Effective Approximate Inference for Nested Simulators

Bradley J. Gram-Hansen[1] **Adam Golinski**[1] **Christian Schroeder de Witt**[1]
Saeid Naderiparizi[2] **Adam Scibior**[2] **Andreas Munk**[2]
Frank Wood[2] **Philip Torr**[1] **Yee Whye Teh**[1]
Atilim Gunes Baydin[1] **Tom Rainforth**[1]
[1]University of Oxford [2]University of British Columbia

Abstract

We introduce two approaches for conducting effective approximate inference in stochastic simulators containing nested stochastic sub-procedures, i.e. internal procedures for which the density cannot be calculated directly such as rejection sampling loops and nested inferences. The resulting class of simulators are used extensively throughout the sciences, but fall outside the standard class of Bayesian models: they are doubly intractable. Drawing inferences from them poses a substantial challenge due to the inability to evaluate even their unnormalized density, preventing the use of standard procedures. In particular, the small number of specialized existing methods that can deal with such models are based around forward sampling and thus scale catastrophically poorly in the dimensionality. To address this, we introduce algorithms based on a two-step approach that first approximates the conditional densities of the individual sub-procedures, before using these approximations to run an MCMC sampler on the full simulator. Because the sub-procedures can be dealt with separately and are lower-dimensional than the overall problem, this two-step process allows them to be isolated, and thus tractably dealt with, without placing restrictions on the overall dimensionality. We show empirically that our approaches provide effective inference in settings that cannot be practically handled by existing methodology.

Preliminary work. Under review by UAI 2021. Do not distribute.

1 Introduction

Stochastic simulators are used in a myriad of scientific and industrial settings, such as epidemiology (Patlolla et al., 2004; Ferguson et al., 2006; Smith et al., 2008), physics (Heermann, 1990; Gleisberg et al., 2009), engineering (Hangos and Cameron, 2001) and climate modelling (Held, 2005). They can be complex and high-dimensional, often incorporating domain-specific expertise accumulated over many years of development.

As shown by the probabilistic programming (Goodman et al., 2012; Gordon et al., 2014; van de Meent et al., 2018) and approximate Bayesian computation (ABC) (Csilléry et al., 2010; Marin et al., 2012) literatures, these simulators can be interpreted as probabilistic generative models, implicitly defining a probability distribution over their internal variables and outputs. As such, they form valid targets for drawing Bayesian inferences. In particular, by constraining selected internal variables or outputs to take on specific values, we implicitly define a conditional distribution, or posterior, over the remaining variables. This effectively allows us to, amongst other things, run the simulator in *reverse*, fixing the outputs to some observed values and inferring what parameter values might have led to them. For example, given a simulator for modeling high-energy physics (Gleisberg et al., 2009), we can run inference on the simulator with an observed energy deposit to infer what decay patterns might have led to them (Baydin et al., 2019a).

Though recent advances in probabilistic programming systems (PPSs) (Le et al., 2017; Tran et al., 2017; Rainforth, 2018; Bingham et al., 2019) have provided convenient mechanisms for encoding and reasoning about such simulators, performing the necessary inference is still often extremely challenging, particularly for complex or high-dimensional problems.

In this paper, we consider a scenario where this inference is particularly challenging to perform: when the simulator makes calls to nested stochastic sub-procedures (NSSPs). These can take several different forms, such as internal rejection sampling loops (Gleisberg et al., 2009; Di Pasquale et al., 2015), nested inference procedures (Smith et al., 2008; Stuhlmüller and Goodman, 2014; Bershteyn et al., 2018; Rainforth, 2018), external sub-simulators we have no control over, or even real-world experiments (Foster et al., 2019).

Their unifying common feature is that the density of their outputs cannot be evaluated up to an input-independent normalizing constant in closed form. This, in turn, means the normalized density of the overall simulator cannot be evaluated. They therefore fall outside the standard class of Bayesian inference problems; they are doubly intractable (Murray, 2007). This prevents one from using most common inference methods, such as conventional Markov Chain Monte Carlo (MCMC) and variational inference approaches. In some cases, such as nested probabilistic programs (Stuhlmüller and Goodman, 2014; Rainforth, 2018), one cannot even directly construct consistent Monte Carlo estimator at all, having to instead turn to *nested* estimation approaches (Rainforth et al., 2018), such as nested importance sampling (Rainforth, 2018; Naderiparizi et al., 2019). These have fundamentally slower convergence rates than standard Monte Carlo approaches (Fort et al., 2017; Rainforth et al., 2018) and thus want to be avoided at all costs.

Even for the subclass of NSSPs where these issues with nesting can be avoided, existing applicable approaches rely on *forward sampling* to get around the lack of a closed form density (Goodman et al., 2012; Wood et al., 2014; Rainforth, 2017). This forward sampling typically suffers acutely from the curse of dimensionality and thus circumvents the use of such approaches on all but simplest of problems.

To address these issues, we introduce two new approaches for performing inference in such models. Both are based around approximating the individual NSSPs. The first, generally applicable, approach directly approximates the conditional density of the NSSP outputs using an amortized inference artefact. This then forms a surrogate density for the NSSP, which, once trained, is used to replace it.

Our second approach focuses on the specific case where the *unnormalized* density of the NSSP can be evaluated in isolation, such as a nested probabilistic program or rejection sampling loop, where but its normalizing constant depends on the NSSP inputs. Here, we train a regressor to approximate the normalizing constant of the NSSP as a function of its inputs. Once learnt, this allows the NSSP to be collapsed into the outer program: the ratio of the known unnormalized density and the approximated normalizing constant can be directly used as a factor in the overall density.

Both approaches lead to an *approximate* version of the overall unnormalized density, which can then be used as a target for conventional inference methods like MCMC and variational inference. Because these approximations can be calculated separately for each NSSP, this allows the approach to scale to higher dimensional overall simulators far more gracefully than existing approaches, opening the door to tractably running inference for much more complex problems. Further, once trained, the approximations can be reused for different datasets and configurations of the outer simulator, thereby amortizing the cost of running multiple different inferences for no extra cost. The approaches themselves are also amenable to automation, making them suitable candidates for PPS inference engines.

We confirm the utility of our approaches using a conceptually simple, but numerically challenging, artificial simulator that has been carefully constructed to allow analytic calculation of a ground truth. Namely, we show that while existing approaches completely break down in more than a few dimensions, our approach is able to gracefully scale with increasing dimensionality and produce effective inference.

2 Background and Problem Formulation

NSSPs arise naturally in many real-world systems. Sometimes they are inherent to the model itself, such as in *nested inference* settings (Rainforth, 2018), whereby restrictions in the flow of information cause a double intractability (Murray et al., 2006), e.g., because we are modeling two agents reasoning about each other (Stuhlmüller and Goodman, 2014). They can also occur because we only have access to a sampler for part of the model and not its density—e.g., because the simulator relies on rejection sampling loops (Gleisberg et al., 2009; Di Pasquale et al., 2015) or comprises of a complex external simulator of its own (Marin et al., 2012). Of particular note, recent breakthroughs in universal probabilistic programming to large-scale scientific simulators (Lezcano Casado et al., 2017; Baydin et al., 2019a; Gram-Hansen et al., 2019), have provided automated ways to translate existing large scale stochastic simulators into probabilistic programming systems, without having to re-write the existing simulator inside the given PPS. However, many of these simulators contain NSSPs.

We now formalize the problem of models containing NSSPs before providing some background on existing strategies for coping with them.

2.1 Problem Formulation

For any simulator or *program*, we can define the program density over valid program traces $x_{1:n_x}$ as:

$$p(x_{1:n_x}) \propto \gamma(x_{1:n_x}) = \prod_{j=1}^{n_x} f_{a_j}(x_j|\phi_j) \prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k) \tag{1}$$

Here n_x is the length of the trace. Each $f_{a_j}(x_j|\phi_j)$ represents the density of the j^{th} random draw, which is made at location a_j and takes in parameters ϕ_j. n_y is a number of *observations*, each of which factor the trace density by $g_{b_k}(y_k|\psi_k)$, where b_k is the location of this observation statement, y_k is the observed value, and ψ_k are parameters of the factorization. All terms—i.e., x_j, n_x, a_j, ϕ_j, n_y, b_k, y_k, ψ_k—may be random variables, but each is deterministically calculable from the trace $x_{1:n_x}$ (see, e.g., Rainforth (2017, Section 4.3.2)).

