical parameters. This may aid in designing better models of neural dynamics and comparison of mechanistic hypotheses with neural data.

Rejection ABC is one of simplest simulation-based inference algorithms. Since the proposal parameters are sampled from the prior, sampling is efficient and easily parallelizable. Efficiency of the overall procedure, however, is determined by whether the prior is chosen such that it is of a similar shape and location as the desired posterior. By using a sampler which updates the proposals recursively, both sampling efficiency and accuracy can be improved. An example of such a sampler, is the Markov chain Monte Carlo (MCMC) ABC algorithm. We discussed both MCMC in general and MCMC ABC in detail in the thesis, and also implemented a MCMC ABC sampler in `pyLFI`. However, in order to focus on the overall objective, we decided to not include investigations with the MCMC ABC sampler in the study. MCMC methods require a great deal of tuning and diagnosis in order to ensure that the chains sample efficiently and converge toward the stationary distribution. The use of MCMC ABC would thus have added an unnecessary layer of complexity to the study. We did nonetheless discuss the MCMC methods in such detail because they have been instrumental in advancing Bayesian statistics. The accurate results we were able to obtain with the simple rejection sampler implies that even better results can be expected by using more advanced samplers. Though, with the inclusion of regression adjustment, we found rejection ABC to be on par with the far more advanced SNPE method. However, in the present study we only investigated low-dimensional problems with synthetic observed data. One should therefore be cautious about generalizing insights this non-exhaustive study might suggest. As with all optimization algorithms, ABC suffers from the curse of dimensionality [56]. With each new parameter to identify, the volume of the parameter space the ABC sampler must search increases exponentially, leading to potentially intractable computational investigations for higher-dimensional problems. The machine learning revolution, to which SNPE belong, has allowed for working with higher-dimensional problems [26]. An applied study of the capabilities of SNPE on models of neural dynamics was carried out by Lueckmann et al. in [33]. Perhaps the most notable difference between the approach of ABC and SNPE, is that SNPE uses *all* model simulations to train

the neural density estimator opposed to ABC that usually ends up rejecting most of the simulations. Another important difference is that once the neural density estimator is trained, it can be used to predict posteriors on any empirical data with just one pass through the neural density estimator, whereas ABC would need to be performed again when presented with another set of empirical data.

The choice of summary statistics is vital in determining the outcome of the inverse modelling with ABC. The set of statistics must be low-dimensional and effectively encode phenomena of the original data in order to constrain the model parameters. We used expert-crafted statistics of spiking activity, and studied their ability to isolate informative behaviors in great detail. Some heuristic is usually necessary to single out the most useful statistics, and we used a correlation analysis to assess the sensitivity of the summary statistics to movement in model parameter values. Based on the relationship between the summary statistics and model parameters measured by the pairwise Pearson's correlation coefficients, we created importance weights where those with the strongest relationships were given heavier weights. Though we found the importance weights to help constrain the model parameters better, this weighting procedure is not particularly robust. The pairwise Pearson's correlation coefficient only reflects linear correlation of variables, and ignores other types of relationship or correlation. There are numerous robust approaches to base both the choice of summary statistics and construction of importance weights on, for instance parameter sensitivity analysis, see e.g. Tennøe et al. [40], or analysis of curvature of an objective function, see e.g. Druckmann et al. [39].

While the choice of summary statistics is of primary importance, the choice of distance metric can also have a substantial impact on the quality of the posterior approximation. In the present study, we used the Euclidean distance. This amounts to a circular acceptance region, as illustrated by Figure 3.1, which implies independence and identical scales of the summary statistics. In order to suppress domination of the summary statistics with largest scale, we used a weighted version of the Euclidean distance that scaled the summary statistics with their respective standard deviation estimated from the prior predictive distribution. Moreover, we also weighted the importance of the summary statistics based on the correlation analysis. Despite these efforts, the Euclidean distance might not have the acceptance region that results in the most accurate ABC posterior approximation. If we, reasonably, suppose that the different summary statistics are dependent and on different scales, their true distribution under the model may be better represented by an elliptical acceptance region rather than a circular. The Mahalanobis distance is an example of a distance metric with elliptical acceptance region. However, the different levels of information encoded by the different summary statistics make finding the optimal acceptance region a challenge. Even with different scales and dependences, a circular acceptance region might result in a more accurate ABC posterior approximation than an elliptical region if the circular is better tied up with the most informative statistics. How the best acceptance region is connected with the choice of summary statistics is discussed in detail by Prangle in [57].

A crucial limitation of ABC algorithms is that only a small number of summary statistics can be handled before the curse of dimensionality enters. We used regression adjustment to make the approximated posteriors more insensitive to the tolerance, which in turn permits the use of more summary statistics. In particular, we used local linear regression adjustment, which weight the parameter sets resulting in simulations close to the observation more heavily. In our implementation, individual components of a parameter set are treated separately. Thus, correlations between them are not accounted for, and we found the sets of posterior samples to be strongly correlated. Moreover, given the complex relationships between the model parameters and summary statistics we used, nonlinear regression approaches, see e.g. [25], may be better at ensuring the key insensitivity regression adjustment aims for.

11

Conclusions

We investigated how simulation-based inference can be applied to inverse modelling of mechanistic models of neural dynamics, where the likelihood is unavailable or intractable. In particular, we explored the ability of approximate Bayesian computation (ABC) and neural density estimation (NDE) to identify the conductance parameters in the Hodgkin-Huxley model and the synaptic weight parameters in the Brunel network model.

