Hijacking Simulators with Universal Probabilistic Programming

Paper ID: 1010

Abstract

Notes on how to develop a formalism to perform inference in population based simulators.

1 Introduction

One challenge in performing inference in such models is the combinatorial increase in *paths* that can be taken within the given probabilistic model. This is because each member of the population has a bounded, but large number of decisions to make. In addition to this, each member of the population can interact with other members of the population, which presents challenges for current inference methods. Because in many simulated settings we cannot correctly construct the functional form of the density of the desired program. This has spawned a large body of research in recent years to understand how to perform inference in likelihood free methods **? Add normalising flow citations** for conditional density estimation. However, whilst many of these methods look to generate a surragote for the entire simulator, we propose amortizing only computational expensive sub-components of the simulator, which we can do in an automated way with probabilistic programming. This enables three things. 1) It reduces the loss of information, as when simulators are amortized with in current settings we require, potentially infinite samples, to provide a useful learnt representation of the simulator. 2) It increases computational efficiency, whilst maintaining expressivity. 3) It converts expensive rejection sampling loops into to learnt functions that can be easily sampled from and used to construct a functional form of the likelihood, that enables the target density to be fully constructed, enabling one to leverage a large class of sample effient inference schemes, namely MCMC.

In particular, we focus on combining the hijacking of simulators via probabilistic programming **?** and state-of-the-art amortization schemes for density estimation **Do home work on current papers**. This enables such a process to happen in automatic way, simplifying existing set-ups, whilst providing more expressive modelling capabilities. We are able to demonstrate this by replacing rejection sampling blocks, which are utilised within many prominent simulators to simulate densities that have no well defined cumulative density function, nor no known process to efficiently sample from such a density exists. By amortizing the rejection sampling blocks, during the inference stage when a simulator is run *forword* we can replace such blocks with their leant surrogate, from which we can then construct the entire density of the simulator, which can then be utilized, *correctly*, contrary to **?**, by sample efficient Markov chain Monte Carlo inference schemes, such as Hamiltonian Monte Carlo (HMC), random-walk Metropolis Hastings (RMH), Light-weight Metropolis Hastings (LMH) and many other schemes. Whilst also significantly reducing memory consumption and computational running time of both the simulator and of the probabilistic programming system, as evaluating amortized functions is cheap and the reduction in rejection sampling blocks significantly reduces the number of unnecessary latent variables tracked **?**.

Without the amortization procedure, as updates are made sequentially MCMC methods are typically not-well suited this type of recursive estimation problem. As each time we get a new data point y_{t+1} we have to run new MCMC simulations for $P(x_{0:t_1}|y_{0:t+1})$, we also cannot directly reuse the samples $\{x^i_{0:t}\}$. However, as we can only run the simulator `forward()` we cannot return back to a previous state and re-run. So adapting existing inference techniques is the only viable option. MCMC

Preprint. Under review.

methods can be effective with a well designed amortized inference procedure (**Note: This is not guaranteed**). Non-MCMC based methods such as importance sampling (IS) and sequential monte carlo (SMC), may also be sensible inference engines, although we shall discuss the limitations within the simulator context in the proceeding sections.

However, to construct this hybrid procedure requires both substantial engineering contributions and a description of the mathematical framework for these non-standard probabilistic programs. We provide both of these in this paper. As to implement this in practice we are required to do the following:

- An automated training procedure for any given simulator.
- Extend the protocols of existing probabilistic programming systems that hijack simulators, to enable:
  1. 1) Only the necessary raw samples from the desired rejection sampling addresses to be collected for training.
  2. 2) A function at inference time that replaces those given addresses with a learnt surrogate.
- Finer grained control over the actual latent variables of interest, to avoid unnecessary memory consumption and ensure that inference problems remain tractable.

There is no doubt that implementing some fo these features will be more invasive to the underlying simulator code than previously just hijacking the raw-random number draws, but, it would enable us to attack difficulties in the inference procedures that are currently not amenable to the current approach.

2 Amortized Rejection Sampling

One common problem in traditional simulators, *programs*, outside of the standard probabilistic programming framework is that they rely heavily on rejection sampling to generate samples from the target density. Not only is rejection sampling computational inefficient, [especially if criteria in which samples are generated has been poorly calibrated] (may leave out this statement as it may make a reviewer question why would amortization work any better.), but it creates complications for inference. As the rejection sampling loops provide no explicit form of the target density, we can not simply evaluate the density and leverage statistically and computationally efficient algorithms such as MCMC sampling. This inhibits our ability to transform an arbitrary stochastic simulator into a probabilistic programming framework **?**. To this end, we derive both a mathematical and engineering based framework to facilitate the amortization of rejection sampling loops in arbitrary simulators, which provides a path way to both leverage a larger, more efficient class of inference methods and convert arbitrary stochastic simulators into probabilistic programs systems.