A NSSP can now be formally defined as a $f_{a_j}(x_j|\phi_j)$ term which cannot be directly evaluated exactly, but where for a given ϕ_j either **[Case A]** we can draw samples from $f_{a_j}(x_j|\phi_j)$ directly or **[Case B]** we have access to an unnormalized form of the density $\gamma^{in}_{a_j}(x_j|\phi_j)$,[1] but do not know the corresponding *input dependent* normalization constant $I_{a_j}(\phi_j)$. In some cases, NSSPs can correspond to both cases (i.e. we can sample and have the unnormalized density), but all NSSPs must conform to at least one of them as otherwise the density would be undefined.

We can now denote the *unnormalized* density for a program containing NSSPs as:

$$\gamma(x_{1:n_x}) = P_{pr}(x_{1:n_x}) \prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k) \quad \text{where} \tag{2}$$

$$P_{pr}(x_{1:n_x}) := \prod_{\{j\in 1:n_x | a_j \notin S_r\}} f_{a_j}(x_j|\phi_j) \prod_{\{j\in 1:n_x | a_j \in S_r\}} P^{in}_{a_j}(x_j|\phi_j) \tag{3}$$

is a representation of the *forward* or *prior* program which ignores all conditioning statements; $S_r = \{a_1, \dots, a_n\}$ represents the set of addresses that produce intractable densities; and we use $P^{in}_{a_j}(x_j|\phi_j)$ to distinguish the NSSPs from tractable sampling terms. We explain how to addresses $a_i \in S_r$ can be extracted automatically from real-world simulators in Appendix A (see also Gram-Hansen et al. (2019); Baydin et al. (2019b)), but for now we will just assume they are known, which is often the case in practice anyway.

[1]More typically, we actually only have access to some pre-image of $\gamma^{in}_{a_j}(x_j|\phi_j)$, which turns out to be sufficient. See Section 3.2.

2.2 Forward Sampling

The issues imposed by NSSPs of type Case A are immediately apparent: we have no direct characterization of the density of the NSSP at all and must somehow leverage our ability to generate samples from the NSSP to run our overall inference. The simplest way to do this is to simply forward sample from the full program (Goodman et al., 2012), that is draw samples from $P_{pr}(x_{1:n_x})$, before weighting these samples using $\prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k)$ (Rainforth, 2017). This likelihood weighting approach equates to importance sampling using the prior as our proposal and thus inevitably scales catastrophically poorly as the dimension increases.

Unfortunately, it transpires to be surprisingly difficult to improve on this naive approach. One setting in which improvements can be made is if that if the number of observations n_y is fixed and these observations are interleaved with the sampling statements. Here we can employ particle based inference methods (Wood et al., 2014), like sequential Monte Carlo (Doucet et al., 2001), to exploit the intermediary information provided by these observations. However, many, if not most, models do not possess such interleaving, for which such methods regress back to simple likelihood weighting.

In principle, one can also use Approximate Bayesian Computation (ABC) methods in such settings (Tavare et al., 1997; Pritchard et al., 1999; Sunnåker et al., 2013). However, these will generally be inferior to simply likelihood weighting: the density of the *likelihood* of our overall program is actually directly evaluable, it is the density of our *prior* that is intractable, meaning that such approximations are not generally necessary.

2.3 Nested Monte Carlo and Nested Inference

The issues imposed by NSSPs of type Case B are arguably more subtle. To understand these, we observe that they are equivalent to *nested probabilistic programs* (Stuhlmüller and Goodman, 2014; Rainforth, 2018): they define their own normalized density and require separate, *nested*, inference procedures to be run—either to draw samples or to estimate their posterior density—for each sample realization of the outer program. Such problems are known as *nested inference* problems (Mantadelis and Janssens, 2011; Rainforth, 2018) and they correspond to a more general class of problems than conventional Bayesian inference. In essence, the nesting corresponds to a local normalization of the density, rendering the problem doubly intractable because we have to solve an intractable inference for *each* realization of the outer program.

Conducting inference in such problems requires the use of *nested estimators*, most typically nested Monte Carlo (NMC, Fort et al. (2017); Rainforth et al. (2018)); naïvely running conventional inference approaches like MCMC leads to inconsistent estimators with substantial asymptotic biases. Rainforth (2018) show how consistent nested estimators can be derived through a careful nesting of importance samplers wherein the estimators for the nested programs use more samples as the number of iterations of the outer program increases. Though this is arguably inevitable (Rainforth et al., 2018), the resulting convergence rates of these nested importance samplers are, unfortunately, fundamentally slower than conventional Monte Carlo. Namely the mean squared errors of their estimators converge at rate $\mathcal{O}(1/N^{2/3})$ rather than $\mathcal{O}(1/N)$ in the number of samples N. Moreover, because of the reliance on importance sampling, such approaches suffer acutely from the curse of dimensionality and cannot be used on anything but the most simple problems. Despite this, they remain the state-of-the-art approach for dealing with such problems (Naderiparizi et al., 2019).

3 Approximating NSSPs

We now introduce our approaches for approximating NSSPs and show how these approximations can, in turn, be used to produce efficient inference algorithms for the overall simulator. Both methods are based on replacing each of the intractable NSSP densities, i.e. $P_{a_j}^{in}(x_j|\phi_j)$ in (3), with an approximation. Once learned, these can then be used to construct a directly evaluable approximate target density $\hat{\gamma}(x_{1:n_x}) \approx \gamma(x_{1:n_x})$ by replacing each $P_{a_j}^{in}(x_j|\phi_j)$ in (3), then running an MCMC sampler (or some other conventional inference method) on $\hat{\gamma}(x_{1:n_x})$. The performance for the resulting estimators thus depends on both the accuracy of approximations $\hat{\gamma}(x_{1:n_x})$, in a manner akin to the error in variational inference but without requiring of mean field assumptions, and the efficiency of the MCMC sampling.

To be more specific, our approaches involve the gradient-based learning of a neural-network-based amortized approximation for each NSSP that takes in the NSSP inputs and either returns an approximation of the density of the outputs (Method 1) or the normalizing constant (Method 2).

Note the critical feature that learning these approximations does not require access to the observations of the outer program (i.e. the g_{b_k}); they operate only on the prior program $P_{pr}(x_{1:n_x})$. Moreover, the learning them can be done *independently* for each NSSP: the outer program is used only to provide typical example inputs to each NSSP and does not effect the relative optimality of the approximation of each NSSP for a given input. As such, the learning of these approximation should, at least in theory, scale only with the size of the individual NSSPs, not the number of NSSPs or the dimensionality of the overall problem.

3.1 Method 1: Surrogate Replacement

Our first method is based around learning an amortized variational approximation of each NSSP. The goal of amortization is to learn a parameterized function that can map from different sets of observations (Kingma and Welling, 2014; Rezende et al., 2014) to parameters that define an approximate posterior distribution under those observations. This means we learn a set of parameters once, during an offline training procedure, that can then be used as an approximation of the posterior for all possible inputs. In the typical setting these inputs would be data, but for us they will correspond to the NSSP inputs ϕ_j.

To be precise, our method replaces each $P_{a_j}^{in}(x_j|\phi_j)$ by an approximate surrogate $q_{a_j}^{in}(x_j|\phi_j;\eta_{a_j})$:

$$P_{pr}(x_{1:n_x}) \simeq q(x_{1:n_x};\kappa) := \prod_{\{j\in 1:n_x | a_j \notin S_r\}} f_{a_j}(x_j|\phi_j) \prod_{\{j\in 1:n_x | a_j \in S_r\}} q_{a_j}^{in}(x_j|\phi_j;\eta_{a_j})$$

where $\kappa = \{\eta_{a_j}; a_j \in S_r\}$ are the surrogate parameters. Following existing amortized variational approaches (Kingma and Welling, 2014; Rezende et al., 2014; Le et al., 2017; Ritchie et al., 2016; Paige and Wood, 2016), each $q_{a_j}^{in}(x_j|\phi_j;\eta_{a_j})$ is taken as a variational distribution parametrized by a deep neural network with weights η_{a_j} and takes ϕ_j as its input. Training of these networks is done by minimizing the Kullback–Leibler (KL) divergence from $P_{pr}(x_{1:n_x})$ to $q(x_{1:n_x};\kappa)$ (Paige and Wood, 2016)

$$\begin{aligned}\mathrm{KL}(P_{pr}(x_{1:n_x})||q(x_{1:n_x};\kappa)) \\ = \int P_{pr}(x_{1:n_x}) \log\left(\frac{P_{pr}(x_{1:n_x})}{q(x_{1:n_x};\kappa)}\right) dx.\end{aligned}$$

