LUND UNIVERSITY

Simulation-based Inference

From Approximate Bayesian Computation and Particle Methods to Neural Density Estimation

Wiqvist, Samuel

2021

Document Version:
Publisher's PDF, also known as Version of record

– CENTRUM SCIENTIARUM MATHEMATICARUM –

Simulation-based Inference

From Approximate Bayesian Computation and Particle Methods to Neural Density Estimation

SAMUEL WIQVIST

Lund University
Faculty of Science
Centre for Mathematical Sciences
Mathematical Statistics

Simulation-based Inference

From Approximate Bayesian Computation and Particle Methods to Neural Density Estimation

Simulation-based Inference
From Approximate Bayesian Computation and Particle Methods to Neural Density Estimation

by Samuel Wiqvist

LUND
UNIVERSITY

Thesis for the degree of Doctor of Philosophy

to be presented, with the permission of the Faculty of Science of Lund University, for public criticism at the Centre of Mathematical Sciences, Lund University, room MH:R, on Friday, the 24th of September 2021 at 1:00 pm.

Faculty opponent: Prof. Darren J. Wilkinson.
School of Mathematics, Statistics and Physics,
Newcastle University, UK.

Organization
LUND UNIVERSITY

Centre of Mathematical Sciences
Box 124
SE–221 00 LUND
Sweden

Document name
DOCTORAL THESIS

Date of disputation
2021-09-24

Sponsoring organization

Author(s)
Samuel Wiqvist

Title and subtitle
Simulation-based Inference– From Approximate Bayesian Computation and Particle Methods to Neural Density Estimation

Abstract
This doctoral thesis in computational statistics utilizes both Monte Carlo methods (approximate Bayesian computation and sequential Monte Carlo) and machine-learning methods (deep learning and normalizing flows) to develop novel algorithms for inference in implicit Bayesian models. Implicit models are those for which calculating the likelihood function is very challenging (and often impossible), but model simulation is feasible. The inference methods developed in the thesis are *simulation-based* inference methods since they leverage the possibility to simulate data from the implicit models. Several approaches are considered in the thesis: Paper II and IV focus on classical methods (sequential Monte Carlo-based methods), while paper I and III focus on more recent machine learning methods (deep learning and normalizing flows, respectively).

Paper I constructs novel deep learning methods for learning summary statistics for approximate Bayesian computation (ABC). To achieve this paper I introduces the *partially exchangeable network* (PEN), a deep learning architecture specifically designed for Markovian data (i.e., partially exchangeable data).

Paper II considers Bayesian inference in stochastic differential equation mixed-effects models (SDEMEM). Bayesian inference for SDEMEMs is challenging due to the intractable likelihood function of SDEMEMs. Paper II addresses this problem by designing a novel a Gibbs-blocking strategy in combination with correlated pseudo-marginal methods. The paper also discusses how custom particle filters can be adapted to the inference procedure.

Paper III introduces the novel inference method *sequential neural posterior and likelihood approximation* (SNPLA). SNPLA is a simulation-based inference algorithm that utilizes normalizing flows for learning both the posterior distribution and the likelihood function of an implicit model via a sequential scheme. By learning both the likelihood and the posterior, and by leveraging the *reverse* Kullback Leibler (KL) divergence, SNPLA avoids *ad-hoc* correction steps and Markov chain Monte Carlo (MCMC) sampling.

Paper IV introduces the *accelerated-delayed acceptance* (ADA) algorithm. ADA can be viewed as an extension of the *delayed-acceptance* (DA) MCMC algorithm that leverages connections between the two likelihood ratios of DA to further accelerate MCMC sampling from the posterior distribution of interest, although our approach introduces an approximation. The main case study of paper IV is a double-well potential stochastic differential equation (DWP-SDE) model for protein-folding data (reaction coordinate data).

Key words
Bayesian statistics, computational statistics, deep learning, mixed-effects, sequential Monte Carlo, stochastic differential equations

Classification system and/or index terms (if any)

Supplementary bibliographical information

Language
English

ISSN and key title

ISBN
9789178959679 (print)
9789178959686 (pdf)

Recipient's notes

Number of pages
218

Price

Security classification

Simulation-based Inference
From Approximate Bayesian Computation and Particle Methods to Neural Density Estimation

by Samuel Wiqvist

LUND
UNIVERSITY

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118
SE-221 00 Lund
Sweden

Doctoral Theses in Mathematical Sciences 2021:09

ISSN: 1404-0034
ISRN: LUNFMS-1029-2021
ISBN: 9789178959679 (print)
ISBN: 9789178959686 (pdf)

Printed in Sweden by Media-Tryck, Lund University, Lund 2021

Contents

Abstract

This doctoral thesis in computational statistics utilizes both Monte Carlo methods (approximate Bayesian computation and sequential Monte Carlo) and machine-learning methods (deep learning and normalizing flows) to develop novel algorithms for inference in implicit Bayesian models. Implicit models are those for which calculating the likelihood function is very challenging (and often impossible), but model simulation is feasible. The inference methods developed in the thesis are *simulation-based* inference methods since they leverage the possibility to simulate data from the implicit models. Several approaches are considered in the thesis: Paper II and IV focus on classical methods (sequential Monte Carlo-based methods), while paper I and III focus on more recent machine learning methods (deep learning and normalizing flows, respectively).

Paper I constructs novel deep learning methods for learning summary statistics for approximate Bayesian computation (ABC). To achieve this paper I introduces the *partially exchangeable network* (PEN), a deep learning architecture specifically designed for Markovian data (i.e., partially exchangeable data).

Paper II considers Bayesian inference in stochastic differential equation mixed-effects models (SDEMEM). Bayesian inference for SDEMEMs is challenging due to the intractable likelihood function of SDEMEMs. Paper II addresses this problem by designing a novel a Gibbs-blocking strategy in combination with correlated pseudo-marginal methods. The paper also discusses how custom particle filters can be adapted to the inference procedure.

Paper III introduces the novel inference method *sequential neural posterior and likelihood approximation* (SNPLA). SNPLA is a simulation-based inference algorithm that utilizes normalizing flows for learning both the posterior distribution and the likelihood function of an implicit model via a sequential scheme. By learning both the likelihood and the posterior, and by leveraging the *reverse* Kullback Leibler (KL)

divergence, SNPLA avoids *ad-hoc* correction steps and Markov chain Monte Carlo (MCMC) sampling.

Paper IV introduces the *accelerated-delayed acceptance* (ADA) algorithm. ADA can be viewed as an extension of the *delayed-acceptance* (DA) MCMC algorithm that leverages connections between the two likelihood ratios of DA to further accelerate MCMC sampling from the posterior distribution of interest, although our approach introduces an approximation. The main case study of paper IV is a double-well potential stochastic differential equation (DWP-SDE) model for protein-folding data (reaction coordinate data).

Popular science summary

Bayesian methods have become increasingly popular in the last 30 years, due to the development of advanced Monte Carlo based sampling-methods and associated powerful and easy-to-use computer programs. In the standard case, the likelihood function is analytically known. If that is not the case, the likelihood function may be unavailable in closed-form or be too computationally expensive to evaluate or even approximate. However, it is typically the case that we can run a computer model that generates artificial data. In this case, the likelihood is said to be known *implicitly* through the model simulations, and we then talk of *implicit models*. In that case, the model is defined by a computer program that generates data for given model parameters. By defining the model via a computer program, we can handle very complicated and flexible models since the only restriction is that we should be able to create a computer program that generates data for given model parameters. The Bayesian inference problem then consists of running the computer program for the implicit model many times to discover the parameter values that are most probable conditionally on the observed data. However, this inference problem can be very computationally challenging for complex implicit models. For this reason, the purpose of this thesis has been to develop new algorithms that solve the inference problem more efficiently. The thesis presents four papers that, in different ways, address inference problems of different implicit model classes and by using different approaches.

Paper II and IV further develop classical Bayesian inference methods. The algorithm presented in paper II is specially designed for mixed-effects stochastic dynamic models. These models are very relevant for pharmaceutical and biological applications. In contrast, paper IV presents a new general Monte Carlo algorithm for Bayesian inference that can be applied to many different types of implicit Bayesian models.

In paper I and III, machine learning methods are used instead. Paper I presents a novel deep learning architecture, and this architecture is used to summarize the informa-

tion in data efficiently. Paper III presents a new general algorithm for inference in implicit Bayesian models that leverages complicated data-generating machine learning methods.

Acknowledgements

Conducting research is a challenging task since you, by the nature of the research work, do not know if your ideas will be valuable until you put in the work to investigate them. However, the challenging research work has been fruitful and enjoyable since I have had the privilege to work with several friendly and intelligent people. First, I would like to thank my main supervisor Umberto Picchini for all the support and joint research efforts during the last five years. It has also been rewarding to collaborate with Jes Frellsen, Pierre-Alexandre Mattei, Andrew Golightly, Ashleigh T. McLean, Julie Lyng Forman, Kresten Lindorff-Larsen, Wouter Boomsma, and Sebastian Persson, as well as my other co-authors. I would finally like to thank my friends, family, and specifically my parents *Weine* and *Helena* for their support during my Ph.D. journey.

Funding information The thesis work was financially supported by the Swedish Research Council (grant: 2013-5167).

Lund, August 16th, 2021

Samuel Wiqvist

List of papers

The following papers are included in this thesis:

I **Partially exchangeable networks and architectures for learning summary statistics in approximate Bayesian computation**
Wiqvist, S., Mattei, P. A., Picchini, U., & Frellsen, J.
International Conference on Machine Learning, (pp. 6798-6807). PMLR.

II **Efficient inference for stochastic differential equation mixed-effects models using correlated particle pseudo-marginal algorithms**
Wiqvist, S., Golightly, A., McLean, A. T., & Picchini, U.
Computational Statistics & Data Analysis, 157, 107151. Elsevier.

III **Sequential neural posterior and likelihood approximation**

Wiqvist, S., Frellsen, J., & Picchini, U.
arXiv, arXiv:2102.06522.

IV **Accelerating delayed-acceptance Markov chain Monte Carlo algorithms**

Wiqvist, S., Picchini, U., Forman, J. L., Lindorff-Larsen, K., & Boomsma, W.
arXiv, arXiv:1806.05982

Additional papers not included in the thesis:

A **PEPSDI: Scalable and flexible inference framework for stochastic dynamic single-cell models**
Persson, S., Welkenhuysen, N., Shashkova, S., **Wiqvist, S.**, Reith, P., W Schmidt, G., Picchini, U., Cvijovic, M.
bioRxiv., bioRxiv:DOI:2021.07.01.450748

List of notation

Notation conventions used in the introductory chapter (however, each paper has its oven self-containing notation standards):

$\Gamma(a, \theta)$	Gamma distribution with shape parameter a and scale parameter θ.
$\mathcal{N}(\mu, \sigma)$	Univariate normal distribution with mean μ and standard deviation σ.
$\mathcal{N}(\mu, \Sigma)$	Multivariate normal distribution with mean vector μ and covariance matrix Σ.
$\mathcal{N}(x\|\mu, \Sigma)$	Multivariate normal distribution for x with mean vector μ and covariance matrix Σ.
$\mathcal{U}(a, b)$	Uniform distribution with lower limit a and upper limit b.
$\overrightarrow{0}$	Zero vector.
$\mathbb{I}$	Identity matrix.
$\mathbb{R}^n$	The real coordinate space of dimension n.
$\mathfrak{X}$	Countable space.
$2^{\mathfrak{X}}$	Power set of $\mathfrak{X}$.
$\mathcal{Y}$	(Possibly) uncountable space.

$\sim$	*Has probability distribution*, or *simulate from*.
$\in$	Belongs to.
$\propto$	Proportional to.
$a \wedge b$	Minimum between a and b.
$\mathbf{1}_x(y)$	Indicator function for the condition $x = y$.
^{T}	Transpose operator.
$P(E)$	Probability for the event E.
x^{obs}	Observed data set $x^{\text{obs}} \in \mathcal{X}$.
$p(\theta)$	Prior distribution of the parameter $\theta \in \mathbb{R}^{\dim \theta}$.
$p(x\|\theta)$	Likelihood function for generic $x \in \mathcal{X}$.
$p(x^{\text{obs}}\|\theta)$	Data likelihood.
$p(\theta\|x)$	Posterior distribution for generic $x \in \mathcal{X}$, i.e. the global posterior.
$p(\theta\|x^{\text{obs}})$	Posterior distribution for observed data $x_{\text{obs}} \in \mathcal{X}$.
x^i	The i:th data set.
x_j	The j:th observation of the data set x (or the j:th element of the vector x).
$a : b$	Sequence of natural numbers from a to b (i.e. `MATLAB` notation).
$x_{a:b}$ or $x^{a:b}$	Values $[x_a, x_{a+1}, \ldots, x_b]$ (or $[x^a, x^{a+1}, \ldots, x^b]$) of x (i.e. `Julia`-style vector slicing notation)

Introduction

This doctoral thesis in computational statistics utilizes both Monte Carlo methods (approximate Bayesian computation and sequential Monte Carlo) and machine-learning methods (deep learning and normalizing flows) to develop novel algorithms for inference in implicit Bayesian models. Implicit models are those for which calculating the likelihood function is very challenging (and often impossible), but model simulation is feasible. Thus, for implicit models, we have that the likelihood function is implicitly defined via model simulation. Therefore, implicit models are very flexible since their only requirement is that, give model parameters, we should be able to simulate data from the model. This also means that we can interpret the implicit model as some *random mechanism* that generates data given model parameters, and this random mechanism can be implemented as some *black-box* computer program.

The inference methods developed in the thesis are called *simulation-based* inference methods since they leverage the possibility to simulate data from the implicit models. Some important simulation-based inference methods discussed in this thesis are: *pseudo-marginal Metropolis-Hastings* (see Section 4.3), *approximate Bayesian computation* (see Section 4.2), *sequential neural posterior estimation* (see Section 4.6.1), and *sequential neural likelihood estimation* (see Section 4.6.2). Some additional methods, that are not discussed in this thesis are: *Bayesian synthetic likelihood* [Price et al., 2018], *Likelihood-free Inference by Ratio Estimation* [Thomas et al., 2020], *Bayesian Optimization for Likelihood-Free Inference* [Gutmann and Corander, 2016], *Amortized Approximate Likelihood Ratio MCMC* [Hermans et al., 2020], and *BayesFlow* [Radev et al., 2020]. Several of these methods are discussed in a recent overview of simulation-based inference [Cranmer et al., 2020] and several simulation-based inference methods are benchmarked in Lueckmann et al. [2021]. Applications of implicit models can, for instance, be found in particle physics, population genetics, epidemiology, and economics [Cranmer et al., 2020]. The field of simulation-based inference is currently developing quickly, and a trend is to leverage machine-learning methods

for inference in implicit models [Cranmer et al., 2020]. Thus, this thesis provides a timely contribution to the field of simulation-based inference. The idea of utilizing machine-learning methods is considered in two out of the four papers included in this thesis.

This introductory chapter has the following outline: Bayesian analysis is introduced in Section 1. Section 2 introduces the Bayesian implicit model, and applications are presented in Section 3. Simulation-based inference methods are discussed in Section 4. This introductory chapter ends with Section 5 where we give an overview of the papers, including my own contributions.

1 Bayesian analysis

Before introducing the Bayesian framework, we have to clarify the interpretation of probability used throughout the thesis. The most common interpretation of probability is the frequentist interpretation of probability, which states that the probability $P(E)$ for an event E is the limit of E's relative frequency over many trials [Venn, 1866]. However, the Bayesian analysis used in this thesis instead utilizes the Bayesian interpretation of the probability. The Bayesian interpretation of the probability states that $P(E)$ is our subjective quantified belief for an event E [De Finetti, 2017]. The Bayesian interpretation makes it possible to state probabilistic statements about non-repetitive events, and importantly for our purpose, it is the probability interpretation that underpins the Bayesian analysis [Gelman et al., 2013, chapter 1].

The Bayesian framework that utilizes the subjective interpretation of probability introduced above is sometimes called the subjective Bayesian analysis, and this approach is concisely explained by [Goldstein, 2006, page 1] as follows:

> You are uncertain about many things in the world. You can quantify your uncertainties as probabilities, for the quantities you are interested in, and conditional probabilities for observations you might make given the things you are interested in. When data arrives, Bayes theorem tells you how to move from your prior probabilities to new conditional probabilities for the quantities of interest.

Let us now formalize this approach: We assume that we have a probabilistic model $p(x|\theta)$ parameterized with the parameter $\theta \in \mathbb{R}^{\dim \theta}$. We also assume that we have

collected some data $x \in \mathcal{X}$. The task that we will consider is parametric inference for the model $p(x|\theta)$. The model, or the likelihood,

$$x \sim p(x|\theta) \tag{1}$$

which for now is some analytical distribution, is such that data $x \in \mathcal{X}$ is modeled conditional on the parameter θ via the conditional distribution $p(x|\theta)$. However, in our setting the model $p(x|\theta)$ is not what we are uncertain about. Instead, we are uncertain about the parameter of the model. i.e. θ. Thus, we will incorporate our *a-priori* beliefs about θ in the prior distribution $p(\theta)$. Now, we have that the prior $p(\theta)$ is some probability distribution that incorporates our subjective probabilistic beliefs regarding the parameter θ that we have before considering the data x. However, since we use the Bayesian framework, someone else can, of course, choose to use some other probability distribution for the prior $p(\theta)$ than the one we have specified, and thereby *a-priori* incorporating different probabilistic beliefs about the parameter θ. Once accessing the data x, we can update our beliefs about the parameter θ by obtaining the posterior distribution $p(\theta|x)$. The posterior distribution is the distribution of the parameter of interest θ conditioned on the data x. Hence, when obtaining the posterior distribution we compute the *inverse probability* for θ *given* x instead of x *given* θ. The posterior distribution is computed via Bayes' rule

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{\int p(x|\theta)p(\theta)d\theta}. \tag{2}$$

In Equation (2), $p(\theta|x)$ is the posterior distribution and $p(x) = \int p(x|\theta)p(\theta)d\theta$ is the marginal likelihood (also sometimes denoted *model evidence*). The marginal likelihood acts as the normalizing constant for the posterior. However, most inference algorithms, e.g Markov chain Monte Carlo (MCMC), only require that we know the posterior up to a proportionality constant independent of θ [Dunson and Johndrow, 2020], and we will therefore not further consider the marginal constant. We instead notice that the posterior, up to proportionality, is given by

$$p(\theta|x) \propto p(x|\theta)p(\theta). \tag{3}$$

The posterior can be obtained either via analytical calculations (if we have a conjugate model), sampling (i.e. MCMC [Brooks et al., 2011]), or direct density estimation (e.g. integrated nested Laplace approximations [Rue et al., 2017] and variational inference [Blei et al., 2017]). The posterior distribution contains information about which parameters are more probable to have generated the observed data (for instance, via the posterior mean or mode). Uncertainty quantifications of θ in the from of interval estimations can also be obtained via quantile intervals, credible intervals, or high

posterior density intervals. Thus, we have that the posterior contains essentially all information regarding the parameter θ that we are uncertain about. However, in our analyses, we will also employ several useful analytical tools such as prior and posterior predictive simulations, which additionally utilize the sampling distribution. The prior predictive distribution is given by [Gelman et al., 2013, chapter 1]

$$p(x) = \int p(x, \theta) d\theta = \int p(x|\theta) p(\theta) d\theta. \tag{4}$$

We notice that the prior predictive distribution is the same component as the marginal likelihood (or evidence) of Equation (3). The prior predictive distribution is unconditional on the data and it can therefore, via prior predictive simulations, be used to investigate how informative the prior distribution is. An example of this type of analysis is provided in Example 1. We notice that generating a sample $x^\star$ from the prior predictive distribution is easily achieved by:

1. Sample $\theta^\star \sim p(\theta)$.
2. Generate $x^\star \sim p(x|\theta^\star)$.

The posterior predictive distribution is given by [Gelman et al., 2013, chapter 1]

$$\begin{aligned} p(x^\star|x) &= \int p(x^\star, \theta|x) d\theta, & (5) \\ &= \int p(x^\star|\theta, x) p(\theta|x) d\theta, & (6) \\ &= \int p(x^\star|\theta) p(\theta|x) d\theta, & (7) \end{aligned}$$

In the posterior predictive distribution x is the observed data set and $x^\star$ is the predicted data set obtained conditionally on x. The step between Equation (6) and Equation (7) is due to the fact that that x and $x^\star$ are conditionally independent given θ. The posterior predictive distribution is important since it can be used for model evaluation purposes. A concrete example of this type of analysis is provided in Example 1. We again notice that generating a sample $x^\star$ from the posterior predictive distribution is easily achieved by:

1. Sample $\theta^\star \sim p(\theta|x)$.
2. Generate $x^\star \sim p(x|\theta^\star)$.

Example 1 (Modeling car stopping distances with Bayesian simple linear regression)

In the spirit of Gabry et al. [2019], here we will develop a Bayesian simple linear regression model for predicting the car stopping distance at different speeds. While this toy example is trivial, it will be akin to the Bayesian approach (implicitly) used in all papers.

The data that we will use is the `R-dataset` *cars* [R Core Team, 2018]. The data is plotted in Figure 1a. We will consider the task of predicting the stopping distance (in feet) as a function of the speed (in miles per hour). And for this purpose, we introduce the following Bayesian simple linear regression model

$$\begin{cases} y_i = \beta_0 + \beta_1 x_i + \epsilon_i,\ \epsilon_i \sim \mathcal{N}(0, \sigma), & i = 1:50, \\ \beta_0 \sim \mathcal{N}(0, 10), \\ \beta_1, \sigma \sim \mathit{\Gamma}(2, 3). \end{cases} \tag{8}$$

In Equation (8), y_i is the stopping distance for observation i and x_i is the corresponding speed. Thus, we have that the observed data is given by $y = [y_1, \ldots, y_{50}]$, while $x = [x_1, \ldots, x_{50}]$ denotes the covariate of the model. The unknown parameter that we are uncertain about is $\theta = [\beta_0, \beta_1, \sigma]$. Our prior beliefs about the parameter are incorporated in the normal and gamma priors of the Bayesian model in Equation (8). The likelihood function for the full data set y is given by

$$p(y|x, \theta) = \prod_{i=1}^{50} \mathcal{N}(\beta_0 + \beta_1 x_i, \sigma). \tag{9}$$

The first step in our analysis is to investigate if our priors are reasonably informative of the parameter. To investigate this, we sample from the prior predictive distribution. Prior predictive samples are presented in Figure 1b, and they show that a large range of data sets can be generated from the prior predictive distribution, indicating that the priors are weakly informative of the parameter.

The next step is to obtain the posterior distribution. For this example, we generate samples from the posterior distribution via the standard Metropolis-Hastings[1](MH) algorithm [Hastings, 1970, Metropolis et al., 1953]. The resulting marginal posterior distributions (see Figure 1c) are quite tight (at-least for β_1 and σ); we can also conclude that β_1 is inferred with the highest precision. However, we do not know the true parameter value; thus, it is not fruitful to assess the inference quality only based on the marginal posterior distributions.

> Therefore, the final step of our analysis is to sample data from the posterior predictive distribution. The posterior predictive samples are presented in Figure 1d. We conclude that the inferred model appears reasonable since the posterior predictive samples in Figure 1d are similar to the observed data set. Thus, when sampling parameters from the posterior, we manage to generate data that resembles the observed data set, which indicates that our model is realistic for our observed data.

Example 1 above shows how to conduct a Bayesian analysis for a simple toy problem. However, several additional steps can be included in the analysis [Gelman et al., 2020]. But, we have here focused on the procedures that are most commonly used in the papers included in this thesis.

2 The Bayesian implicit model

A *simulator*

$$x = \mathcal{M}(\theta), \text{ or, more precisely } x = \mathcal{M}(\theta, z), \tag{10}$$

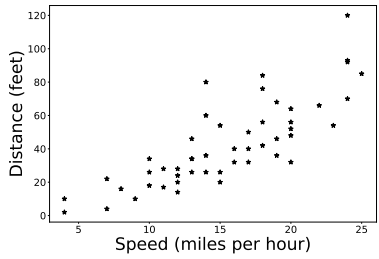
is some *random mechanism* that takes as input the parameter θ and the underlying pseudo-random numbers z and returns a data set x. For the implicit Bayesian statistical model

$$\begin{cases} x & \sim p(x|\theta), \\ \theta & \sim p(\theta), \end{cases} \tag{11}$$

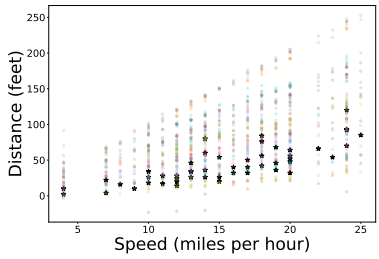
the likelihood function $p(x|\theta) := \mathcal{M}(\theta)$ is now a *simulator* which allows us to simulate data from the likelihood give the parameter θ. However, the likelihood now implicitly depends on the pseudo-random numbers z that are used internally in the random mechanism that simulates data [Cranmer et al., 2020]. Thus we have that the random mechanism that simulates data from the likelihood $p(x|\theta)$ is described by following

$$x \sim p(x|\theta, z), \qquad z \sim g(z). \tag{12}$$

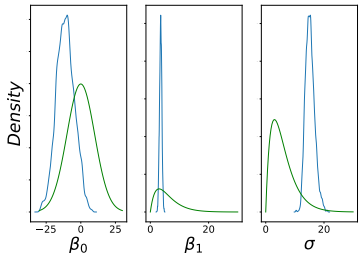
[1]There is a controversy regarding the naming of this algorithm since its name does not correctly credit the co-authors' (and particularly Arianna Rosenbluth's) contributions to the original paper Metropolis et al. [1953] [Carrier, 2021]. However, in this thesis, we will use the name *Metropolis-Hastings* since that is the current naming convention.



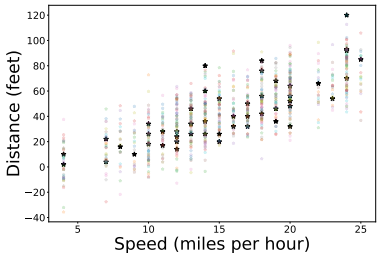
(a) *cars* data set from `R-dataset`



(b) Prior predictive simulations. The observed data set is in black. Each sample is represented with a separate color.



(c) Marginal posteriors. The blue lines show the kernel density estimations of the marginal posteriors. The green lines show the prior distributions.



(d) Posterior predictive simulation. The observed data set is in black. Each sample is represented with a separate color.

Figure 1: Car stopping distance example. The subplots show: (a) observed data; (b) prior predictive simulations; (c) marginal posteriors (and prior distributions); and, (d) posterior predictive simulations.

Hence, the model $p(x|\theta, z)$ will simulate data x for some parameter θ by utilizing some internal pseudo-random numbers z, drawn from some distribution $g(z)$, that governs the internal randomness of the simulator. The simulator described in Equation (12) is illustrated in Figure 2. The likelihood $p(x|\theta)$ is obtained by integrating out the pseudo-random numbers z

$$p(x|\theta) = \int p(x, z|\theta)dz = \int p(x|z, \theta)g(z)dz. \tag{13}$$

However, the integral in Equation (13) is analytically intractable and we can therefore not evaluate the probability density function (pdf) of $p(x|\theta)$. But since simulations from $p(x|\theta)$ are possible we can instead *implicitly* define the likelihood via the random mechanism that simulates data from $p(x|\theta)$. We therefore say the that model

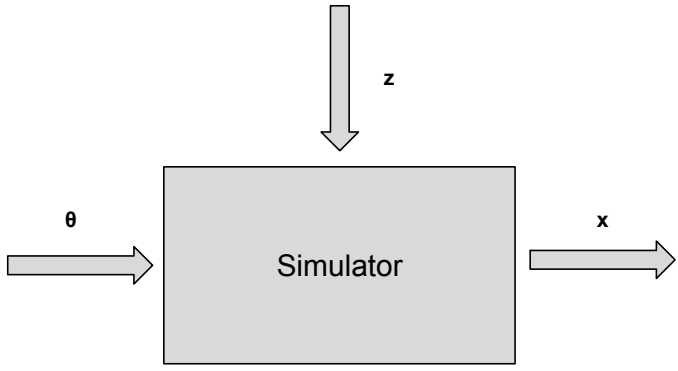


Figure 2: Illustration of the simulator of the Bayesian implicit model.

$p(x|\theta)$ is an an *implicit* statistical model [Bretó et al., 2009], which also entails that the model in Equation (11) is denoted a *Bayesian implicit model*.

To specify the implicit Bayesian model in Equation (11), we also have to introduce a prior distribution $p(\theta)$ over the parameter θ. Similarly, as before, the prior $p(\theta)$ incorporates our a-priori beliefs about the parameter θ.

Implicit models are advantageous since the model (or simulator) $p(x|\theta)$ can be implemented as a black-box computer program. Many scientific models are also described by complex generative processes for which simulation-based inference methods are suitable [Cranmer et al., 2020].

The main goal for our analysis is to in some way approximate, or sample from the posterior distribution

$$p(\theta|x^{\text{obs}}) \propto p(x^{\text{obs}}|\theta)p(\theta). \tag{14}$$

Here the posterior distribution $p(\theta|x^{\text{obs}})$ is the posterior distribution that we obtain when conditioning on the specific data set $x^{\text{obs}} \in \mathcal{X}$, which is assumed to be generated by the model $p(x|\theta)$. In a simulation study we of course know that x^{obs} indeed was generated by $p(x|\theta)$, but for a real-data case study we have to assume that the observed data x^{obs} indeed was generated from the model $p(x|\theta)$. However, we will typically not consider the model uncertainty related to whether the model $p(x|\theta)$ is the *true* data generating process of x^{obs}. For recent amortized simulation-based inference methods methods [Radev et al., 2020, Hermans et al., 2020] the posterior of interest is the *global* posterior distribution denoted by $p(\theta|x)$. The global posterior distribution is the posterior distribution for *any* data set $x \in \mathcal{X}$ produced by

the model $p(x|\theta)$. The distinction between the global posterior and the posterior distribution $p(\theta|x^{\text{obs}})$ might seem subtle, but this distinction will be of importance in Section 4.7 where we will briefly present amortized inference methods.

3 Applications

We will now give some examples of implicit models that are common in the literature and that are also used as examples in the papers included in this thesis.

3.1 Static models

We will present three static models: the *G-and-k distribution*, the *α-stable distribution*, and the *two-moons model*. In this thesis, the G-and-K distribution and the α-stable distribution are used as case studies in paper I. The two-moons model is considered in paper III. (However, please notice that we here present a slightly different version of the two-moons model, compared to the one included in paper III).

G-and-K distribution A classical application in the simulation-based inference literature is the G-and-K distribution, used for instance in Allingham et al. [2009], Picchini and Anderson [2017], and Fearnhead and Prangle [2012]. The G-and-K distribution is a flexible univariate distribution used to model highly skewed data [Prangle, 2017]. The G-and-K distribution is defined via its quantile function and its probability density function (pdf) is unavailable in closed-form [Prangle, 2017]. However, the pdf can be approximated numerically [Prangle, 2017]. The G-and-K distribution is parameterized with $\theta = [A, B, g, k, c]$, and it is common practise to let $c = 0.8$ [Allingham et al., 2009, Picchini and Anderson, 2017, Fearnhead and Prangle, 2012], and a valid distribution is then produced if $B > 0$ and $K \geq 0$ [Prangle, 2017]. A sample $x \in \mathbb{R}$ from the G-and-K distribution is generated via [Prangle, 2017]

$$x = A + B \cdot (1 + c \cdot \tanh(g \cdot \frac{z}{2})) \cdot z \cdot (1 + z^2)^k, \qquad z \sim \mathcal{N}(0, 1). \tag{15}$$

We notice that Equation (15) allows us to generate samples from the model, even though we cannot evaluate the pdf of the model.

α-stable distribution The α-stable distribution is defined via its characteristic function [Ong et al., 2018]. Thus, the α-stable distribution is another distribution with an analytically intractable pdf. Since the pdf is intractable, the inference problem is challenging; however, simulation-based approximate Bayesian inference approaches can be found in Peters et al. [2012], and Ong et al. [2018]. The α-stable distribution is heavy-tailed, and it is therefore used for financial applications [Peters et al., 2012]. The characteristics function $\varphi(s)$ is given by [Ong et al., 2018]

$$\varphi(t) = \begin{cases} \exp\Big(i\delta t - \gamma^\alpha |t|^\alpha \big(1+ \\ \qquad i\beta \tan\tfrac{\pi\alpha}{2} \mathrm{sgn}(t)(|\gamma t|^{1-\alpha} - 1)\big)\Big), & \alpha \neq 1, \\ \exp\Big(i\delta t - \gamma|t|\big(1 + i\beta\tfrac{2}{\pi}\mathrm{sgn}(t)\log(\gamma|t|)\big)\Big), & \alpha = 1, \end{cases} \tag{16}$$

where sgn is the sign function, i.e.,

$$\mathrm{sgn}(t) = \begin{cases} -1 & \text{if } t < 0, \\ 0 & \text{if } t = 0, \\ 1 & \text{if } t > 0. \end{cases} \tag{17}$$

The unknown parameter of the α-stable distribution is $\theta = [\alpha, \beta, \gamma, \delta]$.

Two-moons model A rather simple static model with a complex posterior is the two-moons model [Greenberg et al., 2019]. The main feature of the two-moons model is that the posterior, for certain parameter values, is crescent-shaped [Greenberg et al., 2019]. (The analytical posterior of the two-moons model for a case where the posterior is crescent-shaped is presented in Figure 3.) Thus, for the two-moons model, MCMC-based inference methods (e.g *sequential neural likelihood* [Papamakarios et al., 2019b]) might not work well due to the potentially complex geometry of the posterior. The two-moons model generates data $x \in \mathbb{R}^2$ according to the following process [Greenberg et al., 2019]:

$$a \sim \mathcal{U}(-\pi/2, \pi/2), \tag{18}$$

$$r \sim \mathcal{N}(0.1, 0.01), \tag{19}$$

$$p = (r\cos(a) + 0.25, r\sin(a)), \tag{20}$$

$$x^T = p + \Big(\frac{-|\theta_1 + \theta_2|}{\sqrt{2}}, \frac{-\theta_1 + \theta_2}{\sqrt{2}}\Big). \tag{21}$$

The unknown parameter of the model is $\theta = [\theta_1, \theta_2]$.

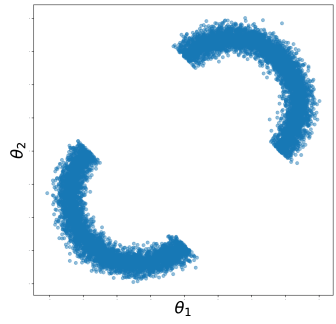


Figure 3: Samples from the analytical posterior of the two-moons model.

3.2 Standard dynamic toy models

Examples of dynamic toy models are: the *moving average model of order* q (MA(q)), and the *Lotka-Volterra model* (LV). In paper II a MA(2) model with additional measurement noise is included as one of the case studies. The LV model is one of the case studies in paper III.

Moving average model A moving average model of order q (MA(q)) is given by

$$x_i = z_i + z_{i-1}\theta_1 + z_{i-2}\theta_2 + \ldots + z_{i-q}\theta_q, \qquad i = 1 : N. \tag{22}$$

Here the z_i's are the unobservable error terms, and N the number of observations. We typically assume that $z_i \overset{iid}{\sim} \mathcal{N}(0, 1)$. Approximate Bayesian computation (ABC) methods have been used for inference in the MA(2) model in Marin et al. [2012] and Jiang et al. [2017]. Exact Bayesian inference can also be obtained since we can evaluate the pdf of the MA(q) model.

A version of the MA(2) model with additional measurement noise is used in paper I. Thus, let us now specifically discuss the MA(2) model. The MA(2) model is identifiable over the following triangle:

$$\theta_1 \in [-2, 2], \qquad \theta_2 \in [-1, 1], \qquad \theta_2 \pm \theta_1 \geq -1, \tag{23}$$

and this triangle, therefore, defines the bounds for the uniform prior in $\mathbb{R}^2$ used in Marin et al. [2012] and Jiang et al. [2017], as well as in paper I. ABC methods often utilizes summary statistics to avoid the *curse-of-dimensionality* (see Section 4.2) and natural summary statistics for the MA(2) model are the first two auto-correlations since these converge to a one-to-one function of the parameter $\theta = [\theta_1, \theta_2]$ [Jiang et al., 2017].

Lotka-Volterra The Lotka-Volterra model (LV) is another dynamic model commonly used as an example in the simulation-based inference literature (see e.g., Papamakarios et al. [2019b], Papamakarios and Murray [2016], Greenberg et al. [2019], and Picchini and Everitt [2019]).

The LV model is a Markov jump process that describes the evolution of the size of a population of predators X and population of prey Y [Wilkinson, 2011].We we will here consider the version of the LV model from Papamakarios et al. [2019b], and that version of the LV model is parameterized by $\theta = [\theta_1, \theta_2, \theta_3, \theta_4]$. For the LV model of Papamakarios et al. [2019b] the following events can occur

1. With rate $\exp(\theta_1) \times X \times Y$: A predator is born, and X increases with one.
2. With rate $\exp(\theta_2) \times X$: A predator dies, and X decreases with one.
3. With rate $\exp(\theta_3) \times Y$: A prey is born, and Y increases with one.
4. With rate $\exp(\theta_4) \times X \times Y$: A predator kills a prey, and Y decreases with one.

The LV model can be simulated exactly via the Gillespie algorithm [Gillespie, 1977]. We can define the population sizes at time t as $[X_t, Y_t]$, and N discrete observations of the LV model during the time period $0 \leq t \leq T$ constitutes the observed data set $[x_{1:N}, y_{1:N}]$, where x_i is the population size at time $0 \leq t_i \leq T$. However, it is common to introduce *hand-picked* summary statistics. (Hand-picked summary statistics are statistics of the data set $[x_{1:N}, y_{1:N}]$ that are assumed to be informative of the parameter θ.) After introducing these summary statistics we assume that the summary statistics are observed instead of the time-series $[x_{1:N}, y_{1:N}]$ [Papamakarios et al., 2019b, Papamakarios and Murray, 2016]. Machine-learning-based summary statistics have also been considered in Greenberg et al. [2019].

3.3 Protein folding modeling via stochastic differential equations

Protein folding is a complex process in which a protein chain is translated into a functional protein molecule. Studying the time dynamics of the protein folding process results in a very high-dimensional problem. Bayesian analysis of this very high-dimensional data is challenging, and one strategy is, therefore, to instead study the time dynamics along a single *reaction coordinate*. The reaction coordinate is a one-dimensional projection of the actual dynamics in high-dimensional space. The

time-dynamics of the reaction coordinate data can be modeled with diffusion processes [Best and Hummer, 2011].

Paper IV utilizes a stochastic differential equation (SDE) model to model the reaction coordinate data (which is presented Figure 4). The reaction coordinate data has the following features: marginal bimodal structure, irregular change-points where the mean shifts, and a local noise structure. Due to these data features, paper IV utilizes the following double-well potential SDE (DWP-SDE) model

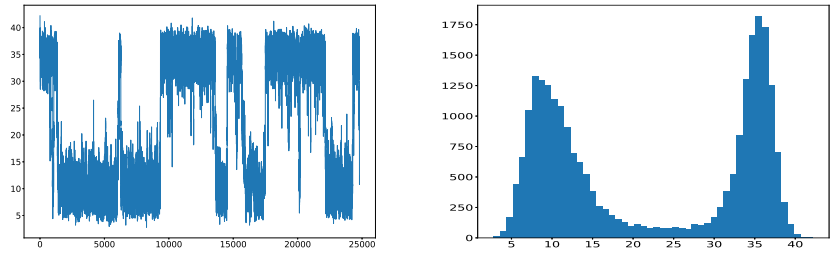
$$\begin{cases} Z_t = X_t + Y_t, \\ dX_t = -\nabla V(X_t)\, dt + \sigma\, dW_t^x, \\ dY_t = -\kappa Y_t\, dt + \sqrt{2\kappa\gamma^2}\, dW_t^y. \end{cases} \tag{24}$$

In Equation (24), Z_t is the one-dimensional observable process, which consist of the sum of the solution of the latent one-dimensional DWP-SDE process X_t and the one-dimensional noise process Y_t. The noise process Y_t is an Ornstein–Uhlenbeck process, which means the the noise process allows for auto-correlated noise. W_t^x and W_t^y are two independent Brownian motions. The potential function $V(X_t)$ in Equation (24) is a DWP parameterized as

$$V(x) = \frac{1}{2}\left|\frac{1}{2}|x-c|^{p_1} - d + gx\right|^{p_2} + \frac{1}{2}Ax^2. \tag{25}$$

The DWP function in Equation (25) is inspired by the potential described in Equation 1 of Fang et al. [2017]. The DWP function in Equation (25) is quite flexible in the sense that many different potentials can be specified by varying the parameters of the potential. The parameters of the DWP function in Equation (25) have the following interpretations: c specifies the location for the potential (i.e. where the potential is centered); d determines the spread of the potential; A is an asymmetry parameter; g compresses the two modes of the long term (stationary) density of the latent process x_t; parameters p_1 and p_2 control the shape of the two modes (if the parameters p_1 and p_2 are set to low values the long term probability distribution becomes more flat with less distinct modes); and σ governs the noise in the latent X_t process. The noise process Y_t of the DWP-SDE model in Equation (24) is an Ornstein-Uhlenbeck process specified by parameters κ and σ, where κ is the auto-correlation level, and γ is the noise intensity. All parameters of the DWP-SDE model in Equation (24) are positive so we would in principle be interested in inferring the full set of parameters $\theta = [\log\kappa, \log\gamma, \log A, \log c, \log d, \log g, \log p_1, \log p_2, \log\sigma]$. However, the parameters A and g are "stiff", i.e., small changes in their values result in considerable changes in the output, and are therefore hard to estimate. The parameter A and g are therefore assumed to be known in paper IV and the following parameters are instead

inferred $\theta = [\log \kappa, \log \gamma, \log c, \log d, \log p_1, \log p_2, \log \sigma]$. Some promising inference results are presented in paper IV. However, additional analyses (still under development) show: 1) that the DWP function in Equation (25) is over-parameterized, and 2) that the DWP-SDE model's performance can improve by considering a less flexible DWP function with fewer parameters.



(**a**) Data time course

(**b**) Empirical marginal distribution of the reaction coordinate data.

Figure 4: Reaction coordinate data.

3.4 Stochastic differential equations mixed-effects models

Stochastic differential equations mixed-effects models (SDEMEMs) is a class of stochastic dynamic models for population inference using mixed-effects, where dynamics are driven by stochastic differential equations [Lavielle, 2014]. Applications of SDEMEMs can for instance be found in pharmacokinetic/pharmacodynamic models [Donnet and Samson, 2013] and single-cell models [Persson et al., 2021] (i.e, in paper *A*). SDEMEMs distinguish between three sources of variability: 1) the between-individual variability, 2) the intrinsic stochasticity of the model, and 3) the residual variability due to measurement error. As an illustration, simulated data from an Ornstein-Uhlenbeck SDEMEM model (i.e, a SDEMEM model with latent dynamics driven by an Ornstein-Uhlenbeck process) is presented in Figure 5. Indeed, in Figure 5 we consider data from M trajectories. Each trajectory is one individual or unit of the model and we see that the trajectories share some common structure, but that each trajectory also has some individual variability.

Mathematically, we describe the SDEMEM by the following

$$\begin{cases} Y_t^i & = h(X_t^i, \epsilon_t^i) \qquad \epsilon_t^i | \xi \overset{indep.}{\sim} p_\epsilon(\xi), \qquad i = 1:M, \\ dX_t^i & = \alpha(X_t^i, \phi^i)\, dt + \sqrt{\beta(X_t^i, \phi^i)}\, dW_t^i, \\ \phi^i & \sim p(\phi^i | \eta). \end{cases} \tag{26}$$

Here Y_t^i and X_t^i are the d-dimensional observable process and latent process for individual i, respectively. Furthermore, we have that the SDEMEM consists of M individuals, and for each individual, we have discrete observations y_j^i. The index i runs over the individuals and the index j denotes the j:th observation (of individual i) at time-point t_j. The latent process X_t^i is a solution to an SDE parameterized with a different parameter (or random effect) ϕ^i for each individual i. Each latent process is also driven by an independent Brownian motion W_t^i. Each latent process models the intrinsic stochasticity of the data for the corresponding individual, and the between-individual variability is accounted for via the random effects. The function h maps the latent process to the observations, and the residual variability due to the measurement error, governed by ξ, also enters the system here. In paper II, h can be both non-linear and non-Gaussian, which allows for flexible modeling. Finally, $p(\phi^i|\eta)$ is a distribution parameterized by η that models the random effects. The parameter of inferential interest is $\theta = [\xi, \eta]$, Hence, we are interested in inferring the parameter η that governs the model for the random effects and the measurement error parameter ξ, but we are not interested in inferring the random effects $\phi^{1:M}$. The SDEMEM can also include a *non-varying* parameter κ that is the same for all individuals. In the case where the non-varying parameter κ is included the parameter of inferential interest is $\theta = [\xi, \kappa, \eta]$.

4 Simulation-based inference

The following section constitutes the main component of this thesis, and the section is structured as follows: Section 4.1 introduces the mathematical tools (state-space models and particle filters, deep learning, and neural density estimation) utilized in the thesis. In Sections 4.2–4.6 we introduce the different simulation-based inference methods that are used in this thesis. Sections 4.2–4.6 also show how the mathematical tools of Section 4.1 are utilized for simulation-based inference. Some additional simulation-based inference methods not primarily considered in the thesis are briefly discussed in Section 4.7.

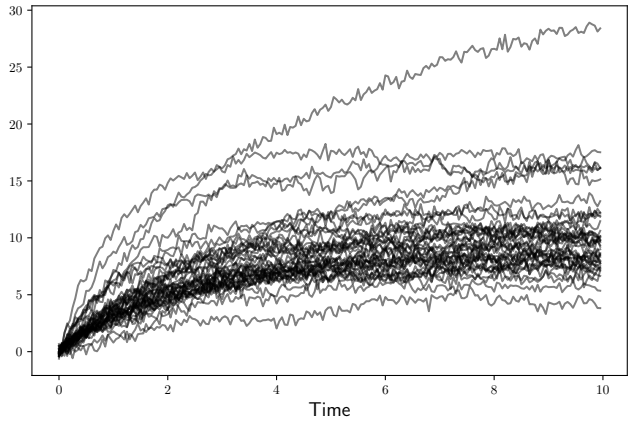


Figure 5: Simulated trajectories of an Ornstein-Uhlenbeck SDEMEM.

4.1 Mathematical tools

4.1.1 State-space models and particle filters

State-space models In experiments we often measure some quantity of interest X. However, our measurements are commonly also effected by some exogenous noise (i.e *measurement error*) so that we actually obtained observations from

$$Y = X + \text{error}. \tag{27}$$

We can also have the case that we obtain observations from

$$Y = h(X) + \text{error}. \tag{28}$$

Here, $h(\cdot)$ is some possible non-linear function.

State-space models (SSMs) are a special case where X is a latent (unobservable) Markov process taking values on a continuous set. The latent Markov process is governed by the transition density $p(x_t|x_s), t > s$. The observable process Y is governed by the conditional density $p(y_t|x_t)$, and is such that observations $Y_1, \ldots, Y_N$ at time points $t_1 : t_N$ are conditionally independent given states of the latent process $X_1, \ldots, X_N$. SSM are computationally tractable (see below) and have been successfully used for applications in signal processing, ecology, epidemiology, neuroscience, and finance [Chopin et al., 2020, chapter 1].

The SSM is given by

$$\begin{cases} y_t & \sim p(y_t|x_t;\theta^y), \\ x_t & \sim p(x_t|x_s;\theta^x), \qquad x_0 \sim p(x_0) \qquad t_0 \leq s < t. \end{cases} \tag{29}$$

Here the process $x_t \in \mathbb{R}^{\dim x}$ is the unobservable latent process, where the initial state is governed by the model $p(x_0)$. The process $y_t \in \mathbb{R}^{\dim_y}$ is the observable process at time point t. The time-discrete representation of the SSM model is illustrated in Figure 6. In Equation (29), θ^y and θ^x are the parameters of the observable and latent process, respectively. Thus, the full set of unknown parameters is $\theta = [\theta^y, \theta^x]$. The data that we observe is the time-series $y = [y_1, y_2, \ldots, y_N]$ of in total N observations at time-points $t_1, t_2, \ldots t_N$. Thus we have that y_i is the observation of system at time t_i. The SSM framework is a flexible modeling framework since both $p(y_t|x_t, \theta^y)$ and $p(x_y|x_s, \theta^x)$ can be non-linear and non-Gaussian. A simple SSM used in the literature for benchmarking purposes is presented in Example 2 [Cappé et al., 2007].

Example 2 (Simple non-linear Gaussian SSM)

Let us now consider the following SSM governed by

$$\begin{cases} y_t & = \frac{x_t^2}{20} + v_t, \\ x_t & = \frac{x_{t-1}}{2} + 25\frac{x_{t-1}}{1+x_{t-1}^2} + 8\cos(1.2t) + u_t, \end{cases} \tag{30}$$

where, $u_t \sim \mathcal{N}(0, \sigma_u)$ and $v_t \sim \mathcal{N}(0, \sigma_v)$. The parameters σ_u and σ_v are typically assumed to be known and fixed at $\sigma_u = \sqrt{10}$ and $\sigma_v = 1$. The initial state for the model in Equation (30) follows $x_0 \sim \mathcal{N}(0, \sqrt{10})$. Notice that Equation (30) introduces the time-discrete representation of the model. However, we can also write the model in Equation (30) in terms of the the general SSM of Equation (29) and we then have

$$\begin{cases} y_t & \sim \mathcal{N}(\frac{x_t^2}{20}, \sigma_u = \sqrt{10}), \\ x_t & \sim \mathcal{N}(\frac{x_s}{2} + 25\frac{x_s}{1+x_s^2} + 8\cos(1.2t), \sigma_v = 1), \\ x_0 & \sim \mathcal{N}(0, \sqrt{10}), \qquad t_0 \leq s < t. \end{cases} \tag{31}$$

In the case where both the latent model $p(x_y|x_s;\theta^x)$ and the observation model $p(y_t|x_t;\theta^y)$ are linear and Gaussian the Kalman filter [Kalman, 1960] can be used to obtain the likelihood function exactly. Cases where $p(x_y|x_s;\theta^x)$ is non-linear but still Gaussian can be handled by advanced Kalman filters (e.g., the extended Kalman

filter [Särkkä, 2013, chapter 5.2]). However, for general non-linear and non-Gaussian SSMs, particle filters (i.e., sequential Monte Carlo) must be deployed.

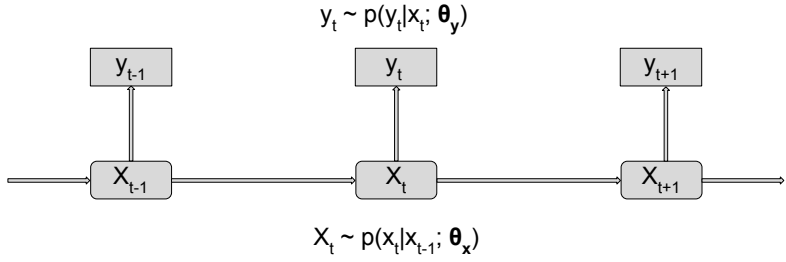


Figure 6: Illustration of the general time-discrete SSM.

Particle filters Particle filters or (sequential Monte Carlo methods) are methods for filtering, inference, and smoothing in state-space models (SSM) [Särkkä, 2013, Cappé et al., 2007, Chopin et al., 2020]. The term *sequential Monte Carlo*, however, also refers to a broader group of methods that can be studied under the general Feynman-Kac model [Chopin et al., 2020, chapter 4]. However, in this thesis, we only use particle filters to estimate the likelihood of SSMs. We will, therefore, not further discuss the Feynman-Kac formulation of particle filters.

The likelihood of the general SSM is given by

$$p(y_{1:N}|\theta) = p(y_1|\theta) \prod_{n=2}^{N} p(y_n|y_{1:n-1}; \theta). \tag{32}$$

where,

$$p(y_n|y_{1:n-1}; \theta) = \int p(y_n, x_n|y_{1:n-1}; \theta) dx_n, \tag{33}$$

$$= \int p(y_n|x_n; \theta^y) p(x_n|y_{1:n-1}; \theta^x) dx_n. \tag{34}$$

However, the integral in Equation (34) is in the general case intractable, but it can be estimated via the following Monte Carlo estimate:

$$\hat{p}(y_n|y_{1:n-1}; \theta) \approx \frac{1}{P} \sum_{p=1}^{P} p(y_n|\tilde{x}_n^p; \theta^y), \qquad \tilde{x}_n^{1:P} \overset{indep.}{\sim} p(x_n|y_{1:n-1}; \theta), \tag{35}$$

where the x_n^p's are draws from $p(x_n|y_{1:n-1};\theta)$. Simulating from $p(x_n|y_{1:n-1};\theta)$ is achieved by utilizing particle filters, such as the bootstrap filter [Gordon et al., 1993, Stewart and McCarty Jr, 1992] (outlined in Algorithm 1). The advantages of the bootstrap filter are: it is easy to implement, it does not rely on selecting suitable proposal distributions, and it is not restricted to special types of SSMs. However, for challenging SSMs, it can be advantageous to use the auxiliary particle filter [Pitt and Shephard, 1999], or if the model allows for it, the bridge filer [Golightly and Wilkinson, 2011].

Algorithm 1 Bootstrap particle filter

Input: Data $y_{1:N}$, number of particles P, and model parameter θ.
Output: The likelihood estimation $\hat{L}_{PF}(\theta)$.

1: Initialize particles: $\tilde{x}_0^{1:P} \sim p(x_0)$.
2: Propagate particles: For $p = 1:P$, sample, $x_1^p \sim p(\,\cdot|\tilde{x}_0^p;\theta^x)$. ▷ First iteration
3: Compute importance weights: For $p = 1:P$, compute,

$$w_1^p = p(y_1|x_1^p), \qquad \tilde{w}_1^p = \frac{w_1^p}{\sum_{p=1}^P w_1^p}. \tag{36}$$

4: Compute likelihood estimate: $\hat{p}(y_1|\theta) = \frac{\sum_{p=1}^P w_1^p}{P}$.
5: **for** $n = 2:N$ **do** ▷ Loop over iterations
6: (optional) Sorting: Euclidean sort the particles $(x_{n-1}^1, ..., x_{n-1}^P)$.
7: Resample: Sample P times with replacement from $(x_{n-1}^1, ..., x_{n-1}^P)$ with associated probabilities $(\tilde{w}_{n-1}^1, ..., \tilde{w}_{n-1}^P)$ to obtain a new sample $(\tilde{x}_{n-1}^1, ..., \tilde{x}_{n-1}^P)$.
8: Propagate: For $p = 1:P$, sample, $x_n^p \sim p(\,\cdot|\tilde{x}_{n-1}^p;\theta^x)$.
9: Compute importance weights: For $p = 1:P$, compute,

$$w_n^p = p(y_n|x_n^p;\theta^y), \qquad \tilde{w}_n^p = \frac{w_n^p}{\sum_{p=1}^P w_n^p}. \tag{37}$$

10: Compute likelihood estimate: $\hat{p}(y_n|y_{1:n-1};\theta) = \frac{\sum_{p=1}^P w_n^p}{P}$.
11: **end for**
12: Compute full likelihood estimate:

$$\hat{L}_{PF} := \hat{p}(y|\theta) = \hat{p}(y_1|\theta) \prod_{n=2}^{N} \hat{p}(y_n|y_{1:n-1};\theta). \tag{38}$$

4.1.2 Deep learning

Deep learning is a class of machine learning methods that can be used for several tasks, including but not limited to classification, regression, and generative models. The defining feature of deep learning methods is that they *automatically* construct useful representations of the input data and combine these simple representations into complex representations [Goodfellow et al., 2016, chapter 1]. Recent well-known applications of deep learning approaches are, for instance, DeepMind's AlphaGo model used to play the game of Go [Singh et al., 2017], and DeepMind's AlphaFold model used to predict protein structures [Jumper et al., 2021].

Deep learning regression methods are used in paper I to learn summary statistics for ABC (the methodology of paper I is discussed in Section 4.2.1). Furthermore, deep generative models (normalizing flows, see Section 4.1.3) are used in Paper III for the construction of a novel simulation-based inference algorithm. We will first focus on deep learning regression methods, then we will give a short presentation of the deep learning architectures used in paper I.

The multilayer perceptron network Inspired by Higham and Higham [2019], we will introduce the multilayer perceptron (MLP) network [Goodfellow et al., 2016, chapter 6] by considering the simple MLP model in Example 3.

Example 3 (A simple 2-hidden-layer MLP network)

Figure 7 presents a simple two-hidden-layer MLP network. In the notation below, the input layer is layer 1, the two hidden layers are layers 2 and 3, and the output layer is layer 4. The function $f : \mathbb{R}^8 \to \mathbb{R}^4$ that the network in Figure 7 learns is

$$f(x) = g^4(w^4 g^3(w^3 g^2(w^2 x + b^2) + b^3) + b^4), \tag{39}$$

where $x \in \mathbb{R}^8$ is the input. The trainable weights $w^{2:4}$ and biases $b^{2:4}$ have the following dimensions

$$\begin{cases} w^2 & \in \mathbb{R}^{12\times 8}, & b^2 \in \mathbb{R}^8, \\ w^3 & \in \mathbb{R}^{10\times 12}, & b^3 \in \mathbb{R}^{10}, \\ w^4 & \in \mathbb{R}^{4\times 10}, & b^4 \in \mathbb{R}^4. \end{cases} \tag{40}$$

Thus, the full set of learnable parameters are $\phi = [w^2, b^2, w^3, b^3, w^4, b^4]$. The functions $g^{2:4}$ are the activation functions that introduces the nonlinearities to the model. Typically, $g^{2:3}$ are *ReLU* activation functions [Glorot et al., 2011], and

since we are considering a regression task g^4 will most commonly be the identity function. Now, let us denote with $a^i, i = 1 : 4$ the i:th action, or output, of the network, we can then write the model in Equation (39) as

$$\begin{cases} a^1 &= x, \\ a^i &= g^i(w^i a^{i-1} + b^i), \qquad i = 2 : 4. \end{cases} \tag{41}$$

Now, assume that we have training data $\left(x^n, y^n(x^n)\right)_{n=1:N}$, consisting of the features $x^{1:N}$ and the associated targets $y^{1:N}(x^{1:N})$. The quadratic loss function with respect of the trainable parameters of the model is then

$$\mathcal{L}(\phi) \propto \sum_{n=1}^{N} \left|\left|y^n(x^1) - a^4(x^n)\right|\right|_2^2. \tag{42}$$

The model in Equation (39) is now trained by minimizing the loss of Equation (42). For this task is typically the *mini-batch stochastic gradient decent* algorithm used. One step of the mini-batch stochastic gradient descent algorithm is computed as following:

1. Select integers $k_{1:m}$ at random without replacement from $1 : N$.

2. Update the parameter ϕ according to

$$\phi \rightarrow \phi - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla \mathcal{L}_{x^{k_i}}(\phi). \tag{43}$$

Here $\nabla \mathcal{L}_{x^{k_i}}(\phi)$ is the gradient of the loss computed for the data point x^{k_i}, and η is the *learning rate*. The learning rate is akin to the step-length of numerical optimization, and the learning rate governs how quickly we move or update the weights ϕ in the weight-space. Assuming that we have selected m such that $\mod(N, m) = 0$, then after running the algorithm above for N/m iterations, we have cycled through the training data $\left(x^n, y^n(x^n)\right)_{n=1:N}$; and, thus, we have completed one epoch. The standard approach to compute the gradient $\nabla \mathcal{L}_{x^{k_i}}(\phi)$, which consists of the partial derivatives of all weights and biases, is to use the *back-propagation* (backprop) algorithm [Rumelhart et al., 1986]. The main feature of backprop is that it only requires one forward-pass of the network to collect all components that are needed to compute the partial derivatives of all weights and biases. Computing one forward-pass consists of feeding some input to the network's input layer and letting this input traverse through the hidden layers until we finally obtain the result of the output layer.

The example above shows via a simple example how the multilayer perceptron model is constructed and how this model is trained. However, in this thesis, models are not implemented from scratch. We instead employ deep learning frameworks with automatic differentiation [Baydin et al., 2018] capabilities (e.g., `PyTorch` [Paszke et al., 2019]), which allows us to easily and efficiently develop novel architectures.

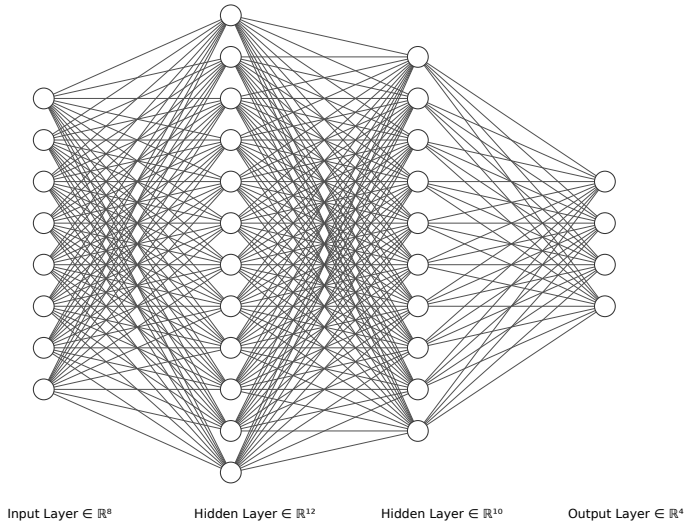


Figure 7: Example of a multilayer perceptron network. The illustration was created with the online tool at `http://alexlenail.me/NN-SVG/index.html`.

Leveraging features of the data While the MLP architecture is simple and useful for certain tasks it has the distinct drawback that knowledge of the structure of the input data $x \in \mathcal{X}$ is not exploited. Which means that the structures of the data x also must be learned since the MLP architecture is not adapted for them. For instance, if x is a time-series the MLP architecture will need to learn both the time-dependency of the features of x and the transformation $f : \mathcal{X} \to \mathcal{Y}$. If we instead employ a convolutional neural network (CNN) [LeCun et al., 1989] the time-dependency of x can more easily be utilized since the CNN architecture is set up to leverage this specific property. Similarly, if x is an image with some temporal structure it is also likely that a CNN architecture that leverages this property of the data would perform better than an MLP architecture. Thus, a benefit of utilizing deep learning models is that the models can be adapted to the features of the data, while still allowing for flexible and non-parametric modelling.

i.i.d data Now, assume that the data $x = [x_1, x_2, \dots x_U] \in \mathfrak{X}^U, x_i \sim \mathfrak{X}$ consists of U units which are *i.i.d*, and that $\mathfrak{X}$ is a countable space. Then, for a permutation invariant function $f : 2^{\mathfrak{X}} \to \mathcal{Y}$, we have that

$$f(x_1, x_2, \dots, x_U) = f(x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(U)}), \tag{44}$$

where π is any permutation of the data. Notice that the range space $\mathcal{Y}$ does not have to be countable. We also say that the function f is exchangeable, since it is invariant to the permutations for the *i.i.d* data x. Thus, if the data x is *i.i.d*, we would like to set up an network architecture that is permutation invariant. Designing permutation invariant networks at least dates back to [Minsky and Papert, 1988, Chater 2] and Shawe-Taylor [1989], and a recent review on the topic is Bloem-Reddy and Teh [2019]. However, we will here focus on Zaheer et al. [2017], in which they introduce the deep learning architecture *DeepSets*. Deepsets is permutation invariant and thus suitable for *i.i.d* data. The DeepSets architecture learns the function $f : \mathfrak{X}^U \to \mathcal{Y}$ which is a composition of the functions $\phi : \mathfrak{X} \to \mathbb{R}$ and $\rho : \mathbb{R} \to \mathcal{Y}$, such that

$$f(x) = \rho(\sum_{i=1}^{U} \phi(x_i)), \tag{45}$$

where ρ and ϕ are suitable functions, and $\mathcal{Y} = \mathbb{R}$. The functions ρ and ϕ are in practice parameterized by MLP networks. Zaheer et al. [2017] provides a universal approximation theorem for the decomposition in Equation (45). Hence, they show that the decomposition in Equation (45) is valid for suitable transformations ϕ and ρ.

Markovian data In paper I we consider the problem of constructing invariant networks for Markovian data of order d. To solve this task, paper I introduces the network architecture *partially exchangeable network* (PEN), which is invariant for Markovian data. The PEN architecture of order d learns the function $f : \mathfrak{X}^U \to \mathcal{Y}$, which is a composition of the functions $\phi : \mathfrak{X}^{d+1} \to \mathbb{R}$ and $\rho : \mathfrak{X}^d \times \mathbb{R} \to \mathcal{Y}$, such that

$$f(x) = \rho(x_{1:d}, \sum_{i=1}^{U-d} \phi(x_{i:(i+d)})). \tag{46}$$

We notice that PEN of order $d = 0$ corresponds to DeepSets. Furthermore, to most efficiently leveraging the Markovian property of the data, the order of the PEN architecture should be the same as the Markov order of the data. In paper I we

provide a universal approximation theorem for the architecture in Equation (46). We also show in paper I how the PEN architecture can be used to develop data-efficient models for learning summary statistics for Markovian models that are informative for use in approximate Bayesian computation algorithms.

We end this section by concluding that deep learning models can be powerful tools for function approximation tasks. An important feature of deep learning architectures is that they can be set up to leverage features of the data, which can allow us to develop more data-efficient and interpretable models.

4.1.3 Conditional neural density estimation

A conditional neural density estimator (CNDE) is a model $\tilde{p}_\phi(u|v)$ that takes inputs $u \in \mathcal{U}$ and $v \in \mathcal{V}$, and outputs the conditional density of $u|v$. Thus we have that

$$\tilde{p}_\phi : \mathcal{U} \times \mathcal{V} \to \mathbb{R}^+. \tag{47}$$

The model $\tilde{p}_\phi(u|v)$ is parameterized with parameters ϕ which typically are the weights of some deep learning architecture. Given training data $(u^n, v^n)_{n=1:N}$ the CNDE model $\tilde{p}_\phi(u|v)$ can be trained by minimize total negative log-probability [Papamakarios et al., 2019b]

$$\mathcal{L}(\phi) = -\sum_{n=1}^{N} \log \tilde{p}_\phi(u^n|v^n). \tag{48}$$

Utilizing CNDEs for inference in Bayesian implicit models CDNEs can be used to learn both the posterior distribution and the likelihood function of implicit models, a feature utilized in paper III. Now, assume that we have samples $(\theta^n, x^n)_{n=1:N}$ from the prior predictive distribution; thus, we have that

$$(\theta^n, x^n) \sim p(x, \theta) = p(x|\theta)p(\theta), \qquad n = 1 : N. \tag{49}$$

If we want to learn the **posterior distribution**, we then introduce the CNDE model $\tilde{p}_\phi(\theta|x)$, which we train with the following loss

$$\mathcal{L}(\phi) = -E_{p(x,\theta)}\big(\log p_\phi(\theta|x)\big) \tag{50}$$

$$\propto E_{p(\theta)}\big(D_{KL}(p(\theta|x)||\tilde{p}_\phi(\theta|x)\big). \tag{51}$$

In Equation (51), $D_{KL}(P||Q)$ is the Kullback–Leibler (KL) divergence between the distributions P and Q. After training the model $\tilde{p}_\phi(\theta|x)$, the posterior distribution

for the observed data set x^{obs} is obtained by $\tilde{p}_\phi(\theta|x^{\text{obs}})$. The loss in Equation (51) is typically approximated via Monte Carlo.

If we instead are interested in learning the **likelihood function**, we can then introduce the corresponding CNDE model $\tilde{p}_\phi(x|\theta)$. The likelihood model $\tilde{p}_\phi(x|\theta)$ is trained via the following loss

$$\mathcal{L}(\phi) = -E_{p(x,\theta)}\big(\log p_\phi(x|\theta)\big) \tag{52}$$

$$\propto E_{p(\theta)}\big(D_{KL}(p(x|\theta)||\tilde{p}_\phi(x|\theta)\big). \tag{53}$$

Again, the loss in Equation (53) is typically approximated via Monte Carlo. Thus, we can conclude that CNDEs can be used to learn either the posterior or the likelihood with training data generated from the prior predictive distribution. However, this approach is very data-inefficient since we do not condition the training data on the observed data x^{obs}. We will, therefore, in Section 4.6 discuss more efficient algorithms for inference in implicit Bayesian models.

Gaussian Mixture Networks A simple CNDE model is the Mixture Density Network (MDN) model [Bishop, 1994]. An MDN models the density $p(u|v)$ via a Gaussian Mixture distribution, such that

$$\tilde{p}_\phi(u|v) = \sum_{k=1}^{K} \mathcal{N}(u|m_k, S_k). \tag{54}$$

where K is the number of mixture components. The mixture components are parameterized with mean vectors $m_{1:K}$ and covariance matrices $S_{1:K}$ computed with an MLP network parameterized with weights ϕ. The MDN model can learn complex distributions if K is large and the network that computes the m_k's and S_k's has a large capacity. MDNs have been used for simulation-based inference in Papamakarios and Murray [2016], and Lueckmann et al. [2017].

Normalizing flows The normalizing flow model [Rezende and Mohamed, 2015] (see also Kobyzev et al. [2020] and [Papamakarios et al., 2019a] for reviews) is a more complicated CNDE that has become popular to use for simulation-based inference recently [Greenberg et al., 2019, Papamakarios et al., 2019b, Radev et al., 2020].