We discussed an implementation of the rejection ABC algorithm with quantile-based tolerance and post-sampling local linear regression adjustment. Even though rejection ABC just sample proposal parameters from the prior densities, we were, in general, successful in identifying the model parameters of interest. By applying local linear regression adjustment, we obtained narrow posterior densities with the ground truth values of the parameters in regions of high posterior density. The posterior error decreases in the limit of small tolerances. With regression adjustment, we found that we could accept more simulations without sacrificing substantial accuracy for the model parameters that were the most constrained by the summary statistics. The choice of summary statistics is crucial for the performance of ABC. In practice, the success of ABC relies on expert-crafted low-dimensional summary statistics that constrain the model parameters of interest. Although the rejection ABC approach were able to perform efficiently, in particular on the Hodgkin-Huxley model, and recover the ground truth parameters, more advanced sampling algorithms with better proposal mechanisms should be considered. The accurate results we were able to obtain with the simple rejection sampler implies that even better results can be expected by using more advanced samplers.

We compared the rejection ABC algorithm with the NDE machine learning tool Sequential Neural Posterior Estimation (SNPE), which trains an artificial neural network to map features of observed data to posteriors over parameters by using adaptively proposed model simulations. Inference on the Brunel network model demonstrates the power and flexibility of SNPE; by training on simulations that included the parameter ranges for two of the network states, SNPE was able to accurately predict posteriors that corresponded to the network's state in the observed data. The advantage of the SNPE approach is that once trained, the network can be applied to any observed data and estimate the posterior densities over model parameters with only a single pass through the network.

A limiting factor of the simulation-based approach to inference is that the algorithms are simulation intensive. Consequently, the efficiency of an algorithm predominantly depends on the computational demands of the simulator. There are also challenges (and opportunities) ahead in scaling and automating simulation-based inference approaches. However, the frontier of simulation-based inference as a methodological research field is rapidly advancing. This activity is promising for the numerous inverse modelling problems in neuroscience. The generality of the simulation-based inference methods allows them to be applied to a wide range of computational investigations in need of improved inference quality. In addition to being able to aid in designing better models of neural dynamics, simulation-based inference may ultimately help to bridge the gap between mechanistic hypotheses and experimental neural data.

12

Future Research

We here provide an outline of potential future research building off of the present work.

Simulation-based inference opens up many interesting avenues for inverse problems with complex simulators. In this study, we have only investigated low-dimensional problems, i.e., inferential tasks with few parameters to identify. However, most models typically have several unknown parameters. The inferential task will likely become more difficult as the number of parameters to identify increases. Building off of the present results on the HH model, a more complex model to perform a similar procedure on is the multi-compartment model of a thalamic interneuron proposed by Halnes et al. in [58]. A sensitivity analysis of this particular model, using the same set of summary statistics as for the HH model in this study, was carried out by Tennøe et al. in [40] and could be used to help constrain the model parameters.

Skaar et al. in [59] estimated the synaptic weight parameters of the Brunel network model from local field potentials (LFPs) using a convolutional neural network (CNN). LFPs are unavailable for point neuron networks, such as the Brunel network, but a hybrid scheme developed by Hagen et al. [60] allows LFPs to be modeled from point neuron networks and was used by Skaar et al. LFP has become a popular measure of neuronal activity, and an interesting study could be to use the simulation-based inference algorithms on the Brunel network model with summary statistics from the LFP.

Until recently, simulation-based (or likelihood-free) inference was synonymous with ABC. The advent of machine learning methods, in particular deep learning, has introduced powerful algorithms for density estimation. Though SNPE and ABC

have already been compared by Lueckmann et al. in [61], there are many comparison studies that can be done. ABC is a principled approach for Bayesian parameter identification, and should not be ruled out as of yet.

Deep learning has also introduced powerful algorithms for learning features from data. The ABC algorithms rely on expert-crafted summary statistics to obtain accurate posterior estimates. Therefore, automated approaches using e.g. deep learning to extract summary statistics with approximate sufficiency would be attractive. Thus, another interesting study could be to compare the results from this study with expert-crafted summary statistics to ones obtained with features learned from the raw neural data generated by the models.

Since simulation-based inference as a methodological research field is rapidly advancing, most case studies use synthetic data. The ultimate goal of simulation-based inference is to enable researchers to identify the model parameters consistent with experimental data. Interesting studies would therefore be ones trying to make contact between real-world experimental data and mechanistic models of neural dynamics.

Appendices

A

Additional Results

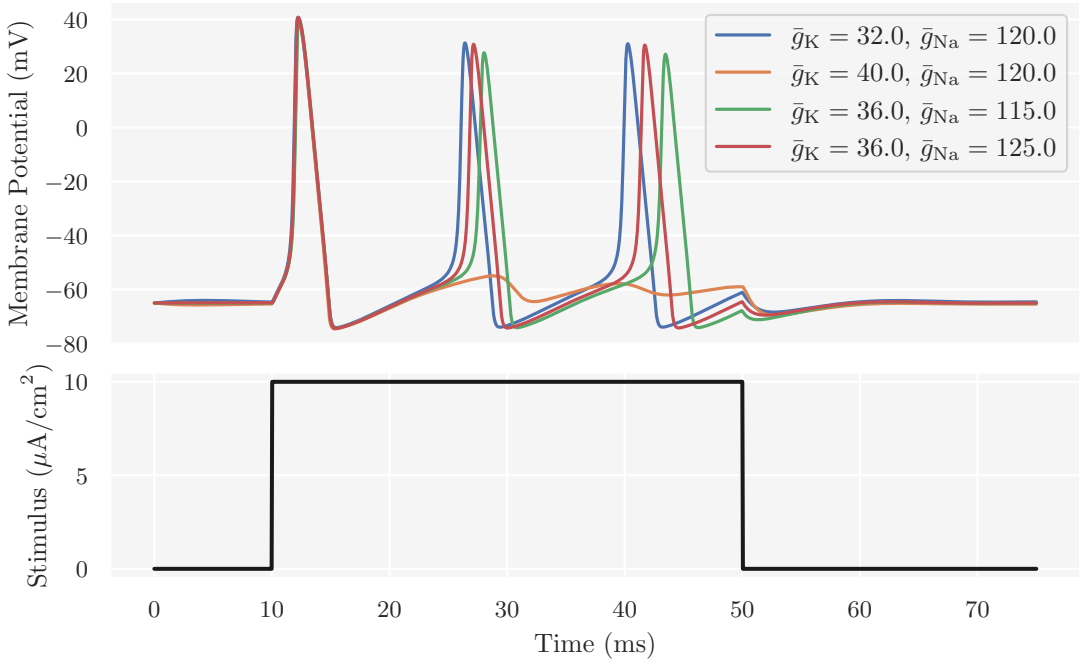


Figure A.1: Simulation of action potentials with the Hodgkin-Huxley simulator for different values of $\bar{g}_K$ and $\bar{g}_{Na}$ (stated in the legend). The rest of the parametrization is given by Table 5.1.