The optimal network parameters are now given by

$$\begin{aligned}\kappa^* &= \underset{\kappa}{\mathrm{argmin}}\, \mathrm{KL}\left(P_{pr}||q_\kappa\right) \\ &= \underset{\{\eta_{a_j};a_j\in S_r\}}{\mathrm{argmin}}\, \mathbb{E}_{P_{pr}}\left[\sum_{\{j\in 1:n_x|a_j\in S_r\}} -\log q_{a_j}^{in}(x_j|\phi_j;\eta_{a_j})\right] \end{aligned} \quad (4)$$

$$\eta_r^* = \underset{\eta_r}{\mathrm{argmax}}\, \mathbb{E}_{P_{pr}}\left[\sum_{j=1}^{n_x} \mathbb{I}(r=a_j)\log(q_r^{in}(x_j|\phi_j;\eta_r))\right] \quad (5)$$

$\forall r \in S_r$. Because each individual problem is low dimensional, we can efficiently learn an approximation to each NSSP in the set S_r, noting from (5) that these effectively break down into separate problems. As the expectations of equation (4) and (5) are with respect to the simulator density, the required minimization can be done using stochastic gradient descent. Namely, we can generate (potentially approximate) input-output pairs $\{\phi_j, x_j\}$ by running the forward simulator and then use the gradient estimator ($\forall r \in S_r$)

$$\nabla_{\eta_r}\text{KL} \approx -\frac{1}{N}\sum_{k=1}^{N}\sum_{j=1}^{n_x}\mathbb{I}(r = a_j^k)\nabla_{\eta_r}\log(q_r^{in}(x_j^k|\phi_j^i;\eta_r)).$$

If the given NSSP is of type Case A, drawing these samples is straightforward as, by assumption, we can then draw samples from each $P_{a_j}^{in}(x_j|\phi_j)$ and, in turn, samples from P_{pr}, which is computationally cheap. However, if our program contains NSSPs of type Case B, this will require us to run a separate nested inference (Rainforth, 2018) on these NSSPs to generate the required x_j^k from the corresponding ϕ_j. Though this is potentially non-trivial, it is, crucially, far easier than running inference on the overall program: because P_{pr} itself does not include any conditioning statements, generating these samples does not require inference to be run for the outer program. As such, each nested inference problem constitutes its own isolated problem which is far simpler than the overall inference. In other words, the role of sampling from P_{pr} is only to generate example input-output pairs for each NSSP, with each surrogate then separately trained using its local pairs.

3.2 Method 2: Normalization Constant Approximation

If all of our NSSPs satisfy Case B, this implies that each has a known unnormalized density on its internal variables and unknown input-dependent normalizing constant that causes a double-intractability. If the functional forms for all these normalizing constants were known, this would be sufficient to collapse all the NSSPs into the outer program and produce a directly evaluable density for the overall program. Our second method thus looks to learn regressors to predict the normalizing constants and thereby facilitate this. Though in this approach we still use the NSSPs after the regressors have been learned, we no long need to perform a *nested inference* on them: we have converted the problem from a doubly-intractable inference, to a conventional inference.

To formalize this, let us for now assume that the x_j returned by each NSSP corresponds to its full set of internal random draws $z_{1:n_x^j}^j$, i.e., $x_j = z_{1:n_x^j}^j$, such that

$$P_{a_j}^{in}(x_j|\phi_j) = \frac{\gamma_{a_j}^{in}(x_j|\phi_j)}{I_{a_j}(\phi_j)} = \frac{\gamma_{a_j}^{in}\left(z_{1:n_x^j}^j\middle|\phi_j\right)}{I_{a_j}(\phi_j)} \tag{6}$$

where $\gamma_{a_j}^{in}(z_{1:n_x^j}^j|\phi_j)$ can be evaluated directly, because it is itself an unnormalized probabilistic program density of the form (1), but $I_{a_j}(\phi_j)$ is an intractable normalization constant. If we now introduce a set of regressors $R_r(\phi_j;\tau_r)$, $\forall r \in S_r$, with parameters τ_r, to approximate each $I_r^{in}(\phi_j)$, we can approximate P_{pr} as

$$P_{pr}(x_{1:n_x}) \simeq \prod_{\{j\in 1:n_x|a_j\notin S_r\}} f_{a_j}(x_j|\phi_j)\prod_{\{j\in 1:n_x|a_j\in S_r\}}\frac{\gamma_{a_j}^{in}\left(z_{1:n_x^j}^j\middle|\phi_j\right)}{R_{a_j}(\phi_j;\tau_{a_j})}.$$

We can extend this approach to the case where $x_j = \Omega(z_{1:n_x^j}^j)$ for some deterministic function Ω, by instead defining our reference measure in the space of $\mathcal{X}_a := \{x_j\}_{j\in 1:n_x|a_j\notin S_r}\cup\{z_{1:n_x^j}^j\}_{j\in 1:n_x|a_j\in S_r}$ and using the pre-image of the prior program density: $P_{pr}(\mathcal{X}_a)$. We can then run inference in this pre-image space and rely on the law of the unconscious statistician to ensure the samples produced are from the desired posterior, see e.g., Rainforth (2017, Section 4.3.2).

Learning the regressors $R_r(\phi_j;\tau_r)$ is done in a similar vein to method 1. Namely we run the simulator forward to gather pairs $\{\phi_j, \hat{I}_r(\phi_j)\}$ for each NSSP, where $\hat{I}_r(\phi_j)$ is an unbiased approximation of $I_r(\phi_j)$, and then use this as a training dataset for learning the regressor. Specifically, for each NSSP we train a neural network regressor to minimize the expected squared error between $R_r(\phi_j;\tau_r)$ and $\hat{I}_r(\phi_j)$. Thus, our objective and gradient update are:

$$\mathcal{L}_r = \mathbb{E}\left[\left(R_r(\phi_j;\tau_r) - \hat{I}_r(\phi_j)\right)^2\right], \tag{7}$$

$$\nabla_{\tau_r}\mathcal{L}_r = \mathbb{E}\left[\nabla_{\tau_r}\left(R_r(\phi_j;\tau_r) - \hat{I}_r(\phi_j)\right)^2\right] \tag{8}$$

where the expectation is over both the estimates and the inputs ϕ_j, with the distribution of the latter defined by running P_{pr} forward and, if necessary, randomly selecting between the inputs that are passed to NSSP r if it is called more than once. This can further be Rao-Blackwellized by averaging over all the inputs passed to the NSSP instead of choosing between them. Thus, by running the simulator forward, collecting samples from the NSSPs generated from sampling the priors of each NSSP, we can make updates based on Monte Carlo estimates of $\nabla_{\tau_r}\mathcal{L}_r$.

This scheme results in $R_r(\phi_j;\tau_r) = I_r(\phi_j)$ in the limit of a large number of training samples if our neural network has sufficient capacity to exactly capture $I_r^{in}(\phi_j)$.

To see this note that

$$\mathbb{E}\left[\left(R_r(\phi_j;\tau_r) - \hat{I}_r(\phi_j)\right)^2 \middle| \phi_j\right] =$$
$$(R_r(\phi_j;\tau_r) - I_r(\phi_j))^2 + \mathbb{E}\left[\left(I_r(\phi_j) - \hat{I}_r(\phi_j)\right)^2 \middle| \phi_j\right]$$

where the second term does not depend on τ_r and the first is minimized when $R_{a_j}(\phi_j;\tau_{a_j})$=$I_{a_j}(\phi_j)$ for all ϕ_j.

Once trained, we can run inference on the approximate, unnormalized, target:

$$\hat{\gamma}(x_{1:n_x}) = \prod_{i=1, a_i \notin S_R}^{n_x} f_{a_i}(x_i;\phi_i) \prod_{k=1,b_k}^{n_y} g_{b_k}(y_k;\phi_k) \prod_{j=1,a_j \in S_R}^{n_x} \frac{\gamma_{a_j}^{in}(x_j|\phi_j)}{R_{a_j}(\phi_j;\tau)}. \tag{9}$$

It is important to note that this method never actually requires us to directly evaluate $\gamma_{a_j}^{in}(x_j|\phi_j)$, instead we introduce the variables z that implicitly produce the correct pushforward distribution on the x's. In essence we are actually defining a higher dimensional auxiliary variable problem that has the desired pushforward on the variables of interest.

3.2.1 An Adjusted Approach for Nested Rejection Samplers

In the case where a NSSP is given by a rejection sampler, then it is preferable to slightly adjust the procedure for Method 2 to utilize the ability of rejection samplers to unbiasedly estimate $1/I(\phi)$.