The unconditional normalizing flow model is a probabilistic model that transforms a simple *base distribution* $u \sim p_u(u), u \in \mathcal{U}$ into some complex *target distribution* $x \sim p_x(x), x \in \mathcal{X}$ via the following transformation

$$x = T(u), \qquad u \sim p_u(u). \tag{55}$$

The function T is parameterized with an *invariant* neural network such that T^{-1} exists, and where both T and T^{-1} are differentiable, i.e. the function T is *diffeomorphic*. Thus, the normalizing flow consists of the two functions

$$T : \mathcal{U} \to \mathcal{X}, \tag{56}$$

$$T^{-1} : \mathcal{X} \to \mathcal{U}. \tag{57}$$

In Equation (56), T takes a simple distribution $p_u(u)$ (typically a Gaussian or uniform distribution) and maps that distribution into the complex target distribution of interest $p_x(x)$. An illustration of the learning process of the normalizing flow transformation T is presented in Figure 8. The function T^{-1} in Equation (57) does the exact inverse operation, i.e. T^{-1} maps the target distribution x into the simple base distribution $p_u(u)$. The probability density function (pdf) of x is computed via the change of variables formula

$$\begin{cases} p(x) &= p_u(u)|\det J_T(u)|^{-1}, \qquad u \sim T^{-1}(x), \\ p(x) &= p_u(T^{-1}(x))|\det J_{T^{-1}}(x)|. \end{cases} \tag{58}$$

In Equation (58), J_T is the Jacobian of T, and correspondingly, $J_{T^{-1}}$ is the Jacobian of T^{-1}. Due to the structure of the pdf, it is easy to construct complex transformations by composing, say, n transformations such that $T = T_1 \circ T_2 \circ \cdots \circ T_n$ where each transformation T_i is diffeomorphic, and where the Jacobian contribution of each transformation T_i can be computed. Normalizing flow models that allow for building these kinds of nested structures are, for instance, RNVP [Dinh et al., 2017], Neural Spline Flow [Durkan et al., 2019], and Masked Autoregressive Flow [Papamakarios et al., 2017].

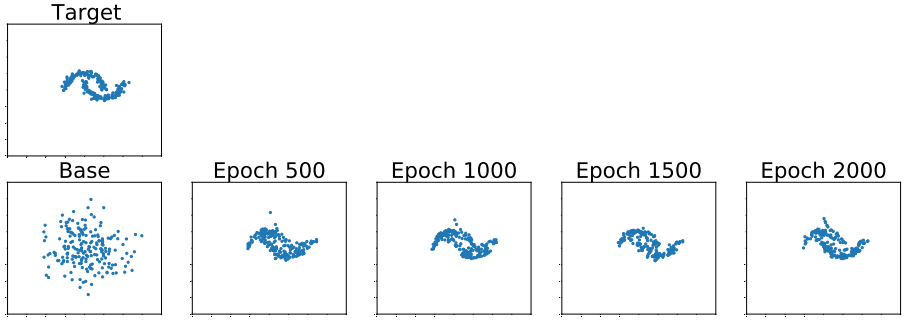


Figure 8: Illustration of a normalizing flow transformation T from the simple base distribution to the target distribution. The figure shows the approximated target distribution after 500, 1000, 1500, and 2000 epochs.

Now, assume that we have a normalizing flow model $\tilde{p}_x(x;\phi)$ (with weights ϕ of the neural network composition T), and that our target distribution is denoted $p_x(x)$.

We want to train $\tilde{p}_x(x;\phi)$ so that it approximates $p_x(x)$. Let us also assume that we can obtain samples from the target $p_x(x)$, then we can use the *forward* KL divergence to fit the flow model by utilizing the following loss function

$$\begin{aligned}\mathcal{L}(\phi) &= D_{KL}\big(p_x(x)||\tilde{p}_x(x;\phi)\big), &(59)\\ &= -E_{p_x(x)}\big(\log \tilde{p}_x(x;\phi)\big) + \text{const.}, &(60)\\ &= -E_{p_x(x)}\big(\log p_u(T^{-1}(x;\theta)) + \log|\det J_{T^{-1}}(x;\phi)\big) + \text{const.}. &(61)\end{aligned}$$

The expectation in Equation (61) is approximated via Monte Carlo.

If we do not have access to samples from the target distribution $p_x(x)$, but we can evaluate the pdf of $p_x(x)$, it is possible to fit $\tilde{p}_x(x;\phi)$ via the *reverse* KL divergence using the loss

$$\begin{aligned}\mathcal{L}(\phi) &= D_{KL}\big(\tilde{p}_x(x;\phi)||p_x(x)\big), &(62)\\ &= E_{\tilde{p}_x(x;\phi)}\big(\log \tilde{p}_x(x;\phi) - \log p_x(x)\big), &(63)\\ &= E_{p_u(u)}\big(\log p_u(u)) - \log|\det J_T(u;\phi)| - \log p_x(T(u;\phi))\big). &(64)\end{aligned}$$

It is shown in Papamakarios et al. [2017] that the forward and the reverse KL divergence are equivalent. The loss in Equation (64) is again typically approximated via Monte Carlo.

In paper III of the thesis we develop a novel simulation-based inference algorithm that leverages *conditional* normalizing flows. Hence, the problem of interest in this thesis is conditional density estimation via normalizing flows. Thus, we want to introduce a normalizing flow model $\tilde{p}_x(x|v;\phi)$ that leans the conditional target distribution $p(x|v)$, where $v \sim \mathcal{V}$ is the quantity we condition on. (For simulation-based inference applications, the quantity will either be the data x or the parameter θ, depending on if we construct a normalizing flow model for the posterior of the likelihood.) However, the normalizing flow model can readily be extended to the conditional case by considering the following transformations T and T^{-1} [Winkler et al., 2019]

$$T : \mathcal{U} \times \mathcal{V} \to \mathcal{X}, \tag{65}$$

$$T^{-1} : \mathcal{X} \times \mathcal{V} \to \mathcal{U}. \tag{66}$$

Additionally, it is also possible to utilize a conditional base distribution $u \sim p_u(u|v)$ such that the quantity we condition on enters both the transformations T and T^{-1}, and the base distribution [Winkler et al., 2019].

Normalizing flows are particularly useful for simulation-based inference since density estimation and sampling both can be done efficiently if one utilizes appropriate normalizing flows architectures [Papamakarios et al., 2019a].

4.2 Approximate Bayesian computation

Approximate Bayesian computation (ABC) [Beaumont et al., 2002, Marin et al., 2012] is a popular simulation-based inference method. A very simple ABC algorithm is ABC rejection-sampling (ABC-RS) [Tavaré et al., 1997], and the main idea of ABC-RS is to simulate parameter-data pairs $(\theta^\star, x^\star)$ from the prior predictive distribution and then keep samples $\theta^\star$ where the associated simulated data $x^\star$ is *similar* to the observed data x^{obs}. Of course, one main question is whether the approach outlined above samples parameters from the intended posterior distribution. We will here address this question by giving an example from Marin et al. [2012].

Example 4 (ABC-RS with exact data-match)

Let us consider the following case: We assume that the data $x \in \mathfrak{X}$ is countable and that we have observations $x^{\text{obs}} \sim p(x|\theta), x^{\text{obs}} \in \mathfrak{X}$. Then, we keep simulating parameter-data pairs from the prior predictive distribution

$$x^\star \sim p(x|\theta^\star), \qquad \theta^\star \sim p(\theta), \tag{67}$$

until we have that $x^\star = x^{\text{obs}}$. The distribution for accepted $\theta^\star$ obtained this way is

$$\begin{aligned} p(\theta^\star) &\propto \sum_{x^\star \in \mathfrak{X}} p(\theta^\star) p(x^\star|\theta^\star) \mathbf{1}_{x^{\text{obs}}}(x^\star), && (68)\\ &\propto p(\theta^\star) p(x^{\text{obs}}|\theta^\star), && (69)\\ &\propto p(\theta^\star|x^{\text{obs}}). && (70) \end{aligned}$$

Thus we indeed manage to sample from the correct distribution by following the procedure above.

However, comparing simulated data sets $x^\star$ to the observed data x^{obs} is often very inefficient for high-dimensional data. To avoid this *curse-of-dimensionality* one typically introduce a function $S(x) = s$ that maps the data into a set of (low-dimensional) summary statistics. To compare simulated summary statistics $s^\star$ with observed summary statistics s^{obs} we have to specify a distance function $\Delta(s^\star, s^{\text{obs}})$. In ABC the likelihood is approximated with via some unnormalized kernel $K_\epsilon(\cdot, \cdot)$ such that

$$\hat{p}^\epsilon(s^{\text{obs}}|\theta) = \int K_\epsilon\big(\Delta(s^\star, s^{\text{obs}})\big) p(s^\star|\theta) ds^\star. \tag{71}$$

Here $\hat{p}^\epsilon(s^{\text{obs}}|\theta)$ is the *ABC likelihood* for the observed summary statistics. By utilizing the likelihood approximation in Equation (71) we have that the ABC posterior

that we aim at sampling from is given by

$$\hat{p}^{\epsilon}_{\text{ABC}}(\theta|s^{\text{obs}}) \propto \hat{p}^{\epsilon}(s^{\text{obs}}|\theta)p(\theta) = \int K_{\epsilon}\big(\Delta(s^{\star}, s^{\text{obs}})\big)p(s^{\star}|\theta)p(\theta)ds^{\star}. \tag{72}$$

From Equation (72) we notice that we can view ABC as a *pseudo-marginal* method since the posterior of interest $\hat{p}^{\epsilon}_{\text{ABC}}(\theta|s^{\text{obs}})$ is obtained by marginalizing out $s^{\star}$ from the extended posterior

$$\hat{p}^{\epsilon}_{\text{ABC}}(\theta, s^{\star}|s^{\text{obs}}) \propto K_{\epsilon}\big(\Delta(s^{\star}, s^{\text{obs}})\big)p(s^{\star}|\theta)p(\theta). \tag{73}$$

For the ABC posterior, a standard choice is to use a uniform kernel, give by

$$K^{\text{unifrom}}_{\epsilon}(x) = \begin{cases} 1, \text{if } |x| \leq \epsilon, \\ 0, \text{otherwise}, \end{cases} \tag{74}$$

and the Mahalanobis distance function which is give by,

$$\Delta^{\text{Mahalanobis}}(s^{*}, s^{\text{obs}}) = \sqrt{(s^{*} - s^{\text{obs}})^{T}W(s^{*} - s^{\text{obs}})},$$

where W is the estimated covariance matrix. However, other choices are available. The threshold parameter ϵ governs the precision of the ABC posterior since we have that

$$\hat{p}^{\epsilon}_{\text{ABC}}(\theta|s^{\text{obs}}) \to p(\theta|s^{\text{obs}}), \text{ as } \epsilon \to 0, \tag{75}$$

and, correspondingly,

$$\hat{p}^{\epsilon}_{\text{ABC}}(\theta|s^{\text{obs}}) \to p(\theta), \text{ as } \epsilon \to \infty. \tag{76}$$

Thus, we wish to run ABC with the smallest possible threshold ϵ, which is computationally feasible. Furthermore, the ideal case would be that $S(\cdot)$ computes the sufficient statistics of the model since we then would have that

$$\hat{p}^{\epsilon}_{\text{ABC}}(\theta|s^{\text{obs}}) = \hat{p}^{\epsilon}_{\text{ABC}}(\theta|x^{\text{obs}}). \tag{77}$$

In Equation (77), $\hat{p}^{\epsilon}_{\text{ABC}}(\theta|s^{\text{obs}})$ is the ABC posterior produced after introducing the summary statistics, and $\hat{p}^{\epsilon}_{\text{ABC}}(\theta|x^{\text{obs}})$ is the ABC posterior produced when directly comparing the data sets. Thus, we would in this case not lose any information regarding the posterior. However, computing the sufficient statistics is unfeasible outside the exponential family and we therefor aim at using *informative* summary statistics,

such that the loss of information due to introducing the summary statistics is small in some sense, and we then have that

$$\hat{p}^{\epsilon}_{\text{ABC}}\big(\theta|s^{\text{obs}}\big)\big) \approx \hat{p}^{\epsilon}_{\text{ABC}}(\theta|x^{\text{obs}}). \tag{78}$$

The problem of learning informative summary statistics is discussed in Section 4.2.1.

We will now introduce the ABC rejection-sampling (ABC-RS) algorithm which is outlined in Algorithm 2. The idea of a rejection sampling algorithm is to sample from some target distribution $f(x)$ via some proposal distribution $g(x)$ [Givens and Hoeting, 2012, chapter 6]. Hence, a proposal $x^\star \sim g(x)$ is sampled from the proposal distribution and the proposal is accepted with probability

$$\frac{f(x^\star)}{Mg(x^\star)}, \tag{79}$$

where $Mg(x^\star) \geq f(x^\star), \forall x^\star \in \mathcal{X}$. For ABC-RS the target that we want to sample from is the extended posterior $\hat{p}^{\epsilon}_{\text{ABC}}(\theta, s^\star|s^{\text{obs}})$. However, we do not sample from the extended posterior directly, instead we sample from some proposal distribution $g(\theta, s^\star)$, and for the ABC-RS algorithm presented here we have that $g(\theta, s^\star) = p(s^\star|\theta)p(\theta)$. The acceptance probability of the ABC-RS algorithm is now give by

$$\frac{\hat{p}^{\epsilon}_{\text{ABC}}(\theta, s^\star|s^{\text{obs}})}{Cg(\theta, s^\star)} \propto \frac{K_\epsilon\big(\Delta(s^\star, s^{\text{obs}})\big)p(s^\star|\theta)p(\theta)}{Cp(s^\star|\theta)p(\theta)}, \tag{80}$$

$$= \frac{K_\epsilon\big(\Delta(s^\star, s^{\text{obs}})\big)}{C}. \tag{81}$$

In Equation (81), C is some constant such that $C \geq K_\epsilon(0)$. The ABC-RS algorithm outlined in Algorithm 2 is the same algorithm as discussed Example 4. But with the important difference that the ABC-RS algorithm in Algorithm 2 produces samples according to Equation (72) instead of the true posterior. In this thesis, we show in paper I how probabilistic symmetries of the prior predictive distribution can be used to develop a data-efficient deep-learning model (i.e. the partially exchangeable network model) for learning summary statistics (see Section 4.2.1). However, in paper I, we only consider the ABC-RS algorithms. Thus developing or utilizing more advanced ABC algorithms have not been the focus of this thesis. Nonetheless, some more advanced algorithms, where the idea is to sample proposals in a more efficient and *guided* way, are: ABC-SMC [Toni, 2010, Sisson et al., 2007, Beaumont et al., 2009], and ABC-MCMC [Marjoram et al., 2003].

Algorithm 2 ABC rejection-sampling

Input: Number of draws from the prior predictive distribution N, ABC-threshold ϵ.
Output: Samples from the ABC posterior $\theta^{1:N'}$.

1: **for** $n = 1 : N$ **do**
2: Sample $\theta^n \sim p(\theta)$.
3: Generate $x^n \sim p(x|\theta^n)$.
4: Accept θ^n with probability

$$\frac{K_\epsilon\Big(\Delta\big(S(x^{\text{obs}}), S(x^n)\big)\Big)}{C}, \tag{82}$$

where $C \geq K_\epsilon(0)$.
5: **end for**

4.2.1 Learning summary statistics

As mentioned earlier, selecting which summary statistics to use is one of the main challenges when employing ABC in practice. Learning summary statistics has been an active research area, and some of the proposed methods are reviewed in [Sisson et al., 2018, chapter 5]. Another more modern approach is not to consider the summary statistics and instead match the data sets directly, for instance, by utilizing the Wasserstein distance [Bernton et al., 2019, Drovandi and Frazier, 2021].

An interesting line of research has been to use regression models to learn the summary statistics. This idea is also considered in paper I and we will therefore review this approach here. The idea of utilizing linear regression models for learning summary statistics was introduced by Fearnhead and Prangle [2012], where they showed that the best summary statistics in terms of the quadratic loss is the posterior mean. However, the posterior means is, of course, unknown, but we can learn the posterior mean as a function of the parameter θ via simulations. In Fearnhead and Prangle [2012] they therefore consider the following regression model

$$\theta_j^i = E(\theta_j|x^i) + \epsilon_j^i = \beta_{0_j} + \beta_j h(x^i) + \epsilon_j^i. \tag{83}$$

Here, the index j runs over the dimension of the parameters space, i.e., $j = 1 : \dim \theta$, the index i runs over a set of simulated data sets from $i = 1 : N$, the function $h(x^i)$ is some (non)-linear transformation of the data, and ϵ_j^i is some mean-zero noise. The model in Equation (83) is fitted to data-parameter pairs $(x^i, \theta^i)_{i=1:N}$ obtained from the prior predictive distribution. Once the linear regression model is fitted, the j:th summary statistic for some new simulated data set $x^\star$ is given by

$S(x^\star) = \hat{\beta}_{0_j} + \hat{\beta}_j h(x^\star)$.

The idea of using regression-based models to learn the summary statistics as the posterior mean was further developed in Jiang et al. [2017] where they used a multilayer perceptron (MLP) network to model the regression function; thus they used the following model

$$\theta^i = E(\theta|x^i) + \epsilon^i = f_\phi(x^i) + \epsilon^i. \tag{84}$$

Here, f_ϕ is an MLP network parameterized with weights ϕ, that are estimated by

$$\min_\phi \frac{1}{N} \sum_{i=1}^{N} \|f_\phi(x^i) - \theta^i\|_2^2. \tag{85}$$

Thus, Jiang et al. [2017] utilize the same loss function as in Fearnhead and Prangle [2012]; hence, the only difference of these two approaches is that Jiang et al. [2017] use a more advanced non-linear regression model, i.e. the MLP network, while Fearnhead and Prangle [2012] utilize a linear regression model. Jiang et al. [2017] show that their MLP approach outperforms the linear regression model of Fearnhead and Prangle [2012]; however, the MLP approach requires a much larger computational budget.

In Creel [2017], they notice that MLPs are not particularly good at handling time-depended input data. They, therefore, introduce a pre-processing step such that they feed the network with a set of summary statistics s^i instead of the data x^i. The drawback of this method is that it requires the user to know which summary statistics are reasonable to use as inputs to the network.

Paper I further develops the deep learning-based approach. Paper I, particularly, considers Markovian data and introduces a network architecture, *partially exchangeable networks* (PEN), that is partially exchangeable and, therefore, invariant for Markovian data. In Paper I we show that PEN is much more data-efficient than MLP. The results of paper I show that PEN works well for Markovian data, and a case study also shows that PEN might be useful for quasi-Markovian data.

Convolutional neural networks (CNNs) have also been used for the task of learning summary statistics for ABC [Åkesson et al., 2020]. The advantage of CNNs is that they can utilize the structure of the time-dependent data, and the CNN is expected to work well if the temporal structure is correctly exploited. However, for Markov models with known order of Markovianity, PEN is naturally suited to exploit such features using small computational resources.

While the network approaches indeed have shown to outperform linear regression models, the main drawback is that all network approaches (and linear regression models) require us to simulate a large number of samples N from the prior predictive distribution, fit the network, and then run the ABC inference. However, training the networks can constitute a computationally challenging task. Therefore, it is of interest to consider approaches that directly target the posterior and thereby circumvent the separate summary statistics learning step. Such methods will be considered in Section 4.6 where we will discuss *direct* density estimation methods, which are methods that can directly learn the posterior (or the likelihood) from model simulations.

4.3 Pseudo-marginal Metropolis-Hastings

The Metropolis-Hastings (MH) algorithm [Metropolis et al., 1953, Hastings, 1970] is a popular method to numerically sample from some target distribution $p^\star(x)$. A recent review that covers the MH algorithm's 50 year old history, as well as current developments of the MH algorithm is Dunson and Johndrow [2020]. The idea of the MH algorithm is to construct a Markov chain, where the stationary distribution is the target distribution $p^\star(x)$. Let us assume that the current value of the Markov chain is $x^\#$. The Markov chain is now advanced by first sampling a proposal $x^\star \sim q(x|x^\#)$, from proposal distribution $q(x|x^\#)$ that generates a proposal $x^\star$ based on the currently accepted value $x^\#$. The proposal $x^\star$ is then accepted according to the acceptance probability

$$\alpha = 1 \wedge \frac{p^\star(x^\star)}{p^\star(x^\#)} \cdot \frac{q(x^\#|x^\star)}{q(x^\star|x^\#)}. \tag{86}$$

A noteworthy feature of the MH algorithm is that we do not need to know the normalizing constant of the target distribution $p^\star(x)$ to compute the acceptance probability in Equation (86). Tuning the proposal distribution so that the MH algorithm does not take too small (or too big) steps in the space of $x \in \mathcal{X}$ is of critical importance when running the MH algorithm. To alleviate this problem, several adaptive MH algorithms have been developed [Haario et al., 2001, Andrieu and Thoms, 2008, Vihola, 2012], and these methods try to *tune* the proposal distribution concurrently during one run of the MH algorithm.

We will now introduce the pseudo-marginal Metropolis-Hastings (PMMH) algorithm in the context of Bayesian parametric inference, in the case where we have obtained some observed data set x^{obs}. In this thesis, PMMH is used for Bayesian

parametric inference in both papers II and IV. Thus, let us now consider the case where we want to sample from some posterior $p(\theta|x^{\text{obs}})$ using the MH algorithm. The acceptance probability is then given by

$$\alpha = 1 \wedge \frac{p(x^{\text{obs}}|\theta^\star)}{p(x^{\text{obs}}|\theta^\#)} \cdot \frac{p(\theta^\star)}{p(\theta^\#)} \cdot \frac{q(\theta^\#|\theta^\star)}{q(\theta^\star|\theta^\#)}. \tag{87}$$

In Equation (87), $\theta^\#$ is the current position of the chain and $\theta^\star$ is the proposed parameter. However, the acceptance probability in Equation (87) is only tractable if we can evaluate the likelihood $p(x^{\text{obs}}|\theta)$. The pseudo-marginal Metropolis-Hastings (PMMH) algorithm [Andrieu and Roberts, 2009, Beaumont, 2003] however allows us to ease this restriction, since PMMH allows us to replace the true likelihood $p(x^{\text{obs}}|\theta)$ with an non-negative unbiased stochastic approximation $\hat{p}(x^{\text{obs}}|\theta)$. The fascinating result of [Andrieu and Roberts, 2009] and Beaumont [2003] is that PMMH still targets the true posterior, while utilizing and unbiased stochastic approximation of the target.

Thus, we are in the cases where $p(x^{\text{obs}}|\theta)$ is intractable. However, instead we target the joint distribution $p(x^{\text{obs}}, z|\theta)$ (where $z \sim g(z)$ is some useful auxiliary variable). In this cases it is possible to write $p(x^{\text{obs}}, z|\theta)$ as

$$p(x^{\text{obs}}, z|\theta) = p(x^{\text{obs}}|\theta, z)p(z|\theta) = p(x^{\text{obs}}|\theta, z)g(z). \tag{88}$$

The last step in Equation (88) holds since we assume independence of z and θ. The likelihood $p(x^{\text{obs}}|\theta)$ can be obtained via marginalization

$$p(x^{\text{obs}}|\theta) = \int p(x^{\text{obs}}, z|\theta)dz = \int p(x^{\text{obs}}|\theta, z)g(z)dz = E_z\big(p(x^{\text{obs}}|\theta, z)\big). \tag{89}$$

The stochastic approximation $\hat{p}_z(x^{\text{obs}}|\theta)$ can now be constructed by estimating the expectation $E_z(p(x^{\text{obs}}|\theta, z))$ via Monte Carlo, thus

$$E_z\big(p(x^{\text{obs}}|\theta, z)\big) \approx \frac{1}{P}\sum_{p=1}^{P} p(x^{\text{obs}}|\theta, z^p), \qquad z^p \overset{iid}{\sim} g(z). \tag{90}$$

Furthermore, the Monte Carlo based stochastic approximation $\hat{p}_z(x^{\text{obs}}|\theta)$ of the likelihood can now be defined as

$$\hat{p}_z(x^{\text{obs}}|\theta) := \frac{1}{P}\sum_{p=1}^{P} p(x^{\text{obs}}|\theta, z^p), \qquad z^p \overset{iid}{\sim} g(z). \tag{91}$$

For the stochastic approximation $\hat{p}_z(x^{\text{obs}}|\theta)$ in Equation (91), the auxiliary variable $z = [z^1, z^2, \ldots, z^p]$ is the set of underlying (pseudo-random) numbers used to compute the stochastic approximation. Before continuing our construction of the PMMH method, we notice the similarities between the Monte Carlo estimate in Equation (91) and the likelihood approximation for a state-space model (SSM) in Equation (35). These similarities are not surprising since the likelihood approximation in Equation (35) is a special case of the more general stochastic approximation in Equation (91) obtained when considering SSMs. The unbiasedness of $\hat{p}_z(x^{\text{obs}}|\theta)$ follows from

$$E_z\big(\hat{p}_z(x^{\text{obs}}|\theta)\big) = E_z\big(\frac{1}{P}\sum_{p=1}^{P} p(x^{\text{obs}}|\theta, z^p)\big), \tag{92}$$

$$= \frac{1}{P}\sum_{p=1}^{P} E_z\big(p(x^{\text{obs}}|\theta, z^p)\big), \tag{93}$$

$$= \frac{1}{P}\sum_{p=1}^{P} \int p(x^{\text{obs}}|\theta, z^p) g(z^p) dz^p, \tag{94}$$

$$= p(x^{\text{obs}}|\theta). \tag{95}$$

Equation (95) shows that the Monte Carlo based likelihood estimate $\hat{p}_z(x^{\text{obs}}|\theta)$ indeed is unbiased; hence, this estimate fulfills the condition necessary for PMMH. The next step is to show that the posterior that we obtain when utilizing stochastic likelihood approximation of Equation (91) indeed is the correct posterior. To do that, let us now consider the true posterior, which is given by

$$p(\theta|x^{\text{obs}}) = \frac{p(x^{\text{obs}}|\theta)p(\theta)}{p(x^{\text{obs}})}. \tag{96}$$

Hence, we can write the evidence $p(x^{\text{obs}})$ as

$$p(x^{\text{obs}}) = \frac{p(x^{\text{obs}}|\theta)p(\theta)}{p(\theta|x^{\text{obs}})}. \tag{97}$$

By, utilizing Equation (97) we can now write the extended posterior $p(\theta, z|x^{\text{obs}})$ as

$$p(\theta, z|x^{\text{obs}}) = \frac{p(x^{\text{obs}}|\theta, z)g(z)p(\theta)}{p(x^{\text{obs}})} = \frac{p(\theta|x^{\text{obs}})p(x^{\text{obs}}|\theta, z)g(z)}{p(x^{\text{obs}}|\theta)}. \tag{98}$$

We are now targeting the joint posterior $p(\theta, z|x^{\text{obs}})$. However, obtaining samples of the marginal posterior $p(\theta|x^{\text{obs}})$ via the extended posterior $p(\theta, z|x^{\text{obs}})$ is in practice very easy to do. Since that is achieved when utilizing Metropolis-Hastings to

sampling from $p(\theta, z|x^{\text{obs}})$, by simply disregarding from recording the z's. We have that the marginalized posterior targets the true posterior since

$$\int p(\theta, z|x^{\text{obs}})dz = \int \frac{p(\theta|x^{\text{obs}})p(x^{\text{obs}}|\theta, z)g(z)}{p(x^{\text{obs}}|\theta)}dz, \tag{99}$$

$$= \frac{p(\theta|x^{\text{obs}})}{p(x^{\text{obs}}|\theta)} \int p(x^{\text{obs}}|\theta, z)g(z)dz, \tag{100}$$

$$= \frac{p(\theta|x^{\text{obs}})}{p(x^{\text{obs}}|\theta)} p(x^{\text{obs}}|\theta), \tag{101}$$

$$= p(\theta|x^{\text{obs}}). \tag{102}$$

In conclusion, instead of directly targeting $p(\theta|x^{\text{obs}})$ PMMH allows us to target the extended posterior $p(\theta, z|x^{\text{obs}})$, where z is some suitable auxiliary variables that allows us to construct and unbiased stochastic approximation of the intractable likelihood. However, PMMH will still produce *exact* Bayesian inference, regardless of choice of P, since the marginal posterior of $p(\theta, z|x^{\text{obs}})$ will target the true posterior. The PMMH algorithm is presented in Algorithm 3, notice in lines 7 and 8 that we only compute the likelihood estimate for the proposal $\theta^\star$.

The method outlined above can, for instance, been used for inference in state-space models (SSMs). Indeed, PMMH has become particularly popular for SSMs since, for SSMs, an non-negative unbiased approximation of the likelihood can be computed using particle filters (i.e., sequential Monte Carlo) [Andrieu et al., 2010]. As mentiond in Section 4.1, the bootstrap particle filter is the most straightforward method to estimate the likelihood of general SMMs. The bootstrap filter is particularly useful for this task since it has the key feature that it can produce a non-negative unbiased approximation of the likelihood [Naesseth et al., 2019, chapter 4.1]. Another advantage when using the bootstrap filter is that PMMH is then a *plug-and-play method* since only forward simulations of the model are used. A non-trivial problem when running the PMMH algorithm is to decide on how many particles to use for the likelihood approximation. While PMMH is theoretically valid for any number of particles P, it is the case that PMMH will be very inefficient (i.e., produce a chain that mixes very slowly or get stuck) if the variance of the likelihood estimation is too high. Furthermore, the variance of the likelihood estimation is inversely proportional to the number of particles used. Doucet et al. [2015] and Sherlock et al. [2015] suggest tuning the number of particles such that the variance of log-likelihood approximation at some central posterior parameter value is around 2. However, this goal can be hard to obtain for challenging SSMs when using the bootstrap filter since an unfeasibly large number of particles then has to be used. For challenging SMMs,

Algorithm 3 Pseudo-marginal Metropolis-Hastings (PMMH)

Input: Number of iterations R, start value θ^0.
Output: Samples $\theta^{1:R}$ from the posterior distribution $p(\theta|x^{\text{obs}})$.

1: Set $\theta^1 = \theta^0$. ▷ First iteration
2: Sample $z^1 \sim g(z)$.
3: Compute $\hat{p}_{z^1}(x^{\text{obs}}|\theta^1)$.
4: **for** $r = 2 : R$ **do** ▷ Iterations $2 : R$
5: Sample $\theta^\star \sim q(\theta|\theta^{r-1})$.
6: Sample $z^\star \sim g(z)$.
7: Compute $\hat{p}_{z^\star}(x^{\text{obs}}|\theta^\star)$.
8: Compute

$$\alpha = 1 \wedge \frac{\hat{p}_{z^\star}(x^{\text{obs}}|\theta^\star)}{\hat{p}_{z^{r-1}}(x^{\text{obs}}|\theta^{r-1})} \cdot \frac{p(\theta^\star)}{p(\theta^{r-1})} \cdot \frac{q(\theta^{r-1}|\theta^\star)}{q(\theta^\star|\theta^{r-1})}. \tag{103}$$

9: Sample $u \sim \mathcal{U}(0,1)$.
10: **if** $u < \alpha$ **then**
11: Set $\theta^r = \theta^\star, z^r = z^\star$.
12: **else**
13: Set $\theta^r = \theta^{r-1}, z^r = z^{r-1}$.
14: **end if**
15: **end for**

several other methods can be employed; it is sometimes possible to employ a more efficient particle filter (e.g see Pitt and Shephard [1999], and Golightly and Wilkinson [2011]). The Monte Carlo within Metropolis (MCWM) algorithm [Beaumont, 2003, Medina-Aguayo et al., 2016, Andrieu and Roberts, 2009] can also alternatively be utilized; at each iteration, MCWM computes two likelihood approximations, one for the likelihood in the numerator and one for likelihood in the denominator, and this procedure reduces the need for likelihood estimate with a low variance; however, MCWM does not produce exact Bayesian inference; and, due to this, using MCWM might not be of interest. Another alternative is to consider the correlated pseudo-marginal method, which is introduced in the next section and is used in paper II.

4.3.1 Correlated pseudo-marginal Metropolis-Hastings

The likelihood ratio of PMMH consists of the two likelihood approximations $\hat{p}_{z^\star}(x^{\text{obs}}|\theta^\star)$ and $\hat{p}_{z^\#}(x^{\text{obs}}|\theta^\#)$. The two approximations are computed using some auxiliary vari-

ables $z^\star$ and $z^\#$, which are the pseudo-random numbers used for the approximations. From line 8 of Algorithm 3, we see that the likelihood ratio of PMMH is given by

$$\frac{\hat{p}_{z^\star}(x^{\text{obs}}|\theta^\star)}{\hat{p}_{z^\#}(x^{\text{obs}}|\theta^\#)}. \tag{104}$$

The idea of the correlated pseudo-marginal (CPMMH) method [Deligiannidis et al., 2018, Dahlin et al., 2015] is to correlate the auxiliary variables $z^\star$ and $z^\#$ such that the corresponding likelihood approximations $\hat{p}_{z^\star}(x^{\text{obs}}|\theta^\star)$ and $\hat{p}_{z^\#}(x^{\text{obs}}|\theta^\#)$ also are correlated. When the two approximations in the likelihood ratio are correlated the variance of the likelihood ratio is reduced, allowing us to use fewer particles for each approximation and still obtaining an algorithm that samples from the posterior distribution efficiently. Hence, for PMMH we have to control the variance of the likelihood approximation $\hat{p}_z(x^{\text{obs}}|\theta)$, but for CPMMH, we only need to control the variance of the likelihood ratio.

For PMMH we have that the auxiliary variables are generated from

$$z^\star \sim g(z). \tag{105}$$

However, CPMMH generalizes PMMH and generates $z^\star$ from $K(z^\star|z^\#)$ where $K(\cdot|\cdot)$ fulfills the detailed balance equation

$$g(z^\#)K(z^\star|z^\#) = K(z^\#|z^\star)g(z^\star). \tag{106}$$

In paper II the correlation is induced by using a Crank-Nicolson proposal [Deligiannidis et al., 2018], for which,

$$g(z^\#) = \mathcal{N}(z^\#|\overrightarrow{0}, \mathbb{I}), \qquad K(z^\star|z^\#) = \mathcal{N}(z^\star|\rho z^\#, (1-\rho^2)\mathbb{I}). \tag{107}$$

In Equation (107), ρ is the correlation level. The correlation can also be induced by using block-correlation [Tran et al., 2016]. The Crank-Nicolson proposal method's main advantage is that the statistician can explicitly control the correlation level via the parameter ρ (a standard choice is to select $\rho \approx 0.9$). The correlation level of the block-correlation method, on the other hand, depends on which problem we consider. To retain the correlation during the particle filter, it is necessary to add a sorting step of the particles in the particle filter [Deligiannidis et al., 2018, Dahlin et al., 2015] (see line 6 of the bootstrap filter in Algorithm 1). A drawback of introducing the correlation is that the auxiliary variables have to be stored, and the memory-complexity of the CPMMH algorithm will, therefore, increase compared to PMMH; however, it is typically still advantageous to use CPMMH since the correlation allows us to

use much fewer particles for CPMMH compared to PMMH. The sorting step in the particle filter also introduces an additional computational cost. Similarly to PMMH, it is not trivial to select how many particles to use for CPMMH, but one method for particle selection is presented in Tran et al. [2016]. Both PMMH and CPMMH produce *exact* Bayesian inference.

4.4 Bayesian inference for stochastic differential equations mixed-effects models

We introduced the stochastic differential equations mixed-effects model (SDEMEM) in Section 3.4, and we will now discuss Bayesian inference for SDEMEMs. Bayesian inference for SDEMEMs is complicated since the likelihood for the data is intractable. In paper II, we use sequential Monte Carlo methods in combination with correlated pseudo-marginal Metropolis-Hastings to develop an efficient inference algorithm. We will now briefly discuss the technical details of the inference procedure developed in paper II. The joint posterior for the parameter $\theta = [\eta, \xi]$, random effects $\phi = \phi^{1:M}$ and latent data $x = (x^i_{1:N_i})_{i=1}^M$ (here x^i_j is the j:th observation, at time-point t^i_j, of the i:th individual, and N_i is the number of observations for individual i), given observed data $y = (y^i_{1:N_i})_{i=1}^M$, is

$$p(\eta, \xi, \phi, x|y) \propto p(\eta)p(\xi)p(\phi|\eta)p(x|\phi)p(y|x, \xi), \tag{108}$$

where,

$$p(\phi|\eta) = \prod_{i=1}^{M} p(\phi^i|\eta), \tag{109}$$

$$p(x|\phi) = \prod_{i=1}^{M} p(x^i_1) \prod_{j=2}^{N_i} p(x^i_j|x^i_{j-1}, \phi^i), \tag{110}$$

$$p(y|x, \xi) = \prod_{i=1}^{M} \prod_{j=1}^{N_i} p(y^i_j|x^i_j, \xi). \tag{111}$$

However we are not interested in obtaining the joint posterior of Equation (108), rather we want to obtain the the marginal posterior over the parameter $\theta = [\eta, \xi]$ and random effects $\phi = \phi^{1:M}$. This marginal posterior is given by

$$p(\eta, \xi, \phi|y) \propto p(\eta)p(\xi)p(\phi|\eta)p(y|\xi, \phi) \tag{112}$$

$$\propto p(\eta)p(\xi) \prod_{i=1}^{M} p(\phi^i|\eta)p(y^i|\xi, \phi^i). \tag{113}$$

In Equation (140), $p(y^i|\xi, \phi^i)$ is the *data likelihood* for individual i. The data likelihood is obtained by

$$p(y^i|\xi, \phi^i) = \int p(y^i|x^i, \xi)p(x^i|\phi^i)dx^i \tag{114}$$

$$= \int \prod_{j=1}^{N_i} p(y_j^i|x_j^i, \xi)p(x_1^i) \prod_{j=2}^{N_i} p(x_j^i|x_{j-1}^i, \phi^i)dx_1^i dx_2^i \dots dx_{N_i}^i \tag{115}$$

However, the integral in Equation (114) is intractable, and in paper II we, therefore, estimate this integral with sequential Monte Carlo (SMC). The intractable data likelihood also motivates the use of pseudo-marginal Metropolis-Hastings in paper II. (The technical details on the general SMC likelihood estimation procedure are presented in Section 4.1.1. Similarly, the technical details of the pseudo-marginal Metropolis-Hastings method are discussed in Section 4.3.)

The following Gibbs algorithm is used in paper II to sample from the marginal posterior of interest:

1. $p(\phi^i, z^i|\eta, \xi, y) \propto p(\phi^i|\eta)\hat{p}_{z_i}(y^i|\xi, \phi^i)g(z^i), \qquad i = 1 : M,$
2. $p(\xi, z|\eta, \phi, y) = p(\xi|\phi, y) \propto p(\xi) \prod_{i=1}^M \hat{p}_{z_i}(y^i|\xi, \phi^i)g(z^i),$
3. $p(\eta|\xi, \phi, y) = p(\eta|\phi) \propto p(\eta) \prod_{i=1}^M p(\phi^i|\eta).$

In the Gibbs algorithm above, $z^i \sim g(z^i)$ are the underlying pseudo-random numbers used to compute the SMC-based approximation of the data likelihood $p_{z_i}(y^i|\xi, \phi^i)$. We notice that the data likelihood must be estimated in both steps 1 and 2 of the Gibbs algorithm. However, step 2 is particularly problematic since we here have to control the variance of the joint data likelihood, which is more computational challenging than controlling the variance of each individual data likelihoods. In paper II, the Gibbs algorithm above is made computationally efficient by leveraging correlated pseudo-marginal methods and a carefully constructed blocking strategy that allows us to update the underlying pseudo-random numbers z only once.

The inference method of paper II outlined above is flexible since it allows for inference for non-linear and non-Gaussian SDEMEMs. In paper II we obtain about one order of magnitude efficiency improvement by leveraging correlated pseudo-marginal method. The method of paper II has been further developed in Persson et al. [2021] (i.e., paper *A*). Persson et al. [2021] considers the task of inference in stochastic dynamical single-cell models. Persson et al. [2021] also shows how the framework of

paper II can be extended to other latent processes. Another recent paper on Bayesian inference for SDEMEM by Botha et al. [2021] was developed independently and concurrently of paper II. Botha et al. [2021] presents several inference methods, including one that is similar to the Gibbs approach outlined above.

4.5 Delayed-acceptance

The *delayed-acceptance* (DA) method [Christen and Fox, 2005] accelerates a MCMC algorithm by utilizing a cheap surrogate model to quickly scan proposals $\theta^\star$, and thereby efficiently reject proposals that are not promising (assuming the surrogate model is sufficiently accurate). The DA method is particular useful in cases where the likelihood function is computationally expensive to evaluate. DA can for these cases be used to minimize the number of likelihood evaluations by only evaluating the likelihood for promising proposals. We will here introduce the DA method in the context of Bayesian parametric inference, in the case where we have obtained some observed data set x^{obs}. So, let us again consider the acceptance probability of a standard MH step

$$\alpha = 1 \wedge \frac{p(x^{\text{obs}}|\theta^\star)}{p(x^{\text{obs}}|\theta^\#)} \cdot \frac{p(\theta^\star)}{p(\theta^\#)} \cdot \frac{q(\theta^\#|\theta^\star)}{q(\theta^\star|\theta^\#)}. \tag{116}$$

Now, assume that we have access to a computationally cheap surrogate model of the likelihood $\tilde{p}(x^{\text{obs}}|\theta)$. The DA method now utilizes two acceptance steps, denoted step 1 and step 2. The acceptance probability of step 1 is given by

$$\alpha_1 = 1 \wedge \frac{\tilde{p}(x^{\text{obs}}|\theta^\star)}{\tilde{p}(x^{\text{obs}}|\theta^\#)} \cdot \frac{p(\theta^\star)}{p(\theta^\#)} \cdot \frac{q(\theta^\#|\theta^\star)}{q(\theta^\star|\theta^\#)}. \tag{117}$$

This acceptance probability is computed for every new proposal $\theta^\star$, but since the surrogate model $\tilde{p}(x^{\text{obs}}|\theta)$ is computationally cheap is step 1 of the DA method also fast to execute. If the proposal $\theta^\star$ *survives* step 1 (i.e. it is not rejected) we then compute the acceptance probability of step 2, given by

$$\alpha_2 = 1 \wedge \frac{p(x^{\text{obs}}|\theta^\star)}{p(x^{\text{obs}}|\theta^\#)} \cdot \frac{\tilde{p}(x^{\text{obs}}|\theta^\#)}{\tilde{p}(x^{\text{obs}}|\theta^\star)}. \tag{118}$$

Thus, the true computationally challenging likelihood $p(x^{\text{obs}}|\theta)$ is only evaluated in step 2, i.e. if the proposal $\theta^\star$ passes the screening of step 1. From the set-up with the two steps we notice that the surrogate posterior model $\tilde{p}(x^{\text{obs}}|\theta)p(\theta)$ acts as an *proposal distribution* for step 2 of the DA method. Finally, the proposal $\theta^\star$ can only be

accepted after evaluating the true likelihood model. Christen and Fox [2005] show that the DA method outlined above indeed targets the true posterior. Thus, DA is a method that, similarly to CPMMH, allows for accelerated exact Bayesian inference.

4.5.1 Accelerated-delayed acceptance

Paper IV introduces the *accelerated-delayed acceptance* (ADA) algorithm. ADA further accelerates the DA method by utilizing relationships of the likelihood ratio of surrogate model

$$\frac{\tilde{p}(x^{\text{obs}}|\theta^{\star})}{\tilde{p}(x^{\text{obs}}|\theta^{\#})}, \tag{119}$$

and the likelihood ratio of the true model

$$\frac{p(x^{\text{obs}}|\theta^{\star})}{p(x^{\text{obs}}|\theta^{\#})}. \tag{120}$$

The main idea of ADA is to arrange the computations is step 2 of the DA method to obtain an acceleration. However, step 1 of DA and ADA is the same. At iteration r of the DA method, the two acceptance probabilities are governed by the following surrogate and true likelihood values $\tilde{p}(x^{\text{obs}}|\theta^{\star})$, $\tilde{p}(x^{\text{obs}}|\theta^{r-1})$, $p(x^{\text{obs}}|\theta^{\star})$, and $p(x^{\text{obs}}|\theta^{r-1})$, which can be arranged in the following four mutually exclusive cases:

$$\text{case 1) } \tilde{p}(x^{\text{obs}}|\theta^{\star}) > \tilde{p}(x^{\text{obs}}|\theta^{r-1}) \textit{ and } p(x^{\text{obs}}|\theta^{\star}) > p(x^{\text{obs}}|\theta^{r-1}), \tag{121}$$

$$\text{case 2) } \tilde{p}(x^{\text{obs}}|\theta^{\star}) < \tilde{p}(x^{\text{obs}}|\theta^{r-1}) \textit{ and } p(x^{\text{obs}}|\theta^{\star}) < p(x^{\text{obs}}|\theta^{r-1}), \tag{122}$$

$$\text{case 3) } \tilde{p}(x^{\text{obs}}|\theta^{\star}) > \tilde{p}(x^{\text{obs}}|\theta^{r-1}) \textit{ and } p(x^{\text{obs}}|\theta^{\star}) < p(x^{\text{obs}}|\theta^{r-1}), \tag{123}$$

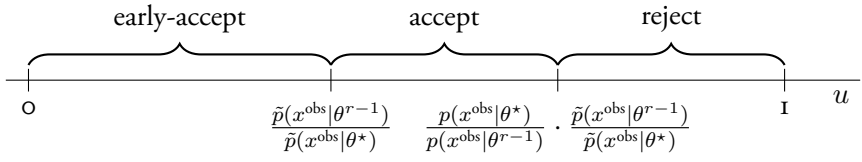
$$\text{case 4) } \tilde{p}(x^{\text{obs}}|\theta^{\star}) < \tilde{p}(x^{\text{obs}}|\theta^{r-1}) \textit{ and } p(x^{\text{obs}}|\theta^{\star}) > p(x^{\text{obs}}|\theta^{r-1}). \tag{124}$$

Now, if we *know* which case iteration r belongs to careful arrangement of the step 2 of the DA method can lead to an acceleration. Now, as an illustration, let us consider case 1 and investigate how an acceleration could be obtained for that case:

Case 1) If we have that $\tilde{p}(x^{\text{obs}}|\theta^{\star}) > \tilde{p}(x^{\text{obs}}|\theta^{r-1})$ *and* $p(x^{\text{obs}}|\theta^{\star}) > p(x^{\text{obs}}|\theta^{r-1})$, then it also holds that $\frac{\tilde{p}(x^{\text{obs}}|\theta^{r-1})}{\tilde{p}(x^{\text{obs}}|\theta^{\star})} < 1$ *and* $\frac{p(x^{\text{obs}}|\theta^{r-1})}{p(x^{\text{obs}}|\theta^{\star})} < 1$. Thus, we also have that

$$\frac{\tilde{p}(x^{\text{obs}}|\theta^{r-1})}{\tilde{p}(x^{\text{obs}}|\theta^{\star})} < \frac{p(x^{\text{obs}}|\theta^{\star})}{p(x^{\text{obs}}|\theta^{r-1})} \cdot \frac{\tilde{p}(x^{\text{obs}}|\theta^{r-1})}{\tilde{p}(x^{\text{obs}}|\theta^{\star})}. \tag{125}$$

From Equation (125) we can conclude that the acceptance region of the second stage can be split into two parts, where one part is only governed by $\frac{\tilde{p}(x^{\text{obs}}|\theta^{r-1})}{\tilde{p}(x^{\text{obs}}|\theta^{\star})}$. In other words, in case 1, the acceptance region is $[0, \frac{p(x^{\text{obs}}|\theta^{\star})}{p(x^{\text{obs}}|\theta^{r-1})} \cdot \frac{\tilde{p}(x^{\text{obs}}|\theta^{r-1})}{\tilde{p}(x^{\text{obs}}|\theta^{\star})}]$. However, Equation (125) allows us to decompose this region into a sub-region that only depends on $\frac{\tilde{p}(x^{\text{obs}}|\theta^{r-1})}{\tilde{p}(x^{\text{obs}}|\theta^{\star})}$, see the illustration below:



Thus, if a proposal $\theta^{\star}$ survives step 1 of ADA, then, in step 2 we fist check if we can *early-accept*, by checking if

$$u < \frac{\tilde{p}(x^{\text{obs}}|\theta^{r-1})}{\tilde{p}(x^{\text{obs}}|\theta^{\star})}, \;\; u \sim \mathcal{U}(0,1). \tag{126}$$

If that is not possible we compute the full acceptance probability given by

$$1 \wedge \frac{p(x^{\text{obs}}|\theta^{\star})}{p(x^{\text{obs}}|\theta^{r-1})} \cdot \frac{\tilde{p}(x^{\text{obs}}|\theta^{r-1})}{\tilde{p}(x^{\text{obs}}|\theta^{\star})}. \tag{127}$$

Hence, in case 1 an acceleration can be obtained by early-accepting a proposal in stage 2 of ADA only based on Equation (126).

In paper IV, we analyze the other cases in a similar way. Thus, ADA allows for acceleration by carefully arranging the computations of step 2 and only computing the likelihood ratio of the true model if early-acceptance (or early-rejection) is not possible.

However, for ADA, an important component is to know *which* case a particular proposal belongs to, *without* computing the true likelihood ratio. In paper IV two approaches are used to classify proposals. One approach is to compute the relative frequency of the four different cases during a pilot run and then use these relative frequencies to select which case to consider for a particular proposal. Another approach is to fit a tree-based classification scheme to data generated from a pilot run. These two approaches and their advantages/disadvantages are discussed in paper IV.

Another important design choice for ADA is how to construct the surrogate likelihood model. For paper IV the surrogate likelihood model is based on a Gaussian

process model that is inspired by the surrogate model utilized in Drovandi et al. [2018].

The ADA method does not target the true posterior since a proposal can be early-accepted without considering the true likelihood ratio. However, ADA allows for accelerated inference since ADA can both reject and accept proposals only based on the surrogate model. The results of paper IV show that the ADA method indeed manages to produce accelerated inference for cases where the likelihood function is time-consuming to evaluate. The inference results of ADA are similar to the results obtained by methods that produce exact Bayesian inference.

4.6 Neural density estimation

The conditional neural density estimation (CNDE) model was introduced in Section 4.1.3. We will here introduce two types of simulation-based inference algorithms that utilize CNDEs: Section 4.6.1 introduces the *sequential neural posterior estimation* SNPE method, and Section 4.6.2 introduces the *sequential neural likelihood estimation* SNL method. We will also in Section 4.6.3 discuss the *sequential neural posterior and likelihood approximation* method of paper III. SNPE and SNL are introduced in full detail since these methods are functional to develop the novel SNPLA method of paper III.

4.6.1 Sequential neural posterior estimation

One of the earlier works employing CNDEs for simulation-based inference approach was Papamakarios and Murray [2016]. They develop a sequential simulation-based inference method, where the idea is to directly learn the posterior distribution. The overall method of Papamakarios and Murray [2016] is denoted *sequential neural posterior estimation* SNPE, and the specific method that they introduce is denoted SNPE-A. The overall idea of SNPE is to fit the CNDE model $\tilde{p}_{\phi_P}(\theta|x^{\text{obs}})$, which directly targets the posterior. The model $\tilde{p}_{\phi_P}(\theta|x)$ can be fitted with samples directly from prior predictive distribution, however, this approach can be very inefficient and SNPE therefore generates training data via a *proposal prior* $\hat{p}_r(\theta|x^{\text{obs}})$ (here $\hat{p}_r(\theta|x^{\text{obs}})$ is the proposal prior at iteration r of the SNPE scheme). The proposal prior $\hat{p}_r(\theta|x^{\text{obs}})$ is set to sequentially approximate the posterior, and the proposal prior, thereby, sequentially becomes more informative of the posterior, rendering a

more efficient sampling scheme. This is obtained by updating the proposal prior as,

$$\hat{p}_r(\theta|x^{\text{obs}}) \leftarrow \frac{p(\theta)}{\hat{p}_{r-1}(\theta|x^{\text{obs}})}\tilde{p}_{\phi_P}(\theta|x^{\text{obs}}). \tag{128}$$

However, the updating step in Equation (128), (sometimes also refereed to as the *correction step*), is somewhat involved since the new proposal prior has to be updated such that it has the correct normalizing constant. The entire SNPE method is presented in Algorithm 4. In Papamakarios and Murray [2016] the approximate posterior $\tilde{p}_{\phi_P}(\theta|x)$ is modeled with a Mixture Density Network (MDN); thus the posterior is approximated with Gaussian mixture distribution. This makes it possible to analytical compute the correction step in Equation (128). An interesting feature of SNPE is that the posterior inference is not dependent on some approximation threshold ϵ, which is the case for ABC. However, a drawback of the specific method of Papamakarios and Murray [2016], i.e., SNPE-A, is that SNPE-A will only work well if the MDN indeed is a reasonable approximation of the posterior, which for posteriors with complex geometries might not be the case.

A paper that directly builds upon Papamakarios and Murray [2016]) is Lueckmann et al. [2017]. In Lueckmann et al. [2017] they also consider a SNPA approach using MDN's to model the posterior. Furthermore, Lueckmann et al. [2017] removes the correction step employed in Papamakarios and Murray [2016] by instead importance weighting the loss function of the posterior model. The importance-weighting scheme allows for a more flexible method, not restricting the proposal prior $\hat{p}_r(\theta|x^{\text{obs}})$ to be a Gaussian mixture. However, the importance-weight scheme can introduce some numerical problems in the optimization scheme [Papamakarios et al., 2019b].

Another SNPE scheme, denoted *automatic posterior transformation* (APT), or SNPE-C (according to the taxonomy of Papamakarios et al. [2019b]) is presented in Greenberg et al. [2019]. Greenberg et al. [2019] avoid the correction step of Papamakarios and Murray [2016] by including the normalizing constant into the loss function via reparameterization. The main drawback of the ATP method is that the integral that computes the normalizing constant is intractable in many cases; however, they employ a scheme called *atomic proposals* that allows them to replace the integral that computes the normalizing constant of Equation (128) with a sum. The scheme in Greenberg et al. [2019] is more generic than the other SNPE methods, and they show that ATP performs well for several experiments, including the two moons model discussed in Section 3.1.

Algorithm 4 SNPE

Input: Untrained posterior model $\tilde{p}_{\phi_P}(\theta|x)$, number of iterations R, number of training samples per iteration N.

Output: Trained posterior model $\tilde{p}_{\phi_P}(\theta|x)$.

1: Set $\hat{p}_0(\theta|x^{\text{obs}}) \leftarrow p(\theta)$, $\mathcal{D} = \{\emptyset\}$

2: **for** $r = 1 : R$ **do**

3: **for** $n = 1 : N$ **do** ▷ Sample training data

$$(\theta^n, x^n) \sim \tilde{p}(\theta, x) = p(x|\theta)\hat{p}_{r-1}(\theta|x^{\text{obs}}). \tag{129}$$

4: **end for**

5: Update training data $\mathcal{D} = [\theta^{1:N}, x^{1:N}] \cup \mathcal{D}$.

6: Update $\tilde{p}_{\phi_P}(\theta|x)$ by minimizing the following loss

$$\mathcal{L}(\phi_P) = -E_{\tilde{p}(\theta,x)}\big(\log \tilde{p}_{\phi_P}(\theta|x)\big). \tag{130}$$

7: Update the proposal distribution, i.e. let

$$\hat{p}_r(\theta|x^{\text{obs}}) \leftarrow \frac{p(\theta)}{\hat{p}_{r-1}(\theta|x^{\text{obs}})}\tilde{p}_{\phi_P}(\theta|x^{\text{obs}}). \tag{131}$$

8: **end for**

4.6.2 Sequential neural likelihood estimation

In Papamakarios et al. [2019b] they let the CNDE model $\tilde{p}_{\phi_L}(x|\theta)$ target the likelihood $p(x|\theta)$ instead of directly targeting the posterior. The method of Papamakarios et al. [2019b] is, therefore, denoted *sequential neural likelihood* (SNL). They use a normalizing flow model to parameterize the likelihood model $\tilde{p}_{\phi_L}(x|\theta)$. Similarly to SNPE, SNL also uses the idea of a proposal prior $\hat{p}_r(\theta|x^{\text{obs}})$ that is sequentially updated to become more informative of the posterior. This is achieved by setting, for each iteration, the new proposal prior to the current posterior approximation. The SNL method is presented in Algorithm 5.

A drawback of SNL is that SNL only implicitly learns the posterior, and sampling from the learned posterior distribution can therefore be computationally expensive. Instead, SNL learns the likelihood model $\tilde{p}_{\phi_L}(x^{\text{obs}}|\theta)$ so samples from the posterior distribution can be obtained via MCMC. However, since SNL relies on MCMC sampling, it can have problems in efficiently learning posterior distributions where MCMC algorithms typically struggle, for instance, with multi-modal posteriors. This issue with SNL is observed in both Greenberg et al. [2019] and paper III.

Algorithm 5 SNL

Input: Untrained likelihood model $\tilde{p}_{\phi_L}(x|\theta)$, number of iterations R, number of training samples per iteration N.

Output: Trained likelihood model $\tilde{p}_{\phi_L}(x|\theta)$.

1: Set $\hat{p}_0(\theta|x^{\text{obs}}) \leftarrow p(\theta)$, $\mathcal{D} = \{\emptyset\}$

2: **for** $r = 1 : R$ **do**

3: **for** $n = 1 : N$ **do** ▷ Sample training data (via MCMC)

$$(\theta^n, x^n) \sim \tilde{p}(\theta, x) = p(x|\theta)\hat{p}_{r-1}(\theta|x^{\text{obs}}). \tag{132}$$

4: **end for**

5: Update training data $\mathcal{D} = [\theta^{1:N}, x^{1:N}] \cup \mathcal{D}$.

6: Update $\tilde{p}_{\phi_L}(x|\theta)$ by minimizing the following loss

$$\mathcal{L}(\phi_L) = -E_{\tilde{p}(\theta,x)}\big(\log \tilde{p}_{\phi_L}(x|\theta)\big). \tag{133}$$

7: Update the proposal distribution, i.e. let

$$\hat{p}_r(\theta|x^{\text{obs}}) \propto \tilde{p}_{\phi_L}(x^{\text{obs}}|\theta)p(\theta). \tag{134}$$

8: **end for**

Lueckmann et al. [2019] introduces another SNL approach denoted *active sequential neural likelihood* (ASNL). The idea of ASNL is to model the likelihood using emulator network. In Lueckmann et al. [2019] are active learning strategies employed to generate the new training data for the emulator $p_\phi(x|\theta)$.

By considering Algorithms 4 and 5 can we conclude that SNL and SNPE are quite similar methods. Both methods utilize a proposal prior $\hat{p}_r(\theta|x^{\text{obs}})$ that is informed of the observed data set x^{obs} and set to make the methods efficient by approximating the posterior distribution. The main difference is that SNPE directly targets the posterior, while SNL targets the likelihood and thus only implicitly learns the posterior. Thus a question that now arise is if it is more advantageous to learn the posterior or the likelihood. The choice between learning the posterior or the likelihood is discussed in [Durkan et al., 2018, Lueckmann et al., 2017] and it is suggested that learning the likelihood might be the preferable task, since learning the posterior introduces restrictions on how new training data can be generated. The approximated likelihood can also easily be reused for new prior distributions; finally, general-purpose CNDE techniques might also make it easier to learn the likelihood. However, a drawback of SNL is that once we have learned the approximate likelihood $\tilde{p}_{\phi_L}(x^{\text{obs}}|\theta)$ we still have to sample from the posterior using MCMC (or using variational inference to

approximate the posterior), which can constitute a challenging task for complex and multiimodal posterior distributions [Greenberg et al., 2019].

4.6.3 Sequential neural posterior and likelihood approximation

We will now briefly discuss the *sequential neural posterior and likelihood approximation* algorithm of paper III. SNPLA is a sequential and nested simulation-based inference method inspired by both SNPE and SNL. However, SNPLA learns both the posterior and the likelihood model. Thus SNPLA has two learnable models:

1. **Posterior model** $\tilde{p}_{\phi_P}(\theta|x^{\text{obs}})$, approximating the posterior distribution $p(\theta|x^{\text{obs}})$.
2. **Likelihood model** $\tilde{p}_{\phi_L}(x|\theta)$. Since we are considering an implicit statistical model, we consider the likelihood model $\tilde{p}_{\phi_L}(x|\theta)$ as approximating the data generating process $p(x|\theta)$.

Similar to SNPE and SNL, SNPLA utilizes a proposal prior that is set to leverage more information from the posterior of interest. However, SNPLA is a more involved method since SNPLA learns both the posterior and the likelihood, thus requiring a more complex construction than SNPE and SNL. The SNPLA method is introduced in full detail in paper III.

The SNPLA algorithm is presented in Algorithm 6. In step 1 of SNPLA, the likelihood model is learned via the forward KL divergence. Thus, the learning process for the likelihood model used in SNPLA is quite straightforward and similar to the SNL method. Training data for the likelihood model is generated via the proposal distribution $\hat{p}_r(\theta|x_{\text{obs}})$ which is set to leverage more information from the posterior model sequentially. The parameter λ governs how quickly we want to leverage information from the posterior model, and we have found it useful to use a quite high $\lambda \approx 0.7 - 0.9$. The posterior model is trained in step 2 of SNPLA. However, in SNPLA the learning process for the posterior model is more complex. First, training data for the posterior model is generated via a *simulation-on-the-fly* approach from the *current* posterior model. This means that each mini-batch of size N_{mini} used in step 2 is simulated from the most recent version of the posterior model. This procedure also utilizes the reverse KL divergence to train the posterior model. Second, the posterior model is not trained according to its true target distribution; the likelihood model $\tilde{p}_{\phi_L}(x|\theta)$ is instead used to approximate the target for the posterior model. However, several approaches designed to alleviate the complex learning process of the posterior model are discussed in paper III. Two approaches are: 1) the hot-start

procedure in line 8 of Algorithm 6, and 2) utilizing a larger batch size N_{mini} in step 2 of SNPLA. The hot-start procedure is used during the first iteration of SNPLA to train the posterior model via samples from the prior predictive distribution, such that the posterior model is then trained according to its true target distribution. The large batch size $N_{\text{mini}} \approx 1000$ of step 2 smooths the learning process and thus avoids catastrophic moves in the weight space of the posterior model.

The results of paper III show that SNPLA performs well compared to SNPE-C and SNL. For instance, posterior inference results obtained via SNPLA are at par with SNPE-C and SNL, despite the fact that SNPLA learns to properties instead of one. The likelihood model learned via SNPLA also performs as well as the likelihood model learned by SNL. Hence, SNPLA produces posterior inference at par with similar methods and can therefore be used if the researcher is interested in both learning the posterior distribution and a proxy of the generative model, i.e. the likelihood.

4.7 Additional methods

Additional methods utilized in paper III for comparison peruses are SMC-ABC [Beaumont et al., 2009] and *Sequential Contrastive Likelihood-free Inference* (SCLFI) [Durkan et al., 2020]. SCLFI is a ratio-based learning approach that combines the methods of Greenberg et al. [2019] and Hermans et al. [2020]. Ratio-based approaches are not considered in this thesis, but these approaches turn the posterior estimation problem into a problem of estimating likelihood ratios.

Furthermore, simulation-based inference is an active research area, and some other relatively recent simulation-based methods that are particularly relevant for Bayesian inference are:

- *Indirect inference* [Gourieroux et al., 1993] and *Method of simulated moments* [McFadden, 1989] are simulation-based inference methods typically used for applications in economy.
- *Synthetic likelihood* (SL) [Wood, 2010] approximates the intractable likelihood with a Gaussian synthetic version learnt from model simulations. *Bayesian synthetic likelihood* Price et al. [2018] is a Bayesian version of the SL method.
- *Likelihood-free Inference by Ratio Estimation* [Thomas et al., 2020] is a ratio-based method.

Algorithm 6 SNPLA

Input: Untrained likelihood model $\tilde{p}_{\phi_L}(x|\theta)$, untrained posterior model $\tilde{p}_{\phi_P}(\theta|x)$, number of iterations R, number of training samples per iteration N, number of training samples per iteration for the posterior model N_P, decay rate $\lambda > 0$.

Output: Trained likelihood model $\tilde{p}_{\phi_L}(x|\theta)$, trained posterior model $\tilde{p}_{\phi_P}(\theta|x)$.

Set $\mathcal{D} = \{\emptyset\}$.

for $r = 1 : R$ **do**

// Step 1: Update likelihood model with training data sampled from a mixture of the prior and the current posterior model //

for $n = 1 : N$ **do**

Sample

$$(\theta^n, x^n) \sim \tilde{p}(\theta, x) = p(x|\theta)\hat{p}_r(\theta|x^{\text{obs}}), \tag{135}$$

where $\hat{p}_r(\theta|x_{\text{obs}}) = \alpha p(\theta) + (1-\alpha)\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$ and, for example, $\alpha = \exp(-\lambda \cdot (r-1))$.

end for

Update training data $\mathcal{D} = [\theta^{1:N}, x^{1:N}] \cup \mathcal{D}$.

Update $\tilde{p}_{\phi_L}(x|\theta)$ by minimizing the following loss

$$\mathcal{L}(\phi_L) = -E_{\hat{p}(\theta,x|x_{\text{obs}})}\big(\log \tilde{p}_{\phi_L}(x|\theta)\big), \tag{136}$$

$$\propto E_{\hat{p}(\theta|x_{\text{obs}})}\Big(D_{KL}\big(p(x|\theta))\big|\big|\tilde{p}_{\phi_L}(x|\theta)\big)\Big). \tag{137}$$

if $r = 1$ **then**

Using the prior predictive samples $[\theta^{1:N}, x^{1:N}]$, update the posterior model by minimizing the following loss

$$\mathcal{L}(\phi_P) \propto -E_{p(\theta,x)=p(x|\theta)p(\theta)}\big(\log \tilde{p}_{\phi_P}(\theta|x)\big). \tag{138}$$

end if

// Step 2: Update the posterior model with training data generated from the current posterior //

for $j = 1 : N_P/N_{\text{mini}}$ **do**

For $i = 1 : N_{\text{mini}}$: Sample $\theta_i \sim \tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$

Update posterior model, i.e. obtain a new ϕ_P by minimizing the loss (reverse KL divergence):

$$\mathcal{L}(\phi_P) = D_{KL}\big(\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})\big|\big|\tilde{p}_{\phi_L}(x_{\text{obs}}|\theta)p(\theta)\big). \tag{139}$$

end for

end for

- *Bayesian Optimization for Likelihood-Free Inference* [Gutmann and Corander, 2016] is simulation-based method that leverges Bayesian optimization.
- *Amortized Approximate Likelihood Ratio MCMC* [Hermans et al., 2020] is an amortized ratio-estimation method.
- *BayesFlow* [Radev et al., 2020] is a amortized methods that learns the *global* posterior distribution from the prior predictive distribution.

5 Outline of papers and author's contributions

This section briefly discusses the publication status and the replicability of the papers. Each paper is then briefly summarised; the summaries also include information regarding my contributions.

Publication status and replicability Paper I and II have been published in the *Proceedings of the 36th International Conference on Machine Learning* and *Computational Statistics & Data Analysis* respectively. Paper III is currently under review for NeurIPS 2021, and paper IV is currently in preparation. The code (and most data) used for all papers are publicly available at `https://github.com/SamuelWiqvist/`.

5.1 Paper I: Partially exchangeable networks and architectures for learning summary statistics in approximate Bayesian computation

In the spirit of Jiang et al. [2017], paper I considers the problem of learning summary statistics for ABC via deep learning. The particular task considered is how to learn summary statistics for Markovian data efficiently. To achieve that paper I introduces the *partially exchangeable network* (PEN) architecture that was introduced in Section 4.1.2. The PEN architecture is invariant to Markovian data, and PEN is, therefore, particularly useful for learning summary statistics of Markovian data since PEN (in contrast to an MLP) does not have to learn the Markovian feature of the data. PEN can be viewed as an extension of DeepSets [Zaheer et al., 2017] that incorporates the invariant properties for Markovian data, while DeepSets leverages the invariant properties of i.i.d data.

The main result of paper I is that PEN manages to learn informative summary statistics for ABC and is more data-efficient than MLPs. Thus, MLP requires more train-

ing data to achieve the same performance as PEN. PEN is more data-efficient since it leverages the invariant properties of the Markovian data, and hence it does not need to learn these invariant properties.

As a curiosity, it can be mentioned that paper I was discussed on Christian Robert's blog [Robert, 2019].

Contributions Samuel Wiqvist (SW) implemented all methods and ran all analyses. SW also contributed considerably to the writing of the paper.

5.2 Paper II: Efficient inference for stochastic differential equation mixed-effects models using correlated particle pseudo-marginal algorithms

Paper II introduces a novel MCMC algorithm for inference in SDEMEMs. The inference procedure introduced in paper II aims at sampling from the following posterior

$$p(\eta, \xi, \phi|y) \propto p(\eta)p(\xi)p(\phi|\eta)p(y|\xi, \phi) \tag{140}$$

$$\propto p(\eta)p(\xi) \prod_{i=1}^{M} p(\phi^i|\eta)p(y^i|\xi, \phi^i). \tag{141}$$

In Equation (141), $p(y^i|\xi, \phi^i)$ is the *data likelihood*. However, the inference problem is complicated by the intractability of the data likelihood. The complexity of the inference problem and the Gibbs algorithm introduced in paper II was outlined in Section 4.4. The Gibbs algorithm of paper II is made computationally efficient by leveraging the correlated pseudo-marginal method and a careful blocking strategy that allows us to update the underlying pseudo-random numbers only once. The efficient improvement that is obtained from using correlated approaches is data and model dependent, but we typically observe an improvement of efficiency of one order of magnitude compared to more standard pseudo-marginal methods.

Another recent paper on Bayesian inference for SDEMEM by Botha et al. [2021] was developed independently and concurrently of paper II. Botha et al. [2021] also considers correlated particle methods for inference in SDEMEMS. The method *component-wise pseudo-marginal* (CWPM) of Botha et al. [2021] is similar to the CPMMH method of paper II. CWPM is also the method that performs the best in Botha et al. [2021].

The method of paper II has been further developed in Persson et al. [2021] (i.e., paper *A*). Persson et al. [2021] adapt the methodology of paper II for inference in stochastic dynamical single-cell models. Persson et al. [2021] also show how the computational challenging step 2 of the Gibbs algorithm of paper II can be avoided by reparameterizing the model and how the methodology of paper II can be adapted to different types of latent processes.

