Neural Posterior Estimation for Simulators with High Parameter Dimensions

Paul Will Maximilian Fuhr

Geboren am 26.02.2002 in Wiesbaden

14. Mai 2024

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Jan Hasenauer

Zweitgutachter: Dr. Stefan Kesselheim

Limes - IRU Mathematics and Life Sciences

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Zusammenfassung

Diese Arbeit trägt zur systematischen Evaluierung von Algorithmen im Bereich simulations-basierter Inferenz [CBL20] bei. Dafür wird die Performanz eines Algorithmus, Neural Posterior Estimation (NPE), für mehrere Inferenzprobleme mit unterschiedlichen Parameterdimensionen (4-81 Parameter) evaluiert. Die Arbeit berichtet berichtet Validierungsstatistiken, die die Glaubwürdigkeit der geschätzten a-posteriori Verteilungen bewerten. Weiter werden Einführungen zu Bayesscher Inferenz, simulations-basierter Inferenz, normalisierten Flüssen und bedingten normalisierten Flüssen sowie zu stochastischen biochemischen Reaktionsnetzwerken gegeben.

Abstract

Recent works have started to systematically evaluate simulation-based inference [CBL20] algorithms. This work contributes to this by evaluating the performance of one algorithm, neural posterior estimation (NPE), for several inference problems with different parameter dimensions, ranging from 4 to 81 parameters. Different validation statistics are reported that assess the faithfulness of the estimated posteriors. The work further provides detailed introductions to Bayesian inference, simulation-based inference, normalizing flows and conditional normalizing flows, and stochastic biochemical reaction networks.

Contents

List of Tables

List of Figures

Notation

A list of relevant symbols can be found in Table 1.

Parameters

The notion of a *parameter vector* or simply *parameters* refers to a vector $\theta \in \mathbb{R}^d$. A component θ_j of $\theta = (\theta_1, ..., \theta_d)$ is then called *parameter*. The first $d-1$ components of θ are denoted by $\theta_{<d} := (\theta_1, \dots, \theta_{d-1})$. For $\theta \in \mathbb{R}^d$ we also define $|\theta| := d$, the dimension of θ, sometimes simply referred to as *number of parameters* or *parameter dimension*.

Probability Distributions, Probability Density Functions and Random Variables

We denote probability density functions, such as $p(\theta)$ or $p(\theta|x_0)$ with lowercase letters. If probability distributions, such as P, Q are defined explicitly, they are denoted with capital letters. Often we write the probability density function but refer to the underlying distribution. For example we often write "we sample from $p(\theta)$" instead of writing "we sample from the distribution that is implicitly defined by the probability density function $p(\theta)$". Random variables and variables are usually denoted in symbols without additional indices, such as θ, x. If these variables have concrete values, e.g are a sample of a random variable, we usually add the index 0, such as x_0, θ_0. For example, $p(\theta|x_0)$ is a probability density function with fixed conditioning variable x_0 that can be evaluated at arbitrary θ.

Symbol	Meaning
$\mathbb{1}$	characteristic function
$\mathcal{N}(\mu, \sigma)$	normal distribution with mean μ and standard deviation σ
$\propto$	proportional to
$\nabla_x f(x, y)$	vector of partial derivatives of f at (x, y) with respect to x
$f(x; \xi)$	evaluation of function f, that depends on parameters ξ, at x
supp f	support $\{x \in \mathcal{X}, s.t\ f(x) > 0\}$ of function $f : \mathcal{X} \to \mathcal{Y}$
Jacobian matrix of f	J_f
$\sim$	sample from, e.g $x \sim p$
$A \cong B$	there exists a homeomorphism between A and B, see Definition A.2
$P_n \xrightarrow{\mathcal{D}} Q$	weak convergence of sequence of probability distributions P_n to probability distribution Q as $n \to \infty$

Table 1: Relevant symbols

1 Introduction

Many domains of science, such as physics [Bre+18], evolutionary biology [Rat+07], neuroscience [Gon+20] and economics [GMR93] make extensive use of mechanistic models that can be used to predict how systems will behave in a variety of circumstances. The expressiveness of modern programming languages allows scientists to implement these models as stochastic numerical simulators and generate synthetic data from them. A key challenge is estimating the parameters of the simulator that are consistent with observed data. We focus on inferring parameters with a Bayesian approach and give an introduction to Bayesian inference in Section 2. The likelihood, that is the probability of the observed data given parameters, is often intractable [1]. This renders conventional approaches difficult. The goal of simulation-based inference (SBI), also known as 'likelihood-free inference', is to perform Bayesian inference for simulators with an intractable likelihood. See Section 3 for an introduction. Recent progress in probabilistic machine learning and generative modelling, such as the development of normalizing flows [RM15], [PPM17] and diffusion models [Son+20], [HJA20] improved sampling from and approximating complex probability distributions. This not only stimulated developments in image, video and audio generation, but also energized the development of new algorithms for SBI [PSM19], [PM16] [GNM19]. Section 3.2 reviews these developments in more detail. These algorithms use the expressive power of neural networks to model the conditional probability distributions between simulated data and parameters, which are then used to perform inference on observed data. Section 4 explains in detail how one can use normalizing flows to construct flexible probability distributions and how one can translate the ideas to model conditional probability distributions.

Recent studies compare and analyze the performance of these algorithms [Lue+21] as well as the validity of the obtained posteriors [Her+21] for different types of simulators. These benchmarks lack performance analysis on simulators with high parameter dimensions, as all benchmark problems had 10 or fewer parameters. In the experiment contained in Section 6, we aim to contribute to filling this gap. We explore how well neural posterior estimation (NPE) (Algorithm 1), one type of SBI algorithm, works when estimating high-dimensional posterior densities. For this, we use a neural spline flow [Dur+19] to infer the approximate posteriors of simulators with different numbers of parameters. We define multiple simulators within a stochastic biochemical reaction network, each with a different number of parameters, ranging from 4 to 81 parameters. An introduction to stochastic biochemical reaction networks, including their definition and algorithms to simulate

[1] impossible to compute analytically, very expensive to be computed numerically

them, is given in Section 5. We report the performance and validity of the obtained posteriors using different evaluation and validation metrics for different simulation budgets.
Similarly to [Her+21], we find that NPE generally produces slightly overconfident results. This can be reduced by using higher simulation budgets (≥ 10000). In addition to that, our experiments indicate that a higher parameter dimension does not lead to increasingly overconfident results if we are working with larger simulation budgets (≥ 10000). On the contrary, when working with lower simulation budgets (1000), we provide evidence that NPE tends to be increasingly overconfident in higher parameter dimensions.

2 Bayesian Inference

A statistical model is a mathematical model that embodies a set of statistical assumptions concerning the parameter-dependent generation of data. Given some observed data we would like to infer the parameters of the statistical model. For example, we may have a model that is able to simulate the evolution of a pandemic given parameters (e.g basic reproduction number). Given the actual evolution of the pandemic, we would like to infer the parameters of the model to predict the future evolution of the pandemic. The following section gives an introduction to parameter inference with a Bayesian approach.

In the following section all probability distributions defined on continuous spaces are assumed to be absolutely continuous, such that they have a well-defined probability density function.

2.1 Bayes' Theorem

In statistical modelling we want to understand the generation of data $X \in \mathbb{R}^m$ dependent of parameters $\theta \in \mathbb{R}^d$. In classical statistics the output X is random with a density or probability mass function $p(x|\theta)$, but θ is treated as a fixed but unknown parameter. Instead, in Bayesian statistics X and θ are both regarded as random variables with a joint density or (probability mass function) $p(\theta, x)$. [YSS05] We recall that we were interested in inferring the parameters θ given observed data $x_0 \in \mathbb{R}^d$.

Definition 2.1. Bayes' Theorem [Bay63] [2] gives us the *posterior*, the conditional probability density (probability mass function respectively) $p(\theta|x_0)$ of parameters θ given x_0:

$$p(\theta|x_0) = \frac{p(x_0|\theta)p(\theta)}{p(x_0)} = \frac{p(x_0|\theta)p(\theta)}{\int p(x_0|\theta)p(\theta)\, d\theta} \tag{2.1}$$

We define the *likelihood* $p(x|\theta)$ as the conditional probability density (probability mass function respectively) of observing data x given parameters θ. For continuous random variables, the likelihood is sometimes called *likelihood function*.
In Bayesian statistics we have a given prior belief that the parameters are distributed according to $p(\theta)$. We thus call $p(\theta)$ the *prior*, for continuous parameters it is sometimes called *prior density*.

Bayes Theorem gives a simple rule to update the prior given new evidence (new observed data). This is illustrated by the following example.

[2] note that the Bayes Theorem is true for probability density functions as well as for probability distributions

Example 2.2. We may like to model the time between events as a random variable $X \sim \exp(\theta)$ distributed according to an exponential distribution with parameter $\theta \in (0, 1]$. The likelihood (function) is given by $p(x|\theta) = \theta e^{-\theta x} \mathbb{1}_{x \in [0,\infty]}$. Assuming we don't have any prior knowledge about θ, we define the prior to be the uniform distribution, namely $p(\theta) = \mathbb{1}_{\theta \in (0,1]}$. If we measure the time between events one time and obtain $x_0 = 1.0$, we can compute the posterior given this observation with Bayes' Theorem (2.1):

$$\begin{aligned} p(\theta|x_0 = 1.0) &= \frac{p(x_0 = 1.0|\theta)p(\theta)}{\int p(x_0 = 1.0|\theta)p(\theta)\, d\theta} \\ &= \frac{\theta e^{-\theta} \mathbb{1}_{\theta \in (0,1]}}{\int_0^1 \theta e^{-\theta}\, d\theta} \\ &= \frac{\theta e^{-\theta} \mathbb{1}_{\theta \in (0,1]}}{1 - e^{-1}} \end{aligned}$$

In the context of simulation-based inference, the data-generating process of the statistical model is specified by a simulator.

Definition 2.3. [CBL20] A *simulator* is a computer program that takes as input a vector of parameters $\theta_0 \in \mathbb{R}^d$, samples a series of internal states or latent variables $z_i \sim p_i(z_i|\theta, z_{<i})$, where $z \in \mathbb{R}^k$ and finally produces an output vector $x_0 \sim p(x|\theta, z)$, where $x_0 \in \mathbb{R}^m$. In short, a simulator takes input $\theta_0 \in \mathbb{R}^d$ and produces output $x_0 \in \mathbb{R}^m$.

Example 2.4. Example 2.2 defines a simulator that takes as input a parameter $\theta_0 \in (0, 1]$ and outputs $x_0 \sim \exp(\theta_0)$.

Remark 2.5. When generating simulator output x_0 from a Simulator given parameters θ_0, we write "sampling $x_0 \sim p(x|\theta_0)$.

The simulator aims to model a real system, e.g the evolution of a pandemic. Similarly to before, given observed data x_0, we would like to infer the parameters with a Bayesian approach, that is, compute the posteriors $p(\theta|x_0)$ [3]. To apply Bayes' Theorem (2.1) we need to know the likelihood of simulator output x_0 given parameters θ, but depending on the simulator, we may not know it. We will now discuss different ways to infer parameters of simulators given observed data x_0 in the presence of a tractable likelihood, (Section 2.1.1) and absence of a tractable likelihood (Section 3).

2.1.1 Likelihood Tractable

For simple simulators, the likelihood function $p(x|\theta)$ is tractable, namely it can be computed (analytically or numerically).

[3] In certain cases it may also be interesting to infer the latent variables, see [CBL20]

Example 2.6. In biochemical reaction networks, we are interested in modelling the time-dependent concentrations of different species, which interact by some internal dynamics. We consider ordinary differential equation models for biochemical reaction networks as in [Frö+17]:

$$\nabla_t x(t,\theta) = f(x,\theta),\ x(t_0) = x_0(\theta),$$

where $x(t,\theta) \in \mathbb{R}^{n_x}$ is the concentration vector at time t and $\theta \in \mathbb{R}^{n_\theta}$ the parameter vector. The parameters often have a biological function, such as synthesis rates. The function $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_x}$ determines the temporal evolution of the concentrations of the biological species. The mapping $x_0 : \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_x}$ provides the parameter dependent initial concentrations at time t_0. We consider the output map $h : \mathbb{R}^{n_x} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_y}$, which models the measurement process, i.e. the output (or observables) $y(t,\theta)$ at time point t as a function of the state variables and the parameters,

$$y(t,\theta) = h(x(t,\theta),\theta)$$

The i-th observable y_i can be the concentration of a particular biochemical species (e.g. $y_i = x_l$) as well as a function of several concentrations and parameters (e.g. $y_i = \theta_m(x_{l_1} + x_{l_2})$.
Say we are only interested in measuring y_1 at $t = 1$. The measurement process is assumed to be noisy, that is,

$$\overline{y} = y_1(1,\theta) + e, \text{ with } e \sim \mathcal{N}(0,\sigma),$$

where σ is assumed to be independent of the models internal state. In consequence, given parameters θ, we can simulate data

$$\overline{y} \sim \mathcal{N}(h_1(x(t,\theta),\theta),\sigma),$$

the likelihood is tractable and given by

$$\mathcal{N}(h_1(x(t,\theta),\theta),\sigma).$$

Given a tractable likelihood, we can draw samples from the true posterior distribution with Markov-Chain-Monte-Carlo (MCMC) methods, such as the Metropolis-Hastings algorithm [Met+53]. To apply these methods we only need a function proportional to the true posterior density which is given by *likelihood* $\times$ *prior* (2.1):

$$p(\theta|x_0) \propto p(x_0|\theta)p(\theta) \tag{2.2}$$

A more detailed introduction to MCMC methods can be found in [Bro+11]. With a tractable likelihood, we can also directly estimate parameters by maximum-likelihood-estimation [Frö+17].

2.1.2 Likelihood Intractable

Unfortunately, the likelihood function is often intractable, as it corresponds to an integral over all possible execution traces of the simulator.

$$p(x|\theta) = \int p(x, z|\theta)\, dz \tag{2.3}$$

Computing this integral for real-life simulators with many stochastic steps and large latent spaces is clearly impossible.

3 Simulation-based Inference

This introduction is based on [CBL20].
Simulation-based inference (SBI) describes methods to infer parameters in the absence of a tractable likelihood function. It entails methods that estimate the posterior density or sample approximately from the posterior density. We describe traditional methods in Section 3.1 and recent methods in Section 3.2.

3.1 Traditional Methods

Approximate Bayesian Computation

Approximate Bayesian computation (ABC) allows to sample approximately from the posterior $p(\theta|x_0)$.
The core idea of rejection-ABC [BZB02] is to sample $\theta_0 \sim p(\theta)$ from the prior, simulate a corresponding output $x_{\theta_0} \sim p(x|\theta_0)$ and accept θ_0 as a true sample if $d(x_0, x_{\theta_0}) < \epsilon$. Here, d is a distance measure on the output space and ϵ a tolerance. [4]
In the limit $\epsilon \to 0$, inference with ABC becomes exact under certain continuity assumptions. This is because we are only accepting parameter values which produce simulation outputs that are arbitrarily close to our observation x_0 . In practice, very small values of ϵ require too many simulations, as the probability of accepting drawn parameter values θ also decreases.
There exist many variants of ABC:
For example, one can define a summary statistic $S : \mathbb{R}^m \to \mathbb{R}^s$, $s < m$, on the output space, which maps the output to a lower dimensional space. One would then accept θ_0 as a true sample if $d_S(S(x_0), S(x_{\theta_0})) < \epsilon$, where d_S is a distance measure on the summary statistic space.
There also exist the established MCMC-ABC [Mar+03] and SMC-ABC [SFT07] [PFS12], which are based on MCMC and Sequential-Monte-Carlo methods respectively. These methods produce correlated samples, but need fewer simulations.

Kernel Density Estimators

Another classical approach to simulation-based inference is based on creating a surrogate model for the likelihood by estimating the distribution of simulated data with histograms or kernel density estimation. Bayesian inference then proceeds using the tractable surrogate likelihood.

[4] Intuitively, this corresponds to approximating the posterior with $p(\theta|x_0) \approx p(\theta|d(x_0, x_{\theta_0}) < \epsilon)$.