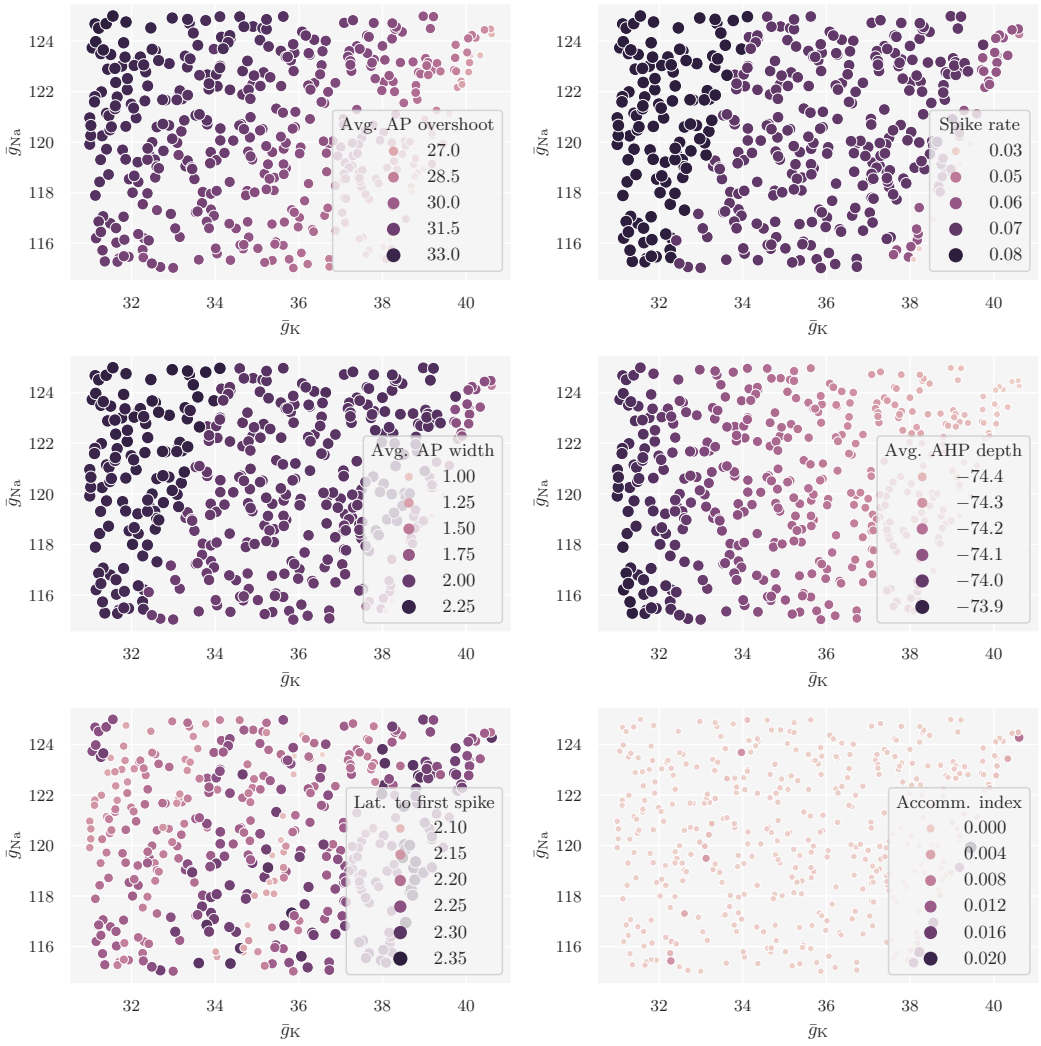
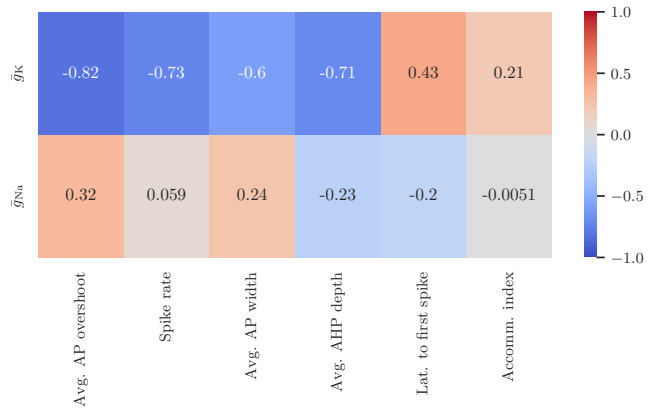
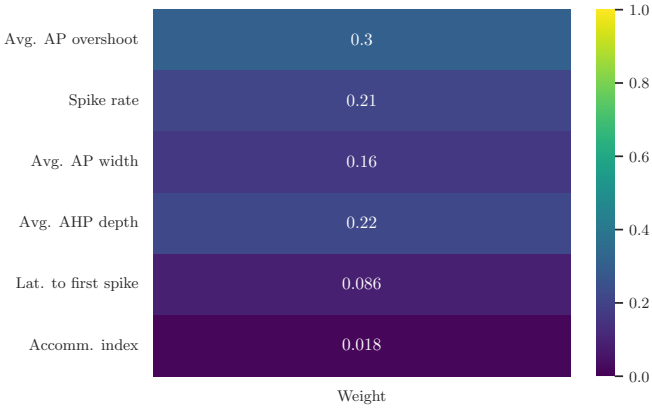


Figure A.2: Similar to Figure 8.4, only with summary statistics simulated under the joint noninformative prior distribution. Of the 2000 samples generated, 1703 were well-defined. Here a subset of 425 samples is shown.