Here we have $I(\phi) = \mathbb{E}[\mathbb{I}(A(z,\phi))]$ where $A(z,\phi)$ represents the rejection criterion, returning true if the sample is to be accepted, and the expectation is with respect to running a single iteration of the rejection sampling loop. The naive Monte Carlo estimate

$$I(\phi) \approx \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}(A(z_n,\phi) = 1), \tag{10}$$

is only unbiased, if N is independent of the z_n, which is not the case for a standard rejection sampler.

Typically, one would like to instead run the rejection sampler in the standard manner: running the sampler until a sample is accepted, at which point we have generated N_a samples. Here N_a is not independent of the z_n, such that the naive estimator in (10) is now biased. However, instead fixing N upfront can return an estimate $\hat{I}(\phi) = 0$ which will in turn can cause the estimate of the normalized density to become infinite, thereby triggering a failure in the overall inference.

This conundrum can be circumvented by instead trying to directly estimate $1/I(\phi)$ and use this as the basis for the regressor. This is possible because rejection samplers have the property $\mathbb{E}[N_a|\phi] = 1/I(\phi)$ as follows:

$$\mathbb{E}[N_a|\phi] = \mathbb{E}\left[\sum_{n=1}^{N_a} 1 \middle| \phi\right] = \mathbb{E}\left[\sum_{n=1}^{\infty} \mathbb{I}(N_a \geq n) \middle| \phi\right]$$
$$= \sum_{n=1}^{\infty} \mathbb{E}\left[\mathbb{I}(N_a \geq n)|\phi\right] = \sum_{n=0}^{\infty}(1 - I(\phi))^n = \frac{1}{I(\phi)}.$$

Therefore, in this setting we learn our regressor R_{a_j} to go from ϕ_j to $1/I(\phi)$, exploiting the fact that N_a is an unbiased estimate of the latter, and subsequently use

$$P_{a_j}^{in}(x_j|\phi_j) \approx \gamma_{a_j}^{in}(x_j|\phi_j) R_{a_j}(\phi_j;\tau_{a_j}) \tag{11}$$

to construct the approximate objective.

It is interesting to further note that

$$\mathbb{E}[\gamma_{a_j}^{in}(x_j|\phi_j) N_a | x_j, \phi_j] = P_{a_j}^{in}(x_j|\phi_j) \tag{12}$$

such that it should in principle also be possible to use this result to develop pseudo-marginal samplers.

4 Experiments

To confirm their ability to provide accurate and efficient inference, we now test our methods on an artificial model where we can easily calculate the ground truth, introduce NSSPs of both types, and adjust the difficulty of the problem by varying its dimensionality. Though simple, we will see that this model is beyond what can be tackled by previous approaches (assuming we do not exploit the analytic solutions). We emphasize here that because existing approaches for dealing with NSSPs are so limited, the only viable way of accurately asserting the performance of approaches is to manually construct a problem to permit analytic simplifications to allow ground truth calculations; hence the artificial nature of these experiments. Further experiments are given in the supplement.

To be more specific, our model comprises of a multivariate Gaussian unknown mean problem, but with a twist as we write the model using NSSPs. This model is chosen for two reasons. First, via conjugacy relationships we can analytically calculate the posterior mean of the outer program. Second, it allows us to easily replace individual variables in the problem with NSSPs of either type Case A or Case B (or both) without changing the ground truth posterior.

4.1 Model Definition

Let $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\boldsymbol{x} \in \mathbb{R}^{n_x}$. We set $\boldsymbol{\mu}$ to a fixed value, randomly generated upfront. $\boldsymbol{\Sigma}$ is also randomly

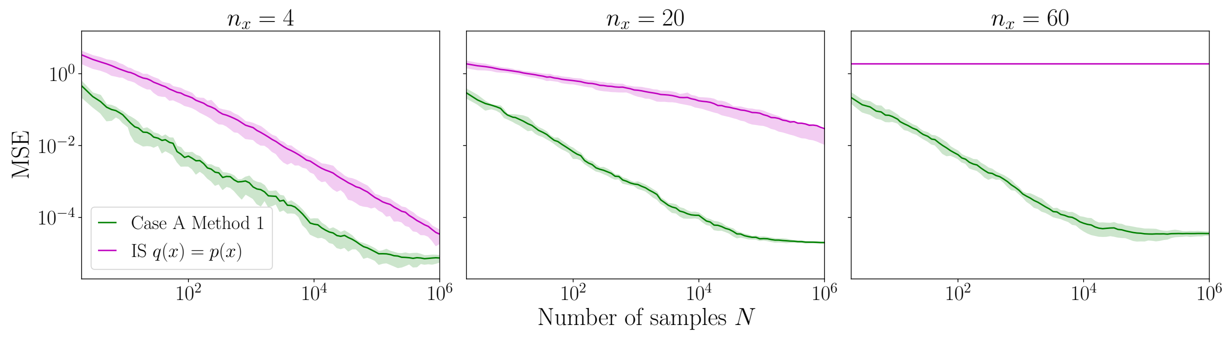

Figure 1: Convergence of the MSE for posterior mean calculated across all latent dimensions, for different problem dimensionalities for the Nested Gaussian model Case A. Shown is importance sampling from the prior (magenta) and our surrogate method (i.e., Method 1, green). The 25%-75% quantiles are shown as shaded regions, formed using 100 independently run chains of 10^6 samples per chain (after burn-in). We see that our method produces significantly lower error than the baseline, particularly as the dimensionality increases.

generated, but in a particular manner that ensures that the following depending structure holds

$$p(\boldsymbol{x}_{1:n_x}) = p(x_1)p(x_2|x_1)\dots p(x_{n_x}|x_{n_x-1}).$$

Details on how this is done are given in the supplement. Given this induced dependency structure, the forward call of the simulator can be written out as

$$\begin{aligned} p(x_1) &= f_{a_1}(x_1|\phi_1) = \mathcal{N}(x_1; \mu_1, \Sigma_{1,1}) \\ p(x_i|x_{i-1}) &= f_{a_i}(x_i|\phi_i) \\ &= \mathcal{N}(x_i; \mu_{i|i-1}(x_{i-1}), \Sigma_{i|i-1}(x_{i-1})), \end{aligned}$$

where $\mu_{i|i-1}(x_{i-1})$ and $\Sigma_{i|i-1}(x_{i-1})$ are conditional means and covariances calculated using standard Gaussian identities (Petersen and Pedersen, 2012). Finally, we introduce a single Gaussian distributed observation $\boldsymbol{y} \in \mathbb{R}^{n_x}$ of the form $g_{b_1}(\boldsymbol{y}|x_{1:n_x}) = \mathcal{N}(\boldsymbol{y}; x_{1:n_x}, I)$, where I is an identity matrix.

To induce nested structures into this model, for each of even addresses we replace $f_{a_i}(x_i|\phi_i)$ with an NSSPs (i.e., if i is even then $a_i \in S_r$, else, $a_i \notin S_r$). These NSSPs implicitly define the same density, but we either only provide a black–box sampler (Case A), or in an input–dependent unnormalized form (Case B).

For the former case, we simply make an external call to a function that returns samples according to $\mathcal{N}(x_i; \mu_{i|i-1}(x_{i-1}), \Sigma_{i|i-1}(x_{i-1}))$, but for which we cannot directly evaluate the density. For the latter case, we define a set of nested probabilistic programs

```
def NSSP_i($\phi_i = x_{i-1}$)
    $z \sim \mathcal{N}(\mu_i, \Sigma_{i,i})$
    $\mu_{i-1|i} = \mu_{i-1} + (z - \mu_i)\Sigma^2_{i-1,i}/\Sigma_{i,i}$
    $\Sigma_{i-1|i} = \Sigma_{i-1,i-1} - \Sigma^2_{i-1,i}/\Sigma_{i,i}$
    factor($\mathcal{N}(\phi_i; \mu_{i-1|i}, \Sigma_{i-1|i})$)
return  $z$
```

where `factor` represents a factoring of the density, i.e. this is a likelihood term of density $\mathcal{N}(\phi_i; \mu_{i-1|i}, \Sigma_{i-1|i})$. Drawing from this NSSP requires us to run separate inference procedures, because the nested model involves a local normalization (Rainforth, 2018). We can consider the nested sub-procedure in isolation and its inputs as fixed variables when calculating the form of its local posterior, but this posterior must be estimated separately for each possible instance of the inputs. Critically, we can also estimate the marginal likelihood of this nested program for input ϕ_i by drawing samples of z and then taking the average of the likelihood evaluations (i.e. the average of the exponential of the factor statements). This thus lets us carry out Method 2.