3 Rejection Sampling

Rejection samplers are heavily used when designing simulators [add SHERPA etc], due to their ease of use and ability to model any arbitrary distribution on $\mathbb{R}^D$, with a sufficiently well designed rejection sampling scheme **???**

In the standard setting we let X be a random variable whose density is $f(x)$ that we cannot easily sample from, but can evaluate. Using a random variable Z, whose density is $g(z)$ that we can efficiently sample from, then we can generate a sample X by sampling Z and accept Z with probability $p_{accept} = \frac{f(x)}{Mg(x)}$, which we can do until acceptance. M is constant that has a finite bound, $M \in (1, \infty)$, on the likelihood ratio $\frac{f(x)}{g(x)}$ over the support of X. M must also satify the following inequality $f(x) \leq Mg(x) \forall x$. This implies that the support of Z must also contained the support of X, that is $g(x) > 0$ whenever $f(x) > 0$. The standard algorithm is defined as follows:

As useful rejection sampling is, it has many short comings. In particular, to simulate proposals for certain distributions, such as truncated normal distributions, regardless of the easy to sample from proposal chosen the computational running time of such rejection statements can be shown to have an expectation equal to infinity. Thus, the ability to learn and replace such rejection sampling blocks with learnt surrogate functions, that are expressive enough to reciprocate the behavior of the