3.2 Frontier in Simulation-based Inference

In recent years new approaches have improved sample efficiency, overall quality of inference and amortization with respect to these traditional methods. Ref. [CBL20] loosely groups them into three main directions of progress.

Machine Learning

A very powerful method used in simulation-based inference is learning approximations of likelihood function, posterior density (or likelihood ratio) based on flexible conditional neural density estimators, such as normalizing flows (which we worked with, see Section 4), continuous normalizing flows [Che+18], diffusion models [Son+20], [HJA20] or Flow-Matching [Lip+22], [ABV23].
In simple terms, conditional neural density estimators $q(\theta|x;\xi)$ learn an approximation parameterized by ξ of the true posterior density $p(\theta|x)$[5]. They are trained by minimizing a discrepancy between true posterior $p(\theta|x)$ and approximated posterior $q(\theta|x)$ while only requiring access to simulated samples of the joint distribution $p(\theta, x) = p(\theta)p(x|\theta)$. Normalizing flows, for example, aim to minimize the expected Kullback-Leibler divergence (KL-divergence) between true posterior and estimated posterior. Therefore, they are similar to kernel density estimators, which also estimate probability densities.
The important difference is that these neural density estimators incorporate neural networks, which are very expressive and flexible. Moreover, we can sample directly from these neural networks without requiring an additional sampling step, such as rejection sampling. As a result, inference algorithms profit directly from the impressive progress in machine learning.

We obtain the most basic SBI algorithm: neural posterior estimation [RM15]. This algorithm consists of constructing a training dataset from simulated data and subsequently training the density estimator on that training dataset.

[5]Note that we emphasize posterior densities to simplify the notation, but learning approximations of the likelihood function works analogously.

Algorithm 1 Neural Posterior Estimation (NPE)

Input: a prior $p(\theta)$, a simulator, simulation budget L, a network architecture A

Output: an estimator q of the posterior density

Initialization: $D \leftarrow \emptyset$, q initialized according to A, $l \leftarrow 0$

while $l < L$ **do**

  sample $\theta_0 \sim p(\theta)$

  simulate $x_0 \sim p(x|\theta_0)$ from the simulator

  add (θ_0, x_0) to D

  $l \leftarrow l + 1$

train q on D to minimize KL-divergence

The training step for normalizing flows is described in detail in Section 4. Note that this approach is amortized. Our density estimator can be evaluated for arbitrary observations x_0.

Active Learning

NPE (Algorithm 1) only samples from the prior when simulating data. In consequence, we learn the posterior for all possible observations x. Such a strategy however is very inefficient if we are only interested in the posterior density $p(\theta|x_0)$ given a concrete observation x_0 or the likelihood function $p(x_0|\theta)$ evaluated at a concrete observation x_0 . An "actively learning" algorithm would be more suitable: These algorithms use proposal priors to sequentially guide the simulator through the parameter space into more informative regions. The density estimator is then trained or iteratively refined on these more informative samples. See [PM16], [GNM19] for sequentially estimating the posterior and [PSM19] for sequentially estimating the likelihood.

Automatic Differentiation and Probabilistic Programming

The integration of automatic differentiation [Bay+18] and probabilistic programming [Bay+19] into the simulation code allows computing additional features when generating simulated samples. Examples include:

$$\nabla_\theta \log p(x, z|\theta), \nabla_z \log p(x|z, \theta)$$

These additional quantities can be used to augment the training data, which can improve inference quality [GS17], [Bre+20].

An overview of the SBI workflow is given in Figure 1. This figure was created by me, but is based on the figures in [CBL20].

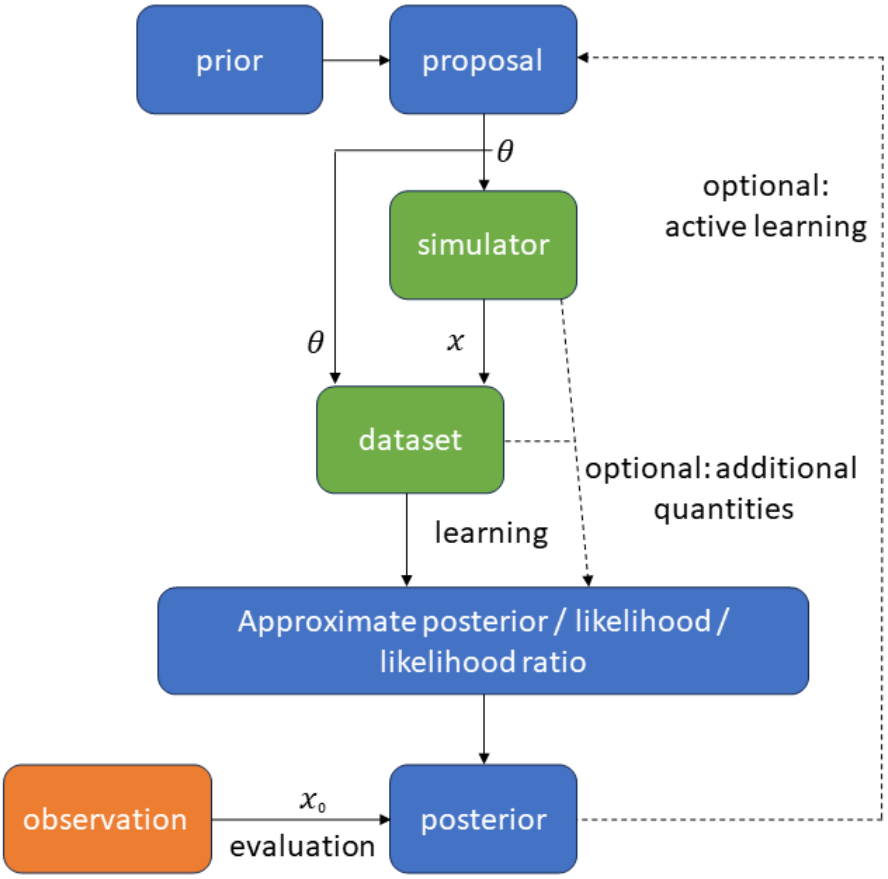

Figure 1: Simulation based inference workflow

4 Normalizing Flows

This introduction is based on [Pap+21]. Normalizing flows provide a general way of constructing flexible probability distributions on $\mathbb{R}^d$, called flow-based distributions, that can be fitted to sampled data of a target distribution. The ideas translate to conditional probability distributions. The following Section 4.1 introduces normalizing flows. Section 4.2 explains how to train normalizing flows to approximate a target distribution and discusses the expressiveness of normalizing flows. We contribute Section 4.3, which explains conditional normalizing flows and the considerations that are needed for probabilities with restricted support (Section 4.3.2). Section 4.4 gives details on the construction of normalizing flows and conditional normalizing flows (our contribution) using invertible neural networks.

4.1 Basic Definitions

To define flow-based distributions, we require one definition.

Definition 4.1. Given a probability distribution P on $\mathbb{R}^d$ and a measurable map $T : \mathbb{R}^d \to \mathbb{R}^k$ the *push-forward* of P is the distribution T^*P on $\mathbb{R}^k$, defined by

$$T^*P(A) = P(T^{-1}(A)) \text{ for } A \subset \mathbb{R}^k \text{ measurable.}$$

General Idea:

Suppose we would like to define a flexible distribution of a d-dimensional real random variable Y. The main idea of flow-based modeling is to express Y as a *transformation* $T : \mathbb{R}^d \to \mathbb{R}^d$ of a random variable U distributed according to a known *base distribution* P_u on $\mathbb{R}^d$:

$$Y = T(U) \text{ with } U \sim P_u \tag{4.1}$$

The random variable Y is then distributed according to T^*P_u, the push-forward of P_u (as defined in Definition 4.1).

Definition 4.2. Let P_u be a probability distribution on $\mathbb{R}^d$, $T : \mathbb{R}^d \to \mathbb{R}^d$ a diffeomorphism[6]. Then, the *flow-based distribution* is the push-forward distribution T^*P_u. If P_u has a well-defined probability density function p_u, also T^*P_u has a well-defined probability density function q which can be obtained by the change of variables formula:

$$q(y) = p_u(T^{-1}(y))|\det J_T(T^{-1}(y))|^{-1} \tag{4.2}$$

where J_T is the $d \times d$-matrix of all partial derivatives of T. This density is called *flow-based density*. The flow-based distribution is sometimes called *flow* or *normalizing flow*.

[6] invertible, T as well as T^{-1} differentiable

Remark 4.3. We can sample from this flow based density according to equation (4.1). We can modify this flow-based distribution by modifying either the diffeomorphism T or the base distribution P_u.

If T has parameters $\phi \in \mathbb{R}^l$ and p_u has parameters $\psi \in \mathbb{R}^s$, this defines a parameterized flow-based density $q(y;\xi)$, where $\xi = (\phi, \psi) \in \mathbb{R}^l \times \mathbb{R}^s$. Sometimes, T is then denoted by T_ϕ.

4.2 Using Flows for Modelling

We can then fit a flow based density $q(y;\xi)$, to a target density $p(y)$ by minimizing some divergence or discrepancy between them. Typically, this is done by minimizing the forward KL-divergence, a measure of similarity between p and q, over the parameters ξ.
As we will see in Section 4.4, the transform T usually incorporates neural networks. We then want to optimize the parameters of the neural networks, such as weights and biases, as well as the other parameters defining the transform T.

Forward KL-divergence

Definition 4.4. Let P, Q be two absolutely continuous probability distributions on $\mathbb{R}^d$ with probability density functions p, q: We then define the forward KL-divergence between P and Q as

$$D_{KL}(P||Q) = D_{KL}(p||q) = D_{KL}(p(x)||q(x)) := \mathbb{E}_{p(x)}[\log \frac{p(x)}{q(x)}] \tag{4.3}$$

By Proposition A.1, the forward KL-divergence between the target distribution $p(y)$ and the flow based distribution $q(y;\xi)$ can be written as

$$\begin{aligned}\mathbb{L}(\xi) &= D_{KL}(p(y)||q(y;\xi)) \\ &= -\mathbb{E}_{p(y)}[\log q(y;\xi)] - \mathbb{H}_{p(y)}(x) \\ &= -\mathbb{E}_{p(y)}[\log q(y;\xi)] + const.\end{aligned}$$

as *const.* does not depend on ξ.
This objective function is well-suited when we have samples from the target distribution, but we cannot evaluate the target density $p(y)$. By Monte-Carlo we have

$$\mathbb{L}(\xi) \approx -\frac{1}{N}\sum_{i=1}^{N} \log q(y^i;\xi) + const.$$

where $\{y^i\}_{i=1}^N$ is sampled from $p(y)$.

Optimization

Usually the parameters are optimized with gradient-based methods, in particular stochastic gradient descent [Rud16]. Gradient descent in its most basic form starts with an initial configuration of ξ and iteratively updates ξ in the direction of the gradient of a loss function $\nabla_\xi \mathbb{L}(\xi)$. In consequence, ξ converges to a local minimum of the loss function. For this, we obtain an unbiased estimate of the gradient of the KL-divergence $\nabla_\xi \mathbb{L}(\xi) = (\nabla_\phi \mathbb{L}(\xi), \nabla_\psi \mathbb{L}(\xi))$ as follows:

$$\nabla_\phi \mathbb{L}(\xi) \approx -\frac{1}{N} \sum_{i=1}^{N} \nabla_\phi \log p_u(T^{-1}(y_i;\phi);\psi) + \nabla_\phi \log |\det J_{T^{-1}}(y_i;\phi)| \quad (4.4)$$

and

$$\nabla_\psi \mathbb{L}(\xi) \approx -\frac{1}{N} \sum_{i=1}^{N} \nabla_\psi \log p_u(T^{-1}(y_i;\phi);\psi) \quad (4.5)$$

To compute equation (4.4) we need to be able to differentiate p_u at $(T^{-1}(y_i;\phi);\psi)$ with respect to all components of $T^{-1}(y_i;\phi)$ and differentiate T^{-1} at $(y_i;\phi)$ with respect to all the components of ϕ. Obviously, the log function is differentiable for $\mathbb{R}_{>0}$. To compute equation (4.5) we need to be able to differentiate p_u at $(u;\psi)$ with respect to the components of ψ.
For most practical examples, this condition is trivial, but it is important to keep in mind when constructing flow-based models. Often $p_u(u;\psi)$ is kept constant, e.g. as a standard normal distribution, and we're only optimizing over ϕ.

In summary, to train our flow-based density to minimize the KL-divergence, we need to be able to compute $p_u(u;\psi)$, $T^{-1}(y;\phi)$ and its Jacobian determinant $|\det J_{T^{-1}}(y_i;\phi)|$, as well as satisfy the conditions listed above. Note that we don't need access to T nor the ability to sample from $p_u(u;\psi)$ to train the flow. However, we would need access to these operations to sample from the model after.

For the next section, we recall the following two definitions:
The Lipschitz constant Lip of a function $f : \mathcal{X} \to \mathcal{Y}$, for $\mathcal{X} \subset \mathbb{R}^{n_\mathcal{X}}$, $\mathcal{Y} \subset \mathbb{R}^{n_\mathcal{Y}}$ is the infimum over $M \in [0, \infty)$, such that $||f(x) - f(x')|| \leq M||x - x'||$ for all $x, x' \in \mathcal{X}$.
For a Borel measure μ on $\mathbb{R}^d$, the support of μ, denoted supp μ, is the set of all $z \in \mathbb{R}^d$, such that $\mu(N_z) > 0$ for every open set N_z containing z.

Expressiveness:

Flows allow very expressive modelling. Indeed, one can show the following theorem as in [Pap+21].

Theorem 4.5. *Let Y^*, U be two random variables on $\mathbb{R}^d$ distributed according to P_{Y^*}, P_U with well-defined densities q^*, p_u respectively, such that the following conditions hold for all $y, u \in \mathbb{R}^d$:*

1. *$q^*(y) > 0$, $p_u(u) > 0$*
2. *$\mathbb{P}(Y_i^* \leq y_i | Y_{<i}^* = y_{<i})$, $\mathbb{P}(U_i \leq u_i | U_{<i} = u_{<i})$ are differentiable with respect to $(y_i, y_{<i}), (u_i, u_{<i})$ respectively*

Under these conditions there exists a diffeomorphism $T : \mathbb{R}^d \to \mathbb{R}^d$, such that the flow-based density q defined by T and p_u is equal to q^*.

Remark 4.6. The main idea of the proof is use the observation that for any one-dimensional random variable X with density $p(x) > 0$ for all $x \in \mathbb{R}$ the function $x \mapsto P(X \leq x)$ is a bijection $\mathbb{R} \to (0, 1)$. This can be used to define $F : \mathbb{R}^d \to (0, 1)^d, u \mapsto z$ via $z_i = \mathbb{P}(U_i \leq u_i | U_{<i} = u_{<i}) = \int_{-\infty}^{u_i} p_u(u | u_{<i})$, which can be shown to be a diffeomorphism that turns p_u into the uniform distribution on $(0, 1)^d$. Likewise, one can define a diffeomorphism G that turns q^* into the uniform distribution on $(0, 1)^d$. The total transformation $G^{-1} \circ F$ then turns p_u to q^*. The exact proof can be found on page 5 in [Pap+21].

Remark 4.7. We illustrate the importance of the first condition with the following example of two distributions defined on $\mathbb{R}$:
Let p_u be the density of the distribution $\mathcal{N}(0, 1)$, $q^* = \mathbb{1}_{[0, \frac{1}{2}]} + \mathbb{1}_{[1, 1\frac{1}{2}]}$. A diffeomorphism T turning p_u into q^* has to satisfy $T(\text{supp } p_u) = \text{supp } q^*$. We see that supp $p_u = \mathbb{R}$ is connected, T is continuous and the image of a connected set under a continuous map is connected [Rud+64]. However, supp q^* is not connected and hence such a diffeomorphism T cannot exist.