(a)



(b)

Figure A.3: Similar to Figure 8.5, but with samples from the joint noninformative prior distribution. **(a)** The pairwise Pearson's correlation coefficients. **(b)** Importance weights calculated from the correlation coefficients (summed to 1).

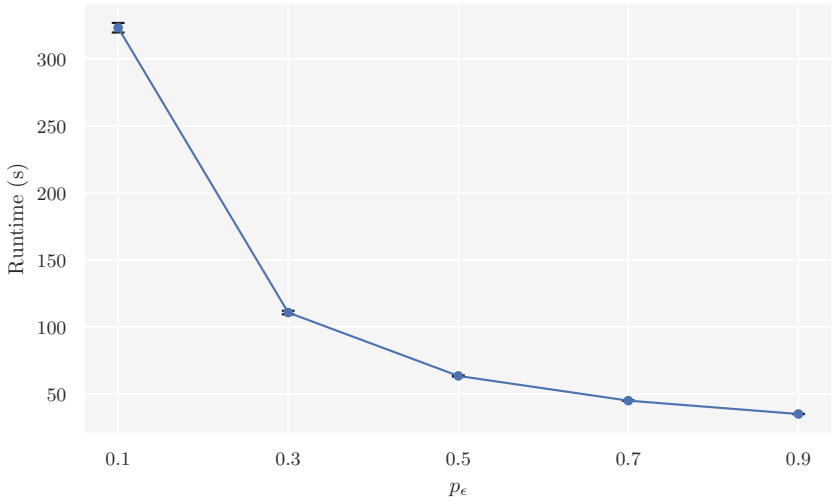


Figure A.4: Illustration of computational run time for obtaining 1000 posterior samples with the HH simulator vs. tolerance quantile, i.e., the proportion of simulations that are accepted.

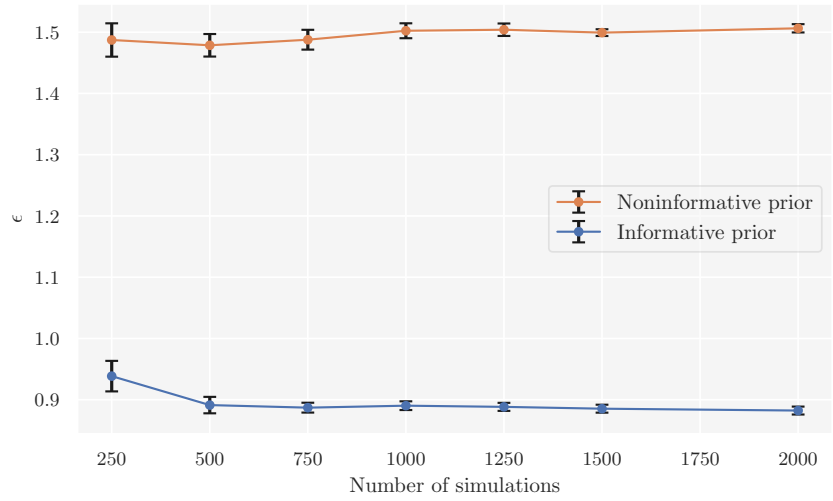


Figure A.5: Estimated tolerance ϵ vs. the number of simulations in the pilot study. Here, both summary statistic scales and weights are set to 1 when calculating the distance, so the estimates might not be representative for all use cases.

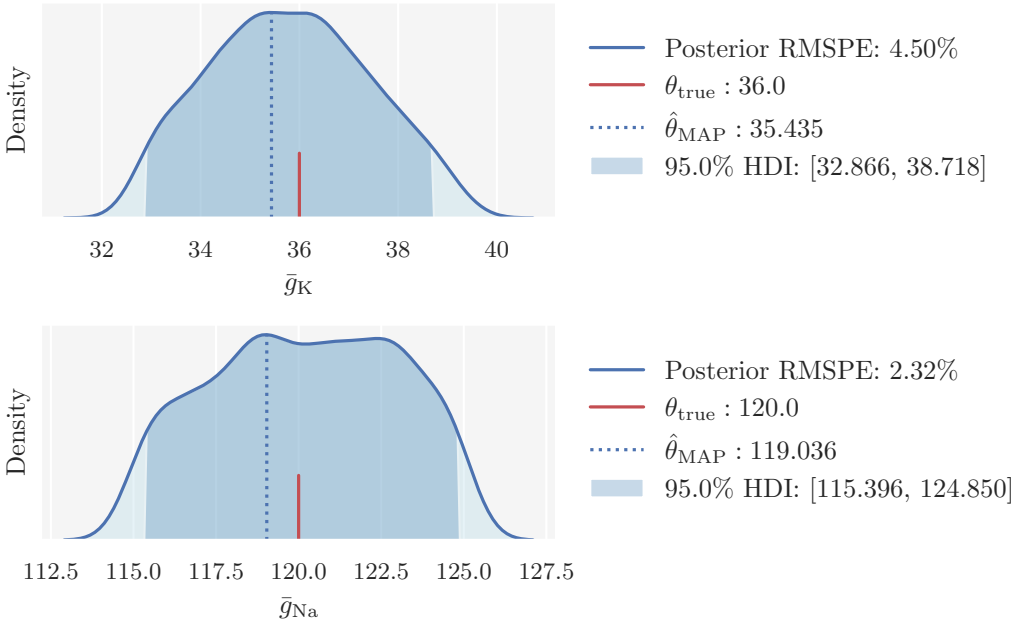


Figure A.6: Original rejection ABC posteriors over the Hodgkin-Huxley model parameters $\bar{g}_{\mathrm{K}}$ (top) and $\bar{g}_{\mathrm{Na}}$ (bottom) with noise-free observed voltage trace. Here, the parameter proposals were sampled from the joint noninformative prior distribution. The dark shaded region indicates the 95% HDI and the dotted line the MAP estimate. The ground truth is indicated by the red line. The legend states the numerical values for each of these, in addition to the RMSPE in the posterior.