Contributions SW contributed to the methodological developments introduced in the paper. For the Ornstein-Uhlenbeck and neural data case studies SW was responsible for writing the code, running the experiments, and analyzing the results. SW also contributed considerably to the writing of the paper.

5.3 Paper III: Sequential neural posterior and likelihood approximation

The *sequential neural posterior and likelihood approximation* (SNPLA) method of paper III was introduced in Section 4.6.3. The main feature of SNPLA is that the algorithm learns both the posterior and the likelihood, thus SNPLA has two trainable models:

1. **Posterior model** $\tilde{p}_{\phi_P}(\theta|x^{\text{obs}})$, approximating the posterior distribution $p(\theta|x^{\text{obs}})$.
2. **Likelihood model** $\tilde{p}_{\phi_L}(x|\theta)$. Since we are considering an implicit statistical model, we consider the likelihood model $\tilde{p}_{\phi_L}(x|\theta)$ as approximating the data generating process $p(x|\theta)$.

The learning process for the likelihood model $\tilde{p}_{\phi_L}(x|\theta)$ is straightforward and resembles the learning process of the SNL method. However, the learning process for the posterior model $\tilde{p}_{\phi_P}(\theta|x^{\text{obs}})$ is more complex, since the posterior model is not trained according to its true target distribution, the likelihood model $\tilde{p}_{\phi_L}(x|\theta)$ is instead used to approximate the target for the posterior model. However, several approaches designed to alleviate the complex learning process of the posterior model are discussed in paper III.

The results of paper III show that, when utilizing the same number of model simulations, SNPLA produces posterior inference results at par with comparable methods (e.g., SNPE-C and SNL). This result is of interest since the learning process of SNPLA, which is about learning two quantities instead of one, is more complex than learning processes of the compared methods.

Paper III also investigates the performance and utility of the likelihood model. The results of paper III show that the likelihood model of SNPLA performs similarly well as the likelihood model of SNL. A case study of paper III also shows how the likelihood model can be used to rapidly scan for parameters from the prior distribution that produce data similar to the observed data.

Contributions The original research idea and the algorithm was primarily developed by SW with help from Jes Frellsen (JF). SW wrote all the code and ran all analyses. The paper was primarily written by SW with contributions from JF and Umberto Picchini (UP).

5.4 Paper IV: Accelerating delayed-acceptance Markov chain Monte Carlo algorithms

Paper IV introduces the *accelerated delayed-acceptance* (ADA) algorithm. ADA is an extension of the *delayed-acceptance* (DA) algorithm [Christen and Fox, 2005] where relationships between the likelihood ratios of the two DA steps are utilized to construct an accelerated algorithm. ADA, similarly to DA, employs a surrogate model of the likelihood to scan parameters proposals, such that the true likelihood function is only evaluated for proposals that are likely to be accepted. Paper IV employs a surrogate likelihood model based on Gaussian processes that is inspired by the surrogate model utilized in Drovandi et al. [2018]. However, unlike DA, ADA will not produce exact Bayesian inference, but ADA allows for an accelerated inference procedure.

The main case study of paper IV considers the problem of modeling protein folding data (reaction coordinate data) via stochastic differential equations (SDEs) (see Section 3.3 for a technical description SDE model considered in paper IV). However, additional analyses (currently not included in the paper IV) indicate that the proposed SDE model in paper IV is over-parameterized and that the SDE model's performance can be improved by considering less flexible SDE model.

The results of paper IV show that the ADA method indeed manages to produce accelerated inference for cases where the likelihood function is time-consuming to evaluate. The inference results of ADA are similar to the results obtained by methods that produce exact Bayesian inference.

Contributions SW developed the double-well potential model, implemented the methods, and ran all analyses. SW also contributed considerably to the writing of

the paper.

References

M. Åkesson, P. Singh, F. Wrede, and A. Hellander. Convolutional neural networks as summary statistics for approximate bayesian computation. *arXiv preprint arXiv:2001.11760*, 2020.

D. Allingham, R. King, and K. L. Mengersen. Bayesian estimation of quantile distributions. *Statistics and Computing*, 19(2):189–201, 2009.

C. Andrieu and G. O. Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37:697–725, 2009.

C. Andrieu and J. Thoms. A tutorial on adaptive mcmc. *Statistics and computing*, 18 (4):343–373, 2008.

C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18, 2018.

M. A. Beaumont. Estimation of population growth or decline in genetically monitored populations. *Genetics*, 164(3):1139–1160, 2003.

M. A. Beaumont, W. Zhang, and D. J. Balding. Approximate bayesian computation in population genetics. *Genetics*, 162(4):2025–2035, 2002.

M. A. Beaumont, J.-M. Cornuet, J.-M. Marin, and C. P. Robert. Adaptive approximate bayesian computation. *Biometrika*, 96(4):983–990, 2009.

E. Bernton, P. E. Jacob, M. Gerber, and C. P. Robert. Approximate bayesian computation with the wasserstein distance. *arXiv preprint arXiv:1905.03747*, 2019.

R. B. Best and G. Hummer. Diffusion models of protein folding. *Physical Chemistry Chemical Physics*, 13(38):16902–16911, 2011.

C. M. Bishop. Mixture density networks. 1994.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

B. Bloem-Reddy and Y. Teh. Probabilistic symmetry and invariant neural networks. arxiv. *arXiv preprint arXiv:1901.06082*, 2019.

I. Botha, R. Kohn, and C. Drovandi. Particle methods for stochastic differential equation mixed effects models. *Bayesian Analysis*, 16(2):575–609, 2021.

C. Bretó, D. He, E. L. Ionides, and A. A. King. Time series analysis via mechanistic models. *The Annals of Applied Statistics*, pages 319–348, 2009.

S. Brooks, A. Gelman, G. Jones, and X.-L. Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

O. Cappé, S. J. Godsill, and E. Moulines. An overview of existing methods and recent advances in sequential monte carlo. *Proceedings of the IEEE*, 95(5):899–924, 2007.

A. Carrier. Flash of genius, May 2021. URL `https://www.radcliffe.harvard.edu/news-and-ideas/flash-of-genius`.

N. Chopin, O. Papaspiliopoulos, et al. *An introduction to sequential Monte Carlo*, volume 4. Springer, 2020.

J. A. Christen and C. Fox. Markov chain monte carlo using an approximation. *Journal of Computational and Graphical statistics*, 14(4):795–810, 2005.

K. Cranmer, J. Brehmer, and G. Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 2020. doi: 10.1073/pnas.1912789117.

M. Creel. Neural nets for indirect inference. *Econometrics and Statistics*, 2:36–49, 2017.

J. Dahlin, F. Lindsten, J. Kronander, and T. B. Schon. Accelerating pseudo-marginal Metropolis-Hastings by correlating auxiliary variables. Available from `https://arxiv.1511.05483v1`, 2015.

B. De Finetti. *Theory of probability: A critical introductory treatment*. John Wiley & Sons, 2017.

G. Deligiannidis, A. Doucet, and M. K. Pitt. The correlated pseudo-marginal method. *J. R. Statist. Soc.B*, 80:839–870, 2018.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. 2017. URL `https://arxiv.org/abs/1605.08803`.

S. Donnet and A. Samson. A review on estimation of stochastic differential equations for pharmacokinetic/pharmacodynamic models. *Advanced drug delivery reviews*, 65 (7):929–939, 2013.

A. Doucet, M. K. Pitt, and R. Kohn. Efficient implementation of Markov chain Monte Carlo when using an unbiased likelihood estimator. *Biometrika*, 102:295–313, 2015.

C. Drovandi and D. T. Frazier. A comparison of likelihood-free methods with and without summary statistics. *arXiv preprint arXiv:2103.02407*, 2021.

C. C. Drovandi, M. T. Moores, and R. J. Boys. Accelerating pseudo-marginal MCMC using Gaussian processes. *Computational Statistics & Data Analysis*, 118: 1–17, 2018.

D. B. Dunson and J. Johndrow. The hastings algorithm at fifty. *Biometrika*, 107(1): 1–23, 2020.

C. Durkan, G. Papamakarios, and I. Murray. Sequential neural methods for likelihood-free inference. *arXiv preprint arXiv:1811.08723*, 2018.

C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems*, pages 7511–7522, 2019.

C. Durkan, I. Murray, and G. Papamakarios. On contrastive learning for likelihood-free inference. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2771–2781, 2020.

S.-C. Fang, D. Y. Gao, G.-X. Lin, R.-L. Sheu, and W.-X. Xing. Double well potential function and its optimization in the n-dimensional real space: part i. *Journal of Industrial and Management Optimization*, 13(3):1291–1305, 2017.

P. Fearnhead and D. Prangle. Constructing summary statistics for approximate bayesian computation: Semi-automatic approximate bayesian computation. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(3):419–474, 2012.

J. Gabry, D. Simpson, A. Vehtari, M. Betancourt, and A. Gelman. Visualization in bayesian workflow. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 182(2):389–402, 2019.

A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian data analysis*. CRC press, 2013.

A. Gelman, A. Vehtari, D. Simpson, C. C. Margossian, B. Carpenter, Y. Yao, L. Kennedy, J. Gabry, P.-C. Bürkner, and M. Modrák. Bayesian workflow. *arXiv preprint arXiv:2011.01808*, 2020.

D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361, 1977.

G. H. Givens and J. A. Hoeting. *Computational statistics*, volume 703. John Wiley & Sons, 2012.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

M. Goldstein. Subjective bayesian analysis: principles and practice. *Bayesian analysis*, 1(3):403–420, 2006.

A. Golightly and D. J. Wilkinson. Bayesian parameter inference for stochastic biochemical network models using particle markov chain monte carlo. *Interface focus*, 1(6):807–820, 2011.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

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

C. Gourieroux, A. Monfort, and E. Renault. Indirect inference. *Journal of applied econometrics*, 8(S1):S85–S118, 1993.

D. Greenberg, M. Nonnenmacher, and J. Macke. Automatic posterior transformation for likelihood-free inference. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2404–2414, 2019.

M. U. Gutmann and J. Corander. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *The Journal of Machine Learning Research*, 17 (1):4256–4302, 2016.

H. Haario, E. Saksman, and J. Tamminen. An adaptive metropolis algorithm. *Bernoulli*, pages 223–242, 2001.

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

J. Hermans, V. Begy, and G. Louppe. Likelihood-free MCMC with amortized approximate ratio estimators. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4239–4248, 2020.

C. F. Higham and D. J. Higham. Deep learning: An introduction for applied mathematicians. *SIAM Review*, 61(4):860–891, 2019.

B. Jiang, T.-y. Wu, C. Zheng, and W. H. Wong. Learning summary statistic for approximate bayesian computation via deep neural network. *Statistica Sinica*, pages 1595–1618, 2017.

J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, pages 1–11, 2021.

R. E. Kalman. A new approach to linear filtering and prediction problems. 1960.

I. Kobyzev, S. Prince, and M. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

M. Lavielle. *Mixed effects models for the population approach: models, tasks, methods and tools*. CRC press, 2014.

Y. LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, 19:143–155, 1989.

J.-M. Lueckmann, P. J. Goncalves, G. Bassetto, K. Öcal, M. Nonnenmacher, and J. H. Macke. Flexible statistical inference for mechanistic models of neural dynamics. In *Advances in Neural Information Processing Systems*, pages 1289–1299, 2017.

J.-M. Lueckmann, G. Bassetto, T. Karaletsos, and J. H. Macke. Likelihood-free inference with emulator networks. In F. Ruiz, C. Zhang, D. Liang, and T. Bui, editors, *Proceedings of The 1st Symposium on Advances in Approximate Bayesian Inference*, volume 96 of *Proceedings of Machine Learning Research*, pages

32–53. PMLR, 02 Dec 2019. URL http://proceedings.mlr.press/v96/lueckmann19a.html.

J.-M. Lueckmann, J. Boelts, D. S. Greenberg, P. J. Gonçalves, and J. H. Macke. Benchmarking simulation-based inference. *arXiv preprint arXiv:2101.04653*, 2021.

J.-M. Marin, P. Pudlo, C. P. Robert, and R. J. Ryder. Approximate bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012.

P. Marjoram, J. Molitor, V. Plagnol, and S. Tavaré. Markov chain monte carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 100(26):15324–15328, 2003.

D. McFadden. A method of simulated moments for estimation of discrete response models without numerical integration. *Econometrica: Journal of the Econometric Society*, pages 995–1026, 1989.

F. J. Medina-Aguayo, A. Lee, and G. O. Roberts. Stability of noisy Metropolis–Hastings. *Statistics and Computing*, 26:1187–1211, 2016.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

M. L. Minsky and S. A. Papert. Perceptrons: expanded edition, 1988.

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

V. M. Ong, D. J. Nott, M.-N. Tran, S. A. Sisson, and C. C. Drovandi. Variational bayes with synthetic likelihood. *Statistics and Computing*, 28(4):971–988, 2018.

G. Papamakarios and I. Murray. Fast ε-free inference of simulation models with bayesian conditional density estimation. In *Advances in Neural Information Processing Systems*, pages 1028–1036, 2016.

G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019a.

G. Papamakarios, D. Sterratt, and I. Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 837–848. PMLR, 16–18 Apr 2019b. URL `http://proceedings.mlr.press/v89/papamakarios19a.html`.

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

S. Persson, N. Welkenhuysen, S. Shashkova, S. Wiqvist, P. Reith, G. W Schmidt, U. Picchini, and M. Cvijovic. Pepsdi: Scalable and flexible inference framework for stochastic dynamic single-cell models. *bioRxiv preprint bioRxiv:2021.07.01.450748*, 2021.

G. W. Peters, S. A. Sisson, and Y. Fan. Likelihood-free bayesian inference for α-stable models. *Computational Statistics & Data Analysis*, 56(11):3743–3756, 2012.

U. Picchini and R. Anderson. Approximate maximum likelihood estimation using data-cloning ABC. *Computational Statistics & Data Analysis*, 105:166–183, 2017.

U. Picchini and R. G. Everitt. Stratified sampling and bootstrapping for approximate bayesian computation. *arXiv preprint arXiv:1905.07976*, 2019.

M. K. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American statistical association*, 94(446):590–599, 1999.

D. Prangle. gk: An R package for the g-and-k and generalised g-and-h distributions. *arXiv:1706.06889*, 2017.

L. F. Price, C. C. Drovandi, A. Lee, and D. J. Nott. Bayesian synthetic likelihood. *Journal of Computational and Graphical Statistics*, 27(1):1–11, 2018.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018. URL `https://www.R-project.org/`.

S. T. Radev, U. K. Mertens, A. Voss, L. Ardizzone, and U. Köthe. Bayesflow: Learning complex stochastic models with invertible neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015.

C. Robert. a pen for abc, Feb 2019. URL `https://xianblog.wordpress.com/2019/02/13/a-pen-for-abc/`.

H. Rue, A. Riebler, S. H. Sørbye, J. B. Illian, D. P. Simpson, and F. K. Lindgren. Bayesian computing with inla: a review. *Annual Review of Statistics and Its Application*, 4:395–421, 2017.

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

S. Särkkä. *Bayesian filtering and smoothing*. Number 3. Cambridge University Press, 2013.

J. Shawe-Taylor. Building symmetries into feedforward networks. In *1989 First IEE International Conference on Artificial Neural Networks,(Conf. Publ. No. 313)*, pages 158–162. IET, 1989.

C. Sherlock, A. Thiery, G. O. Roberts, and J. S. Rosenthal. On the effciency of pseudo-marginal random walk Metropolis algorithms. *The Annals of Statistics*, 43 (1):238–275, 2015.

S. Singh, A. Okun, and A. Jackson. Learning to play go from scratch. *Nature*, 550 (7676):336–337, 2017.

S. A. Sisson, Y. Fan, and M. M. Tanaka. Sequential monte carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 104(6):1760–1765, 2007.

S. A. Sisson, Y. Fan, and M. Beaumont. *Handbook of approximate Bayesian computation*. Chapman and Hall/CRC, 2018.

L. Stewart and P. McCarty Jr. Use of bayesian belief networks to fuse continuous and discrete information for target recognition, tracking, and situation assessment. In *Signal Processing, Sensor Fusion, and Target Recognition*, volume 1699, pages 177–185. International Society for Optics and Photonics, 1992.

S. Tavaré, D. J. Balding, R. C. Griffiths, and P. Donnelly. Inferring coalescence times from dna sequence data. *Genetics*, 145(2):505–518, 1997.

O. Thomas, R. Dutta, J. Corander, S. Kaski, and M. U. Gutmann. Likelihood-free inference by ratio estimation. *Bayesian Analysis*, 2020.

T. Toni. *Approximate Bayesian computation for parameter inference and model selection in systems biology*. PhD thesis, Imperial College London, 2010.

M.-N. Tran, R. Kohn, M. Quiroz, and M. Villani. Block-wise pseudo-marginal metropolis-hastings. *arXiv:1603.02485*, 2016.

J. Venn. *The logic of chance: an essay on the foundations and province of the theory of probability, with especial reference to its logical bearings and its application to moral and social science, and to statistics*. Macmillan, 1866.

M. Vihola. Robust adaptive metropolis algorithm with coerced acceptance rate. *Statistics and Computing*, 22(5):997–1008, 2012.

D. J. Wilkinson. *Stochastic modelling for systems biology*. CRC Press, 2011.

C. Winkler, D. Worrall, E. Hoogeboom, and M. Welling. Learning likelihoods with conditional normalizing flows. *arXiv preprint arXiv:1912.00042*, 2019.

S. N. Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104, 2010.

M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

Paper I

Wiqvist, S., Mattei, P. A., Picchini, U., & Frellsen, J.
Partially exchangeable networks and architectures for learning summary statistics in approximate Bayesian computation
International Conference on Machine Learning, (pp. 6798-6807). PMLR.

The paper included here is a slightly typographically modified single column version of the paper accepted for ICML 2019.

Partially Exchangeable Networks and Architectures for Learning Summary Statistics in Approximate Bayesian Computation

Samuel Wiqvist[1,⋆], Pierre-Alexandre Mattei[2,⋆], Umberto Picchini[3], Jes Frellsen[2]

[1]Centre for Mathematical Sciences, Lund University, Lund, Sweden
[2]Department of Computer Science, IT University of Copenhagen, Copenhagen, Denmark
[3]Department of Mathematical Sciences, Chalmers University of Technology and the University of Gothenburg, Gothenburg, Sweden

Abstract

We present a novel family of deep neural architectures, named partially exchangeable networks (PENs) that leverage probabilistic symmetries. By design, PENs are invariant to block-switch transformations, which characterize the partial exchangeability properties of conditionally Markovian processes. Moreover, we show that any block-switch invariant function has a PEN-like representation. The DeepSets architecture is a special case of PEN and we can therefore also target fully exchangeable data. We employ PENs to learn summary statistics in approximate Bayesian computation (ABC). When comparing PENs to previous deep learning methods for learning summary statistics, our results are highly competitive, both considering time series and static models. Indeed, PENs provide more reliable posterior samples even when using less training data.

1 Introduction

We propose a novel neural network architecture to ease the application of approximate Bayesian computation (ABC), a.k.a. *likelihood-free* inference. The architecture, called partially exchangeable network (PEN), uses partial exchangeability in Markovian data, allowing us to perform ABC inference for time series models with Markovian structure. Empirically, we also show that we can target non-Markovian time series data with PENs. Since the DeepSets architecture [Zaheer et al., 2017] turns out to be a special case of PEN, we can also perform ABC inference for static models. Our work is about automatically construct summary statistics of the data that are informative for model parameters. This is a main challenge in the practical application of ABC algorithms, since such summaries are often *handpicked* (i.e. ad-hoc summaries are constructed from model domain expertise), or these are automatically constructed using a number of approaches as detailed in 2. Neural networks have been previously used to automatically construct summary statistics for ABC. Jiang et al. [2017] and Creel [2017] employ standard multilayer perceptron (MLP) networks for learning the summary statistics. Chan et al. [2018] introduce a network that exploits the exchangeability property in exchangeable data. Our PEN architecture is a new addition to the tools for automatic construction of summary statistics, and PEN produces competitive inference results compared to Jiang et al. [2017], which in turn was shown outperforming the semi-automatic regression method by Fearnhead and Prangle [2012]. Moreover, our PEN architecture is more data efficient and when reducing the training data PEN outperforms Jiang et al. [2017], the factor of reduction being of order 10 to 10^2 depending on cases.

Our **main contributions** are:

- Introducing the partially exchangeable networks (PENs) architecture;
- Using PENs to automatically learn summary statistics for ABC inference. We consider both static and dynamic models. In particular, our network architecture is specifically designed to learn summary statistics for dynamic models.

⋆Equal contribution.

2 Approximate Bayesian computation

Approximate Bayesian computation (ABC) is an increasingly popular inference method for model parameters θ, in that it only requires the ability to produce artificial data from a stochastic model *simulator* [Beaumont et al., 2002, Marin et al., 2012]. A simulator is essentially a computer program, which takes θ, makes internal calls to a random number generator, and outputs a vector of artificial data. The implication is that ABC can be used to produce approximate inference when the likelihood function $p(y|\theta)$ underlying the simulator is intractable. As such ABC methods have been applied to a wide range of disciplines [Sisson et al., 2018]. The fundamental idea in ABC is to generate parameter proposals $\theta^\star$ and accept a proposal if the simulated data $y^\star$ for that proposal is similar to observed data y^{obs}. Typically this approach is not suitable for high-dimensional data, and a set of summary statistics of the data is therefore commonly introduced to break the *curse-of-dimensionality*. So, instead of comparing $y^\star$ to y^{obs}, we compare summary statistics of the simulated data $s^\star = S(y^\star)$ to those of observed data $s^{\text{obs}} = S(y^{\text{obs}})$. Then we accept the proposed $\theta^\star$ if $s^\star$ is close to s^{obs} in some metric. Using this scheme, ABC will simulate draws from the following approximate posterior of θ

$$p^{\epsilon}_{\text{ABC}}(\theta|s^{\text{obs}}) \propto \int K_\epsilon(\Delta(s^\star, s^{\text{obs}}))p(s^\star|\theta)p(\theta)ds^\star,$$

where $p(\theta)$ is the prior of θ, Δ is a distance function between observed and simulated summaries (we use a Mahalanobis distance, see the supplementary material), $K_\epsilon(\cdot)$ is a kernel, which in all our applications is the uniform kernel returning 1 if $\Delta(s^\star, s^{\text{obs}}) < \epsilon$ and 0 otherwise, and $\epsilon > 0$ is the so-called ABC-threshold. A smaller ϵ produces more accurate approximations to the true summaries posterior $p(\theta|s^{\text{obs}})$, though this implies a larger computational effort due to the increasing number of rejected proposals. An additional issue is that ideally we would like to target $p(\theta|y^{\text{obs}})$, not $p(\theta|s^{\text{obs}})$, but again unless sufficient statistics are available (impossible outside the exponential family), and since $\epsilon > 0$, we have to be content with samples from $p^{\epsilon}_{\text{ABC}}$.

In this work we do not focus on *how* to sample from $p^{\epsilon}_{\text{ABC}}(\theta|s^{\text{obs}})$ (see Sisson et al., 2018 for possibilities). Therefore, we employ the simplest (and also most inefficient) ABC algorithm, the so called "ABC rejection sampling" [Pritchard et al., 1999]. We will use the "reference table" version of ABC rejection sampling (e.g. Cornuet et al., 2008), which is as follows:

- Generate $\tilde{N}$ independent proposals $\theta^i \sim p(\theta)$, and corresponding data $y^i \sim p(y|\theta^i)$ from the simulator;
- Compute the summary statistics $s^i = S(y^i)$ for each $i = 1, ..., \tilde{N}$;
- Compute the distances $\Delta(s^i, s^{\text{obs}})$ for each $i = 1, ..., \tilde{N}$;
- Retain proposals θ^i corresponding to those $\Delta(s^i, s^{\text{obs}})$ that are smaller than the x-th percentile of all distances.

The retained θ^i's form a sample from $p^{\epsilon}_{\text{ABC}}$ with ϵ given by the selected xth percentile. An advantage of this approach is that it allows to easily compare the quality of the ABC inference based on several methods for computing the summaries, under the same computational budget $\tilde{N}$. Moreover, once the "reference table" $(\theta^i, y^i)_{1 \leq i \leq \tilde{N}}$ has been produced in the first step, we can recycle these simulations to produce new posterior samples using several methods for computing the summary statistics.

2.1 Learning summary statistics

Event though ABC rejection sampling is highly inefficient due to proposing parameters from the prior $p(\theta)$, this is not a concern for the purpose of our work. In fact, our main focus is *learning* the summary statistics $S(\cdot)$. This is perhaps the most serious difficulty affecting the application of ABC methodology to practical problems. In fact, we require summaries that are informative for θ, as a replacement for the (unattainable) sufficient statistics. A considerable amount of research has been conducted on how to construct informative summary statistics (see Blum et al., 2013 and Prangle, 2015 for an overview). However their selection is still challenging since no state-of-the-art methodology exists that can be applied to arbitrarily complex problems. Fearnhead and Prangle [2012] consider a regression-based approach where they also show that the best summary statistic, in terms of the minimal quadratic loss, is the posterior mean. The latter is however

unknown since $p(\theta|y^{\text{obs}})$ itself is unknown. Therefore, they introduce a simulation approach based on a linear regression model

$$\theta_j^i = E(\theta_j|y^i) + \xi_j^i = b_{0_j} + b_j h(y^i) + \xi_j^i \tag{1}$$

with ξ_j^i some mean-zero noise. Here $j = 1, ..., \dim(\theta)$ and $h(y^i)$ is a vector of (non)-linear transformations of "data" y^i (here y^i can be simulated or observed data). Therefore Fearnhead and Prangle [2012] have $\dim(\theta)$ models to fit separately, one for each component of vector θ. Of course, these fittings are to be performed *before* ABC rejection is executed, so this is a step that anticipates ABC rejection, to provide the latter with suitable summary statistics. The parameters in each regression (1) are estimated by fitting the model by least squares to a new set of N simulated data-parameter pairs $(\theta^i, y^i)_{1\leq i\leq N}$ where, same as for ABC rejection, the θ^i are generated from $p(\theta)$ and the y^i are generated from the model simulator conditionally on θ^i. To clarify the notation: N is the number of data-parameter pairs used to fit the linear regression model in (1), while $\tilde{N}$ is the number of parameter-data pair proposals used in ABC rejection sampling. However the two sets of parameter-data pairs $(\theta^i, y^i)_{1\leq i\leq N}$ and $(\theta^i, y^i)_{1\leq i\leq \tilde{N}}$ are different since these serve two separate purposes. They are generated in the same way but independently of each other. After fitting (1), estimates $(\hat{b}_{0_j}, \hat{b}_j)$ are returned and $\hat{b}_{0_j} + \hat{b}_j h(y)$ is taken as jth summary statistic, $j = 1, ..., \dim(\theta)$. We can then take $S_j(y^{\text{obs}}) = \hat{b}_{0_j} + \hat{b}_j h(y^{\text{obs}})$ as jth component of $S(y^{\text{obs}})$, and similarly take $S_j(y^\star) = \hat{b}_{0_j} + \hat{b}_j h(y^\star)$. The number of summaries is therefore equal to the size of θ.

This approach is further developed in Jiang et al. [2017] where a MLP deep neural network regression model is employed, and replaces the linear regression model in (1). Hence, Jiang et al. [2017] has the following regression model

$$\theta^i = E(\theta|y^i) + \xi^i = f_\beta(y^i) + \xi^i$$

where f_β is the MLP parametrized by the weights β. Jiang et al. [2017] estimate β from

$$\min_\beta \frac{1}{N} \sum_{i=1}^{N} \|f_\beta(y^i) - \theta^i\|_2^2, \tag{2}$$

where $(\theta^i, y^i)_{1\leq i\leq N}$ are the parameter-data pairs that the network f_β is fitted to.

The deep neuronal network with multiple hidden layers considered in Jiang et al. [2017] offers stronger representational power to approximate $E(\theta|y)$ (and hence learn an informative summary statistic), compared to using linear regression, if the posterior mean is a highly non-linear function of y. Moreover, experiments in Jiang et al. [2017] show that indeed their MLP outperforms the linear regression approach in Fearnhead and Prangle [2012] (at least for their considered experiments), although at the price of a much larger computational effort. For this reason in our experiments we compare ABC coupled with PENs with the ABC MLP from Jiang et al. [2017].

In Creel [2017] a deep neural network regression model is used. He also introduces a pre-processing step such that instead of feeding the network with the data set y^{obs}, the network is fed with a set of statistics of the data s^{obs}. This means that, unlike in Jiang et al. [2017], in Creel [2017] the statistician must already know "some kind" of initial summary statistics, used as input, and then the network returns another set of summary statistics as output, and the latter are used for ABC inference. Our PENs do not require any initial specification of summary statistics.

3 Partially exchangeable networks

Even though the likelihood function is intractable in the likelihood-free setting, we may still have insights into properties of the data generating process. To that end, given our data set $y \in \mathcal{Y}^M$ with M units, we will exploit some of the invariance properties of its prior predictive distribution $p(y) = \int_\theta p(y|\theta)p(\theta)d\theta$. As discussed in 2, the regression approach to ABC [Fearnhead and Prangle, 2012] involves to learn the regression function $y \mapsto E(\theta|y)$, where $E(\theta|y)$ is the posterior mean. Our goal in this section is to leverage the invariances of the Bayesian model $p(y)$ to design deep neural architectures that are fit for this purpose.

3.1 Exchangeability and partial exchangeability

The simplest form of model invariance is *exchangeability*. A model $p(y)$ is said to be exchangeable if, for all permutations σ in the symmetric group S_M, $p(y) = p(y_{\sigma(1)}, ..., y_{\sigma(M)})$. For example, if the observations are independent and identically distributed (i.i.d.) given the parameter, then $p(y)$ is exchangeable. A famous theorem of de Finetti [1929], which was subsequently generalized in various ways (see e.g. the review of Diaconis, 1988), remarkably shows that such conditionally i.i.d. models are essentially the only exchangeable models.

If the model is exchangeable, it is clear that the function $y \mapsto E(\theta|y)$ is permutation invariant. It is therefore desirable that a neural network used to approximate this function should also be permutation invariant. The design of permutation invariant neural architectures has been the subject of numerous works, dating at least back to Minsky and Papert [1988, Chap. 2] and Shawe-Taylor [1989]. A renewed interest in such architectures came about recently, notably through the works of Ravanbakhsh et al. [2017], Zaheer et al. [2017], and Murphy et al. [2019]—a detailed overview of this rich line of work can be found in Bloem-Reddy and Teh [2019]. Most relevant to our work is the DeepSets architecture of Zaheer et al. [2017] that we generalize to partial exchangeability, and the approach of Chan et al. [2018], who used permutation invariant networks for ABC.

However, the models considered in ABC are arising from intractable-likelihoods scenarios, which certainly are not limited to exchangeable data, quite the opposite, e.g. stochastic differential equations [Picchini, 2014], state-space models and beyond [Jasra, 2015]. To tackle this limitation, we ask: *could we use a weaker notion of invariance to propose deep architectures suitable for such models?* In this paper, we answer this question for a specific class of non-i.i.d. models: Markov chains. To this end, we make use of the notion of *partial exchangeability* studied by Diaconis and Freedman [1980]. This property can be seen as a weakened version of exchangeability where $p(y)$ is only invariant to a subset of the symmetric group called *block-switch transformations*. Informally, for $d \in \mathbb{N}$, a d-block-switch transformation interchanges two given disjoint blocks of $y \in \mathcal{Y}^M$ when these two blocks start with the same d symbols and end with the same d symbols.

Definition 1 (**Block-switch transformation**). *For increasing indices $b = (i, j, k, l) \in \{0, \dots, M\}^4$ such that $j - i \geq d$ and $l - k \geq d$, the **d-block-switch transformation** $T_b^{(d)}$ is defined as follows: if $y_{i:(i+d)} = y_{k:(k+d)}$ and $y_{(j-d):j} = y_{(l-d):l}$ then*

$$y = y_{1:i-1}\ y_{i:j}\ y_{(j+1):(k-1)}\ y_{k:l}\ y_{(l+1):M} \tag{3}$$

$$T_b^{(d)}(y) = y_{1:i-1}\ y_{k:l}\ y_{(j+1):(k-1)}\ y_{i:j}\ y_{(l+1):M}. \tag{4}$$

If $y_{i:(i+d)} \neq y_{k:(k+d)}$ or $y_{(j-d):j} \neq y_{(l-d):l}$ then the block-switch transformation leaves y unchanged: $T_b^{(d)}(y) = y$.

Definition 2 (**Partial exchangeability**). *Let A be a metric space. A function $F : \mathcal{Y}^M \to A$ is said to be d-**block-switch invariant** if $F(y) = F(T_b^{(d)}(y))$ for all $y \in \mathcal{Y}$ and for all d-block-switch transformations $T_b^{(d)}$. Similarly, a model $p(y)$ is d-**partially exchangeable** if for all d-block-switch transformations $T_b^{(d)}$ we have $p(y) = p(T_b^{(d)}(y))$.*

Note that 0-partial exchangeability reduces to exchangeability and that all permutations are 0-block-switch transformations.

It is rather easy to see that, if $p(y|\theta)$ is a Markov chain of order d, then $p(y)$ is partially exchangeable (and therefore $y \mapsto E(\theta|y)$ is d-block-switch invariant). In the limit of infinite data sets, Diaconis and Freedman [1980] showed that the converse was also true: any partially exchangeable distribution is conditionally Markovian. This result, which is an analogue of de Finetti's theorem for Markov chains, justifies that *partial exchangeability is the right symmetry to invoke when dealing with Markov models.*

3.2 From model invariance to network architecture

When dealing with Markovian data, we therefore wish to model a regression function $y \mapsto E(\theta|y)$ that is d-block-switch invariant. Next theorem gives a general functional representation of such functions, in the case where $\mathcal{Y}$ is countable.

Theorem 1. *Let $F : \mathcal{Y}^M \to A$ be d-block-switch invariant. If $\mathcal{Y}$ is countable, then there exist two functions $\phi : \mathcal{Y}^{d+1} \to \mathbb{R}$ and $\rho : \mathcal{Y}^d \times \mathbb{R} \to A$ such that*

$$\forall y \in \mathcal{Y}^M,\ F(y) = \rho\left(y_{1:d}, \sum_{i=1}^{M-d} \phi\left(y_{i:(i+d)}\right)\right). \tag{5}$$

Proof. Let $\sim$ be the equivalence relation over $\mathcal{Y}^M$ defined by

$$x \sim y \iff \exists b_1, \ldots, b_k,\ \ y = T_{b_1}^{(d)} \circ \cdots \circ T_{b_k}^{(d)}(x).$$

Let $\mathrm{cl} : \mathcal{Y}^M \to \mathcal{Y}^M/\!\sim$ be the projection over the quotient set. According to the properties of the quotient set, since F is d-block-switch invariant, there exists a unique function $g : \mathcal{Y}^M/\!\sim \to A$ such that $F = g \circ \mathrm{cl}$.

Since $\mathcal{Y}$ is countable, $\mathcal{Y}^{d+1}$ is also countable and there exists an injective function $c : \mathcal{Y}^{d+1} \to \mathbb{N}$. Consider then the function

$$\nu : y \mapsto \left(y_{1:d}, \sum_{i=1}^{M-d} 2^{-c(y_{i:(i+d)})}\right),$$

which is clearly d-block-switch invariant. There exists a unique function $h : \mathcal{Y}^M/\!\sim \to \nu(\mathcal{Y}^M)$ such that $\nu = h \circ \mathrm{cl}$.

We will now show that h is a bijection. By construction, h is clearly surjective. Let us now prove its injectivity. We thus have to show that, for all $x, y \in \mathcal{Y}^M$, $\nu(x) = \nu(y)$ implies $x \sim y$. Let $x, y \in \mathcal{Y}^M$ such that $\nu(x) = \nu(y)$. We have therefore $x_{1:d} = y_{1:d}$ and

$$\sum_{i=1}^{M-d} 2^{-c(x_{i:(i+d)})} = \sum_{i=1}^{M-d} 2^{-c(y_{i:(i+d)})}.$$

The uniqueness of finite binary representations then implies that $\{x_{i:(i+d)}\}_{i \le M-d} = \{y_{i:(i+d)}\}_{i \le M-d}$. According to Diaconis and Freedman [1980, Proposition 27], those two conditions imply that $x \sim y$, which shows that h is indeed injective.

Since h is a bijection, $\nu = h \circ \mathrm{cl}$ implies that $\mathrm{cl} = h^{-1} \circ \nu$ which leads to $F = g \circ h^{-1} \circ \nu$. Finally, expanding this gives

$$\forall y \in \mathcal{Y}^M,\ F(y) = g \circ h^{-1}\left(y_{1:d}, \sum_{i=1}^{M-d} 2^{-c(y_{i:(i+d)})}\right),$$

which is the desired form with $\phi(y) = 2^{-c(y)}$ and $\rho = g \circ h^{-1}$. □

When $d = 0$, the representation reduces to

$$F(y) = \rho\left(\sum_{i=1}^{M} \phi\left(y_i\right)\right), \tag{6}$$

and we exactly recover Theorem 2 from Zaheer et al. [2017]—which also assumes countability of $\mathcal{Y}$—and the DeepSets representation. While an extension of our theorem to the uncountable case is not straightforward, we conjecture that a similar result holds even with uncountable $\mathcal{Y}$. A possible way to approach this conjecture is to study the very recent and fairly general result of Bloem-Reddy and Teh [2019]. We note that the experiments on an autoregressive time series model in 4.3, which is a Markovian process, support this conjecture.

Partially exchangeable networks The result in 1 suggests how to build d-block-switch invariant neural networks: we replace the functions ρ and ϕ in (5) by feed forward neural networks and denote this construction a d-partially exchangeable network (PEN-d or PEN of order d). In this construction, we will call ϕ the *inner network*, which maps a d-length subsequence $y_{i:i+d}$ into some representation $\phi(y_{i:i+d})$, and ρ is the *outer network* that maps the first d symbols of the input, and the sum of the representations of all d-length subsequences of the input, to the output. We note that DeepSets networks are a special case of the PENs that corresponds to PEN-0.

3.3 Using partially exchangeable networks for learning summary statistics for ABC

While PENs can by used for any exchangeable data, in this paper we use it for learning summary statistics in ABC. In particular, we propose the following regression model for learning the posterior mean

$$\theta^i = E(\theta|y^i) + \xi^i = \rho_{\beta_\rho}\left(y^i_{1:d}, \sum_{l=1}^{M-d} \phi_{\beta_\phi}(y^i_{l:l+d})\right) + \xi^i.$$

Here β_ϕ are the weights for the inner network, and β_ρ are the weights for the outer network that maps its arguments into the posterior mean of the unknown parameters, which is the ABC summary we seek. When using PENs to learn the summary statistics we obtain the weights for the networks using the same criterion as in (2), except that instead of using the MLP network we use a PEN network for the underlying regression problem.

When targeting static models we employ a PEN-0, i.e. a DeepSets network, since a static model can be viewed as a zero-order Markov model. For time series models we use a PEN-d, where $d > 0$ is the order of the assumed data generating Markov process.

4 Experiments

We present four experiments: two static models (g-and-k and α-stable distributions), and two time series models (autoregressive and moving average models). Full specification of the experimental settings is provided as supplementary material. The code was written in Julia 1.0.0 [Bezanson et al., 2017] and the framework Knet [Yuret, 2016] was used to build the deep learning models. The code can be found at `https://github.com/SamuelWiqvist/PENs-and-ABC`. All experiments are simulation studies and the data used can be generated from the provided code. We compare approximate posteriors to the true posteriors using the Wasserstein distance, which we compute via the POT package [Flamary and Courty, 2017]. This distance can be sensitive to the number of posterior samples used, however, we observed that our results are fairly robust to variations in the number of samples. In all experiments we used 100 posterior samples to estimate the Wasserstein distance, except for the AR2 model where we used 500 samples. We also employ two different MLP networks: "MLP small", where we use approximately the same number of weights as for the PEN-d network; and "MLP large", which has a larger number of weights than PEN-d.

4.1 g-and-k distribution

The g-and-k distribution is defined by its quantile function via four parameters, and not by its probability density function since the latter is unavailable in closed form. This means that the likelihood function is "intractable" and as such exact inference is not possible. However, it is very simple to simulate draws from said distribution (see the supplementary material), which means that g-and-k models are often used to test ABC algorithms [Prangle, 2017].

The unknown parameters are $\theta = [A, B, g, k]$ (for full specification of the g-and-k distribution, see the supplementary material). The prior distributions are set to $p(A) \sim \Gamma(2,1)$, $p(B) \sim \Gamma(2,1)$, $p(g) \sim \Gamma(2, 0.5)$, and $p(k) \sim \Gamma(2,1)$ ($\Gamma(\alpha, \beta)$ is the Gamma distribution with shape parameter α and rate parameter β). We perform a simulation study with ground-truth parameters $A = 3$, $B = 1$, $g = 2$, $k = 0.5$ (same ground-truth parameter values as in Allingham et al., 2009, Picchini and Anderson, 2017, Fearnhead and Prangle, 2012). Our data set comprises $M = 1,000$ realizations from a g-and-k distribution.

We compare five different methods of constructing the summary statistics for ABC: (i) the handpicked summary statistics in Picchini and Anderson [2017], i.e. $S(y) = [P_{20}, P_{40}, P_{60}, P_{80}, \text{skew}(y)]$ (P_i is the ith percentile and skew(y) is the skewness); (ii) "MLP small"; (iii) "MLP large"; (iv) a MLP network with a preprocessing step, denoted "MLP pre", where we feed the network with the empirical distribution function of the data instead of feeding it with the actual data; and (v) PEN-0 (DeepSets) since the data is i.i.d. the order of the Markov model is 0).

The probability density function for the g-and-k distribution can be approximated via finite differences, as implemented in the `gk` R package [Prangle, 2017]. This allow us to sample from an almost exact posterior

distribution using standard Markov chain Monte Carlo (MCMC). We evaluate the inference produced using summaries constructed from the five methods (i–v) by comparing the resulting ABC posteriors to the "almost exact" posterior (computed using MCMC). ABC inferences are repeated over 100 independent data sets, and for a different number of training data observations for DNN models. The results are presented in Figure 1 and we can conclude that PEN-0 generates the best results. Furthermore, PEN-0 is also more data efficient since it performs considerably better than other methods with limited number of training observations. It seems in fact that PEN-0 requires 10 times less training data than "MLP pre" to achieve the same inference accuracy. However all methods performed poorly when too few training observations are used. The results also show that when MLP is fed with the observations it generates poor results, but if we instead use "MLP pre" and send in the empirical distribution function, in the spirit of Creel [2017], we obtain considerably better results.

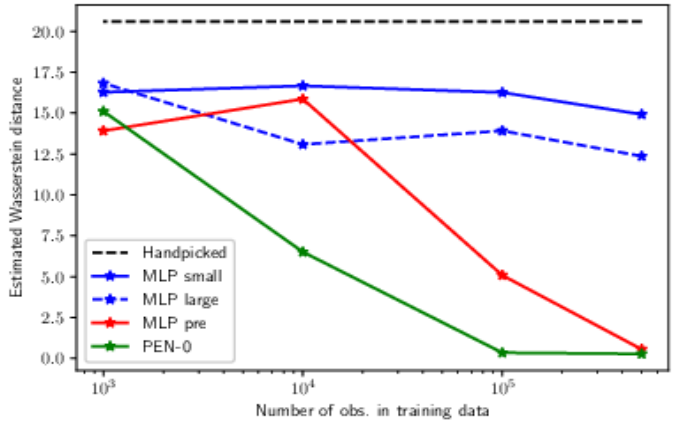


Figure 1: Results for g-and-k distribution: The estimated Wasserstein distances (mean over 100 repetitions) when comparing the MCMC posterior with ABC posteriors.

4.2 α-stable distribution

The α-stable is a heavy-tailed distribution defined by its characteristic function (see supplementary material). Its probability density function is intractable and inference is therefore challenging. Bayesian methods for the parameters can be found in e.g. Peters et al. [2012] and Ong et al. [2018]. Unknown parameters are $\theta = [\alpha, \beta, \gamma, \delta]$. We follow Ong et al. [2018] and transform the parameters:

$$\tilde{\alpha} = \log \frac{\alpha - 1.1}{2 - \alpha}, \ \tilde{\beta} = \log \frac{\beta + 1}{1 - \beta}, \ \tilde{\gamma} = \log \gamma, \text{and } \tilde{\delta} = \delta.$$

This constraints the original parameters to $\alpha \in [1.1, 2]$, $\beta \in [-1, 1]$, and $\gamma > 0$. Independent Gaussian priors and ground-truth parameters are as in Ong et al. [2018]: $\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\delta} \sim N(0, 1)$; ground-truth values for the untransformed parameters are: $\alpha = 1.5$, $\beta = 0.5$, $\gamma = 1$, and $\delta = 0$. Observations consist of $M = 1,000$ samples.

We compare methods for computing summary statistics as we did in 4.1 for the g-and-k distribution. However, since here the true posterior distribution is unavailable, we evaluate the different methods by com-

paring the root-mean square error (RMSE) between ground-truth parameter values and the ABC posterior means, see 1. From 1 we conclude that PEN-0 performs best in terms of RMSE. Similarly to the g-and-k example we also see that "MLP pre" (see 4.1 for details) performs considerably better than MLP. We now look at the resulting posteriors. In Figure 2 five posteriors from five independent experiments are presented (here we have used $5 \cdot 10^5$ training data observations). Inference results when using handpicked summary statistics are poor and for $\tilde{\gamma}$ the posterior resembles the prior. Posterior inference is worst for "MLP large". Results for "MLP pre" and PEN-0 are similar, at least in the case depicted in Figure 2 where we use $5 \cdot 10^5$ training data observations. However, in terms of RMSE, PEN-0 returns the best results when we reduce the number of training data observations.

Table 1: Results for α-stable distribution. Root-mean square error (RMSE) when comparing posterior means to the ground-truth parameters (over 25 repetitions), for different methods of computing the summary statistics, and different number of training observations (between brackets).

	Handpicked	MLP (small)	MLP (large)	MLP pre	PEN-0
RMSE ($5 \cdot 10^5$)	0.64	0.18	0.15	0.07	0.05
RMSE (10^5)	0.64	0.19	0.17	0.07	0.06
RMSE (10^4)	0.64	0.21	0.37	0.07	0.06
RMSE (10^3)	0.64	0.72	0.62	0.40	0.07

4.3 Autoregressive time series model

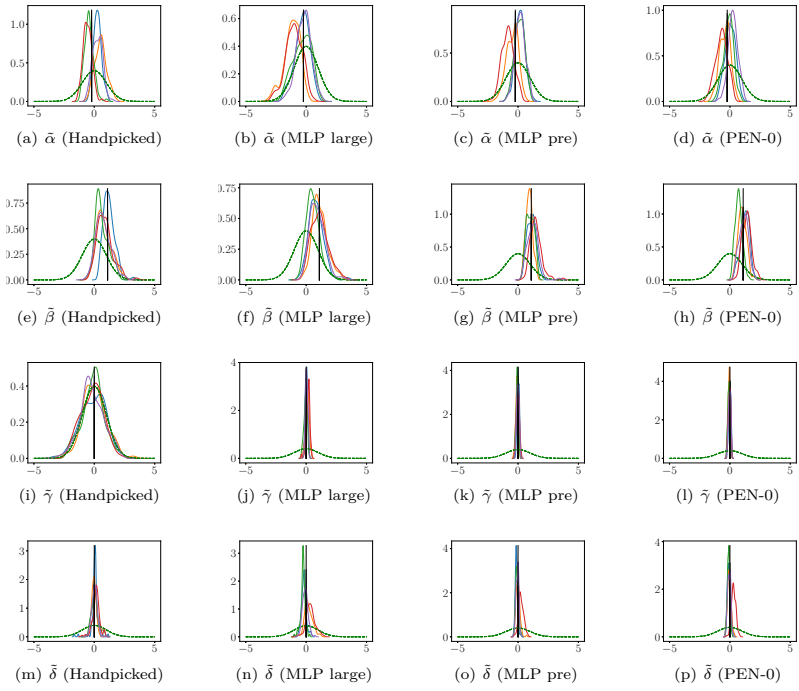
An autoregressive time series model of order two (AR(2)) follows:

$$y_l = \theta_1 y_{l-1} + \theta_2 y_{l-2} + \xi_l, \qquad \xi_l \sim N(0, 1).$$

The AR(2) model is identifiable if the following are fulfilled: $\theta_2 < 1 + \theta_1, \theta_2 < 1 - \theta_1, \theta_2 > -1$ [Fuller, 1976]. We let the resulting triangle define the uniform prior for the model. The ground-truth parameters for this simulation study are set to $\theta = [0.2, -0.13]$, and the data size is $M = 100$. AR(2) is a Markov model, hence and the requirement for PEN-d with $d > 0$ is fulfilled.

We compare five methods for computing the summaries: (i) handpicked summary statistics, i.e. $S(y) = [\gamma(y, 1), \gamma(y, 2), \gamma(y, 3), \gamma(y, 4), \gamma(y, 5)]$ ($\gamma(y, i)$ is autocovariance at lag i), which are reasonable summary statistics since autocovariances are normally employed in parameter estimation for autoregressive models, for instance when using the Yule–Walker equations; (ii) "MLP small" network; (iii) "MLP large"; (iv) PEN-0 (DeepSets); and (v) PEN-2. Since AR(2) is a time series model it makes sense to use PEN-2, and PEN-0 results are reported only in the interest of comparison. Here we do not consider the "MLP pre" method used in Section 4.1 and 4.2, since the empirical distribution function does not have any reasonable meaning for time series data. The likelihood function for AR(2) is known and we can therefore sample from the true posterior using MCMC.

Results are in Figure 3. PEN-2 outperforms MLP, for example we can see that the precision achieved when PEN-2 is trained on 10^3 training observations can be achieved by MLP when trained on 10^5 observations, implying an improvement of a 10^2 factor. Approximate and exact posteriors are in Figure 4 and we conclude that posteriors for both MLP and PEN-2 are similar to the true posterior when many training observations are used. However, the approximate posterior for MLP degrades significantly when the number of training observations is reduced and is very uninformative with 10^3 and even with 10^4 observations, while for PEN-2 the quality of the approximate posterior distribution is only marginally reduced.



(a) $\tilde{\alpha}$ (Handpicked) (b) $\tilde{\alpha}$ (MLP large) (c) $\tilde{\alpha}$ (MLP pre) (d) $\tilde{\alpha}$ (PEN-0)

(e) $\tilde{\beta}$ (Handpicked) (f) $\tilde{\beta}$ (MLP large) (g) $\tilde{\beta}$ (MLP pre) (h) $\tilde{\beta}$ (PEN-0)

(i) $\tilde{\gamma}$ (Handpicked) (j) $\tilde{\gamma}$ (MLP large) (k) $\tilde{\gamma}$ (MLP pre) (l) $\tilde{\gamma}$ (PEN-0)

(m) $\tilde{\delta}$ (Handpicked) (n) $\tilde{\delta}$ (MLP large) (o) $\tilde{\delta}$ (MLP pre) (p) $\tilde{\delta}$ (PEN-0)

Figure 2: Results for α-stable distribution: Approximate marginal ABC posteriors. Results obtained using $5 \cdot 10^5$ training data observations. The green dashed line is the prior distribution. The colored lines show posteriors from 5 independent experiments. These posteriors are not cherry-picked.

4.4 Moving average time series with observational noise model

We consider a partially observed time series, with latent dynamics given by a moving average MA(2) model and observations perturbed with Gaussian noise:

$$\begin{cases} y_l = x_l + \xi_l^y, & \xi_l^y \sim N(0, \sigma_\epsilon = 0.3), \\ x_l = \xi_l + \theta_1 \xi_{l-1}^x + \theta_2 \xi_{l-2}^x, & \xi_l^x \sim N(0, 1), \end{cases}$$

where the ξ_l^x and ξ_l^y are all independent. An MA(2) process without observational noise is identifiable if $\theta_1 \in [-2, 2]$, $\theta_2 \in [-1, 1]$, and $\theta_2 \pm \theta_1 \geq -1$. Same as in Jiang et al. [2017], we define a uniform prior over this triangle. We use the same setting as in Jiang et al. [2017] and set the ground-truth parameters for the simulation study to $\theta = [0.6, 0.2]$. We only observe $\{y_l\}$ and the number of observations is $M = 100$.

The latent dynamics are not Markovian, hence the Markov property required for PEN of order larger than 0 is not fulfilled, however, the quasi-Markov structure of the data might still allow us to successfully use PEN-d with an order d larger than 0. An additional complication is given by the observational noise ξ_l^y, further perturbing the dynamics. Once more, we compare five methods for computing the summary statistics: (i) handpicked summaries $S(y) = [\gamma(y, 1), \gamma(y, 2)]$, i.e. we follow Jiang et al. [2017]; (ii) "MLP small"; (iii) "MLP large"; (iv) PEN-0 (DeepSets); and (v) PEN-10. Same as for the AR(2) example, here PEN-0 results are reported only in the interest of a comparison with PEN-10, as for a time-series model it is expected from

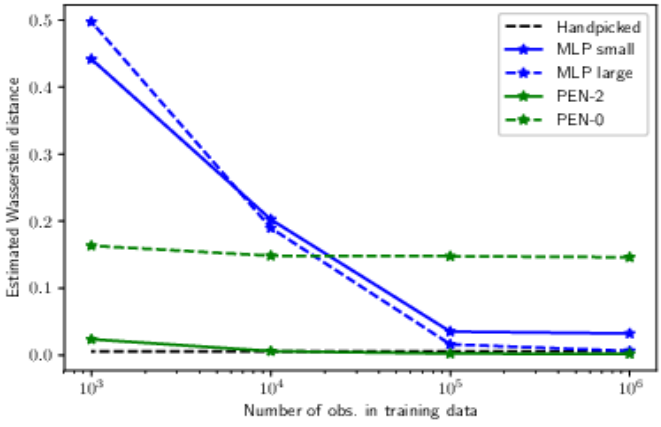


Figure 3: Results for AR(2) model: Estimated Wasserstein distances (mean over 100 data sets) when comparing the true posterior with ABC posteriors, for varying sizes of training data when using DNN models.

PEN-0 to be suboptimal. Also in this case the likelihood function is available, and we can compute the true posterior distribution. Once more, we compare the approximate posteriors to the true posterior over 100 different data sets, see Figure Figure 5. We conclude that PEN-10 performs slightly better than MLP when the training data set is large, and that PEN-10 outperforms MLP when we restrict the size of the training data. Once more, we notice that PEN-10 implies a factor ≥ 10 in terms of savings on the size of the training data.

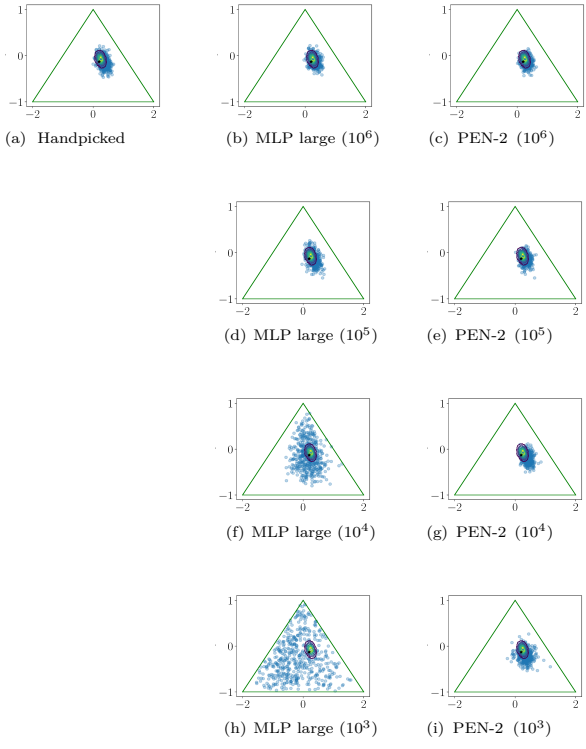
5 Discussion

Simulation experiments show that our partially exchangeable networks (PENs) achieve competitive results in learning summary statistics for use in ABC algorithms, outperforming the other deep learning methods that we have considered. Moreover, PENs require much smaller training data to achieve the same inference accuracy of competitors: in our experiments a reduction factor of order 10 to 10^2 was observed.

As mentioned in Section 2, in this work we were not focused on the specific ABC algorithm used for sampling, but only on learning summary statistics for ABC. However, in future work we plan to use our approach for constructing summary statistics alongside more sophisticated variants of ABC methods, such as those which combine ABC with Markov chain Monte Carlo [Sisson and Fan, 2011] or sequential techniques [Beaumont et al., 2009].

Murphy et al. [2019] recently shed light on some limitations of the DeepSets architecture, and proposed to improve it by replacing the sum fed to the outer network by another pooling techinque called *Janossy pooling*. Since the drawbacks they inspect are also likely to affect our architectures, extending Janossy pooling to the PEN framework might constitute a valuable improvement.

Our experiments show that the performance of the MLP networks using different choices for the number of weights is quite similar, and that PEN outperforms MLP even when MLP has access to a larger number of weights compared to PEN. The main insight is that PENs by design incorporate the (partial) exchangeability property of the data, whereas the MLPs have to learn this property. Exchangeability and partial exchange-



(a) Handpicked (b) MLP large (10^6) (c) PEN-2 (10^6)

(d) MLP large (10^5) (e) PEN-2 (10^5)

(f) MLP large (10^4) (g) PEN-2 (10^4)

(h) MLP large (10^3) (i) PEN-2 (10^3)

Figure 4: Results for AR(2) model. The green line indicates the prior distribution, the contour plot is from the exact posterior and the blue dots are 100 samples from the several ABC posteriors. The number in parenthesis indicates number of observations in the training data set. These posteriors are not cherry-picked.

ability can in principle be expressed in an MLP, but for small data sets these properties will be difficult to learn, and we expect that the model will overfit to the training data. One approach to alleviate this problem for MLPs is to perform data augmentation. However, it is not straightforward to perform data augmentation for continuous Markovian data, unless we have access to the underlying data generating process. In ABC the assumption is that we do have access to this process, but data generation may be computational expensive, and in a more general application we may not have access to the process.

Although we have applied the PEN architecture to the problem of learning summary statistics for ABC, notice that PEN is a general architecture and could be used for other applications. One example would be time series classification.

The main limitation for PEN is that it is designed for Markovian data or, when considering the special case of DeepSets (i.e. PEN-0), for exchangeable data. However, in the MA(2) example we achieve good inference results even though the MA(2) model is itself non-Markovian *and* observations are perturbed with measurement noise.

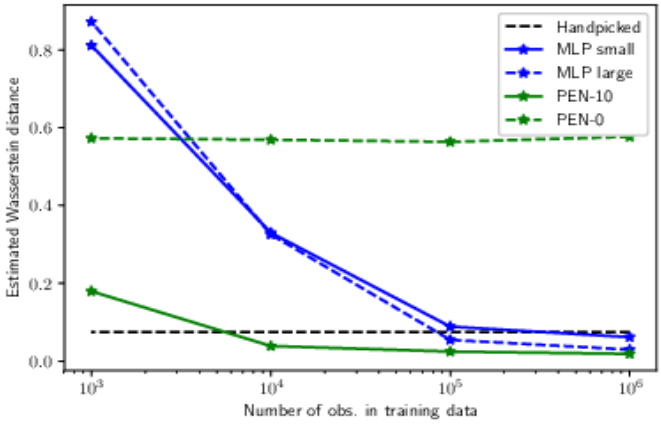


Figure 5: Results for MA(2) model: Estimated Wasserstein distances (mean over 100 data sets) when comparing the true posterior with ABC posteriors.

Acknowledgements

Research was partially supported by the Swedish Research Council (VR grant 2013-05167). We would also like to thank Joachim Hein and colleagues at LUNARC, Lund University, for helping out on setting up the GPU environment used for the simulations.

References

D. Allingham, R. King, and K. L. Mengersen. Bayesian estimation of quantile distributions. *Statistics and Computing*, 19(2):189–201, 2009.

M. A. Beaumont, W. Zhang, and D. J. Balding. Approximate Bayesian computation in population genetics. *Genetics*, 162(4):2025–2035, 2002.

M. A. Beaumont, J.-M. Cornuet, J.-M. Marin, and C. P. Robert. Adaptive approximate Bayesian computation. *Biometrika*, 96(4):983–990, 2009.

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

B. Bloem-Reddy and Y. W. Teh. Probabilistic symmetry and invariant neural networks. *arXiv:1901.06082*, 2019.

M. G. Blum, M. A. Nunes, D. Prangle, S. A. Sisson, et al. A comparative review of dimension reduction methods in approximate Bayesian computation. *Statistical Science*, 28(2):189–208, 2013.

J. Chan, V. Perrone, J. Spence, P. Jenkins, S. Mathieson, and Y. Song. A likelihood-free inference framework for population genetic data using exchangeable neural networks. In S. Bengio, H. Wallach, H. Larochelle,

K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8603–8614. Curran Associates, Inc., 2018.

J.-M. Cornuet, F. Santos, M. A. Beaumont, C. P. Robert, J.-M. Marin, D. J. Balding, T. Guillemaud, and A. Estoup. Inferring population history with DIY ABC: a user-friendly approach to approximate Bayesian computation. *Bioinformatics*, 24(23):2713–2719, 2008.

M. Creel. Neural nets for indirect inference. *Econometrics and Statistics*, 2:36–49, 2017.

B. de Finetti. Funzione caratteristica di un fenomeno aleatorio. In *Atti del Congresso Internazionale dei Matematici: Bologna dal 3 al 10 di settembre 1928*, pages 179–190, 1929.

P. Diaconis. Recent progress on de Finetti's notions of exchangeability. *Bayesian statistics*, 3:111–125, 1988.

P. Diaconis and D. Freedman. de Finetti's theorem for Markov chains. *The Annals of Probability*, pages 115–130, 1980.

P. Fearnhead and D. Prangle. Constructing summary statistics for approximate bayesian computation: semi-automatic approximate Bayesian computation. *Journal of the Royal Statistical Society: Series B*, 74(3): 419–474, 2012.

R. Flamary and N. Courty. POT Python optimal transport library, 2017. URL `https://github.com/rflamary/POT`.

W. A. Fuller. *Introduction to time series analysis*. New York: John Wiley & Sons, 1976.

A. Jasra. Approximate Bayesian computation for a class of time series models. *International Statistical Review*, 83(3):405–435, 2015.

B. Jiang, T.-y. Wu, C. Zheng, and W. H. Wong. Learning summary statistic for approximate Bayesian computation via deep neural network. *Statistica Sinica*, pages 1595–1618, 2017.

J.-M. Marin, P. Pudlo, C. P. Robert, and R. J. Ryder. Approximate Bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012.

M. Minsky and S. Papert. *Perceptrons (expanded edition) MIT Press*. 1988.

R. L. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *International Conference on Learning Representations*, 2019.

V. M. Ong, D. J. Nott, M.-N. Tran, S. A. Sisson, and C. C. Drovandi. Variational bayes with synthetic likelihood. *Statistics and Computing*, 28(4):971–988, 2018.

G. W. Peters, S. A. Sisson, and Y. Fan. Likelihood-free bayesian inference for α-stable models. *Computational Statistics & Data Analysis*, 56(11):3743–3756, 2012.

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

U. Picchini and R. Anderson. Approximate maximum likelihood estimation using data-cloning ABC. *Computational Statistics & Data Analysis*, 105:166–183, 2017.

D. Prangle. Summary statistics in approximate Bayesian computation. *arXiv:1512.05633*, 2015.

D. Prangle. gk: An R package for the g-and-k and generalised g-and-h distributions. *arXiv:1706.06889*, 2017.

J. K. Pritchard, M. T. Seielstad, A. Perez-Lezaun, and M. W. Feldman. Population growth of human Y chromosomes: a study of Y chromosome microsatellites. *Molecular biology and evolution*, 16(12):1791–1798, 1999.

S. Ravanbakhsh, J. Schneider, and B. Poczos. Deep learning with sets and point clouds. *International Conference on Learning Representations (ICLR) - workshop track*, 2017.

J. Shawe-Taylor. Building symmetries into feedforward networks. In *Artificial Neural Networks, 1989., First IEE International Conference on (Conf. Publ. No. 313)*, pages 158–162. IET, 1989.

S. A. Sisson and Y. Fan. *Handbook of Markov Chain Monte Carlo*, chapter Likelihood-free Markov chain Monte Carlo. Chapman and Hall, 2011.

S. A. Sisson, Y. Fan, and M. Beaumont. *Handbook of Approximate Bayesian Computation*. Chapman and Hall/CRC, 2018.

D. Yuret. Knet: beginning deep learning with 100 lines of julia. In *Machine Learning Systems Workshop at NIPS*, volume 2016, page 5, 2016.

M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

Supplementary Material to:

Partially Exchangeable Networks and Architectures for Learning Summary Statistics in Approximate Bayesian Computation

Contents

1 Approximate Bayesian computation rejection sampling

1.1 Settings for ABC rejection sampling "reference table" algorithm

In section 2 of the main paper we denote with x the ABC threshold. For g-and-k and α-stable models we consider for x the 0.1th percentile, and for AR(2) and MA(2) the 0.02th percentile of all distances. The number of proposals for g-and-k and α-stable models is $\tilde{N} = 100,000$, and for AR(2) and MA(2) $\tilde{N} = 500,000$.

1.2 The ABC distance function

In all our inference attempts we always used ABC rejection sampling and only needed to change the method used to compute the summary statistics. We employed the Mahalanobis distance

$$\Delta(s^*, s^{\text{obs}}) = \sqrt{(s^* - s^{\text{obs}})^\top A (s^* - s^{\text{obs}})},$$

where in our case A is the identity matrix, except when using *hand-picked* summary statistics for the g-and-k distribution, and in such case A is a diagonal matrix with diagonal elements $1/w^2$, with w a vector with entries $w = [0.22; 0.19; 0.53; 2.97; 1.90]$, as in Picchini and Anderson [2017].

2 Regularization

We use early-stopping for all networks. The early-stopping method used is to train the network over N epochs and then select the set of weights, out of the N sets, that generated the lowest evaluation error.

3 g-and-k distribution

- The full set of parameters for a g-and distribution is $[A, B, g, k, c]$, However, we follow the common practice of keeping c fixed to $c = 0.8$ and assume $B > 0$ and $k \geq 0$ Prangle [2017].
- Here is a procedure to simulate a single draw from the distribution: we first simulate a draw z from a standard Gaussian distribution, $z \sim N(0, 1)$, then we plug z into

$$Q = A + B \cdot (1 + c \cdot \tanh(g \cdot z/2)) \cdot z \cdot (1 + z^2)^k$$

  and obtain a realization Q from a g-and-k distribution.
- The network settings are presented in Table 1, 2, 3, and 4;
- The number of weights for the different networks are presented in Table 5;
- Values outside of the range $[-10, 50]$ are considered to be outliers and these values are replaced (at random) with values inside the data range. The data cleaning scheme is applied to both the observed and generated data;
- When computing the empirical distribution function we evaluate this function over 100 equally spaced points between 0 and 50;
- Number of training observations: $5 \cdot 10^5$, 10^5, 10^4, and 10^3. Evaluation data observations $5 \cdot 10^3$.

Table 1: g-and-k: Network settings for MLP small

Layer	Dim. in	Dim. out	Activation
Input	1000	25	relu
Hidden 1	25	25	relu
Hidden 2	25	12	relu
Output	12	4	linear

Table 2: g-and-k: Network settings for MLP large.

Layer	Dim. in	Dim. out	Activation
Input	1000	100	relu
Hidden 1	100	100	relu
Hidden 2	100	50	relu
Output	50	4	linear

Table 3: g-and-k: Network settings MLP pre

Layer	Dim. in	Dim. out	Activation
Input	100	100	relu
Hidden 1	100	100	relu
Hidden 2	100	50	relu
Output	50	4	linear

Table 4: g-and-k: Network settings for PEN-0

ϕ network

Layer	Dim. in	Dim. out	Activation
Input	1	100	relu
Hidden 1	100	50	relu
Output	50	10	linear

ρ network

Layer	Dim. in	Dim. out	Activation
Input	10	100	relu
Hidden 1	100	100	relu
Hidden 2	100	50	relu
Output	50	4	linear

4 α-stable distribution

- The characteristic function $\varphi(x)$ for the α-stable distribution is given by Ong et al. [2018]

Table 5: g-and-k: Number of weights for the different networks

Network	# weights
MLP small	26039
MLP large	115454
MLP pre	25454
PEN-0	22214

$$\varphi(x) = \begin{cases} \exp\Big(i\delta t - \gamma^\alpha |t|^\alpha \big(1 + i\beta \tan\tfrac{\pi\alpha}{2} \operatorname{sgn}(t)(|\gamma t|^{1-\alpha} - 1)\big)\Big), & \alpha \neq 1, \\ \exp\Big(i\delta t - \gamma|t|\big(1 + i\beta\tfrac{2}{\pi}\operatorname{sgn}(t)\log(\gamma|t|)\big)\Big), & \alpha = 1, \end{cases}$$

where sgn is the sign function, i.e.

$$\operatorname{sgn}(t) = \begin{cases} -1 & \text{if } t < 0, \\ 0 & \text{if } t = 0, \\ 1 & \text{if } t > 0. \end{cases}$$

- The network settings are presented in Table 6, 7, 8, and 9;
- The number of weights for the different networks are presented in Table 10;
- Values outside of the range $[-10, 50]$ are considered to be outliers and these values are replaced (at random) with values inside the data range. The data cleaning scheme is applied to both the observed and generated data;
- All data sets are standardized using the "robust scalar" method, i.e. each data point y_i is standardized according to

$$\frac{y_i + Q_1(y)}{Q_3(y) - Q_1(y)}$$

where Q_1 and Q_3 are the first and third quantiles respectively;

- When computing the empirical distribution function we evaluate this function over 100 equally spaced points between -10 and 100;
- The root-mean-squared error (RMSE) is computed as

$$\text{RMSE} = \sqrt{\frac{1}{R}\sum_{i=1}^{R}\{(\hat{\theta}_i^1 - \theta^1)^2 + (\hat{\theta}_i^2 - \theta^2)^2 + (\hat{\theta}_i^3 - \theta^3)^2 + (\hat{\theta}_i^4 - \theta^4)^2\}}$$

where $\theta = [\theta^1, \theta^2, \theta^3, \theta^4]$ are ground-truth parameter values and $[\hat{\theta}_i^1, \hat{\theta}_i^2, \hat{\theta}_i^3, \hat{\theta}_i^4]_{1 \leq i \leq R}$ are ABC posterior means. R is the number of independent repetitions of the inference procedure;

- Number of training observations: $5 \cdot 10^5$, 10^5, 10^4, and 10^3. Evaluation data observations $5 \cdot 10^3$.