Therefore we would like supp $q^* \cong$ supp p_u to be homeomorphic because the diffeomorphism T, as a homeomorphism, preserves the topology of supp p_u, i.e the number of connected components, number of "holes", etc.
If, conversely, supp $P_{Y^*} \ncong$ supp P_U, the diffeomorphism can have exploding Lipschitz constants, as is emphasized by the following Theorem 4.8. [Cor+20]. The intuition is that to counterbalance the topology mismatch, the diffeomorphism tries be discontinuous. This is exemplified by the two distributions above, where T would have to be discontinuous to accurately map supp P_U to supp P_{Y^*}.

Theorem 4.8. *Let Y^*, U be two random variables on $\mathbb{R}^d$ distributed according to P_{Y^*}, P_U, such that supp $P_{Y^*} \ncong$ supp P_U. Then, for any sequence of diffeomorphisms $T_n : \mathbb{R}^d \to \mathbb{R}^d$, we can have $T_n^* P_U \xrightarrow{\mathcal{D}} P_Y^*$ only if*

$$\lim_{n \to \infty} \max(Lip\ T_n, Lip\ T_n^{-1}) = \infty$$

Remark 4.9. This implies the following, when trying to fit a flow-based distribution T^*P_U to a target distribution P_{Y^*}, where supp $P_{Y^*} \not\cong$ supp P_U. During training, the flow-based distribution can only converge weakly to the target distribution, if either the Lipschitz constant of the forward transformation or the Lipschitz constant of the inverse transformation goes to ∞. Exploding Lipschitz constants are problematic, because the function can become non-numerically invertible, which can lead to numerical instabilities and generally unexpected function behavior. This is important to consider when modelling probability densities with complicated topologies.

4.3 Conditional Normalizing Flows

We would now like a way to construct flexible conditional probability densities $q(\theta|x)$ for inputs $(\theta, x) \in \mathbb{R}^d \times \mathbb{R}^m$ based on normalizing flows. For this, the transform T_x is conditioned on an input x that interacts with the other parameters ϕ of the transform. The chosen architecture of the transform will enforce that T_x is a diffeomorphism for all inputs x, such that the corresponding flow-based density $q(\theta|x)$ will always be a probability density for fixed x.
We can then write the conditional probability density function analogously to equation 4.2:

$$q(\theta|x) = p_u(T_x^{-1}(\theta))|\det J_{T_x}(T_x^{-1}(\theta))|^{-1} \tag{4.6}$$

where J_{T_x} is the $d \times d$-matrix of all partial derivatives of T_x.
Given a fixed conditioning variable x_0, we can also sample θ_0 from the flow-based conditional density analogous to equation (4.1):

$$\theta_0 = T_{x_0}(U) \text{ with } U \sim P_u \tag{4.7}$$

Training of Conditional Normalizing Flows

The above described training can be translated to fitting a conditional normalizing flow $q(\theta|x)$ to a conditional density $p(\theta|x)$ when only having access to samples of the joint distribution $p(\theta, x)$. Since we want to minimize some discrepancy between $p(\theta|x)$ and $q(\theta|x)$, we simply choose this discrepancy to be the KL-divergence between $p(\theta|x_0)$ and $q(\theta|x_0)$ averaged over different values of x_0.
More formally, we define

$$\mathbb{L}(\xi) := \mathbb{E}_{p(x)} D_{KL}(p(\theta|x)||q(\theta|x;\xi))$$

which is, by Prop. A.1

$$\begin{aligned} &= -\mathbb{E}_{p(x)}[\mathbb{E}_{p(\theta|x)}[\log q(\theta|x;\xi)] + \mathbb{H}_{p(\theta|x)}(\theta)] \\ &= -\mathbb{E}_{p(\theta,x)}[\log q(\theta|x;\xi)] - \mathbb{E}_{p(\theta)}[\mathbb{H}_{p(\theta|x)}(x)] \\ &= -\mathbb{E}_{p(\theta,x)}[\log q(\theta|x;\xi)] + const. \end{aligned}$$

as *const.* does not depend on ξ.
Once again, Monte-Carlo gives

$$\mathbb{L}(\xi) \approx -\frac{1}{N}\sum_{i=1}^{N} \log q(\theta_i|x_i;\xi) + const.$$

where $\{(\theta_i, x_i)\}_{i=1}^N$ is a sample from the joint distribution $p(\theta, x)$.

Analogously to equations 4.4, 4.5, we obtain an unbiased estimate of $\nabla_\xi \mathbb{L}(\xi)$ and can proceed with optimization analogous to 4.2.

4.3.1 Other Divergences

Alternatively, we may fit our model by minimizing the reverse KL-divergence. This is suitable when we have the ability to evaluate the target density $p(x)$, but not necessarily sample from it. Other divergences, such as f-divergences or Integral Probability metrics, could also be used, depending on the context. More detailed information on this can be found in [Pap+21].

4.3.2 Probabilities with Restricted Support

In general, the flow-based density $q(\theta|x_0)$ for fixed x_0 is a probability density defined on $\mathbb{R}^d$, that is supp $q(\theta|x_0) \subseteq \mathbb{R}^d$. In some cases we may prefer

$$\text{supp } q(\theta|x_0) \subseteq A \subsetneq \mathbb{R}^d.$$

For example in the context of estimating the posterior density with a flow-based density, we would like the support of $q(\theta|x_0)$ to be contained in the support of the prior, that is supp $q(\theta|x_0) \subseteq$ supp $p(\theta)$.
In these cases we set the normalized flow-based density to

$$q^*(\theta|x_0) := \frac{q(\theta|x_0)\mathbb{1}_{\theta\in A}}{C}.$$

where

$$C := \int_A q(\theta|x_0)\, d\theta \leq 1,$$

The normalizing factor C can be estimated via:

$$C \approx \frac{1}{K}\sum_{i=1}^{K} \mathbb{1}_{\theta_0^i \in A} \text{ with } \{\theta_0^i\}_{i=1}^K \sim q(\theta|x_0)$$

Likewise, we should then still sample according to 4.7, but accept θ_0 as a correct sample only if $\theta_0 \in A$. If acceptance rates are very low, one can incorporate iterative MCMC-sampling [Bro+11] techniques, that produce correlated samples, but are more computationally efficient. Low acceptance rates are also indicative of a bad approximation of the target density. Then

one should consider if the used network is expressive enough and possibly modify the used flow, e.g train a deeper network.

For our work we worked with a conditional normalizing flow, a neural spline flow [Dur+19], implemented in the SBI package [Tej+20]. In common implementations of normalizing flows, the base density is often kept constant, e.g as a standard normal distribution. The following section gives an overview of how to construct the parameterized diffeomorphism as an invertible neural network. We explain how one can include the conditioning variable to construct conditional normalizing flows (our contribution) (Section 4.4.1). In Section 4.4.1, we further give an overview of how the neural spline flows are constructed, which were relevant for our work. The specifics can be found in [Pap+21] and [Dur+19].

4.4 Constructing Flows

We first note that *diffeomorphisms* are composable: Given two diffeomorphisms T_1, T_2, their composition $T_1 \circ T_2$ is also a diffeomorphism. Its inverse and Jacobian determinant are given by

$$(T_2 \circ T_1)^{-1} = T_2^{-1} \circ T_1^{-1}$$

and

$$\det J_{T_2 \circ T_1}(u) = \det J_{T_2}(T_1(u)) \det J_{T_1}(u)$$

Therefore, we can construct a more complex transformation T by composing any number of simple transformations T_k as follows:

$$T = T_K \circ \ldots \circ T_1$$

We will now discuss ways to construct the simple transformations

$$T_\phi : \mathbb{R}^d \to \mathbb{R}^d, \mathbf{z} \mapsto T_\phi(\mathbf{z}) =: \mathbf{z}'$$

where $\mathbf{z}' = (z'_1, \ldots, z'_d)$, $\mathbf{z} = (z_1, \ldots, z_d)$, $\mathbf{z}_{<i} = (z_1, \ldots, z_d)$.
In practice, we implement either T_ϕ or T_ϕ^{-1}. Either way we must ensure that the model is invertible and has a tractable Jacobian determinant in order to sample from the flow (equation (4.1)) and evaluate the flow-based density (equation (4.2))

4.4.1 Autoregressive Flows

A widely used family of flows are autoregressive flows. These are used for example in [PPM17].

Definition 4.10. Autoregressive flows define T_ϕ as follows:
Given $\mathbf{z}$, we can compute $\mathbf{z}'$ iteratively via

$$z_i' = \tau(z_i; \mathbf{h_i}) \text{ where } \mathbf{h_i} = c_i(\mathbf{z}_{<i})$$

We call $c_i : \mathbb{R}^{i-1} \to \mathbb{R}^s$ the i-th *conditioner* and $\tau : \mathbb{R} \to \mathbb{R}$ the *transformer*. The transformer $\tau : \mathbb{R} \to \mathbb{R}$ is a strictly monotonic function of z_i that is parameterized by $\mathbf{h_i}$ and specifies how the flow acts on z_i to output z_i'.

Remark 4.11. Given $\mathbf{z}'$ we can iteratively construct an inverse $\mathbf{z}$ via

$$z_i = \tau^{-1}(z_i'; \mathbf{h_i}) \text{ where } \mathbf{h_i} = c_i(\mathbf{z}_{<i})$$

One can show by induction over i that each z_i is unique and hence the above construction is invertible.

Remark 4.12. The Jacobian of T_ϕ is triangular: Since $T_\phi(\mathbf{z})_i = z_i'$ does not depend on z_j for $j > i$, we have $\frac{\partial (T_\phi)_i}{\partial z_j}(\mathbf{z}) \equiv 0$ for $j > i$. Therefore, we can write the Jacobian of T_ϕ as follows:

$$J_{T_\phi}(\mathbf{z}) = \begin{pmatrix} \frac{\partial (T_\phi)_1}{\partial z_1}(\mathbf{z}) & & \mathbf{0} \\ & \ddots & \\ L(\mathbf{z}) & & \frac{\partial (T_\phi)_d}{\partial z_d}(\mathbf{z}) \end{pmatrix}$$

where $L(\mathbf{z})$ is some function of $\mathbf{z}$ that is irrelevant for the computation of the determinant. Since the determinant of any triangular matrix is equal to the product of its diagonal elements, the absolute determinant of $J_{T_\phi}(\mathbf{z})$ can be calculated as follows:

$$|\det J_{T_\phi}(\mathbf{z})| = |\prod_{i=1}^{d} \frac{\partial (T_\phi)_i}{\partial z_i}| = |\prod_{i=1}^{d} \frac{\partial \tau(z_i; \mathbf{h_i})}{\partial z_i}|$$

Remark 4.13. We usually implement $c_i : \mathbb{R}^{i-1} \to \mathbb{R}^s$ as a neural network. If we are working with a conditional normalizing flow, we choose c_i to be a function of $\mathbf{z}_{<i}$ and the conditioning variable x, such that $\mathbf{h_i} = c_i(\mathbf{z}_{<i}, x)$. We could choose c_i to be a function of additional quantities, such as those described in Section 3.2.

By definition 4.10, implementing an autoregressive flow comes down to implementing the transformer and the conditioner.

Implementing the Transformer

Examples to implement the transformer:

- **Affine transformers:** $\tau(z_i; h_i) = \alpha_i z_i + \beta_i$ where $\mathbf{h_i} = \alpha_i, \beta_i$
  They are used in e.g masked autoregressive flows [PPM17].

- **Spline-based transformers:** implement the transformer as a monotonic spline, namely a piecewise function consisting of K segments where each segment is is a simple function that is easy to invert. For neural spline flows [Dur+19], rational-quadratic splines [GD82] were used.
- **Other:** Integration-based transformers [WL19]

Implementing the conditioner

A first example is masked conditioners as in [PPM17]. The i-th conditioner as a fully connected neural net, mapping $c : z \mapsto \mathbf{h_i}$ that masks the connections from $z_{\geq i}$ to $\mathbf{h_i}$.
This enforces the necessary condition that c is only a function of $z_{<i}$ A second example is given with coupling layers. Neural spline flows use coupling layers, which allow faster inverse computation resulting in significant training speedups.

Definition 4.14. *Coupling layers* are a special type of autoregressive flow that define T_ϕ as follows: Let $0 < k \leq d$ be fixed and $(\mathbf{h_1}, ..., \mathbf{h_{k-1}})$ constant. Then $\tau(z_i, \mathbf{h_i}), i = 1, \ldots, k-1$ is predefined, e.g as the identity function[7]. Given $\mathbf{z}$, we compute z' via

$$z'_i = \tau(z_i, \mathbf{h_i}) \text{ for } i < k \text{ with constant } (\mathbf{h_1}, \ldots, \mathbf{h_{k-1}})$$

$$z'_i = \tau(z_i, \mathbf{h_i}) \text{ for } i \geq k \text{ with } (\mathbf{h_k}, \ldots, \mathbf{h_d}) = C(\mathbf{z}_{<k})$$

where $C : \mathbb{R}^{k-1} \to \mathbb{R}^{(d-k+1)\times s}$ realizes $d - k + 1$ conditioners at once.

Remark 4.15. This defines indeed an autoregressive flow since the conditioner computing $(\mathbf{h_k}, \ldots, \mathbf{h_d})$ only depends on $\mathbf{z}_{<k}$.

Remark 4.16. The inverse transformation is given by:

$$\mathbf{z}_{<k} = \tau^{-1}(z'_i, \mathbf{h_i}) \text{ for } i < k \text{ with constant } (\mathbf{h_1}, \ldots, \mathbf{h_{k-1}})$$

$$z_i = \tau^{-1}(z'_i, \mathbf{h_i}) \text{ for } i \geq k \text{ with } (\mathbf{h_k}, ..., \mathbf{h_d}) = C(\mathbf{z}_{<k})$$

Remark 4.17. Once again, for conditional normalizing flows, the conditioner C can take a conditioning variable x as additional input.

The main advantage of coupling layers is that it allows faster computation of the inverse $\mathbf{z}$ given $\mathbf{z}'$. In the inverse computation for general autoregressive flows, all $\mathbf{z}_{<i}$ need to have been computed before z_i, so that $z_{<i}$ is available to the conditioner for computing $\mathbf{h_i}$. Therefore $\mathbf{z}$ can only be computed sequentially in $\mathbb{O}(d)$. Conversely, for coupling layers only $\mathbf{z}_{<k}$ needs to be computed before computing $\mathbf{z}_{\geq k}$. However, $(z_1, \ldots, z_{k-1})$ can be

[7] This is the case in the sbi implementation of Neural Spline Flows

computed in parallel as well as $(z_k, \ldots, z_d)$ in a second step. In consequence, $\mathbf{z}$ can be computed in $\mathbb{O}(2) = \mathbb{O}(1)$.
A faster inverse computation leads to significant speedups when evaluating the flow-based density (see equation (4.2)), which also results in training speed-ups.

In Practice

The following illustrates how coupling layers are used to construct and implement conditional normalizing flows. Figure 2 gives an overview of the corresponding transformation T_x. To obtain the total transformation, any number of these simple transformation are composed while alternating to which components of θ we apply the identity transformation.

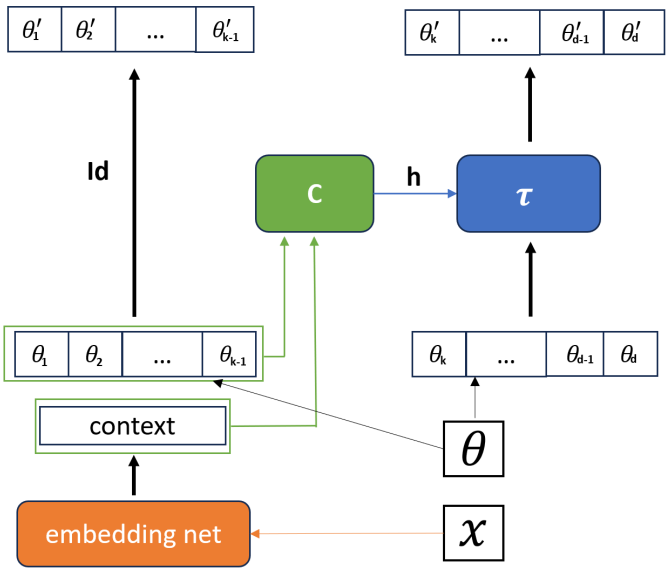

Figure 2: Example of a transformation T_x corresponding to a conditional normalizing flow

Given a fixed conditioning variable x and vector θ, the transformation transforms θ in the following way:
The basic architecture is a coupling layer, such that the first $k-1$ components of θ remain unchanged. The conditioner C, a neural net, receives the first $k-1$ components of θ as well as the context as input. The context is the output of an embedding net (neural net) applied on x. This conditioner outputs a vector $\mathbf{h} = (\mathbf{h_k}, \ldots, \mathbf{h_d})$, where each $\mathbf{h_i}$ defines a bijective function $\tau(\,\cdot\,, \mathbf{h_i}) : \mathbb{R} \to \mathbb{R}$, which is applied on θ_i for $i = k, \ldots, d$ to produce the final output θ'_i.