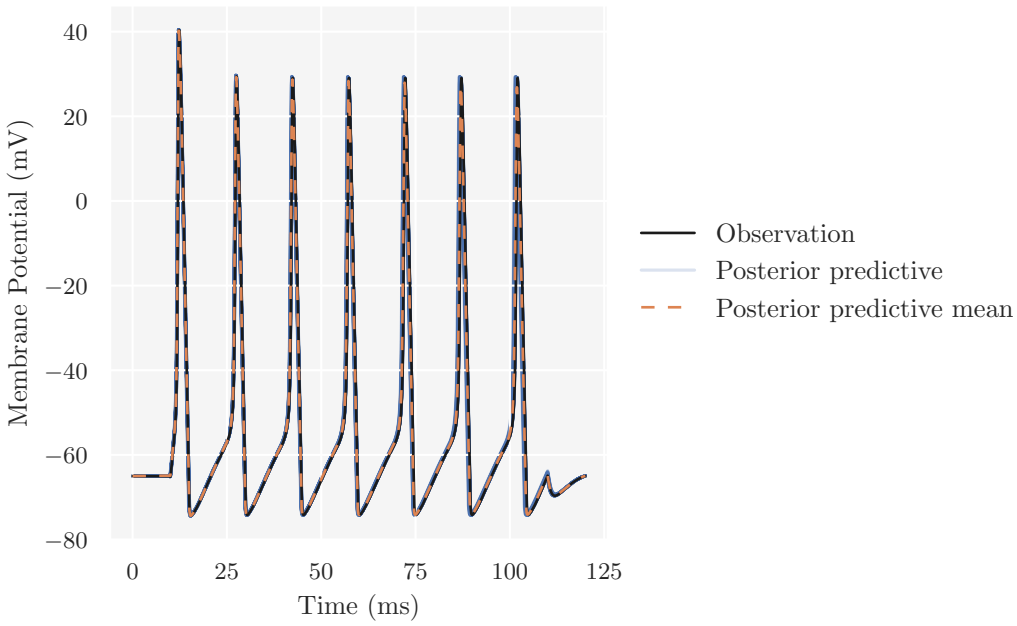


Figure A.7: Graphical posterior predictive check comparing the observed voltage trace to simulated data predicted by the Hodgkin-Huxley model under the regression adjusted joint posterior predictive distribution that is shown as marginal posteriors in Figure 8.12.

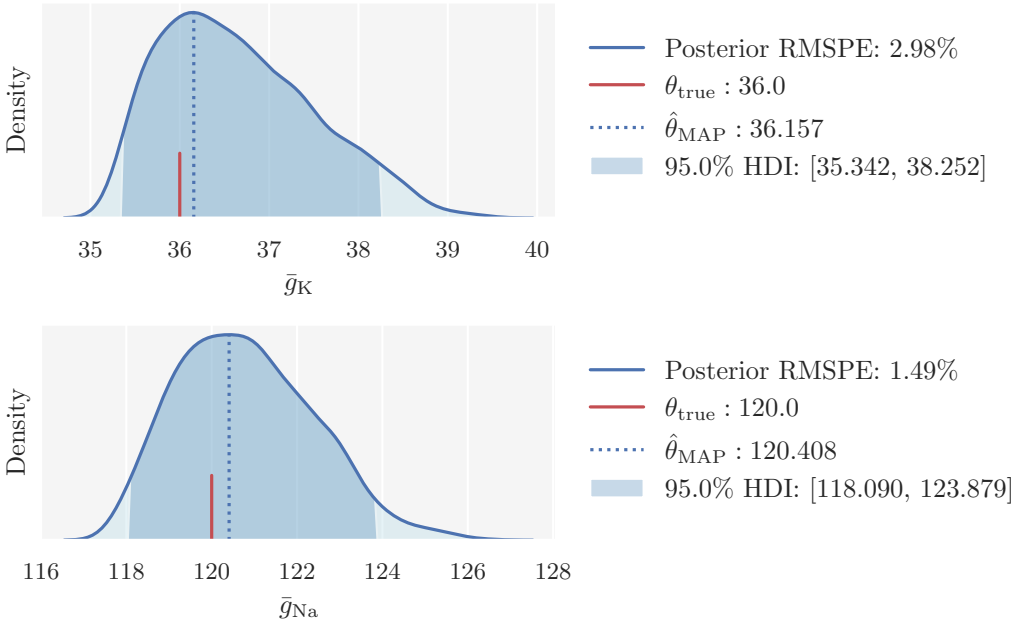


Figure A.8: Original rejection ABC posteriors over the Hodgkin-Huxley model parameters $\bar{g}_{\mathrm{K}}$ (top) and $\bar{g}_{\mathrm{Na}}$ (bottom) with noisy observed voltage trace. Here, the parameter proposals were sampled from the joint informative prior distribution. The dark shaded region indicates the 95% HDI and the dotted line the MAP estimate. The ground truth is indicated by the red line. The legend states the numerical values for each of these, in addition to the RMSPE in the posterior.