Table 6: α-stable: Network settings for MLP small.

Layer	Dim. in	Dim. out	Activation
Input	1002	25	relu
Hidden 1	25	25	relu
Hidden 2	25	12	relu
Output	12	4	linear

Table 7: α-stable: Network settings for MLP large.

Layer	Dim. in	Dim. out	Activation
Input	1002	100	relu
Hidden 1	100	100	relu
Hidden 2	100	50	relu
Output	50	4	linear

Table 8: α-stable: Network settings MLP pre.

Layer	Dim. in	Dim. out	Activation
Input	100	100	relu
Hidden 1	100	100	relu
Hidden 2	100	50	relu
Output	50	4	linear

Table 9: α-stable: Network settings for PEN-0.

ϕ network

Layer	Dim. in	Dim. out	Activation
Input	1	100	relu
Hidden 1	100	50	relu
Output	50	20	linear

ρ network

Layer	Dim. in	Dim. out	Activation
Input	22	100	relu
Hidden 1	100	100	relu
Hidden 2	100	50	relu
Output	50	4	linear

Table 10: α-stable: Number of weights for the different networks

Network	# weights
MLP small	26089
MLP large	115654
MLP pre	25454
PEN-0	23924

5 Autoregressive time series model

- The network settings are presented in Table 11, 12, 13, and 14;
- The number of weights for the different networks are presented in Table 15;
- Number of training observations: 10^6, 10^5, 10^4, and 10^3. Evaluation data observations 10^4.

6 Moving average time series with observational noise model

- The network settings are presented in Table 16, 17, 18, and 19;
- The number of weights for the different networks are presented in Table 20;
- Number of training observations: 10^6, 10^5, 10^4, and 10^3. Evaluation data observations $5 \cdot 10^5$.

Table 11: AR(2): Network settings for MLP small.

Layer	Dim. in	Dim. out	Activation
Input	100	55	relu
Hidden 1	55	55	relu
Hidden 2	55	25	relu
Output	25	2	linear

Table 12: AR(2): Network settings for MLP large.

Layer	Dim. in	Dim. out	Activation
Input	100	100	relu
Hidden 1	100	100	relu
Hidden 2	100	50	relu
Output	50	2	linear

Table 13: AR(2): Network settings for PEN-0.

ϕ network

Layer	Dim. in	Dim. out	Activation
Input	1	100	relu
Hidden 1	100	50	relu
Output	50	10	linear

ρ network

Layer	Dim. in	Dim. out	Activation
Input	10	50	relu
Hidden 1	50	50	relu
Hidden 2	50	20	relu
Output	20	2	linear

Table 14: AR(2): Network settings for PEN-2.

ϕ network

Layer	Dim. in	Dim. out	Activation
Input	3	100	relu
Hidden 1	100	50	relu
Output	50	10	linear

ρ network

Layer	Dim. in	Dim. out	Activation
Input	12	50	relu
Hidden 1	50	50	relu
Hidden 2	50	20	relu
Output	20	2	linear

Table 15: AR(2): Number of weights for the different networks

Network	# weights
MLP small	10087
MLP large	25352
PEN-0	9922
PEN-2	10222

Table 16: MA(2): Network settings for MLP small.

Layer	Dim. in	Dim. out	Activation
Input	100	60	relu
Hidden 1	60	60	relu
Hidden 2	60	25	relu
Output	25	2	linear

Table 17: MA(2): Network settings for MLP large.

Layer	Dim. in	Dim. out	Activation
Input	100	100	relu
Hidden 1	100	100	relu
Hidden 2	100	50	relu
Output	50	2	linear

Table 18: MA(2): Network settings for PEN-0.

ρ network

Layer	Dim. in	Dim. out	Activation
Input	1	100	relu
Hidden 1	100	50	relu
Hidden 2	50	10	relu

ϕ network

Layer	Dim. in	Dim. out	Activation
Input	10	50	relu
Hidden 1	50	50	relu
Hidden 2	50	20	relu
Output	20	2	linear

Table 19: MA(2): Network settings for PEN-10

ρ network

Layer	Dim. in	Dim. out	Activation
Input	11	100	relu
Hidden 1	100	50	relu
Hidden 2	50	10	relu

ϕ network

Layer	Dim. in	Dim. out	Activation
Input	20	50	relu
Hidden 1	50	50	relu
Hidden 2	50	20	relu
Output	20	2	linear

Table 20: MA(2): Number of weights for the different networks

Network	# weights
MLP small	11297
MLP large	25352
PEN-0	9922
PEN-10	11422

References

V. M. Ong, D. J. Nott, M.-N. Tran, S. A. Sisson, and C. C. Drovandi. Variational bayes with synthetic likelihood. *Statistics and Computing*, 28(4):971–988, 2018.

U. Picchini and R. Anderson. Approximate maximum likelihood estimation using data-cloning ABC. *Computational Statistics & Data Analysis*, 105:166–183, 2017.

D. Prangle. gk: An R package for the g-and-k and generalised g-and-h distributions. *arXiv:1706.06889*, 2017.

Paper II

Wiqvist, S., Golightly, A., McLean, A. T., & Picchini, U.
Efficient inference for stochastic differential equation mixed-effects models using correlated particle pseudo-marginal algorithms
Computational Statistics & Data Analysis, 157, 107151. Elsevier.

The paper included here is a slightly typographically modified version of the paper accepted for CSDA.

Efficient inference for stochastic differential equation mixed-effects models using correlated particle pseudo-marginal algorithms

Samuel Wiqvist[a,*], Andrew Golightly[b], Ashleigh T. McLean[b], Umberto Picchini[c]

[a] *Centre for Mathematical Sciences, Lund University, Sweden*
[b] *School of Mathematics, Statistics and Physics, Newcastle University, UK*
[c] *Mathematical Sciences, Chalmers University of Technology and the University of Gothenburg, Sweden*

Abstract

Stochastic differential equation mixed-effects models (SDEMEMs) are flexible hierarchical models that are able to account for random variability inherent in the underlying time-dynamics, as well as the variability between experimental units and, optionally, account for measurement error. Fully Bayesian inference for state-space SDEMEMs is performed, using data at discrete times that may be incomplete and subject to measurement error. However, the inference problem is complicated by the typical intractability of the observed data likelihood which motivates the use of sampling-based approaches such as Markov chain Monte Carlo. A Gibbs sampler is proposed to target the marginal posterior of all parameter values of interest. The algorithm is made computationally efficient through careful use of blocking strategies and correlated pseudo-marginal Metropolis-Hastings steps within the Gibbs scheme. The resulting methodology is flexible and is able to deal with a large class of SDEMEMs. The methodology is demonstrated on three case studies, including tumor growth dynamics and neuronal data. The gains in terms of increased computational efficiency are model and data dependent, but unless bespoke sampling strategies requiring analytical derivations are possible for a given model, we generally observe an efficiency increase of one order of magnitude when using correlated particle methods together with our blocked-Gibbs strategy.

Keywords: Bayesian inference; random effects; sequential Monte Carlo; state-space model

1. Introduction

Stochastic differential equations (SDEs) are arguably the most used and studied stochastic dynamic models. SDEs allow the representation of stochastic time-dynamics, and are ubiquitous in applied research, most notably in finance [42], systems biology [49], pharmacokinetic/pharmacodynamic modelling [28] and neuronal modelling. SDEs extend the possibilities offered by ordinary differential equations (ODEs), by allowing random dynamics. As such, they can in principle replace ODEs in practical applications, to offer a richer mathematical representation for complex phenomena that are intrinsically non-deterministic. However, in practice switching from ODEs to SDEs is usually far from trivial, due to the absence of closed form solutions to SDEs (except for the simplest toy problems), implying the need for numerical approximation procedures [24]. Numerical approximation schemes, while useful for simulation purposes, considerably complicate statistical inference for model parameters. For reviews of inference strategies for SDE models, see e.g. [19] (including Bayesian approaches) and [41] (classical approaches). Generally, in the non-Bayesian framework, the literature for parametric inference approaches for SDEs is vast, however there is no inference procedure that is applicable to general nonlinear SDEs and that is also easy to implement on a computer. This is due to the lack of explicit transition densities for most SDE models. The problem is particularly difficult for measurements that are observed without error, i.e. Markovian observations. On the other hand, the Bayesian literature offers powerful solutions to the inference problem, when observations arise from state-space models. In our case, this means that if we assume that observations are observed with error, and that the latent process is a Markov process, then the literature based on sequential Monte Carlo (particle filters) is readily available in the form of pseudo-marginal methods [3], and closely related particle MCMC methods [2], which we introduce in Section 4.

*Corresponding author
Email address: samuel.wiqvist@matstat.lu.se (Samuel Wiqvist)

Our goal is to produce novel Gibbs samplers embedding special types of pseudo-marginal algorithms allowing for exact Bayesian inference in a specific class of state-space SDE models. In this paper, we consider "repeated measurement experiments", modeled via mixed-effects, where the dynamics are Markov processes expressed via stochastic differential equations. These dynamics are assumed directly unobservable, i.e. are only observable up to measurement error. The practical goal is to fit observations pertaining to several "units" (i.e. independent experiments, such as measurements on different subjects) simultaneously, by formulating a state-space model having parameters randomly varying between the several individuals. The resulting model is typically referred to as a *stochastic differential equation mixed-effects model* (SDEMEM). SDEMEMs are interesting because, in addition to explaining intrinsic stochasticity in the time-dynamics, they also take into account random variation between experimental units. The latter variation permits the understanding of between-subjects variability within a population. When considered in a state-space model, these two types of variability (population variation and intrinsic stochasticity) are separated from the third source of variation, namely residual variation (measurement error). Thanks to their generality, and the ability to separate the three levels of variation, SDEMEMs have attracted attention, see e.g. [14] for a review and [47] for a more recent account. See also section 2 for a discussion on previous literature.

In the present work, we mainly focus on a general, *plug-and-play* approach for exact Bayesian inference in SDEMEMs, meaning that analytic calculations are not necessary thanks to the flexibility of the underlying sequential Monte Carlo (SMC) algorithms. We also describe a non plug-and-play approach to handle specific situations. As in [36], our random effects and measurement error can have arbitrary distributions, provided that the measurement error density can be evaluated point-wise. Unlike [36], we use a Gibbs sampler to target the marginal parameter posterior. Subject specific, common and random effect population parameters are updated in separate blocks, with pseudo-marginal Metropolis-Hastings (PMMH) steps used to update the subject specific and common parameters, and Metropolis-Hastings (MH) steps used to update the random effect population parameters. We believe that, to date, our work results in the most general plug-and-play approach to inference for state-space SDEMEMs (a similar method has been concurrently and independently introduced (July 25 2019 on arXiv), in [5]; see the discussion in Section 6). However, the price to pay for such generality is that the use of pseudo-marginal methods guided by SMC algorithms is computationally consuming. In order to make pseudo-marginal methods scale better as the number of observations is increased, we exploit recent advances based on correlated PMMH (CPMMH). We combine CPMMH with a novel blocking strategy and show that it is possible to reduce considerably the number of required particles, and hence reduce the computational requirements for exact Bayesian inference. In our experiments, unless specific models admit bespoke efficient sampling strategies (e.g. Section 5.3 where it was possible to implement an advanced particle filter), CPMMH based algorithms with our novel blocking strategy are one order of magnitude more efficient than standard PMMH. Occasionally we even observed a 40-fold increase in efficiency, as in Section 5.1.

The remainder of this paper is organized as follows. Background literature is discussed in Section 2. Stochastic differential mixed-effects models and the inference task are introduced in Section 3. Our proposed approach to inference is described in Section 4. Applications are considered in Section 5, including a simulation study considering an Ornstein-Uhlenbeck SDEMEM, a tumor-growth model and finally a challenging neuronal data case-study. A discussion is in Section 6. Julia and R codes can be found at `https://github.com/SamuelWiqvist/efficient_SDEMEM`.

2. Background literature

Here we rapidly review key papers on inference for SDEMEMs, and refer the reader to `https://umbertopicchini.github.io/sdemem/` for a comprehensive list of publications. Early attempts at inference for SDEMEMs use methodology borrowed from standard (deterministic) nonlinear mixed-effects literature such as FOCE (first order conditional estimation) combined with the extended Kalman filter, as in [32]. This approach can only deal with SDEMEMs having a constant diffusion coefficient, see instead [29] for an extension to state-dependent diffusion coefficients. The resulting inference in [32] is approximate maximum likelihood estimation, and no uncertainty quantification is given. Moreover, only Gaussian random effects are allowed and measurement error is also assumed Gaussian. Other maximum likelihood approaches are in [33] and [34], where a closed-form series expansion for the unknown transition density is found using the method in [1], however the methodology can only be applied to reducible multivariate diffusions without measurement error. [13] discuss inference for SDEMEMs in a Bayesian framework. They implement a Gibbs sampler when the SDE (for each subject) has an explicit solution, and consider Gaussian random effects and Gaussian measurement error. When no explicit solution exists, they approximate the diffusion process

using the Euler-Maruyama approximation. The approach of [15] is of particular interest, since it is the first attempt to employ particle filters for inference in SDEMEMs: they construct an exact maximum likelihood strategy based on stochastic approximation EM (SAEM), where latent trajectories are "proposed" via particle Markov chain Monte Carlo. The major problem with using SAEM is the need for sufficient summary statistics for the "complete likelihood", which makes the methodology essentially impractical for arbitrarily complex models. [9] also use SAEM, but they avoid the need for the (usually unavailable) summary statistics for the complete likelihood, and propose trajectories using the extended Kalman filter instead of particle MCMC. Unlike in [15], the inference in [9] is approximate and measurement error and random effects are required to be Gaussian. [39] analyze multivariate diffusions under the conditions that the random effects are Gaussian distributed and that both fixed parameters and random effects enter linearly in the SDE. [48] work with the Euler-Maruyama approximation and adopt a data augmentation approach to integrate over the uncertainty associated with the latent diffusion process, by employing carefully designed bridge constructs inside a Gibbs sampler. A linear noise approximation (LNA) is also considered. However, the limitations are that the observation equation has to be a linear combination of the latent states and measurement error has to be Gaussian. In addition, producing the bridge construct in the data augmentation approach or the LNA-based likelihood requires some careful analytic derivations. Consequently, neither approach can be regarded as a plug-and-play method (that is, a method that only requires forward simulation and evaluation of the measurement error density). In [36], approximate and exact Bayesian approaches for a tumor growth study were considered: the approximate approach was based on synthetic likelihoods [50, 38], where summary statistics of the data are used for the inference, while exact inference used pseudo-marginal methodology via an auxiliary particle filter, which is suited to target measurements observed with a small error. It was found that using a particle approach to integrate out the random effects was very time consuming. Even though the data set was small (comprising 5-8 subjects to fit, depending on the experimental group, and around 10 observations per subject), the number of particles required to approximate each individual likelihood was in the order of thousands. This is very time consuming when the number of "subjects" (denoted M in the rest of this work) increases.

3. Stochastic differential mixed-effects models

Consider the case where we have M experimental units randomly chosen from a theoretical population. Our goal is to perform inference based on simultaneously fitting all data from the M units. Now assume that the experiment we are analyzing consists in observing a stochastically evolving dynamic process, and that associated with each unit i is a continuous-time d-dimensional Itô process $\{X_t^i, t \geq 0\}$ governed by the SDE

$$dX_t^i = \alpha(X_t^i, \kappa, \phi^i, D^i)\, dt + \sqrt{\beta(X_t^i, \kappa, \phi^i, D^i)}\, dW_t^i, \quad X_0^i = x_0^i, \quad i = 1, \ldots, M. \tag{1}$$

Here, α is a d-vector of drift functions, the diffusion coefficient β is a $d \times d$ positive definite matrix with a square root representation $\sqrt{\beta}$ such that $\sqrt{\beta}\sqrt{\beta}^T = \beta$, W_t^i is a d-vector of (uncorrelated) standard Brownian motion processes and D^i are unit-specific static or time-dependent deterministic input (e.g. covariates, forcing functions), see e.g. [29]. The p-vector parameter $\kappa = (\kappa_1, \ldots, \kappa_p)^T$ is common to all units whereas the q-vectors $\phi^i = (\phi_1^i, \ldots, \phi_q^i)^T$, $i = 1, \ldots, M$, are unit-specific random effects. In the most general random effects scenario we let $\pi(\phi^i|\eta)$ denote the joint distribution of ϕ^i, parameterised by the r-vector $\eta = (\eta_1, \ldots, \eta_r)^T$. The model defined by (1) allows for differences between experimental units through different realizations of the Brownian motion paths W_t^i and the random effects ϕ^i, accounting for inherent stochasticity within a unit, and variation between experimental units respectively.

We assume that each experimental unit $\{X_t^i, t \geq 0\}$ cannot be observed exactly, but observations $y^i = (y_1^i, \ldots, y_n^i)^T$ are available. Without loss of generality, we assume units are observed at the same integer time points $\{1, 2, ..., n\}$, that is in the following we write n instead of, say, n_i for all i. However this is only for convenience of notation, and we could easily accommodate the possibility of different units i having different values n_i and that, in turn, units are observed at different sets of times. The observations are assumed conditionally independent (given the latent process) and we link them to the latent process via

$$Y_t^i = h(X_t^i, S^i, \epsilon_t^i), \qquad \epsilon_t^i | \xi \overset{indep}{\sim} p_\epsilon(\xi), \qquad i = 1, ..., M \tag{2}$$

where Y_t^i is a d_o-vector, ϵ_t is a random d_o-vector, $d_o \leq d$, ϵ_t^i is the measurement noise, S^i is (as D^i) a unit-specific deterministic input, and $h(\cdot)$ is a possibly nonlinear function of its arguments. In the applications in Section 5 we have $D^i = S^i = \emptyset$, the empty set, for every i, and hence for simplicity of notation we disregard

D^i and S^i in the rest of the paper. However having non-empty sets does not introduce any additional complication to our methodology. Notice, the possibility to have $d_0 < d$ implies that we may have some coordinate of the $\{X_t^i\}$ system that is unobserved at some (or all) t. We denote the density linking Y_t^i and X_t^i by $\pi(y_t^i|x_t^i, \xi)$. An important special case that arises from our flexible observation model is when $h(X_t^i, \epsilon_t^i) = F^T X_t^i + \epsilon_t^i$ for a constant matrix F and $\epsilon_t^i|\Sigma \overset{indep}{\sim} N(0, \Sigma)$, allowing for observation of a linear combination of components of X_t^i, subject to additive Gaussian noise. Notice that our methodology in Sections 3.1–4.4 can be applied to an arbitrary $h(\cdot)$, provided this can be evaluated pointwise for any value of its arguments. For example, in Section 5.2 we have that $h(\cdot)$ is the logarithm of the sum of the components of a bivariate X_t^i.

We refer to the model constituted by the system (1)-(2) as a SDEMEM. This is a state-space model, due to the Markov property of the Itô processes $\{X_t^i, t \geq 0\}$, and the assumption of conditional independence of observations on latent processes. The model is flexible: equation (1) explains the intrinsic stochasticity in the dynamics (via β) and the variation between-units (via the random effects ϕ^i), while (2) explains residual variation (measurement error, via ξ).

3.1. Bayesian inference

Denote with $x = (x^1, \ldots, x^M)^T$ the set of unobserved states collected across all M diffusion processes $\{X_t^i\}$ at the same set of integer times $\{1, 2, ..., n\}$ as for data $y = (y^1, \ldots, y^M)^T$. Then given data $y = (y^1, \ldots, y^M)^T$, latent values x, the joint posterior for the common parameters κ, fixed/random effects $\phi = (\phi^1, \ldots, \phi^M)^T$, hyperparameters η and measurement error parameters ξ is

$$\pi(\kappa, \eta, \xi, \phi, x|y) \propto \pi(\kappa, \eta, \xi)\pi(\phi|\eta)\pi(x|\kappa, \phi)\pi(y|x, \xi) \tag{3}$$

where $\pi(\kappa, \eta, \xi)$ is the joint prior density ascribed to κ, η and ξ. These three parameters may be assumed *a priori* independent, and then we can write $\pi(\kappa, \eta, \xi) = \pi(\kappa)\pi(\eta)\pi(\xi)$, though this needs not be the case and we can easily assume *a priori* correlated parameters. In addition we have that

$$\pi(\phi|\eta) = \prod_{i=1}^{M} \pi(\phi^i|\eta), \tag{4}$$

$$\pi(y|x, \xi) = \prod_{i=1}^{M} \prod_{j=1}^{n} \pi(y_j^i|x_j^i, \xi) \tag{5}$$

and

$$\pi(x|\kappa, \phi) = \prod_{i=1}^{M} \pi(x_1^i) \prod_{j=2}^{n} \pi(x_j^i|x_{j-1}^i, \kappa, \phi^i). \tag{6}$$

Note that $\pi(x_j^i|x_{j-1}^i, \kappa, \phi^i)$ will be typically intractable. In this case, we assume that it is possible to generate draws (up to arbitrary accuracy) from $\pi(x_j^i|x_{j-1}^i, \kappa, \phi^i)$ using a suitable numerical approximation. For example, the Euler-Maruyama approximation of (1) is

$$\Delta X_t^i \equiv X_{t+\Delta t}^i - X_t^i = \alpha(X_t^i, \kappa, \phi^i)\,\Delta t + \sqrt{\beta(X_t^i, \kappa, \phi^i)}\,\Delta W_t^i$$

and therefore

$$X_{t+\Delta t}^i = X_t^i + \alpha(X_t^i, \kappa, \phi^i)\,\Delta t + \sqrt{\beta(X_t^i, \kappa, \phi^i)}\,\Delta W_t^i \tag{7}$$

where $\Delta W_t^i \sim N(0, I_d \Delta t)$ and the time-step Δt, which need not be the inter-observation time, is chosen by the practitioner to balance accuracy and efficiency.

In what follows, we assume that interest lies in the marginal posterior for all parameters, given by $\pi(\kappa, \eta, \xi, \phi|y) = \int \pi(\kappa, \eta, \xi, \phi, x|y)dx$, where

$$\pi(\kappa, \eta, \xi, \phi|y) \propto \pi(\kappa)\pi(\eta)\pi(\xi)\pi(\phi|\eta)\pi(y|\kappa, \xi, \phi) \tag{8}$$

$$\propto \pi(\kappa)\pi(\eta)\pi(\xi) \prod_{i=1}^{M} \pi(\phi^i|\eta)\pi(y^i|\kappa, \xi, \phi^i). \tag{9}$$

This factorization suggests a Gibbs sampler with separate blocks for each parameter vector that sequentially takes draws from the full conditionals

1. $\pi(\phi|\kappa,\eta,\xi,y) \propto \prod_{i=1}^{M} \pi(\phi^i|\eta)\pi(y^i|\kappa,\xi,\phi^i)$,
2. $\pi(\kappa|\eta,\xi,\phi,y) = \pi(\kappa|\phi,\xi,y) \propto \pi(\kappa)\prod_{i=1}^{M} \pi(y^i|\kappa,\xi,\phi^i)$,
3. $\pi(\xi|\kappa,\eta,\phi,y) = \pi(\xi|\kappa,\phi,y) \propto \pi(\xi)\prod_{i=1}^{M} \pi(y^i|\kappa,\xi,\phi^i)$,
4. $\pi(\eta|\kappa,\xi,\phi,y) = \pi(\eta|\phi) \propto \pi(\eta)\prod_{i=1}^{M} \pi(\phi^i|\eta)$.

Of course, in practice, the observed individual data likelihood $\pi(y^i|\kappa,\xi,\phi^i) = \int p(y^i,x^i|\kappa,\xi,\phi^i)dx^i$ will be intractable. In what follows, therefore, we consider a Metropolis-within-Gibbs strategy, and in particular introduce auxiliary variables u to allow pseudo-marginal Metropolis-Hastings updates.

4. A pseudo-marginal approach

Consider again the intractable target in (8) and suppose that we can unbiasedly estimate the intractable observed data likelihood $\pi(y|\kappa,\xi,\phi) = \int p(y,x|\kappa,\xi,\phi)dx$. To this end let

$$\hat{\pi}_u(y|\kappa,\xi,\phi) = \prod_{i=1}^{M} \hat{\pi}_{u^i}(y^i|\kappa,\xi,\phi^i)$$

denote a (non-negative) unbiased estimator of $\pi(y|\kappa,\xi,\phi)$, where $u = (u^1,\ldots,u^M)^T$ is the collection of auxiliary (vector) variables used to produce the corresponding estimate, with density $\pi(u) = \prod_{i=1}^{M} g(u^i)$. In the context of inference for SDEs, the u may be the collection of pseudo-random standard Gaussian draws, these being necessary to simulate increments of the Brownian motion paths when implementing a numerical scheme such as Euler-Maruyama (Section 4.2), or produce draws from transition densities (in the rare instances when these are known). Notice in fact that the u need not have a specific distribution, though in stochastic simulation we need access to pseudo-random variates that are often uniform or Gaussian distributed [11]. When inference methods use particle filters, pseudo-random variates are also employed in the resampling step, and hence these variates can be included into u.

Now, the pseudo-marginal Metropolis-Hastings (PMMH) scheme targets

$$\pi(\kappa,\eta,\xi,\phi,u|y) \propto \pi(\kappa)\pi(\eta)\pi(\xi)\pi(\phi|\eta)\hat{\pi}_u(y|\kappa,\xi,\phi)\pi(u) \tag{10}$$

for which it is easily checked that

$$\begin{aligned}\int \pi(\kappa,\eta,\xi,\phi,u|y)du &\propto \pi(\kappa)\pi(\eta)\pi(\xi)\pi(\phi|\eta)\int \hat{\pi}_u(y|\kappa,\xi,\phi)\pi(u)du \\ &\propto \pi(\kappa,\eta,\xi,\phi|y).\end{aligned}$$

Hence, marginalising out u gives the marginal parameter posterior in (8). Directly targeting the high dimensional posterior $\pi(\kappa,\eta,\xi,\phi,u|y)$ with PMMH is likely to give very small acceptance rates. The structure of the SDMEM naturally admits a Gibbs sampling strategy. We outline our novel Gibbs samplers in the next section.

4.1. Gibbs sampling and blocking strategies

The form of (10) immediately suggests a Gibbs sampler that sequentially takes draws from the full conditionals. However, we can design two types of Gibbs samplers. Our first, novel strategy is denoted "naive Gibbs", where the u^i are updated with both the subject specific and common parameters.

Naive Gibbs:

1. $\pi(\phi^i,u^i|\kappa,\eta,\xi,y^i) \propto \pi(\phi^i|\eta)\hat{\pi}_{u^i}(y^i|\kappa,\xi,\phi^i)g(u^i)$, $i = 1,\ldots,M$,
2. $\pi(\kappa,u|\eta,\xi,\phi,y,u) = \pi(\kappa,u|\phi,\xi,y) \propto \pi(\kappa)\prod_{i=1}^{M} \hat{\pi}_{u^i}(y^i|\kappa,\xi,\phi^i)g(u^i)$,
3. $\pi(\xi,u|\kappa,\eta,\phi,y,u) = \pi(\xi,u|\kappa,\phi,y) \propto \pi(\xi)\prod_{i=1}^{M} \hat{\pi}_{u^i}(y^i|\kappa,\xi,\phi^i)g(u^i)$,
4. $\pi(\eta|\kappa,\xi,\phi,y,u) = \pi(\eta|\phi) \propto \pi(\eta)\prod_{i=1}^{M} \pi(\phi^i|\eta)$.

Note that step 1 consists of a set of draws of M conditionally independent random variables since

$$\pi(\phi,u|\kappa,\eta,\xi,y) = \prod_{i=1}^{M} \pi(\phi^i,u^i|\kappa,\eta,\xi,y^i).$$

Hence, step 1 gives a sample from $\pi(\phi, u|\kappa, \eta, \xi, y)$. Draws from the full conditionals in 1-3 can be obtained by using Metropolis-Hastings within Gibbs. Taking the $[\phi^i, u^i]$ block as an example, we use a proposal density of the form $q(\phi^{i*}|\phi^i)g(u^{i*})$ and accept a move from $[\phi^i, u^i]$ to $[\phi^{i*}, u^{i*}]$ with probability

$$\min\left\{1,\ \frac{\pi(\phi^{i*}|\cdot)}{\pi(\phi^i|\cdot)} \times \frac{\hat{\pi}_{u^{i*}}(y^i|\phi^{i*}, \cdot)}{\hat{\pi}_{u^i}(y^i|\phi^i, \cdot)} \times \frac{q(\phi^i|\phi^{i*})}{q(\phi^{i*}|\phi^i)}\right\}.$$

Effectively, samples from the full conditionals in 1-3 are obtained via draws from pseudo-marginal MH kernels.

The above strategy is somewhat naive, since the auxiliary variables u need only be updated once per Gibbs iteration, instead in steps 1 to 3 of the naive Gibbs procedure vectors u^i are simulated anew in each of the three steps (notice $g(u^i)$ appears in each of the first three steps). We therefore propose to update the blocks $[\phi^i, u^i]$, $i = 1, \ldots, M$ in step 1 only, and condition on the most recent value of u in the remaining steps. We call this second, novel strategy "blocked Gibbs".

Blocked Gibbs:

1. $\pi(\phi^i, u^i|\kappa, \eta, \xi, y^i) \propto \pi(\phi^i|\eta)\hat{\pi}_{u^i}(y^i|\kappa, \xi, \phi^i)g(u^i)$, $i = 1, \ldots, M$,
2. $\pi(\kappa|\eta, \xi, \phi, y, u) = \pi(\kappa|\phi, \xi, y, u) \propto \pi(\kappa) \prod_{i=1}^M \hat{\pi}_{u^i}(y^i|\kappa, \xi, \phi^i)$,
3. $\pi(\xi|\kappa, \eta, \phi, y, u) = \pi(\xi|\kappa, \phi, y, u) \propto \pi(\xi) \prod_{i=1}^M \hat{\pi}_{u^i}(y^i|\kappa, \xi, \phi^i)$,
4. $\pi(\eta|\kappa, \xi, \phi, y, u) = \pi(\eta|\phi) \propto \pi(\eta) \prod_{i=1}^M \pi(\phi^i|\eta)$.

The aim of blocking in this way is to reduce the variance of the acceptance probability associated with steps 2 and 3, which involve the product of M estimates as opposed to a single estimate in each constituent part of step 1. Also, notice $g(u^i)$ appears only in the first step. The effect of blocking in this way is explored empirically in Section 5.

4.2. Estimating the likelihood

It remains that we can generate non-negative unbiased estimates $\hat{\pi}_u(y|\kappa, \xi, \phi)$. This can be achieved by running a sequential Monte Carlo procedure, also known as particle filter. The simplest approach is to use the bootstrap particle filter [43, 22] (see also [25]) that, for a single experimental unit, recursively draws from the filtering distribution $\pi(x_t^i|y_{1:t}^i, \kappa, \xi, \phi^i)$ for each $t = 1, \ldots, n$. Here, $y_{1:t}^i$ denotes the observations of experiment i for time-steps $1, \ldots, t$. Essentially, a sequence of importance sampling and resampling steps are used to propagate a weighted sample $\{(x_{t,k}^i, w(u_{t,k}^i)), k = 1, \ldots, N_i\}$ from the filtering distribution, where N_i is the number of particles for unit i. Note that we let the weight depend explicitly on the t-th component of the auxiliary variable $u^i = (u_1^i, \ldots, u_n^i)$, associated with experimental unit i. At time t, the particle filter uses the approximation

$$\hat{\pi}(x_t^i|y_{1:t}^i, \kappa, \xi, \phi^i) \propto \pi(y_t^i|x_t^i, \xi) \sum_{k=1}^{N_i} \pi(x_t^i|x_{t-1,k}^i, \kappa, \phi^i) w(u_{t-1,k}^i). \tag{11}$$

A simple importance sampling/resampling strategy follows, where particles are resampled (with replacement) in proportion to their weights, propagated via $x_{t,k}^i = f_t(u_{t,k}^i) \sim \pi(\cdot|x_{t-1,k}^i, \kappa, \phi^i)$ and reweighted by $p(y_t^i|x_{t,k}^i, \xi)$. Here, $f_t(\cdot)$ is a deterministic function of $u_{t,k}^i$ (as well as the parameters and previous latent state, suppressed for simplicity) that gives an explicit connection between the particles and auxiliary variables. An example of $f_t(\cdot)$ is to take the Euler-Maruyama approximation

$$f_t(u_{t,k}^i) = x_{t-1,k}^i + \alpha(x_{t-1,k}^i, \kappa, \phi^i)\,\Delta t + \sqrt{\beta(x_{t-1,k}^i, \kappa, \phi^i)\Delta t} \times u_{t,k}^i$$

where $u_{t,k}^i \sim N(0, I_d)$ and Δt is a suitably chosen time-step. In practice, unless Δt is sufficiently small to allow an accurate Euler-Maruyama approximation, $f_t(u_{t,k}^i)$ will describe recursive application of the numerical approximation.

Algorithm 1 provides a complete description of the bootstrap particle filter when applied to a single experimental unit. However notice the addition of a non-standard and optional sorting step 2b', which turns useful when implementing a correlated pseudo-marginal approach, as described in Section 4.3. For the resampling step we follow [10] among others and use systematic resampling (see e.g. [31]), which only requires simulating a single uniform random variable at each time point. It is straightforward to augment

Algorithm 1 Bootstrap particle filter for experimental unit i

Input: parameters κ, ϕ^i, ξ, auxiliary variables u^i, data y^i and the number of particles N_i.
Output: estimate $\hat{\pi}_{u^i}(y^i|\kappa,\xi,\phi^i)$ of the observed data likelihood.

1. Initialisation ($t=1$).
   (a) **Sample** the prior. Put $x^i_{1,k} = f_1(u^i_{1,k}) \sim \pi(\cdot)$, $k=1,\ldots,N_i$.
   (b) **Compute** the weights. For $k=1,\ldots,N_i$ set
   $$\tilde{w}(u^i_{1,k}) = \pi(y^i_1|x^i_{1,k},\xi), \qquad w(u^i_{1,k}) = \frac{\tilde{w}(u^i_{1,k})}{\sum_{j=1}^{N_i}\tilde{w}(u^i_{1,j})}.$$
   (c) **Update** observed data likelihood estimate. Compute $\hat{\pi}_{u^i_1}(y^i_1|\kappa,\xi,\phi^i) = \sum_{k=1}^{N_i}\tilde{w}(u^i_{1,k})/N_i$.
2. For times $t=2,3,\ldots,n$:
   (b') **(optional) Sorting.** Use Euclidean sorting on particles $\{x^i_{t-1,1},...,x^i_{t-1,N_i}\}$ if using CPMMH.
   (b) **Resample.** Obtain ancestor indices a^k_{t-1}, $k=1,\ldots,N_i$ using systematic resampling on the collection of weights $\{w(u^i_{t-1,1}),\ldots,w(u^i_{t-1,N_i})\}$.
   (c) **Propagate.** Put $x^i_{t,k} = f_t(u^i_{t,k}) \sim \pi\big(\cdot\,|x^i_{t-1,a^k_{t-1}},\kappa,\xi,\phi^i\big)$, $k=1,\ldots,N_i$.
   (d) **Compute** the weights. For $k=1,\ldots,N_i$ set
   $$\tilde{w}(u^i_{t,k}) = \pi(y^i_t|x^i_{t,k},\xi), \qquad w(u^i_{t,k}) = \frac{\tilde{w}(u^i_{t,k})}{\sum_{j=1}^{N_i}\tilde{w}(u^i_{t,j})}.$$
   (e) **Update** observed data likelihood estimate. Compute
   $$\hat{\pi}_{u^i_{1:t}}(y^i_{1:t}|\kappa,\xi,\phi^i) = \hat{\pi}_{u^i_{1:t-1}}(y^i_{1:t-1}|\kappa,\xi,\phi^i)\hat{\pi}_{u^i_t}(y^i_t|y^i_{1:t-1},\kappa,\xi,\phi^i)$$
   where $\hat{\pi}_{u^i_t}(y^i_t|y^i_{1:t-1},\kappa,\xi,\phi^i) = \sum_{k=1}^{N_i}\tilde{w}(u^i_{t,k})/N_i$.

the auxiliary variable u^i to include the random variables used in the resampling step. As a by-product of the particle filter, the observed data likelihood $\pi(y^i|\kappa,\xi,\phi^i)$ can be estimated via the quantity

$$\hat{\pi}_{u^i}(y^i|\kappa,\xi,\phi^i) = N_i^{-n}\prod_{t=1}^{n}\sum_{k=1}^{N_i}\tilde{w}(u^i_{t,k}). \tag{12}$$

Moreover, the corresponding estimator can be shown to be unbiased [8, 37].

The full Gibbs sampler for generating draws from the joint posterior (10) is given by Algorithm 2. For ease of exposition, we have blocked the updates for κ and ξ, but note that the use of separate updates for these parameters is straightforward. The precise implementation of step 4 of the Gibbs sampler is likely to be example specific, and we anticipate that a direct draw of $\eta^{(j)} \sim \pi(\cdot|\phi^{(j)})$ will often be possible. For example when the components of ϕ are assumed to be normally distributed and η consists of the corresponding means and precisions, for which a semi-conjugate prior specification is possible, see Section 5.1.

Executing Algorithm 2 requires $n\sum_{i=1}^{M}N_i$ draws from the transition density governing the SDE in (1) *per iteration*. In scenarios where the transition density is intractable, draws of a suitable numerical approximation are required. For example, we may use the Euler-Maruyama discretisation with time step $\Delta t = 1/m$, where $m \geq 1$ is chosen to limit the associated discretisation bias (and typically $m \gg 1$). In this case, order $mn\sum_{i=1}^{M}N_i$ draws of (7) are required. As discussed by [2], the number of particles per experimental unit, N_i, should be scaled in proportion to the number of data points n. Consequently, the use of PMMH kernels is likely to be computationally prohibitive in practice. We therefore consider the adaptation of a recently proposed correlated PMMH method for our problem.

4.3. A correlated pseudo-marginal approach

Consider again the task of sampling the full conditional $\pi(\phi^i,u^i|\kappa,\eta,\xi,y^i)$ associated with the ith experimental unit. In steps 2(a–c) of Algorithm 2, a (pseudo-marginal) Metropolis-Hastings step is used whereby the auxiliary variables u^i are proposed from the associated pdf $g(\cdot)$ (notice we could introduce a subject-specific $g_i(\cdot)$, but we refrain from doing so in the interest of a lighter notation). As discussed by [10] (see also [7]), the proposal kernel need not be restricted to the use of $g(u^i)$. The correlated PMMH (CPMMH)

Algorithm 2 Blocked Gibbs sampler

Input: Data y, initial parameter values ϕ, κ, ξ, η and number of iterations n_{iters}.
Output: $\{\phi^{(j)}, \kappa^{(j)}, \xi^{(j)}, \eta^{(j)}\}_{j=1}^{n_{\text{iters}}}$.

1. Initialise $\phi^{(0)} = (\phi^{1,(0)}, \ldots, \phi^{M,(0)})$, $\kappa^{(0)}$, $\xi^{(0)}$. Draw $u^{i,(0)} \sim g(\cdot)$ and run Algorithm 1 for $i = 1, \ldots, M$ with $u^{i,(0)}$, $\phi^{i,(0)}, \kappa^{(0)}, \xi^{(0)}$ and y^i to obtain $\hat{\pi}_{u^{i,(0)}}(y^i|\kappa^{(0)}, \xi^{(0)}, \phi^{i,(0)})$. Set the iteration counter $j = 1$.
2. Update subject specific parameters. For $i = 1, \ldots, M$:
   (a) Propose $u^{i*} \sim g(\cdot)$ and $\phi^{i*} \sim q(\cdot|\phi^{i,(j-1)})$.
   (b) Compute $\hat{\pi}_{u^{i*}}(y^i|\kappa^{(j-1)}, \xi^{(j-1)}, \phi^{i*})$ by running Algorithm 1 with u^{i*}, ϕ^{i*}, $\kappa^{(j-1)}$, $\xi^{(j-1)}$ and y^i.
   (c) With probability
$$\min\left\{1\,,\ \frac{\pi(\phi^{i*}|\eta)}{\pi(\phi^{i,(j-1)}|\eta)} \times \frac{\hat{\pi}_{u^{i*}}(y^i|\kappa^{(j-1)}, \xi^{(j-1)}, \phi^{i*})}{\hat{\pi}_{u^{i,(j-1)}}(y^i|\kappa^{(j-1)}, \xi^{(j-1)}, \phi^{i,(j-1)})} \times \frac{q(\phi^{i,(j-1)}|\phi^{i*})}{q(\phi^{i*}|\phi^{i,(j-1)})}\right\} \tag{13}$$
   put $\phi^{i,(j)} = \phi^{i*}$ and $u^{i,(j)} = u^{i*}$. Otherwise, store the current values $\phi^{i,(j)} = \phi^{i,(j-1)}$ and $u^{i,(j)} = u^{i,(j-1)}$.
3. Update common parameters.
   (a) Propose $(\kappa^*, \xi^*) \sim q(\cdot|\kappa^{(j-1)}, \xi^{(j-1)})$.
   (b) Compute $\hat{\pi}_{u^{(j)}}(y|\kappa^*, \xi^*, \phi^{(j)}) = \prod_{i=1}^{M} \hat{\pi}_{u^{i,(j)}}(y^i|\kappa^*, \xi^*, \phi^{i,(j)})$ by running Algorithm 1 for $i = 1, \ldots, M$ with $u^{i,(j)}$, $\phi^{i,(j)}$, κ^*, ξ^* and y^i.
   (c) With probability
$$\min\left\{1\,,\ \frac{\pi(\kappa^*)\pi(\xi^*)}{\pi(\kappa^{(j-1)})\pi(\xi^{(j-1)})} \times \frac{\hat{\pi}_{u^{(j)}}(y|\kappa^*, \xi^*, \phi^{(j)})}{\hat{\pi}_{u^{(j)}}(y|\kappa^{(j-1)}, \xi^{(j-1)}, \phi^{(j)})} \times \frac{q(\kappa^{(j-1)}, \xi^{(j-1)}|\kappa^*, \xi^*)}{q(\kappa^*, \xi^*|\kappa^{(j-1)}, \xi^{(j-1)})}\right\} \tag{14}$$
   put $(\kappa^{(j)}, \xi^{(j)}) = (\kappa^*, \xi^*)$. Otherwise, store the current values $(\kappa^{(j)}, \xi^{(j)}) = (\kappa^{(j-1)}, \xi^{(j-1)})$.
4. Update random effect population parameters. Draw $\eta^{(j)} \sim \pi(\cdot|\phi^{(j)})$.
5. If $j = n_{\text{iters}}$, stop. Otherwise, set $j := j + 1$ and go to step 2.

scheme generalises the PMMH scheme by generating a new u^{i*} from $K(u^{i*}|u^i)$ where $K(\cdot|\cdot)$ satisfies the detailed balance equation

$$g(u^i)K(u^{i*}|u^i) = g(u^{i*})K(u^i|u^{i*}). \tag{15}$$

It is then straightforward to show that a MH scheme with proposal kernel $q(\phi^{i*}|\phi^i)K(u^{i*}|u^i)$ and acceptance probability (13) satisfies detailed balance with respect to the target $\pi(\phi^i, u^i|\kappa, \eta, \xi, y^i)$.

We take $g(u^i)$ as a standard Gaussian density and $K(u^{i*}|u^i)$ as the kernel associated with a Crank–Nicolson proposal [10]. Hence

$$g(u^i) = \mathrm{N}\left(u^i;\, 0\,,\, I_d\right) \qquad \text{and} \qquad K(u^{i*}|u^i) = \mathrm{N}\left(u^{i*};\, \rho u^i\,,\, \left(1-\rho^2\right) I_d\right)$$

where I_d is the identity matrix whose dimension d is determined by the number of elements in u^i. The parameter ρ is chosen to be close to 1, to induce strong positive correlation between $\hat{\pi}_{u^i}(y^i|\kappa, \Sigma, \phi^i)$ and $\hat{\pi}_{u^{i*}}(y^i|\kappa, \Sigma, \phi^{i*})$, thus reducing the variance of the acceptance probability in (13), which is beneficial because it reduces the chance of accepting an overestimation of the likelihood function. Taking $\rho = 0$ gives the special case that $K(u^{i*}|u^i) = g(u^{i*})$, which corresponds to the standard PMMH. Iteration j of step 2 of Algorithm 2 then becomes

2. For $i = 1, \ldots, M$:
   (a) Propose $\phi^{i*} \sim q(\cdot|\phi^{i,(j-1)})$. Draw $\omega \sim \mathrm{N}(0, I_d)$ and put $u^{i*} = \rho u^{i,(j-1)} + \sqrt{1-\rho^2}\omega$.
   (b) Compute $\hat{\pi}_{u^{i*}}(y^i|\kappa^{(j-1)}, \xi^{(j-1)}, \phi^{i*})$ by running Algorithm 1 with u^{i*}, ϕ^{i*}, $\kappa^{(j-1)}$, $\xi^{(j-1)}$ and y^i.
   (c) With probability given by (13) put $\phi^{i,(j)} = \phi^{i*}$ and $u^{i,(j)} = u^{i*}$. Otherwise, store the current values $\phi^{i,(j)} = \phi^{i,(j-1)}$ and $u^{i,(j)} = u^{i,(j-1)}$.

Care must be taken here when executing Algorithm 1 in Step 2(b). Upon changing ϕ^i and u^i, the effect of the resampling step is likely to prune out different particles, thus breaking the correlation between successive estimates of observed data likelihood. Sorting the particles before resampling can alleviate this problem [10]. We follow [6] (see also [20]) by using a simple Euclidean sorting procedure which, for the case of a 1-dimensional latent state (e.g. when $\dim(X_t^i) = 1$ for every t) implies, prior to resampling the particles, to sort the particles from the smallest to the largest. This is step 2b' in algorithm 1, denoted "optional" as it only applies to CPMMH, not PMMH.

4.4. Tuning the number of particles for likelihood approximation

It remains that we can choose the number of particles N_i to be used to obtain estimates of the observed data likelihood contributions $\hat{\pi}_{u^i}(y^i|\kappa, \xi, \phi^i)$. Note that we allow a different number of particles per experimental unit to accommodate differing lengths of the y^i and potential model misspecification at the level of an individual unit. In the case of PMMH, a simple strategy is to fix ϕ^i, κ and ξ at some central posterior value (obtained from a pilot run), and choose N_i so that the variance of the log-likelihood (denoted $\sigma^2_{N_i}$) is around 2 [16, 40]. When using a CPMMH kernel, we follow [45, 6] by choosing N_i so that $\sigma^2_{N_i} = 2.16^2/(1-\rho_l^2)$ where ρ_l is the estimated correlation between $\log \hat{\pi}_{u^i}(y^i|\kappa, \xi, \phi^i)$ and $\log \hat{\pi}_{u^{i*}}(y^i|\kappa, \xi, \phi^i)$. Hence, an initial pilot run (with the number of particles set at some conservative value) is required to determine plausible values of the parameters. This pilot run can also be used to give estimates of $var(\phi^i|y^i)$, $i = 1, \ldots, M$, each of which can subsequently be used as the innovation variance in a Gaussian random walk proposal for ϕ^i.

4.5. Tuning the proposal distributions

The block structure of the Gibbs sampler (Algorithm 2) requires two proposal densities: $\phi^{i*} \sim q(\cdot|\phi^{i,(j-1)})$ and $(\kappa^*, \xi^*) \sim q(\cdot|\kappa^{(j-1)}, \xi^{(j-1)})$ that have to be chosen to achieve an algorithm that efficiently explores the posterior parameter space.

In Sections 5.1 and 5.3 we employ the generalized Adaptive Metropolis (AM) algorithm [4] to tune the two proposal distributions. Regarding the generation of proposals ϕ^{i*}, in the first step of the blocked Gibbs scheme we tune subject-specific proposal distributions, separately for each ϕ^{i*}. In addition to these M proposal distributions we also tune a proposal distribution for (κ^*, ξ^*). Thus, we automatically tune overall $M+1$ proposal distributions via the generalized AM algorithm. Additionally, in Sections 5.1 and 5.3 we found that the use of different proposal distributions for each ϕ^{i*} was beneficial since random effects for the different subjects varied around very different values.

5. Applications

5.1. Ornstein-Uhlenbeck SDEMEM

We consider the following Ornstein-Uhlenbeck (OU) SDEMEM

$$\begin{cases} Y_t^i &= X_t^i + \epsilon_t^i, \quad \epsilon_t^i \overset{indep}{\sim} \mathrm{N}(0, \sigma_\epsilon^2), \qquad i = 1, ..., M \\ dX_t^i &= \theta_1^i(\theta_2^i - X_t^i)dt + \theta_3^i dW_t^i. \end{cases} \tag{16}$$

Here $\theta_2^i \in \mathbb{R}$ is the stationary mean for the $\{X_t^i\}$ process, $\theta_1^i > 0$ is a growth rate (expressing how rapidly the system reacts to perturbations) and θ_3^i is the diffusion coefficient. The OU process is a standard toy-model in that it is completely tractable, that is the associated SDE has a known (Gaussian) transition density, e.g. [19]. This fact, coupled with the assumption that the $Y_t^i|X_t^i$ are conditionally Gaussian and linear in the latent states, implies that we can apply the Kalman filter to evaluate the likelihood function exactly. Therefore, exact inference is possible for the OU SDEMEM (both maximum likelihood and Bayesian). For all units i we simulate $n = 200$ observations, with constant observational time-step Δt. In our setup, all random effects $(\theta_1^i, \theta_2^i, \theta_3^i)$ are assumed strictly positive, and therefore we work with their log-transformed version and set $\phi^i = (\log\theta_1^i, \log\theta_2^i, \log\theta_3^i)$, where

$$\phi_j^i|\eta \overset{indep}{\sim} N(\mu_j, \tau_j^{-1}), \qquad j = 1, 2, 3$$

and $\eta = (\mu_1, \mu_2, \mu_3, \tau_1, \tau_2, \tau_3)$, with τ_j the precision of ϕ_j^i. The SDEMEM (16) has no parameters κ that are shared among subjects, and the full set of parameters that we want to infer is $(\mu_1, \mu_2, \mu_3, \tau_1, \tau_2, \tau_3, \sigma_\epsilon)$.

As already mentioned, we can compute the likelihood $\pi(y|\phi, \sigma_\epsilon) = \prod_{i=1}^{M} \pi(y^i|\phi^i, \sigma_\epsilon)$ exactly, using a Kalman filter (see [44] and [14] for a description pertaining SDEMEMs). The filter can then be used in Algorithm 2, that is we avoid using the particle filter (Algorithm 1) and replace it with the Kalman filter in Algorithm 2. Results from Algorithm 2 when using this approach are denoted with "Kalman". The transition density for the latent state is known and therefore we do not need to use an Euler-Maruyama discretization when propagating the states forward in the particle filter. Instead we propagate the particles using the simulation scheme induced by the exact transition density:

$$X_{t+\Delta t}^i = \theta_2^i + (X_t^i - \theta_2^i)e^{-\theta_1^i \Delta t} + \sqrt{\frac{\theta_3^{i2}}{2\theta_1^i}(1 - e^{-2\theta_1^i \Delta t})} \times u_t^i, \tag{17}$$

where $u_t^i \sim N(0,1)$ independently for all t and all i. Clearly, the u_t^i appearing in (17) are among the variates that we will correlate, when implementing CPMMH, in addition to the variates produced in the resampling steps.

We compare "Kalman" to four further methods: "naive PMMH", where we employ Algorithm 2 with the naive Gibbs scheme (see Section 4.1), "PMMH", which is Algorithm 2, "CPMMH-099", which is Algorithm 2 with a Crank-Nicolson proposal for the u^i using a correlation of $\rho = 0.99$, and "CPMMH-0999" where we use a correlation of $\rho = 0.999$. The number of particle used for each method was selected using the methods described in Section 4.4. All five methods return exact Bayesian inference, and while this is obvious for "Kalman", we remind the reader that this holds also for the other four approaches as these are instances of the pseudo-marginal approach. Therefore, special interest is in efficiency comparisons between the last four algorithms, "Kalman" being the obvious gold-standard.

We simulated data from the model in (16) with the following settings (data are in Figure 1): $M = 40$ experimental units, $n = 200$ observations for each unit using a time step $\Delta t = 0.05$, $\sigma_\epsilon = 0.3$, and $\eta = (\mu_1, \mu_2, \mu_3, \tau_1, \tau_2, \tau_3) = (-0.7, 2.3, -0.9, 4, 10, 4)$. The prior for the observational noise standard deviation σ_ϵ was set to a Gamma distribution $Ga(1, 0.4)$, and the priors for the η parameters were set to

$$\begin{cases} \mu_j|\tau_j \overset{indep}{\sim} \mathrm{N}(\mu_{0_j}, M_{0_j}\tau_j), & j = 1,2,3, \\ \tau_j \overset{indep}{\sim} Ga(\alpha_j, \beta_j), \end{cases} \tag{18}$$

where,

$$\begin{aligned} (\mu_{0_1}, M_{0_1}, \alpha_1, \beta_1) &= (0,1,2,1), \\ (\mu_{0_2}, M_{0_2}, \alpha_2, \beta_2) &= (1,1,2,0.5), \\ (\mu_{0_3}, M_{0_3}, \alpha_3, \beta_3) &= (0,1,2,1). \end{aligned}$$

The priors in (18) are semi-conjugate and we can therefore use a tractable Gibbs step to sample η in step 4 of Algorithm 2. An extended introduction to the semi-conjugate prior, including the tractable posterior can be found in [30].

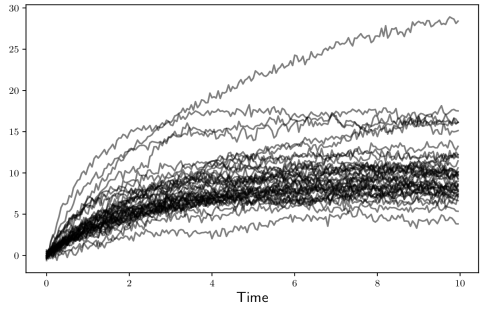


Figure 1: Simulated data from the OU-SDEMEM model.

We ran all four methods for 60k iterations, considering the first 10k iterations to be the burn-in period. We set the starting value for σ_ϵ at $\sigma_{\epsilon_0} = 0.2$, which is far from its ground truth value. The starting values for the random effects ϕ_j^i were set to their prior means. The proposal distributions were adaptively tuned using the generalized AM algorithm and the particle filters were implemented on a single-core computer, thus no parallelization was utilized. We used the same number of particles $N_i \equiv N$ for all units. Results are in Table 1 and Figures 2-3. As a reference for the efficiency of the considered samplers, we take the minimum ESS per minute (mESS/m in Table 1) as measured on PMMH-naive as "base/default" value and set it to 1 in the rightmost column of Table 1. The minimum ESS per minute for the other samplers are relative to the PMMH-naive value. The mESS value is computed over all parameter chains (including individual

random effects), i.e. the chains for ϕ, σ_ϵ and η. From Table 1 we conclude that CPMMH is about 20 to 40 times more efficient than PMMH in terms of mESS/m, depending on which correlation level we use. Furthermore, "Kalman" is about 5140 times more efficient than PMMH. However, the latter comparison is not very interesting since the Kalman filter can be applied only to a very restricted class of models. The marginal posteriors in Figure 2–3 show that the several methods generate very similar posterior inference, which is reassuring. We left out the inference results from CPMMH-0999 for reasons of clarity. However we observed that with $N = 50$ CPMMH-0999 produces a slightly biased inference for σ_ϵ, due to failing to adequately mix over the auxiliary variable u, while inference for the remaining parameters is similar to the other considered methods. We verified (results not shown) that using $N = 100$ is enough to repair this problem. From Figure 2–3 we can conclude that all parameters, with the possible exclusion of τ_2, are well inferred. Regarding τ_2, this is the precision for θ_2^i, the latter representing the stationary mean for a OU model. Clearly, by looking at Figure 1, the occasional outlier in the upper part of the Figure may contribute to underestimating the true precision of the stationary mean. To check if CPMMH indeed is necessary, we tried to run PMMH with 100 particles (i.e., the same number of particles as for CPMMH-099). The inference results produced with PMMH with 100 particles gave considerable mismatch (in terms of posterior output) for both the η parameters and σ_ϵ relative to that obtained from CPMMH-099, resulting from the extremely poor mixing of the chain.

In summary, CPMMH is able to return reliable inference with a much smaller number of particles than PMMH, while resulting in a procedure that is about 20 to 40 times more efficient than PMMH (the 40-times figure is valid if we are ready to accept a small bias in σ_ϵ). Again, for most models exact inference based on a closed-form expression for the likelihood function is unavailable, therefore being able to obtain accurate inference using a computationally cheaper version of PMMH is very appealing.

Notice that while for this simple case study PMMH-naive has the same mESS than PMMH, this is not the case for the case study in Section 5.2, where using the blocked-Gibbs sampler produces a much larger mESS value compared to naive-Gibbs.

Algorithm	ρ	N	CPU (m)	mESS	mESS/m	Rel.
Kalman	-	-	1.23	443.27	357.61	5140.18
PMMH-naive	0	3000	4601.87	229.01	0.05	1.00
PMMH	0	3000	4086.91	232.94	0.06	1.16
CPMMH-099	0.99	100	200.37	234.54	1.17	23.58
CPMMH-0999	0.999	50	110.88	235.63	2.13	41.48

Table 1: OU SDEMEM. Correlation ρ, number of particles N, CPU time (in minutes m), minimum ESS (mESS), minimum ESS per minute (mESS/m) and relative minimum ESS per minute (Rel.) as compared to PMMH-naive. All results are based on 50k iterations of each scheme, and are medians over 5 independent runs of each algorithm on different data sets. We could only produce 5 runs due to the very high computational cost of PMMH.

5.1.1. Investigating the choice of number of particles

A crucial problem when running methods based on particle filters is the selection of the number of particles N. In this section we investigate this problem by running CPMMH-099 and CPMMH-0999 with $N = [5, 10, 20, 50, 100]$ particles using 25 different (simulated) data sets. We also ran the Kalman algorithm using the 25 different data sets for comparison purposes. In this analysis, we are only interested in investigating the quality and computational efficiency of the inference. Hence, we initialised all algorithms at the ground truth parameter values and ran each algorithm for 60k iterations, and discarding the first 10k iterations as burnin period. We first estimated the Wasserstein distance, between the marginal posteriors for σ_ϵ and η from the CPMMH algorithms and the corresponding Kalman-based marginal posteriors. This distance was computed via the POT package [18] (we do not compute the Wasserstein distance for the marginal posterior of the random effects ϕ^i, since this is not of central interest for us). All Wasserstein distances are based on the last 5k samples of the corresponding chains. To obtain a performance measure that takes into account both the quality of the inference and the computational effort, we multiply the Wasserstein distances by the runtimes (in minutes) of the CPMMH algorithms, and obtain the performance measure *Wasserstein distance* $\times$ *runtime (m)*; see Figure 4 and 5. Smaller values of this measure are to be preferred as they indicate high computational efficiency and/or accurate inference. The reason for considering this performance measure is to take the quality of the inference into account, since for $N < 20$ we noticed that it is possible to obtain chains that do not indicate adequate convergence within a reasonable time-frame.

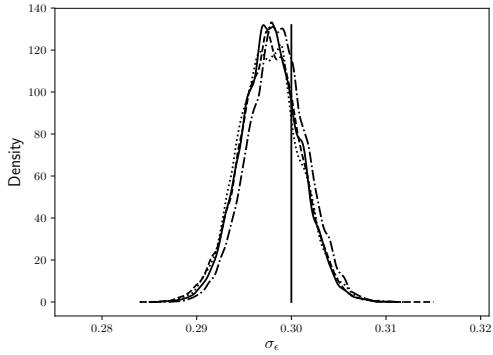


Figure 2: OU SDEMEM: marginal posterior distributions for σ_ϵ. Solid line is Kalman, dashed line is PMMH-naive, dotted line is PMMH, dash-dotted line is CPMMH-099, vertical line is the ground truth.

We can conclude that, on average, results for different correlation levels are similar. However, for σ_ϵ we obtain a better performance when using more particles (lower *Wasserstein distance* × *runtime (m)* value), this resulting from inaccurate inference for σ_ϵ when using too few ($N < 50$) particles, leading to a large Wasserstein distance. However, this is not the case for η since Figure 5 shows that the performance is better with fewer particles, a result that we obtain since the inference for η is good even when using few particles (though not reported, in our analyses we observed that the Wasserstein distances for η are similar across all attempted values of N). Thus, if we want to infer the measurement noise parameter σ_ϵ accurately, in this case we will have to use $N \geq 50$ particles, while the inference for η is satisfactory, even with fewer particles.

Another issue that we analyse is the variability of mESS for the different data sets, based on 50k iterations of CPMMH. To investigate this we computed the 25th and 75th percentiles of mESS for CPMMH-099 with $N = 100$ and CPMMH-0999 with $N = 50$ based on the inference results on all unknown parameters from 25 simulated data sets. We obtain that the 25th and 75th percentiles of mESS for CPMMH-099 ($N = 100$) are $[227, 240]$, and for CPMMH-0999 ($N = 50$) are $[227, 252]$. Given that the several mESS are computed on different datasets, some degree of variation in the measure is expected and we conclude that the observed mESS variability is fairly small.

5.2. Tumor growth SDEMEM

We consider a stochastic differential mixed effects model that has been used to describe the tumor volume dynamics in mice receiving a treatment. Here we study a simplified version of the model in [36], and is given by

$$\begin{aligned} dX^i_{1,t} &= \left(\beta^i + (\gamma^i)^2/2\right) X^i_{1,t} dt + \gamma^i X^i_{1,t} dW^i_{1,t} \\ dX^i_{2,t} &= \left(-\delta^i + (\psi^i)^2/2\right) X^i_{2,t} dt + \psi^i X^i_{2,t} dW^i_{2,t} \end{aligned} \tag{19}$$

for experimental units $i = 1, \ldots, M$. Here, $W_{1,t}$ and $W_{2,t}$ are uncorrelated Brownian motion processes, $X^i_{1,t}$ and $X^i_{2,t}$ are respectively the volume of surviving tumor cells and volume of cells killed by a treatment for mouse i. Let $V^i_t = X^i_{1,t} + X^i_{2,t}$ denote the total tumor volume at time t in mouse i. The observation model is given by

$$Y^i_t = \log V^i_t + \epsilon^i_t, \qquad \epsilon^i_t \overset{indep}{\sim} \mathrm{N}(0, \sigma^2_\epsilon). \tag{20}$$

Let $\phi^i = (\log \beta^i, \log \gamma^i, \log \delta^i, \log \psi^i)$. We complete the SDEMEM specification via the assumption that

$$\phi^i_j | \eta \overset{indep}{\sim} \mathrm{N}(\mu_j, \tau_j^{-1}), \qquad j = 1, \ldots, 4 \tag{21}$$

so that $\eta = (\mu_1, \ldots, \mu_4, \tau_1, \ldots, \tau_4)$.

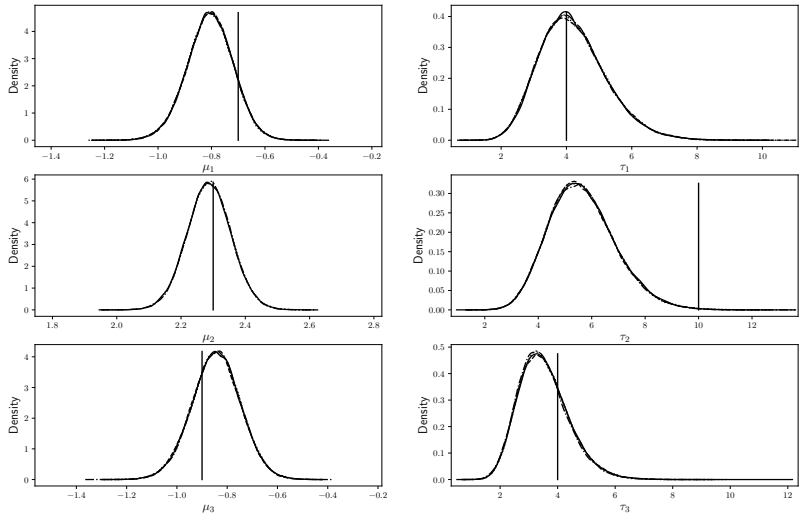


Figure 3: OU SDEMEM: marginal posterior distributions for $\eta = (\mu_1, \mu_2, \mu_3, \tau_1, \tau_2, \tau_3)$. Solid line Kalman, dashed line PMMH-naive, dotted line PMMH, dash-dotted line CPMMH-099, vertical line ground truth.

We recognise that $X^i_{1,t}$ and $X^i_{2,t}$ are geometric Brownian motion processes and (19) can be solved analytically to give

$$\begin{aligned} X^i_{1,t}|X^i_{1,0} = x^i_{1,0} &\sim logN\left(\log(x^i_{1,0}) + \beta^i t\,,\, (\gamma^i)^2 t\right) \\ X^i_{2,t}|X^i_{2,0} = x^i_{2,0} &\sim logN\left(\log(x^i_{2,0}) - \delta^i t\,,\, (\psi^i)^2 t\right) \end{aligned} \tag{22}$$

where $logN(\cdot,\cdot)$ denotes the log-Normal distribution. Despite the availability of a closed form solution to the underlying SDE model, the observed data likelihood is intractable, due to the nonlinear form of (20) as a function of $\log(X^i_{1,t} + X^i_{2,t})$. Nevertheless, a tractable approximation can be found, by linearising $\log V^i_t$. The resulting linear noise approximation (LNA) is derived in Appendix B, and in what follows, we compare inference under the gold standard PMMH to that obtained under the LNA.

We mimicked the real data application in [36] by generating 21 observations at integer times for $M = 10$ replicates. We took

$$\eta = (\log 0.29, \log 0.25, \log 0.09, \log 0.34, 10, 10, 10, 10)$$

and sampled $\phi^i_j|\eta$ using (21). The latent SDE process was then generated using (22) with an initial condition of $x^i_0 = (75, 75)^T$ (assumed known for all units), and each observation was corrupted according to (20) with $\sigma^2_e = 0.2$. The resulting data traces are consistent with the observations on total tumor volume of those subjects receiving chemo therapy in [36] and can be seen in Figure 6. We adopted semi conjugate, independent $\mathrm{N}(-2, 1)$ and $Ga(2, 0.2)$ priors for the μ_j and τ_j respectively. We took $\log \sigma_e \sim \mathrm{N}(0, 1)$ to complete the prior specification. Given the use of synthetic data of equal length for each experimental unit, we pragmatically took the number of particles as $N_i = N$, $i = 1, \ldots, 10$. Our choice of N was guided by the tuning advice of Section 4.4. For example, with CPMMH we obtain typical ρ_L values of around 0.75, when parameter values are fixed at an estimate of the posterior mean. This gives $\sigma^2_N = 10.6$ which is achieved with $N = 7$ particles. To avoid potentially sticky behaviour of the chain in the posterior tails, we choose the conservative value $N = 10$. We compare four approaches: naive PMMH (where the u^i are updated with

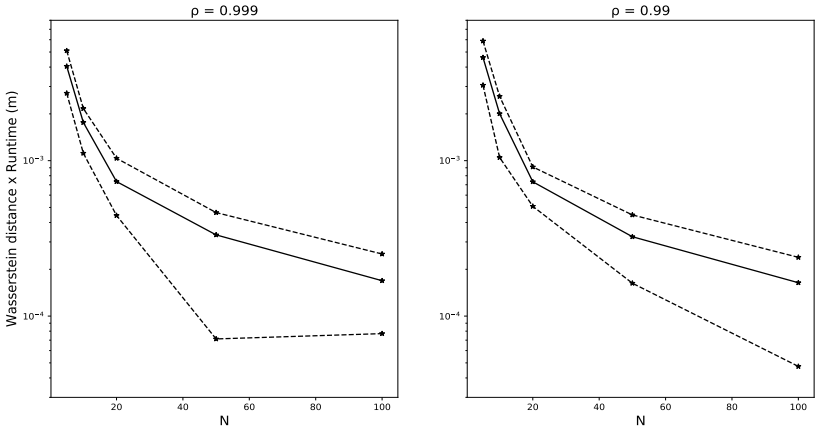


Figure 4: OU SDEMEM: *Wasserstein distance* × *runtime (m)* performance measure for the marginal posterior of σ_ϵ, for several values of N and using $\rho = 0.999$ (left) and $\rho = 0.99$ (right). The solid line represents the mean value obtained from the 25 different data sets. The dashed confidence bands represent the 25th and 75th percentiles.

both the subject specific and common parameters), PMMH (where the u^i are only updated with the subject specific parameters – Algorithm 2), CPMMH (Algorithm 2 with a Crank-Nicolson proposal on the u^i) and the LNA based approach. We ran each scheme for $500k$ iterations. The results are summarised in Table 2 and Figure 7.

Algorithm	ρ	N	CPU (m)	mESS	mESS/m	Rel.
LNA	-	-	1286	3676	2.858	13
PMMH - naive	0	30	3098	665	0.215	1
PMMH	0	30	2963	2559	0.864	4
CPMMH	0.999	10	957	2311	2.415	11

Table 2: Tumor model. Correlation ρ, number of particles N, CPU time (in minutes m), minimum ESS (mESS), minimum ESS per minute (mESS/m) and relative minimum ESS per minute (Rel.) as compared to PMMH-naive. All results are based on 500k iterations of each scheme.