In the neural spline flow used for the experiment (Section 6) each $\mathbf{h_i}$ defines a bijective rational quadratic spline $\tau(\,\cdot\,, \mathbf{h_i}) : \mathbb{R} \to \mathbb{R}$, which is applied on θ_i for $i = k, \ldots, d$ to produce the final output θ'_i. In the used implementation [Tej+20], k is fixed to $\frac{d}{2}$ and 5 of these simple transformations are composed to obtain the total transformation.

5 Stochastic Biochemical Reaction Networks

This introduction is based on [WBS19].
Consider a domain, for example, a cell nucleus, that contains a number of chemical species. Given a set of chemical formulae that determine how the chemical species interact, we would like a framework to simulate the system. Gillespie presents such a framework in [Gil92], [Gil77].

5.1 A Computational Definition

The population count for a chemical species is a non-negative integer called its *copy number*, $x_i(t)$ denotes the copy number of species $\mathrm{x_i}$ at time t. All the species population counts at a given time t are then captured by the *state vector*, a vector of copy numbers, $X(t) = [x_1(t), ..., x_N(t)]$. We can understand a reaction as a change caused to the state vector, because a chemical reaction transforms molecules of species $x_{r_1}, \dots x_{r_k}$ (reactants) to molecules of species $x_{p_1}, \dots x_{p_l}$ (products). We call this net change the *stoichiometric vector* of the reaction. If reaction j occurs, then a new state is obtained by adding its stoichiometric vector ν_j to the previous state vector, namely

$$X(t) = X(t^-) + \nu_j \tag{5.1}$$

where t^- denotes the time immediately preceding the reaction event.

Example 5.1. Let us consider a simple biochemical reaction network consisting of species $\mathrm{x_1}, \mathrm{x_2}, \mathrm{x_3}$. The species can only react in the following way:

$$\mathrm{x_1} + \mathrm{x_2} \rightarrow 2\mathrm{x_3}$$

In words, the reaction equation reads: one $\mathrm{x_1}$ molecule and one $\mathrm{x_2}$ molecule react to form $2\mathrm{x_3}$ molecules".
The stoichiometric vector of the reaction is $\nu = [-1, -1, 2]$. Thus, given a state vector $X(t^-) = [3, 4, 5]$ at time t^-, the new state vector after a reaction event at time t is given by

$$X(t) = X(t^-) + \nu = [2, 3, 7]$$

5.2 Stochastic Law of Mass Action

An important concept in simulating biochemical reaction networks is the *stochastic law of mass action*. It states that the reaction rate is proportional to the probability of a collision of the reactants, which is proportional to the number of possible reactant combinations, namely the product of the copy numbers of the reactants. Informally,

$$\begin{aligned} & P(\text{Reaction j occurs in } [t, t+dt)) \\ \propto & \text{ no. of possible reactant combinations} \times dt \\ = & \text{ constant} \times \text{no. of possible reactant combinations} \times dt \\ = & a_j(X(t))dt, \end{aligned}$$

where we call $a_j(X(t))$ the propensity function of reaction j.
See Ref. [Gil92] for a more detailed derivation. Here the positive constant is known as the *kinetic rate* parameter. The kinetic rate parameter can be understood as the speed at which a chemical reaction takes place. In Example 5.1, given a kinetic rate k_1 the propensity function of the reaction is $a_1(X(t^-)) = k_1 \cdot x_1(t^-) \cdot x_2(t^-) = k_1 \cdot 3 \cdot 4$.

5.3 Simulation

Given an initial state vector $X(0)$, and a system of m reactions, each with a kinetic rate k_i, we would like to simulate the evolution of the system. There are several algorithms, such as the stochastic simulation algorithm by Gillespie [Gil77] or the modified next reaction method [And07], that can generate exact simulations. We discuss both these algorithms in this section. For approximate algorithms, that do not generate exact simulations, but are more computationally efficient, we refer the reader to the tau-leaping method [Gil01] and Ref. [WBS19].

Stochastic Simulation Algorithm

Given a sufficiently small time interval, $[t, t+dt)$, we have that the probability of multiple reactions occurring in this interval is zero. In such a case, the reactions are disjoint events. Therefore, the probability of any reaction event occurring in $[t, t+dt)$ is the sum of the individual event probabilities,

$$\begin{aligned} & P(\text{Any Reaction occurs in } [t, t+dt)) \\ = & P(\text{Reaction 1 occurs in } [t, t+dt)) + \ldots + P(\text{Reaction M occurs in } [t, t+dt)) \\ = & a_0(X(t))dt, \end{aligned}$$

where $a_0(X(t)) = a_1(X(t)) + \ldots + a_M(X(t))$ is the total reaction propensity function.

The time difference between reactions Δt, can be considered as a random variable. Gillespie [Gil77] demonstrates that it is distributed according to an exponential distribution with rate $a_0(X(t))$, that is, $\Delta t \sim \exp(a_0(X(t)))$. This allows us to compute the time of the next reaction $s \in [t, t + dt)$.
We then just have to
(i) randomly choose a reaction j, where each reaction i is chosen with probability $a_i(X(s))/a_0(X(s))$
(ii) update the state vector according to the corresponding stoichiometric vector ν_j.
We arrive at Gillespie's Stochastic Simulation Algorithm [Gil77]:

Algorithm 2 Stochastic Simulation Algorithm

Input: Initial state vector X_0, simulation time T
Output: Final state vector X, final time t
Initialization: $t \leftarrow 0$, $X \leftarrow X_0$
while $t \leq T$ **do**
  Calculate propensities: $a_1(X), \ldots, a_M(X)$ and
  $a_0(X) \leftarrow a_1(X) + \ldots + a_M(X)$
  sample time to next reaction: $\Delta t \sim \exp(a_0(X))$
  if $t + \Delta t > T$ **then**
    Terminate simulation
  Randomly select j from $\{1, \ldots, M\}$ with probabilities
  $P(j = 1) = \frac{a_1(X)}{a_0(X)}, \ldots, P(j = M) = \frac{a_M(X)}{a_0(X)}$
  Update state: $X \leftarrow X + \nu_j$
  Update time: $t \leftarrow t + \Delta t$

Modified Next Reaction Method

The main advantage of the next reaction method [GB00] and of the modified next reaction method [And07] is that the next reaction time of each reaction is tracked separately. The next reaction to occur is the one with the smallest next reaction time, therefore no random selection of reaction events is required. Hence, we need to draw 2 random numbers for each reaction for the stochastic simulation algorithm and just 1 random number for the modified next reaction method, which is why this method is computationally superior.

Algorithm 3 Modified Next Reaction Method

Input: Initial state vector X_0, simulation time T

Output: Final state vector X, final time t

Initialization: $t \leftarrow 0$, $X \leftarrow X_0$ and scaled times $t_1 \leftarrow 0, t_2 \leftarrow 0, \dots t_M \leftarrow 0$

Generate M first reaction times $s_1, \dots, s_M \sim \exp(1)$

while $t \leq T$ **do**

  Calculate propensities: $a_1(X), \dots, a_M(X)$

  Rescale time to next reaction:

  if $a_j(X) = 0$ **then**

    $\Delta t_j = \infty$

  else

    $\Delta t_j \leftarrow (s_j - t_j)/a_j(X)$

  for $j = 1, \dots, M$

  Choose reaction k, such that $\Delta t_k = \min\{\Delta t_1, \dots, \Delta t_M\}$

  if $t + \Delta t_k > T$ **then**

    Terminate simulation

  Update rescaled times, $t_j \leftarrow t_j + a_j(X)\Delta t_k$ for $j = 1, \dots, M$, update state: $X \leftarrow X + \nu_k$ and global time $t \leftarrow t + \Delta t_k$

  Generate scaled next reaction time for reaction k, $\Delta s_k \sim \exp(1)$

  Update next scaled reaction time $s_k = s_k + \Delta s_k$

Remark 5.2. The main idea of the algorithm is to track an internal time t_j for each reaction separately, which increases dependent on the propensity $a_j(X(t))$. If s reaction events happened at global times $t^1, \dots, t^s$ and $t^0 = 0$, then t_j is given by

$$t_j(t^s) = \sum_{i=0}^{s} a_j(X(t^s)(t^{s+1} - t^s) \approx \int_0^{t^s} a_j(X(t))dt$$

Anderson argues that that the number of times the j-th reaction has taken place up to time t_s can be realized as a unit rate poisson process with internal time $t_j(t)$. We recall that the time between events for a unit rate poisson process is exponentially distributed, also with a unit rate. With these notions, $\Delta t_j \leftarrow (s_j - t_j)/a_j(X)$ is the difference between the internal time t_j and the internal reaction time s_j rescaled by the current rate of increase of the internal time, namely $a_j(X(t))$. A reaction k is then chosen, when this difference Δt_k is the smallest. A formal derivation of the algorithm is found in [And07].

For this work, I implemented code to simulate arbitrary biochemical reaction networks with the modified next reaction method extending the implementation provided in the pyABC [KRH18] tutorial on Markov jump processes. I also ensured that the simulations could run on multiple CPU cores in parallel by using the functionality provided in the `joblib` library [Job20].

6 Experiment

6.1 Introduction

Recent advances in probabilistic modelling have led to numerous simulation-based inference algorithms which do not require numerical evaluation of likelihoods as the ones described in Section 3.2. Recent studies compare and analyse the performance of these algorithms [Lue+21] as well as the validity of the obtained posteriors [Her+21] for different types of simulators. These benchmarks lack performance analysis on simulators with a high parameter dimension, as all benchmark problems had 10 or fewer parameters. In this experiment, we aim to contribute to filling this gap. We explore how well NPE (Algorithm 1), one type of SBI algorithm, works when estimating high-dimensional posterior densities. We use a neural spline flow [Dur+19] to infer the approximate posteriors of different simulators (Section 6.2). For this, we define multiple simulators within a stochastic biochemical reaction network (Section 6.2), each with a different number of parameters. For an introduction to stochastic biochemical reaction networks, including their definition and algorithms to simulate them, please return to Section 5, for explanations regarding normalizing flows and neural spline flows to Section 4. We define different evaluation and validation procedures in Section 6.3 and report results for different simulation budgets in Section 6.4. Similarly to [Her+21], we find that NPE generally produces slightly overconfident results. This can be reduced by using higher simulation budgets (≥ 10000). In addition to that, our experiments indicate that a higher parameter dimension does not lead to increasingly overconfident results if we are working with larger simulation budgets (≥ 10000). On the contrary, when working with lower simulation budgets (1000), we provide evidence that NPE tends to be increasingly overconfident in higher parameter dimensions.

6.2 Experimental Setup

This section explains the setup of our experiment, including the definition of the considered simulators and the density estimators used for inference. We further discuss the CO_2 emissions emitted during model training.

Biochemical Reaction Network with Increasing Parameter Dimension

For $n \in \mathbb{N}$, we consider the following biochemical reaction network :
The network consists of $N := n + \binom{n}{2}$ different species

$$\{x_1, \ldots, x_n\} \cup \{x_i x_j, 1 \leq i < j \leq n\}.$$

The dynamics of the species population counts are governed by reactions of the following forms: Firstly, $\binom{n}{2}$ equilibrium reactions R_{ij}

$$\mathrm{x_i} + \mathrm{x_j} \underset{k'_{ij}}{\overset{k_{ij}}{\rightleftharpoons}} \mathrm{x_i x_j} \tag{6.1}$$

and secondly, decay reactions D_i

$$\mathrm{x_i} \xrightarrow{\mathrm{d_i}} \emptyset \tag{6.2}$$

We have $2 \cdot \binom{n}{2} + n$ estimable parameters, let them be denoted by θ. We work with this network for $n = 1, \ldots, 9$. The important property is that the parameter dimension as well as the number of species increases in n, as can be seen in Table 2.

category / n	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
number of parameters	4	9	16	25	36	49	64	81
number of species N	3	6	10	15	21	28	36	45

Table 2: Information on the experiments

Simulator

We set the initial species population counts to 100 for the reactants x_i and to 0 for the products $x_i x_j$.

We simulate the evolution of the network as a Markov Jump Process with the Modified Next Reaction Method [And07] for 0.1 seconds. Some notes on the implementation of the simulator are found in Section5.3. We observe the species population counts at 200 evenly spaced time points. In three different experiments, we observe different species:

- **Exp1:** Observation of all species, s.t. the output of our simulator has dimension $200 \times N$, see Table 2
- **Exp2:** Observation of only the reactants, s.t. the output of our simulator has dimension $200 \times n$
- **Exp3:** Observation of only the species x_1, x_2, $x_1 x_2$ s.t. the output of our simulator has dimension 200×3 for all n.

An example of a simulator output for **Exp1**, $n = 3$, $N = 6$ can be found in Figure 12.

Inference

For each of these simulators, we infer approximate posterior densities $q(\theta|x)$ (NPE, see Algorithm 1) by using a neural spline flow ([Dur+19], Section 4.4.1). We iterate this with simulation budgets of 1000, 10000 and 100000 simulations.
The following entities are fixed throughout all experiments:

- **Prior:** We fix a uniform prior on $[0,1]^{|\theta|}$
- **Neural Network:** Architecture as implemented in the SBI package [Tej+20], see 4.4.1.
- **Embedding Net** (to embed the observations x_0)**:** Fully connected MLP with 3 hidden layers, 70 hidden units and output dimension 20

An overview of fixed training hyperparameters can be found in Table 3.

batch size	50
validation fraction	10%
patience [8]	5
optimizer	Adam [KB15]
validation fraction	10%

Table 3: Training hyperparameters

For each $exp \in \{\textbf{Exp1}, \textbf{Exp2}, \textbf{Exp3}\}$, $n \in \{2, 3, \ldots, 9\}$, $simulations \in \{1000, 10000, 100000\}$, we obtain a posterior density estimator $q(\theta|x)$. The chosen architecture allows us to evaluate normalized densities[9] $q(\theta|x)$ for arbitrary inputs (θ, x) and sample from $q(\theta|x_0)$ for some fixed observation x_0.

CO_2 emissions of model training

For Exp2 and Exp3, we used the CodeCarbon tool to track the energy consumption while training models. Table 4 shows the average energy consumption of each model, the corresponding approximate CO_2 emissions in Germany [Sta24], and the amount of beef from a german supermarket one would have to eat to have the same carbon impact [RGW20] for Exp2 and Exp3. Note that this does not take into account the amount CO_2-eqs emitted when producing the computing hardware. Since we trained more than 250 models (for Exp1 and Exp2 we trained each density estimator 5 times) and also had to evaluate my models on sufficiently big datasets, the work on my thesis had a non-negligible impact. Compared to the 8.33 tons of CO_2-eqs emitted per capita in Germany 2023 [Sta23], the impact of this work is negligible.

[9] probability densities on supp $p(\theta)$

	Exp2	**Exp3**
energy consumed (kWh)	0.048	0.052
emissions (g CO_2-eqs)	18.11	19.75
beef (g)	1.34	1.46

Table 4: Average carbon emission impact of training one model.