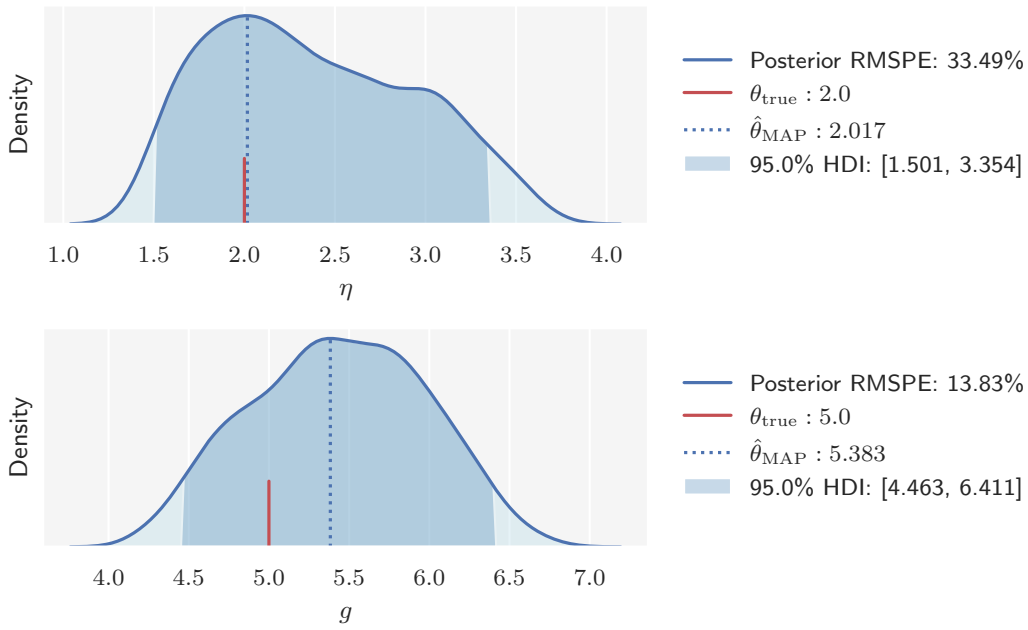


Figure A.9: Original rejection ABC posteriors over the Brunel network model parameters η (top) and g (bottom) with observed data from the AI state. The dark shaded region indicates the 95% HDI and the dotted line the MAP estimate. The ground truth is indicated by the red line. The legend states the numerical values for each of these, in addition to the RMSPE in the posterior.

B

Derivations

B.1 Alternative Hodgkin-Huxley Formulation

We here provide an example on how the alternative formulation of the original Hodgkin-Huxley model with reversed polarity of the membrane potential and shifted resting potential to $-65\,\mathrm{mV}$ can be derived. In its original formulation, the potassium channel rate coefficient is on the form:

$$\alpha_n = 0.01 \cdot \frac{V + 10}{\exp\left(\frac{V+10}{10}\right) - 1}.$$

If we let $A = 0.01$, $x = V + 10$, and $y = 10$, such that

$$\alpha_n = A \cdot \frac{x}{\exp\left(\frac{x}{y}\right) - 1},$$

we easily find that an alternative form is:

$$\alpha_n = A \cdot \frac{x}{\exp\left(\frac{x}{y}\right) - 1} \cdot \frac{(-1)}{(-1)} = A \cdot \frac{-x}{1 - \exp\left(\frac{x}{y}\right)}$$

Reversing the polarity and shifting the resting membrane changes $x = (V + 10)$ into $x = -(V + 55)$. Inserted into the last equation, we obtain the alternative formulation:

$$\alpha_n = 0.01 \cdot \frac{V + 55}{1 - \exp\left(-\frac{V+55}{10}\right)},$$

B.2 Derivation of vtrap

Here, we provide a derivation of vtrap (Equation 7.1). The point of departure is

$$\text{rate} = \frac{x}{\exp(x/y) - 1}.$$

The Taylor series expansion of $\exp(x/y)$ is

$$\exp(x/y) = \sum_{n=0}^{\infty} \frac{(x/y)^n}{n!},$$

and truncation at the second-order approximation gives

$$\exp(x/y) \approx 1 + \frac{x}{y} + \frac{(x/y)^2}{2} + ...$$

Thus,

$$\text{rate} \approx \frac{x}{1 + \frac{x}{y} + \frac{(x/y)^2}{2} - 1} = \frac{1}{\frac{1}{y}\left(1 + \frac{x}{2y}\right)} = \frac{y}{1 + \frac{x}{2y}}.$$

Next, we manipulate the expression further:

$$\text{rate} \approx \frac{y}{1 + \frac{x}{2y}} \cdot \frac{1 - \frac{x}{2y}}{1 - \frac{x}{2y}} = \frac{y\left(1 - \frac{x}{2y}\right)}{1 - \left(\frac{x}{2y}\right)^2}.$$

If $x/y << 1$, then the term $(x/2y)^2 \to 0$ much faster than $(x/2y)$. Thus, we can make the approximation:

$$\text{rate} \approx y\left(1 - \frac{x}{2y}\right)$$

for $x/y << 1$.

References

[1] Society for Neuroscience. *Brain Facts: A Primer on the Brain and Nervous System*. 8th ed. Washington, DC: Society for Neuroscience, 2018 (cit. on pp. 1, 39).

[2] David Sterratt et al. *Principles of Computational Modelling in Neuroscience*. 1st ed. Cambridge, UK: Cambridge University Press, 2011 (cit. on pp. 1, 39, 41, 44, 46, 51).

[3] S Tavare et al. "Inferring Coalescence Times From DNA Sequence Data." In: *Genetics* 145.2 (1997), pp. 505–518 (cit. on pp. 2, 28).

[4] Jonathan K Pritchard et al. "Population growth of human Y chromosomes: A study of y chromosome microsatellites." In: *Molecular biology and evolution* 16.12 (1999), pp. 1791–1798 (cit. on pp. 2, 28).

[5] Mark A Beaumont, Wenyang Zhang, and David J Balding. "Approximate Bayesian Computation in Population Genetics." In: *Genetics* 162.4 (2002), pp. 2025–2035 (cit. on pp. 2, 31, 32).

[6] George Papamakarios and Iain Murray. *Fast ϵ-free Inference of Simulation Models with Bayesian Conditional Density Estimation*. 2016. arXiv: 1605.06376 [stat.ML] (cit. on pp. 3, 36, 96).