4.2 Evaluation

We now consider running Method 1 on the Case A variation of our model and Method 2 on the Case B variation. Note that Method 2 cannot be run for problems that are only of type Case A, while Method 1 will generally be inferior to Method 2 when the latter can be run (such that we do not generally recommend doing this) because it requires us to regress from the inputs to a full distribution, rather than just the scalar normalizing constant.[2] All the same, results for running Method 1 on Case B are given in the supplement. We note that for both methods the real time spent training the approximations was comparable to that for running the subsequent MCMC sampling.

Recall that in the training phase of Method 1 we require only the input–output pairs from the NSSPs; these can

[2]A potential exception to this is if the individual NSSPs contain a large number of internal random variables z (Section 3.2), as here the final MCMC sampling for Method 2 is on a much higher dimensional space than for Method 1.

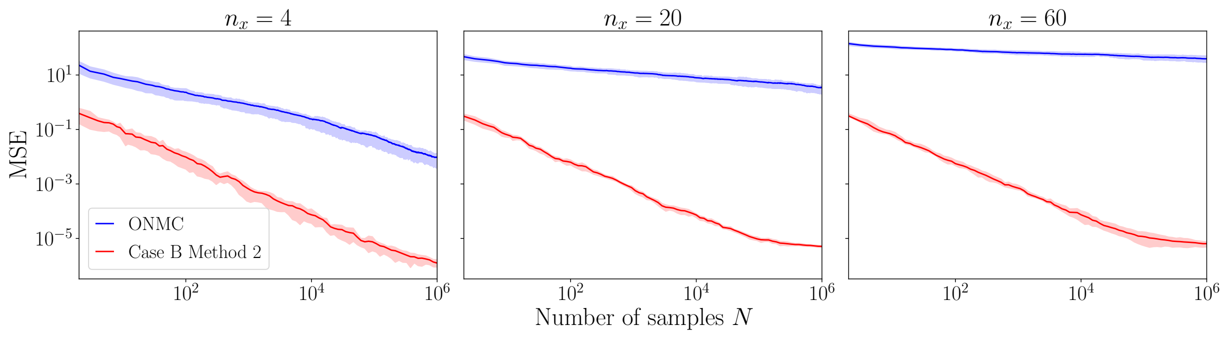

Figure 2: Convergence of the MSE for posterior mean calculated across all latent dimensions, for different problem dimensionalities for the Nested Gaussian model Case B. Shown is an online nested importance sampling estimate (blue) and our normalization approximation method (i.e., Method 2, red). Conventions as per Fig. 1.

all be directly generated by forward simulation. This can also be straightforwardly batched so training is efficient. Once trained we run Metropolis-Hastings (MH) over the whole simulator with the NSSPs replaced by their approximations. Full details of the NN architectures and training procedure are provided in Appendix B. We compare to a baseline of importance sampling from the prior as described in Section 2.2, noting that neither particle methods nor ABC approaches are helpful in this setting; despite the crudeness of approach, we are not aware of any stronger baselines in the literature because of the unusual nature of the problem. The results are given in Figure 1, where we use our ground truth for the posterior mean of $\boldsymbol{x}$ (see Appendix C) to calculate a mean squared error (MSE) for variants of the model with different dimensionality. We use a burn in period of 10^4 MCMC samples (not included in the plot). We see that our approach significantly outperforms the baseline, while this improvement becomes more noticeable as the dimensionality increases. In fact, by the time $n_x = 60$, the baseline effectively fails to generate any good samples of note, while our approach maintains its performance.

Training of Method 2, on the other hand, requires each NSSP to return both marginal likelihood estimate for training its regressor, and an approximate sample of its posterior so that the forward sampling of the program can continue. For the former, we draw a number of samples the NSSP's local prior (i.e. numerous z) and return the average likelihood evaluations resulting from the factor statements, giving an unbiased estimate of the marginal likelihood. We then draw one of these samples in proportion to its "weight" (i.e. its likelihood evaluation) as the returned sample, in a manner akin to self-normalized importance sampling with resampling. We note here that these returned samples do not need to be exact posterior samples for effective training (and indeed they are not): we are only doing this sampling to generate example inputs for training the regressors of later NSSPs. Again estimation steps can be batched, so training is efficient. Full training details are again provided in Appendix B.2.

Figure 2 shows the results for this case, where we now instead compare to the online nested importance sampling (ONMC) approach of Rainforth (2018), noting that, to the best of our knowledge, nesting importance sampling approaches (or slight variations therein, e.g. Naderiparizi et al. (2019)) are the only general–purpose consistent estimators for this class of problems currently present in the literature. We again see that our approach provides substantial improvements, with these again becoming more pronounced as the dimensionality increases.

It is noticeable that both our methods converge to biased solutions as we increase the number of samples at inference time. However, this is to be expected as they are based on approximating the density of the NSSP. Critically though, this error can be reduced by performing more training of the approximation networks, increasing their capacity, or using more accurate estimates in the training procedure (e.g. more importance samples for each NSSP when training Method 2). The key is that the methods are providing effective estimates, even for the higher dimensional problems where the baselines break down completely.

5 Conclusions and Future Work

We have introduced two approaches for performing effective inference in simulators containing nested stochastic sub-procedures (NSSPs). These are both based on first separately approximating each individual NSSP—either using an amortized inference surrogate on its outputs or a combination of the original program and a regressed approximation of its marginal likelihood from its inputs—and then using these approximations to form an approximation for the overall program that can be used as a target for scalable inference algorithms like MCMC and variational inference. We have shown that this provides substantial gains over existing baselines on a synthetic model where we can analytically derive the ground truth. In particular, our approaches have allowed us to tackle problems of a much higher dimension that those which can be efficiently tackled by existing approaches.

Our work provides a promising pathway to perform statistically principled and computationally efficient inference in large scale simulators. Indeed there are a host of current simulator-based inference problems currently actively being researched in the literature where the bottleneck is the restrictions that NSSPs impose on our ability to run inference (Baydin et al., 2019a; Gram-Hansen et al., 2019). We hope that this work will substantially increase the viability of such ventures, thereby leading to a large number of downstream applications.

References

Atılım Güneş Baydin, Lukas Heinrich, Wahid Bhimji, Lei Shao, Saeid Naderiparizi, Andreas Munk, Jialin Liu, Bradley Gram-Hansen, Gilles Louppe, Lawrence Meadows, Philip Torr, Victor Lee, Prabhat, Kyle Cranmer, and Frank Wood. Efficient probabilistic inference in the quest for physics beyond the standard model. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2019a.

Atılım Güneş Baydin, Lei Shao, Wahid Bhimji, Lukas Heinrich, Lawrence F. Meadows, Jialin Liu, Andreas Munk, Saeid Naderiparizi, Bradley Gram-Hansen, Gilles Louppe, Mingfei Ma, Xiaohui Zhao, Philip Torr, Victor Lee, Kyle Cranmer, Prabhat, and Frank Wood. Etalumis: Bringing probabilistic programming to scientific simulators at scale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, 2019b. Association for Computing Machinery. ISBN 9781450362290. doi: 10.1145/3295500.3356180.

Anna Bershteyn, Jaline Gerardin, Daniel Bridenbecker, Christopher W Lorton, Jonathan Bloedow, Robert S Baker, Guillaume Chabot-Couture, Ye Chen, Thomas Fischle, Kurt Frey, et al. Implementation and applications of emod, an individual-based multi-disease modeling platform. *Pathogens and Disease*, 76(5):fty059, 2018.

Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, 2019.

Katalin Csilléry, Michael GB Blum, Oscar E Gaggiotti, and Olivier François. Approximate Bayesian computation (ABC) in practice. *Trends in Ecology & Evolution*, 25(7):410–418, 2010.

Valentina Di Pasquale, Salvatore Miranda, Raffaele Iannone, and Stefano Riemma. A simulator for human error probability analysis. *Reliability Engineering & System Safety*, 139:17–32, 2015.

Arnaud Doucet, Nando De Freitas, and Neil Gordon. An introduction to sequential monte carlo methods. In *Sequential Monte Carlo methods in practice*, pages 3–14. Springer, 2001.

Neil M Ferguson, Derek AT Cummings, Christophe Fraser, James C Cajka, Philip C Cooley, and Donald S Burke. Strategies for mitigating an influenza pandemic. *Nature*, 442(7101):448–452, 2006.

Gersende Fort, Emmanuel Gobet, and Eric Moulines. MCMC design-based non-parametric regression for rare event. application to nested risk computations. *Monte Carlo Methods and Applications*, 23(1):21–42, 2017.