6.3 Evaluation Procedures

This section explains the different procedures used to validate and evaluate the estimated posteriors. This entails posterior predictive checks (Section 16), the first metric $-\mathbb{E}[\log q(\theta_0|x_0)]$ (Section 6.3.2), the second metric $\mathbb{E}_{q(\theta|x_0)}[||\theta - \theta_0||]$ (Section 6.3.3), a validation scheme based on expected coverage probabilities of $(1-\alpha)$-credible regions (Section 6.3.4) and Simulation-based Calibration (Section 6.3.4).

6.3.1 Posterior Predictive Checks

To validate our results, we conducted posterior predictive checks. Given (simulated) observation x_0, we can consider the *posterior predictive distribution*, that is

$$p(\tilde{x_0}|x_0) = \int p(\tilde{x_0}|\theta)q(\theta|x_0)d\theta$$

Note that we can sample from the posterior predictive distribution by sampling $\tilde{\theta_0} \sim q(\theta|x_0)$ and then simulate $\tilde{x_0} \sim p(x|\tilde{\theta_0})$. There exist statistical tests that compute a test statistic on samples of the posterior predictive distribution. These tests then give p-values of seeing such a test statistic under the assumption that the null hypothesis $H_0 : p(\theta|x_0) = q(\theta|x_0)$ is true. This works well for one-dimensional simulator outputs where a test statistic with known distribution is easier to define [BVH00]. It is difficult to extend this approach to high dimensional observations, which is why we restrict ourselves to visually comparing samples of the posterior predictive distribution with the true observation x_0. For fixed observation x_0 and dataset of the posterior predictive distribution $D := \{(\tilde{x_0}^i)\}_{i=1}^{2000}$, we compute 80%, 95%, 99% confidence intervals in each dimension of the output space on D and compare these to the true value of x_0. We iterate this for each density estimator with 10 different values for x_0. This should not be understood as a metric, but rather as a sanity check that informs us if our posteriors are fundamentally wrong.

6.3.2 First Metric: Negative Expected Log Probability

As a first evaluation metric we would like to consider the same objective function that is used to train our posterior density estimators.

$$\mathbb{L}(q) := -\mathbb{E}_{p(\theta,x)}[\log q(\theta|x)] \tag{6.3}$$

By Proposition A.1 it also relates directly to an expected KL-divergence between estimated and true posterior

$$\begin{aligned}
&- \mathbb{E}_{p(\theta,x)}[\log q(\theta|x)] \\
&= -\mathbb{E}_{p(x)}\mathbb{E}_{p(\theta|x)}[\log q(\theta|x)] \\
&= \mathbb{E}_{p(x)} D_{KL}(p(\theta|x)||q(\theta||x)) + \mathbb{E}_{p(x)}[\mathbb{H}_{p(\theta|x)}(\theta)] \\
&= \mathbb{E}_{p(x)} D_{KL}(p(\theta|x)||q(\theta||x)) + const.
\end{aligned}$$

We can evaluate this metric on a test dataset of N samples

$$\{(\theta_0^i, x_0^i)\}_{i=1}^N \text{ with } \theta_0^i \sim p(\theta) \text{ and } x_0^i \sim p(x|\theta_0^i)$$

via

$$\mathbb{L}(q) \approx \frac{1}{N} \sum_{i=1}^{N} \log q(\theta_0^i|x_0^i)$$

Intuitively, this metric should be low when areas of high density with respect to p also have high density with respect to q and high when areas of high density with respect to p have low density with respect to q. Therefore, this metric could be indicative of the average similarity between p and q.

Evaluation is Computationally Expensive:

To evaluate $\log q(\theta|x_0)$, we have to compute the normalizing constant by sampling K times from the underlying unnormalized density, see 4.3.2. To compute all the normalizing constants, we require $K \times N$ forward passes of our neural network. Furthermore, we also want to evaluate this metric on several models. To finish in a reasonable amount of time, we fixed $K = 100$ and computed $\mathbb{L}(q)$ according to the following procedure with $\epsilon = 0.1$.

Algorithm 4 Convergence Procedure

Input: A dataset $D = \{(\theta_0^i, x_0^i)\}_{i=1}^{10000} \sim p(\theta, x)$, a threshold ϵ
Output: Estimated negative log probabilities $\mathbb{L}(q)$
Initialization: $N \leftarrow 10000$, $n \leftarrow 1000$, $\mathbb{L}(q)_p \leftarrow -\infty$
while $n \leq N$ **do**
  Compute $\mathbb{L}(q)$ on the first n samples of D
  if $|\mathbb{L}(q) - \mathbb{L}(q)_p| < \epsilon$ **then**
    return L
  $\mathbb{L}(q)_p \leftarrow \mathbb{L}(q)$
  $n \leftarrow n + 1000$

6.3.3 Second Metric: Average Distance to True Samples

We computed

$$\begin{aligned} M_2(q) &= \mathbb{E}_{(\theta_0,x_0)\sim p(\theta,x),\tilde{\theta_0}\sim q(\theta|x_0)}[||\theta_0 - \tilde{\theta_0}||_2] \\ &= \mathbb{E}_{x_0\sim p(x)}\mathbb{E}_{\theta_0\sim p(\theta|x_0),\tilde{\theta_0}\sim q(\theta|x_0)}[||\theta_0 - \tilde{\theta_0}||_2] \\ &\approx \frac{1}{N}\sum_{i=1}^{N}\frac{1}{K}\sum_{k=1}^{K}||\theta_0^i - \tilde{\theta_0}^{i,k}||_2 \end{aligned}$$

on an evaluation dataset of

$$\{(\theta_0^i, x_0^i)\}_{i=1}^N = 10000, \{(\tilde{\theta_0}^{i,k})\}_{k=1}^K \text{ with } (\theta_0^i, x_0^i) \sim p(\theta, x) \text{ and } \tilde{\theta_0}^{i,k} \sim q(\theta|x_0^i).$$

with $K = 1000$ and analogous to the convergence procedure 4 to be more computationally efficient. As an average distance between estimated and true parameter, this metric could be indicative of the quality of our posterior. However, we understand that, as a euclidean distance, this metric will naturally increase with the parameter dimension. To improve our understanding of high-dimensional probability densities, we are still interested in the scaling behavior of this metric.

6.3.4 Validation

The following section, namely Definitions 6.1, 6.2 and Proposition 6.3, formalize the following intuitive idea. If our posteriors are correct, we should expect the following: If we sample $\theta_0 \sim p(\theta)$ from the prior, generate output $x_0 \sim p(x|\theta_0)$ and then compute any region which contains $1-\alpha$ of the mass of the estimated posterior $q(\theta|x_0)$ with respect to one parameter θ_j, the probability of θ_{0j} to be located in that region should be exactly $1-\alpha$. If it is lower, we are overconfident when estimating parameter θ_j, because we locate more mass to regions, where, on average, the true parameters are not located. If it is higher, we are underconfident, because our $(1-\alpha)$- credible regions are wider than they should be.

Mathematical Formalism:

The formalism is based on the formalism in [Her+21]. Our contribution consists in translating the ideas in [Her+21] to covering single parameters.

Definition 6.1. Given a conditional density $p(\theta|x)$, fixed observation x_0 and confidence level $1-\alpha$, a subset $A := A_{\alpha,j,p,x_0} \subseteq \mathbb{R}$ is called a $(1-\alpha)$-*credible region* for parameter θ_j , if

$$\int_A \int_{\mathbb{R}^{n-1}} p(\theta|x_0)\, d(\theta_1, \ldots, \theta_{j-1}, \theta_{j+1}, \ldots, \theta_n)\, d\theta_j = 1-\alpha.$$

Definition 6.2. We define the *expected coverage probability* of a $(1-\alpha)$-credible region of θ_j for $q(\theta|x)$ as

$$P_{\alpha,j,q} = \mathbb{E}_{p(\theta,x)}[\mathbb{1}(\theta \in A_{\alpha,j,q,x})] = \mathbb{E}_{x_0 \sim p(x), \theta_0 \sim p(\theta|x_0)}[\mathbb{1}(\theta_0 \in A_{\alpha,j,q,x_0})]$$

We will sometimes write *expected coverage* for simplicity reasons.

Proposition 6.3. *If our estimated density is equal to the true posterior density, namely $p(\theta|x) = q(\theta|x)$, we have that the expected coverage probability $P_{\alpha,j,q} = 1-\alpha$.*

Proof.

$$\begin{aligned} P_{\alpha,j,q} &= \mathbb{E}_{p(\theta,x)}[\mathbb{1}(\theta \in A_{\alpha,j,q,x})] \\ &= \mathbb{E}_{p(x)}\mathbb{E}_{p(\theta|x)}[\mathbb{1}(\theta \in A_{\alpha,j,p,x})] \\ &= \mathbb{E}_{p(x)}[1-\alpha] \\ &= 1-\alpha \end{aligned}$$

□

Remark 6.4. Estimated values of the expected coverages close to the true value of $1-\alpha$ are a necessary condition of a healthy posterior. However, the deviation from the true value $1-\alpha$ should not be understood as a metric. Computing the expected coverage by using the prior as an estimated posterior would achieve perfect results. However, the inference would clearly be wrong.

We now describe how the expected coverage $P_{\alpha,j,q}$ of a $(1-\alpha)$-credible region for parameter θ_j is estimated. First, we have to be able to compute $(1-\alpha)$-credible regions.

Computation of credible regions

If we can sample from the estimated posterior density q, we can construct $D = \{{\theta_j}^i\}_{i=1}^K$ from a sampled Dataset $\{\theta^i\}_{i=1}^K \sim q(\theta|x_0)$. We now choose an approximate $(1-\alpha)$-credible region $A := [a,b]$ by setting

$$a = \min\{\gamma | 100 \cdot \frac{\alpha}{2}\% \text{ of the values in } D \text{ are smaller than } \gamma\}$$

and

$$b = \max\{\gamma | 100 \cdot \frac{\alpha}{2}\% \text{ of the values in } D \text{ are bigger than } \gamma\},$$

namely we compute the $\frac{\alpha}{2}$-quantile (a) and the $1-\frac{\alpha}{2}$ quantile (b) of D. See torch.quantiles [Pas+19] for details.

Estimation of $P_{\alpha,j,q}$

We can estimate the expected coverage probability given a dataset from the joint distribution

$$\{(\theta_0^i, x_0^i)\}_{i=1}^N \sim p(\theta, x)$$

with the estimator

$$E_N = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(\theta_0^i \in A_{\alpha,j,q,x_0^i})$$

If we assume $Y_i := \mathbb{1}(\theta_0^i \in A_{\alpha,j,q,x_0^i})$ are independent and identically distributed Lebesgue integrable random variables with mean $\mu = P_{\alpha,j,q}$ and finite variance σ^2, we have that

$$E_N \to \mu \text{ as } N \to \infty$$

by the Law of large numbers.
By the Central Limit Theorem, we have that for $N \geq 30$

$$E_N \approx \mathcal{N}(\mu, \frac{S}{\sqrt{N}})$$

where S is the sample standard deviation of $\{Y_i\}_{i=1}^N$.
Therefore, we can say with 95% confidence that

$$P_{\alpha,j,q} \in [E_N - 1.96\frac{S}{\sqrt{N}}, E_N + 1.96\frac{S}{\sqrt{N}}]$$

and with 68% confidence that

$$P_{\alpha,j,q} \in [E_N - 1.0\frac{S}{\sqrt{N}}, E_N + 1.0\frac{S}{\sqrt{N}}]$$

If q is the correct posterior, we would expect $P_{\alpha,j,q} = 1 - \alpha$ by Proposition 6.3. Instead we check if the true value $1 - \alpha$ is contained in the 68% or 95% - confidence interval for the expected coverage respectively.
For our experiments, we computed 68% and 95% confidence intervals for the expected coverage probability of a $(1-\alpha)$-credible region for each parameter θ_j for $\alpha \in \{0.2, 0.05, 0.01\}$.

6.3.5 Simulation-based Calibration

We briefly summarize the most important ideas from [Tal+18]. Simulation-based calibration (SBC) provides a general way to validate a Bayesian inference workflow. Central is the following observation.

Proposition 6.5. *Integrating the exact posteriors over the Bayesian joint distribution returns the prior distribution:*

$$p(\theta) = \int p(\theta|\tilde{x})p(\tilde{x}|\tilde{\theta}))p(\tilde{\theta})\, d\tilde{\theta}\, d\tilde{x}$$

Proof. By marginalizing over $\tilde{x}$ and then over $\tilde{\theta}$, we have

$$\begin{aligned} p(\theta) &= \int p(\theta|\tilde{x})p(\tilde{x})\, d\tilde{x} \\ &= \int p(\theta|\tilde{x})p(\tilde{\theta}|\tilde{x})p(\tilde{x})\, d\tilde{x}\, d\tilde{\theta} \\ &= \int p(\theta|\tilde{x})p(\tilde{x}|\tilde{\theta}))p(\tilde{\theta})\, d\tilde{\theta} d\tilde{x} \end{aligned}$$

□

Therefore, any discrepancy between the data averaged posterior and the prior indicates a problem with our posterior estimation.

Remark 6.6. Considering the following sequence

$$\tilde{\theta_0} \sim p(\theta), \tilde{x} \sim p(x|\tilde{\theta}),\ \{\theta^i\}_{i=1}^L \sim p(\theta|\tilde{x})$$

Proposition 6.5 implies that the prior sample $\tilde{\theta_0}$ and an exact posterior sample $\{\theta^i\}_{i=1}^L$ will be distributed according to the same distribution.

Definition 6.7. Let f be any one-dimensional variable $f : \Theta \to \mathbb{R}$, where Θ is the parameter space. With the notions of 6.6, we define the *rank statistic* of the prior sample relative to the posterior sample as

$$r(\{f(\theta^1), \ldots, f(\theta^L)\}, f(\tilde{\theta})) = \sum_{i=1}^{L} \mathbb{1}_{f(\theta^i)<f(\tilde{\theta})}$$

[Tal+18] proves the following theorem.

Theorem 6.8. *Let $\tilde{\theta} \sim p(\theta), \tilde{x} \sim p(x|\tilde{\theta})$ and $\{\theta^i\}_{i=1}^L \sim p(\theta|\tilde{x})$ for any joint distribution $p(\theta, x)$. The rank statistic of any one-dimensional random variable over θ is uniformly distributed over the integers $[0, L]$.*

Remark 6.9. This implies the following: Given our posterior estimator $q(\theta|x)$, a dataset from the joint distribution $\{(\tilde{\theta}^i, \tilde{x}^i\}_{i=1}^N$ and fixed $f : \Theta \to \mathbb{R}$, we can compute N rank statistics, let us simply refer to them as N *ranks*. If our posteriors are correct these ranks should be distributed approximately according to a uniform distribution. This can then be inspected visually with a histogram plot or cumulative distribution plot. Inspecting it as described in [Tal+18] can help detect systemic biases in the posterior, such as a shifted posterior mean or an overconfident posterior.

Assessing Overconfidence

An algorithm that computes posteriors that are, on average, overconfident with respect to or narrower than the true posterior produces a histogram of rank statistics with a characteristic ∪-shape, see Figure 3. It is clear that an estimated posterior that is on average narrower than the true posterior produces a data-averaged posterior that is also narrower than the prior. Since the data averaged posterior is narrow, on average $f(\theta^i)$ will have similar values for different i such that it is likely that on average $f(\theta)$ will be bigger or smaller than most $f(\theta^i)$ resulting in higher frequencies of very high and very low ranks.

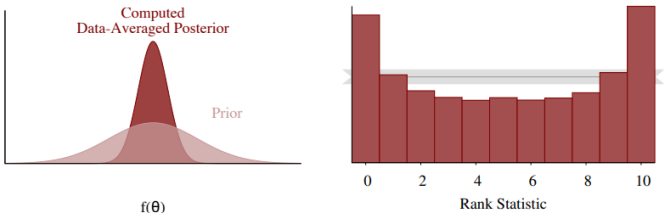

Figure 3: ∪-shaped ranks and corresponding data averaged posterior, taken from [Tal+18]

One can also use the Kolmogorov-Smirnov test [BZ14], which gives p-values on the maximum difference of the empirical cumulative distribution function of the ranks and the cumulative distribution function of a uniform distribution under the null-hypothesis that the ranks are indeed drawn from a uniform distribution.