[7] Jan-Matthis Lueckmann et al. *Flexible statistical inference for mechanistic models of neural dynamics*. 2017. arXiv: 1711.01861 [stat.ML] (cit. on pp. 3, 36, 96).

[8] David S. Greenberg, Marcel Nonnenmacher, and Jakob H. Macke. *Automatic Posterior Transformation for Likelihood-Free Inference*. 2019. arXiv: 1905.07488 [cs.LG] (cit. on pp. 3, 36, 96).

[9] A. L Hodgkin and A. F Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve." In: *The Journal of physiology* 117.4 (1952), pp. 500–544 (cit. on pp. 4, 44, 116).

[10] Nicolas Brunel. "Dynamics of Sparsely Connected Networks of Excitatory and Inhibitory Spiking Neurons." In: *Journal of Computational Neuroscience* 8.3 (2000), pp. 183–208 (cit. on pp. 4, 44, 50, 53, 54, 116).

[11] Alvaro Tejero-Cantero et al. "sbi: A toolkit for simulation-based inference." In: *Journal of Open Source Software* 5.52 (2020), p. 2505 (cit. on pp. 5, 69).

[12] Scott A. Sisson, Yanan Fan, and Mark A. Beaumont. "Overview of ABC." In: *Handbook of Approximate Bayesian Computation*. Ed. by Scott A. Sisson, Yanan Fan, and Mark A. Beaumont. 1st ed. Handbooks of Modern Statistical Methods. Boca Raton, Fl: CRC Press, 2019. Chap. 1, pp. 3–54 (cit. on p. 6).

[13] Andrew Gelman et al. *Bayesian Data Analysis*. 3rd ed. Texts in statistical science. Boca Raton, Fl: CRC Press, 2014 (cit. on pp. 6, 9, 18).

[14] Osvaldo Martin. *Bayesian Analysis with Python*. 2nd ed. Texts in statistical science. Birmingham, UK: Packt Publishing, 2018 (cit. on p. 9).

[15] D. S. Sivia and J. Skilling. *Data Analysis - A Bayesian Tutorial*. 2nd ed. Oxford Science Publications. New York, NY: Oxford University Press, 2006 (cit. on p. 9).

[16] Nicholas Metropolis et al. "Equation of State Calculations by Fast Computing Machines." In: *The Journal of chemical physics* 21.6 (1953), pp. 1087–1092 (cit. on p. 17).

[17] W. K Hastings. "Monte Carlo sampling methods using Markov chains and their applications." In: *Biometrika* 57.1 (1970), pp. 97–109 (cit. on p. 17).

[18] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Springer Series in Statistics. New York, NY: Springer New York, 2009 (cit. on p. 19).

[19] Christopher M Bishop. *Pattern recognition and machine learning*. 1st ed. Information science and statistics. New York, NY: Springer New York, 2006 (cit. on pp. 19, 36).

[20] Ilya Narsky and Frank C Porter. *Statistical Analysis Techniques in Particle Physics: Fits, Density Estimation and Supervised Learning*. 1st ed. Weinheim: John Wiley & Sons, Incorporated, 2013 (cit. on p. 19).

[21] David W Scott. "On optimal and data-based histograms." In: *Biometrika* 66.3 (1979), pp. 605–610 (cit. on p. 20).

[22] David Freedman and Persi Diaconis. "On the histogram as a density estimator: L2 theory." In: *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* 57.4 (1981), pp. 453–476 (cit. on p. 20).

[23] B.W Silverman. *Density estimation for statistics and data analysis*. 1st ed. Monographs on statistics and applied probability. London, UK: Chapman and Hall, 1986 (cit. on p. 21).

[24] Drleft. *Comparison of 1D histogram and KDE*. 2010. URL: https://commons.wikimedia.org/wiki/File:Comparison_of_1D_histogram_and_KDE.png (visited on 10/30/2021) (cit. on p. 21).

[25] Scott A. Sisson, Yanan Fan, and Mark A. Beaumont, eds. *Handbook of Approximate Bayesian Computation*. 1st ed. Handbooks of Modern Statistical Methods. Boca Raton, Fl: Chapman and Hall/CRC, 2018 (cit. on pp. 26, 27, 32, 119).

[26] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. "The frontier of simulation-based inference." In: *Proceedings of the National Academy of Sciences* 117.48 (2020), pp. 30055–30062 (cit. on pp. 26, 117).

[27] George Papamakarios. *Neural Density Estimation and Likelihood-free Inference*. 2019. arXiv: 1910.13233 [stat.ML] (cit. on p. 26).

[28] D. B Rubin. "Bayesianly justifiable and relevant frequency calculations for the applied statistician." In: *The Annals of statistics* 12.4 (1984), pp. 1151–1172 (cit. on p. 28).

[29] Lorenzo Pacchiardi et al. "Distance-learning For Approximate Bayesian Computation To Model a Volcanic Eruption." In: *Sankhya B* 83.1 (2021), pp. 288–317 (cit. on p. 30).

[30] James J Palestro et al. *Likelihood-Free Methods for Cognitive Science*. 1st ed. Cham: Springer International Publishing, 2018 (cit. on p. 32).

[31] Jarno Lintusaari et al. "Fundamentals and Recent Developments in Approximate Bayesian Computation." In: *Systematic biology* 66.1 (2017), e66–e82 (cit. on p. 34).

[32] Laurent Dinh, David Krueger, and Yoshua Bengio. *NICE: Non-linear Independent Components Estimation*. 2015. arXiv: 1410.8516 [cs.LG] (cit. on p. 36).

[33] Pedro J Goncalves et al. "Training deep neural density estimators to identify mechanistic models of neural dynamics." In: *eLife* 9 (2020), e56261 (cit. on pp. 38, 117).