Figure 7 shows marginal posterior densities of the components of η. We see that inferences for these parameters are consistent with the true values that generated the data (with similar results obtained for the other parameters) and that inference via CPMMH is consistent with that from the gold-standard PMMH. Similar results are obtained for σ_ϵ (not shown for brevity). At the same time, from Table 2 we note that CPMMH with $\rho = 0.999$ is about 11 times more efficient than the naive PMMH and almost 3 times more efficient than PMMH with additional blocking. Finally, the LNA-based approach provides an accurate alternative to PMMH, except for τ_4. However, everything considered, CPMMH is to be preferred here as its computational efficiency is comparable to LNA, but unlike the latter, CPMMH provides accurate inference for all parameters, and unlike LNA the CPMMH approach is plug-and-play.

5.2.1. Use of the Euler-Maruyama approximation

We anticipate that for many applications of interest, an analytic solution of the underlying SDE will not be available. It is common place to use a numerical approximation in place of an intractable analytic solution. The simplest such approximation is the Euler-Maruyama (E-M) approximation. In this section, we investigate the effect of the E-M on the performance of PMMH and CPMMH for the tumor growth model.

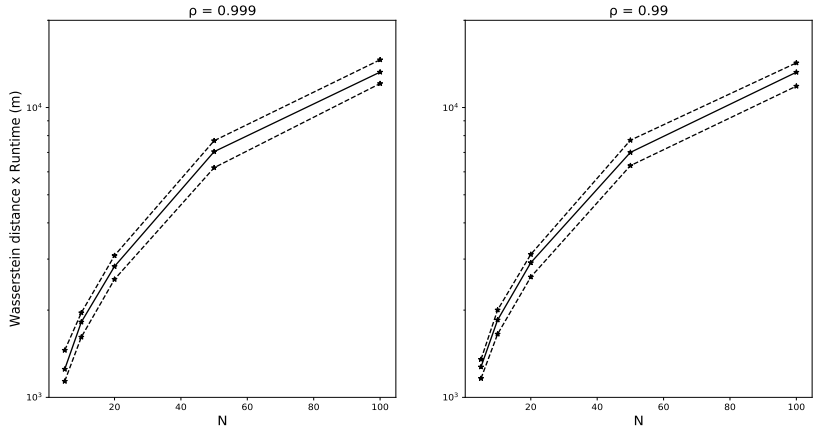


Figure 5: OU SDEMEM: *Wasserstein distance × runtime (m)* performance measure for the marginal posterior of η, for several values of N and using $\rho = 0.999$ (left) and $\rho = 0.99$ (right). The solid line represents the mean value obtained from the 25 different data sets. The dashed confidence bands represent the 25th and 75th percentiles.

The Euler-Maruyama approximation of (19) is

$$\Delta X^i_{1,t} = \left(\beta^i + (\gamma^i)^2/2\right) X^i_{1,t}\Delta t + \gamma^i X^i_{1,t}\Delta W^i_{1,t}$$
$$\Delta X^i_{2,t} = \left(-\delta^i + (\psi^i)^2/2\right) X^i_{2,t}\Delta t + \psi^i X^i_{2,t}\Delta W^i_{2,t}$$

where, for example, $\Delta X^i_{1,t} = X^i_{1,t+\Delta t} - X^i_{1,t}$ and $\Delta W^i_{1,t} \sim N(0, \Delta t)$, with other terms defined similarly. To allow arbitrary accuracy of E-M, the inter-observation time length Δt is replaced by a stepsize $\Delta t = 1/L$ for the numerical integration, for integer $L \geq 1$. We find that using $L = 5$ (giving 4 intermediate times between observation instants) allows sufficient accuracy (compared to the analytic solution) to permit use of the same tuning choices when re-running PMMH (including the naive scheme) and CPMMH. Our findings are summarised by Table 3.

Algorithm	ρ	N	CPU (m)	mESS	mESS/m	Rel.
PMMH - naive	0	30	7947	990	0.123	1
PMMH	0	30	7651	2240	0.293	2.4
CPMMH	0.999	10	1893	2172	1.15	9.2

Table 3: Tumor model (Euler-Maruyama). Correlation ρ, number of particles N, CPU time (in minutes m), minimum ESS (mESS), minimum ESS per minute (mESS/m) and relative minimum ESS per minute (Rel.) as compared to PMMH-naive. All results are based on 500k iterations of each scheme.

Unsurprisingly, inspection of Table 3 reveals that relative performance between the three computing pseudo-marginal schemes is similar to that obtained when using the analytic solution; CPMMH provides almost an order of magnitude increase in terms of mESS/m over a naive PMMH approach. We note that use of the Euler-Maruyama approximation requires computation and storage of an additional $1/\Delta t$ innovations per SDE component, inter-observation interval, particle and subject, thus accounting for the increase in CPU time compared to when using the analytic solution. Nevertheless, we find that our proposed approach is able to accommodate an intractable SDE scenario and provides a worthwhile increase in performance over competing approaches.

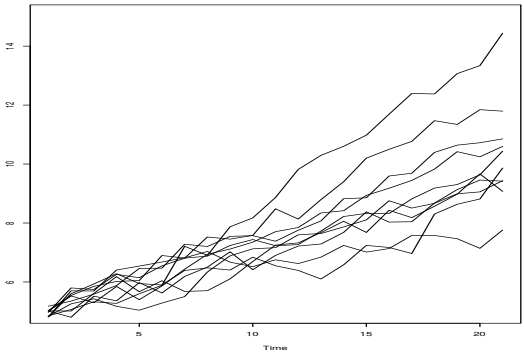


Figure 6: Simulated data from the tumor growth model.

5.2.2. Comparison with ODEMEM

To highlight the potential issues that arise by ignoring inherent stochasticity, we consider inference for an ordinary differential equation mixed effects model (ODEMEM) of tumor growth. We take the SDEMEM in (19) and set $\gamma^i = \psi^i = 0$ to give

$$\begin{aligned} dx^i_{1,t} &= \beta^i x^i_{1,t} dt, \\ dx^i_{2,t} &= -\delta^i x^i_{2,t} dt \end{aligned} \tag{23}$$

for $i = 1, \ldots, M$. The observation model and random effects distributions remain unchanged from (20) and (21) upon omitting $\log \gamma^i$ and $\log \psi^i$ from ϕ^i. The ODE system in (23) can be solved to give

$$x^i_{1,t} = x^i_{1,0} \exp\{\beta^i t\}, \qquad x^i_{1,t} = x^i_{1,0} \exp\{\delta^i t\}.$$

The likelihood associated with each experimental unit is then obtained simply as

$$\pi(y^i|\phi^i, \sigma_e) = \prod_{t=1}^{21} \mathrm{N}\left(y^i_t; \log(x^i_{1,t} + x^i_{2,t}), \sigma^2_e\right).$$

Fitting the ODEMEM to the synthetic data set from Section 5.2 is straightforward, via a Metropolis-within-Gibbs scheme. Figures 8 and 9 summarise our findings. Unsurprisingly, since the ODEMEM is unable to account for intrinsic stochasticity, the observation standard deviation is massively over-estimated. Figure 8 shows little agreement between the marginal posteriors under the ODEMEM and SDEMEM for this parameter. In terms of model fit, both the observation (Y^1_t) and latent process ($X^1_t = \log V^1_t$) predictive distributions for unit 1 are over concentrated for the ODEMEM. Similar results (not shown) are obtained for the other experimental units. Notably, from Figure 9, around half of the actual simulated X_t values lie outside of the 95% credible interval under the ODEMEM.

5.3. Neuronal data

Here we consider a much more challenging problem: modelling a large number of observations pertaining neuronal data. In particular, we are interested in modelling the neuronal membrane potential across inter-spike intervals (ISIs). The problem of modelling the membrane potential from ISIs measurements using SDEs has already been considered numerous times, also using SDEMEMs, see [35]. In fact here we analyze the same data considered in [27] and [35], or actually a subset thereof, due to computational constraints. The "leaky integrate-and-fire" appears to be one of the most common models, in both artificial neural network applications and descriptions of biological systems. Deterministic and stochastic implementations of the model are possible. In the stochastic version, under specific assumptions [26], it coincides with the Ornstein-Uhlenbeck stochastic process and has been extensively investigated in the neuronal context, for

instance in [12]. Consider Figure 10 as an illustrative example, reporting values of neuronal membrane depolarization studied in [23]. Inter-spike-intervals are the observations considered between "firing" times of the neuron, the latter being represented by the spikes appearing in Figure 10 (notice these are not the data we analysed. This figure is only used for illustration). Data corresponding to the near-deterministic spikes are removed, and what is left constitutes data from several ISIs. As in [35], we consider data from different ISIs as independent. Hence, M is the number of considered ISIs. These are 312 in total, however, because of computational limitations, we will only analyze a subset of 100 ISIs, hence our results are based on $M = 100$ and a total of 162,610 observations. A challenge is posed by the fact that some ISIs are much longer than others (in our case they vary between 600 and 2,500 observations), meaning that longer ISIs could typically require a larger N to avoid particle depletion, but using the same large N to approximate all M likelihood terms would be a waste of computational resources. This is why CPMMH comes particularly useful, as it allows to keep a small N across all units while still avoiding sticky behaviour in the MCMC chains. Data from the 100 ISIs are plotted on a common time-scale in Figure 11 (after some translation to let each ISI start approximately at zero value at time zero). These consist of membrane potentials measured every 0.15 msec intracellularly from the auditory system of a guinea pig (for details on data acquisition and processing, see [51]).

Outside the mixed-effects context, if we denote the neuronal input with ν, and if the neuron is supposed to operate in a stationary state during some time of interest, then ν would be assumed constant during this period. [35] generalize by assuming that in addition to ν there is a random component changing from one ISI to the next, which could be caused by the naturally occurring variations of environment signaling, by experimental irregularities or by other sources of noise not included in the model. This fact can then be modeled by assuming that each ISI has its own input ν^i, and [35] specifically assume that the ν^i are iid Gaussian distributed with mean ν. An extension of the model in [35] is the following state-space type SDEMEM

$$\begin{cases} Y_t^i &= X_t^i + \epsilon_t^i, \quad \epsilon_t^i \overset{indep}{\sim} \mathrm{N}(0, \sigma_\epsilon^2), \qquad i = 1, ..., M, \\ dX_t^i &= (-\lambda^i X_t^i + \nu^i)dt + \sigma^i dW_t^i. \end{cases} \tag{24}$$

where the diffusion process $\{X_t^i; t \geq 0\}$ models the membrane potential [mV] in the ith ISI, with input ν^i [mV/msec]. The spontaneous voltage decay (in the absence of input) for the ith ISI is $(\lambda^i)^{-1}$ [msec], which means that the stationary mean for $\{X_t^i\}$ is ν^i/λ^i, see e.g. [12] for details. The diffusion coefficients σ^i have unit [mV/$\sqrt{\text{msec}}$]. Clearly, we assume that we are unable to observe $\{X_t^i\}$ directly, and instead can only observe a noisy realization from $\{Y_t; t \geq 0\}$. Differences with the SDEMEM in [35] are that: (i) their observations were assumed unaffected by measurement noise, i.e. observations were directly available from $\{X_t^i; t \geq 0\}$, $i = 1, ..., M$, which is a convenient assumption easing calculations towards obtaining exact maximum likelihood estimation, but that it is generally possible to argue against; (ii) in [35] the only random effect was ν^i, and remaining parameters were fixed-effects, while in the present case we have random effects λ^i and σ^i in addition to ν^i. Of course here we also need to estimate σ_ϵ, which was not done in [35] since no measurement error was assumed.

As in Section 5.1 the random effects are constrained to be positive and we therefore define $\phi^i = (\phi_1^i, \phi_2^i, \phi_3^i) = (\log \lambda^i, \log \nu^i, \log \sigma^i)$, where

$$\phi_j^i | \eta \overset{indep}{\sim} \mathrm{N}(\mu_j, \tau_j^{-1}), \qquad j = 1, 2, 3,$$

and $\eta = (\mu_1, \mu_2, \mu_3, \tau_1, \tau_2, \tau_3)$, with τ_j the precision of ϕ_j^i. Since we here have a similar setting as in Section 5.1, we employ the same semi-conjugate priors with hyperparameters

$$\begin{aligned} (\mu_{0_1}, M_{0_1}, \alpha_1, \beta_1) &= (\log(0.1), 1, 2, 1), \\ (\mu_{0_2}, M_{0_2}, \alpha_2, \beta_2) &= (\log(1.5), 1, 2, 1), \\ (\mu_{0_3}, M_{0_3}, \alpha_3, \beta_3) &= (\log(0.5, 1, 2, 1). \end{aligned}$$

The considered data are measured with techniques ensuring high precision, and we assume the following prior $\log \sigma_\epsilon \sim \mathrm{N}(-1, 1)$. Because of the small measurement noise, we expect that a bootstrap filter will perform poorly, leading to a very noisy approximation of the likelihood $\pi(y|\phi, \sigma_\epsilon) = \prod_{i=1}^{M} \pi(y^i|\phi^i, \sigma_\epsilon)$. To be able to obtain a good approximation of the likelihood, we instead use the bridge particle filter found in

[21], since, as explained below, the bootstrap filter is statistically inadequate for this experiment (moreover, it is also computationally inadequate, since it would require a too large number of particles, which was impossible to handle with the limited memory of our computer). In Appendix A, we derive the bridge filter for the model in (24), and we also compare the forward propagation of the particles that we obtain using the bootstrap filter and the bridge filter. In Appendix A.2 we see that the likelihood approximation obtained from the bootstrap filter is very inaccurate, which is due to its inability to handle measurements with small observational noise. Consequently, the number of particles required to give likelihood estimates with low variance is computationally prohibitive. Therefore, for this example, we only report results based on the bridge filter (which is not a plug-and-play method).

We use the following four algorithms already defined in Section 5.1: Kalman, which obviously here is the gold-standard method; PMMH, using the bridge filter with $N = 1$ particle; CPMMH-0999 using the bridge filter also with 1 particle, and CPMMH-09 using the bridge filter with 1 particle. We find that, due to propagating particles conditional on the next observation, using a single particle was enough to give likelihood estimates with low variance. We ran all algorithms for 100k iterations, considering the first 20k iterations as burn-in. The starting value for σ_ϵ was set far away from the posterior mean that we obtained from a pilot run of the Kalman algorithm, and the starting values for the random effects ϕ_j^i were set to their prior means. For all algorithms, the proposal distributions were tuned adaptively using the generalized AM algorithm as described in Section 4.5. We ran the algorithms on a single-core computer so no parallelization was utilized. Posterior marginals in Figures 12-13 show that inference results for all algorithms are very similar, except for CPMMH-0999, for which posterior samples of σ_ϵ are inconsistent with the output from the other competing schemes. We note that the case of $N = 1$ can be seen to correspond to a joint update of the parameters and latent process x. Inducing strong positive correlation between successive values of u therefore results in extremely slow mixing over the latent process and in turn, the parameters. This is particularly evident for σ_ϵ, whose update requires calculation of likelihood estimates over all experimental units. Reducing ρ to 0.9 appears to alleviate this problem. Runtimes and ESS values are in Table 4. As expected, Kalman is the most efficient algorithm, being 19 times more efficient than PMMH is terms of ESS/min. However, here PMMH and CPMMH have the same efficiency in terms of ESS/min. Thus, CPMMH does not seem to produce any efficiency improvement for this case study. This is due to the efficiency of the bridge filter in guiding state proposals towards the next observation, and therefore allowing us to run PMMH with very few particles, thus making the potential improvement brought by CPMMH essentially null.

We compare our results with those in [35]. Since we have assumed that the random effects $\phi^i = (\phi_1^i, \phi_2^i, \phi_3^i) = (\log \lambda^i, \log \nu^i, \log \sigma^i)$ are Gaussian, then the $(\lambda^i, \nu^i, \sigma^i)$ are log-Normal distributed with means (λ, ν, σ) and standard deviations $(\sigma_\lambda, \sigma_\nu, \sigma_\sigma)$ respectively. By plugging the posterior means for $(\log \lambda^i, \log \nu^i, \log \sigma^i)$ as returned by "Kalman" into the formulas for the mean and standard deviation of a lognormal distribution, we obtain that $\lambda = 0.036$ $(\sigma_\lambda = 0.009)$ [1/msec], $\nu = 0.406$ $(\sigma_\nu = 0.105)$ [mV/msec], and $\sigma = 0.433$, $(\sigma_\sigma = 0.072)$. In [35] we used a maximum likelihood approach, which is a fast enough procedure for Markovian data (there we did not assume a state-space model) that allowed us to obtain point estimates using all 312 ISIs (instead of 100 ISIs as in this case), but still slow enough to not permit bootstrapped confidence intervals to be obtained. Therefore, there we reported intervals based on asymptotic normality. There we had point estimates $\hat{\nu} = 0.494$ and $\hat{\sigma}_\nu = 0.072$, which are similar to our Bayesian estimation. It makes sense that the inferences are not very different, as in the end our estimation of σ_ϵ is very small, meaning that we could assume nearly Markovian data. However here we have also inferences for random effects λ^i and σ^i, whereas in [35] these were assumed fixed (unknown) effects with maximum likelihood estimates $\hat{\lambda} = 0.047$ [1/msec] (it can be obtained from Table 1 in [35] via $1/0.021 = 47.62$ [1/sec]) and $\hat{\sigma} = 0.427$ [mV/$\sqrt{\text{msec}}$] (it can be obtained from Table 1 in [35] by converting 0.0135 [V/$\sqrt{\text{sec}}$] into [mV/$\sqrt{\text{msec}}$]). We appreciate how close our posterior means based on 100 ISIs are to the maximum likelihood estimates using 312 ISIs.

Algorithm	ρ	N	CPU (m)	mESS	mESS/m	Rel.
Kalman	-	-	56	630	11.30	18.9
PMMH	-	1	479	287	0.6	1.0
CPMMH-09	0.9	1	655	400	0.61	1.0
CPMMH-0999	0.999	1	653	372	0.57	1.0

Table 4: Neuronal model. Correlation ρ, number of particles N, CPU time (in minutes m), minimum ESS (mESS), minimum ESS per minute (mESS/m), and relative minimum ESS per minute (Rel.) as compared to PMMH. All results are based on 100k iterations of each scheme.

6. Discussion

We have constructed an efficient and general inference methodology for the parameters of stochastic differential equation mixed-effects models (SDEMEMs). While SDEMEMs are a flexible class of models for "population estimation", their use has been limited by technical difficulties that make the execution of inference algorithms (both classic and Bayesian) computationally intensive. Our work proposed strategies to both (i) produce Bayesian inference for very general SDEMEMs, without the limitations of previous methods; (ii) alleviate the computational requirements induced by the generality of our methods. The SDEMEMs we considered are general in the sense that the underlying SDEs can be nonlinear in the states and in the parameters; the random parameters can have any distribution (not restricted to the Gaussian family); the observations equation does not have to be a linear combination of the latent states. We produced a Metropolis-within-Gibbs algorithm (hereafter Gibbs sampler, Algorithm 2) with carefully constructed blocking strategies, where the technically difficult approximation to the unavailable likelihood function is efficiently handled via correlated particle filters. The use of correlated particle filters brings in the well-known benefit of requiring fewer particles compared to the particle marginal Metropolis-Hastings (PMMH) algorithm. In our experiments, the novel blocked-Gibbs sampler embedding a correlated PMMH (CPMMH) shows that it is possible to considerably reduce the number of required particles while still obtaining a value of the effective sample size (ESS) that is comparable to using standard PMMH in the Gibbs sampler. This means that the Gibbs sampler with embedded CPMMH is computationally efficient and on two out of three examples of increasing complexity we found that our algorithm is much more efficient than a similar algorithm using the standard PMMH, sometimes even 40 times more efficient. Some care must be taken when choosing ρ, which governs the level of correlation between successive likelihood estimates. Taking $\rho \approx 1$ can result in the sampler failing to adequately mix over the auxiliary variables. We found that this problem was exacerbated when using relatively few particles (such as $N = 1$), but can be overcome by reducing ρ. The fact that our approach is an instance of the pseudo-marginal methodology of [3] implies that we produce exact (simulation-based) Bayesian inference for the parameters of our SDEMEMs, regardless the number of particles used. We mostly focus on producing "plug-and-play" methodology (but see below for exceptions), meaning that no preliminary analytic calculations should be required to run our methods, and forward simulation from the SDEs simulator should be enough. Instead, what is necessary to set is the number of particles N and, when correlated particles filters are used (CPMMH), the correlation parameter ρ (however this one is easily set within the interval $[0.90, 0.999]$). Finally, the usual settings for the MCMC proposal distribution should be decided (covariance matrix of the proposal function $q(\cdot)$). However, for the neuronal data example we had to employ a bridge filter, since the observational noise is very low for this case study, causing the bootstrap filter to perform poorly. The bridge filter is not plug-and-play (as discussed below), however in this paper we have decided to include a non-plug-and-play method to show how to analyze complex case studies with existing state-of-art sequential Monte Carlo filters. When considering a plug-and-play approach, our proposed methodology relies on the use of the bootstrap particle filter, within which particles are propagated according to the SDE solution or an approximation thereof. We note that in scenarios where the observations are particularly informative (e.g. the neuronal data case study in Section 5.3), it may be beneficial to propagate particles conditional on the observations, by using a carefully chosen bridge construct. We refer the reader to [20] for details on the use of such constructs within a CPMMH scheme for SDEs. However, notice that in order to use the constructs in [20] the conditional distribution of observations (i.e. (2) in our context) must be Gaussian. This is the underlying assumption that is exploited in [5] to enable the use of bridge constructs in inference for SDEMEMs. In [5] they also use methods based on correlated particle filters, in a work which has been proposed independently and concurrently to ours (July 25 2019 on arXiv). See for example their "component-wise pseudo-marginal" (CWPM) method, which is similar to the naive Gibbs strategy we also propose, and they found that CWPM was the best strategy among a battery of explored methods. In order to correlate the particles, [5] advocate the use of the blockwise pseudo-marginal strategy of [46]: this way, at each iteration of a CPMMH algorithm they randomly pick a unit in the set $\{1, .., M\}$, and only for that unit they update the corresponding auxiliary variates, whereas for the remaining $M - 1$ units they reuse the same auxiliary variates u^i as employed in the last accepted likelihood approximation. This approach implies an estimated correlation between log-likelihoods of around $1 - 1/M$, which also implies that the correlation level is completely guided by the number of units. This means that for a small M (e.g. $M = 5$ or 10, implying a correlation of 0.80 and 0.90 respectively) a blockwise pseudo-marginal strategy might not be as effective as it could be. On the other hand, assuming a very efficient and scalable implementation allowing measurements from $M = 10,000$ units, the blockwise pseudo-marginal approach would produce highly correlated particles, which can sometimes be detrimental

by not allowing enough variety in the auxiliary variates, and ultimately producing long-term correlations in the parameter chains, as we have documented in Section 5.3 when using a low number of particles N. We therefore think it is advantageous to use a method that allows the statistician to decide on the amount of injected correlation: even though this means having one more parameter to set (ρ in our treatment), we find this decision to be rather straightforward, as mentioned above.

We hope this work can push forward the use of SDEMEMs in applied research, as even though inference methods for SDEMEMs have been available from around 2005, the limitation of theoretical or computational possibilities have implied that only specific SDEMEMs could be efficiently handled, while other SDEMEMs needed ad-hoc solutions or computationally very intensive algorithms. We believe our work is promising as a showcase of the possibility to employ very general SDEMEMs for practical applications.

Acknowledgments

SW was supported by the Swedish Research Council (Vetenskapsrådet 2013-05167). UP was supported by the Swedish Research Council (Vetenskapsrådet 2019-03924). We thank the staff at the Center for Scientific and Technical Computing at Lund University (LUNARC) for help in setting up the computer environment used for the computations in Section 5.1 and 5.3. We thank J. F. He for making the neuronal data available. We thank the editor and three anonymous reviewers for useful and insightful comments on this paper.

References

[1] Ait-Sahalia, Y. (2008). Closed-form likelihood expansions for multivariate diffusions. *The Annals of Statistics*, 36(2):906–937.

[2] Andrieu, C., Doucet, A., and Holenstein, R. (2010). Particle Markov chain Monte Carlo methods (with discussion). *J. R. Statist. Soc. B*, 72(3):1–269.

[3] Andrieu, C. and Roberts, G. O. (2009). The pseudo-marginal approach for efficient computation. *Annals of Statistics*, 37:697–725.

[4] Andrieu, C. and Thoms, J. (2008). A tutorial on adaptive mcmc. *Statistics and computing*, 18(4):343–373.

[5] Botha, I., Kohn, R., and Drovandi, C. (2020). Particle methods for stochastic differential equation mixed effects models. *Bayesian Analysis doi:10.1214/20-BA1216*.

[6] Choppala, P., Gunawan, D., Chen, J., Tran, M.-N., and Kohn, R. (2016). Bayesian inference for state space models using block and correlated pseudo marginal methods. Available from `http://arxiv.org/abs/1311.3606`.

[7] Dahlin, J., Lindsten, F., Kronander, J., and Schon, T. B. (2015). Accelerating pseudo-marginal Metropolis-Hastings by correlating auxiliary variables. Available from `https://arxiv.1511.05483v1`.

[8] Del Moral, P. (2004). *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Springer, New York.

[9] Delattre, M. and Lavielle, M. (2013). Coupling the SAEM algorithm and the extended Kalman filter for maximum likelihood estimation in mixed-effects diffusion models. *Statistics and its interface*, 6(4):519–532.

[10] Deligiannidis, G., Doucet, A., and Pitt, M. K. (2018). The correlated pseudo-marginal method. *J. R. Statist. Soc.B*, 80:839–870.

[11] Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag, New York.

[12] Ditlevsen, S. and Lansky, P. (2005). Estimation of the input parameters in the ornstein-uhlenbeck neuronal model. *Physical review E*, 71(1):011907.

[13] Donnet, S., Foulley, J.-L., and Samson, A. (2010). Bayesian analysis of growth curves using mixed models defined by stochastic differential equations. *Biometrics*, 66(3):733–741.

[14] Donnet, S. and Samson, A. (2013a). A review on estimation of stochastic differential equations for pharmacokinetic/pharmacodynamic models. *Advanced drug delivery reviews*, 65(7):929–939.

[15] Donnet, S. and Samson, A. (2013b). Using PMCMC in EM algorithm for stochastic mixed models: theoretical and practical issues. *Journal de la Société Française de Statistique*, 155(1):49–72.

[16] Doucet, A., Pitt, M. K., and Kohn, R. (2015). Efficient implementation of Markov chain Monte Carlo when using an unbiased likelihood estimator. *Biometrika*, 102:295–313.

[17] Fearnhead, P., Giagos, V., and Sherlock, C. (2014). Inference for reaction networks using the linear noise approximation. *Biometrics*, 70(2):457–466.

[18] Flamary, R. and Courty, N. (2017). POT Python optimal transport library.

[19] Fuchs, C. (2013). *Inference for diffusion processes with applications in Life Sciences*. Springer.

[20] Golightly, A., Bradley, E., Lowe, T., and Gillespie, C. S. (2019). Correlated pseudo-marginal schemes for time-discretised stochastic kinetic models. *CSDA to appear*.

[21] Golightly, A. and Wilkinson, D. J. (2011). Bayesian parameter inference for stochastic biochemical network models using particle markov chain monte carlo. *Interface focus*, 1(6):807–820.

[22] Gordon, N. J., Salmond, D. J., and Smith, A. F. M. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings-F*, 140:107–113.

[23] Höpfner, R. (2007). On a set of data for the membrane potential in a neuron. *Mathematical Biosciences*, 207(2):275–301.

[24] Kloeden, P. E. and Platen, E. (1992). *Numerical solution of stochastic differential equations*. Springer.

[25] Künsch, H. R. (2013). Particle filters. *Bernoulli*, 19:1391–1403.

[26] Lanski, P. (1984). On approximations of stein's neuronal model. *Journal of theoretical biology*, 107(4):631–647.

[27] Lansky, P., Sanda, P., and He, J. (2006). The parameters of the stochastic leaky integrate-and-fire neuronal model. *Journal of Computational Neuroscience*, 21(2):211–223.

[28] Lavielle, M. (2014). *Mixed effects models for the population approach: models, tasks, methods and tools*. Chapman and Hall/CRC.

[29] Leander, J., Almquist, J., Ahlström, C., Gabrielsson, J., and Jirstrand, M. (2015). Mixed effects modeling using stochastic differential equations: illustrated by pharmacokinetic data of nicotinic acid in obese Zucker rats. *The AAPS journal*, 17(3):586–596.

[30] Murphy, K. P. (2007). Conjugate bayesian analysis of the gaussian distribution. `https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf`.

[31] Murray, L., Lee, A., and Jacob, P. E. (2016). Parallel resampling in the particle filter. *Journal of Computational and Graphical Statistics*, 25(3):789–805.

[32] Overgaard, R. V., Jonsson, N., Tornøe, C. W., and Madsen, H. (2005). Non-linear mixed-effects models with stochastic differential equations: implementation of an estimation algorithm. *Journal of pharmacokinetics and pharmacodynamics*, 32(1):85–107.

[33] Picchini, U., De Gaetano, A., and Ditlevsen, S. (2010). Stochastic differential mixed-effects models. *Scandinavian Journal of statistics*, 37(1):67–90.

[34] Picchini, U. and Ditlevsen, S. (2011). Practical estimation of high dimensional stochastic differential mixed-effects models. *Computational Statistics & Data Analysis*, 55(3):1426–1444.

[35] Picchini, U., Ditlevsen, S., De Gaetano, A., and Lansky, P. (2008). Parameters of the diffusion leaky integrate-and-fire neuronal model for a slowly fluctuating signal. *Neural computation*, 20(11):2696–2714.

[36] Picchini, U. and Forman, J. L. (2019). Bayesian inference for stochastic differential equation mixed effects models of a tumor xenography study. *Journal of the Royal Statistical Society: Series C*, 68(4):887–913.

[37] Pitt, M. K., dos Santos Silva, R., Giordani, P., and Kohn, R. (2012). On some properties of Markov chain Monte Carlo simulation methods based on the particle filter. *J. Econometrics*, 171(2):134–151.

[38] Price, L. F., Drovandi, C. C., Lee, A., and Nott, D. J. (2018). Bayesian synthetic likelihood. *Journal of Computational and Graphical Statistics*, 27(1):1–11.

[39] Ruse, M. G., Samson, A., and Ditlevsen, S. (2019). Inference for biomedical data by using diffusion models with covariates and mixed effects. *Journal of the Royal Statistical Society: Series C*.

[40] Sherlock, C., Thiery, A., Roberts, G. O., and Rosenthal, J. S. (2015). On the effciency of pseudo-marginal random walk Metropolis algorithms. *The Annals of Statistics*, 43(1):238–275.

[41] Sørensen, H. (2004). Parametric inference for diffusion processes observed at discrete points in time. *International Statistical Review*, 72(3):337–354.

[42] Steele, J. M. (2012). *Stochastic calculus and financial applications*, volume 45. Springer Science & Business Media.

[43] Stewart, L. and McCarty Jr, P. (1992). Use of bayesian belief networks to fuse continuous and discrete information for target recognition, tracking, and situation assessment. In *Proc. SPIE Signal Processing, Sensor Fusion and Target Recognition*, volume 1699, pages 177–185.

[44] Tornøe, C. W., Overgaard, R. V., Agersø, H., Nielsen, H. A., Madsen, H., and Jonsson, E. N. (2005). Stochastic differential equations in NONMEM®: implementation, application, and comparison with ordinary differential equations. *Pharmaceutical research*, 22(8):1247–1258.

[45] Tran, M.-N., Kohn, R., Quiroz, M., and Villani, M. (2016a). The block pseudo-marginal sampler. *arXiv:1603.02485*.

[46] Tran, M.-N., Kohn, R., Quiroz, M., and Villani, M. (2016b). Block-wise pseudo-marginal metropolis-hastings. *arXiv:1603.02485*.

[47] Whitaker, G. A. (2016). *Bayesian inference for stochastic differential mixed-effects models*. PhD thesis, Newcastle University.

[48] Whitaker, G. A., Golightly, A., Boys, R. J., and Sherlock, C. (2017). Bayesian inference for diffusion driven mixed-effects models. *Bayesian Analysis*, 12:435–463.

[49] Wilkinson, D. J. (2018). *Stochastic Modelling for Systems Biology*. Chapman & Hall/CRC Press, Boca Raton, Florida, 3rd edition.

[50] Wood, S. N. (2010). Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104.

[51] Yu, Y.-Q., Xiong, Y., Chan, Y.-S., and He, J. (2004). Corticofugal gating of auditory information in the thalamus: an in vivo intracellular recording study. *Journal of Neuroscience*, 24(12):3060–3069.

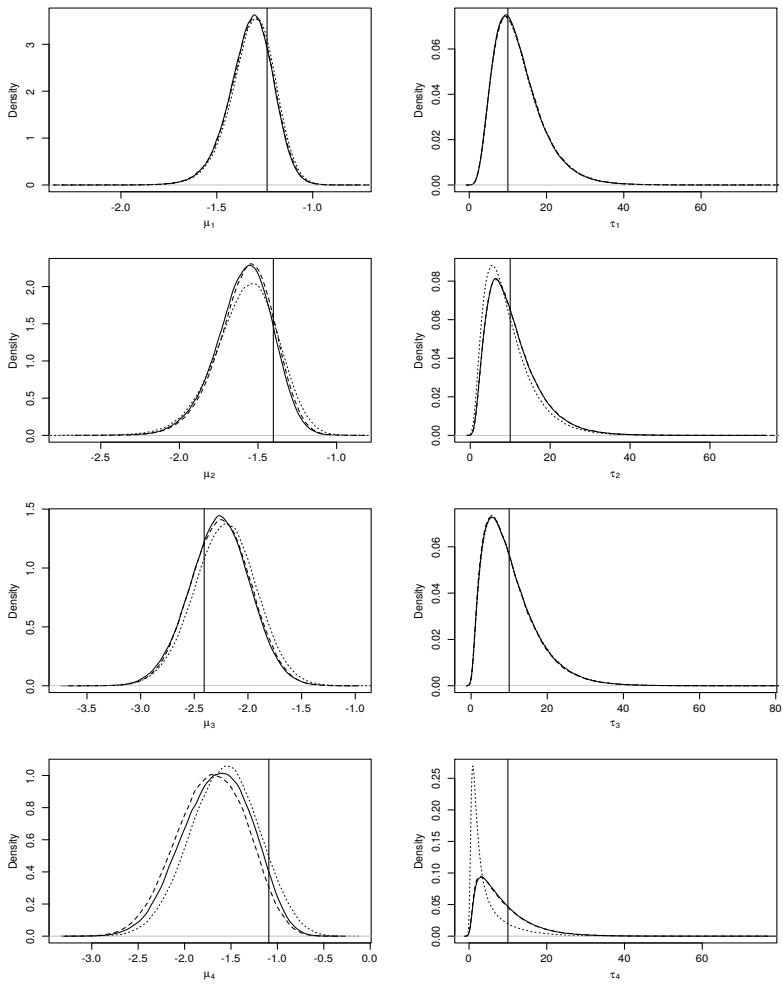


Figure 7: Marginal posterior distributions for μ_i and τ_i, $i = 1, \ldots, 4$. Dotted line shows results from LNA scheme, solid line is from the CPMMH scheme and dashed line is the PMMH Scheme.

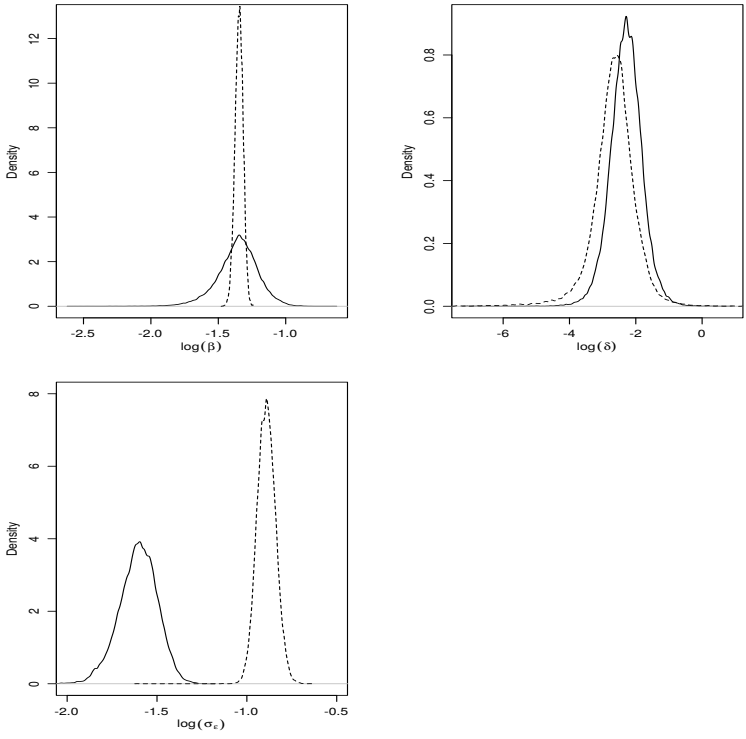


Figure 8: Marginal posterior distributions for the (logged) subject specific parameters $\log \beta^1$, $\log \delta^1$, and the observation standard deviation $\log \sigma_e$. Dashed line shows results from ODEMEM, solid line is from SDEMEM.

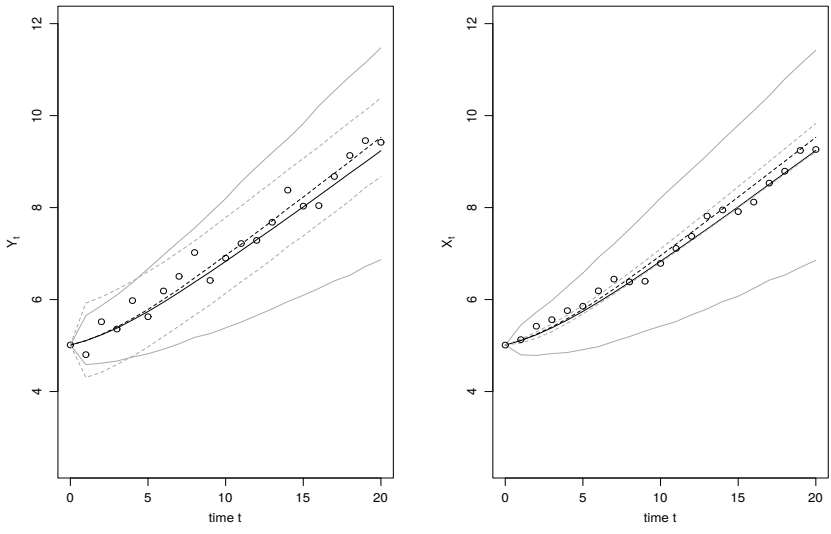


Figure 9: Posterior predictive mean (black) and 95% credible intervals (grey) for the observed process Y_t^1 (circles, left panel) and the latent process $X_t^1 = \log V_t^1$ (circles, right panel). Dashed line shows results from ODEMEM, solid line is from SDEMEM.

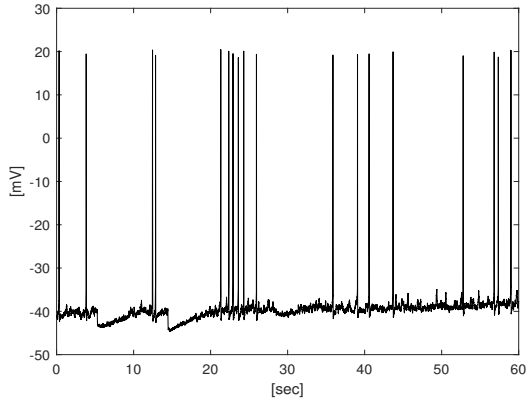


Figure 10: An exemplificative plot of depolarization [mV] vs time [sec] (data from [23]).

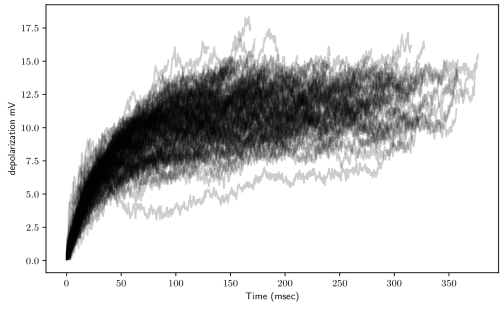


Figure 11: Observations from 100 ISIs.

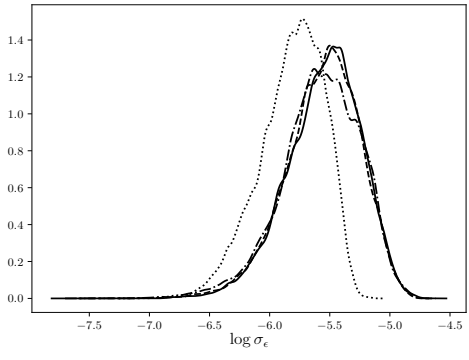


Figure 12: Neuronal model: marginal posterior distributions for $\log \sigma_\epsilon$. Solid line is Kalman, dashed line is PMMH, dotted line is CPMMH-0999, dash-dotted line CPMMH-09.

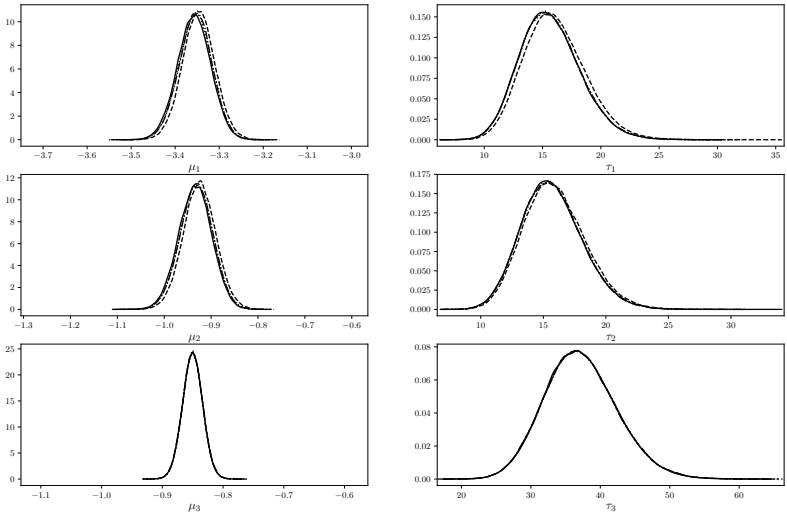


Figure 13: Neuronal model: marginal posterior distributions for $\eta = (\mu_1, \mu_2, \mu_3, \tau_1, \tau_2, \tau_3)$. Solid line is Kalman, dashed line is PMMH, dotted line is CPMMH-0999, dash-dotted line CPMMH-09.

Appendix A. Bridge particle filter

Appendix A.1. Deriving the bridge filter

This section is not strictly pertaining mixed-effects modelling, hence we disregard the subject's index. We consider the bridge particle filter proposed in [21], with the exception that there an SDE was numerically solved using the Euler-Maruyama scheme. Here we provide the bridge particle filter for the special case where the exact (Gaussian) transition density is available, as considered for case studies in Sections 5.1 and 5.3. Since we do not require numerical discretization, in terms of the notation established in [21] we have that $m = 1$ and $j = 0$. Furthermore, we let Δobs denote the step-length for the observational times grid. Thus we have that $\Delta t = \Delta$obs and $\Delta j = 0 = \Delta$obs.

Here the bridge filter is derived for the example in section 5.3. The analytical transition density for the X_t process in (5.3) is

$$X_{t+\Delta t}|X_t = x_t \sim \mathrm{N}\Big(x_t e^{-\lambda\Delta t} + \frac{\nu}{\lambda}(1 - e^{-\lambda\Delta t}), \frac{\sigma^2}{2\lambda}(1 - e^{-2\lambda\Delta t})\Big).$$

The joint density for $X_{t+\Delta t}$ and $Y_{t+\Delta t}$, conditional on X_t, is

$$\begin{pmatrix} X_{t+\Delta t} \\ Y_{t+\Delta t} \end{pmatrix} |X_t = x_t \sim \mathrm{N}\Big\{ \begin{pmatrix} \alpha_0 \\ \alpha_0 \end{pmatrix}, \begin{pmatrix} \beta_0 & \beta_0 \\ \beta_0 & \beta_0 + \sigma_\epsilon^2 \end{pmatrix} \Big\}$$

where $\alpha_0 = x_t e^{-\lambda\Delta t} + \frac{\nu}{\lambda}(1 - e^{-\lambda\Delta t})$, and $\beta_0 = \frac{\sigma^2}{2\lambda}(1 - e^{-2\lambda\Delta t})$. The conditional distribution used as proposal distribution in the bridge filter is

$$\hat{\pi}(x_{t+\Delta t}|x_t, y_{t+\Delta t}) = \mathrm{N}(x_{t+\Delta t}; \mu, \Sigma), \tag{A.1}$$

where $\mu = \alpha_0 + \beta_0(\beta_0 + \sigma_\epsilon^2)^{-1}(y_{t+\Delta t} - \alpha_0)$, $\Sigma = \beta_0(1 - [\beta_0 + \sigma_\epsilon^2]^{-1}\beta_0)$.

Equation (A.1) can be used to propagate particles forward, which is a much more efficient approach than in the bootstrap filter case, where the sampler is miopic to the next observation, while (A.1) is able to look-ahead towards the next observation $y_{t+\Delta t}$. Thus, the bridge filter is similar in structure to Algorithm 1 with the difference that here the particles propagation step consists in sampling from (A.1), and the weights are given by

$$\tilde{w}_{t+\Delta t,k} = \frac{\pi(y_{t+\Delta t}|x_{t+\Delta t,k}, \sigma_\epsilon^2)\pi(x_{t+\Delta t,k}|x_{t,k})}{\hat{\pi}(x_{t+\Delta t,k}|x_{t,k}, y_{t+\Delta t})}, \qquad w_{t+\Delta t,k} = \frac{\tilde{w}_{t+\Delta t,k}}{\sum_{j=1}^N \tilde{w}_{t+\Delta t,j}}, \qquad k = 1, ..., N.$$

Appendix A.2. Comparing the bootstrap filter and the bridge particle filter

To compare the performance of the bootstrap and the bridge filter, we run both filters with the same number of particles (500 particles for each subject) using the 100 ISIs neuronal data from Section 5.3. Parameters are set at the posterior means obtained from the Kalman algorithm. The comparison is interesting since it illustrates the well known issue of running particle filters when the observational error is small (here we have that $\sigma_\epsilon \approx 0.001$), and hence it is expected that the bootstrap filter will produce sub-optimal results. This is due to its inability to "target" the next observation, thus producing very small weights due to the small σ_ϵ. In Figure A.14, we compare the forward propagation of the particles for one ISI chosen at random. It is evident that the bridge filter follows the data more closely. Furthermore, we run each filter independently for 100 times and compare the averages of the log-likelihood values, the standard deviation of the 100 log-likelihood estimations, and the runtimes, see Table A.5. We can easily notice the superiority of the bridge filter returning an averaged log-likelihood value very close to the one provided by the Kalman filter. In particular, notice how the log-likelihood estimation is very unreliable (due to the small observation error).

We now compare the inference results for CPMMH when using the bridge filter and the bootstrap filter. We ran four algorithms: Kalman, PMMH with N = 1 particles using the bridge filter, CPMMH-09 with N = 1 particles using the bridge filter, CPMMH-099 with N = 100 particles using the bootstrap filter. We ran, Kalman, PMMH, and CPMMH-09 for 100k iterations, and ran CPMMH-099 for only 35k iterations, as this case is computationally more intensive. In Figure A.15 we see that when using the bootstrap filter driven

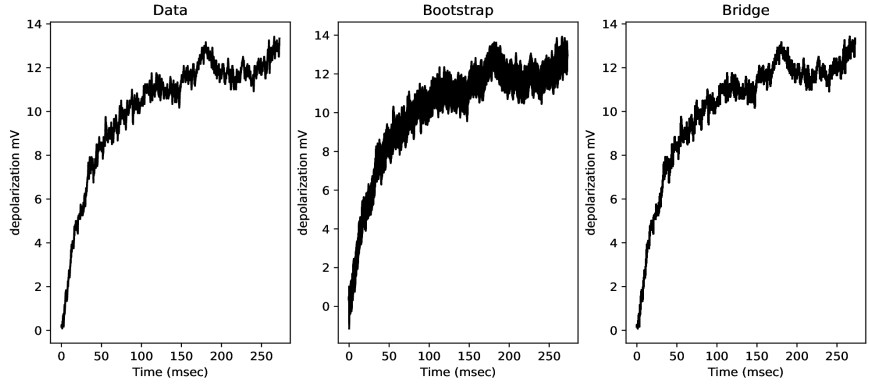


Figure A.14: Neuronal model: forward propagation of the particles for bootstrap and bridge filter for one ISI (chosen at random; this ISI contained 1817 data points). Leftmost panel: observed data for that ISI. Central panel: forward propagation of the particles from the bootstrap filter. Rightmost panel: forward propagation of the particles from the bridge filter.

Table A.5: Comparing 100 log-likelihood estimations for the bootstrap and bridge filter.

	Log-likelihood	Std. Dev.	Runtime (sec)
Kalman	62091	-	0.012
Bootstrap	-2594152	119905	21.51
Bridge	62291	0.34	27.50

inference scheme, the σ_ϵ chain fails to adequately explore regions of high posterior density. We emphasise that this is due to using too few particles ($N = 100$). It is clear from Table A.5 that the number of particles required to match the efficiency of the bridge filter is computationally infeasible. Marginal posteriors for the remaining parameters (not shown) are however similar for all algorithms. The reason why the population parameters η appear to be unaffected by these issues, unlike σ_ϵ, is that step 4 of the Gibbs algorithms in section 4.1 (both versions, naive and blocked one) does not depend on the approximated likelihood, whereas step 2 (which samples σ_ϵ) does depend on it.

Appendix B. Tumor growth – Linear noise approximation

The linear noise approximation (LNA) can be derived in a number of more or less formal ways. We present a brief informal derivation here and refer the reader to [17] and the references therein for further details. We remark that the LNA is not a necessary feature of our general plug-and-play methodology outlined in Section 4 and Algorithm 2.

Appendix B.1. Setup

Consider the tumor growth model in (19), (20) and (21) and a single experimental unit so that the superscript i can be dropped from the notation. To obtain a tractable observed data likelihood, we construct the linear noise approximation of $\log V_t = \log(X_{1,t} + X_{2,t})$.

Let $Z_t = (Z_{1,t}, Z_{2,t}, Z_{3,t})^T = (\log V_t, \log X_{1,t}, \log X_{2,t})^T$. The SDE satisfied by Z_t can be found using the Itô formula, for which we obtain

$$dZ_t = \alpha(Z_t, \phi)dt + \sqrt{\beta(Z_t, \phi)}dW_t$$

where

$$\alpha(Z_t, \phi) = \begin{pmatrix} \left\{\beta + 0.5\gamma^2\right\} e^{Z_{2,t}-Z_{1,t}} + \left\{-\delta + 0.5\tau^2\right\} e^{Z_{3,t}-Z_{1,t}} - 0.5\left\{\gamma^2 e^{2(Z_{2,t}-Z_{1,t})} + \psi^2 e^{2(Z_{3,t}-Z_{1,t})}\right\} \\ \beta \\ -\delta \end{pmatrix}$$

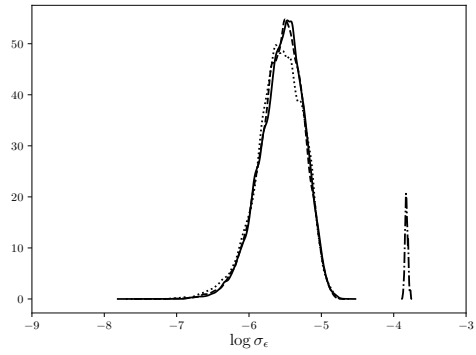


Figure A.15: Neuronal model: marginal posterior distributions for $\log \sigma_\epsilon$. Solid line is Kalman, dashed line is PMMH using the bridge filter, dotted line is CPMMH-09 using the bridge filter, dash-dotted line is CPMMH-099 using the bootstrap filter. The marginal posteriors for Kalman, PMMH, and CPMMH-09 have been multiplied by a factor 40 for pictorial reasons.

$$\beta(Z_t, \phi) = \begin{pmatrix} \gamma^2 e^{2(Z_{2,t}-Z_{1,t})} + \tau^2 e^{2(Z_{3,t}-X_{1,t})} & \gamma^2 e^{2(Z_{2,t}-Z_{1,t})} & \psi^2 e^{2(Z_{3,t}-Z_{1,t})} \\ \gamma^2 e^{2(Z_{2,t}-Z_{1,t})} & \gamma^2 & 0 \\ \psi^2 e^{2(Z_{3,t}-Z_{1,t})} & 0 & \psi^2 \end{pmatrix}.$$

We apply the linear noise approximation (LNA) by partitioning Z_t as $Z_t = m_t + R_t$ where m_t is a deterministic process satisfying

$$\frac{dm_t}{dt} = \alpha(m_t, \phi) \tag{B.1}$$

and $\{R_t, t \geq 0\}$ is a residual stochastic process satisfying

$$dR_t = \{\alpha(Z_t, \phi) - \alpha(m_t, \phi)\}\, dt + \sqrt{\beta(Z_t, \phi)} dW_t.$$

By Taylor expanding α and β about the deterministic process m_t and retaining the first two terms in the expansion of α, and the first term in the expansion of β, we obtain an approximate residual stochastic process $\{\tilde{R}_t, t \geq 0\}$ satisfying

$$d\tilde{R}_t = J_t \tilde{R}_t dt + \sqrt{\beta(m_t, \phi)} dW_t$$

where J_t is the Jacobian matrix with (i,j)th element $(J_t)_{i,j} = \partial \alpha_i(m_t, \phi)/\partial m_{j,t}$. Assuming initial values $m_0 = z_0$ and $\tilde{R}_0 = 0$, the approximating distribution of Z_t is given by

$$Z_t | Z_0 = z_0 \approx \mathrm{N}(m_t, H_t) \tag{B.2}$$

where m_t satisfies (B.1) and, after several calculations which we omit for brevity, H_t is the solution to

$$\frac{dH_t}{dt} = H_t J_t^T + \beta(m_t, \phi) + J_t H_t. \tag{B.3}$$

Appendix B.2. Inference

Note that the observation model in (20) can be written as

$$Y_t = P^T Z_t + \epsilon_t, \qquad \epsilon_t \overset{indep}{\sim} \mathrm{N}(0, \sigma_e^2). \tag{B.4}$$

where P is a 3×1 'observation vector' with first entry 1 and zeroes elsewhere. The linearity of (B.2) and (B.4) yields a tractable approximation to the marginal likelihood $\pi(y|\phi, \sigma_e)$, which we denote by $\pi_{\mathrm{LNA}}(y|\phi, \sigma_e)$. The approximate marginal likelihood $\pi_{\mathrm{LNA}}(y|\phi, \sigma_e)$ can be factorised as

$$\pi_{\mathrm{LNA}}(y|\phi, \sigma_e) = \pi_{\mathrm{LNA}}(y_1|\phi, \sigma_e) \prod_{i=2}^{n} \pi_{\mathrm{LNA}}(y_i | y_{1:i-1}, \phi, \sigma_e) \tag{B.5}$$

where $y_{1:i-1} = (y_1, \ldots, y_{i-1})^T$. Suppose that $Z_1 \sim \mathrm{N}(a, C)$ *a priori*, for some constants a and C. The marginal likelihood under the LNA, $\pi_{\mathrm{LNA}}(y_{1:n}|\phi, \sigma_e) := \pi_{\mathrm{LNA}}(y|\phi, \sigma_e)$ can be obtained via a forward filter, which is given in Algorithm 3.

Algorithm 3 Forward filter

Input: Data y, parameter values ϕ and σ_e.
Output: Observed data likelihood $\pi_{\mathrm{LNA}}(y|\phi, \sigma_e)$.

1. Initialisation. Compute
$$\pi_{\mathrm{LNA}}(y_1|\phi, \sigma_e) = N\left(y_1\,;\, P^T a\,,\, P^T C P + \sigma_e^2\right)$$
where $\mathrm{N}(\cdot\,;\, a\,,\, C)$ denotes the Gaussian density with mean vector a and variance matrix C. The posterior at time $t = 1$ is therefore $Z_1|y_1 \sim \mathrm{N}(a_1, C_1)$ where
$$a_1 = a + CP\left(P^T C P + \sigma_e^2\right)^{-1}\left(y_1 - P^T a\right)$$
$$C_1 = C - CP\left(P^T C P + \sigma_e^2\right)^{-1} P^T C\,.$$

2. For $i = 1, 2, \ldots, n-1$,
   (a) Prior at $i+1$. Initialise the LNA with $m_i = a_i$ and $H_i = C_i$. Integrate the ODEs (B.1) and (B.3) forward to $i+1$ to obtain m_{i+1} and H_{i+1}. Hence
   $$Z_{i+1}|y_{1:i} \sim \mathrm{N}(m_{i+1}, H_{i+1})\,.$$
   (b) One step forecast. Using the observation equation, we have that
   $$Y_{i+1}|y_{1:i} \sim N\left(P^T m_{i+1}, P^T H_{i+1} P + \sigma_e^2\right)\,.$$
   Compute
   $$\begin{aligned}\pi_{\mathrm{LNA}}(y_{1:i+1}|\phi, \sigma_e) &= \pi_{\mathrm{LNA}}(y_{1:i}|\phi, \sigma_e)\pi_{\mathrm{LNA}}(y_{i+1}|y_{1:i}, \phi, \sigma_e)\\ &= \pi_{\mathrm{LNA}}(y_{1:i}|\phi, \sigma_e)\, N\left(y_{i+1}\,;\, P^T m_{i+1}\,,\, P^T H_{i+1} P + \sigma_e^2\right)\,.\end{aligned}$$
   (c) Posterior at $i+1$. Combining the distributions in (a) and (b) gives the joint distribution of Z_{i+1} and Y_{i+1} (conditional on $y_{1:i}$ and ϕ) as
   $$\begin{pmatrix} Z_{i+1} \\ Y_{i+1} \end{pmatrix} \sim N\left\{\begin{pmatrix} m_{i+1} \\ P^T m_{i+1} \end{pmatrix}, \begin{pmatrix} H_{i+1} & H_{i+1}P \\ P^T H_{i+1} & P^T H_{i+1} P + \sigma_e^2 \end{pmatrix}\right\}$$
   and therefore $Z_{i+1}|y_{1:i+1} \sim \mathrm{N}(a_{i+1}, C_{i+1})$ where
   $$a_{i+1} = m_{i+1} + H_{i+1}P\left(P^T H_{i+1} P + \sigma_e^2\right)^{-1}\left(y_{i+1} - P^T m_{i+1}\right)$$
   $$C_{i+1} = H_{i+1} - H_{i+1}P\left(P^T H_{i+1} P + \sigma_e^2\right)^{-1} P^T H_{i+1}\,.$$

Inference for the SDEMEM defined by (19), (20) and (21) may be performed via a Gibbs sampler that draws from the following full conditionals

1. $\pi_{\mathrm{LNA}}(\phi|\eta, \sigma_e, y) \propto \prod_{i=1}^M \pi(\phi^i|\eta)\pi_{\mathrm{LNA}}(y^i|\sigma_e, \phi^i)$,
2. $\pi_{\mathrm{LNA}}(\sigma_e|\eta, \phi, y) \propto \pi(\sigma_e) \prod_{i=1}^M \pi_{\mathrm{LNA}}(y^i|\sigma_e, \phi^i)$,
3. $\pi(\eta|\sigma_e, \phi, y) \propto \pi(\eta) \prod_{i=1}^M \pi(\phi^i|\eta)$.

Paper III

Wiqvist, S., Frellsen, J., & Picchini, U.
Sequential neural posterior and likelihood approximation
arXiv, arXiv:2102.06522.

Sequential Neural Posterior and Likelihood Approximation

Samuel Wiqvist[1], Jes Frellsen[2,⋆], Umberto Picchini[3,⋆]

[1]Centre for Mathematical Sciences, Lund University, Lund, Sweden
[2]Department of Applied Mathematics and Computer Science, Technical University of Denmark, Denmark
[3]Department of Mathematical Sciences, Chalmers University of Technology and the University of Gothenburg, Gothenburg, Sweden

Abstract

We introduce the *sequential neural posterior and likelihood approximation* (SNPLA) algorithm. SNPLA is a normalizing flows-based algorithm for inference in implicit models, and therefore is a simulation-based inference method that only requires simulations from a generative model. SNPLA avoids Markov chain Monte Carlo sampling and correction-steps of the parameter proposal function that are introduced in similar methods, but that can be numerically unstable or restrictive. By utilizing the reverse KL divergence, SNPLA manages to learn both the likelihood and the posterior in a sequential manner. Over four experiments, we show that SNPLA performs competitively when utilizing the same number of model simulations as used in other methods, even though the inference problem for SNPLA is more complex due to the joint learning of posterior and likelihood function. Due to utilizing normalizing flows SNPLA generates posterior draws much faster (4 orders of magnitude) than MCMC-based methods.

1 Introduction

Simulation-based inference (SBI) refers to methods that allow for inference in implicit statistical models, meaning that the likelihood function is only known implicitly via simulations from a generative model. In this work we introduce the *sequential neural posterior and likelihood approximation* (SNPLA), a SBI algorithm for Bayesian inference that bypasses expensive Markov chain Monte Carlo (MCMC) sampling by efficiently generating draws from an approximate posterior distribution. Additionally, and unlike other similar SBI methods, SNPLA learns a computationally cheap approximation of the likelihood function, thus allowing simulations from this "learned" likelihood to be efficiently performed.

Traditionally, approximate Bayesian computation (ABC) [Beaumont et al., 2002, Marin et al., 2012] is the most popular methodology for inference in implicit models. Other important methods include: synthetic likelihoods (SL) [Wood, 2010, Price et al., 2018], Bayesian optimization [Gutmann and Corander, 2016], classification based methods such as likelihood-free inference by ratio estimation (LFIRE) [Thomas et al., 2020], and pseudomarginal methods [Andrieu and Roberts, 2009, Andrieu et al., 2010]. More recent methods are reviewed by Cranmer et al. [2020] and have been benchmarked in Lueckmann et al. [2021]. Also, in recent years normalizing flows [Kobyzev et al., 2020] have turned especially popular in simulation-based inference [Papamakarios et al., 2019b, Greenberg et al., 2019, Radev et al., 2020], due to the ease of probabilistic sampling and density evaluation [Papamakarios et al., 2019a]. Some of the methods that are particularly relevant for our work are: *sequential neural posterior estimation* (SNPE) (SNPE-A [Papamakarios and Murray, 2016], SNPE-B [Lueckmann et al., 2017], and SNPE-C [Greenberg et al., 2019]); *sequential neural likelihood estimation* (SNL) [Papamakarios et al., 2019b]; *sequential neural ratio estimation* (SNRE-A [Hermans et al., 2020], and SNRE-B [Durkan et al., 2020b]); and BayesFlow [Radev et al., 2020].

Our work will focus in particular on SNL and SNPE, since SNPLA is inspired by both. The main disadvantage of SNPE is the *correction step* that must be used so that SNPE learns the correct posterior distribution. SNL avoids this correction step (since SNL learns the likelihood model). However, SNL relies on MCMC for sampling from the posterior, which is time-consuming and restricts which posteriors can

⋆Equal contribution.
Contact: samuel.wiqvist@matstat.lu.se; jefr@dtu.dk; picchini@chalmers.se.
Code: https://github.com/SamuelWiqvist/snpla.

reasonably be learned with success, since MCMC exploration of complex surfaces (e.g. multi-modal targets) can be challenging. Our **main contribution** is the proposed SNPLA method which addresses both these issues. Namely, (i) SNPLA avoids the correction step in SNPE by utilizing the reverse Kullback–Leibler (KL) divergence, and (ii) SNPLA bypasses the typically expensive MCMC runs by using normalizing flows to model both an approximate posterior and an approximate likelihood, resulting in efficient sampling from both. Empirically, we show that SNPLA is on average almost 10^4 times faster than SNL in producing posterior samples (Table 1). **Additionally**, it can be highly valuable to access the normalizing flow-based likelihood model, learned as a by-product of (ii), since this model approximates the data generating process. Thus by sampling from the learned likelihood model, one can rapidly generate artificial data from an approximate generative model. Of course, the latter will be most informative for input parameters that are similar to those that generated the observed data set x_{obs}. An example of the usefulness of learning the likelihood model is illustrated in Section 16. **Finally**, we also show that it is possible to simultaneously learn summary statistics of the data, altogether with the likelihood model and the posterior model, thus providing a flexible plug-and-play inference framework. The **code for replicating the results** can be found at https://github.com/SamuelWiqvist/snpla.

2 Simulation-based inference for implicit models

The implicit statistical model is given by

$$\theta \sim p(\theta),\ x \sim p(x|\theta),$$

where $p(x|\theta)$ is the likelihood function associated to generic data x (hence is sometimes denoted "global likelihood"), and whose functional form we assume unknown. However, we assume the likelihood to be implicitly encoded via an associated computer *simulator* that allows us to generate artificial data, conditionally on an input given by some arbitrary parameter θ and a stream of pseudorandom numbers (and possibly additional covariates or inputs that we do not explicitly represent in our notation). The parameter prior $p(\theta)$ specifies our a-priori beliefs regarding θ. Implicit models are flexible since they only require us to specify a simulator and not the functional form of the likelihood.

As motivated in the Introduction, we are going to focus on SNPE and SNL. SNPE directly learns an approximation $\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$ to the parameter posterior conditionally on observed data x_{obs}, while SNL learns an approximation $\tilde{p}_{\phi_L}(x|\theta)$ of the global likelihood. The models are parameterized with weights ϕ_P and ϕ_L respectively. However, the global likelihood model $\tilde{p}_{\phi_L}(x|\theta)$ is trained with data influenced by the observed data set x_{obs}. This means that $\tilde{p}_{\phi_L}(x|\theta)$ will be most accurate for values of θ having high density under the posterior $p(\theta|x_{\text{obs}})$.

Both SNPE and SNL are sequential schemes that are made data-efficient by employing a proposal distribution $\hat{p}(\theta|x_{\text{obs}})$ that is sequentially adapted to leverage more information from the most recent approximation of the posterior. The SNPE and the SNL schemes are outlined in the supplementary material. For SNPE we have that the proposal distribution is corrected with the factor $p(\theta)/\hat{p}(\theta|x_{\text{obs}})$. This *correction step* is necessary to ensure that the newly constructed parameter proposal is valid, see Papamakarios and Murray [2016] for details. The correction step can either be done analytically [Papamakarios and Murray, 2016], numerically [Lueckmann et al., 2017], or via reparameterization [Greenberg et al., 2019]. The correction step can introduce complexities into SNPE. For example, the closed-form correction of Papamakarios and Murray [2016] can be numerically unstable (if the proposal prior has higher precision than the estimated conditional density) and is restricted to Gaussian and uniform proposals, limiting both the robustness and flexibility of the approach. Regarding the correction in Lueckmann et al. [2017], the introduction of importance weights greatly increases the variance of parameter updates during learning, which can lead to slow or inaccurate inference Greenberg et al. [2019]. For SNL this correction is not necessary since SNL learns the likelihood model $\tilde{p}_{\phi_L}(x|\theta)$. On the other hand, SNL uses MCMC to sample $\theta \sim \hat{p}(\theta|x_{\text{obs}})$, which is time-consuming. For instance, our analysis shows (see right sub-table of Table 1) that SNPLA generates posterior samples on average 12,000 times more rapidly than SNL. Also, MCMC sampling can be unfeasible for some targets with complex geometries.

3 Sequential neural posterior and likelihood approximation

Here we detail our proposed method. The main idea of the SNPLA method is to jointly learn both an approximation $\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$ of the parameter posterior, and an approximation $\tilde{p}_{\phi_L}(x|\theta)$ of the likelihood function. Thus SNPLA has two learnable models:

1. **Parameter posterior model** $\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$, approximating the parameter posterior distribution $p(\theta|x_{\text{obs}})$.
2. **Likelihood model** $\tilde{p}_{\phi_L}(x|\theta)$. Since we are considering an implicit statistical model, we consider the likelihood model $\tilde{p}_{\phi_L}(x|\theta)$ as approximating the data generating process $p(x|\theta)$.

Both the posterior model and the likelihood model are parameterized via normalizing flows, with weights ϕ_P and ϕ_L respectively. The use of normalizing flows is critical since these can be trained using either the forward or the reverse KL divergence [Papamakarios et al., 2017]. The relevant properties and notation for normalizing flows used for SNPLA are presented in the supplementary material.

For SNPLA the likelihood model $\tilde{p}_{\phi_L}(x|\theta)$ is learned via training data sampled from a proposal distribution $\hat{p}(\theta|x_{\text{obs}})$. However, the obtained likelihood approximation $\tilde{p}_{\phi_L}(x|\theta)$ is also used to train the posterior model $\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$, so that we jointly learn both the posterior and the likelihood. The SNPLA scheme is outlined in Algorithm 1.

Algorithm 1: SNPLA

Input: Untrained likelihood model $\tilde{p}_{\phi_L}(x|\theta)$, untrained posterior model $\tilde{p}_{\phi_P}(\theta|x)$, number of iterations R, number of training samples per iteration N, number of training samples per iteration for the posterior model N_P, decay rate $\lambda > 0$.

Output: Trained likelihood model $\tilde{p}_{\phi_L}(x|\theta)$, trained posterior model $\tilde{p}_{\phi_P}(\theta|x)$.

Set $\mathcal{D}_{\mathcal{L}} = \{\emptyset\}$.

for $r = 1 : R$ **do**

`/* Step 1: Update likelihood model with training data sampled from a mixture of the prior and the current posterior model */`

Sample for $n = 1 : N$

$$(\theta_n, x_n) \sim p(x|\theta)\hat{p}_r(\theta|x_{\text{obs}}),$$

where $\hat{p}_r(\theta|x_{\text{obs}}) = \alpha p(\theta) + (1-\alpha)\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$ and, for example, $\alpha = \exp(-\lambda \cdot (r-1))$.