We inspected rank histogram plots for $f_j : \theta \mapsto \theta_j, j = 1, \ldots, |\theta|$, which focuses on detecting systemic biases with regards to estimating one single parameter. This allows us to compare the results to the results obtained for the methods described in the previous Section 6.3.4.

6.4 Experimental Results

We find that using more simulations generally improves performance and that it becomes more important in higher parameter dimensions. We also find that generally NPE gives overconfident posteriors, which can be reduced by using more simulations. This problem is amplified when working with low simulation budgets (1000) in high dimensions. Conversely, for simulation budgets of 10000 or more simulations, the estimated posteriors do not become increasingly overconfident when we have more parameters.

6.4.1 Posterior Predictive Checks

Results did not look worrying across experiments and simulation budgets as the true observations are usually covered by the 80%-, 95%-, 99%- confidence intervals, although they looked better in lower dimensions. This is exemplified in Figure 16 and can be explored with further plots in Appendix B.1.

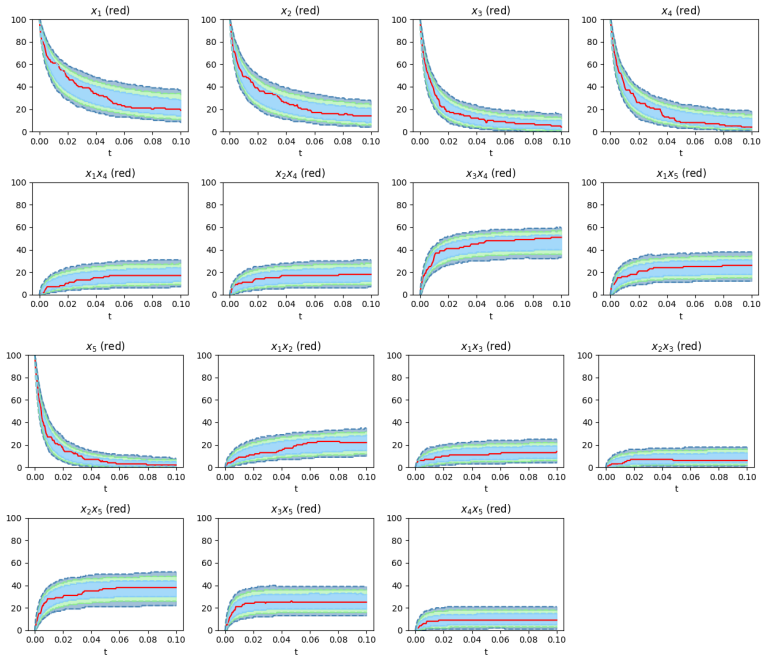

Figure 4: 80%, 95%, 99% - confidence bands for samples of the posterior predictive distribution compared to true observation x_0 (red) for each species observed in **Exp1**, $n = 5$. Density estimator q was trained on 100000 samples.

6.4.2 First metric: Negative Expected Log Probability

Figures 5 and 6 report the negative expected log probability of true samples $-\mathbb{E}[\log q(\theta_0|x_0)]$ of the density estimators trained on simulation budgets of 1000, 10000 or 100000 simulations for **Exp1** and **Exp2** respectively. We also compare to the results that obtained by choosing $q(\theta|x_0)$ to be equal to the prior $p(\theta)$.

Results

- For both experiments across different parameter dimensions, we see that larger simulation budgets lead to better scores. Using larger simulation budgets becomes more important in high dimensions, which can be seen by the increasing divergences of the scores given different simulation budgets.
- Training with just 1000 simulations in higher parameter dimensions leads to scores that are worse than the prior: This is the case when the number of parameters is bigger or equal to 36 for **Exp2** and bigger or equal to 49 parameters for **Exp1**.
- We cannot observe that $-\mathbb{E}[\log q(\theta_0|x_0)]$ scales consistently with respect to the number of parameters.

6.4.3 Second Metric: Average Distance to True Samples

We report the average distances of true samples to estimated samples $M_2(q) = \mathbb{E}[||\theta_0 - \tilde{\theta_0}||_2]$ as well as the scaled average distances $\frac{M_2(q)}{\sqrt{|\theta|}}$ for **Exp1** and **Exp2** for different simulation budgets in Figure 7.

Results

- As can be seen in Figure 7 the average distance $M_2(q)$ scales approximately with the square root of the number of parameters.
- The average distance becomes smaller when using higher simulation budgets.

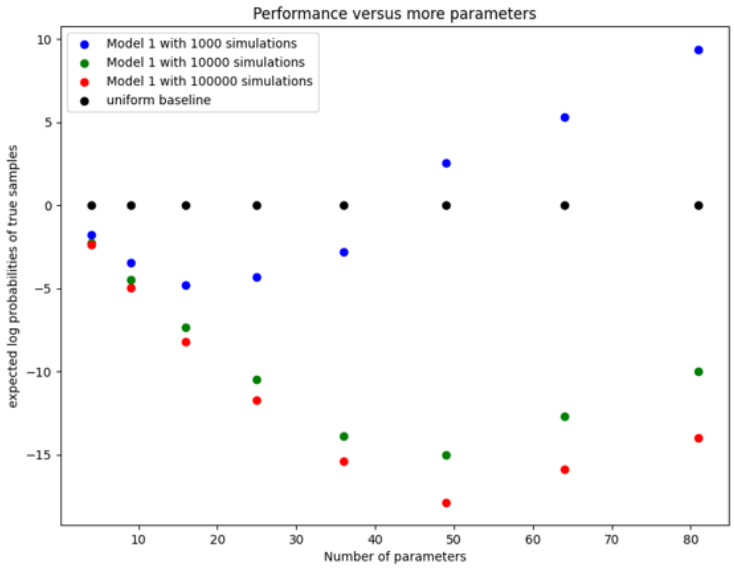

Figure 5: Negative expected log probability of true samples $-\mathbb{E}[\log q(\theta_0|x_0)]$ of density estimators trained on different simulation budgets for **Exp1**

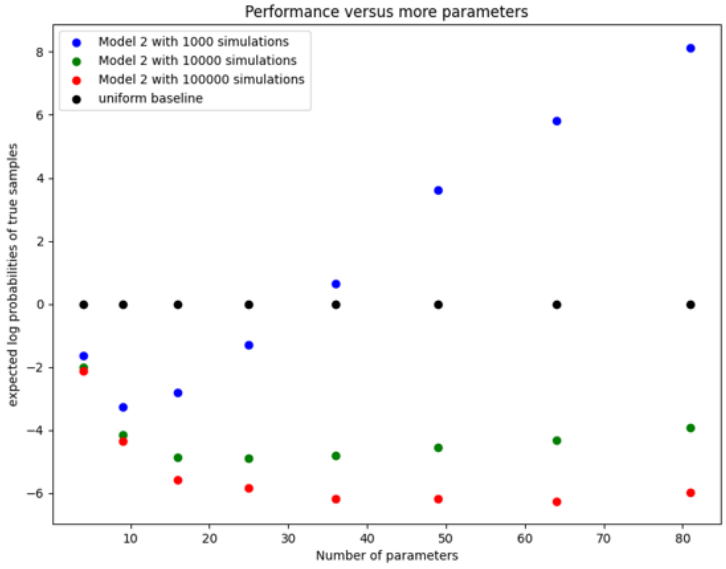

Figure 6: Negative expected log probability of true samples $-\mathbb{E}[\log q(\theta_0|x_0)]$ of density estimators trained on different simulation budgets for **Exp2**

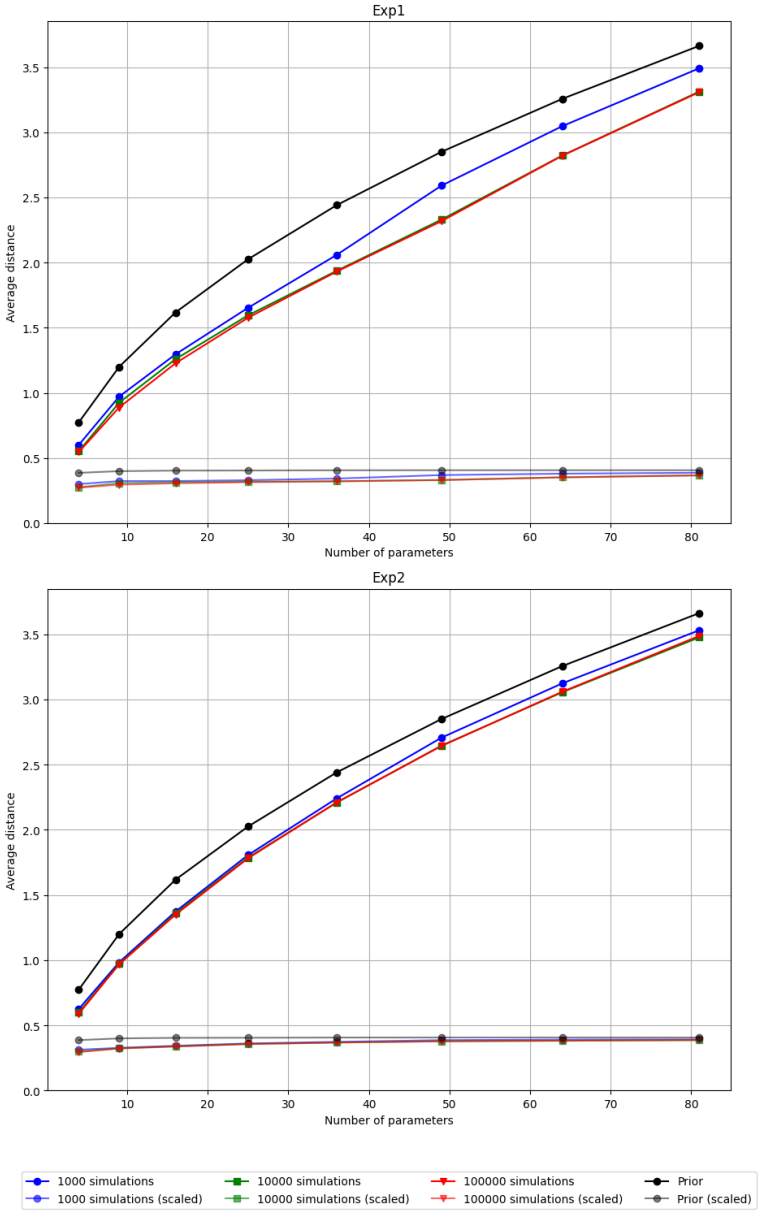

Figure 7: Average distances $M_2(q) = \mathbb{E}_{\tilde{\theta_0} \sim q(\theta|x_0)}[||\theta_0 - \tilde{\theta_0}||_2]$ and scaled average distances $\frac{M_2(q)}{\sqrt{|\theta|}}$ for different simulation budgets and the prior for **Exp1** and **Exp2**

6.4.4 Validation

We report 95% and 68% confidence intervals for the expected coverages of 0.8-, 0.95- and 0.99-credible regions for all the density estimators trained for **Exp1**, **Exp2**, **Exp3**.
For example, Figure 8 reports 95% confidence bands for expected coverages of 0.8 credible regions for all parameters for the density estimators trained for **Exp2**, $n = 4$. Figure 10 reports 95% confidence bands for expected coverages of 0.99 credible regions for all parameters for the density estimators trained for **Exp1**, $n = 2$. Figure 9 reports the expected coverages averaged over all parameters $\frac{1}{|\theta|}\sum_{j=1}^{|\theta|} P_{\alpha,j,q}$ for **Exp1**, **Exp2** and **Exp3**.

Main Results

- All posterior estimators are slightly overconfident. This is exemplified by Figures 8, 10 and by the Figures in Appendix B.2, where most values are below the black line. It can also be seen in Figure 9, where we mostly obtain average values that are smaller than 0.8, 0.95, 0.99 respectively.

- More simulations lead to estimated posteriors that are less overconfident. This can be seen in Figures 8, 10, and in Appendix B.2. It is consistent with Figure 9, where lower simulation budgets lead almost always to worse scores.

- For simulation budgets of 10000 and 100000, the estimated posteriors do not become more overconfident when the parameter dimension increases. The averages reported in Figure 9 for 10000 and 100000 simulations do not clearly decrease with an increasing number of parameters.

- For training with just 1000 simulations, the estimated posteriors become more overconfident when the parameter dimension increases, as the averages reported in Figure 9 for 1000 simulations decrease with an increasing number of parameters.

- The degree of overconfidence is small, especially for the models trained with 100000 simulations. For these models, the 95% confidence intervals for the expected coverages usually contain the corresponding true value (0.8, 0.95, 0.99 respectively): Across the three experiments and the different parameter dimensions, the 95% confidence intervals of the expected coverage contain the true value 0.8 on average 89% of the time. Likewise, the true value 0.95 is contained on average 90% of the time and the true value 0.99 is contained on average 88% of the time. Tables 5, 6, and 7 illustrate that this is consistent across different parameter dimensions and experiments.

We conclude that NPE, when trained with enough simulations, can produce valid but slightly overconfident posteriors across different simulator models and different parameter dimensions.

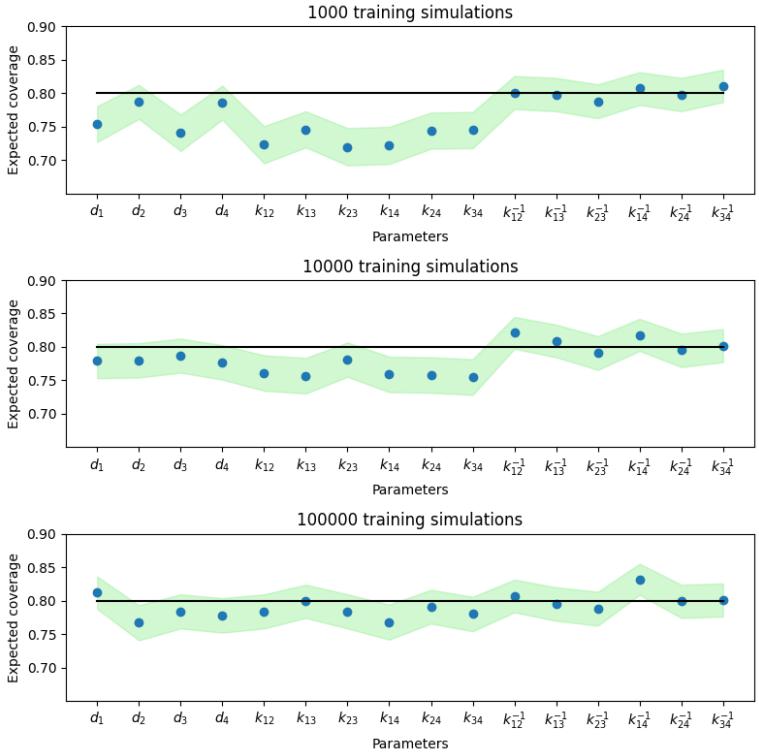

Figure 8: 95% confidence bands for expected coverages of 0.8 credible regions for the density estimators trained for **Exp2**, $n = 4$

Exp / n	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Exp1	0.750	0.889	0.875	1.000	0.889	0.898	0.891	0.951
Exp2	0.500	0.889	0.812	0.840	0.917	0.857	0.859	0.901
Exp3	1.000	1.000	0.938	0.880	0.889	0.918	0.969	0.951

Table 5: Share of parameters where the true value (0.8) is in the 95% confidence interval of the expected coverage for 100000 simulations