Adam Foster, Martin Jankowiak, Elias Bingham, Paul Horsfall, Yee Whye Teh, Thomas Rainforth, and Noah Goodman. Variational Bayesian optimal experimental design. In *Advances in Neural Information Processing Systems*, pages 14036–14047, 2019.

Tanju Gleisberg, Stefan Höche, F Krauss, M Schönherr, S Schumann, F Siegert, and J Winter. Event generation with SHERPA 1.1. *Journal of High Energy Physics*, 2009(02):007, 2009.

Noah Goodman, Vikash Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. *arXiv preprint arXiv:1206.3255*, 2012.

Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*, pages 167–181. ACM, 2014.

Bradley Gram-Hansen, Christian Schröder de Witt, Tom Rainforth, Philip HS Torr, Yee Whye Teh, and Atılım Güneş Baydin. Hijacking malaria simulators with probabilistic programming. *ICML Workshop on AI for Social Good*, 2019.

Katalin M Hangos and Ian T Cameron. *Process modelling and model analysis*, volume 4. Academic press London, 2001.

Dieter W Heermann. Computer-simulation methods. In *Computer Simulation Methods in Theoretical Physics*, pages 8–12. Springer, 1990.

Isaac M Held. The gap between simulation and understanding in climate modeling. *Bulletin of the American Meteorological Society*, 86(11):1609–1614, 2005.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

Tuan Anh Le, Atılım Güneş Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54 of *Proceedings of Machine Learning Research*, pages 1338–1348, Fort Lauderdale, FL, USA, 2017. PMLR.

Mario Lezcano Casado, Atılım Güneş Baydin, David Martinez Rubio, Tuan Anh Le, Frank Wood, Lukas Heinrich, Gilles Louppe, Kyle Cranmer, Wahid Bhimji, Karen Ng, and Prabhat. Improvements to inference compilation for probabilistic programming in large-scale scientific simulators. In *Neural Information Processing Systems (NIPS) 2017 workshop on*

Deep Learning for Physical Sciences (DLPS), Long Beach, CA, US, December 8, 2017, 2017.

Theofrastos Mantadelis and Gerda Janssens. Nesting probabilistic inference. *arXiv preprint arXiv:1112.3785*, 2011.

Jean-Michel Marin, Pierre Pudlo, Christian P Robert, and Robin J Ryder. Approximate Bayesian computational methods. *Statistics and Computing*, 22(6): 1167–1180, 2012.

Iain Murray, Zoubin Ghahramani, and David JC MacKay. MCMC for doubly-intractable distributions. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pages 359–366. AUAI Press, 2006.

Iain Andrew Murray. *Advances in Markov chain Monte Carlo methods*. PhD thesis, University of London, 2007.

Saeid Naderiparizi, Adam Ścibior, Andreas Munk, Mehrdad Ghadiri, Atılım Güneş Baydin, Bradley Gram-Hansen, Christian Schroeder de Witt, Robert Zinkov, Philip H. S. Torr, Tom Rainforth, Yee Whye Teh, and Frank Wood. Amortized rejection sampling in universal probabilistic programming. *arXiv:1910.09056 [cs, stat]*, 2019.

Brooks Paige and Frank Wood. Inference networks for sequential Monte Carlo in graphical models. In *International Conference on Machine Learning*, pages 3040–3049, 2016.

Padmavathi Patlolla, Vandana Gunupudi, Armin R Mikler, and Roy T Jacob. Agent-based simulation tools in computational epidemiology. In *International Workshop on Innovative Internet Community Systems*, pages 212–223. Springer, 2004.

Kaare Brandt Petersen and Michael Syskind Pedersen. The Matrix Cookbook. *https://archive.org/details/imm3274/mode/2up*, 2012.

J. K. Pritchard, M. T. Seielstad, A. Perez-Lezaun, and M. W. Feldman. Population growth of human y chromosomes: a study of y chromosome microsatellites. *Molecular Biology and Evolution*, 16(12):1791–1798, 1999. ISSN 0737-4038. doi: 10.1093/oxfordjournals.molbev.a026091.

Thomas William Gamlen Rainforth. *Automating inference, learning, and design using probabilistic programming*. PhD thesis, University of Oxford, 2017.

Tom Rainforth. Nesting probabilistic programs. *arXiv preprint arXiv:1803.06328*, 2018.

Tom Rainforth, Rob Cornish, Hongseok Yang, Andrew Warrington, and Frank Wood. On nesting Monte Carlo estimators. In *International Conference on Machine Learning*, pages 4267–4276, 2018.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on International Conference on Machine Learning-Volume 32*, pages II–1278, 2014.

Daniel Ritchie, Paul Horsfall, and Noah D Goodman. Deep amortized inference for probabilistic programs. *arXiv preprint arXiv:1610.05735*, 2016.

T Smith, N Maire, A Ross, M Penny, N Chitnis, A Schapira, A Studer, B Genton, C Lengeler, Fabrizio Tediosi, et al. Towards a comprehensive simulation model of malaria epidemiology and control. *Parasitology*, 135(13):1507–1516, 2008.

Andreas Stuhlmüller and Noah D Goodman. Reasoning about reasoning by nested conditioning: Modeling theory of mind with probabilistic programs. *Cognitive Systems Research*, 28:80–99, 2014.

Mikael Sunnåker, Alberto Giovanni Busetto, Elina Numminen, Jukka Corander, Matthieu Foll, and Christophe Dessimoz. Approximate bayesian computation. *PLoS Comput Biol*, 9(1):e1002803, 2013.

S. Tavare, D. J. Balding, R. C. Griffiths, and P. Donnelly. Inferring coalescence times from DNA sequence data. *Genetics*, 145(2):505–518, 1997. ISSN 0016-6731.

Dustin Tran, Matthew D Hoffman, Rif A Saurous, Eugene Brevdo, Kevin Murphy, and David M Blei. Deep probabilistic programming. *arXiv preprint arXiv:1701.03757*, 2017.

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming. *arXiv:1809.10756 [cs, stat]*, 2018.

Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pages 1024–1032, 2014.

A Locating Addresses in Simulators

Running the introduced inference approaches requires one to identify calls in the simulator corresponding to NSSPs. Often this can be done manually as these will be known ahead of time. However, in certain cases, particularly for large scale simulators or probabilistic programs containing rejection sampling loops, it may not actually be trivial to manually identify and then replace these NSSPs in the code to allow our approach to be used.

Thankfully, there has been substantial recent progress on how this can be done in the literature through the concept of *hijacking* a simulator. Namely, recent work by Gram-Hansen et al. (2019); Baydin et al. (2019a) described a hijacking process that takes an arbitrary stochastic simulator, defines a joint density, and hijacks the underlying random primitives of that simulator by passing their control to a probabilistic programming system (PPS). This backend PPS can then both manipulate the running of this simulator by overwriting calls to the random primitives and also track the control flows of the simulator to identify rejection sampling loops and other NSSPs. Thus, in turn, these methods should, at least in principle, be able to automate the running of our inference approaches.

Viewed from another perspective, these hijacking based PPS approaches are often used to tackle problems of the same class as our work, but at present their backend inference engines rely on simple importance sampling procedures. As such, they are not sufficiently powerful to satisfactorily perform inference in the types of simulators they are designed to target, such as the OpenMalaria simulator (Smith et al., 2008) and the prominent Covid-19 simulator of Ferguson et al. (2006). The techniques developed in this paper provide a pathway to running inference in models like this where this was previously completely infeasible, due to their ability to scale gracefully with the dimensionality of the model, while existing approaches scale exponentially poorly and quickly become redundant.

B Further Experiments and Experimental Details

B.1 Method 1 for Case B Problems

Method 2 (i.e. regressing the marginal likelihood) is typically preferable when all our NSSPs conform to Case B but not Case A (i.e. we have access to their unnormalized densities, but cannot generate samples directly). This is because a) it is simpler to regress to a one dimensional marginal likelihood that a set of variational parameters that approximate the output distribution, and b) it does not introduce additional approximations from running inference on the individual NSSPs during training (as our marginal likelihood estimates will generally be unbiased, even if they are high variance). However, this is not universally the case: Method 1 (i.e. learning an amortized surrogate density) can still sometimes be preferable in Case B scenarios, particular if in the individual NSSPs are high–dimensional themselves. This is because a) the final MCMC inference is lower dimensional because it does not include the *internal* random draws of the NSSPs, and b) sometimes producing approximate posterior samples from a Case B NSSP is easier than producing a reasonable marginal likelihood estimate (e.g. if the NSSP is internally high–dimensional it might be feasible to generate good approximate output samples by MCMC, but not low–variance marginal likelihood estimates through importance sampling).