Update training data $\mathcal{D}_{\mathcal{L}} = [\theta_{1:N}, x_{1:N}] \cup \mathcal{D}_{\mathcal{L}}$.

Update $\tilde{p}_{\phi_L}(x|\theta)$ by minimizing the following loss

$$\mathcal{L}(\phi_L) = -E_{\hat{p}(\theta,x|x_{\text{obs}})}\big[\log \tilde{p}_{\phi_L}(x|\theta)\big] \propto E_{\hat{p}(\theta|x_{\text{obs}})}\Big[D_{KL}\big(p(x|\theta))\big|\big|\tilde{p}_{\phi_L}(x|\theta)\big)\Big].$$

if $r = 1$ **then**

`/* Step 2': Hot-start for learning the posterior model */`

Using the prior-predictive samples $[\theta_{1:N}, x_{1:N}]$, update the posterior model by minimizing the following loss

$$\mathcal{L}(\phi_P) \propto -E_{p(\theta,x)=p(x|\theta)p(\theta)}\big[\log \tilde{p}_{\phi_P}(\theta|x)\big].$$

`/* Step 2: Update the posterior model with training data generated from the current posterior model */`

for $j = 1 : N_P/N_{\text{mini}}$ **do**

For $i = 1 : N_{\text{mini}}$: Sample $\theta_i \sim \tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$

Update posterior model, i.e. obtain a new ϕ_P by minimizing the loss (reverse KL divergence):

$$\mathcal{L}(\phi_P) = D_{KL}\big(\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})\big|\big|\tilde{p}_{\phi_L}(x_{\text{obs}}|\theta)p(\theta)\big).$$

Motivation for the construction Assume that $\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$ and $\tilde{p}_{\phi_L}(x|\theta)$ are universal approximators, and that we run SNPLA for one iteration (i.e. $R = 1$) without the hot-start procedure. In this optimal case, after training in **step 1**, we will have that

$$E_{\hat{p}(\theta|x_{\text{obs}})}\Big[D_{KL}\big(p(x|\theta))\big|\big|\tilde{p}_{\phi_L}(x|\theta)\big)\Big] = 0 \Longrightarrow \tilde{p}_{\phi_L}(x|\theta) = p(x|\theta).$$

Thus we have now fully learnt the global likelihood, since training data is generated from the prior-predictive distribution at iteration one. Subsequently, after training in **step 2**, we will have that

$$\begin{aligned} D_{KL}\big(\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})\big|\big|\tilde{p}_{\phi_L}(x_{\text{obs}}|\theta)p(\theta)\big) = 0 \Longrightarrow \tilde{p}_{\phi_P}(\theta|x_{\text{obs}}) &\propto \tilde{p}_{\phi_L}(x_{\text{obs}}|\theta)p(\theta) \\ &\overset{\text{optimal case}}{=} p(x_{\text{obs}}|\theta)p(\theta) \end{aligned}$$

Thus, in the optimal case, we learn the true global likelihood and the true parameter posterior. Of course, in practice, we will utilize models with limited capacity, and we will also need to run SNPLA for several iterations to leverage informative training data for the observed data set x_{obs} that we are considering.

Properties In step 1, the likelihood model $\tilde{p}_{\phi_L}(x|\theta)$ is updated with data generated via the proposal distribution $\hat{p}(\theta|x_{\text{obs}})$, where the latter is set to sequentially leverage more information from the posterior model $\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$. The parameter λ governs how rapidly we want to leverage information from the posterior model. In practice, we have found it useful to use a rather high $\lambda \approx 0.7 - 0.9$, so that the proposal distribution is quickly adapted. For more information regarding the choice of λ see Section 4.5. N governs the number of training data points used in this step. However, since we need to run N simulations in step 1 it is advantageous to keep N conservative, particularly if the model simulator is slow.

In step 2' the posterior model is updated with samples from the prior-predictive distribution. This step is included on pragmatic grounds since we found SNPLA to exhibit convergence problems if this step is not included. Step 2' acts as a pre-training step of the posterior model $\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$ such that the posterior model is set to learn its true target in the first iteration of the algorithm.

Finally, in step 2, the posterior model is updated with samples generated from the current version of the posterior model. These samples are generated via a *simulation-on-fly* scheme where each mini-batch is simulated from the most recent version of the posterior model. This means that in step 2 we loop over the number of mini-bathes N_P/N_{mini} (N_{mini} being the number of samples in one mini-batch, and N_P the total number of samples), and for each mini-batch we simulate new training data. Thus, each mini-batch used in step 2 is unique, since it is simulated from the most recent posterior approximation. Since the training data in step 2 is generated from the flow model, we can take N_P to be much larger than N, and typically one order of magnitude larger.

The posterior model is challenging to learn since it is set to target an approximation of its true target. Furthermore, the learning of the posterior model can be sensitive to potentially catastrophic moves in the weight space, since each mini-batch is generated on-the-fly. However, our experience also shows that the learning of the likelihood model is considerably easier. The learning process of the posterior model has been made more robust by utilizing several strategies: one component of these strategies is to include the hot-start approach in step 2' (something that we already discussed above). We have also found it to be useful to use a large batch size in step 2 ($N_{\text{mini}} \approx 1000$) since a large batch size smooths the training process and thus avoids potentially catastrophic moves. We have also seen that the convergence problems can be addressed by carefully selecting and tuning the ADAM optimizer's learning rate. We have obtained the best results when using a moderate to large learning rate for the posterior model during the first few iterations, which is then sequentially decreased. This allows the posterior model to rapidly explore the weights space in the early iterations and find a reasonable approximation to the posterior. When decreasing the learning rate, we avoid large catastrophic moves of the posterior model's weights, while allowing the posterior model to be fine-tuned when accessing more data.

3.1 Learning the summary statistics

So far, we denoted with $\tilde{p}(\theta|x_{\text{obs}})$ the posterior model that SNPLA learns. However, we can also condition on some function $S(\cdot)$ of the data. In that case, we obtain the following model

$$\tilde{p}_{\phi_P}(\theta|S_{\phi_{P_S}}(x_{\text{obs}})), \tag{1}$$

where ϕ_{P_S} denotes weights that are specific to the $S(\cdot)$ function, since in our work we assume $S(\cdot)$ parameterized with some neural network. We can interpret the function $S(\cdot)$ as mapping the data x_{obs} into a set of summary statistics of the data $S(x_{\text{obs}})$. It is possible to automatically learn $S(\cdot)$ altogether with the likelihood model and the posterior model, as we show in section 4.1, thus providing a general and flexible learning framework. Considering (1) can be particularly useful if x_{obs} is high-dimensional and/or contains some spatial or temporal structure. Thus we want the network $S_{\phi_{P_S}}(\cdot)$ to leverage the features of the data x_{obs} and therefore, for exchangeable data, we could for instance use a DeepSets network Zaheer et al. [2017] (as we do in section 4.1), and for Markovian time-series data it is possible to use a partially exchangeable network Wiqvist et al. [2019]. The network $S_{\phi_{P_S}}(\cdot)$ is trained jointly with the posterior model in (1). So, when incorporating trainable summary statistics, line 6 and 9 of Algorithm 1 is modified so that the posterior model and the summary statistics network are updated according to the following loss

$$\mathcal{L}(\phi_P, \phi_{P_S}) = D_{KL}\big(\tilde{p}_{\phi_P}(\theta|S_{\phi_{P_S}}(x_{\text{obs}}))\big|\big|\tilde{p}_{\phi_L}(x_{\text{obs}}|\theta)p(\theta)\big).$$

Importantly, notice that the summary statistics network $S_{\phi_{P_S}}(\cdot)$ is not included in the likelihood model $\tilde{p}_{\phi_L}(x|\theta)$. The likelihood model still learns a model of the full data set x_{obs}, and not a model for the set of summary statistics computed by $S_{\phi_{P_S}}(\cdot)$. However, for complex data with spatial or temporal structures $\tilde{p}_{\phi_L}(x_{\text{obs}}|\theta)$ can be set up so that the these data features are leveraged in the likelihood model. For instance, specialized flow models for images, audio, and text data have been developed Papamakarios et al. [2019a], and these can be used in the likelihood model to leverage the data features.

4 Experiments

We consider the following case-studies: a multivariate Gaussian (MV-G) model, the two-moons (TM) model [Greenberg et al., 2019], the Lotka-Volterra (LV) model, and the Hodgkin-Huxley (HH) model. The full experimental setting is presented in the supplementary material.

4.1 Proof-of-concept: Multivariate Gaussian

We consider the following conjugate MV Gaussian example from Radev et al. [2020]

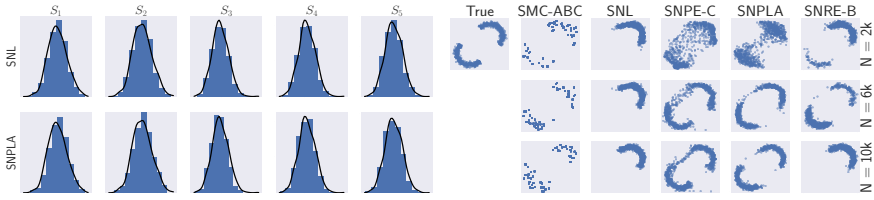
$$\mu \sim N(\mu|\mu_\mu, \Sigma_\mu), \qquad x \sim N(x|\mu, \Sigma),$$

where both Gaussians have dimension two. The covariance Σ is assumed to be known, so the main goal here is to infer the posterior for the mean $p(\mu|x_{\text{obs}})$. The posterior $p(\mu|x_{\text{obs}})$ is analytically known and we evaluate the quality of the inference by computing the KL divergence between the analytical and the approximate posterior, same as in Radev et al. [2020], for details see the supplementary material.

We consider three versions of the MV Gaussian study: (i) "five observations", where data x_{obs} consists of five two-dimensional samples, and the likelihood model $\tilde{p}_{\phi_L}(x|\theta)$ is set to directly target these five observations; (ii) "summary statistics" where data are given by five summary statistics obtained from a two-dimensional vector of 100 observations, and the likelihood model is set to target the summary statistics of these data; and finally (iii) "learnable summary statistics", where we use a small DeepSets network Zaheer et al. [2017] to automatically learn the summary statistics following the method in Section 3.1 based on five observations. We ran all methods independently for 10 times (each time with a different set of observed data), using $R = 10$ iterations with $N = 2{,}500$ model simulations for each iteration. For SNPLE we used $N_P = 40{,}000$ ("five observations" and "learnable summary statistics"), and $N_P = 10{,}000$. ("summary statistics"). For SMC-ABC, however, we utilized up to, in total, $N = 10^6$ model simulations.

Posterior inference for the three versions of the MV Gaussian model is in Figures 2(a), 2(b), and 2(c) (samples from the resulting posteriors are presented as supplementary material). We conclude that all methods, except SMC-ABC, perform similarly well for "five observations". For "summary statistics" SNPLA performs the best, followed by SNPE-C and SNL, while both SMC-ABC and SNRE-B under-perform for the given numbers of model simulations. For "learnable summary statistics" we have that SNPLA and SNRE-B are converging slightly worse compared to SNL, SNPE-C.

For "summary statistics" we also check the performance of the likelihood model in SNL and SNPLA by sampling from the approximate posterior predictive distribution obtained from the learned likelihood models.



(a) MV-G "summary statistics" (b) TM

Figure 1: MV-G and TM: (a) Marginal densities of summaries from the true likelihood model (solid lines) and simulated summary statistics from the learned likelihood model (histograms), produced conditionally on parameters θ from the true posterior; (b) Samples from the approximate posterior distributions and the exact one. Results from one of the ten runs. Results from the first iteration (top), from the third iteration (middle) and the fifth iteration (bottom).

That is, we sampled summary statistics for 1,000 times from the trained likelihood model $\tilde{p}_{\phi_L}(x|\theta)$. These results are presented in Figure 1(a), and we conclude that SNL and SNPLA perform similarly well. The samples from the approximate posterior predictive distributions also match well with the samples from the analytical posterior predictive.

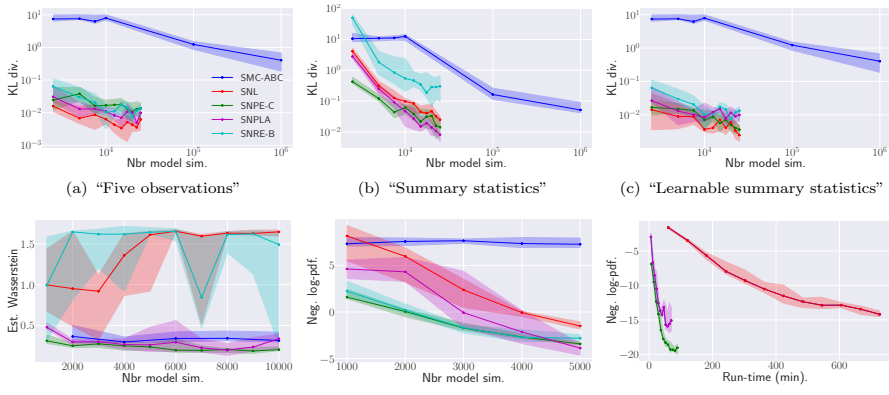
4.2 Complex posterior: Two-moons

The two-moons (TM) example [Greenberg et al., 2019] is a more complex static model. An interesting feature is that the posterior, in some cases, is crescent-shaped. For a technical description of the model, see the supplementary material. We ran all methods 10 times (each time with the same observed data set) for $R = 10$ iterations and using $N = 1{,}000$ model simulations for each iteration, and for SNPLA $N_P = 60{,}000$ samples. However, for SMC-ABC we instead used $R = 5$ with $N = 2{,}000$. For this experiment we cannot use the KL divergence to evaluate the performance of the posterior inference, and we therefore evaluate the posterior accuracy via the Wasserstein distance between the analytical posterior and the approximate posteriors (for details see the supplementary material).

Inference results are in Figures 2(d) and 1(b). We conclude that SNPE-C performs the best. SNPLA also performs well, in particular in many cases the distances from SNPLA are smaller than those from SMC-ABC, however the variability in performance for SNPLA is larger. Indeed while SMC-ABC manages to "visit" both crescent moons, the extent of the exploration for each moon is quite poor (notice in Figure 1(b) it is only apparent that SMC-ABC has fewer samples than claimed; actually many of these overlap on top of each-other). Finally, SNL and SNRE-B perform significantly worse than the other methods. SNL and SNRE-B presumably do not perform well here since both use an MCMC sampler, which can struggle to efficiently explore the bimodal target. These results are in line with those in Greenberg et al. [2019]. We also compare the approximate likelihood models that we learned via SNL and SNPLA. To this end, we simulated data from the learned likelihood models at parameter values simulated from the true posterior. To investigate the performance of the likelihood models we compute the estimated Wasserstein distance between samples from the analytical likelihood and the likelihood models of SNL and SNPLA. The median and quantiles (Q_{25}, Q_{75}) of the distances follow: SNL: 0.111, (0.103, 0.136), SNPLA: 0.146, (0.121, 0.164). Thus, we again conclude that the likelihood models from SNL and SNPLA perform similarly.

4.3 Time-series with summary statistics: Lotka-Volterra

Here, we consider the Lotka-Volterra (LV) case study from Papamakarios et al. [2019b]. Observations are assumed to be a set of 9 summary statistics computed from the 2-dimensional time-series (for details, see the supplementary material). We ran all methods 10 times (each time with the same observed data set) for $R = 5$ iterations each using $N = 1{,}000$ model simulations, and for SNPLA $N_P = 10{,}000$ samples. We followed



(a) "Five observations" (b) "Summary statistics" (c) "Learnable summary statistics"

(d) Two-moons (e) Lotka-Volterra (f) Hodgkin-Huxley

Figure 2: MV Gaussian/Two-moons/Lotka-Volterra/Hodgkin-Huxley: Posterior inference results for "five observations", "summary statistics", "learnable summary statistics", Two-moons, Lotka-Volterra, and Hodgkin-Huxley. SMC-ABC results (blue); SNL (red); SNPE-C (green); SNRE-B (cyan); SNPLA (magenta). The solid lines show the median value over the attempts and the shaded areas show the range for the 25th and 75th percentile.

Papamakarios et al. [2019b] and evaluated the obtained posterior distribution by computing its negative log-pdf at the ground-truth parameters (for details, see the supplementary material).

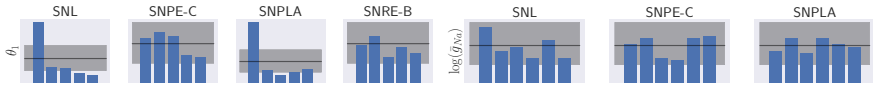
Results are in Figure 2(e) (samples from the resulting posterior distributions are presented in the supplementary material). For a large number of model simulations, SNPE-C, SNRE-B, and SNPLA perform similarly well. However, for $N < 4{,}000$ SNPLA is less precise than SNPE-C and SNRE-B.The true posterior is not known for this experiment and we therefore evaluate the quality of the posterior by utilizing the simulation-based calibration (SBC) procedure Talts et al. [2018]. The SBC results for the first parameter are presented Figure 3(a) (the results for the other parameters are presented in the supplementary material). The SBC results show that SNPLA and SNL produce posteriors that are not as well calibrated as the ones obtained from SNPE-C and SNRE-B. The SBC results for SNL are inline with the results presented in Papamakarios et al. [2019b]. It could be the case that SNPLA produced worse calibrated posterior results due to having a more complex task to perform.

We also ran posterior predictive simulations (results presented in the supplementary material), that is we simulated from the *true* model conditionally on the obtained parameter posterior draws. We conclude that posterior predictive simulations from SMC-ABC and SNL do not correspond well with the observed data. However, for SNPE-C, SNPR-B, and SNPLA we observe a more realistic behaviour. The high variability in the model simulations for all methods is due to the intrinsic stochasticity of the LV model which is characteristic of the model even when supplying realistic parameter values.

4.4 Neural model: Hodgkin-Huxley model

We now consider the Hodgkin-Huxley (HH) model Hodgkin and Huxley [1952], which is used in neuroscience to model the dynamics of a neuron's membrane potential as a function of some stimulus (injected current) and a set of parameters. Following Lueckmann et al. [2017], Papamakarios et al. [2019b], the likelihood $p(x|\theta)$ is defined on 19 summary statistics computed from simulated voltage time-series. Further details about the model can be found in the supplementary material. For our simulation-study, we have 10 unknown parameters, and the observed data was simulated from the model.

We ran SNPE-C, SNL and SNPLA 10 times for $R = 12$ iterations with $N = 2{,}000$ model simulations for



(a) LV (b) HH

Figure 3: LV and HH: simulation-based calibration for (a) LV (for the first parameter i.e. θ_1) and (b) HH (for the first parameter i.e. $\log(\bar{g}_{Na})$). Histogram's counts falling within the grey areas denote good calibration.

each iterations. For SNPLA, we used $N_P = 10{,}000$ samples for each iteration. For each method, the quality of the posterior inference was evaluated by computing the corresponding (approximate) negative log-pdf of the posterior at the ground-truth parameter. We also investigate the quality of the posterior approximation by utilizing SBC analysis, and running posterior predictive simulations. The posterior predictive simulations are obtained by sampling parameter values from the resulting posterior approximation, and then running the HH model simulator at these parameter values.

Posterior inference from using the same number of model simulations is in Figure 2(f). From Figure 2(f) we note that, in terms of the negative log-pdf of the posterior evaluated at the ground truth parameters and for the same number of model simulations, SNL and SNPLA performs similarly well, however, SNPLA converges much faster than SNL in terms of wall-clock time. For instance, obtaining a negative log-pdf value of -15 takes 69 minutes for SNPLA, while SNL obtains the same accuracy in 726 minutes. The SBC results for the first parameter in Figure 3(b) however indicate that all methods produce well-calibrated posteriors (the SBC results for the other parameters are provided in the supplementary material). Samples from the resulting posterior approximations are presented in the supplementary material, and these show that SNPE-C and SNPLA perform better than SNL. Also, the approximate marginals overall resemble the inference results presented in Papamakarios et al. [2019b] and Lueckmann et al. [2017], with the exception of the σ parameter, which could be due to the somewhat different experimental setting we use. Posterior predictive simulations (presented in the supplementary material) show that SNPE-C and SNPLA perform the same and, to a greater extent, resemble the observed data, while SNL is somewhat slightly worse.

To check the quality of the learned likelihood model, we investigate if we can use the latter to rapidly scan for parameter proposals generating data that are similar to the observed data. This is achieved by sampling parameter values from the prior and then computing the number of spikes in the associated simulated data (the number of spikes is one of the summary statistics in the learned likelihood), both when using the true model and when using the trained likelihood model. We then scan the proposals that have produced 4-8 spikes (i.e. parameters that generated data similar to the observed data). For this analysis, the likelihood model agrees with the true model in 75% of the times. Scanning 1000 proposals via the true likelihood model took 4500 sec., while the learned likelihood model utilized 0.2 sec. for the same task, a 2.3×10^4 acceleration in the runtime.

4.5 Hyper-parameter sensitivity analysis

The left sub-table of Table 1 reports the results from an analysis where we run each method for ten times, each time with randomly selected hyper-parameters, and always using the same data. We conclude that SNL and SNPLA are the methods that are the most sensitive to the hyper-parameters.

We also ran SNPLA for 10 different λ values (with $\lambda = 0.6, \ldots, 0.95$), on the same data (results presented in the supplementary material), for all experiments and keeping all other hyper-parameters fixed. This analysis shows that the rate of convergence depends on the value of λ. However, SNPLA produced a reasonable approximation of the posterior for all attempted λ's.

4.6 Run-time analysis

The run-times for generating 1,000 posterior draws are in the right sub-table of Table 1. We conclude that SNPLA on average generates posterior samples 12,000 times faster than SNL. Thus, when the number of model simulations are the same we have that SNPLA will train faster than SNL and SNRE-B In our experiments

Table 1: Left sub-table: Robust coefficient of variation $\frac{IQR}{\text{median}} \cdot 0.75$ (the closer to zero the better) of the performance measures. Right sub-table: Median run-time (in sec.) for generating 1,000 samples from the resulting posterior. MV Gaussian cases: (i) is "five observations", (ii) is "summary statistics", and (iii) is "learnable summary statistics".

	Robust coefficient of variation				Runtime (sec.)			
Experiment	SNL	SNPE-C	SNPLA	SNRE-B	SNL	SNPE-C	SNPLA	SNRE-B
MV-G (i)	0.607	0.550	0.689	0.421	290	0.028	0.027	96
MV-G (ii)	766	0.473	0.851	0.926	1,841	0.024	0.042	118
MV-G (iii)	0.649	0.345	0.547	0.421	287	0.027	0.046	88
LV	-4.739	-0.192	-0.263	-0.249	2,294	0.083	0.085	166
TM	0.008	0.223	1.108	0.007	303	0.028	0.023	77
HH	NA	NA	NA	NA	1,824	0.167	0.197	NA

all methods have access to the same number of model simulations and we have that the training run-time for SNPLA is on average 5.6 times faster compared to SNL, and 2 times faster compared to SNRE-B (the training run-times are reported in the supplementary material).

5 Discussion

In four case studies, we have shown that SNPLA produces similar posterior inference as other simulation-based algorithms when all methods have access to the same number of model simulations. This is a rather interesting finding since the learning task for SNPLA is more complex compared to the other methods that we compare with, given that SNPLA is set to learn both the posterior model and the likelihood model. However, the variability of the inference obtained with SNPLA is somewhat consistently higher, which suggests that SNPLA would need to access more model simulations to obtain inference results *at par* with SNL and SNPE-C. Considering SNPLA's complex learning task that would, however, not be surprising.

The computational acceleration in posterior sampling brought by using normalizing flows modelling is staggering. SNPLA (and SNPE-C) generates posterior samples thousands of times more efficiently than SNL and SNRE-B, since draws from SNPLA and SNPE-C are generated by a forward pass of a normalizing flow network, while those from SNL and SNRE-B are generated via MCMC. In Section 4.6 we have shown that this leads to a substantial speed-up in terms of training run-time for SNPLA compared to SNL and SNRE-B. For the HH model, we have also shown that SNPLA can converge faster than SNL in terms of wall-clock time.

Recently, for simulation-based methods, it has been discussed [Durkan et al., 2018] if it is of advantage to learn the likelihood or the posterior. With this work we show that this question can be circumvented by learning both. However, for posterior inference it seems beneficial to use SNPE-C over SNPLA (at least for the considered examples), but if it is of interest to learn simultaneously the parameters and a cheap model simulator then SNPLA offers this possibility, unlike other considered methods. An advantage with learning both is that we obtain an approximation of the distribution that is typically of interest, i.e. the posterior, and also obtain an approximate model simulator via the learned likelihood model. Since the latter is a normalizing flow, this opens the possibility for the rapid simulation of artificial data, when the "true simulator" $p(x|\theta)$ cannot be used more than a handful of times due to e.g. computational constraints. For the HH model we show that the approximate likelihood model can be used for-instance to rapidly scan parameter proposals from the prior predictive. To achieve this, we do not require any semi-supervised learning (human-intervention based labelling) unlike in Wrede and Hellander [2019]. In other contexts, we could generate many samples from the approximate likelihood to estimate tail probabilities for rare events via (otherwise expensive) Monte Carlo simulations. Of course the uncertainty in the approximate simulator's output will be larger when imputed parameters are very different from those learned from the actual data.

Ethics We have presented an algorithm that learns the posterior and likelihood function of an implicit model. However, implicit models could be used for malicious intents, and our algorithm can also be used for such applications.

Acknowledgements

The computations were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC) at LUNARC partially funded by the Swedish Research Council through grant agreement no. 2018-05973. We would also like to thank the developers of the following `Python` packages for providing the software tools used in this paper: `sbi` [Tejero-Cantero et al., 2020], `nflows` [Durkan et al., 2020a], `PyTorch` [Paszke et al., 2019], and `POT: Python Optimal Transport` [Flamary and Courty, 2017]. UP acknowledges support from the Swedish Research Council (Vetenskapsrådet 2019-03924) and the Chalmers AI Research Centre (CHAIR).

References

C. Andrieu and G. O. Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37:697–725, 2009.

C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

M. A. Beaumont, W. Zhang, and D. J. Balding. Approximate bayesian computation in population genetics. *Genetics*, 162(4):2025–2035, 2002.

M. A. Beaumont, J.-M. Cornuet, J.-M. Marin, and C. P. Robert. Adaptive approximate bayesian computation. *Biometrika*, 96(4):983–990, 2009.

N. T. Carnevale and M. L. Hines. *The NEURON book*. Cambridge University Press, 2006.

K. Cranmer, J. Brehmer, and G. Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 2020. doi: 10.1073/pnas.1912789117.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. 2017. URL `https://arxiv.org/abs/1605.08803`.

C. Durkan, G. Papamakarios, and I. Murray. Sequential neural methods for likelihood-free inference. *arXiv preprint arXiv:1811.08723*, 2018.

C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems*, pages 7511–7522, 2019.

C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios. nflows: normalizing flows in PyTorch, Nov. 2020a. URL `https://doi.org/10.5281/zenodo.4296287`.

C. Durkan, I. Murray, and G. Papamakarios. On contrastive learning for likelihood-free inference. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2771–2781, 2020b.

R. Flamary and N. Courty. Pot python optimal transport library, 2017. URL `https://pythonot.github.io/`.

D. Greenberg, M. Nonnenmacher, and J. Macke. Automatic posterior transformation for likelihood-free inference. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2404–2414, 2019.

M. U. Gutmann and J. Corander. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *The Journal of Machine Learning Research*, 17(1):4256–4302, 2016.

J. Hermans, V. Begy, and G. Louppe. Likelihood-free MCMC with amortized approximate ratio estimators. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4239–4248, 2020.

A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

I. Kobyzev, S. Prince, and M. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

J.-M. Lueckmann, P. J. Goncalves, G. Bassetto, K. Öcal, M. Nonnenmacher, and J. H. Macke. Flexible statistical inference for mechanistic models of neural dynamics. In *Advances in Neural Information Processing Systems*, pages 1289–1299, 2017.

J.-M. Lueckmann, J. Boelts, D. S. Greenberg, P. J. Gonçalves, and J. H. Macke. Benchmarking simulation-based inference. *arXiv preprint arXiv:2101.04653*, 2021.

J.-M. Marin, P. Pudlo, C. P. Robert, and R. J. Ryder. Approximate bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012.

G. Papamakarios and I. Murray. Fast ε-free inference of simulation models with bayesian conditional density estimation. In *Advances in Neural Information Processing Systems*, pages 1028–1036, 2016.

G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019a.

G. Papamakarios, D. Sterratt, and I. Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 837–848. PMLR, 16–18 Apr 2019b. URL `http://proceedings.mlr.press/v89/papamakarios19a.html`.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

L. F. Price, C. C. Drovandi, A. Lee, and D. J. Nott. Bayesian synthetic likelihood. *Journal of Computational and Graphical Statistics*, 27(1):1–11, 2018.

S. T. Radev, U. K. Mertens, A. Voss, L. Ardizzone, and U. Köthe. Bayesflow: Learning complex stochastic models with invertible neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015.

S. Talts, M. Betancourt, D. Simpson, A. Vehtari, and A. Gelman. Validating bayesian inference algorithms with simulation-based calibration. *arXiv preprint arXiv:1804.06788*, 2018.

A. Tejero-Cantero, J. Boelts, M. Deistler, J.-M. Lueckmann, C. Durkan, P. J. Gonçalves, D. S. Greenberg, and J. H. Macke. sbi: A toolkit for simulation-based inference. *Journal of Open Source Software*, 5(52): 2505, 2020. doi: 10.21105/joss.02505. URL `https://doi.org/10.21105/joss.02505`.

O. Thomas, R. Dutta, J. Corander, S. Kaski, and M. U. Gutmann. Likelihood-free inference by ratio estimation. *Bayesian Analysis*, 2020.

S. Wiqvist, P.-A. Mattei, U. Picchini, and J. Frellsen. Partially exchangeable networks and architectures for learning summary statistics in approximate Bayesian computation. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019.

S. N. Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104, 2010.

F. Wrede and A. Hellander. Smart computational exploration of stochastic gene regulatory network models using human-in-the-loop semi-supervised learning. *Bioinformatics*, 35(24):5199–5206, 2019.

M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

Supplementary Material to:

Sequential Neural Posterior and Likelihood Approximation

Contents

1 Introduction to the supplementary material

This document primarily presents additional details on the experimental settings and inference results. However, we also discuss some technicalities on setting up the normalizing flow models.

2 Normalizing flows for simulation-based inference

Here we introduce some basic notions that are necessary in order to follow our method. The normalizing flow model, introduced in Rezende and Mohamed [15] (see Kobyzev et al. [10], Papamakarios et al. [13] for reviews), is a probabilistic model that transforms a simple *base distribution* $u \sim p_u(u)$ into some complex distribution $x \sim p_x(x)$ via the following transformation

$$x = T(u), \qquad u \sim p_u(u).$$

The function T is parameterized with an *invariant* neural network such that T^{-1} exists, and that both T and T^{-1} are differentiable, i.e. the function T is *diffeomorphic*. The probability density function (pdf) for x is computed via the Jacobian J_T of T (or $J_{T^{-1}}$ of T^{-1}) in the following two equivalent ways:

$$\begin{cases} p(x) &= p_u(u)|\det J_T(u)|^{-1}, \qquad u \sim T^{-1}(x), \\ p(x) &= p_u(T^{-1}(x))|\det J_{T^{-1}}(x)|. \end{cases}$$

Due to this structure of the pdf, it is easy to construct complex transformations by composing, say, n transformations such that $T = T_1 \circ T_2 \circ \cdot \circ T_n$ where each transformation T_i is diffeomorphic, and where the Jacobian contribution for each T_i can be computed. Flow models that allow for building these kinds of nested structures are, for instance, RNVP [3], Neural Spline Flow [4], and Masked Autoregressive Flow [12]. Now, assume that we have a normalizing flow model $\tilde{p}_x(x;\phi)$ (with weights ϕ for neural network T), and that our target distribution is denoted $p_x(x)$. We want to train $\tilde{p}_x(x;\phi)$ so that it approximates $p_x(x)$. Let us also assume that we can obtain samples from the target $p_x(x)$, then we can use the *forward* KL divergence to fit the flow model by utilizing the following loss function

$$\begin{aligned} \mathcal{L}(\phi) &= D_{KL}\big[p_x(x)\big|\big|\tilde{p}_x(x;\phi)\big], \\ &= -E_{p_x(x)}\big[\log \tilde{p}_x(x;\phi)\big] + \text{const.}, \\ &= -E_{p_x(x)}\big[\log p_u(T^{-1}(x;\theta)) + \log|\det J_{T^{-1}}(x;\phi)|\big] + \text{const.}. \end{aligned} \tag{1}$$

The loss in (1) is typically (and also in our case) evaluated via Monte Carlo.

If we do not have access to samples from the target distribution $p_x(x)$, but we can evaluate the pdf of $p_x(x)$, it is possible to fit $\tilde{p}_x(x;\phi)$ via the *reverse* KL divergence using the loss

$$\begin{aligned} \mathcal{L}(\phi) &= D_{KL}\big[\tilde{p}_x(x;\phi)\big|\big|p_x(x)\big], \\ &= E_{\tilde{p}_x(x;\phi)}\big[\log \tilde{p}_x(x;\phi) - \log p_x(x)\big], \\ &= E_{p_u(u)}\big[\log p_u(u)) - \log|\det J_T(u;\phi)| - \log p_x(T(u;\phi))\big]. \end{aligned}$$

It is shown in Papamakarios et al. [12] that the forward and the reverse KL divergence are equivalent. The possibility to fit the flow model $\tilde{p}_x(x;\beta)$ via the reverse KL divergence is critical for our method, since SNPLA uses it to train the posterior model without using a proposal distribution.

3 Pseudo-code for SNL and SNPE

SNL and SNPE are presented in Algorithms 1 and 2.

Algorithm 1: SNL

Input: Untrained likelihood model $\tilde{p}_{\phi_L}(x|\theta)$, number of iterations R, number of training samples per iteration N.

Output: Trained likelihood model $\tilde{p}_{\phi_L}(x|\theta)$.

Set $\hat{p}_0(\theta|x_{\text{obs}}) \leftarrow p(\theta)$, $\mathcal{D} = \{\emptyset\}$

for $r = 1 : R$ **do**

  For $n = 1 : N$ sample

$$(\theta_n, x_n) \sim \tilde{p}(\theta, x) = p(x|\theta)\hat{p}_{r-1}(\theta|x_{\text{obs}}).$$

  Update training data $\mathcal{D} = [\theta_{1:N}, x_{1:N}] \cup \mathcal{D}$.

  Update $\tilde{p}_{\phi_L}(x|\theta)$ by minimize the following loss

$$\mathcal{L}(\phi_L) = -E_{\tilde{p}(\theta,x)}\big(\log \tilde{p}_{\phi_L}(x|\theta)\big).$$

  Update the proposal distribution, i.e. let

$$\hat{p}_r(\theta|x_{\text{obs}}) \propto \tilde{p}_{\phi_L}(x_{\text{obs}}|\theta)p(\theta).$$

Algorithm 2: SNPE

Input: Untrained posterior model $\tilde{p}_{\phi_P}(\theta|x)$, number of iterations R, number of training samples per iteration N.

Output: Trained posterior model $\tilde{p}_{\phi_P}(\theta|x)$.

Set $\hat{p}_0(\theta|x_{\text{obs}}) \leftarrow p(\theta)$, $\mathcal{D} = \{\emptyset\}$

for $r = 1 : R$ **do**

  For $n = 1 : N$ sample

$$(\theta_n, x_n) \sim \tilde{p}(\theta, x) = p(x|\theta)\hat{p}_{r-1}(\theta|x_{\text{obs}}).$$

  Update training data $\mathcal{D} = [\theta_{1:N}, x_{1:N}] \cup \mathcal{D}$.

  Update $\tilde{p}_{\phi_P}(\theta|x)$ by minimize the following loss

$$\mathcal{L}(\phi_P) = -E_{\tilde{p}(\theta,x)}\big(\log \tilde{p}_{\phi_P}(\theta|x)\big).$$

  Update the proposal distribution, i.e. let

$$\hat{p}_r(\theta|x_{\text{obs}}) \leftarrow \frac{p(\theta)}{\hat{p}_{r-1}(\theta|x_{\text{obs}})}\tilde{p}_{\phi_P}(\theta|x_{\text{obs}}).$$

4 Computer environment

The code for replicating the experiments can be found at https://github.com/SamuelWiqvist/snpla. All experiments were implemented in `Python 3.7.4`, with normalizing flow models built using the `nflows` package [5]. The `sbi` package [16] was used to run SMC-ABC, SNPE-C, SNRE-B, and SNL. We used the `Neuron` software [2] to produce all model simulations for the HH model. Full specifications of the computer environments used are provided in the `env_local.ylm` and `env_unnamed.txt` files. Regarding licenses for the main already existing assets: `Python 3.7.4` is licensed under the *PSF License Agreement*; `sbi` is licensed under the *GNU Affero General Public License v3.0*; `nflows` is licensed under the *MIT License*; and `Neuron` is licensed under the *GNU GPL*. For information regarding licenses of the new assets see the `LICENSE` file.

5 Experimental setting

For all experiments , except the HH model, we compare the following inference methods: SMC-ABC [1], SNL [14], SNPE-C [8], SNRE-B [6], and SNPLA. For HH we only compare SNL, SNPE-C and SNPLA. The experiments are set-up so that all methods have access to the same number of model simulations. The normalizing flow models used by SNL, SNPE-C, and SNPLA were set to be the same. The masked autoregressive flow architecture [12] was used to construct the flow models.

6 Performance measures

In this section, we give the definitions of the performance measures used.

6.1 Kullback–Leibler (KL) divergence

The KL divergence for two continuous distributions P and Q is defined as

$$D_{KL}(P|Q) = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx,$$

where p and q are the associated probability density functions (pdf). In some cases, the KL divergence is analytically known, and in equation (2) below we give the analytical formula for when P and Q are Gaussian.

6.2 Wasserstein distance

The p^{th} Wasserstein distance between two random variables X and Y on the metric space (M, d) is given by

$$W_p(\mu, \nu) = \left(\inf E[d(X, Y)^p]\right)^{1/p},$$

where d is the distance function on the associated metric space, μ and ν are the marginal distributions of X and Y respectively, and the infimum is taken over all joint distributions of the random variables X and Y with marginals μ and ν. For TM we used the `POT: Python Optimal Transport` package [7] (utilizing the default settings) to estimate the 1^{st} Wasserstein distance.

6.3 Negative log-pdf at ground-truth parameter

Assume that we have the posterior approximation $\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$, and that the ground-truth parameter is $\theta^\star$. The approximate negative log-pdf of the posterior evaluated at the ground-truth parameter is now given by $\log \tilde{p}_{\phi_P}\theta^\star|x_{\text{obs}})$. In our work we compute the negative log-pdf of ground-truth parameter by first approximating $\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$ with a Gaussian distribution, which is estimated from the posterior samples. This step is included to have a consistent approximation of the posterior across models.

7 Incorporating uniform priors in the normalizing flow model for the posterior

The normalizing flow posterior model $\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$ learns the composite transformation $T = T_1 \circ T_2 \circ \cdot \circ T_n$ that maps some base distribution u into the parameter posterior distribution $p(\theta|x_{\text{obs}})$. However, if we have a uniform prior for parameter $\theta \sim U(a, b)$, then we know that the posterior will only have non-zero mass on the interval $[a, b]$. This information can be leveraged in the normalizing flow posterior model $\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$ by adding a scaled and shifted sigmoid function $\sigma_{scale,shift}$ as the last transformation in the chain of transformations that constitutes T. Thus, in that case we have that $T = T_1 \circ T_2 \circ \cdot \circ T_n \circ \sigma_{scale,shift}$. This way, $\tilde{p}_{\phi_P}(\theta|x_{\text{obs}})$ will have positive mass only on the interval $[a, b]$. This is used in all cases where we have uniform priors.

8 Multivariate Gaussian (MV Gaussian)

8.1 Model specification

The conjugate MV Gaussian model of dimension 2 is given by

$$\begin{cases} \mu & \sim N(\mu|\mu_\mu, \Sigma_\mu), \\ x & \sim N\left(\mu, \Sigma = \begin{bmatrix} 1.3862 & 1.4245 \\ 1.4245 & 1.5986 \end{bmatrix}\right). \end{cases}$$

The hyperparameters μ_μ and Σ_μ are set to

$$\mu_\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad \Sigma_\mu \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}.$$

Each simulated dataset was generated by first sampling ground-truth parameters for the mean vector μ_{gt} from the corresponding prior distribution. We did this for each of the several versions of the MV Gaussian model.

8.2 Summary statistics used in the "summary statistics" case-study

We used the following summary statistics for the "summary statistics" case: the two sample means of the data, the two sample variances and the sample covariance between the two components, i.e. the sufficient statistics for the model.

8.3 Computing the KL divergence

The KL divergence between the analytical posterior $p(\mu|x,\Sigma)$ and some approximation $p^\star(\mu|x)$ is given by

$$\begin{aligned} D_{KL}\big(p(\mu|x,\Sigma)\big|\big|p^\star(\mu|x)\big) &= D_{KL}\big(N(\mu|m,\Sigma)\big|\big|p^\star(\mu|m^\star,\Sigma^\star)\big) \\ &= \frac{1}{2}\Big[\log\frac{\det\Sigma^{\star^{-1}}}{\det\Sigma^{-1}} + Tr(\Sigma^{\star^{-1}}\Sigma) - D + (m-m^\star)^T\Sigma^{\star^{-1}}(m-m^\star)\Big], \end{aligned} \tag{2}$$

where m is the mean and Σ the covariance matrix of the analytical posterior. Additionally, $m^\star$ and $\Sigma^\star$ are the sample mean and sample covariance matrix based on 1,000 samples from the posterior approximation, respectively. D is the dimension of x, i.e. $D = 2$ in our case.

8.4 Posterior inference

Samples from the resulting posterior approximations (for one run) are presented in Figure 1. For the analysis, see the main paper.

9 Complex posterior: Two-moons (TM)

9.1 Model specification

A full technical description for the TM model can be found in the supplementary material for Greenberg et al. [8]. We followed the model specification in Greenberg et al. [8], thus we set the observed data to be $x_{\text{obs}} = [0,0]^T$. However, we used a slightly setting of the model and for our experiment we have that: $r \sim N(1.0, 0.1^2)$, and $p = (\cos(a)+1, r\sin(a))$. Due to this setting we use the following priors for θ_i:

$$\theta_i \sim U(-2,2), \qquad i = 1:2.$$

10 Time-series with summary statistics: Lotka-Volterra (LV)

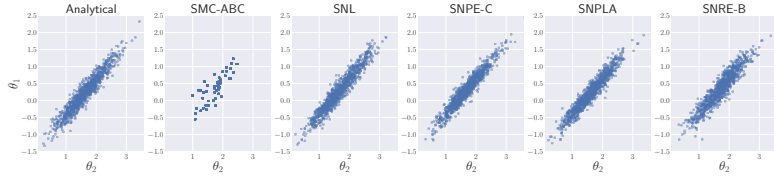
10.1 Model specification

We considered the Markov-jump process version of the LV model presented in the supplementary material of Papamakarios et al. [14], including the model specification. The observed data, generated with ground-truth parameters $\theta_{\text{gt}} = [\log 0.01, \log 0.5, \log 1, \log 0.01]^T$, is in Figure 2. For all parameters their prior is uniform on $[-5,2]$.

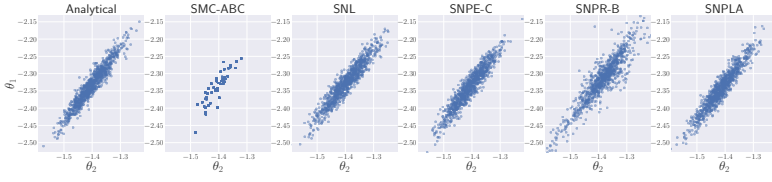
10.2 Summary statistics

We used the same nine summary statistics as in Papamakarios et al. [14], that is:

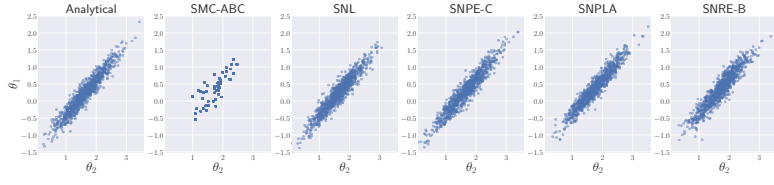
- Mean and log variance of each time-series.
- Auto-correlations of each time-series at time lags 1 and 2.
- Cross-correlation between the two time-series.



(a) "Five observations"



(b) "Summary statistics"



(c) "Learnable summary statistics"

Figure 1: MV:Gaussian: Samples from the resulting posterior approximations for "five observations", "summary statistics", and "learnable summary statistics". Results for one data set.

Before running the inference, a pilot-run procedure was used to make it possible to standardize the summary statistics so that, after standardization, the several components of each summary had a similar relevance. That is, in such a pilot run, we generated $1,000$ samples from the prior-predictive distribution: from these summaries, we then computed their trimmed means and trimmed standard deviations (we trimmed the upper and lower 1.25% of the distribution) to eliminate the effect of very extreme outliers. Afterwards, we ran each inference procedure using standardized summaries (both observed and simulated).

10.3 Dealing with bad simulations

Following [14] we used the Gillespie algorithm to simulate trajectories from the Lotka-Volterra model. The maximum allowed number of steps to advance the Gillespie simulation, for each given trajectory, was set to 10,000. After 10,000 steps we considered as the output of the simulator at θ the partially simulated trajectory and set to zero the remaining part of the path towards the simulation end-time. Thus, we did not remove parameter proposals that rendered poor simulations.

10.4 Posterior inference

Samples from the resulting posterior approximations for one of the runs are presented in Figure 3. For the analysis, see the main paper.

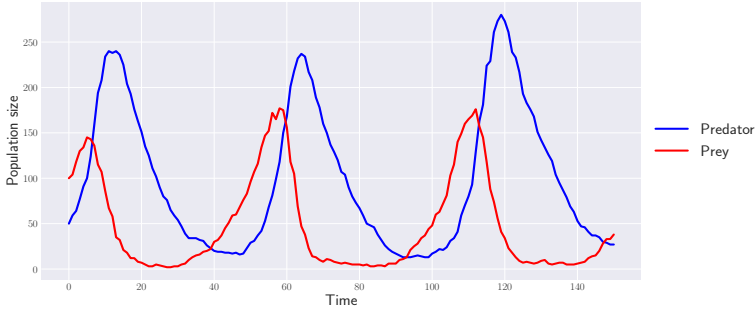


Figure 2: Lotka-Volterra: Simulated data set.

10.5 Posterior predictive simulation

The posterior predictive simulation are presented in Figure 4. For the analysis, see the main paper.

11 Neural model: Hodgkin-Huxley model (HH)

11.1 Model specification

The HH equations [9] model the dynamics of a neuron's membrane potential as a function of some stimulus (injected current) and a set of parameters. In our experiment, we use the same model formulation as in [11, 14]. A full description of the model can be found in the supplementary material for [14]. Our experimental setting is also similar to [11, 14].

We considered 10 unknown parameters $\theta = [\log(\bar{g}_{Na}), \log(\bar{g}_K), \log(g_{leak}), \log(E_{Na}), \log(-E_K), \log(-E_{leak}), \log(\bar{g}_M), \log(\tau_{max}), \log(V_t), \log(\sigma)]$. The ground-truth values are:

$$\begin{cases} \bar{g}_{Na} = 200\ (s/cm^2), & \bar{g}_K = 50\ (s/cm^2), \\ g_{leak} = 0.1\ (s/cm^2), & E_{Na} = 50\ (mV), \\ E_K = -100\ (mV), & E_{leak} = -70\ (mV), \\ \bar{g}_M = 0.07\ (s/cm^2), & \tau_{max} = 1000\ (mV), \\ V_t = 60\ (mV), & \sigma = 1\ (uA/cm^2). \end{cases}$$

We followed [14] and let the uniform prior for each unknown parameter θ_i be

$$\theta_i \sim U(\theta_i^\star - \log(2), \theta_i^\star + \log(1.5)), \tag{3}$$

where $\theta_i^\star$ is the ground-truth vale for θ_i. We considered C, $\kappa_{\beta n1}$, $\kappa_{\beta n2}$ as known, and these were fixed at

$$C = 1\ (uF/cm^2), \qquad \kappa_{\beta n1} = 0.5\ (ms^{-1}), \qquad \kappa_{\beta n2} = 40\ (mV).$$

We generated "observed data" from the HH model for 200 ms with a time-step of 0.025 ms. We used the `Neuron` software [2] to produce all model simulations.

11.2 Summary statistics

The likelihood $p(x|\theta)$ was defined on a set of 19 summary statistics that we computed from the voltage time-series produced by the `Neuron` simulator. We used the same 18 summary statistics as in [14], and as an

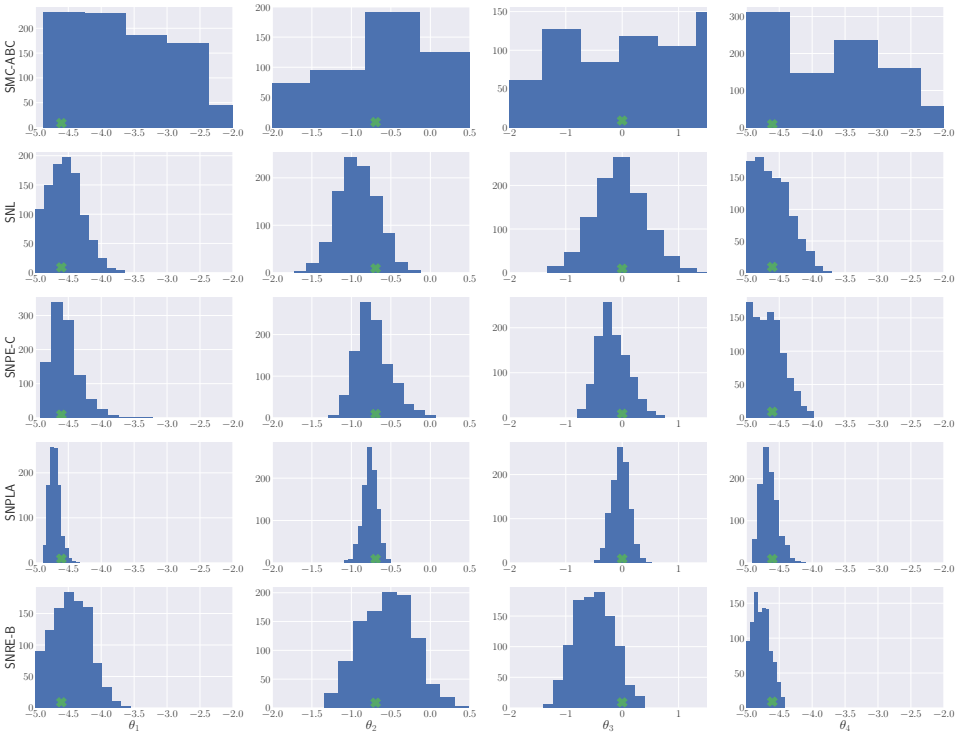


Figure 3: Lotka-Volterra: Samples from the resulting posterior approximations. The green marker shows the true parameter value. Results for one data set.

additional summary statistic, we included the number of spikes in the voltage time-series. The number of spikes in the data set is computed with the same methods as in [11]. Before running the inference, a pilot-run procedure was used to standardize the summary statistics using a whitening transform. Thus we used a similar standardization scheme for the summary statistics as in [14].

11.3 Posterior inference

The posterior samples from SNPE-C, SNL, and SNPLA are in Figures 5, 6, and 7 respectively. For the analysis, see the main paper.

11.4 Posterior predictive simulations

The posterior predictive simulations are presented in Figure 8. For the analysis, see the main paper.

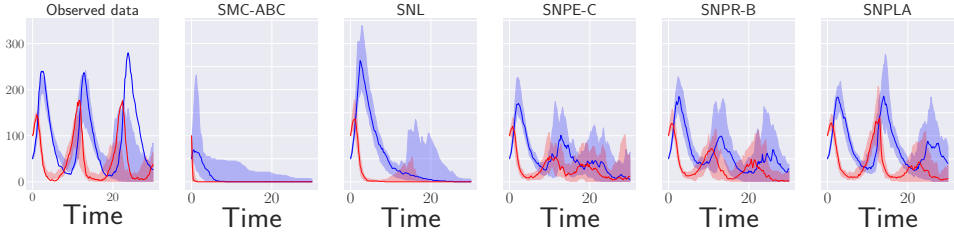


Figure 4: LV: Posterior predictive simulations. The solid lines show the median value over the attempts and the shaded areas show the range for the 25th and 75th percentile.

11.5 Prediction of the number of spikes

Figure 9 presents a comparison between the true number of spikes vs. the predicted number of spikes, for the case where parameters θ are generated from the resulting posterior distribution. We see that the predicted number of spikes in Figure 9 quite well corresponds to the true number of spikes. Figure 10 shows the same analysis but for the case where where parameters θ are generated from the prior. However, the trained likelihood model occasionally predicted a very large number of spikes. Thus, in Figure 10 we have removed 267 (out of 1000) cases where the trained likelihood model predicted the number of spikes to be > 1000. Unsurprisingly, compared to Figure 9 here we have that the predicted number of spikes matches the true number of spikes with higher uncertainty.

12 Training run-times

The training run-times are presented in Table 1.

Table 1: Median training run-time (in sec.). MV Gaussian cases: (i) is "five observations", (ii) is "summary statistics", and (iii) is "learnable summary statistics".

Experiment	SNL	SNPE-C	SNPLA	SNRE-B
MV-G (i)	7374	1179	1311	2676
MV-G (ii)	13202	1696	1441	3340
MV-G (iii)	7153	1803	1371	2586
LV	6741	3628	3539	6167
TM	3131	663	556	895
HH	42554	5240	4196	NA

13 Train/validation/test splits

The train/validation/test splits used for the different experiments are presented in Table 2. Regarding validation data: for SNL, SNPE-C, and SNRE-B, the validation data is set to a fraction valfrac of the training data. Thus, if we use N samples to train the model, $N \times val - frac$ of these samples will be used for validation. This approach of splitting the training and validation data is also used when training the likelihood model for SNPLA. However, a somewhat different approach is used when training SNPLA's posterior model. Due to the *simulation-on-the-fly* approach, for SNPLA's posterior model we instead use $N_p \times$ valfrac *additional* simulations for validation purposes.

Table 2: Train/validation/test splits for all experiments. R is the number of rounds, N is the number of model simulations per round, N_p the number of total samples from the posterior model used to train SNPLA's posterior model, valfrac is the fraction of the training data used for validation, $N_{test,post}$ is the number of posterior samples from the posterior approximation at each round, $N_{test,like}$ is the number of sample from the resulting likelihood model (only applicable to SNL and SNPLA). MV Gaussian cases: (i) is "five observations", (ii) is "summary statistics", and (iii) is "learnable summary statistics".

Experiment	R	N	N_p	valfrac	$N_{test,post}$	$N_{test,like}$
MV-G (i)	10	2,500	40,000	0.1	1,000	NA
MV-G (ii)	10	2,500	10,000	0.1	1,000	1,000
MV-G (iii)	10	2,500	40,000	0.1	1,000	NA
TM	10	1,000	60,000	0.1	1,000	1,000
LV	5	1,000	10,000	0.1	1,000	NA
HH	12	2,000	10,000	0.1	1,000	NA

14 Hyper-parameter settings

The hyper-parameter settings, for all experiments, are in Tables 3-6. Regarding the learn-rate settings: 0.0005 is the default learn-rate used in `sbi`, and 0.001 is the default in the Adam optimizer found in `PyTorch`. We decreased the learn rate for SNPLA's posterior model using the `PyTorch` function `torch.optim.lr_scheduler.ExponentialLR` with multiplicative factor γ_p, see Table 6 (for details on how the multiplicative factor γ_p is used, see the documentation for `torch.optim.lr_scheduler.ExponentialLR`).

Table 3: SNL: Hyper-parameter-setting for the different experiments for SNL. lr is the learn rate. MV Gaussian cases: (i) is "five observations", (ii) is "summary statistics", and (iii) is "learnable summary statistics".

Experiment	lr
MV-G (i)	0.0005
MV-G (ii)	0.0005
MV-G (ii)	0.0005
TM	0.0005
LV	0.0005[a]
Hodgkin-Huxley	0.0005

[a]From a numerical-stability standpoint we found it for this case to be beneficial to exponentially decrease the learn rate with a decay rate of 0.98

Table 4: SNPE-C: Hyper-parameter-setting for the different experiments for SNPE-C. lr is the learn rate. MV Gaussian cases: (i) is "five observations", (ii) is "summary statistics", and (iii) is "learnable summary statistics".

Experiment	lr
MV-G (i)	0.0005
MV-G (ii)	0.0005
MV-G (iii)	0.0005
TM	0.0005
LV	0.0005
HH	0.0005

Table 5: SNRE-B: Hyper-parameter-setting for the different experiments for SNPE-B. lr is the learn rate. MV Gaussian cases: (i) is "five observations", (ii) is "summary statistics", and (iii) is "learnable summary statistics".

Experiment	lr
MV-G (i)	0.0005
MV-G (ii)	0.0005
MV-G (iii)	0.0005
TM	0.0005
LV	0.0005
HH	NA

Table 6: SNPLA: Hyper-parameter-setting for the different experiments for SNPLA. lr_L is the learn rate for the likelihood model, lr_P is the learn rate for the posterior model, γ_P is the multiplicative factor of the decrease for the learn rate of the posterior model, and λ is the exponential decrease rate for the prior. MV Gaussian cases: (i) is "five observations", (ii) is "summary statistics", and (iii) is "learnable summary statistics".

Experiment	lr_L	lr_P	γ_P	λ
MV-G (i)	0.001	0.002	0.95	0.7
MV-G (ii)	0.001	0.002	0.95	0.7
MV-G (iii)	0.001	0.002	0.95	0.7
TM	0.001	0.001	0.9	0.7
LV	0.001	0.001	0.9	0.9
HH	0.001	0.001	0.95	0.8

15 Sensitivity analysis: hyper-parameter ranges

The Hyper-parameter-ranges used for the sensitivity analysis are presented in Table 7.

Table 7: Hyper-parameter-ranges: SNL shows the Hyper-parameter-ranges used for MV-G ((i), (ii), (iii)) and TM, SNL-LV shows the Hyper-parameter-ranges used for LV. SNPE-C, SNPLA, and SNRE-B shows the Hyper-parameter-ranges used for MV-G (i,ii,iii), TM, and LV

Method	lr	γ_{lr}	lr_L	lr_P	γ_P	λ
SNL	$[10^{-4}, 10^{-2}]$	NA	NA	NA	NA	NA
SNL-LV	$[10^{-4}, 10^{-2}]$	$[0.9, 0.999]$	NA	NA	NA	NA
SNPE-C	$[10^{-4}, 10^{-2}]$	NA	NA	NA	NA	NA
SNPLA	NA	NA	$[10^{-4}, 10^{-2}]$	$[10^{-4}, 10^{-2}]$	$[0.8, 0.999]$	$[0.65, 0.95]$
SNRE-B	$[10^{-4}, 10^{-2}]$	NA	NA	NA	NA	NA

16 SNPLA for different values of λ

Figure 11 presents the performance measures that we obtain when running SNPLA with different λ values. We conclude that the difference in performance when changing λ is moderate, and that for all cases we obtain a resulting performance measure indicating adequate posterior approximation.

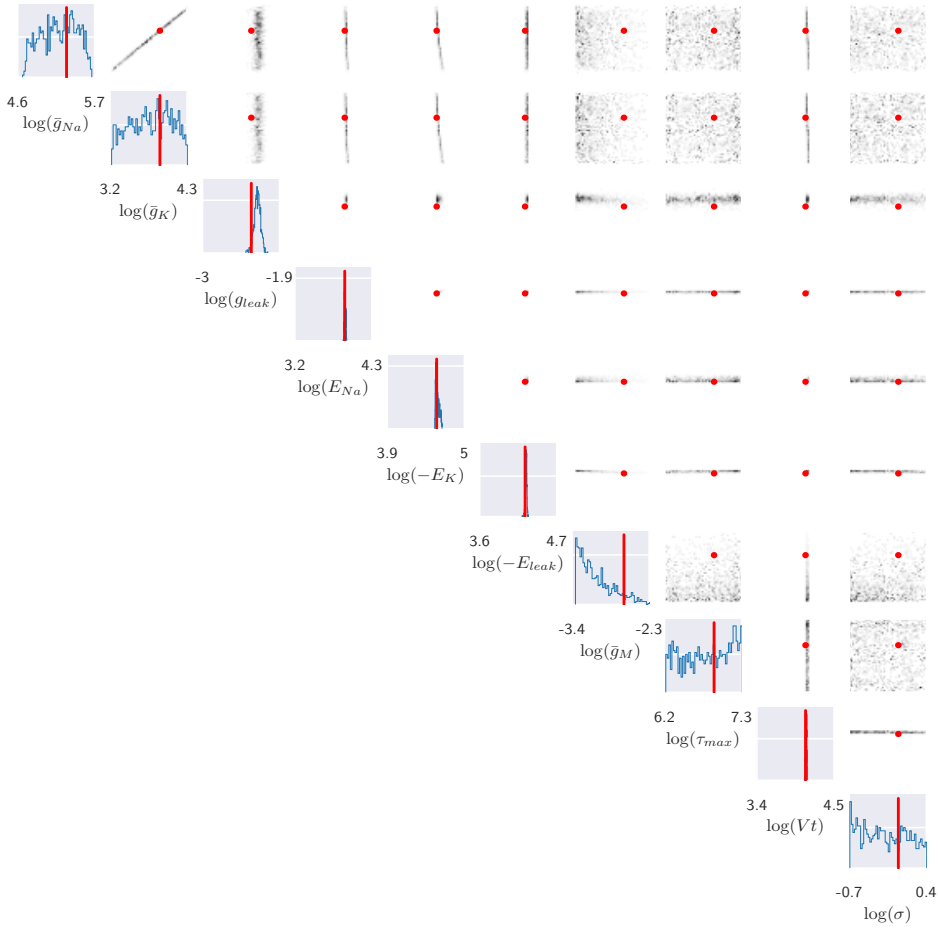


Figure 5: Hodgkin-Huxley: Samples from the posterior approximation for SNPE-C. Results for one data set.

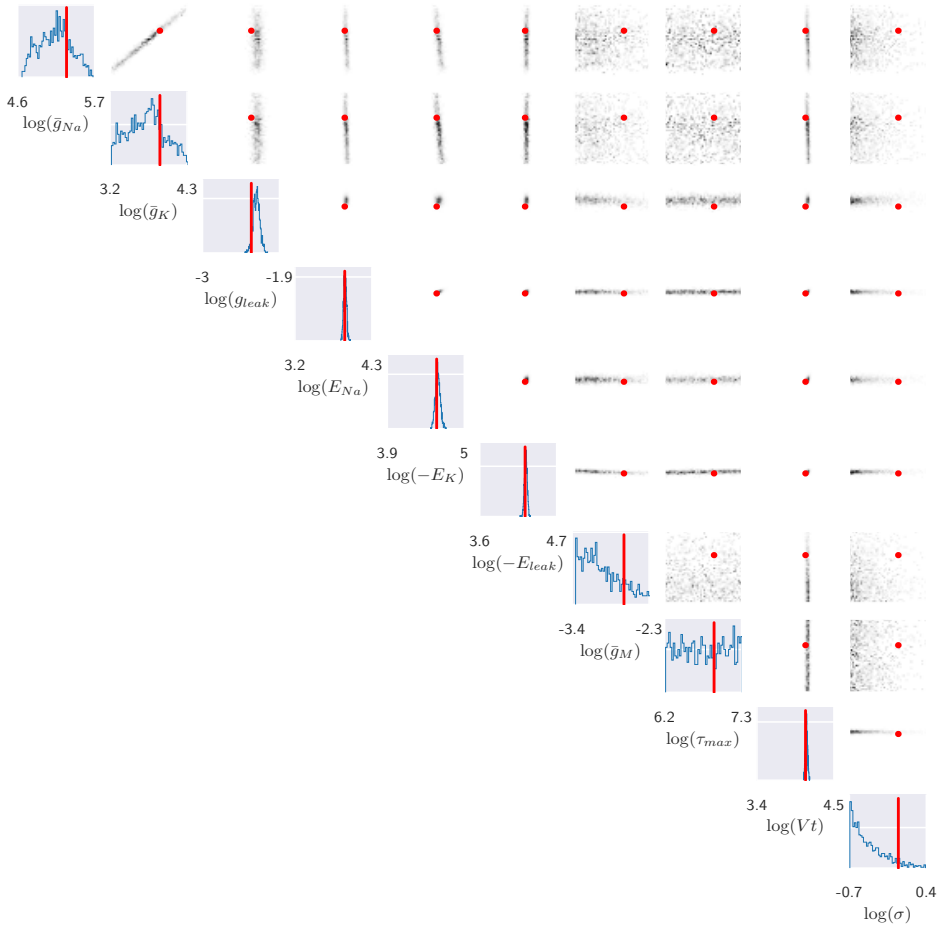


Figure 6: Hodgkin-Huxley: Samples from the posterior approximation for SNL. Results for one data set.

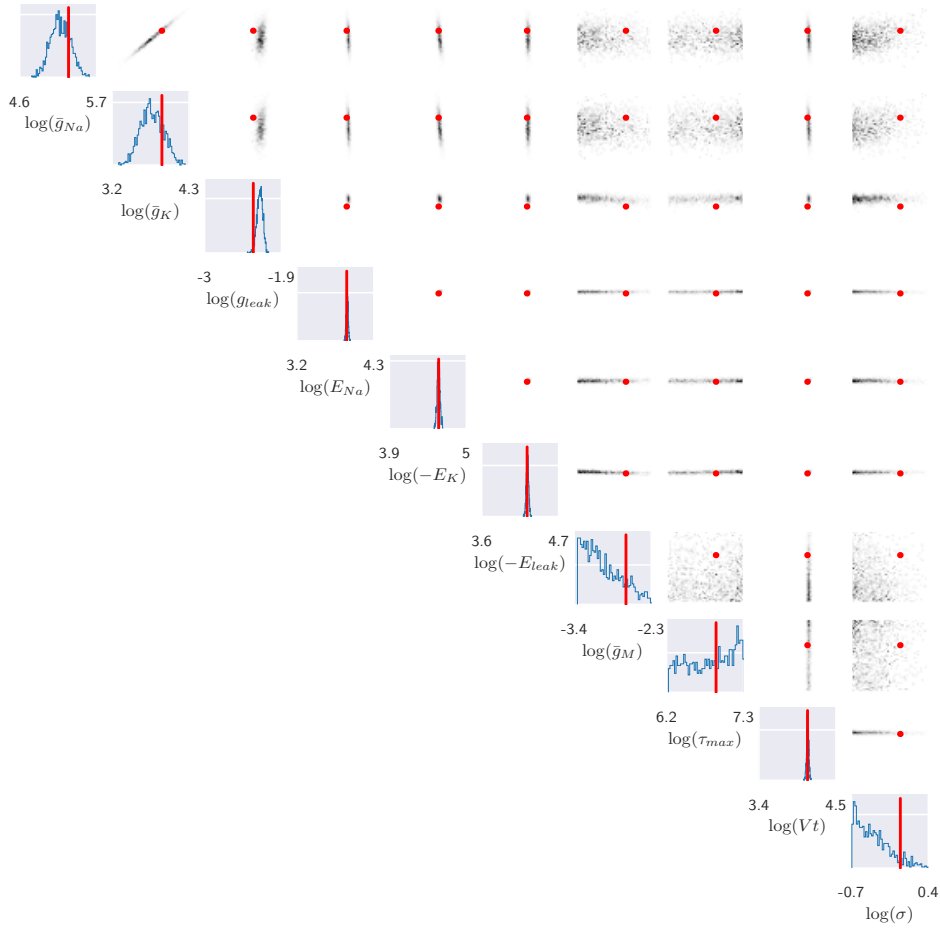


Figure 7: Hodgkin-Huxley: Samples from the posterior approximation for SNPLA. Results for one data set.

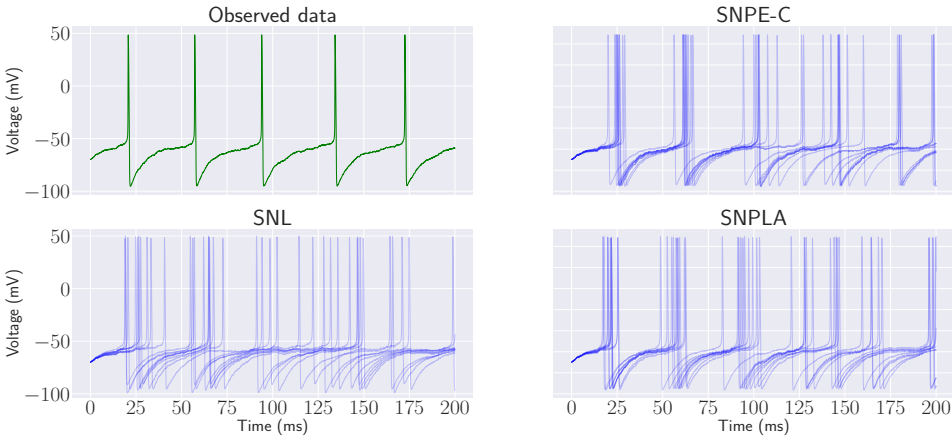


Figure 8: HH: Posterior predictive paths. Results for one data set.