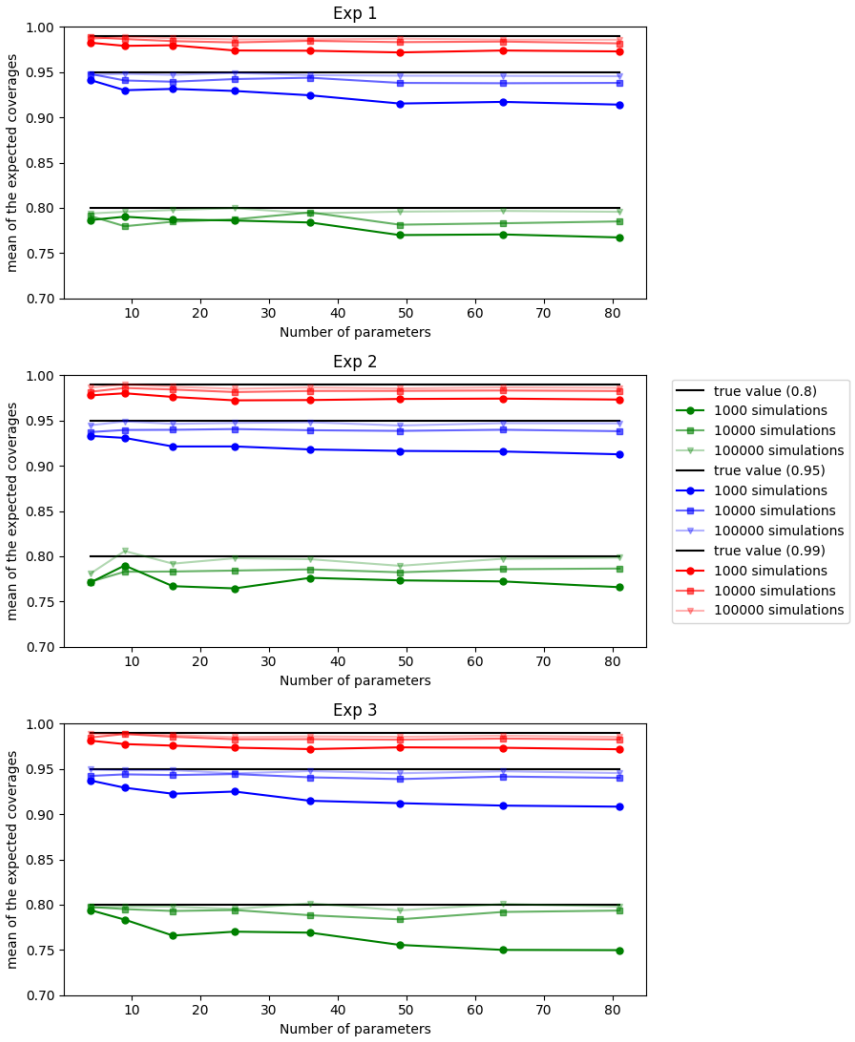

Figure 9: Expected coverages averaged over all parameters $\frac{1}{|\theta|}\sum_{j=1}^{|\theta|} P_{\alpha,j,q}$. Reported for coverage of 0.8 (green), 0.95 (blue), 0.99 (red)-credible regions and simulation budgets of 1000 simulations (highest colour intensity), 10000 simulations (medium colour intensity), 100000 simulations (lowest colour intensity) for **Exp1**, **Exp1**, **Exp2**, **Exp3**. The black lines represent the correct values.

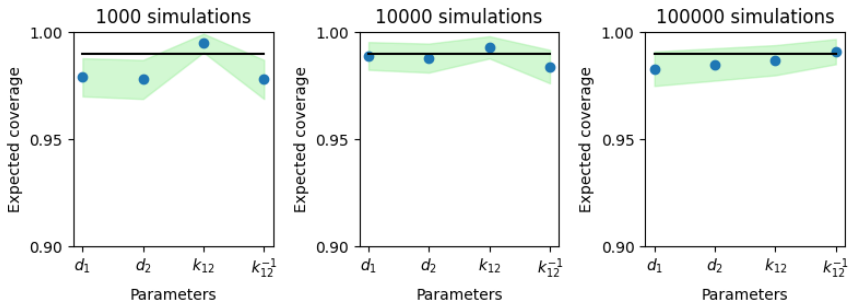

Figure 10: 95% confidence bands for expected coverages of 0.99 credible regions for the density estimators trained for **Exp1**, $n = 2$

6.4.5 Simulation-based Calibration

Inspection of the rank histograms confirmed that the estimated posteriors based on 1000 simulations are slightly overconfident, especially in higher dimensions. This is exemplified by the slightly ∪-shaped histogram plots in Figure 11.

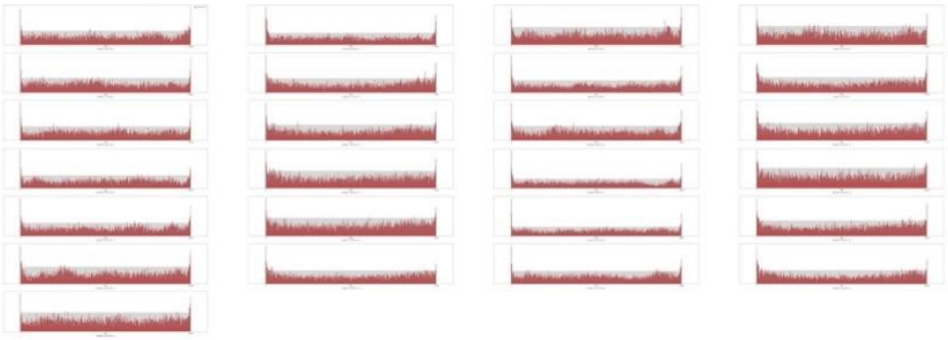

Figure 11: Rank histogram for $n = 5$, **Exp2** for density estimator trained on 1000 simulations

7 Discussion and Outlook

Although it may be the more realistic application of SBI algorithms, working with a simulator with an intractable likelihood prohibits access to true posterior samples. This rendered performance evaluation difficult. The proposed evaluation metrics in Sections 6.3.2 and 6.3.3 are ill-suited to compare performance across different simulators. The first metric relates to the KL-divergence but also depends on the entropy of the true posterior, which we are not able to estimate. The second metric, as a Euclidean distance, obviously increases in the parameter dimension.
To improve this, one could evaluate SBI algorithms on problems with a high parameter dimension that have a tractable likelihood or at least the ability to generate true posterior samples. Then we could evaluate estimated posteriors with 2-sample based approaches, such as the maximum mean discrepancy [Gre+12] or classifier-2-sample-tests [LO16], similarly to [Lue+21].

This work demonstrates empirically for 3 related simulators with different parameter dimensions that NPE can produce overconfident posterior approximations. Across different simulators and different parameter dimensions, we observed expected coverages of $(1-\alpha)$-credible regions that are on average smaller than the true value $(1-\alpha)$, indicating overconfidence. We also saw that this problem is amplified when working with low simulation budgets and high parameter dimensions: For the density estimator trained on 1000 simulations, the deviation of the expected coverage from the true value increased in the parameter dimension. This increased overconfidence also gives a possible explanation for the deteriorating scores obtained in Section 6.3.2: We saw that for a parameter dimension $|\theta| \geq 49$ for **Exp1** and $|\theta| \geq 36$ for **Exp2** respectively, we obtain worse scores than the prior on the first metric $-\mathbb{E}[q(\theta_0|x_0)]$.

Previously, [Her+21] demonstrated for different simulators that different simulation-based inference algorithms can produce overconfident posteriors. We argue that the reliability of simulation-based inference will depend on tackling this issue. We think that this could already be improved by paying stronger attention to not overfitting the training data, incorporating common techniques from deep learning, such as dropout [Sri+14]. Further research should explore if this issue persists when using a more advanced density estimator, such as a continuous normalizing flow [Che+18] or a diffusion model [HJA20]. Ref. [Her+21] also gives an overview of current research directions to tackle this and produce more conservative, that is not overconfident, posterior approximations.

In our experiment, this issue is mitigated by the use of higher simulation budgets ($\geq$ 10000). For simulation budgets of 100000 simulations, NPE

produces quite valid posteriors even in high parameter dimensions. This is demonstrated by the fact that the corresponding 95% confidence intervals of the expected coverages cover the true value $(1 - \alpha)$, on average, more than 88% of the time. This should not be understood as a metric, but rather as a positive signal regarding the applicability of SBI Algorithms on inference problems with higher parameter dimensions (≥ 20).

Limitations

As part of the given setup, we only worked on a small amount of possible simulators and also computed the validation statistics from Section 6.4.4 only for one run of NPE for each simulation budget and simulator. This would have to be repeated for more different simulators from different fields and for several runs of NPE to give more robust results.
In this work, we assumed that the simulator perfectly models the real data generating process and focused on the performance of the inference algorithm. In practice, there might be a gap between the simulation and reality. This issue should also be taken into account when systematically evaluating SBI algorithms. Within the context of our simulators this would be improved in future work by adding a noise model on top of the simulator and then inferring the posteriors of noisy observations.

Outlook

The goal of benchmarking initiatives such as [Lue+21] for SBI algorithms is to provide domain scientists with insights into the applicability of a given SBI algorithm to their problem. We think that previous initiatives could be extended. Conditional neural density estimators make use of embedding networks to embed the conditional variable in a lower dimensional space. We think that the choice of embedding network can be crucial, especially when working with high-dimensional data. For example images require a different embedding network than e.g. time series. We would like to see future benchmark initiatives address this, such that domain scientists can make a well-reasoned decision as to which embedding network would be most suitable for the problem at hand.

References

[ABV23] Michael S Albergo, Nicholas M Boffi, and Eric Vanden-Eijnden. "Stochastic interpolants: A unifying framework for flows and diffusions". In: *arXiv preprint arXiv:2303.08797* (2023).

[And07] David F Anderson. "A modified next reaction method for simulating chemical systems with time dependent propensities and delays". In: *The Journal of chemical physics* 127.21 (2007).

[Bay+18] Atilim Gunes Baydin et al. "Automatic differentiation in machine learning: a survey". In: *Journal of machine learning research* 18.153 (2018), pp. 1–43.

[Bay+19] Atilim Güneş Baydin et al. "Etalumis: Bringing probabilistic programming to scientific simulators at scale". In: *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 2019, pp. 1–24.

[Bay63] Thomas Bayes. "LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFR S". In: *Philosophical transactions of the Royal Society of London* 53 (1763), pp. 370–418.

[Bre+18] Johann Brehmer et al. "Constraining effective field theories with machine learning". In: *Physical review letters* 121.11 (2018), p. 111801.

[Bre+20] Johann Brehmer et al. "Mining gold from implicit models to improve likelihood-free inference". In: *Proceedings of the National Academy of Sciences* 117.10 (2020), pp. 5242–5249.

[Bro+11] Steve Brooks et al. *Handbook of markov chain monte carlo*. CRC press, 2011.

[BVH00] Johannes Berkhof, Iven Van Mechelen, and Herbert Hoijtink. "Posterior predictive checks: Principles and discussion". In: *Computational Statistics* 15 (2000), pp. 337–354.

[BZ14] Vance W Berger and YanYan Zhou. "Kolmogorov–smirnov test: Overview". In: *Wiley statsref: Statistics reference online* (2014).

[BZB02] Mark A Beaumont, Wenyang Zhang, and David J Balding. "Approximate Bayesian computation in population genetics". In: *Genetics* 162.4 (2002), pp. 2025–2035.

[CBL20] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. "The frontier of simulation-based inference". In: *Proceedings of the National Academy of Sciences* 117.48 (2020), pp. 30055–30062.

[Che+18] Ricky TQ Chen et al. "Neural ordinary differential equations". In: *Advances in neural information processing systems* 31 (2018).

[Cor+20] Rob Cornish et al. “Relaxing bijectivity constraints with continuously indexed normalising flows”. In: *International conference on machine learning*. PMLR. 2020, pp. 2133–2143.

[Dur+19] Conor Durkan et al. “Neural spline flows”. In: *Advances in neural information processing systems* 32 (2019).

[Frö+17] Fabian Fröhlich et al. “Scalable parameter estimation for genome-scale biochemical reaction networks”. In: *PLoS computational biology* 13.1 (2017), e1005331.

[GB00] Michael A Gibson and Jehoshua Bruck. “Efficient exact stochastic simulation of chemical systems with many species and many channels”. In: *The journal of physical chemistry A* 104.9 (2000), pp. 1876–1889.

[GD82] John A Gregory and Roger Delbourgo. “Piecewise rational quadratic interpolation to monotonic data”. In: *IMA Journal of Numerical Analysis* 2.2 (1982), pp. 123–130.

[Gil01] Daniel T Gillespie. “Approximate accelerated stochastic simulation of chemically reacting systems”. In: *The Journal of chemical physics* 115.4 (2001), pp. 1716–1733.

[Gil77] Daniel T Gillespie. “Exact stochastic simulation of coupled chemical reactions”. In: *The journal of physical chemistry* 81.25 (1977), pp. 2340–2361.

[Gil92] Daniel T Gillespie. “A rigorous derivation of the chemical master equation”. In: *Physica A: Statistical Mechanics and its Applications* 188.1-3 (1992), pp. 404–425.

[GMR93] Christian Gourieroux, Alain Monfort, and Eric Renault. “Indirect inference”. In: *Journal of applied econometrics* 8.S1 (1993), S85–S118.

[GNM19] David Greenberg, Marcel Nonnenmacher, and Jakob Macke. “Automatic posterior transformation for likelihood-free inference”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2404–2414.

[Gon+20] Pedro J Gonçalves et al. “Training deep neural density estimators to identify mechanistic models of neural dynamics”. In: *Elife* 9 (2020), e56261.

[Gre+12] Arthur Gretton et al. “A kernel two-sample test”. In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 723–773.

[GS17] Matthew Graham and Amos Storkey. “Asymptotically exact inference in differentiable generative models”. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 499–508.

[Her+21] Joeri Hermans et al. "A trust crisis in simulation-based inference? your posterior approximations can be unfaithful". In: *arXiv preprint arXiv:2110.06581* (2021).

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[Job20] Joblib Development Team. *Joblib: running Python functions as pipeline jobs*. 2020. URL: `https://joblib.readthedocs.io/`.

[KB15] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. San Diega, CA, USA, 2015.

[KRH18] Emmanuel Klinger, Dennis Rickert, and Jan Hasenauer. "pyABC: distributed, likelihood-free inference". In: *Bioinformatics* 34.20 (2018), pp. 3591–3593.

[Lip+22] Yaron Lipman et al. "Flow matching for generative modeling". In: *arXiv preprint arXiv:2210.02747* (2022).

[LO16] David Lopez-Paz and Maxime Oquab. "Revisiting classifier two-sample tests". In: *arXiv preprint arXiv:1610.06545* (2016).

[Lue+21] Jan-Matthis Lueckmann et al. "Benchmarking simulation-based inference". In: *International conference on artificial intelligence and statistics*. PMLR. 2021, pp. 343–351.

[Mar+03] Paul Marjoram et al. "Markov chain Monte Carlo without likelihoods". In: *Proceedings of the National Academy of Sciences* 100.26 (2003), pp. 15324–15328.

[Met+53] Nicholas Metropolis et al. "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6 (1953), pp. 1087–1092.

[Pap+21] George Papamakarios et al. "Normalizing flows for probabilistic modeling and inference". In: *Journal of Machine Learning Research* 22.57 (2021), pp. 1–64.

[Pas+19] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

[PFS12] Gareth W Peters, Yanan Fan, and Scott A Sisson. "On sequential Monte Carlo, partial rejection control and approximate Bayesian computation". In: *Statistics and Computing* 22 (2012), pp. 1209–1222.

[PM16] George Papamakarios and Iain Murray. "Fast ε-free inference of simulation models with bayesian conditional density estimation". In: *Advances in neural information processing systems* 29 (2016).

[PPM17] George Papamakarios, Theo Pavlakou, and Iain Murray. "Masked autoregressive flow for density estimation". In: *Advances in neural information processing systems* 30 (2017).

[PSM19] George Papamakarios, David Sterratt, and Iain Murray. "Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows". In: *The 22nd international conference on artificial intelligence and statistics*. PMLR. 2019, pp. 837–848.

[Rat+07] Oliver Ratmann et al. "Using likelihood-free inference to compare evolutionary dynamics of the protein networks of H. pylori and P. falciparum". In: *PLoS Computational Biology* 3.11 (2007), e230.