In this section, we consider running Method 2 for NSSPs which are only of type Case B. For this, we use exactly the same Nested Gaussian model as the one described in Section 4. We assume that the model is specified using the nested probabilistic programs `NSSP_i`, but that we cannot draw samples directly from them. It is still possible to apply Method 1 here by treating the problem as a nested inference. That is, whenever we encounter a NSSP during training, we run a local inference to generate the required (approximate) output samples associated with the provided input, that is approximate samples from the posterior distribution of `NSSP_i`(ϕ_i) given input ϕ_i. Sampling from the posterior of each NSSP constitutes an separate, one-dimensional, non-nested inference problem, and we will need to run inference on each NSSP we encounter in the forward run of the simulator in the process of generating the samples for training of our conditional density estimators.

Among many inference methods we could use to solve such an inference problem, we decide to use Metropolis-Hastings. The details of the training procedure are given in Appendix B.2, and are analogous to both Method 1 for Case A and Method 2 for Case B experiments in the main paper in all aspects they share.

In Figure 3 we compare to the same ONMC baseline considered in the main paper. We see that the performance is similar to when we used Method 2, but with slightly higher final errors.

B.2 Training details for the Nested Gaussian

In this section we give the details of the training schemes used. We first give the details common for all the models we trained, and then describe model-specific details for individual models in subsections.

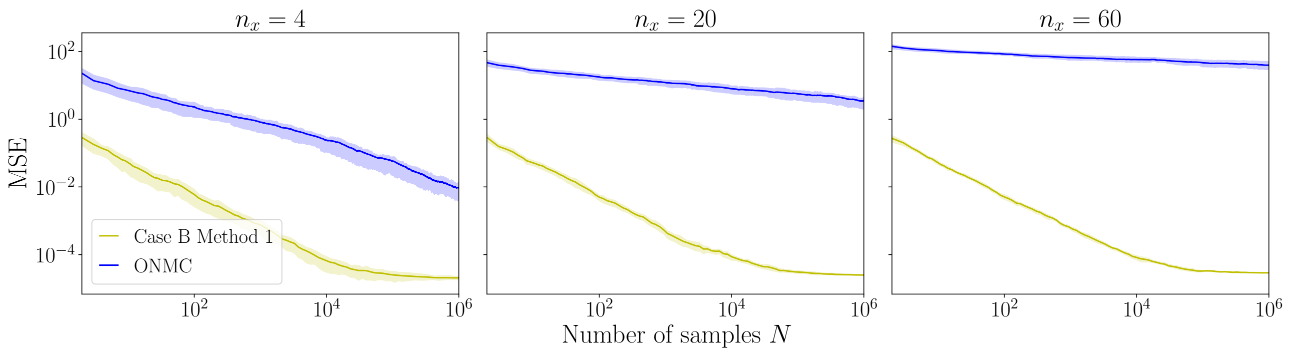

Figure 3: Convergence of the MSE for posterior mean calculated across all latent dimensions, for different problem dimensionalities for the Nested Gaussian model Case B. Shown is nested importance sampling (blue) and the surrogate method (i.e., Method 1 for Case B, yellow). The 25%-75% quantiles, shaded regions, are over 100 independently run chains of 10^6 samples per chain. As expected, we see that both our approach produces significantly lower error that the baseline as the dimensionality increases.

All the neural networks used for determining the parameters of the conditional density estimator for Method 1 (i.e., mapping from the value of ϕ_i to the parameters of the parametric density estimator approximating the posterior distribution of the NSSP) and regressing the marginal likelihood of the NSSP for Method 2 (i.e., mapping from the value of ϕ_i to the approximation of the marginal likelihood of the NSSP) were MLPs with 64 hidden units per layer and 4 hidden layers.

We trained a separate density estimator (Method 1) or regressor (Method 2) for each instance of the NSSP which means that we trained 2, 10, 30 separate neural networks for $d = 4, 20, 60$, respectively.

We trained for 10^4 gradient updates, each with newly generated batch of data from the forward run of the program. Each batch had 10^3 examples. The initial learning rate for all experiments was 10^{-3}. We used `ReduceLROnPlateau` learning rate scheduler with patience of 10^3 and decrease factor of 0.29.

Final inference on the program with the probability density of the NSSPs approximated using the learned components (for both Method 1 and Method 2) were run using Metropolis-Hastings with isotropic Gaussian proposal, with proposal standard deviations $1.0, 0.4, 0.2$ for $d = 4, 20, 60$, respectively. The MCMC chains were initiated at a vector of all zeros and used 10^4 samples for burn in (not included in plots).

B.2.1 Method 1 for Case A

The distribution family used for amortized surrogates was taken to just be a one dimensional Gaussian, $\mathcal{N}(m(\phi), s(\phi))$, where m and s are MLPs as described earlier that take in the NSSP input ϕ and return a mean and standard deviation respectively. As such, this approximation family is able to encapsulate the true NSSP conditional distributions exactly if sufficiently accurate MLPs can be learned. Training these conditional density estimators was done as explained in the main paper.

B.2.2 Method 2 for Case B

We obtain a Monte Carlo estimate of the marginal likelihood for each instance of the NSSP by taking 10^4 samples of z in the `NSSP_i`(ϕ) and forming an average over the evaluations of the likelihood (given by the exponentiation of the `factor`$(\cdot)$ statement). This estimation is trivially vectorized on a GPU, which means that taking this relatively large number of samples in the estimate is only nominally slower than it would be to only use a single sample. As such, the training phase of Method 2 is generally much quicker than one might expect.

B.2.3 Method 1 for Case B

The conditional density estimators used were the same as per Section B.2.1. As described in Section B.1, in this setting we run separate MCMC inference on each instance of the NSSP. We ran a separate chain for each element in the training batch. We ran Metropolis-Hastings with an isotropic Gaussian proposal, with proposal standard deviation of 3.0 which yielded an average acceptance rate around 44%. The chains were initialized by using the conditional density estimator. Each chain was run for 25 samples, with only the last sample presented to the density estimator as a training sample.

C Derivation of the Ground Truth

We now derive the ground truth posterior for the Gaussian model used in the main paper. The key here is to reverse engineer the prior covariance, $\mathbf{\Sigma}$, in such a way that the following Markov dependency structure will hold

$$p(\boldsymbol{x}_{1:n_x}) = p(x_1)p(x_2|x_1)\dots p(x_{n_x}|x_{n_x-1}).$$

This then allows us to replace any arbitrary $p(x_i|x_{i-1})$ with an NSSP for the form given in the main paper (or equivalently a direct sampler) without requiring any information from the $< i-1$ indices.

To show how this can be done, let us start by noting the standard results for the marginals of an arbitrary Gaussian (Petersen and Pedersen, 2012). Namely, if $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$ and

$$\boldsymbol{x} = \begin{bmatrix}\boldsymbol{x}_a \\ \boldsymbol{x}_b\end{bmatrix}, \ \boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b\end{bmatrix}, \ \mathbf{\Sigma} = \begin{bmatrix}\mathbf{\Sigma}_a & \mathbf{\Sigma}_c \\ \mathbf{\Sigma}_c^T & \mathbf{\Sigma}_b\end{bmatrix},$$

then $\boldsymbol{x}_b|\boldsymbol{x}_a \sim \mathcal{N}(\boldsymbol{\mu}_{b|a}, \mathbf{\Sigma}_{b|a})$, where

$$\boldsymbol{\mu}_{b|a} = \boldsymbol{\mu}_b + \mathbf{\Sigma}_c^T\mathbf{\Sigma}_a^{-1}(\boldsymbol{x}_a - \boldsymbol{\mu}_a)$$
$$\mathbf{\Sigma}_{b|a} = \mathbf{\Sigma}_b - \mathbf{\Sigma}_c^T\mathbf{\Sigma}_a^{-1}\mathbf{\Sigma}_c.$$

Now taking $\boldsymbol{x}_b = \boldsymbol{x}_i$ and $\boldsymbol{x}_a = \boldsymbol{x}_{1:i-1}$, we thus have $\boldsymbol{x}_i|\boldsymbol{x}_{1:i-1} \sim \mathcal{N}(\boldsymbol{\mu}_{i|1:i-1}, \mathbf{\Sigma}_{i|1:i-1})$ where

$$\boldsymbol{\mu}_{i|1:i-1} = \boldsymbol{\mu}_i + \mathbf{\Sigma}_{1:i-1,i}^T\mathbf{\Sigma}_{1:i-1,1:i-1}^{-1}(\boldsymbol{x}_{1:i-1} - \boldsymbol{\mu}_{1:i-1})$$
$$\mathbf{\Sigma}_{i|1:i-1} = \mathbf{\Sigma}_{i,i} - \mathbf{\Sigma}_{1:i-1,i}^T\mathbf{\Sigma}_{1:i-1,1:i-1}^{-1}\mathbf{\Sigma}_{1:i-1,i}.$$