[34] Peter Dayan and L.F. Abbott. *Theoretical neuroscience : computational and mathematical modeling of neural systems*. 1st ed. Cambridge, Mass: MIT Press, 2001 (cit. on pp. 39, 44).

[35] Molecular Devices. *What is an action potential?* URL: https://cs231n.github.io/neural-networks-1/ (visited on 10/30/2021) (cit. on p. 40).

[36] Standford University. *CS231n: Convolutional Neural Networks for Visual Recognition*. URL: https://www.moleculardevices.com/applications/patch-clamp-electrophysiology/what-action-potential (visited on 10/30/2021) (cit. on p. 42).

[37] Wulfram Gerstner et al. *Neuronal dynamics : from single neurons to networks and models of cognition*. 1st ed. Cambridge: Cambridge University Press, 2014 (cit. on p. 44).

[38] Marc-Oliver Gewaltig, Abigail Morrison, and Hans Ekkehard Plesser. "NEST by Example: An Introduction to the Neural Simulation Tool NEST." In: *Computational Systems Neurobiology*. Dordrecht: Springer Netherlands, 2012, pp. 533–558 (cit. on p. 52).

[39] Shaul Druckmann et al. "A novel multiple objective optimization framework for constraining conductance-based neuron models by experimental data." In: *Frontiers in Neuroscience* 1.1 (2007), pp. 7–18 (cit. on pp. 59, 61, 118).

[40] Simen Tennøe, Geir Halnes, and Gaute T Einevoll. "Uncertainpy: A Python Toolbox for Uncertainty Quantification and Sensitivity Analysis in Computational Neuroscience." In: *Frontiers in Neuroinformatics* 12 (2018), p. 49 (cit. on pp. 61, 118, 122).

[41] Michael G.B. Blum. "Regression Approaches for ABC." In: *Handbook of Approximate Bayesian Computation*. Ed. by Scott A. Sisson, Yanan Fan, and Mark A. Beaumont. 1st ed. Handbooks of Modern Statistical Methods. Boca Raton, Fl: CRC Press, 2019. Chap. 3, pp. 71–85 (cit. on p. 66).

[42] Nicholas T Carnevale and Michael L Hines. *The NEURON Book*. Cambridge: Cambridge University Press, 2006 (cit. on p. 67).

[43] Charles R. Harris et al. "Array programming with NumPy." In: *Nature* 585.7825 (2020), pp. 357–362 (cit. on p. 69).

[44] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." In: *Nature Methods* 17 (2020), pp. 261–272 (cit. on p. 69).

[45] J. D. Hunter. "Matplotlib: A 2D graphics environment." In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95 (cit. on p. 69).

[46] Wes McKinney et al. "Data structures for statistical computing in python." In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. Vol. 445. 2010, pp. 56–61 (cit. on p. 69).

[47] Michael L. Waskom. "seaborn: statistical data visualization." In: *Journal of Open Source Software* 6.60 (2021), p. 3021 (cit. on p. 69).

[48] Michael M. McKerns et al. *Building a Framework for Predictive Science*. 2012. arXiv: 1202.1056 [cs.MS] (cit. on p. 69).

[49] Tanguy Fardet et al. *NEST 2.20.2*. Version 2.20.2. Aug. 2021. DOI: 10.5281/zenodo.5242954 (cit. on p. 69).

[50] Garcia S. et al. "Neo: an object model for handling electrophysiology data in multiple formats." In: *Frontiers in Neuroinformatics* 8:10 (2014) (cit. on p. 69).

[51] M. Denker, A. Yegenoglu, and S. Grün. "Collaborative HPC-enabled workflows on the HBP Collaboratory using the Elephant framework." In: *Neuroinformatics 2018*. 2018, P19. DOI: 10.12751/incf.ni2018.0019 (cit. on p. 69).

[52] J.R Dormand and P.J Prince. "A family of embedded Runge-Kutta formulae." In: *Journal of computational and applied mathematics* 6.1 (1980), pp. 19–26 (cit. on p. 69).

[53] George Papamakarios, Theo Pavlakou, and Iain Murray. *Masked Autoregressive Flow for Density Estimation*. 2018. arXiv: 1705.07057 [stat.ML] (cit. on p. 96).

[54] Joshua H Goldwyn and Eric Shea-Brown. "The what and where of adding channel noise to the Hodgkin-Huxley equations." In: *PLoS computational biology* 7.11 (2011), e1002247–e1002247 (cit. on p. 98).

[55] Aidan C Daly et al. "Hodgkin-Huxley revisited: reparametrization and identifiability analysis of the classic action potential model with approximate Bayesian methods." In: *Royal Society open science* 2.12 (2015), pp. 150499–150499 (cit. on p. 116).

[56] Mikael Sunnåker et al. "Approximate Bayesian Computation." In: *PLOS Computational Biology* 9.1 (2013), pp. 1–10 (cit. on p. 117).

[57] Dennis Prangle. "Adapting the ABC Distance Function." In: *Bayesian Analysis* 12.1 (2017), pp. 289–309 (cit. on p. 118).

[58] Geir Halnes et al. "A Multi-Compartment Model for Interneurons in the Dorsal Lateral Geniculate Nucleus." In: *PLOS Computational Biology* 7.9 (2011), pp. 1–12 (cit. on p. 122).

[59] Jan-Eirik W. Skaar et al. "Estimation of neural network model parameters from local field potentials (LFPs)." In: *PLOS Computational Biology* 16.3 (2020), pp. 1–30 (cit. on p. 122).

[60] Espen Hagen et al. "Hybrid Scheme for Modeling Local Field Potentials from Point-Neuron Networks." In: *Cerebral cortex (New York, N.Y. 1991)* 26.12 (2016), pp. 4461–4496 (cit. on p. 122).

[61] Jan-Matthis Lueckmann et al. "Benchmarking Simulation-Based Inference." In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by Arindam Banerjee and Kenji Fukumizu. Vol. 130. Proceedings of Machine Learning Research. 2021, pp. 343–351 (cit. on p. 123).