[RGW20] Guido Reinhardt, Sven Gärtner, and Tobias Wagner. "Ökologische Fußabdrücke von Lebensmitteln und Gerichten in Deutschland". In: *IFEU—Institut für Energie-und Umweltforschung: Heidelberg, Germany* (2020).

[RM15] Danilo Rezende and Shakir Mohamed. "Variational inference with normalizing flows". In: *International conference on machine learning*. PMLR. 2015, pp. 1530–1538.

[Rud+64] Walter Rudin et al. *Principles of mathematical analysis*. Vol. 3. McGraw-hill New York, 1964.

[Rud16] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[SFT07] Scott A Sisson, Yanan Fan, and Mark M Tanaka. "Sequential monte carlo without likelihoods". In: *Proceedings of the National Academy of Sciences* 104.6 (2007), pp. 1760–1765.

[Son+20] Yang Song et al. "Score-based generative modeling through stochastic differential equations". In: *arXiv preprint arXiv:2011.13456* (2020).

[Sri+14] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[Sta23] Statista. *Greenhouse Gas Emissions per Capita in Germany*. 2023. URL: https://www.statista.com/statistics/1388886/greenhouse-gas-emissions-per-capita-germany/.

[Sta24] Statista. *Emissionen Strom Deutschland und Frankreich*. May 9, 2024. URL: https://de.statista.com/statistik/daten/studie/1421117/umfrage/emissionen-strom-deutschland-und-frankreich/#:~:text=In%20Deutschland%20lag%20der%20Emissionsfaktor,Thema%20Energiewende%20finden%20Sie%20hier.

[Tal+18] Sean Talts et al. "Validating Bayesian inference algorithms with simulation-based calibration". In: *arXiv preprint arXiv:1804.06788* (2018).

[Tej+20] Alvaro Tejero-Cantero et al. "sbi: A toolkit for simulation-based inference". In: *Journal of Open Source Software* 5.52 (2020), p. 2505. DOI: 10.21105/joss.02505. URL: https://doi.org/10.21105/joss.02505.

[WBS19] David J Warne, Ruth E Baker, and Matthew J Simpson. "Simulation and inference algorithms for stochastic biochemical reaction networks: from basic concepts to state-of-the-art". In: *Journal of the Royal Society Interface* 16.151 (2019), p. 20180943.

[WL19] Antoine Wehenkel and Gilles Louppe. "Unconstrained monotonic neural networks". In: *Advances in neural information processing systems* 32 (2019).

[YSS05] G Alastair Young, Richard L Smith, and Richard L Smith. *Essentials of statistical inference*. Vol. 16. Cambridge University Press, 2005.

A Appendix A: Further Definitions and Theorems

Proposition A.1. *Let P, Q be probability measures on $\mathcal{B}(\mathbb{R}^d)$, that are absolutely continuous with respect to the lesbegue measure. Let p, q denote their probability density functions. Then*

$$D_{KL}(P||Q) + \mathbb{H}_p(x) = -\mathbb{E}_{p(x)}[\log q(x)],$$

where $\mathbb{H}_p(x) = -\mathbb{E}_{p(x)}[\log p(x)]$ denotes the entropy of P.

Proof.

$$\begin{aligned} & D_{KL}(P||Q) \\ & = \mathbb{E}_{p(x)}[\log(\frac{p(x)}{q(x)})] \\ & = \mathbb{E}_{p(x)}[-\log q(x) + \log p(x)] \\ & = -\mathbb{E}_{p(x)}[\log q(x)] - \mathbb{H}_p(x) \end{aligned}$$

□

Definition A.2. Let $\mathcal{X}, \mathcal{Y} \subset \mathbb{R}^d$. A homeomorphism is a map $f : \mathcal{X} \to \mathcal{Y}$, such that

1. f is bijective
2. f is continuous
3. the inverse f^{-1} is continuous as well.

We write $\mathcal{X} \cong \mathcal{Y}$ and say $\mathcal{X}$ is homeomorph to $\mathcal{Y}$.

B Appendix B: Additional Figures

Simulator

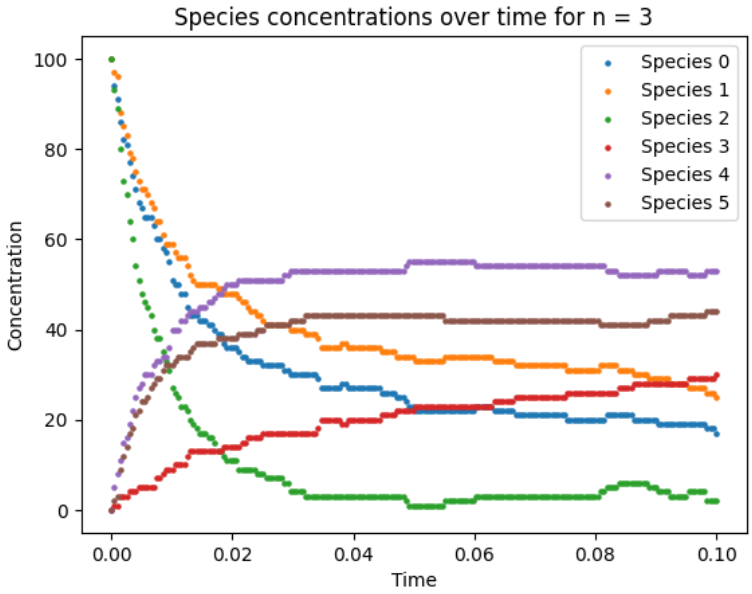

Figure 12: Example of Simulator output for n = 3. Time is an arbitrary unit.

B.1 Posterior Predictive Checks

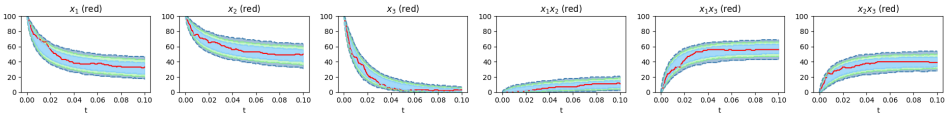

Figure 13: 80%, 95%, 99% - confidence bands for samples of the posterior predictive distribution compared to true observation x_0 (red) for each species observed in **Exp1**, $n = 3$, simulation budget = 10000

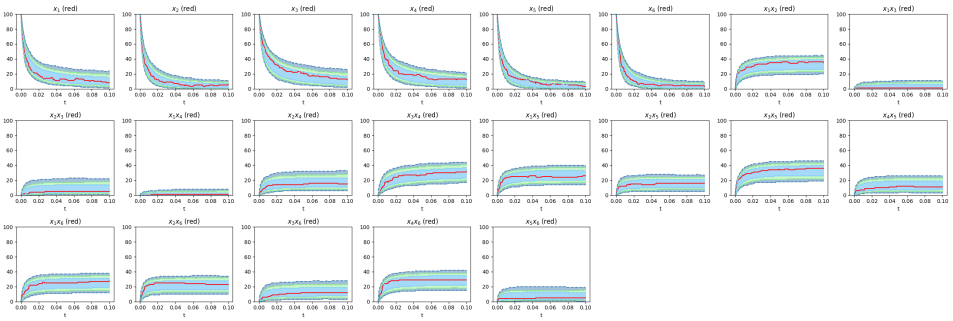

Figure 14: 80%, 95%, 99% - confidence bands for samples of the posterior predictive distribution compared to true observation x_0 (red) for each species observed in **Exp1**, $n =$, simulation budget $= 10000$

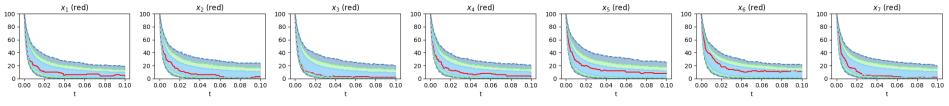

Figure 15: 80%, 95%, 99% - confidence bands for samples of the posterior predictive distribution compared to true observation x_0 (red) for each species observed in **Exp2**, $n = 7$, simulation budget $= 1000$

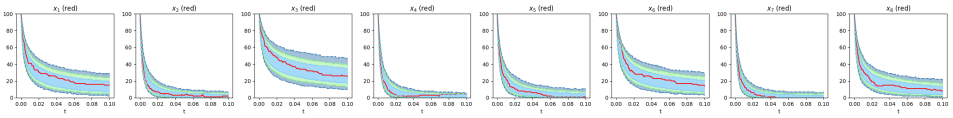

Figure 16: 80%, 95%, 99% - confidence bands for samples of the posterior predictive distribution compared to true observation x_0 (red) for each species observed in **Exp2**, $n =$, simulation budget $= 100000$

B.2 Validation

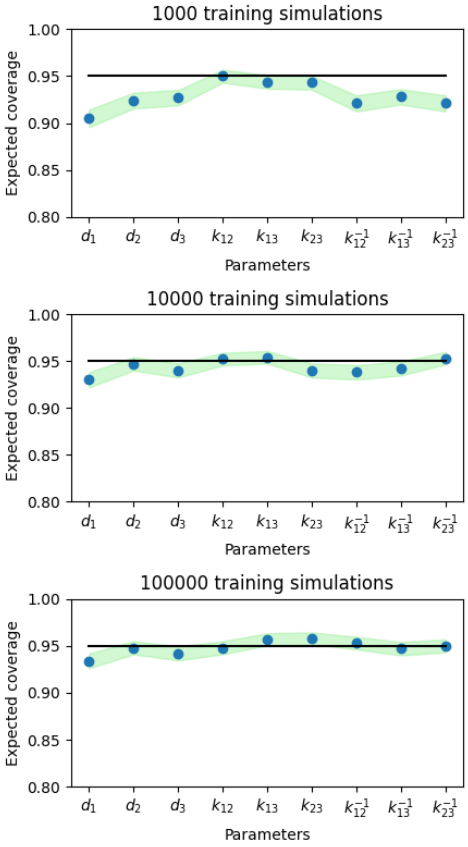

Figure 17: 68% confidence bands for expected coverages of 0.95 credible regions for the density estimators trained for **Exp3**, $n = 3$

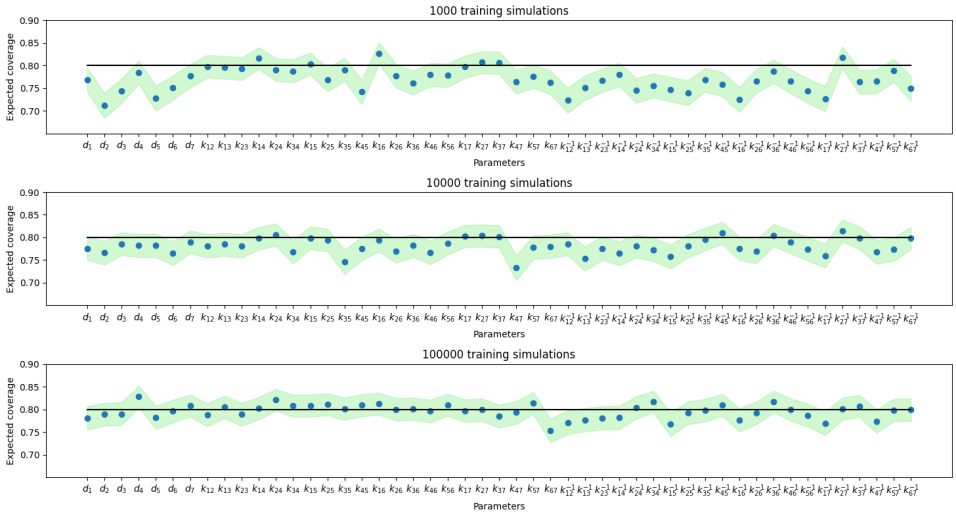

Figure 18: 95% confidence bands for expected coverages of 0.8 credible regions for the density estimators trained for **Exp1**, $n = 7$

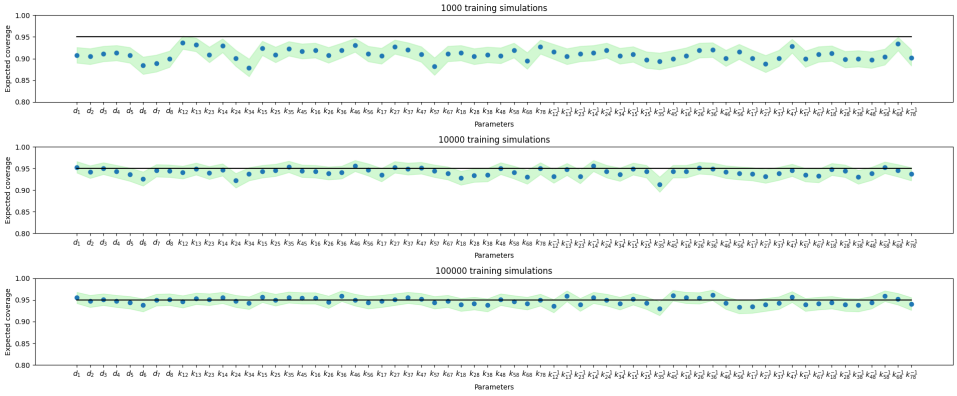

Figure 19: 95% confidence bands for expected coverages of 0.95 credible regions for the density estimators trained for **Exp3**, $n = 8$

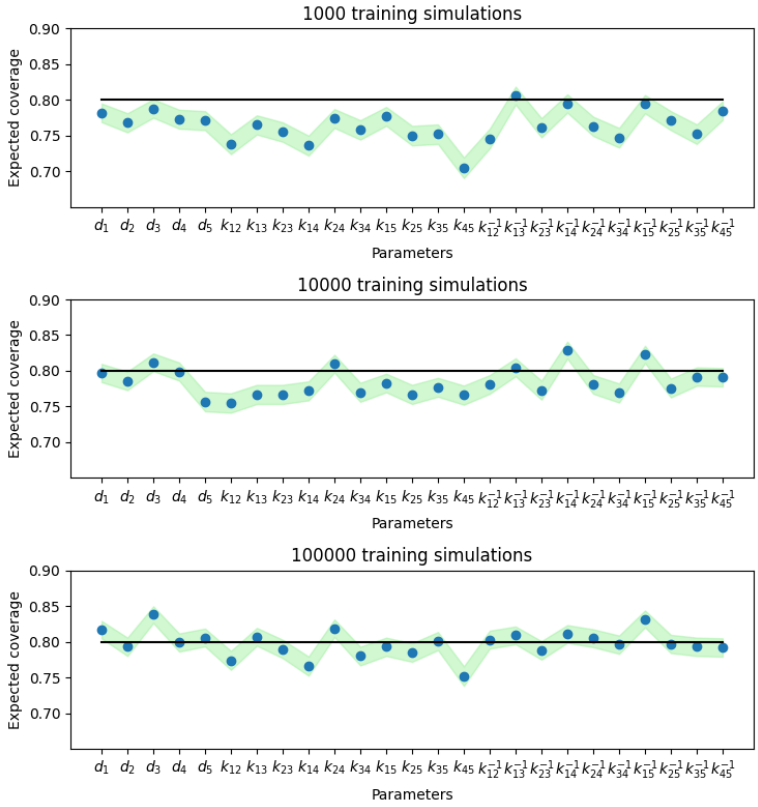

Figure 20: 68% confidence bands for expected coverages of 0.8 credible regions for the density estimators trained for **Exp2**, $n = 5$

Exp / n	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Exp1	1.000	1.000	1.000	1.000	0.889	0.918	0.812	0.938
Exp2	1.000	0.889	1.000	0.960	0.972	0.857	0.938	0.963
Exp3	1.000	0.889	1.000	0.840	0.917	0.898	0.969	0.914

Table 6: Share of parameters where the true value (0.95) is in the 95% confidence interval of the expected coverage for 100000 simulations

Exp / n	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Exp1	1.000	1.000	0.938	0.760	0.889	0.898	0.750	0.840
Exp2	0.750	0.889	0.938	0.720	0.889	0.796	0.906	0.901
Exp3	1.000	1.000	0.938	0.880	0.889	0.857	0.859	0.877

Table 7: Share of parameters where the true value (0.99) is in the 95% confidence interval of the expected coverage for 100000 simulations

Code

Relevant code can be found at https://github.com/pfuhr/BachelorThesis.