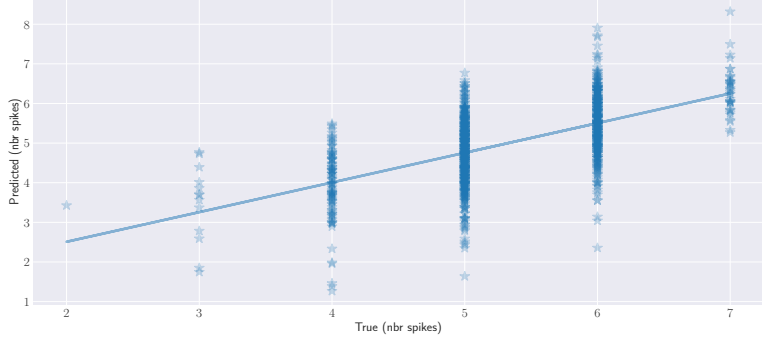


Figure 9: HH: Number of spikes prediction when using parameters from the resulting posterior. Results are based on a single data set. The solid line represents the line of best fit.

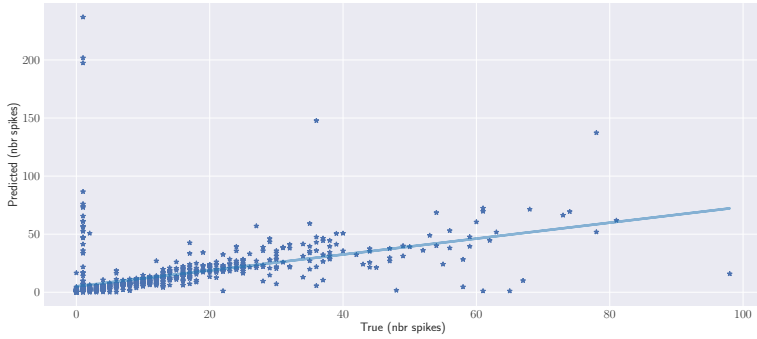


Figure 10: HH: Number of spikes prediction when using parameters generated from the prior. Results are based on a single data set. The solid line represents the line of best fit.

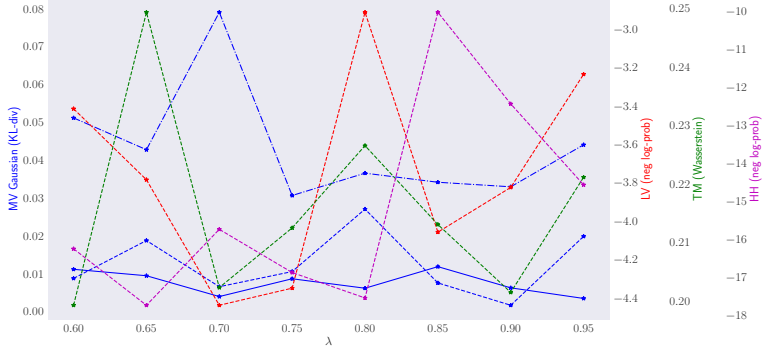


Figure 11: Results for different λ values.

References

[1] M. A. Beaumont, J.-M. Cornuet, J.-M. Marin, and C. P. Robert. Adaptive approximate bayesian computation. *Biometrika*, 96(4):983–990, 2009.

[2] N. T. Carnevale and M. L. Hines. *The NEURON book*. Cambridge University Press, 2006.

[3] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. 2017. URL `https://arxiv.org/abs/1605.08803`.

[4] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems*, pages 7511–7522, 2019.

[5] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios. nflows: normalizing flows in PyTorch, Nov. 2020. URL `https://doi.org/10.5281/zenodo.4296287`.

[6] C. Durkan, I. Murray, and G. Papamakarios. On contrastive learning for likelihood-free inference. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2771–2781, 2020.

[7] R. Flamary and N. Courty. Pot python optimal transport library, 2017. URL `https://pythonot.github.io/`.

[8] D. Greenberg, M. Nonnenmacher, and J. Macke. Automatic posterior transformation for likelihood-free inference. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2404–2414, 2019.

[9] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[10] I. Kobyzev, S. Prince, and M. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[11] J.-M. Lueckmann, P. J. Goncalves, G. Bassetto, K. Öcal, M. Nonnenmacher, and J. H. Macke. Flexible statistical inference for mechanistic models of neural dynamics. In *Advances in Neural Information Processing Systems*, pages 1289–1299, 2017.

[12] G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

[13] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

[14] G. Papamakarios, D. Sterratt, and I. Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 837–848. PMLR, 16–18 Apr 2019. URL `http://proceedings.mlr.press/v89/papamakarios19a.html`.

[15] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015.

[16] A. Tejero-Cantero, J. Boelts, M. Deistler, J.-M. Lueckmann, C. Durkan, P. J. Gonçalves, D. S. Greenberg, and J. H. Macke. sbi: A toolkit for simulation-based inference. *Journal of Open Source Software*, 5(52): 2505, 2020. doi: 10.21105/joss.02505. URL `https://doi.org/10.21105/joss.02505`.

Paper IV

Wiqvist, S., Picchini, U., Forman, J. L., Lindorff-Larsen, K., & Boomsma, W. Accelerating delayed-acceptance Markov chain Monte Carlo algorithms *arXiv*, arXiv:1806.05982

Accelerating delayed-acceptance Markov chain Monte Carlo algorithms

Samuel Wiqvist*, Umberto Picchini◇*, Julie Lyng Forman†, Kresten Lindorff-Larsen‡, Wouter Boomsma⋆

*Centre for Mathematical Sciences, Lund University, Sweden
◇Department of Mathematical Sciences, Chalmers University of Technology and the University of Gothenburg, Sweden
†Dept. Public Health, section of Biostatistics, University of Copenhagen, Denmark
‡The Linderstrøm-Lang Centre for Protein Science, Department of Biology, University of Copenhagen, Denmark
⋆Department of Computer Science, University of Copenhagen, Denmark

Abstract

Delayed-acceptance Markov chain Monte Carlo (DA-MCMC) samples from a probability distribution via a two-stages version of the Metropolis-Hastings algorithm, by combining the target distribution with a "surrogate" (i.e. an approximate and computationally cheaper version) of said distribution. DA-MCMC accelerates MCMC sampling in complex applications, while still targeting the exact distribution. We design a computationally faster, albeit approximate, DA-MCMC algorithm. We consider parameter inference in a Bayesian setting where a surrogate likelihood function is introduced in the delayed-acceptance scheme. When the evaluation of the likelihood function is computationally intensive, our scheme produces a 2-4 times speed-up, compared to standard DA-MCMC. However, the acceleration is highly problem dependent. Inference results for the standard delayed-acceptance algorithm and our approximated version are similar, indicating that our algorithm can return reliable Bayesian inference. As a computationally intensive case study, we introduce a novel stochastic differential equation model for protein folding data.

Keywords: Bayesian inference, Gaussian process, pseudo marginal MCMC, protein folding, stochastic differential equation

1 Introduction

We introduce a new strategy to accelerate Markov chain Monte Carlo (MCMC) sampling when the evaluation of the target distribution is computationally expensive. We build on the "delayed-acceptance" (DA) strategy developed in Christen and Fox [2005] where a fast, "two-stages" DA-MCMC algorithm is proposed while still targeting the desired distribution exactly. We produce an approximated and accelerated delayed-acceptance MCMC algorithm (ADA-MCMC), where in exchange of exactness we obtain results even more rapidly than the standard DA-MCMC. In a computationally intensive case study, the run-time for ADA-MCMC is 2–4 times faster than for standard DA-MCMC.

The methodology we consider is general, as our novel method pertains sampling from arbitrary distributions. However, in the interest of our applications, we will focus on Bayesian inference, and then suggest how to implement our ideas for general problems. In Bayesian inference we aim at sampling from the posterior distribution $p(\theta|y) \propto p(y|\theta)p(\theta)$, where θ are model parameters, y denotes data, $p(y|\theta)$ is the likelihood function, and $p(\theta)$ is the prior distribution of θ. We assume that the point-wise evaluation of the likelihood $p(y|\theta)$ (or an approximation thereof) is computationally intensive, because the underlying probabilistic model is complex and/or the data y is large. For those situations, DA-MCMC algorithms turn particularly useful. In the approach originally outlined in Christen and Fox [2005] a DA strategy decomposes an MCMC move into two stages. At the first stage a proposal can either be rejected, according to a "surrogate of the posterior" (one that is computationally cheap to evaluate and chosen to approximate the desired posterior), or be sent to the second stage. If the proposal is not rejected at the first stage, at the second stage an acceptance probability is used that corrects for the discrepancy between the approximate surrogate and the desired posterior, and at this stage the proposal can finally be accepted or rejected. The advantage of using DA-MCMC is that the computationally expensive posterior only appears in the second stage, whereas the surrogate posterior in the first stage is cheap to evaluate. Therefore, in the first stage the surrogate posterior rapidly screens proposals, and rejects those that are unlikely to be accepted at the second stage, if the surrogate model is reliable. When considering a Bayesian approach, we build a surrogate of the computationally expensive likelihood function, while we assume the cost of evaluating the prior to be negligible. Therefore the expensive likelihood appears only in the second stage. Some implementations of the DA approach in Bayesian inference can be found e.g. in Golightly et al. [2015], Sherlock et al. [2017], and Banterle et al. [2015], and similar approaches based on approximate Bayesian computation (ABC) can be found in Picchini [2014], Picchini and Forman [2016], and Everitt and Rowińska [2017].

In this work, the sequence of computations pertaining the second stage of DA-MCMC are arranged so to find further opportunities to avoid the evaluation of the expensive likelihood. This leads to our accelerated and approximated ADA-MCMC. The computational benefit of using ADA-MCMC is that, unlike DA-MCMC, once a parameter proposal reaches the second stage, the expensive likelihood is not necessarily evaluated, but this comes at the price of introducing an approximation in the sampling procedure. We test and compare delayed-acceptance algorithms, particle marginal methods for exact Bayesian inference, and Markov-chain-within-Metropolis on two case studies: The stochastic Ricker model, and a novel state-space model for protein folding data, with dynamics expressed via a stochastic differential equation (SDE). Therefore, in this work we contribute with: (i) a novel, approximate and accelerated delayed-acceptance MCMC algorithm, and (ii) a novel double-well potential state-space model for protein folding data. For practical applications, we use Gaussian processes to specify surrogates of the likelihood function, though this is not an essential component of our approach and other surrogates of the likelihood can be considered. We found that the acceleration produced by ADA-MCMC, compared to DA-MCMC, is dependent on the specific application. If the exact or approximate likelihood function used in the second stage of the algorithm is not computationally intensive to evaluate, then our method produces negligible benefits. Therefore, the use of our ADA-MCMC, just as the standard DA-MCMC, is beneficial when each evaluation of the likelihood has a non-negligible impact on the total computational budget. Then, the time savings due to ADA-MCMC are proportional to the number of MCMC iterations where the evaluation of the likeli-

hood at the second stage is avoided. In terms of inference quality, we find that ADA-MCMC returns results that are very close to DA-MCMC, so our approximations do not seem to harm the accuracy of the resulting inference.

The outline of this paper is as follows: The delayed-acceptance (DA) scheme and our novel accelerated DA algorithm are introduced in a general framework in Section 2. The Gaussian process (GP) surrogate model is introduced in Section 3. The DA-GP-MCMC algorithm and the accelerated version ADA-GPMCMC are introduced in Section 4. A simulation study for the stochastic Ricker model is in Section 5.1. The protein folding data and the novel double-well potential stochastic differential equation model are introduced in Section 5.2. A discussion in Section 6 closes our work. Further supplementary material is available, outlining: particle Markov chain Monte Carlo methods for state-space models, implementation guidelines for the algorithms, a further simulation study, and diagnostic analyses. The code used to generate results can be found at https://github.com/SamuelWiqvist/adamcmcpaper and in the supplementary material.

2 Delayed-acceptance MCMC

We first introduce the delayed-acceptance (DA-MCMC) scheme due to Christen and Fox [2005] in full generality, then we specialize it for Bayesian inference. Our accelerated delayed-acceptance (ADA-MCMC) algorithm is introduced in section 2.1. We are interested in sampling from some distribution $p(x)$ using Metropolis-Hastings [Hastings, 1970]. Metropolis-Hastings proceeds by evaluating random moves produced by a Markov kernel from the current value of x to a new $x^\star$. The sequence of accepted moves forms a Markov chain having $p(x)$ as stationary distribution. Now, assume that the point-wise evaluation of $p(x)$ is computationally expensive. The main idea behind a DA-MCMC approach is to delay (or avoid as much as possible) the evaluation of the computationally expensive $p(x)$, by first trying to early-reject the proposal $x^\star$ using some surrogate (cheap to evaluate) deterministic or stochastic model $\tilde{p}(x)$. To enable early-rejections while still targeting the distribution $p(x)$, a two-stages acceptance scheme is introduced in Christen and Fox [2005]. Say that we are at the rth iteration of the Metropolis-Hastings algorithm, and denote with x^{r-1} the state of the chain produced at the previous iteration. At the "first stage" of DA-MCMC we evaluate the acceptance probability (though at this stage we do not really accept any proposal as explained below)

$$\alpha_1 = \min\Big(1, \frac{\tilde{p}(x^\star)}{\tilde{p}(x^{r-1})} \cdot \frac{g(x^{r-1}|x^\star)}{g(x^\star|x^{r-1})}\Big), \tag{1}$$

where $g(x|y)$ is the transition kernel used to generate proposals, i.e. at the rth iteration $x^\star \sim g(x|x^{r-1})$. If the proposal $x^\star$ "survives" the first stage (i.e. if it is not rejected) it is then promoted to the second stage where it is accepted with probability α_2,

$$\alpha_2 = \min\Big(1, \frac{p(x^\star)}{p(x^{r-1})} \cdot \frac{\tilde{p}(x^{r-1})}{\tilde{p}(x^\star)}\Big). \tag{2}$$

Therefore $x^\star$ can only be accepted at the second stage, while it can be rejected both at the first and second stage. A computational speed-up is obtained when $x^\star$ is early-rejected at the first stage, as there the expensive $p(x^\star)$ is not evaluated. Hence, to obtain a significant speed-up it is important to early-reject "bad" proposals that would likely be rejected at the second stage. The probability α_2 corrects for the approximation introduced in the first stage and the resulting Markov chain has the correct stationary distribution $p(x)$. This result holds if g is p-irreducible and reversible, and if $g(x|y) > 0$ implies $\tilde{p}(x) > 0$. From (2) it is evident how the surrogate model acts as a proposal distribution. See Franks and Vihola [2017] for a comparison in terms of asymptotic variances of Monte Carlo estimators provided via importance sampling, pseudo-marginal and delayed-acceptance methods.

In a Bayesian framework we are interested in sampling from the posterior $p(\theta|y) \propto p(y|\theta)p(\theta)$. Furthermore, for the cases of interest to us, the log-likelihood function (or an approximation thereof)

$\ell(\theta) := \log p(y|\theta)$, is computationally expensive while the prior distribution is assumed cheap to evaluate. By introducing a deterministic or stochastic surrogate likelihood $\tilde{L}(\theta) := \exp(\tilde{\ell}(\theta))$, DA has first stage acceptance probability α_1, where

$$\alpha_1 = \min\Big(1, \frac{\tilde{L}(\theta^\star)}{\tilde{L}(\theta^{r-1})} \cdot \frac{p(\theta^\star)}{p(\theta^{r-1})} \cdot \frac{g(\theta^{r-1}|\theta^\star)}{g(\theta^\star|\theta^{r-1})}\Big),$$

with transition kernel g. Similarly, by setting $L(\theta) := \exp(\ell(\theta))$, the second stage acceptance probability is

$$\alpha_2 = \min\Big(1, \frac{L(\theta^\star)}{L(\theta^{r-1})} \cdot \frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)}\Big).$$

An extension of the DA-MCMC scheme due to Sherlock et al. [2017] is to generate a proposal $\theta^\star$ from a different transition kernel $\tilde{g}(\cdot|\theta^{r-1})$, and with a small but positive probability $\beta_{MH} \in (0,1)$ allow the evaluation of the proposal $\theta^\star$ in an ordinary Metropolis-Hastings algorithm, with acceptance probability denoted α_{MH},

$$\alpha_{MH} = \min\Big(1, \frac{L(\theta^\star)}{L(\theta^{r-1})} \cdot \frac{p(\theta^\star)}{p(\theta^{r-1})} \cdot \frac{\tilde{g}(\theta^{r-1}|\theta^\star)}{\tilde{g}(\theta^\star|\theta^{r-1})}\Big). \tag{3}$$

In this case the proposal can be immediately accepted or rejected as in a regular MCMC. The transition kernel g should have a somewhat larger variance than $\tilde{g}$. With probability $1-\beta_{MH}$ a proposal is instead evaluated using the two-stages DA-MCMC algorithm. When considering this "extended version" of DA-MCMC (where β_{MH} is introduced) it is preferable to use a small β_{MH} in order not to lose too much of the acceleration implied by a DA approach. Our experience also indicates that this extension can be critical to better explore the tails of the posterior distribution, compared to a standard DA-MCMC that uses $\beta_{MH} = 0$. This "mixture" of the two Metropolis-Hastings kernels (i.e. the acceptance kernel for the DA scheme, and the acceptance kernel in (3)) produces a valid MCMC algorithm, since both kernels in the standard cases target the correct posterior [Rosenthal and Roberts, 2007].

2.1 Accelerated delayed-acceptance MCMC

There have been a number of attempts at accelerating the original DA-MCMC of Christen and Fox [2005]. For example, in a Bayesian framework, Banterle et al. [2015] propose to break down the posterior into the product of d chunks. The Metropolis-Hastings acceptance ratio becomes the product of d acceptance ratios, each of which can be sequentially evaluated against one of d independent uniform variates. The acceleration is given by the possibility to "early-reject" a proposal, as soon as one of those acceptance ratios leads to a rejection (in the same spirit of Solonen et al., 2012). However, an acceptance requires instead the scanning of all d components, i.e. the full posterior. Quiroz et al. [2017] never use the full data set in the second stage of DA and instead construct an approximated likelihood from subsamples of the data, which is particularly relevant for Big Data problems (see references therein and Angelino et al., 2016). Remarkably, Quiroz et al. [2017] prove that even when the full likelihood is approximated using data subsamples, the resulting chain has the correct stationary distribution. However, they assume data to be conditionally independent, a strong condition which does not apply to case studies considered in the present work.

We now introduce the novel, accelerated DA-MCMC algorithm, shortly ADA-MCMC. The main idea behind ADA-MCMC is that, under some assumptions on how the likelihood function and the surrogate model relate, it is possible to arrange the computations in the second stage to obtain an acceleration in the computations. This is implied by the possibility to avoid the evaluation of the expensive likelihood in the second stage, in some specific circumstances. However, this also implies that ADA-MCMC is an approximated procedure, since a proposal can sometimes be accepted according to the surrogate model. We introduce ADA-MCMC in a Bayesian setting where the surrogate model pertains the likelihood function. However, the idea can straightforwardly be adapted to the

case where a surrogate model of a generic distribution $p(x)$ is used, as in Equations (1)-(2). The more general setting is briefly described later in this section. As previously mentioned, at the rth iteration the DA algorithm is governed by the values of the likelihood function $L(\theta^\star)$ and $L(\theta^{r-1})$, and the values of the surrogate model $\tilde{L}(\theta^\star)$ and $\tilde{L}(\theta^{r-1})$. These four values can be considered arranged in four mutually exclusive scenarios:

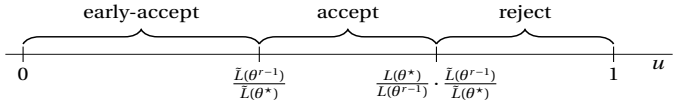
$$\begin{aligned}
&\text{case 1) } \tilde{L}(\theta^\star) > \tilde{L}(\theta^{r-1}) \textit{ and } L(\theta^\star) > L(\theta^{r-1}),\\
&\text{case 2) } \tilde{L}(\theta^\star) < \tilde{L}(\theta^{r-1}) \textit{ and } L(\theta^\star) < L(\theta^{r-1}),\\
&\text{case 3) } \tilde{L}(\theta^\star) > \tilde{L}(\theta^{r-1}) \textit{ and } L(\theta^\star) < L(\theta^{r-1}),\\
&\text{case 4) } \tilde{L}(\theta^\star) < \tilde{L}(\theta^{r-1}) \textit{ and } L(\theta^\star) > L(\theta^{r-1}).
\end{aligned}$$

We study each case separately to investigate any opportunity for accelerating the computations in the second stage of DA-MCMC, under the assumption that the relations between the evaluations of $\tilde{L}$ and L hold. Afterwards, we suggest ways to determine approximately which of the four possibilities we should assume to hold, for any new proposal $\theta^\star$, without evaluating the expensive likelihood $L(\theta^\star)$.

Case 1) Under the assumption that $\tilde{L}(\theta^\star) > \tilde{L}(\theta^{r-1})$ *and* $L(\theta^\star) > L(\theta^{r-1})$ it is clear that $\frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)} < 1$ *and* $\frac{L(\theta^{r-1})}{L(\theta^\star)} < 1$. It also holds that

$$\frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)} < \frac{L(\theta^\star)}{L(\theta^{r-1})} \cdot \frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)}. \tag{4}$$

Hence, the acceptance region for the second stage can be split in two parts, where one part is "governed" by $\frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)}$ only. To clarify, at the second stage of the standard DA-MCMC, acceptance of a proposed $\theta^\star$ takes place if $u < \frac{L(\theta^\star)}{L(\theta^{r-1})} \cdot \frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)}$ where $u \sim U(0,1)$ is uniformly distributed in [0,1], hence, the acceptance region is $[0, \frac{L(\theta^\star)}{L(\theta^{r-1})} \cdot \frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)}]$. However, because of (4) we are allowed to further decompose the acceptance region, as presented below:



Hence, if a proposal $\theta^\star$ has survived the first stage and we assume that we are in case 1, we can first check whether we can "early-accept" the proposal (i.e. without evaluating the expensive likelihood), that is, check if

$$u < \frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)}, \tag{5}$$

and if this is the case $\theta^\star$ is (early)-accepted and stored, and we can move to the next iteration of ADA-MCMC. If $\theta^\star$ is not early-accepted, we can look into the remaining part of the [0,1] segment to determine if the proposal can be accepted or rejected. Hence, when early-acceptance is denied, the expensive likelihood $L(\theta^\star)$ is evaluated and the proposal is accepted and stored if

$$u < \frac{L(\theta^\star)}{L(\theta^{r-1})} \cdot \frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)}, \tag{6}$$

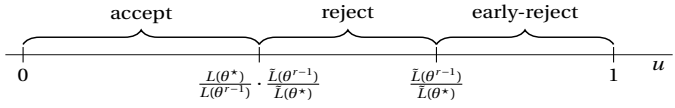
and rejected otherwise, and we can move to the next iteration of ADA-MCMC. Since the acceptance region for the second stage is split in two parts (early-acceptance and acceptance), the *same* random number u is used in (5) and (6). By splitting the region it is possible to early-accept proposals without evaluating $L(\theta^\star)$, and thereby obtaining a speed-up.

Case 2) If this case holds, then $\frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)} > 1$ and $\frac{L(\theta^\star)}{L(\theta^{r-1})} < 1$. Hence, it is not possible to obtain any early-accept or early-reject opportunity in this case.

Case 3) If this case holds, then $\frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)} < 1$ and $\frac{L(\theta^\star)}{L(\theta^{r-1})} < 1$. Hence, it also holds that

$$\frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)} > \frac{L(\theta^\star)}{L(\theta^{r-1})} \cdot \frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)}.$$

The rejection region is $[\frac{L(\theta^\star)}{L(\theta^{r-1})} \cdot \frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)}, 1]$ and this can be split in two parts, where one part is only governed by $\frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)}$, see below:



By simulating a $u \sim U(0,1)$, we can first check if the proposal can be early-rejected. This happens if $u > \frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)}$. If the proposal is not early-rejected, it is accepted if

$$u < \frac{L(\theta^\star)}{L(\theta^{r-1})} \cdot \frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)},$$

and rejected otherwise. Hence, in case 3 there is a chance to early-reject $\theta^\star$ without evaluating $L(\theta^\star)$.

Case 4) Under the assumption we have that $\frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)} > 1$ and $\frac{L(\theta^\star)}{L(\theta^{r-1})} > 1$, and we can immediately accept the proposal without evaluating $L(\theta^\star)$, since $\frac{L(\theta^\star)}{L(\theta^{r-1})} \cdot \frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)} > 1$.

Clearly, assuming a specific case to be the "right one", for proposal $\theta^\star$, is a decision subject to probabilistic error. This is why ADA-MCMC is an approximate version of DA-MCMC. Of course, the crucial problem is to determine which of the four cases to assume to hold for the proposed $\theta^\star$. One method is to consider a pre-run of some MCMC algorithm, to estimate the probability p_j for each of the four different cases, where p_j is the true but unknown probability that case j holds, $j = 1, ..., 4$. This is of course a possibly computationally heavy procedure, however, for the specific algorithms we study in Section 4, such a pre-run is necessary to construct the surrogate model for the log-likelihood, hence the estimation of the p_j comes as a simple by-product of the inference procedure. Then, once the estimates $\hat{p}_j$ are obtained, for a new $\theta^\star$ one first checks if $\frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)} < 1$ or if $\frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)} > 1$. If $\frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)} > 1$ then we can either be in case 2 or 4. We toss a uniform u and if $u < \hat{p}_2$ case 2 is selected with probability $\hat{p}_2$ (and otherwise case 4 is selected, since $\hat{p}_4 = 1 - \hat{p}_2$). Correspondingly, if $\frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)} < 1$ then we can be either in case 1 or 3. We toss a uniform $u \sim U(0,1)$, and if $u < \hat{p}_1$ case 1 is selected (otherwise case 3 is selected, since $\hat{p}_3 = 1 - \hat{p}_1$). Another approach is to model the probabilities as a function of θ. Hence, we are then interested in computing the probabilities $\hat{p}_1(\theta)$, $\hat{p}_2(\theta)$,$\hat{p}_3(\theta)$, and $\hat{p}_4(\theta)$. For this task, we can for instance use logistic regression, or some other classification algorithm. The problem of the selection of cases 1–4 is discussed in detail in Section 4.1.

We stated early that ADA can also be used in a non-Bayesian setting, where we target a generic distribution $p(x)$ for some $x \in \mathcal{X}$. In that case we need to introduce a corresponding surrogate model $\tilde{p}(x)$. The rth iteration of ADA will then be governed by the four values $\tilde{p}(x^\star)$, $\tilde{p}(x^{r-1})$, $p(x^\star)$, and $p(x^{r-1})$, where $x^\star$ is a proposed value $x^* \in \mathcal{X}$. These can be arranged into four cases, similarly to what previously described: case 1) $\tilde{p}(x^\star) > \tilde{p}(x^{r-1})$ *and* $p(x^\star) > p(x^{r-1})$, 2) $\tilde{p}(x^\star) < \tilde{p}(x^{r-1})$ *and* $p(x^\star) < p(x^{r-1})$, 3) $\tilde{p}(x^\star) > \tilde{p}(x^{r-1})$ *and* $p(x^\star) < p(x^{r-1})$, and 4) $\tilde{p}(x^\star) < \tilde{p}(x^{r-1})$ *and* $p(x^\star) > p(x^{r-1})$. Therefore, by adapting the methodology, possibilities for early-rejection and early-acceptance of a proposal $x^\star$ can straightforwardly be obtained regardless of whether we pursue a Bayesian analysis or not.

3 Modeling the log-likelihood function using Gaussian processes

We have outlined our methodology without reference to a specific choice for the surrogate likelihood. A possibility is to use Gaussian process regression to obtain a surrogate log-likelihood $\log \tilde{L}$. Gaussian processes (GPs) is a class of statistical models that can be used to describe the uncertainty about an unknown function. In our case, the unknown function is the log-likelihood $\ell(\theta) = \log p(y|\theta)$. A GP has the property that the joint distribution for the values of the unknown function, at a finite collection of points, has a multivariate normal distribution. As such, each Gaussian process is fully specified by a mean function m, and a covariance function k [Rasmussen and Williams, 2006]. We introduce a GP regression model, similar to the one used in Drovandi et al. [2018], as a computationally cheap proxy to the unknown log-likelihood $\ell(\theta)$. Our GP model uses covariates that are powers and interactions of the d parameters of interest $\theta = (\theta_1, ..., \theta_d)$ (see the supplementary material). The GP model assumes

$$\ell(\theta) \sim \mathcal{GP}(m_\beta(\theta), k_\phi(\theta, \theta')),$$

where $\eta = [\phi, \beta]$ are the auxiliary parameters for the mean and covariance function respectively. Since η is in general unknown, this must be estimated by fitting the GP model to some "training data". In our case, training data is obtained by running a a number of preliminary MCMC iterations, and collect all generated parameter proposals and corresponding log-likelihood values. The GP regression considers the log-likelihood values as "responses" and the proposed parameters are used to construct the covariates. Once $\hat{\eta}$ is available, then for any new $\theta^\star$ we obtain a proxy to the unknown log-likelihood that is computationally much faster to evaluate than $\ell(\theta^\star)$. The training data we fit the GP model to is denoted $\mathcal{D}$, and how this data is collected is explained in Section 4. Using the same assumptions for the Gaussian process model as in Drovandi et al. [2018], we have that the predictive distribution for the GP model is available in closed form. Therefore, for given $\mathcal{D}$ and $\hat{\eta}$ we can easily produce a draw from said distribution, which is Gaussian, and given by

$$\ell(\theta^\star)|\mathcal{D}, \hat{\eta} \sim \mathcal{N}(\bar{\ell}(\theta^\star), \mathrm{Var}(\ell(\theta^\star))). \tag{7}$$

See the supplementary material for the definitions of $\bar{\ell}(\theta)$ and $\mathrm{Var}(\ell(\theta))$. It is computationally very rapid to produce draws from (7) at any new $\theta^\star$, which is why we use GP prediction as a surrogate of the log-likelihood within DA algorithms. The derivation of (7), and more details pertaining the GP model are found in the supplementary material.

4 Delayed-acceptance Gaussian process Markov chain Monte Carlo

We now make use of the fitted GP model discussed in Section 3 as a surrogate of the log-likelihood function, within DA-MCMC and ADA-MCMC. By sampling a GP log-likelihood $\ell_{GP}(\theta^\star) := \ell(\theta^\star)|\mathcal{D}, \hat{\eta}$ from (7) for some $\theta^\star$, we denote with $\hat{L}_{GP}(\theta^\star) = \exp(\ell_{GP}(\theta^\star))$ the GP prediction of the corresponding likelihood function. In addition to be computationally intensive to evaluate, the true likelihood $L(\theta)$ might also be unavailable in closed form. However, it is often possible to obtain Monte Carlo approximations returning non-negative unbiased estimates of L. We denote with $\hat{L}_u(\theta)$ such unbiased estimate. For our case studies, $\hat{L}_u(\theta)$ is obtained via sequential Monte Carlo (SMC, also known as particle filter, see Kantas et al., 2015 and Schön et al., 2018 for reviews). A simple example of SMC algorithm (the bootstrap filter) and its use within particle-marginal methods [Andrieu and Roberts, 2009] for inference in state-space models are presented in the supplementary materials. Two types of pseudo-marginal methods, particle MCMC (PMCMC) and Markov-chain-within-Metropolis (MCWM), are there described. In the supplementary material we give a brief technical presentation of PMCMC and MCWM.

Notice that MCMC algorithms based on GP-surrogates have already been considered, e.g. in Meeds and Welling [2014] and Drovandi et al. [2018]. Meeds and Welling [2014] assume that the latent process has a Gaussian distribution with unknown moments, and these moments are estimated

via simulations using "synthetic likelihoods". There, the discrepancy between the simulated (Gaussian) latent states and observed data is evaluated using a Gaussian ABC kernel, where ABC stands for "approximate Bayesian computation", see Marin et al. [2012] for a review. This computationally expensive setting is fitted to "training data", then used in place of the (unknown) likelihood into a pseudo-marginal MCMC algorithm. The work in Drovandi et al. [2018] builds up on the ideas found in Meeds and Welling [2014], with the difference that the former does not use synthetic likelihoods nor ABC to produce training data. Instead they use the MCWM algorithm to collect many log-likelihood evaluations at all proposed parameter values, then fit a GP regression model on these training data. Finally, they use the fitted GP regression in a pseudo-marginal algorithm, without ever resorting to expensive likelihood calculations. As opposed to Drovandi et al. [2018], we make use of both a surrogate of the likelihood and (with low frequency) of the expensive likelihood approximated via a particle filter. We call DA-GP-MCMC a delayed acceptance MCMC algorithm using predictions from GP regression as a surrogate of the likelihood function. Similarly, we later introduce our accelerated version ADA-GP-MCMC.

The DA-GP-MCMC procedure is detailed in Algorithm 1. Using the notation in Section 2, we now have that the first stage acceptance probability for DA-GP-MCMC is

$$\alpha_1 = \min\Big(1, \frac{\hat{L}_{GP}(\theta^\star)}{\hat{L}_{GP}(\theta^{r-1})} \cdot \frac{p(\theta^\star)}{p(\theta^{r-1})} \cdot \frac{g(\theta^{r-1}|\theta^\star)}{g(\theta^\star|\theta^{r-1})}\Big).$$

The second stage acceptance probability is

$$\alpha_2 = \min\Big(1, \frac{\hat{L}_u(\theta^\star)}{\hat{L}_u(\theta^{r-1})} \cdot \frac{\hat{L}_{GP}(\theta^{r-1})}{\hat{L}_{GP}(\theta^\star)}\Big).$$

As mentioned in Section 2, for our applications we found it beneficial to use the extended DA-MCMC introduced in Sherlock et al. [2017]. However, this is in general not a requirement for using DA-MCMC. The DA-GP-MCMC algorithm is preceded by the following two steps, required to collect training data and fit the GP regression to these data:

1. Collect training data using MCWM: A MCWM algorithm is run to approximately target $p(\theta|y)$, where a bootstrap particle filter using N particles is employed to obtain $\hat{L}_u(\theta)$, until the chain has reached apparent stationarity. When using MCWM we do not target the exact posterior for a finite number of particles N, however, this is not a concern to us. In fact, we use MCWM as in Drovandi et al. [2018], namely to "harvest" a large number of (approximate) log-likelihood function evaluations, in order to learn the dependence between loglikelihoods and corresponding parameters. Indeed, in this phase we store as training data $\mathscr{D}$ *all* the proposed parameters $\theta^\star$ (regardless of whether these are accepted or rejected from MCWM) and their corresponding log-likelihoods $\ell_u(\theta^\star)$. Hence, all parameter proposals and corresponding log-likelihoods from MCWM (excluding some sufficiently long burnin period) are stored as training data $\mathscr{D} = \{\theta^{*i}, \ell_u^{*i}\}$, (where here the superscript i ranges from 1 to the number of iterations post-burnin). We also collect the generated Markov chain θ^i and their corresponding log-likelihood estimations in $\tilde{\mathscr{D}} = \{\theta^i, \ell_u^i\}$. Basically the difference between $\mathscr{D}$ and $\tilde{\mathscr{D}}$ is that parameters θ^i in the latter are the standard output of a Metropolis-Hastings procedure, i.e. $\tilde{\mathscr{D}}$ may contain "repeated parameters" (when rejections occur). Instead $\mathscr{D}$ contains all simulated proposals. We motivate the use for set $\tilde{\mathscr{D}}$ in Section 4.1.

2. Fit the GP model: The Gaussian process model is fitted to the training data $\mathscr{D}$ using the method described in Section 3.

Algorithm 1 DA-GP-MCMC algorithm

Input: Number of iterations R, a GP model fitted to the training data, a starting value θ^0 and corresponding $\hat{L}_u(\theta^0)$.
Output: The chain $\theta^{1:R}$.

for $r = 1, ..., R$ **do**
  Propose $\theta^\star \sim g(\cdot|\theta^{r-1})$. ▷ Run two stages DA scheme
  Sample from (7) to predict independently $\ell_{GP}(\theta^\star)$ and $\ell_{GP}(\theta^{r-1})$. Define $\hat{L}_{GP}(\theta^\star) := \exp(\ell_{GP}(\theta^\star))$ and $\hat{L}_{GP}(\theta^{r-1}) := \exp(\ell_{GP}(\theta^{r-1}))$.
  Compute $\alpha_1 = \min\left(1, \frac{\hat{L}_{GP}(\theta^\star)}{\hat{L}_{GP}(\theta^{r-1})} \cdot \frac{g(\theta^{r-1}|\theta^\star)}{g(\theta^\star|\theta^{r-1})} \cdot \frac{p(\theta^\star)}{p(\theta^{r-1})}\right)$.
  Draw $u \sim \mathcal{U}(0,1)$.
  if $u > \alpha_1$ **then** ▷ Early-reject
    Set $\theta^r = \theta^{r-1}$.
  else
    Compute $\hat{L}_u(\theta^\star)$. ▷ Second stage update scheme
    Compute $\alpha_2 = \min(1, \frac{\hat{L}_u(\theta^\star)}{\hat{L}_u(\theta^{r-1})} \cdot \frac{\hat{L}_{GP}(\theta^{r-1})}{\hat{L}_{GP}(\theta^\star)})$.
    Draw $u \sim \mathcal{U}(0,1)$.
    if $u \leq \alpha_2$ **then** ▷ Accept proposal
      Set $\theta^r = \theta^\star$.
    else
      Set $\theta^r = \theta^{r-1}$. ▷ Reject proposal
    end if
  end if
end for

4.1 Accelerated delayed-acceptance Gaussian process MCMC

Our accelerated delayed-acceptance Gaussian process MCMC algorithm (ADA-GP-MCMC) is described in Algorithm 2. Same as for DA-GP-MCMC, also ADA-GP-MCMC is preceded by two phases (collection of training data and GP regression). After fitting the GP model, the training data is also used to produce a "selection method" for the four cases introduced in Section 2.1. As already mentioned in Section 2.1, we can either select which case to use independently of the current proposal $\theta^\star$, or make the selection of cases a function of $\theta^\star$. We introduce three selection methods, where the first one selects which case to assume independently of $\theta^\star$, while the other two depend on the proposal.

Biased coin: In the most naive approach, selecting a case between 1 and 3, or between 2 and 4 can be viewed as the result of tossing a biased coin. Hence, we just compute the relative frequency of occurrence for cases 1, 2, 3 and 4 (see Section 2.1) as observed in the training data. These are obtained as follows: using the fitted GP model we predict log-likelihoods $\ell_{GP}(\theta) \equiv \ell(\theta)|\mathscr{D}, \hat{\eta}$ using (7) for all collected $\theta \in \Theta$ (Θ denotes the matrix of the θ proposals that belong to the training data $\mathscr{D}$). Then we obtain corresponding $\hat{L}_{GP} := \exp(\ell_{GP}(\theta))$, for all $\theta \in \Theta$. Now, since all the corresponding $\hat{L}_u(\theta)$ are already available as training data, it is possible to compute said relative frequencies $\hat{p}_j$ of occurrence for each case j ($j = 1, .., 4$). At iteration r of the ADA-GP-MCMC algorithm, for proposal $\theta^\star$, and supposing we have survived the first stage, then if $\hat{L}_{GP}(\theta^\star) > \hat{L}_{GP}(\theta^{r-1})$ we draw from the Bernoulli($\hat{p}_1$) distribution and go for case 1 if the draw equals one, and go for case 3 otherwise. If instead $\hat{L}_{GP}(\theta^\star) < \hat{L}_{GP}(\theta^{r-1})$ we draw from Bernoulli($\hat{p}_2$) and go for case 2 if the draw equals one, and go for case 4 otherwise.

State-dependent selection: The biased coin model does not take into account the specific value of the current proposal $\theta^\star$, that is, the same $\hat{p}_j$ are applied to all proposals during a run of ADA-GP-MCMC. We could instead estimate $\hat{p}_j(\theta)$ using logistic regression or a decision tree model. When using logistic regression, we have two regression models to estimate, one for cases 1 and 3, and one for cases 2 and 4. By combining the training data $\mathscr{D}$, and the accepted proposals stored in $\tilde{\mathscr{D}}$, we have access to both the particle filter evaluations corresponding to all generated proposals, and to the ones for the accepted proposals. Using $\mathscr{D}$ and $\tilde{\mathscr{D}}$ we can now classify which case each proposal *should* belong to. This is done by computing GP predictions, independently for both sets of parameters stored in $\mathscr{D}$ and $\tilde{\mathscr{D}}$. Note, after computing the GP predictions we have (i) particle filter predictions and GP predictions for all proposals in $\mathscr{D}$, i.e. $\hat{L}_u(\theta^\star)$ and $\hat{L}_{GP}(\theta^\star)$, and (ii) particle filter predictions

and GP predictions for all accepted proposals in $\tilde{\mathscr{D}}$, hence, $\hat{L}_u(\theta^{r-1})$ and $\hat{L}_{GP}(\theta^{r-1})$. We can now loop over the proposals in the training data and assign labels for which of the four cases each proposal belongs to. As an example, after labelling is performed, all proposals in the training data that are classified to belong to case 1 or 3 are denoted $\theta^{\star}_{1,3}$, and an associated indicator vector $y_{1,3}$, having 1 for proposals belonging to case 1 and 0 for proposals belonging to case 3, is created. We now fit a logistic regression model on $\{\theta^{\star}_{1,3}, y_{1,3}\}$, where the $\theta^{\star}_{1,3}$ take the role of "covariates" and the $y_{1,3}$ are binary "responses". We denote with $\hat{p}_1(\theta)$ the resulting fitted probability of selecting case 1 (so that $\hat{p}_3(\theta) = 1 - \hat{p}_1(\theta)$). In a similar way, after labelling is performed, all proposals in the training data that are classified to belong to case 2 or 4 are denoted $\theta^{\star}_{2,4}$, with associated indicator vector $y_{2,4}$. We fit a logistic regression model on $\{\theta^{\star}_{2,4}, y_{2,4}\}$ to obtain $\hat{p}_2(\theta)$ (and $\hat{p}_4(\theta) = 1 - \hat{p}_2(\theta)$).

All the above is preliminary to starting ADA-GP-MCMC. Then we proceed as described for the biased coin case, with minimal notation adjustment. Namely for a new proposal $\theta^{\star}$, if $\hat{L}_{GP}(\theta^{\star}) > \hat{L}_{GP}(\theta^{r-1})$ we decide between case 1 and 3 by drawing from Bernoulli($\hat{p}_1(\theta^{\star})$). If instead $\hat{L}_{GP}(\theta^{\star}) < \hat{L}_{GP}(\theta^{r-1})$ we draw from Bernoulli($\hat{p}_2(\theta^{\star})$) to decide between case 2 and 4. Alternatively, in place of a logistic regression model we can use decision trees, but still employ the same ideas as for logistic regression. Decision trees can perform better at modeling non-linear dependencies in the data. Importantly, a decision tree does not produce an estimation of the probabilities for each case (hence, we do not obtain a direct estimation of $\hat{p}_j(\theta)$), instead a classification decision is computed, which will directly select which case to assume for the given proposal $\theta^{\star}$. We obtained the best results with the decision tree model. We have found beneficial to include, as a covariate in the decision tree model, the ratio between the GP-based log-likelihood estimates at the current proposal and the previous log-likelihood estimate.

In conclusion, we have introduced three selection methods. In Algorithm 2 the selection methods are denoted $s_{13}(\cdot)$ (for selection between case 1 and 3) and $s_{24}(\cdot)$ (for selecting between case 2 and 4), to highlight the fact that different selection methods are available. In the supplementary material we describe how to test the fit of the GP model and the performance of the selection method.

Algorithm 2 ADA-GP-MCMC algorithm

Input: Number of iterations R, a GP model fitted to the training data, model $s_{13}()$ to select between case 1 and 3, model $s_{24}()$ to select between case 2 and 4, a starting value θ^0 and corresponding $\hat{L}_u(\theta^0)$.

for $r = 1, ..., R$ **do**
  Propose $\theta^{\star} \sim g(\cdot|\theta^{r-1})$. ▷ Run A-DA scheme
  Sample from the predictive distribution of the GP model to obtain independently $\ell_{GP}(\theta^{\star})$ and $\ell_{GP}(\theta^{r-1})$. Define $\hat{L}_{GP}(\theta^{\star}) := \exp(\ell_{GP}(\theta^{\star}))$ and $\hat{L}_{GP}(\theta^{r-1}) := \exp(\ell_{GP}(\theta^{r-1}))$.
  Compute $\alpha_1 = \min\left(1, \frac{\hat{L}_{GP}(\theta^{\star})}{\hat{L}_{GP}(\theta^{r-1})} \cdot \frac{g(\theta^{r-1}|\theta^{\star})}{g(\theta^{\star}|\theta^{r-1})} \cdot \frac{p(\theta^{\star})}{p(\theta^{r-1})}\right)$.
  Draw $u \sim \mathscr{U}(0,1)$.
  if $u < \alpha_1$ **then** ▷ Run second stage of the A-DA scheme
    if $\hat{L}_{GP}(\theta^{\star}) > \hat{L}_{GP}(\theta^{r-1})$ **then**
      Select case 1 or 3 according to the model $s_{13}(\theta^{\star})$.
      Run the accelerated delayed-acceptance scheme for the selected case.
    else
      Select case 2 or 4 according to the model $s_{24}(\theta^{\star})$.
      Run the accelerated delayed-acceptance scheme for the selected case.
    end if
  else ▷ Early-reject
    Set $\theta^r = \theta^{r-1}$.
  end if
end for

5 Case studies

In Section 5.1 we consider the Ricker model, which has been used numerous times as a toy model to compare inference methods (e.g. Fearnhead and Prangle [2012], Fasiolo et al. [2016] to name a few). In Section 5.2 we consider a novel double-well potential stochastic differential equation (DW-SDE) model for protein folding data, which is a considerably more complex case study. An additional simulation study for the DW-SDE model, diagnostics and further methodological sections are presented in

the supplementary material. The code can be found at https://github.com/SamuelWiqvist/adamcmcpaper.

5.1 Ricker Model

The Ricker model is used in ecology to describe how the size of a population varies in time and follows

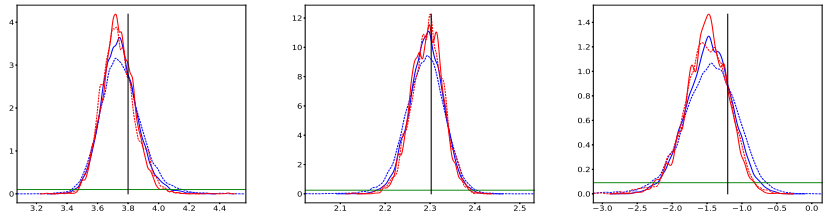
$$\begin{cases} y_{t+1} \sim \mathscr{P}(\phi x_{t+1}), \\ x_{t+1} = r x_t e^{-x_t + \epsilon_t}, \epsilon_t \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2), \end{cases} \tag{8}$$

where $\mathscr{P}(\lambda)$ is the Poisson distribution with mean λ. The $\{x_t\}$ process is a latent (i.e. unobservable) Markov process and realizations from the observable process $\{y_t\}$ are conditionally independent given the latent states, since the ϵ_t are assumed independent. Even though the model is fairly simple its dynamics are highly non-linear and close to chaotic for some choice of the parameter values [Wood, 2010]. The likelihood function is also both analytically and numerically intractable, if evaluated at parameters very incompatible with the observed data, see Fasiolo et al. [2016] for a review of inference methods applied to the Ricker model.

We are interested in $\theta = [\log r, \log \phi, \log \sigma]$, and we use PMCMC, MCWM, DA-GP-MCMC, and ADA-GP-MCMC for this task. That is, MCWM is not only used to provide the training data for fitting a GP regression, but also to provide inference results, in the interest of comparison between methods. PMCMC is used to provide exact Bayesian inference. A data set containing $T = 50$ observations, generated from the model with ground-truth parameters $\theta_{true} = [3.80, 2.30, -1.20]$ at integer sampling times $t \in [1, 2, ..., T]$, and the starting value x_0 for the latent state was deterministically set to $x_0 = 7$ and considered as a known constant throughout.

Results obtained with PMCMC and MCWM are produced using in total 52,000 iterations (including a burnin period of 2,000 iterations), and $N = 1,000$ particles (the standard deviation of the log-likelihood obtained from the particle filter is about 0.5). The proposal distribution was adaptively tuned using the generalized AM algorithm (Andrieu and Thoms, 2008, Mueller, 2010), which is set to target an acceptance rate of 40%. For DA-GP-MCMC algorithm, we used the last 2,000 iterations of a previous MCWM run to obtain training data. Prior to fitting the GP model we removed the 10% of the cases having the lowest log-likelihood values from the training data, as these cases badly affected the GP predictions. After fitting the GP model, we use the "extended" version of the DA algorithm discussed in section 2 and set $\beta_{MH} = 0.15$ (that is a 15% probability to skip the delayed-acceptance step and execute a regular Metropolis-Hastings step), $N = 1,000$, and ran DA-GP-MCMC for further 50,000 iterations. The Gaussian kernels for the Metropolis random walks, g and $\tilde{g}$, were kept fixed during the entire run of the DA-GP-MCMC algorithm: specifically, $\tilde{g}$ used the covariance matrix Σ returned by the final iteration of the MCWM algorithm that was used to collect training data, and g was set to a kernel having slightly larger terms in the covariance, i.e. we used a covariance $a^2\Sigma$ with $a > 1$. An important modification of DA-GP-MCMC as described in Algorithm 1, is that in our case studies the value $\hat{L}_u(\theta^{r-1})$ at the denominator of α_2 is "refreshed". Hence, we employ a MCWM updating procedure in the second stage. This is to obtain a reasonable high acceptance rate and to avoid problems with stickiness. The same modification was used for ADA-GP-MCMC. At the second-stage of the r-th iteration of ADA-GP-MCMC, a decision tree model was used to select a case from the four ones discussed in sections 2.1 and 4.1.

Wide uniform priors were employed for all unknown parameters; $p(\log r) \sim \mathcal{U}(0, 10)$, $p(\log \phi) \sim \mathcal{U}(0, 4)$ and $p(\log \sigma) \sim \mathcal{U}(-10, 1)$. The starting values were also deliberately set far away from the true parameter values: $\log r_0 = 1.10$, $\log \phi_0 = 1.10$, and $\log \sigma_0 = 2.30$. Results are presented in Table 1 and Figure 1. We can conclude that all parameters are well inferred. The results for the different algorithms are also similar. The parameter with the highest estimation uncertainty is σ, which is in not surprising since σ is the parameter that governs the noise in the model, and this is often the hardest parameter to estimate from discretely observed measurements. Notice that results produced by ADA-GP-MCMC are essentially identical to those from DA-GP-MCMC. We find this very encouraging since the most relevant way to judge inference results from the accelerated ADA procedure is to compare those to the standard DA algorithm rather than, say, PMCMC.



(a) $\log r$. (b) $\log \phi$. (c) $\log \sigma$.

Figure 1: Marginal posteriors for the Ricker model: PMCMC (blue solid line), MCWM (blue dashed line), DA-GP-MCMC (red solid line), and ADA-GP-MCMC (red dashed line). Priors distributions are denoted with green lines (these look "cut" as we zoom on the bulk of the posterior), and the true parameter values are marked with black vertical lines.

Table 1: Ricker model: Posterior means (2.5th and 97.5th quantiles) for PMCMC, MCWM, DA-GP-MCMC, and ADA-GP-MCMC.

	True value	PMCMC	MCWM	DA-GP-MCMC	ADA-GP-MCMC
$\log r$	3.80	3.75 [3.53, 4.00]	3.75 [3.51, 4.05]	3.74 [3.54, 3.96]	3.73 [3.54, 3.97]
$\log \phi$	2.30	2.29 [2.21, 2.36]	2.29 [2.20, 2.37]	2.29 [2.23, 2.36]	2.29 [2.22, 2.36]
$\log \sigma$	-1.58	-1.47 [-2.13, -0.85]	-1.46 [-2.3, -0.75]	-1.5 [-2.12, -0.95]	-1.51 [-2.16, -0.92]

Properties of the algorithms are presented in Table 2. Before discussing these results, we emphasize that the benefits of our accelerated procedure are to be considered when the case study has a likelihood that is computationally very challenging, and this is not the case for the present example, see instead Section 5.2. The ADA-GP-MCMC algorithm is the fastest algorithm, though only marginally faster than DA-GP-MCMC (4.2 times faster than MCWM and 1.09 times faster than DA-GP-MCMC), while MCWM is the slowest one. Not surprisingly, PMCMC is almost twice as fast as MCWM, and this is because PMCMC only requires one evaluation of the particle filter per iteration, while the MCWM requires two evaluations. The four algorithms are, however, essentially equally efficient, as from the min ESS/sec values.

The estimated probabilities $\hat{p}_j$ for the four different cases characterizing ADA-GP-MCMC (recall that $\hat{p}_3 = 1 - \hat{p}_1$ and $\hat{p}_4 = 1 - \hat{p}_2$), and the percentage for each case to hold, i.e. the probability that the selected case indeed is the correct one, are presented in Table 3. We notice that the probability for the different cases vary considerably, and also that the percentages that the assumption holds vary for the different cases. We also notice that the performance of the selection algorithm is much better for case 2 than for case 4: this is due to the unbalance of the two classes, meaning that in our training data case 2 occurs more frequently than case 4, and therefore it is more difficult to estimate the latter case accurately.

5.2 Double-well potential stochastic differential equation model for protein folding data

We now consider a computationally intensive case study concerning statistical inference for protein folding data. The challenges for this case study are: (a) the sample size is large, data being a long time-series (about 2.5×10^4 observations), (b) the non-linear dynamics, and (c) the presence of local perturbations. "Protein folding" is the last and crucial step in the transformation of genetic information, encoded in DNA, into a functional protein molecule. Studying the time-dynamics of real protein folding dynamics results in a very high dimensional problem, which is difficult to analyze using exact Bayesian methodology. Therefore, for reasons of simplification and tractability, the dynamics of a

Table 2: Ricker model: Efficiency of PMCMC, MCWM, DA-GP-MCMC, and ADA-GP-MCMC. Timings for (A)DA-GP-MCMC do not include the training data collection and the fitting of the GP model.

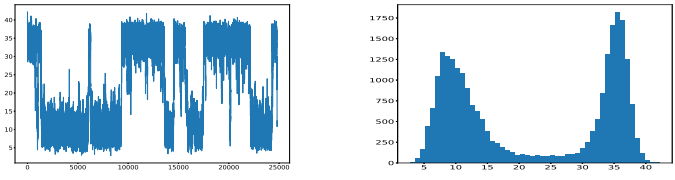
	Seconds per 1000 iter.	Acceptance rate (%)	min ESS/sec	Skip DA run MH update (%)	Early-rejections (%)
PMCMC	20.26	40.21	2.53	NA	NA
MCWM	39.83	39.70	1.26	NA	NA
DA-GP-MCMC	10.32	7.66	1.99	14.75	81.05
ADA-GP-MCMC	9.46	7.89	1.75	15.02	80.49

Table 3: Ricker model: Estimated probabilities for the different cases and percentage of times the assumption for the different cases in the ADA-GP-MCMC algorithm holds.

	Case 1	Case 2	Case 3	Case 4
Est. probab. $(\hat{p}_1, \hat{p}_2, \hat{p}_3, \hat{p}_4)$	0.59	0.91	0.41	0.09
Perc. assum. holds (%)	73.51	88.24	39.80	21.21

protein are often modelled as diffusions along a single "reaction coordinate", that is one-dimensional diffusion models are considered to model a projection of the actual dynamics in high-dimensional space (Best and Hummer, 2011).

The (reaction coordinate) data is in Figure 2. We notice that data have a marginal bimodal structure, with irregular change-points where the mean of the data shifts, and a local noisy structure. A class of models shown to be suitable for statistical modeling of protein folding (at least when these data result into a low-dimensional projection of the original data) is given by stochastic differential equations (SDEs), see Forman and Sørensen [2014] and Picchini and Forman [2016]. Monte Carlo inference methods are very computationally intensive for these models (in Picchini and Forman, 2016 data sub-sampling and special approximate Bayesian computation methods were used to accelerate the inference problem). We now introduce a novel double-well potential stochastic differential equation (DWP-SDE) model for protein folding data. This model is faster to simulate than the one proposed in Forman and Sørensen [2014] and Picchini and Forman [2016]. The DWP-SDE model is



(a) (b)

Figure 2: Data time course (left) and its marginal distribution (right).

defined as

$$\begin{cases} z_t = x_t + y_t, \\ dx_t = -\nabla V(x_t)\,dt + \sigma\, dW_t^x, \\ dy_t = -\kappa\, y_t\, dt + \sqrt{2\kappa\gamma^2}\, dW_t^y. \end{cases} \tag{9}$$

Here $\{z_t\}$ is the observable process, consisting in the sum of the solutions to the double-well potential SDE process $\{x_t\}$ and process $\{y_t\}$, the latter being unobservable and representing autocorrelated

error. Here $\nabla V(\cdot)$ is the gradient of the double-well potential function $V(\cdot)$ with respect to x_t, further specified by six parameters introduced in (10). Finally W_t^X and W_t^Y are independent standard Wiener processes, that is their increments dW_t^X and dW_t^Y are independent, Gaussian distributed with zero mean and variance dt. We consider the following double-well potential function

$$V(x) = \frac{1}{2}\left|\frac{1}{2}|x-c|^{p_1} - d + gx\right|^{p_2} + \frac{1}{2}Ax^2, \tag{10}$$

which is based on the potential described in equation 1 in Fang et al. [2017]. The formulation in (10) is fairly general, in the sense that many different potentials can be specified by varying its parameters. The parameters in (10) have the following interpretation: c specifies the location for the potential (i.e. where the potential is centered); d determines the spread of the potential; A is an asymmetry parameter; g compresses the two modes of the long term (stationary) density of process $\{X_t\}$; parameters p_1 and p_2 control the shape of the two modes (if the parameters p_1 and p_2 are set to low values the long term probability distribution becomes more flat with less distinct modes); σ governs the noise in the latent $\{X_t\}$ process. The error-model Y_t is an Ornstein-Uhlenbeck process specified by two parameters: κ is the autocorrelation level, and γ is the noise intensity. In principle, inference should be conducted for $[\log\kappa, \log\gamma, \log A, \log c, \log d, \log g, \log p_1, \log p_2, \log\sigma]$. However, the model parameters A and g are "stiff", i.e. small changes in their values result in considerable changes in the output, and are therefore hard to estimate. Estimating all the parameters of the DWP-SDE model is also a complex task since a larger data set seems needed to capture the stationary distribution of the data. We will therefore consider the easier task of estimating the parameters $\theta = [\log\kappa, \log\gamma, \log c, \log d, \log p_1, \log p_2, \log\sigma]$. The remaining parameters, A and g, will be fixed to arbitrary values, as discussed later.

Simulating the y_t process in (9) is easy since the transition density for the Ornstein-Uhlenbeck process process is known. We have that

$$y_{t+\Delta_t}|y_t = x \sim \mathcal{N}(xe^{-\kappa\Delta_t}, \gamma^2(1-e^{-2\kappa\Delta_t})),$$

where $\Delta_t > 0$. The transition density for the x_t process is not analytically known, and we use the Euler-Maruyama scheme to propagate the x_t process, that is we use

$$x_{t+\delta_t}|x_t = x \approx x - \nabla V(x)\delta_t + \sigma\epsilon_t,$$

where $\epsilon_t \sim \mathcal{N}(0, \delta_t^2)$, and $\delta_t > 0$ is the stepsize for the Euler-Maruyama numerical integration scheme (typically $\delta_t \ll \Delta_t$).

Let us now consider the likelihood function for the z_t process in (9), for a set of discrete observations $z = [z_1, \ldots, z_T]$ that we assume observed at integer sampling times $t \in [1, 2, ..., T]$. Corresponding (unobservable) values for the X_t process at the same sampling times are $[x_1, \ldots, x_T]$. In addition, we denote with x the set $x = [x_0, x_1, ..., x_T]$, which includes an arbitrary value x_0 from which simulations of the latent system are started. The likelihood function can be written as

$$\begin{aligned} L(\theta) = p(z|\theta) &= p(z_1|\theta)\prod_{t=2}^{T} p(z_t|z_1, \ldots, z_{t-1}, \theta), \\ &= \int p(z_1, \ldots, z_T|x_0, \ldots x_T, \theta)p(x_0, \ldots, x_T|\theta)dx_0 \cdots x_T, \\ &= \int p(z_1, \ldots, z_T|x_0, \ldots x_T, \theta)p(x_0)\prod_{t=1}^{T} p(x_t|x_{t-1}, \theta)dx_0 \cdots x_T. \end{aligned}$$

The last product in the integrand is due to the Markov property of X_t. Also, we have introduced a density $p(x_0)$, and if x_0 is deterministically fixed (as in our experiments) this density can be discarded. We cannot compute the likelihood function analytically (as the integral is typically intractable), but we can use sequential Monte Carlo (for example, the bootstrap filter in supplementary material) to compute an unbiased approximation $\hat{p}(z|\theta)$, which allows us to use PMCMC or MCWM for the inference.

Furthermore, the Z_t process is a transformation of the measurement noise that follows an Ornstein-Uhlenbeck process, and the density for $p(z_1,\dots,z_T|x_0,\dots x_T,\theta)$ is known [Picchini and Forman, 2016]. We have that

$$p(z_1,\dots,z_T|x_0,\dots x_T,\theta) = \frac{1}{\gamma}\cdot\phi\Big(\frac{z_1-x_1}{\gamma}\Big)\cdot\prod_{t=2}^{T}\frac{1}{\gamma\sqrt{1-e^{-2\kappa\Delta_t}}}\cdot\phi\Big(\frac{z_t-x_t-e^{-\kappa\Delta_t}(z_{t-1}-x_{t-1})}{\gamma\sqrt{1-e^{-2\kappa\Delta_t}}}\Big),$$

where $\Delta_t = t_i - t_{i-1}$, and $\phi(\cdot)$ denotes the density function for the standard Gaussian distribution.

We now explain how an unbiased approximation to $p(z|\theta)$ is computed. To facilitate this explanation we introduce the following notation: let $x_{t-1}^{1:N}$ denote the set of N particles we have at time $t-1$ before resampling is performed (see the bootstrap filter algorithm in the supplementary material). Let $\tilde{x}_{t-1}^{1:N}$ denote the resampled particles that are used to propagate the latent system forward to time t (using Euler-Maruyama). We approximate $p(z|\theta)$ unbiasedly with $\hat{p}(z|\theta)$ as

$$\hat{p}(z|\theta) = \hat{p}(z_1|\theta)\prod_{t=2}^{T}\hat{p}(z_t|z_1,\dots,z_{t-1},\theta) = \hat{p}(z_1|\theta)\Big\{\prod_{t=2}^{T}\frac{1}{N}\sum_{n=1}^{N}w_t^n\Big\},$$

where the weights w_t^n are

$$w_t^n = \frac{1}{\gamma\sqrt{1-e^{-2\kappa\Delta_t}}}\cdot\phi\Big(\frac{z_t-x_t^n-e^{-\kappa\Delta_t}(z_{t-1}-\tilde{x}_{t-1}^n)}{\gamma\sqrt{1-e^{-2\kappa\Delta_t}}}\Big), \qquad t\geq 2$$

and

$$\hat{p}(z_1|\theta) = \frac{1}{N}\sum_{n=1}^{N}w_1^n, \text{ with } w_1^n = \frac{1}{\gamma}\cdot\phi\Big(\frac{z_1-\tilde{x}_1^n}{\gamma}\Big).$$

5.2.1 Inference for protein folding data

We now consider the data in Figure 2. We fixed A and g to $A=-0.0025$ and $g=0$ as these parameters are difficult to identify, as already mentioned. Ideally, we should estimate A and g, however, the data that we have access to seem to be not informative enough to infer all parameters simultaneously. We set Gaussian priors as follows (notice these are not really motivated by biophysical considerations, we just set priors to be weakly informative): $p(\log\kappa)\sim\mathcal{N}(-0.7,0.8^2)$, $p(\log\gamma)\sim\mathcal{N}(-0.7,0.8^2)$, $p(\log c)\sim\mathcal{N}(3.34,0.173^2)$, $p(\log d)\sim\mathcal{N}(2.3,0.4^2)$, $p(\log p_1)\sim\mathcal{N}(0,0.5^2)$, $p(\log p_2)\sim\mathcal{N}(0,0.5^2)$, and $p(\log\sigma)\sim\mathcal{N}(0.69,0.5^2)$. The starting parameter values were set to $\exp(\theta_0)=[0.5,2,20,15,1.5,1.5,2.5]$.

We use MCWM, DA-GP-MCMC, and ADA-GP-MCMC to estimate the unknown parameters. For each iteration of MCWM we compute 4 unbiased approximations of the likelihood function, one for each core of our computer, using $N=250$ particles for each of the 4 likelihoods. Taking the sample average of these likelihoods produces another unbiased estimate of the likelihood, but with a smaller variance than the individual ones (this is obviously true and also studied in detail in Drovandi, 2014). However, given the length of the time-series, the obtained approximated likelihood is still fairly variable, and should we use PMCMC this would produce sticky chains. Therefore MCWM comes to our help for this example, as "refreshing" the denominator of the acceptance ratio helps escaping from sticky points, occurring when the likelihood approximation is overestimated.

We used the following settings with MCWM: 20,000 iterations in total and a burnin of 10,000 iterations. The proposal distribution used the generalized AM algorithm, set to target an acceptance rate of 15%. The training part for DA-GP-MCMC and ADA-GP-MCMC was the output of an MCWM algorithm with the settings specified above. We fit a GP model to the output from the first 5,000 iterations of MCWM obtained after burnin. In a similar manner as for the Ricker model the two transition kernels g and $\tilde{g}$ were based on the covariance matrices returned by the final iteration of the MCWM algorithm. A decision tree model, similar to the one used for the Ricker model, was used for the selection problem. Then we ran DA-GP-MCMC and ADA-GP-MCMC for 10,000 iterations, using

$\beta_{MH} = 0.15$. Same as with the Ricker model, a MCWM-style updating scheme was used in the second stage of both DA and ADA algorithms.

Marginal posteriors are in Figure 4, and inference results are in Table 4 and, same as for the Ricker model, we conclude that all three algorithms generate similar posterior inference. Algorithmic properties are in Table 5, and we conclude that in this case we obtain a higher speed-up compared to the Ricker model. Results are commented in detail in section 5.3. The estimated probabilities for the selection of the four different cases are in Table 6, and we observe that case 4 is the least likely case. Similarly as for the Ricker model, and due to the same reasons, the performance of the selection algorithm is much better for case 2 than for case 4.

To further illustrate inference results, we randomly pick posterior draws from the high-density region of the posterior distribution, and conditionally to these we run forward simulations using the model in (9). In Figure 3 we show three such forward simulations obtained from parameters sampled via MCMW and ADA-GP-MCMC. These look similar, which is not surprising since the posterior distribution that we obtain for the two methods also are similar. The number of regime switches appears underestimated compared to data. The forward simulations also show that we over-estimate the probability mass in the folded regime. This is likely due to not having estimated A and g from data. The values set for these two parameters are likely suboptimal, and (conditionally to those) the resulting inference for the remaining parameters is probably biased. We believe we require a longer dataset to be able to fit correctly all parameters, including A and g.

Table 4: DWP-SDE model: Posterior means (2.5th and 97.5th quantiles) for MCWM, DA-GP-MCMC, and ADA-GP-MCMC.

	MCWM	DA-GP-MCMC	ADA-GP-MCMC
$\log\kappa$	0.73 [0.42,1.19]	0.74 [0.45,1.12]	0.76 [0.42,1.29]
$\log\gamma$	0.53 [0.45,0.59]	0.52 [0.44,0.6]	0.52 [0.44,0.59]
$\log c$	3.09 [3.08,3.11]	3.1 [3.08,3.1]	3.1 [3.08,3.11]
$\log d$	3.36 [2.94,3.84]	3.32 [2.89,3.89]	3.32 [2.91,3.81]
$\log p_1$	0.46 [0.35,0.57]	0.45 [0.34,0.58]	0.45 [0.34,0.56]
$\log p_2$	-0.08 [-0.26, 0.09]	-0.07 [-0.26,0.08]	-0.08 [-0.25,0.07]
$\log\sigma$	0.68 [0.56,0.8]	0.68 [0.57,0.78]	0.69 [0.57,0.82]

Table 5: DWP-SDE model: Efficiency of MCWM, DA-GP-MCMC, and ADA-GP-MCMC. Timings for (A)DA-GP-MCMC do not include the training data collection and the fitting of the GP model.

	Minutes per 1000 iter.	Acceptance rate (%)	min ESS/min	Second stage direct (%)	Early-rejections (%)
MCWM	75.88	18.5	0.39	NA	NA
DA-GP-MCMC	24.81	3.96	0.69	15.27	68.95
ADA-GP-MCMC	15.37	3.34	0.94	14.52	69.21

Table 6: DWP-SDE model: Estimated probabilities for the different cases and percentage of times the assumption for the different cases in the ADA-GP-MCMC algorithm holds.

	Case 1	Case 2	Case 3	Case 4
Est. prob.	0.22	0.91	0.78	0.09
Perc. assum. holds (%)	43.14	87.67	65.92	25.38

5.3 Analysis of ADA-GP-MCMC

In the following we simplify the notation and refer to ADA-GP-MCMC and DA-GP-MCMC as ADA and DA. To analyze the runtime speed-up produced by ADA we execute multiple runs of both DA and

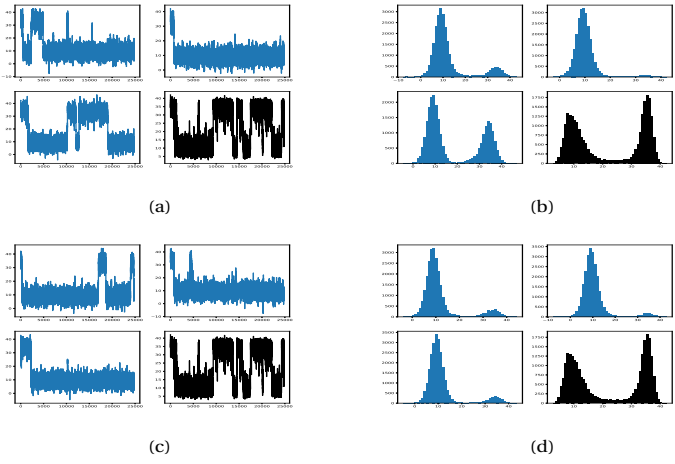
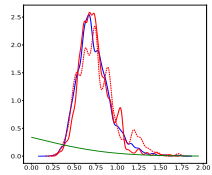


Figure 3: Trajectories in blue are forward simulated from the DWP-SDE using draws from MCWM (a) and ADA-GP-MCMC (c). Black trajectories are data. Corresponding marginal distributions from MCWM (b) and ADA-GP-MCMC (d).