For our induced dependency structure to hold, we require $\boldsymbol{\mu}_{i|1:i-1} = \boldsymbol{\mu}_{i|i-1}$ and $\mathbf{\Sigma}_{i|1:i-1} = \mathbf{\Sigma}_{i|i-1}$, where $\boldsymbol{\mu}_{i|i-1}$ and $\mathbf{\Sigma}_{i|i-1}$ are derived by instead taking $\boldsymbol{x}_a = \boldsymbol{x}_{i-1}$ to yield

$$\boldsymbol{\mu}_{i|i-1} = \boldsymbol{\mu}_i + (\boldsymbol{x}_{i-1} - \boldsymbol{\mu}_{i-1})\mathbf{\Sigma}_{i-1,i}/\mathbf{\Sigma}_{i-1,i-1}$$
$$\mathbf{\Sigma}_{i|i-1} = \mathbf{\Sigma}_{i,i} - \mathbf{\Sigma}_{i-1,i}^2/\mathbf{\Sigma}_{i-1,i-1}.$$

We thus need the following equations to hold for all $x_{1:i-2}$

$$\begin{aligned}\mathbf{\Sigma}_{1:i-1,i}^T\mathbf{\Sigma}_{1:i-1,1:i-1}^{-1}&(\boldsymbol{x}_{1:i-1} - \boldsymbol{\mu}_{1:i-1}) \\ &= (\boldsymbol{x}_{i-1} - \boldsymbol{\mu}_{i-1})\mathbf{\Sigma}_{i-1,i}/\mathbf{\Sigma}_{i-1,i-1}\end{aligned} \quad (13)$$

$$\mathbf{\Sigma}_{1:i-1,i}^T\mathbf{\Sigma}_{1:i-1,1:i-1}^{-1}\mathbf{\Sigma}_{1:i-1,i} = \mathbf{\Sigma}_{i-1,i}^2/\mathbf{\Sigma}_{i-1,i-1}. \quad (14)$$

Considering the first of these, we see that we must have

$$\mathbf{\Sigma}_{1:i-1,i}^T\mathbf{\Sigma}_{1:i-1,1:i-1}^{-1} = \left[\mathbf{0}, \frac{\mathbf{\Sigma}_{i-1,i}}{\mathbf{\Sigma}_{i-1,i-1}}\right],$$

noting that the $\mathbf{0}$ term originates from the fact that changing $x_{1:i-2}$ must not lead to changes in the left–hand side of (13). Rearranging gives

$$\mathbf{\Sigma}_{1:i-1,i}^T = \left[\mathbf{0}, \frac{\mathbf{\Sigma}_{i-1,i}}{\mathbf{\Sigma}_{i-1,i-1}}\right]\mathbf{\Sigma}_{1:i-1,1:i-1} \quad (15)$$

such that we can construct $\mathbf{\Sigma}_{1:i-1,i}^T$ from $\mathbf{\Sigma}_{1:i-1,1:i-1}$ and $\mathbf{\Sigma}_{i-1,i}$ using a matrix multiplication. Simple substitution into the left–hand size of (14) shows that this also provides a solution to our second required equality as well. Thus any $\mathbf{\Sigma}$ that satisfies (15) for all i will produce our desired dependency relationship and further ensure that the NSSP used in the main paper defines the desired target density.

We can construct such a $\mathbf{\Sigma}$ by the following process:

Algorithm 1 Generate $\mathbf{\Sigma}$

```
1: Generate $\mathbf{\Sigma}_{1,1}$
2: for $i = 2$ to $n_x$ do
3:     Generate $\mathbf{\Sigma}_{i,i}$ and $\mathbf{\Sigma}_{i-1,i}$
4:     $\mathbf{\Sigma}_{i,1:i-1} \leftarrow \left[\mathbf{0}, \frac{\mathbf{\Sigma}_{i-1,i}}{\mathbf{\Sigma}_{i-1,i-1}}\right]\mathbf{\Sigma}_{1:i-1,1:i-1}$
5:     $\mathbf{\Sigma}_{1:i-1,i} \leftarrow \mathbf{\Sigma}_{i,1:i-1}^T$
6: end for
```

Note that the reassignments do not change the values of $\mathbf{\Sigma}_{i-1,i}$ as there is a canceling that ensures this remains the same. Here the $\mathbf{\Sigma}_{i,i}$ can be generated in completely arbitrary manner—provided they are positive (e.g. we could sample them from a gamma distribution)—but the generation of $\mathbf{\Sigma}_{i-1,i}$ must be done carefully to ensure that the covariance matrix remains positive–definite. This is achieved by ensuring that $\mathbf{\Sigma}_{i-1,i}^2 < \mathbf{\Sigma}_{i,i}\mathbf{\Sigma}_{i-1,i-1}$ so that all the conditional covariances are positive.

We generated $\mathbf{\Sigma}_{i,i}$ and $\mathbf{\Sigma}_{i-1,i}$ as following: $\epsilon_i \sim \text{Uniform}[0,1]$, $\mathbf{\Sigma}_{i,i} \leftarrow 1 + \epsilon_i$, and $r_i \sim \text{Uniform}[0, 0.7]$, $\mathbf{\Sigma}_{i-1,i} \leftarrow r_i \cdot \mathbf{\Sigma}_{i-1,i-1}$, while the mean $\boldsymbol{\mu}$ of our joint distribution was generated from a standard multivariate normal $\boldsymbol{\mu} \sim \mathcal{N}(0, I)$. This generation process is performed only once for each dimensionality of the problem $n_x = 4, 20, 60$, and then those 3 matrices $\mathbf{\Sigma}$ are reused for all of our experiments.

To finish our derivation of the ground truth we now simply need to incorporate our likelihood term $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \boldsymbol{x}, \mathbf{\Sigma}_{\boldsymbol{y}})$ (where we take $\mathbf{\Sigma}_{\boldsymbol{y}} = I$ in practice). This is just a standard Gaussian unknown mean problem, yielding $\boldsymbol{x}|\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{\boldsymbol{x}|\boldsymbol{y}}, \mathbf{\Sigma}_{\boldsymbol{x}|\boldsymbol{y}})$ where (Murphy, 2012)

$$\boldsymbol{\mu}_{\boldsymbol{x}|\boldsymbol{y}} = \left(\mathbf{\Sigma}^{-1} + \mathbf{\Sigma}_{\boldsymbol{y}}^{-1}\right)^{-1}\left(\mathbf{\Sigma}^{-1}\boldsymbol{\mu} + \mathbf{\Sigma}_{\boldsymbol{y}}^{-1}\boldsymbol{y}\right) \quad (16)$$

$$\mathbf{\Sigma}_{\boldsymbol{x}|\boldsymbol{y}} = \left(\mathbf{\Sigma}^{-1} + \mathbf{\Sigma}_{\boldsymbol{y}}^{-1}\right)^{-1}. \quad (17)$$

References

Atılım Güneş Baydin, Lukas Heinrich, Wahid Bhimji, Lei Shao, Saeid Naderiparizi, Andreas Munk, Jialin Liu, Bradley Gram-Hansen, Gilles Louppe, Lawrence Meadows, Philip Torr, Victor Lee, Prabhat, Kyle Cranmer, and Frank Wood. Efficient probabilistic inference in the quest for physics beyond the standard model. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2019.

Neil M Ferguson, Derek AT Cummings, Christophe Fraser, James C Cajka, Philip C Cooley, and Donald S Burke. Strategies for mitigating an influenza pandemic. *Nature*, 442(7101):448–452, 2006.

Bradley Gram-Hansen, Christian Schröder de Witt, Tom Rainforth, Philip HS Torr, Yee Whye Teh, and Atılım Güneş Baydin. Hijacking malaria simulators with probabilistic programming. *ICML Workshop on AI for Social Good*, 2019.

Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

Kaare Brandt Petersen and Michael Syskind Pedersen. The Matrix Cookbook. *https://archive.org/details/imm3274/mode/2up*, 2012.

T Smith, N Maire, A Ross, M Penny, N Chitnis, A Schapira, A Studer, B Genton, C Lengeler, Fabrizio Tediosi, et al. Towards a comprehensive simulation model of malaria epidemiology and control. *Parasitology*, 135(13):1507–1516, 2008.