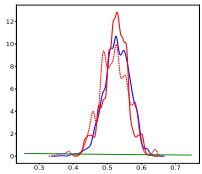
ADA. We focus on four metrics measured over 1,000 MCMC iterations: runtimes for DA and ADA; the speed-up attained by ADA, expressed as how much faster ADA is in comparison with DA; the number of particle filter evaluations in the second stage for DA and ADA (notice, in the DA case this corresponds exactly to the number of times the second stage is reached); the reduction in the number of particle filter evaluations for ADA compared to DA. Since we are interested in analyzing the speed-up potential of ADA and not necessarily the inference results we set $\beta_{MH} = 0$, hence, we never skip the ADA/DA part of the algorithms.

Furthermore, we run our analyses independently on 100 simulated datasets (see the supplementary material) using 1200 particles equally distributed across 4 cores. Results are in Figure 5. We conclude that ADA is about 2 to 4 times as fast as DA. The number of particle filter evaluations for ADA is reduced by a factor of about 3.

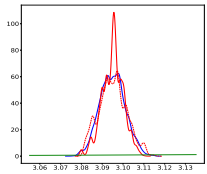
Regarding ADA, it is interesting to study how often each of the four possible cases illustrated in Section 2.1 are selected, and how likely it is that we run a particle filter conditionally on the selected case. Table 7 reports our findings for the Ricker model and DWP-SDE. We notice that proposals are not equally likely to be sent to each of the four cases, and that case 4 is the least likely case for a proposal to be sent to. It is perhaps surprising to observe the marked difference in the percentages of proposals sent to case 3 and case 4, as both cases correspond to likelihood ratios (ratio of GP likelihoods and ratio of particle filter likelihoods) that disagree in sign at the evaluated proposal. Furthermore, we can also conclude that the probability of running the particle filter varies for the different cases. Not surprisingly, given how the cases are defined, the probability for case 2 is 1 and is 0 for case 4. We also note that the probability of running the particle filter in case 3 is much lower compared to case 1: this means that whenever case 1 is selected for proposal $\theta^\star$ it turns that the event $u < \tilde{L}(\theta^{r-1}) / \tilde{L}(\theta^\star)$ is less likely than the event having the opposite inequality. If instead case 3 is selected, event $u > \frac{\tilde{L}(\theta^{r-1})}{\tilde{L}(\theta^\star)}$ is less likely than the event having the opposite inequality.



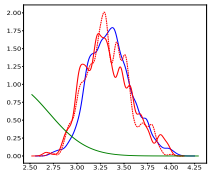
(a) $\log\kappa$.



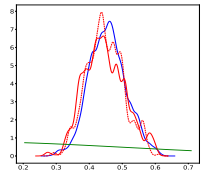
(b) $\log\gamma$.



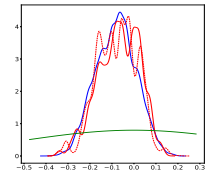
(c) $\log c$.



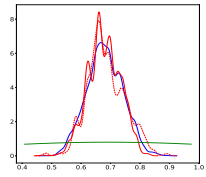
(d) $\log d$.



(e) $\log p_1$.



(f) $\log p_2$.



(g) $\log\sigma$.

Figure 4: Marginal posteriors for the DWP-SDE model: MCWM (blue solid line), DA-GP-MCMC (red solid line), and ADA-GP-MCMC (red dashed line). Priors are denoted with green lines (these look "cut" as we zoom on the bulk of the posterior).

Table 7: Percentage of proposals sent to the different cases (mean over 100 iterations of the ADA-MCMC algorithm), and probability of running the particle filter given the specific selected case (mean over 100 iterations of the ADA-MCMC algorithm).

	Percentage of proposals in each case (%)				Prob. of running particle filter in each case			
	Case 1	Case 2	Case 3	Case 4	Case 1	Case 2	Case 3	Case 4
Ricker model	62.59	12.61	21.31	3.59	0.82	1	0.40	0
DWP-SDE protein folding data.	18.80	6.82	73.28	1.09	0.98	1	0.024	0

6 Summary

We have provided ways to speed up MCMC sampling by introducing a novel, approximate version of the so-called "delayed-acceptance" MCMC introduced in Christen and Fox [2005]. More specifically, our ADA-MCMC algorithm can be used to accelerate MCMC sampling for Bayesian inference by exploiting possibilities to avoid the evaluation of a computationally expensive likelihood function.

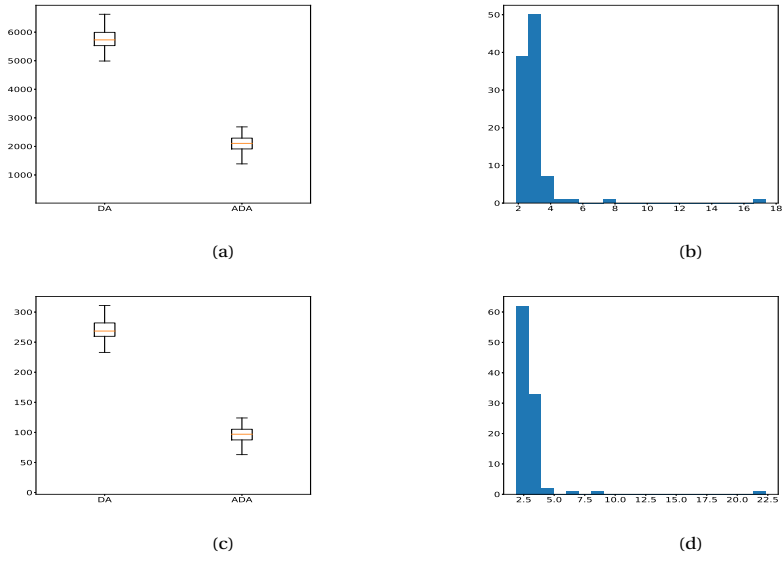


Figure 5: Speed-up analysis for the DWP model across 100 independent simulations, each for 1000 iterations, using the simulated data set. Subfigures: a) Run-times (sec) for DA (left boxplot) and ADA (right boxplot); b) Speed-up of ADA relative to DA; c) Number of particle filter evaluations in the second stage of DA (left) and ADA (right); d) Reduction in number of particle filter evaluations in the second stage for ADA compared to DA.

While the standard DA-MCMC only accepts proposals by evaluating the likelihood function associated to the exact posterior, instead ADA-MCMC in some specific cases can accept proposals even without the evaluation of the likelihood. Clearly, this is particularly relevant in statistical experiments where the likelihood function is not analytically available and is expensive to approximate. This is typical when unbiased approximations of the likelihood are used in pseudo-marginal algorithms for exact Bayesian inference [Andrieu and Roberts, 2009]. Another situation where ADA-MCMC comes useful is when the likelihood function turns expensive due to the size of the data.

Both DA-MCMC and ADA-MCMC depend on the construction of surrogates of the likelihood function. Unfortunately, producing a useful (i.e. informative) surrogate of the likelihood has its own cost. In fact, the construction of the surrogate model is typically the result of a "learning" procedure, where the output of a preliminary MCMC run (obtained using the expensive likelihood) is used to understand the relationship between simulated parameters and simulated data (e.g. using neuronal-networks as in Papamakarios et al., 2018), or between simulated log-likelihoods and parameter proposals (as in Drovandi et al., 2018).

ADA-MCMC samples from an approximate posterior distribution, while the original DA-MCMC algorithm is an exact algorithm. However, our case studies suggest that the approximative posterior inference returned by ADA-MCMC is close to the one obtained with DA-MCMC and Markov-chain-within-Metropolis (MCWM). This result is possibly connected with the quality of the surrogate model. If a poor surrogate model was used, the inference obtained using ADA-MCMC could be biased compared to DA-MCMC. The reason for this is that, in some cases, ADA-MCMC allows us to accept a proposal merely based on the surrogate model.

ADA-MCMC only generates an acceleration in the computations if the evaluation of the likelihood is time-consuming. If this evaluation is relatively fast, ADA-MCMC does not bring any significant gain compared to DA-MCMC (and in this case, any delayed-acceptance procedure should not be considered in the first place). An example of the latter case is shown with the Ricker model case study. However, for the DWP-SDE model, each likelihood evaluation using a particle filter requires about 2-10 seconds, depending on how many particles we use, and the benefits of using our novel approach are clear. Also, for this specific application, the expensive particle filter is invoked 2 to 5 times less often for ADA-MCMC than for DA-MCMC.

ADA-MCMC is not limited to the Bayesian setting and can be used to sample from a generic distribution, as mentioned in Section 2.1. Furthermore, when considering the inference problem in a Bayesian setting, ADA-MCMC can straightforwardly be paired with some other surrogate model than the Gaussian process regression model we employ. Hence, ADA-MCMC is a general algorithm for Monte Carlo sampling that can be exploited in multiple ways, other than the ones we have illustrated.

SUPPLEMENTARY MATERIAL

Further methodological tools: Details on PMCMC, MCWM, the bootstrap filter, GP regression, diagnostics, further simulation studies and setup for the implementations. (PDF file).

Julia code: the Julia code used to run the experiments is available at: https://github.com/SamuelWiqvist/adamcmcpaper.

References

C. Andrieu and G. O. Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37:697–725, 2009.

C. Andrieu and J. Thoms. A tutorial on adaptive MCMC. *Statistics and Computing*, 18(4):343–373, 2008.

E. Angelino, M. J. Johnson, R. P. Adams, et al. Patterns of scalable Bayesian inference. *Foundations and Trends® in Machine Learning*, 9(2-3):119–247, 2016.

M. Banterle, C. Grazian, A. Lee, and C. P. Robert. Accelerating metropolis-hastings algorithms by delayed acceptance. *arXiv preprint arXiv:1503.00996*, 2015.

R. B. Best and G. Hummer. Diffusion models of protein folding. *Physical Chemistry Chemical Physics*, 13(38):16902–16911, 2011.

J. A. Christen and C. Fox. Markov chain Monte Carlo using an approximation. *Journal of Computational and Graphical statistics*, 14(4):795–810, 2005.

C. C. Drovandi. Pseudo-marginal algorithms with multiple CPUs. https://eprints.qut.edu.au/61505/, 2014.

C. C. Drovandi, M. T. Moores, and R. J. Boys. Accelerating pseudo-marginal MCMC using Gaussian processes. *Computational Statistics & Data Analysis*, 118:1–17, 2018.

R. G. Everitt and P. A. Rowińska. Delayed acceptance ABC-SMC. *arXiv:1708.02230*, 2017.

S.-C. Fang, D. Y. Gao, G.-X. Lin, R.-L. Sheu, and W.-X. Xing. Double well potential function and its optimization in the n-dimensional real space: part i. *Journal of Industrial and Management Optimization*, 13(3):1291–1305, 2017.

M. Fasiolo, N. Pya, and S. Wood. A comparison of inferential methods for highly nonlinear state space models in ecology and epidemiology. *Statistical Science*, 31(1):96–118, 2016.

P. Fearnhead and D. Prangle. Constructing summary statistics for approximate Bayesian computation: semi-automatic approximate Bayesian computation. *Journal of the Royal Statistical Society: Series B*, 74(3):419–474, 2012.

J. L. Forman and M. Sørensen. A transformation approach to modelling multi-modal diffusions. *Journal of Statistical Planning and Inference*, 146:56–69, 2014.

J. Franks and M. Vihola. Importance sampling and delayed acceptance via a Peskun type ordering. *arXiv:1706.09873*, 2017.

A. Golightly, D. A. Henderson, and C. Sherlock. Delayed acceptance particle MCMC for exact inference in stochastic kinetic models. *Statistics and Computing*, 25(5):1039–1055, 2015.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

N. Kantas, A. Doucet, S. S. Singh, J. Maciejowski, and N. Chopin. On particle methods for parameter estimation in state-space models. *Statistical Science*, 30(3):328–351, 2015.

J. Marin, P. Pudlo, C. Robert, and R. Ryder. Approximate Bayesian computational methods. *Statistics and Computing*, pages 1–14, 2012.

E. Meeds and M. Welling. GPS-ABC: Gaussian process surrogate approximate Bayesian computation. In *Proceeding of the 30th conference on Uncertainty in Artificial Intelligence (UAI)*, 2014.

C. L. Mueller. Exploring the common concepts of adaptive MCMC and covariance matrix adaptation schemes. In A. Auger, J. L. Shapiro, L. D. Whitley, and C. Witt, editors, *Theory of Evolutionary Algorithms*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2010.

G. Papamakarios, D. C. Sterratt, and I. Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. *arXiv preprint arXiv:1805.07226*, 2018.

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

U. Picchini and J. L. Forman. Accelerating inference for diffusions observed with measurement error and large sample sizes using approximate Bayesian computation. *Journal of Statistical Computation and Simulation*, 86(1):195–213, 2016.

M. Quiroz, M.-N. Tran, M. Villani, and R. Kohn. Speeding up MCMC by delayed acceptance and data subsampling. *Journal of Computational and Graphical Statistics*, 2017. doi: 10.1080/10618600.2017.1307117.

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT press, 2006.

J. Rosenthal and G. Roberts. Coupling and ergodicity of adaptive MCMC. *Journal of Applied Probablity*, 44:458–475, 2007.

T. B. Schön, A. Svensson, L. Murray, and F. Lindsten. Probabilistic learning of nonlinear dynamical systems using sequential Monte Carlo. *Mechanical Systems and Signal Processing*, 104:866–883, 2018.

C. Sherlock, A. Golightly, and D. A. Henderson. Adaptive, delayed-acceptance MCMC for targets with expensive likelihoods. *Journal of Computational and Graphical Statistics*, 26(2):434–444, 2017.

A. Solonen, P. Ollinaho, M. Laine, H. Haario, J. Tamminen, and H. Järvinen. Efficient MCMC for climate model parameter estimation: parallel adaptive chains and early rejection. *Bayesian Analysis*, 7(3):715–736, 2012.

S. N. Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310): 1102–1104, 2010.

Accelerating delayed-acceptance Markov chain Monte Carlo algorithms

Samuel Wiqvist*, Umberto Picchini◇*, Julie Lyng Forman†, Kresten Lindorff-Larsen‡, Wouter Boomsma⋆

*Centre for Mathematical Sciences, Lund University, Sweden
◇Department of Mathematical Sciences, Chalmers University of Technology and the University of Gothenburg, Sweden
†Dept. Public Health, section of Biostatistics, University of Copenhagen, Denmark
‡The Linderstrøm-Lang Centre for Protein Science, Department of Biology, University of Copenhagen, Denmark
⋆Department of Computer Science, University of Copenhagen, Denmark

Supplementary material

Contents

1 Technical details for the GP model

Following Drovandi et al. [2018], the unknown log-likelihood function is assumed to be quadratic in θ. A quadratic mean function m for the GP model is therefore specified as

$$m_\beta(\theta) = \beta_0 + \sum_{i=1}^{d} \beta_i \theta_i + \sum_{j \geq i=1}^{d} \beta_{ij} \theta_i \theta_j = \begin{bmatrix} 1 & \theta_1 & \theta_2 & \dots & \theta_d \theta_d \end{bmatrix}. \tag{1}$$

In (1) β is a vector of unknown regression coefficients $\beta = [\beta_1, \beta_2, \dots, \beta_{dd}]^\top$. We also assume that the log-likelihood function is fairly smooth, and we use an automatic relevance determination squared exponential covariance function (ardSE), defined as

$$k_\phi(\theta, \theta') = \sigma_k \exp(-1/2(\theta - \theta')^\top P^{-1}(\theta - \theta')) + \sigma \mathbb{1}(\theta = \theta'),$$

where P is a diagonal matrix, with diagonal entries $[l_1^2, ..., l_{dd}^2]$. The parameters of the covariance function are $\phi = [\sigma \quad \sigma_k \quad l_1 \quad \dots \quad l_{dd}]$, where σ is the "nugget", σ_k the output standard deviation, and the l_i's the length scales for each dimension. The full set of parameters for the GP model is therefore $\eta = [\phi \quad \beta]$.

We first pre-estimate β alone using linear regression, to ease the joint optimization problem described in a moment. When pre-estimating β we remove a small number of cases having very low likelihood values. These are considered as outliers and are removed in order to ease the optimization problem. Once this first estimate of β is available, the GP model is fitted to $\mathscr{D}$ using maximum likelihood, i.e. both parameters in $\eta = [\phi \quad \beta]$ are jointly estimated (a starting value for β is provided by its pre-estimated value) by minimizing the GP negative log-likelihood $g(\eta)$ with respect to η, where

$$\begin{gathered} g(\eta) = -\log p(\ell(\theta)|\eta) = \\ (\ell(\theta) - m_\beta(\theta))^\top K_\phi(\Theta,\Theta)^{-1}(\ell(\theta) - m_\beta(\theta)) + \log(\det K_\phi(\Theta,\Theta)) + c. \end{gathered} \tag{2}$$

We used $\det(A)$ to denote the determinant of the matrix A, while c is a constant not affecting the optimization. Here Θ denotes the matrix of the θ proposals that belong to the training data $\mathscr{D}$. The matrix $K_\phi(\Theta,\Theta)$ is the covariance matrix for all the proposals in the matrix Θ. The gradient for the negative log-likelihood (2) is analytically known, and we have that

$$\frac{\partial g}{\partial \beta} = -2m_\beta(\theta)^\top K_\phi(\Theta,\Theta)^{-1}(\ell(\theta) - m_\beta(\theta)\beta),$$

and

$$\begin{aligned} \frac{\partial g}{\partial \phi_i} = -(\ell(\theta) - m_\beta(\theta)\beta)^\top K_\phi(\Theta,\Theta)^{-1}\frac{\partial K_\phi(\Theta,\Theta)}{\partial \phi_i}K_\phi(\Theta,\Theta)^{-1}(\ell(\theta) - m_\beta(\theta)\beta) + \\ \operatorname{tr}(K_\phi(\Theta,\Theta)^{-1}\frac{\partial K_\phi(\Theta,\Theta)}{\partial \phi_i}), \end{aligned}$$

where $\operatorname{tr}(A)$ denotes the trace of the matrix A. We can now use a gradient-based optimization algorithm (and in practice we use the conjugate gradient algorithm) to fit the GP model to the training data $\mathscr{D}$, and we obtain $\hat{\eta} = [\hat{\phi} \quad \hat{\beta}]$ by minimizing (2).

It is simple, and computationally cheap, to generate predictions from the fitted GP model since the predictive distribution is known in closed-form [Rasmussen and Williams, 2006]. This predictive distribution is just the posterior distribution of $\ell(\theta)$ given the training data $\mathscr{D}$ and conditionally to $\hat{\eta}$. That is, for a newly proposed parameter $\theta^\star$

$$\ell(\theta^\star)|\mathscr{D},\hat{\eta} \sim \mathscr{N}(\bar{\ell}(\theta^\star), \operatorname{Var}(\ell(\theta^\star))), \tag{3}$$

where

$$\bar{\ell}(\theta^\star) = m_{\hat{\beta}}(\theta^\star) + K_{\hat{\phi}}(\theta^\star,\Theta)K_{\hat{\phi}}(\Theta,\Theta)^{-1}(\ell(\Theta) - m_{\hat{\beta}}(\Theta)), \tag{4}$$

and

$$\operatorname{Var}(\ell(\theta^\star)) = K_{\hat{\phi}}(\theta^\star,\theta^\star) - K_{\hat{\phi}}(\theta^\star,\Theta)K_{\hat{\phi}}(\Theta,\Theta)^{-1}K_{\hat{\phi}}(\Theta,\theta^\star). \tag{5}$$

Notice that the (expensive) matrix inversion $K_{\hat{\phi}}(\Theta,\Theta)^{-1}$ in (4)–(5) should only be produced once, since it does not depend on the proposed $\theta^\star$.

The predictive distributions allows for three different types of predictions:

1. *Mean prediction*: The log-likelihood function at a certain $\theta^\star$ is deterministically predicted from its mean value at $\theta^\star$, that is $\bar{\ell}(\theta^\star)$.

2. *Noisy prediction*: Predicting the log-likelihood by sampling from the predictive distribution (3) and including the "nugget" σ in $K_\phi(\theta^\star,\theta^\star)$. Hence, $K_\phi(\theta^\star,\theta^\star)$ is computed as $K_\phi(\theta^\star,\theta^\star) = \sigma_k + \sigma$.

3. *Noise-free prediction*: Sample from (3) where the "nugget" σ is not included, thereby obtaining a non-noisy prediction. The term $K_\phi(\theta^\star,\theta^\star)$ is therefore computed as $K_\phi(\theta^\star,\theta^\star)=\sigma_k$.

Same as in Drovandi et al. [2018], we are interested in modeling $\ell(\theta)$, and not a noisy estimate of it, and we will therefore use noise-free predictions. In conclusion, in our delayed-acceptance algorithms we will generate proxies to the unknown $\ell(\theta)$ by sampling from the GP predictive (3) using a noise-free approach.

2 Particle marginal methods for state-space models

The challenge of approximating the likelihood function for complex models with "intractable likelihoods" has generated a large body of literature in the past fifteen years, most notably approximate Bayesian computation (ABC, see the reviews Sisson and Fan, 2011 and Karabatsos and Leisen, 2017) and pseudo-marginal (particle) MCMC algorithms (Beaumont, 2003, Andrieu and Roberts, 2009, Andrieu et al., 2010). Pseudo-marginal algorithms in particular have found an immediate success in inference for state-space models using sequential Monte Carlo (or particle filters); reviews are Jacob [2015] and Kantas et al. [2015].

Pseudo-marginal algorithms build on the interplay between Markov chain Monte Carlo (MCMC), importance sampling and sequential Monte Carlo (SMC, or particle filters) algorithms. The crucial result is that when the likelihood $p(y|\theta)$ is not available analytically but obtaining a non-negative unbiased estimator $\hat{p}(y|\theta)$ is possible, then a Metropolis-Hastings algorithm using $\hat{p}(y|\theta)$ instead of $p(y|\theta)$ will generate a Markov chain having $p(\theta|y)$ as stationary distribution. This means that it is possible to target the exact posterior even when we deal with an (unbiased) approximation to the likelihood function, rather than the exact likelihood. Andrieu and Roberts [2009] discuss the problem by estimating unbiasedly the unavailable likelihood using N draws from an importance sampler, and the remarkable result is once more that exact Bayesian sampling from $p(\theta|y)$ is possible for any finite value of N. Andrieu et al. [2010] frame their particle MCMC (PMCMC) approach for a large class of statistical models, including state-space models (SSM, Cappé et al., 2005). For SSM an unbiased estimator $\hat{p}(y|\theta)$ is given by particle filters using N particles (here and in the following we write $\hat{p}(y|\theta)\equiv\hat{p}_N(y|\theta)$ since the resulting inference for θ is theoretically unaffected by the value of N). In Andrieu et al. [2010] the PMCMC algorithms PMMH (particle marginal Metropolis-Hastings) and PG (particle Gibbs) target the posterior $p(\theta,x_{1:T}|y_{1:T})$ exactly, where $y_{1:T}$ is the sequence of measurements from process $\{y_t\}$ in (6) collected at T discrete times which, to simplify the notation, we assume to be the integers $\{1,2,...,T\}$. With $x_{1:T}$ we denote the corresponding latent (unobservable) dynamics, see (6). We employ the following notation for sequences of variables $z_{1:T}\equiv\{z_1,...,z_T\}$. Therefore PMMH and PG solve simultaneously the parameter inference and the state filtering problem. In the next sections we clarify how these pseudo-marginal methods (PMM) and the delayed-acceptance (DA) framework interact, while emphasizing once more that in order to run a DA algorithm, including our accelerated DA method, the PMM framework is not necessary, nor is our methodology specific for dynamic models such as SSM but can be applied also to "static" models.

A SSM can be written as

$$\begin{cases} y_t \sim p(y_t|x_t;\theta_y) \\ x_t \sim p(x_t|x_s,\theta_x), \quad x_0\sim p(x_0), \quad t_0\le s<t, \end{cases} \tag{6}$$

where $x_0\equiv x_{t_0}$ is a random initial state with initial distribution $p(x_0)$, observations $y_t\in\mathbb{R}^{d_y}$ depend on a finite dimensional unknown parameter θ_y, and observations are conditionally independent given the latent state $\{x_t\}_{t\ge t_0}$, with $x_t\in\mathbb{R}^{d_x}$, and $d_x,d_y\ge 1$. Here $\{x_t\}$ is a continuous Markov process equipped with a transition density $p(x_t|x_s,\cdot)$ for $s<t$ and depending on another finite dimensional unknown parameter θ_x. Therefore we have that $\theta=(\theta_x,\theta_y)$ is the parameter object of our inference. In this work we consider posterior inference for θ, hence our ideal target is $p(\theta|y_{1:T})$, however, instead of calling the algorithms "pseudo-marginal", we call them PMCMC, since we use particle filters

to approximate the likelihood function. But recall that we are not interested in the filtering problem for $x_{1:T}$.

Despite the existence of these powerful and flexible algorithms, computing an (unbiased) estimator of the likelihood function can be computationally time-consuming for complex models. Computationally cheap surrogate models have therefore been used to accelerate instances of the PMCMC algorithm. As an example, in Drovandi et al. [2018] a surrogate model based on Gaussian processes (GP) is used to replace the time-consuming sequential Monte Carlo estimation of the likelihood function. After an initial, computationally expensive "training phase", a GP regression model is fitted to the output of the training phase (consisting of proposed parameter values and log-likelihoods estimated via particle filters), and the estimated GP is then used as a (cheap) surrogate of the log-likelihood function, allowing for considerable computational acceleration in the MCMC sampling.

Another approach is to not entirely replace the sequential Monte Carlo estimation of the likelihood function, but only compute these estimations for parameter proposals that are not "early-rejected" by the surrogate model. This is a delayed-acceptance (DA) approach, used for example in Golightly et al. [2015] and Sherlock et al. [2017]. As already mentioned, DA-MCMC has two important properties: the ergodicity of the chain is preserved, and the resulting Markov chain targets the true posterior distribution of θ. In Golightly et al. [2015] the surrogate model is based on Langevin diffusion approximations and linear noise approximations. In Sherlock et al. [2017] the surrogate estimation of the likelihood function is computed using previous estimations via a search-tree. Hence, quite different surrogate models can be employed and still resulting in an valid DA-MCMC for exact Bayesian inference.

2.1 Particle Markov chain Monte Carlo

The likelihood function for the SSM (6) can be written as

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

where

$$p(y_t|y_{1:t-1};\theta) = \int p(y_t|x_t;\theta)p(x_t|y_{1:t-1};\theta)dx_t$$

and the latter integral can be efficiently approximated by drawing N "particles" $x_t^n \sim p(x_t|y_{1:t-1};\cdot)$ then taking the sample average $\sum_{n=1}^{N} p(y_t|x_t^n;\cdot)/N$, and similarly to approximate $p(y_1|\cdot)$. This can be accomplished using sequential Monte Carlo methods, such as the bootstrap particle filter [Gordon et al., 1993] given in Algorithm 1. The bootstrap filter returns a non-negative unbiased estimator of the likelihood function $\hat{L}_{PF} \equiv \hat{p}(y_{1:T}|\theta)$, where the expectation of $\hat{L}_{PF}$ is taken with respect to the law underlying the generation of the random variates necessary for the implementation of Algorithm 1. For successful implementations, the number of particles N should be tuned so that the standard deviation of the estimated log-likelihood $\log \hat{L}_{PF}$ does not exceed the value 2 at any given θ, to assure good performance of the PMCMC [Pitt et al., 2012], and avoid problems of sticky chains [Sherlock et al., 2015].

The particle Markov chain Monte Carlo algorithm (PMCMC) in Algorithm 2 uses $\hat{L}_{PF}$ in an otherwise standard Metropolis-Hastings algorithm, to sample from the parameter posterior $p(\theta|y_{1:T})$ exactly, for any value of N (Beaumont, 2003, Andrieu and Roberts, 2009), even though N does have an impact on the mixing properties of the algorithm, as discussed below. An algorithm closely related to PMCMC is Monte Carlo within Metropolis (MCWM), given in Algorithm 3 and due to Beaumont [2003] (but see Medina-Aguayo et al., 2016 for theoretical properties). The only difference between MCWM and PMCMC is that in MCWM the likelihood value at the denominator of the acceptance probability is re-estimated anew as $\hat{L}_{PF}(\theta^{r-1})$. That is, at each iteration of MCWM the estimated likelihood at the denominator of α in step 5 of Algorithm 3 is "refreshed". Notice in particular the double estimations of the likelihood in steps 3–4. Hence, each iteration of the MCWM algorithm requires two

Algorithm 1 Bootstrap particle filter

Input: Data $y_{1:T}$, number of particles N, and model parameters θ.
Output: The likelihood estimation $\hat{L}_{PF}(\theta)$.

Initialize particles $\tilde{x}_0^n \sim p(x_0)$.
for $t = 1, \ldots, T$ **do**
  if $t = 1$ **then**
    For $n = 1, \ldots, N$, propagate particles, $x_1^n \sim p(\cdot|\tilde{x}_0^n)$.
    For $n = 1, \ldots, N$, evaluate importance weights, $w_1^n = p(y_1|x_1^n)$.
    Estimate $\hat{p}(y_1|\theta) = \frac{\sum_{n=1}^N w_1^n}{N}$.
    For $n = 1, \ldots, N$, normalize importance weights, $\bar{w}_1^n = \frac{w_1^n}{\sum_{n=1}^N w_1^n}$.
  else
    Re-sample N times with replacement from $(x_{t-1}^1, ..., x_{t-1}^N)$ with associated probabilities $(\bar{w}_{t-1}^1, ..., \bar{w}_{t-1}^N)$ to obtain a new sample $(\tilde{x}_{t-1}^1, ..., \tilde{x}_{t-1}^N)$.
    For $n = 1, \ldots, N$, propagate particles, $x_t^n \sim p(\cdot|\tilde{x}_{t-1}^n)$.
    For $n = 1, \ldots, N$, evaluate importance weights, $w_t^n = p(y_t|x_t^n)$.
    Estimate $\hat{p}(y_t|y_{1:t-1};\theta) = \frac{\sum_{n=1}^N w_t^n}{N}$.
    For $n = 1, \ldots, N$, normalize importance weights, $\bar{w}_t^n = \frac{w_t^n}{\sum_{n=1}^N w_t^n}$.
  end if
end for
Estimated likelihood $\hat{L}_{PF} := \hat{p}(y|\theta) = \hat{p}(y_1|\theta) \prod_{t=2}^T \hat{p}(y_t|y_{1:t-1};\theta)$.

estimations of the likelihood function, which is a drawback if the estimation is computationally intensive. The mathematical properties of the MCWM algorithm are less well understood than for PMCMC. The main advantage is, however, that MCWM in many cases generates a chain that mixes better than PMCMC, even when the estimation of the likelihood function is imprecise [Medina-Aguayo et al., 2016]. With MCWM one often avoids problems of stickiness in the simulated Markov chain, a problem that the PMCMC algorithm can suffer from, in particular if the number of particles used in the particle filter is low [Sherlock et al., 2015]. In fact, this causes the estimated likelihoods to have high variability, allowing for the acceptance of the occasional over-estimated $\hat{p}(y_{1:T}|\theta)$ to end-up at the denominator of α in Algorithm 2, hence reducing the chance for newer proposals to be accepted. By "refreshing" the denominator at each iteration, MCWM alleviates this pathology. However, while PMCMC targets the true posterior $p(\theta|y_{1:T})$, this does not hold for MCWM. However, Medina-Aguayo et al. [2016] gives mild conditions on the particle weights such that the stationary distribution targeted by MCWM algorithm will converge to the true posterior distribution as $N \to \infty$. Simulation results show that, for finite N, the marginal posteriors obtained from MCWM are often wider than the true marginals implied by the PMCMC algorithm, and MCWM therefore generates a conservative estimation of the posterior distribution [Drovandi et al., 2018].

3 Implementation details

Unless else stated, all calculations were carried out on the LUNARC cluster available at Lund University (Sweden), where each node has access to two Intel Xeon E5-2650 v3 (2.3 Ghz, 10-core) CPUs, http://www.lunarc.lu.se. The algorithms are implemented with Julia 0.5.2 [Bezanson et al., 2017], and the code is available at https://github.com/SamuelWiqvist/adamcmcpaper.

For the considered case studies, the parameters in θ are all positive, and for convenience we conduct inference on their natural logarithms. The prior distributions will also be set on the log-scale. The weights w_t^n in the particle filter can sometimes take very large and small values, and for numerical stability these are computed on the log-scale. We also make use of standard methods such as subtracting the largest log-weight at time t from the log-weights at time t, prior to exponentiate them [Cappé et al., 2007]. Regarding the computation of the sum of the weights, required to compute the denominator of the normalized weights $\bar{w}_t^n$, the so-called log-sum-exp trick turns useful [Murphy,

Algorithm 2 PMCMC algorithm

Input: Number of iterations R, starting parameters θ^0, and corresponding $\hat{L}_{PF}(\theta^0)$.
Output: The chain $\theta^{1:R}$.

for $r = 1, \ldots, R$ **do**
  Propose $\theta^\star \sim g(\cdot|\theta^{r-1})$.
  Run Algorithm 1 to estimate $\hat{L}_{PF}(\theta^\star)$.
  Compute $\alpha = \min(1, \frac{\hat{L}_{PF}(\theta^\star)}{\hat{L}_{PF}^{r-1}(\theta^{r-1})} \cdot \frac{p(\theta^\star)}{p(\theta^{r-1})} \cdot \frac{g(\theta^{r-1}|\theta^\star)}{g(\theta^\star|\theta^{r-1})})$.
  Draw $u \sim \mathcal{U}(0,1)$.
  if $u \leq \alpha$ **then**
    Set $\theta^r = \theta^\star$.
  else
    Set $\theta^r = \theta^{r-1}$.
  end if
end for

Algorithm 3 MCWM algorithm

Input: Number of iterations R, starting parameters θ^0.
Output: The chain $\theta^{1:R}$.

for $r = 1, \ldots, R$ **do**
  Propose $\theta^\star \sim g(\cdot|\theta^{r-1})$.
  Run Algorithm 1 to estimate $\hat{L}_{PF}(\theta^\star)$.
  Run Algorithm 1 to estimate $\hat{L}_{PF}(\theta^{r-1})$.
  Compute $\alpha = \min(1, \frac{\hat{L}_{PF}(\theta^\star)}{\hat{L}_{PF}(\theta^{r-1})} \cdot \frac{p(\theta^\star)}{p(\theta^{r-1})} \cdot \frac{g(\theta^{r-1}|\theta^\star)}{g(\theta^\star|\theta^{r-1})})$.
  Draw $u \sim \mathcal{U}(0,1)$.
  if $u \leq \alpha$ **then**
    Set $\theta^r = \theta^\star$.
  else
    Set $\theta^r = \theta^{r-1}$.
  end if
end for

2012]. In Algorithm 1 particles are resampled using the stratified resampling algorithm [Kitagawa, 1996]. The execution of the bootstrap filter for the Ricker model is relatively cheap, since the model is fairly simple and the data set used is small (it only contains $T = 50$ observations). We can, therefore, easily compute exact Bayesian inference by using the PMCMC algorithm, since it is possible to run the particle filter with sufficiently many particles, so that the standard deviation of the estimated log-likelihood is less than 2.

The DWP-SDE model is a more complex case study, and the particle filter is time-consuming since the data set contains 25,000 observations. On a standard desktop computer it can therefore be computational unfeasible to run the PMCMC algorithm. We assign $N \approx 200 - 1200$ particles to separate cores of the LUNARC cluster (possibly over multiple nodes), and run independent particle filters in parallel (this can also be replicated on a multiprocessor desktop by running several independent estimations of the likelihood). A simple method, exploiting multiple particle filters running in parallel on multiple cores (or multiple CPUs), is in Drovandi [2014], and consists of averaging out likelihood approximations obtained at different cores. Since the likelihood approximations are computed on the log-scale we have to compute the average of the exponential of the log-likelihood approximation, and then take the logarithm of this average. This scheme allows us to obtain an unbiased approximation of the likelihood function with lower variance, compared to the approximation obtained from a single particle filter. The negative log-likelihood function g in (2) is minimized using the function `optimize`, found in the Julia package `Optim.jl`. In particular, we used a conjugate-gradient algorithm. As a measure of efficiency of the different Markov chains produced by the different algorithms, we compute the minimal ESS/(time unit), where ESS is the effective sample size. That is, the ESS for each parameter's chain is obtained via the R-package `mcmcse`, then the minimum ESS value across

all chains is found, and this value is then divided by the run-time. Hence, min ESS/(time unit) tells us how many independent samples the algorithm is generating per time-unit, when we consider the least efficient chain.

4 Diagnostics for the GP model and selection methods

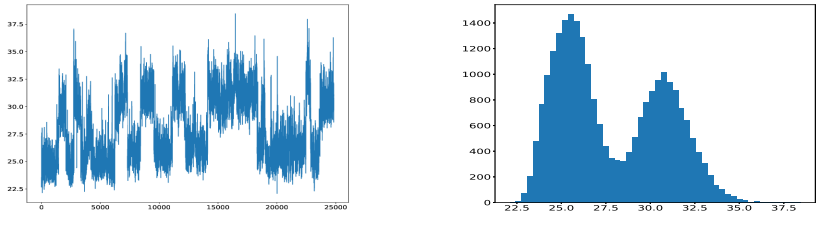
For diagnostic purposes of the predictive accuracy of the fitted models (GP and selection methods $s_{13}()$ and $s_{24}()$), we can split the *training* data, to obtain *testing* data. Basically, what we have denoted as $\mathscr{D}$ and $\tilde{\mathscr{D}}$, can be partitioned as $\mathscr{D} = [\mathscr{D}_1, \mathscr{D}_2]$ and $\tilde{\mathscr{D}} = [\tilde{\mathscr{D}}_1, \tilde{\mathscr{D}}_2]$. Then $\mathscr{D}_1$ (and $\tilde{\mathscr{D}}_1$) can be used to fit the GP model, while $\mathscr{D}_2$ (and $\tilde{\mathscr{D}}_2$) is the "test data", which is not used to fit the GP model, nor to fit the selection methods. Instead the test data is merely used used to evaluate the performance of the GP model and the selection methods, as typically done with predictive models. In this case by considering data that is not used to fit the GP model.

To test the fit of the GP model, we predict likelihood values from the GP for each proposal in the test data in $\mathscr{D}_2$, and compare the GP predictions to the corresponding particle filter predictions that are stored in $\mathscr{D}_2$.

Testing the performance of the selection methods is a slightly more involved process. For each proposal in the test data $\mathscr{D}_2$ we compute corresponding GP predictions, and we also compute a new set of particle filter predictions. We then use the GP predictions and check if proposal r belongs to case 1 and 3, or case 2 and 4. Assume that proposal $\theta^{\star,r}$ belongs to case 1 and 3. Then run the selection method $s_{1,3}(\theta^{\star,r})$ for proposal r, and check which case proposal r belongs to. After having determined which case proposal r belongs to, according to the selection method, we check if the same case is selected using the new particle filter predictions, where we use the definition of the four cases (see Section 2.1 in the paper) to determine which case we should select, according to the new particle filter predictions. Using this method we can calculate how likely it is that the new particle filter predictions and the selection method are consistent.

5 DWP-SDE model: Simulation study

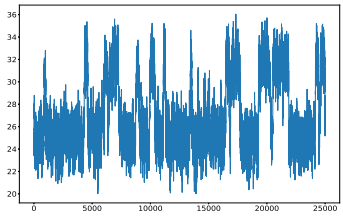
Here we simulate data from model DWP-SDE model, and then produce Bayesian inference for the parameters. Simulated data of length $T = 25,000$ are produced using ground-truth parameters θ_{true} set to $\exp(\theta_{\text{true}}) = [0.3, 0.9, 0.01, 28.5, 4, 0.03, 1.5, 1.8, 1.9]$. Similarly to the paper, we consider parameters A and g as known and fixed to $A = 0.01$ and $g = 0.03$. The other parameters are treated as unknown. The simulated data are reported in Figure 2. The parameters were set to produce data resembling data set 1 in Figure 1 which is an additional protein folding dataset.



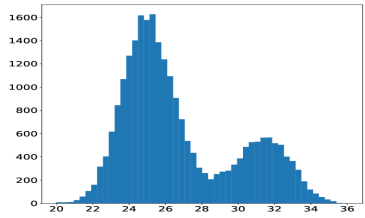
(a) Data.

(b) Marginal distribution of the data.

Figure 1: Additional protein folding dataset.



(a) Simulated data.



(b) Marginal distribution of the simulated data.

Figure 2: Data generated from the DWP-SDE model.

Table 1: Posterior means (2.5th and 97.5th quantiles) for MCWM, DA-GP-MCMC, and ADA-GP-MCMC.

	True value	MCWM	DA-GP-MCMC	ADA-GP-MCMC
$\log\kappa$	-1.2	-1.2 [-1.43,-0.97]	-1.21 [-1.43,-0.99]	-1.21 [-1.44,-0.95]
$\log\gamma$	-0.11	-0.11 [-0.25,0.02]	-0.1 [-0.24,0.01]	-0.11 [-0.25,0.03]
$\log c$	3.35	3.35 [3.34,3.36]	3.35 [3.34,3.36]	3.35 [3.34,3.36]
$\log d$	1.39	1.44 [1.17,1.81]	1.41 [1.18,1.69]	1.43 [1.16,1.85]
$\log p_1$	0.41	0.43 [0.29,0.63]	0.42 [0.29,0.57]	0.43 [0.28,0.65]
$\log p_2$	0.59	0.51 [0.1, 0.82]	0.54 [0.18,0.88]	0.52 [0.02,0.92]
$\log\sigma$	0.64	0.65 [0.48,0.81]	0.64 [0.49,0.77]	0.65 [0.48,0.79]

We set Gaussian priors: $p(\log\kappa) \sim \mathcal{N}(-0.7, 0.5^2)$, $p(\log\gamma) \sim \mathcal{N}(-0.7, 0.5^2)$, $p(\log c) \sim \mathcal{N}(3.34, 0.173^2)$, $p(\log d) \sim \mathcal{N}(1.15, 0.2^2)$, $p(\log p_1) \sim \mathcal{N}(0.69, 0.5^2)$, $p(\log p_2) \sim \mathcal{N}(0, 0.5^2)$, and $p(\log\sigma) \sim \mathcal{N}(0, 0.5^2)$. The starting parameter values were set far from the ground truth, as $\exp(\theta_0) = [2, 2, 30, 10, 2, 2, 2]$. The algorithm settings for MCWM, DA-GP-MCMC, and ADA-GP-MCMC are the same as in Section 5.2.1 in the paper.

Notice, before fitting the GP model, we removed the 1% of the observations having the lowest log-likelihood from the training data, in order to obtain a more robust prediction. Marginal posteriors from the two methods are in Figure 3. These results are very similar, given the diffuse priors (also, see the posterior quantile intervals in Table 1). All parameters are well inferred and we manage to capture the true parameter values. From Table 2 we see that the speed-up for ADA-GP-MCMC is larger in this case, compared to the Ricker model, since ADA-GP-MCMC is 4.6 times faster than MCWM, and 1.5 times faster than DA-GP-MCMC. The algorithm efficiency measure min ESS/minute in Table 2 indicates that ADA-GP-MCMC is somewhat more efficient than both MCWM and DA-GP-MCMC. In Table 3 we present the estimated probabilities for the four different cases, and we can conclude that

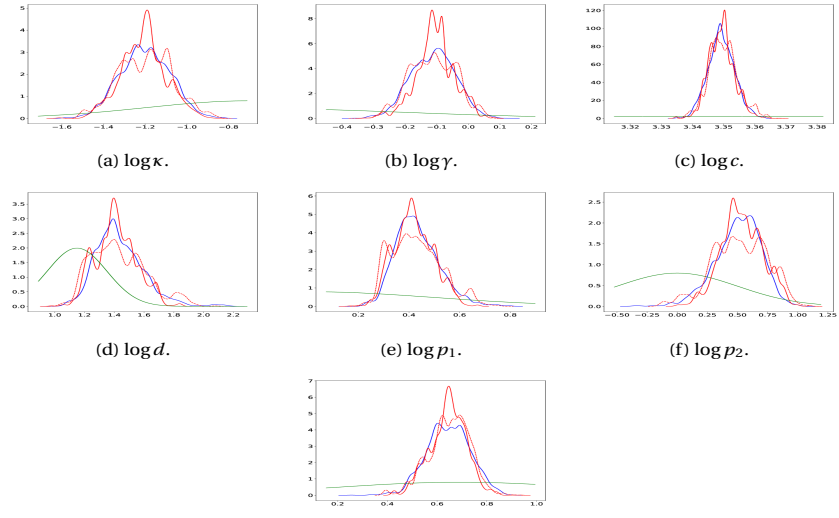
Table 2: Algorithm properties for the the MCWM, DA-GP-MCMC, and ADA-GP-MCMC algorithm.

	Minutes per 1000 iter.	Acceptance rate (%)	min ESS/min	Second stage direct (%)	Early-rejections (%)
MCWM	60.29	19.84	0.57	NA	NA
DA-GP-MCMC	20.67	3.80	0.67	15.01	67.03
ADA-GP-MCMC	13.39	4.02	1.04	15.10	67.01

Table 3: Estimated probabilities for the different cases and percentage of times the assumption for the different cases in the ADA-GP-MCMC algorithm holds.

	Case 1	Case 2	Case 3	Case 4
Est. prob. $(\hat{p}_1, \hat{p}_2, \hat{p}_3, \hat{p}_4)$	0.22	0.90	0.78	0.09
Perc. assum. holds	38.05	84.40	69.17	27.90

case 4 is the least likely case. We also notice that the performance of the selection algorithm is much better for case 2 than for case 4: this is due to the unbalance of the two classes, meaning that in our training data case 2 occurrs more frequently than case 4, and therefore it is more difficult to estimate the latter case accurately.



(a) $\log \kappa$. (b) $\log \gamma$. (c) $\log c$.

(d) $\log d$. (e) $\log p_1$. (f) $\log p_2$.

(g) $\log \sigma$.

Figure 3: Marginal posteriors based on simulated data: MCWM (blue solid line), DA-GP-MCMC (red solid line), and ADA-GP-MCMC (red dashed line). Priors are denoted with green lines (these look "cut" as we zoom on the bulk of the posterior).

We now sample parameters from the high-density region of the posterior distribution and run forward simulations of the DWP-SDE model, similarly to the main paper. In Figure 4 we present three forward simulations, conditionally to parameters from MCMW and ADA-GP-MCMC. The forward simulations in Figure 4 resemble the simulated data better than the forward simulations in the main paper resemble the real data. This seems to point to the fact that the arbitrarily chosen values for A and g in the real-data case study are suboptimal, and (conditionally to those) the inference for the other parameters is probably biased.

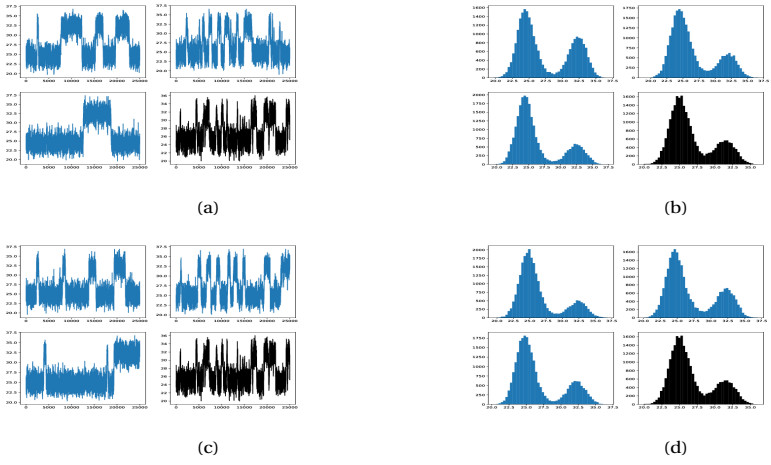


Figure 4: Trajectories obtained by forward simulating the DWP-SDE model based on parameter estimations from MCWM and ADA-GP-MCMC (samples from the high density region of the posterior distribution picked at random). Corresponding marginal distributions. Forward simulations are in blue; real data are in black. Subfigures: a) trajectories from MCWM, b) marginal distributions from MCWM, c) trajectories from ADA-GP-MCMC, and d) marginal distributions from ADA-GP-MCMC.

6 Pseudo-code for algorithms

Algorithm 4 DA-GP-MCMC algorithm

Input: Number of iterations R, probability to run standard MH update β_{MH}, a GP model fitted to the training data, a starting value θ^0 and corresponding $\hat{L}_u(\theta^0)$.
Output: The chain $\theta^{1:R}$.

for $r = 1, ..., R$ **do**
  Draw $u \sim \mathcal{U}(0,1)$.
  if $u \leq \beta_{MH}$ **then** ▷ Skip DA-part
    Propose $\theta^\star \sim \tilde{g}(\cdot|\theta^{r-1})$.
    Run a single iteration of PMCMC or MCWM for proposal $\theta^\star$.
  else
    Propose $\theta^\star \sim g(\cdot|\theta^{r-1})$. ▷ Run two stages DA scheme
    Sample from (3) to predict independently $\ell_{GP}(\theta^\star)$ and $\ell_{GP}(\theta^{r-1})$. Define $\hat{L}_{GP}(\theta^\star) := \exp(\ell_{GP}(\theta^\star))$ and $\hat{L}_{GP}(\theta^{r-1}) := \exp(\ell_{GP}(\theta^{r-1}))$.
    Compute $\alpha_1 = \min\left(1, \frac{\hat{L}_{GP}(\theta^\star)}{\hat{L}_{GP}(\theta^{r-1})} \cdot \frac{g(\theta^{r-1}|\theta^\star)}{g(\theta^\star|\theta^{r-1})} \cdot \frac{p(\theta^\star)}{p(\theta^{r-1})}\right)$.
    Draw $u \sim \mathcal{U}(0,1)$.
    if $u > \alpha_1$ **then** ▷ Early-reject
      Set $\theta^r = \theta^{r-1}$.
    else
      Estimate the likelihood $\hat{L}_u(\theta^\star)$. ▷ Second stage update scheme
      Compute $\alpha_2 = \min(1, \frac{\hat{L}_u(\theta^\star)}{\hat{L}_u(\theta^{r-1})} \cdot \frac{\hat{L}_{GP}(\theta^{r-1})}{\hat{L}_{GP}(\theta^\star)})$.
      Draw $u \sim \mathcal{U}(0,1)$.
      if $u \leq \alpha_2$ **then** ▷ Accept proposal
        Set $\theta^r = \theta^\star$.
      else
        Set $\theta^r = \theta^{r-1}$. ▷ Reject proposal
      end if
    end if
  end if
end for

Algorithm 5 ADA-GP-MCMC algorithm

Input: Number of iterations R, probability to run standard MH update β_{MH}, a GP model fitted to the training data, model $s_{13}()$ to select between case 1 and 3, model $s_{24}()$ to select between case 2 and 4, a starting value θ^0 and corresponding $\hat{L}_u(\theta^0)$.

for $r = 1, ..., R$ **do**
  Draw $u \sim \mathscr{U}(0,1)$.
  if $u \leq \beta_{MH}$ **then** ▷ Skip DA-part
    Propose $\theta^\star \sim \tilde{g}(\cdot|\theta^{r-1})$.
    Run a single iteration of PMCMC or MCWM for proposal $\theta^\star$.
  else
    Propose $\theta^\star \sim g(\cdot|\theta^{r-1})$. ▷ Run A-DA scheme
    Sample from the predictive distribution of the GP model to predict independently $\ell_{GP}(\theta^\star)$ and $\ell_{GP}(\theta^{r-1})$. Define $\hat{L}_{GP}(\theta^\star) := \exp(\ell_{GP}(\theta^\star))$ and $\hat{L}_{GP}(\theta^{r-1}) := \exp(\ell_{GP}(\theta^{r-1}))$.
    Compute $\alpha_1 = \min\left(1, \frac{\hat{L}_{GP}(\theta^\star)}{\hat{L}_{GP}(\theta^{r-1})} \cdot \frac{g(\theta^{r-1}|\theta^\star)}{g(\theta^\star|\theta^{r-1})} \cdot \frac{p(\theta^\star)}{p(\theta^{r-1})}\right)$.
    Draw $u \sim \mathscr{U}(0,1)$.
    if $u < \alpha_1$ **then** ▷ Run second stage of the A-DA scheme
      if $\hat{L}_{GP}(\theta^\star) > \hat{L}_{GP}(\theta^{r-1})$ **then**
        Select case 1 or 3 according to the model $s_{13}(\theta^\star)$.
        Run the accelerated delayed-acceptance scheme for the selected case.
      else
        Select case 2 or 4 according to the model $s_{24}(\theta^\star)$.
        Run the accelerated delayed-acceptance scheme for the selected case.
      end if
    else ▷ Early-reject
      Set $\theta^r = \theta^{r-1}$.
    end if
  end if
end for

7 MCMC trace plots and diagnostics plots for the GP model

Here we show some material pertaining our simulation and data analysis studies. We first report material pertaining the first application (stochastic Ricker model), then the second application (modelling of protein folding data).

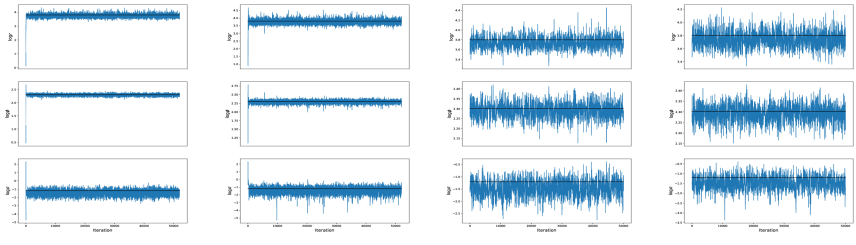
Quantities denoted as "residuals" are computed as:

$$r_i = \ell_{PF}(\theta^{\star,i}) - \ell_{GP}(\theta^{\star,i}), \qquad i = 1, \ldots, N_{test}$$

where N_{test} is the number of observations in the test data $\mathscr{D}_2$.

Ricker model

Here follow trace plots for MCMC chains obtained under different methods.



(a) PMCMC. (b) MCMC. (c) ADA-GP-MCMC. (d) ADA-GP-MCMC.

Fit of the GP model.

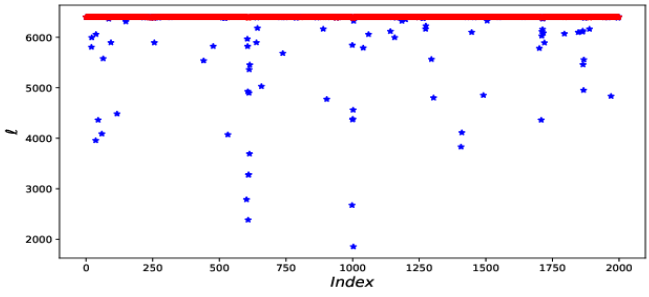
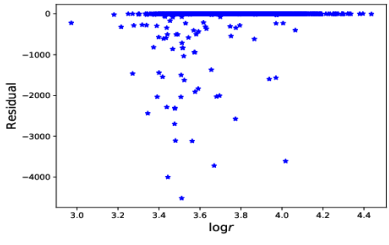
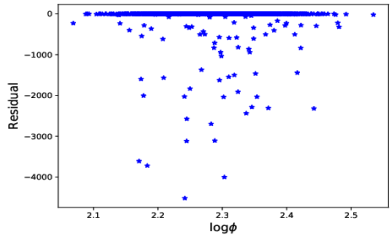


Figure 6: Log-likelihood estimations; particle filter (blue), Gaussian process model (red).

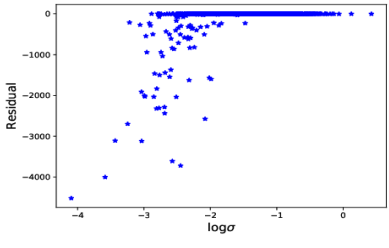
Residual plots.



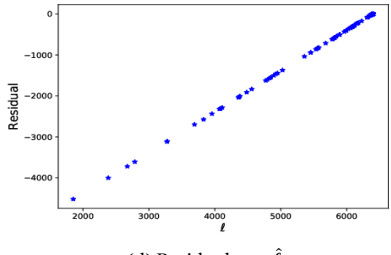
(a) Residuals vs. $\log r$.



(b) Residuals vs. $\log \phi$.



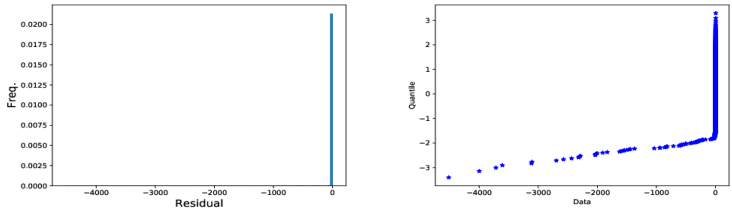
(c) Residuals vs. $\log \sigma$.



(d) Residuals vs. $\hat{\ell}_{PF}$.

Figure 7: Residual plots.

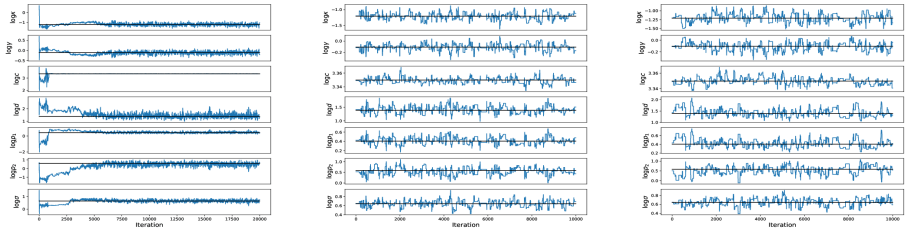
Histogram and normal probability plot of the residuals.



(a) Histogram. (b) Normal probability plot.

DWP-SDE model for simulated data

Here follow trace plots for MCMC chains obtained under different methods.



(a) MCWM. (b) DA-GP-MCMC. (c) ADA-GP-MCMC.

Fit of the GP model.

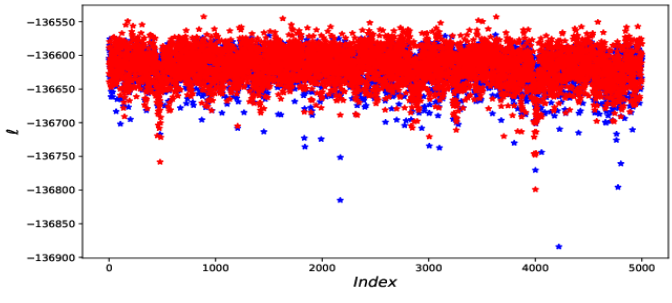
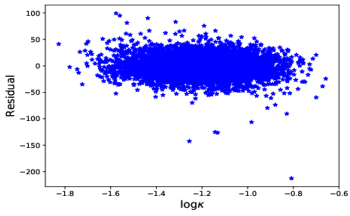
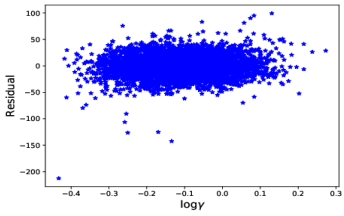


Figure 10: Log-likelihood estimations; particle filter (blue), Gaussian process model (red).

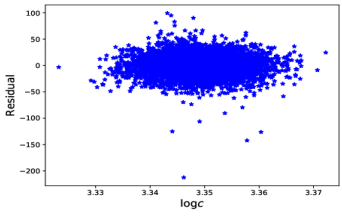
Residual plots.



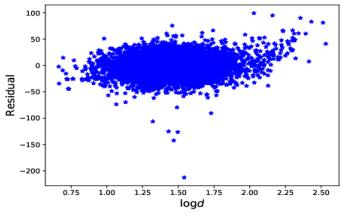
(a) Residuals vs. $\log\kappa$.



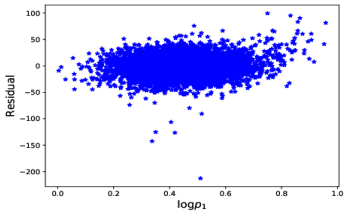
(b) Residuals vs. $\log\gamma$.



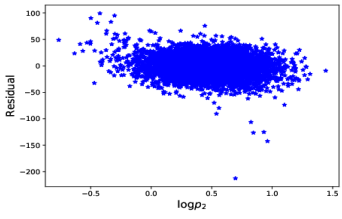
(c) Residuals vs. $\log c$.



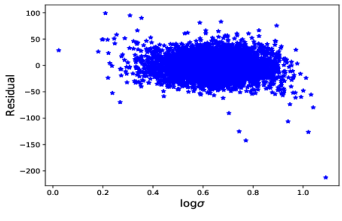
(d) Residuals vs. $\log d$.



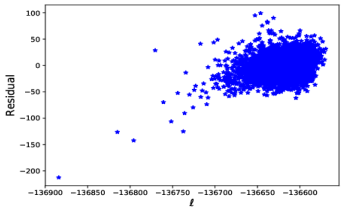
(e) Residuals vs. $\log p_1$.



(f) Residuals vs. $\log p_2$.

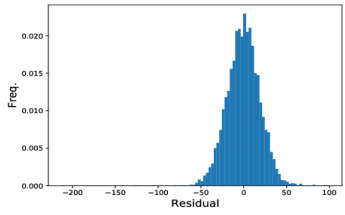


(g) Residuals vs. $\log\sigma$.

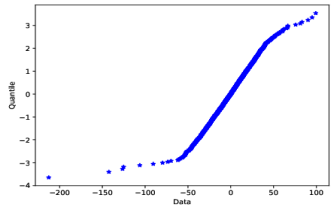


(h) Residuals vs. $\hat{\ell}_{PF}$.

Histogram and normal plot of residuals.



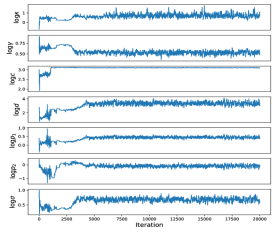
(a) Histogram.



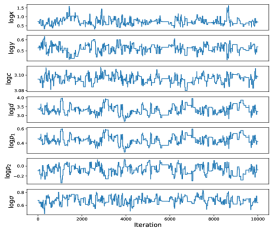
(b) Normal plot.

DWP-SDE model for protein folding data

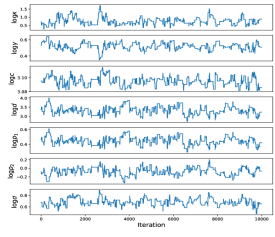
Here follow trace plots for MCMC chains obtained under different methods.



(a) MCWM.



(b) DA-GP-MCMC.



(c) ADA-GP-MCMC.

Fit of the GP model.

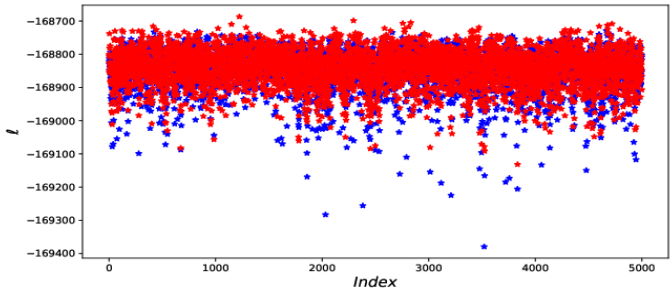
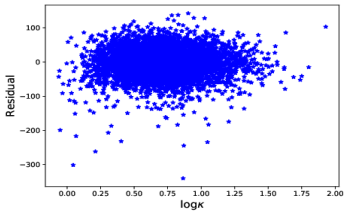
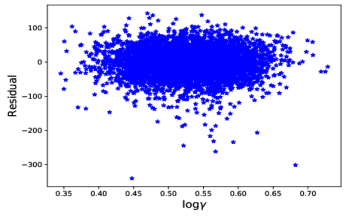


Figure 14: Log-likelihood estimations; particle filter (blue), Gaussian process model (red).

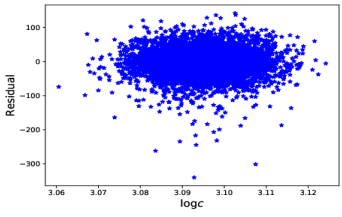
Residual plots.



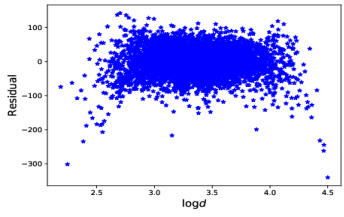
(a) Residuals vs. $\log\kappa$.



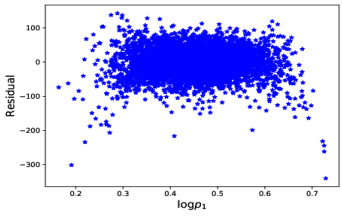
(b) Residuals vs. $\log\gamma$.



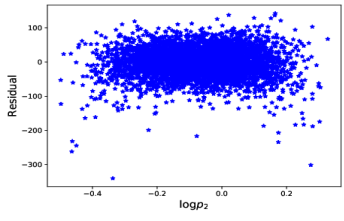
(c) Residuals vs. $\log c$.



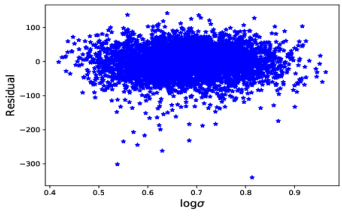
(d) Residuals vs. $\log d$.



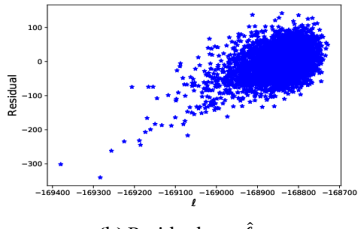
(e) Residuals vs. $\log p_1$.



(f) Residuals vs. $\log p_2$.

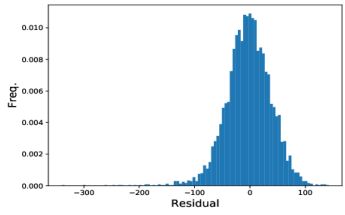


(g) Residuals vs. $\log\sigma$.

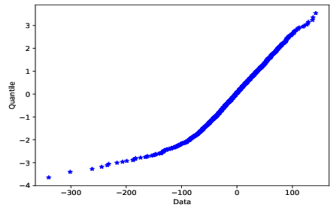


(h) Residuals vs. $\hat{\ell}_{PF}$.

Histogram and normal plot of residuals.



(a) Histogram.



(b) Normal plot.

References

C. Andrieu and G. O. Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37:697–725, 2009.

C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

M. A. Beaumont. Estimation of population growth or decline in genetically monitored populations. *Genetics*, 164(3):1139–1160, 2003.

J. Bezanson, A. Edelman, S. Karpinski, and V. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

O. Cappé, E. Moulines, and T. Ryden. *Inference in Hidden Markov Models*. Springer, 2005.

O. Cappé, S. J. Godsill, and E. Moulines. An overview of existing methods and recent advances in sequential Monte Carlo. *Proceedings of the IEEE*, 95(5):899–924, 2007.

C. C. Drovandi. Pseudo-marginal algorithms with multiple CPUs. https://eprints.qut.edu.au/61505/, 2014.

C. C. Drovandi, M. T. Moores, and R. J. Boys. Accelerating pseudo-marginal MCMC using Gaussian processes. *Computational Statistics & Data Analysis*, 118:1–17, 2018.

A. Golightly, D. A. Henderson, and C. Sherlock. Delayed acceptance particle MCMC for exact inference in stochastic kinetic models. *Statistics and Computing*, 25(5):1039–1055, 2015.

N. J. Gordon, D. J. Salmond, and A. F. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)*, volume 140, pages 107–113. IET, 1993.

P. E. Jacob. Sequential Bayesian inference for implicit hidden Markov models and current limitations. *ESAIM: Proceedings and Surveys*, 51:24–48, 2015.

N. Kantas, A. Doucet, S. S. Singh, J. Maciejowski, and N. Chopin. On particle methods for parameter estimation in state-space models. *Statistical Science*, 30(3):328–351, 2015.

G. Karabatsos and F. Leisen. An approximate likelihood perspective on ABC methods. Forthcoming in *Statistics Surveys*. Also available as arXiv:1708.05341, 2017.

G. Kitagawa. Monte Carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of computational and graphical statistics*, 5(1):1–25, 1996.

F. J. Medina-Aguayo, A. Lee, and G. O. Roberts. Stability of noisy Metropolis–Hastings. *Statistics and Computing*, 26:1187–1211, 2016.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

M. K. Pitt, R. dos Santos Silva, P. Giordani, and R. Kohn. On some properties of Markov chain Monte Carlo simulation methods based on the particle filter. *Journal of Econometrics*, 171(2):134–151, 2012.

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT press, 2006.

C. Sherlock, A. H. Thiery, G. O. Roberts, J. S. Rosenthal, et al. On the efficiency of pseudo-marginal random walk Metropolis algorithms. *The Annals of Statistics*, 43(1):238–275, 2015.

C. Sherlock, A. Golightly, and D. A. Henderson. Adaptive, delayed-acceptance MCMC for targets with expensive likelihoods. *Journal of Computational and Graphical Statistics*, 26(2):434–444, 2017.

S. Sisson and Y. Fan. *Handbook of Markov Chain Monte Carlo*, chapter Likelihood-free MCMC. Chapman & Hall/CRC, New York.[839], 2011.

Printed by Media-Tryck, Lund 2021 NORDIC SWAN ECOLABEL 3041 0903

Doctoral Theses in Mathematical Sciences 2021:09

ISRN LUNFMS-1029-2021
ISBN 978-91-7895-967-9
ISSN 1404-0034