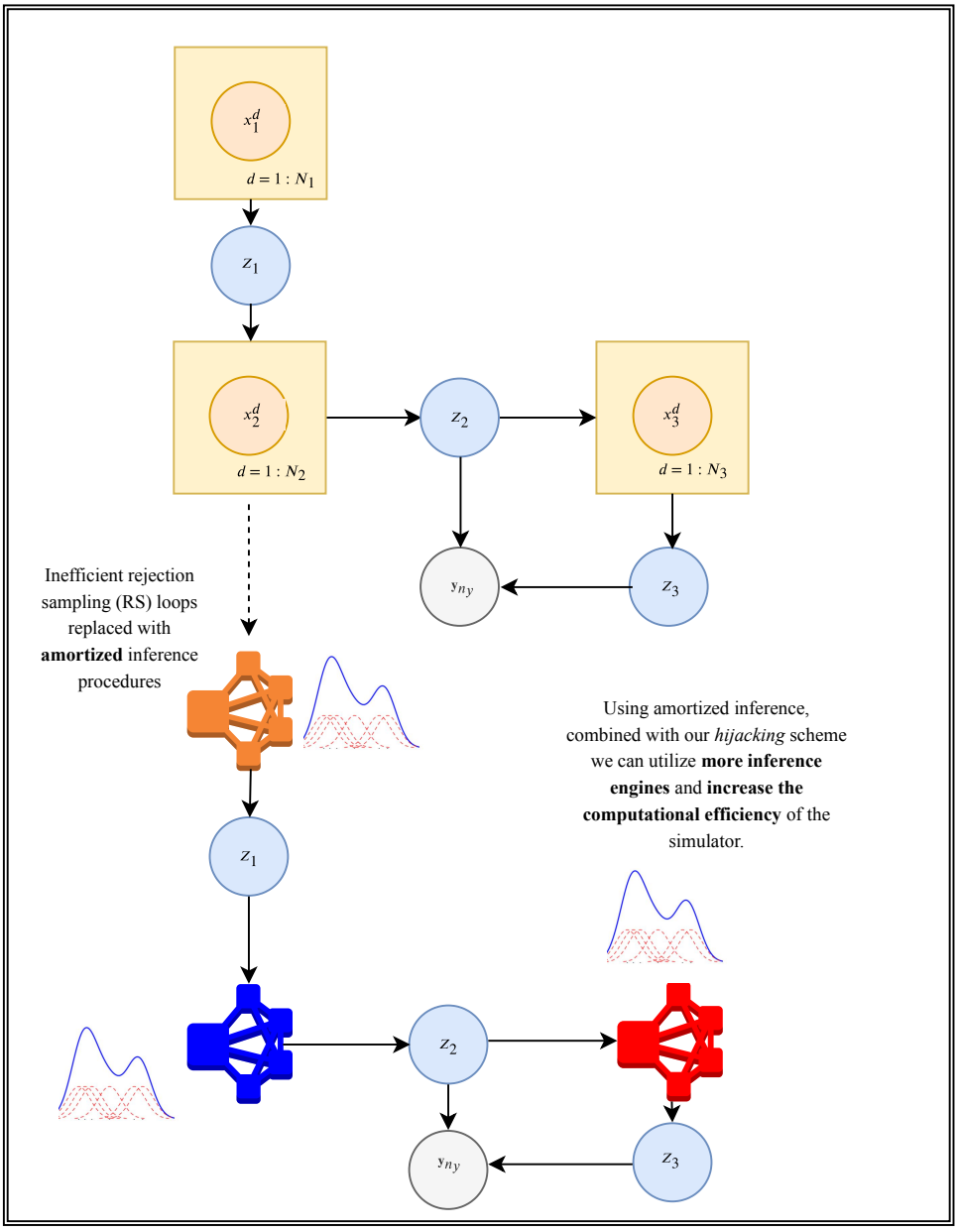

Figure 1: Using PyProb we can locate all regions of inefficient sampling and replace those points in the code with learnt functions that represent the distributions of the rejection sampling statements. Z_i represents the marginal and by learning surrogate functions for each of the rejection sampling loops. In doing so we can learn approximate raw sample distributions $f(x_i|\eta)$ for each rejection sampling loop. This means that we can evaluate the function at each part in the simulator, enabling us to construct a target density that can be evaluated when using MCMC inference schemes such as RMH.

Algorithm 1 Rejection Sampling

At iteration i s.t $i \geq 1$
$X_i \sim g(X_i)$
$U_i \sim \mathcal{U}(0, 1)$
if $U_i \leq M \frac{f(X_i)}{g(X_i)}$ **then**
    accept $X_i \sim f$
else
    $i \leftarrow i + 1$, repeat
end if

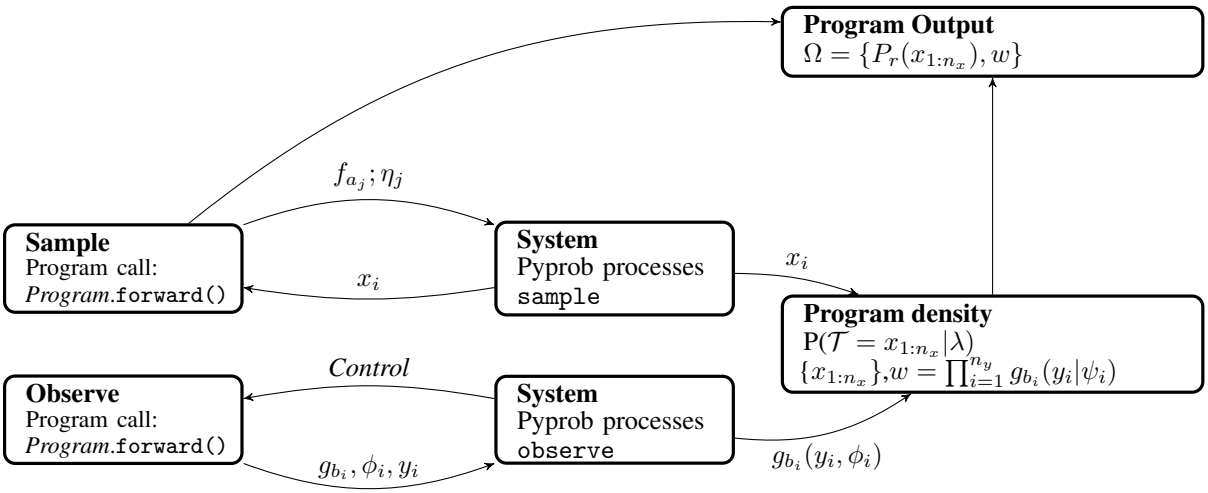

Figure 2: An overview of the hijacking system. x_i represent latent variables. f_{a_i} represent raw sample calls, g_{b_j} represent conditioning statements. $\mathcal{T}$ corresponds to the program trace. w represents the collective weights. r represents run number. n_x represents the number of latent variables and n_y represents the number of observations, which will always be fixed, but is model dependent. λ represents all the other variables generated by running the simulator `.forward()`.

rejection sampling block buys the user two things. A) It reduces the computational complexity of the simulator. B) As we now have a direct functional form of the density, we can correctly leverage more computationally and sample efficient inference schemes.

4 Defining the Density and Sub-Densities within Simulators

To construct the mathematical framework in this setting we employ the notation of **?**, where *raw samples* are represented by $f_{a_i}(x_i|\eta)$ and can

In order to correctly characterize this problem we first provide a useful diagram demonstrating the density of the system.

We do this by replacing the rejection sampling blocks with learnt functions from an amortized inference procedure Figure **??**. This then enables us to replace rejection sampling loops with amortized functions that are quick to evaluate, provide a functional form to the density and significantly reduce memory over-heads, as compared to **?**, due to not having to track the thousands of raw sample calls f_{a_i} generated in standard simulators.

Consider the following program, that is representative of a simple stochastic simulator and two types of rejection sampler:

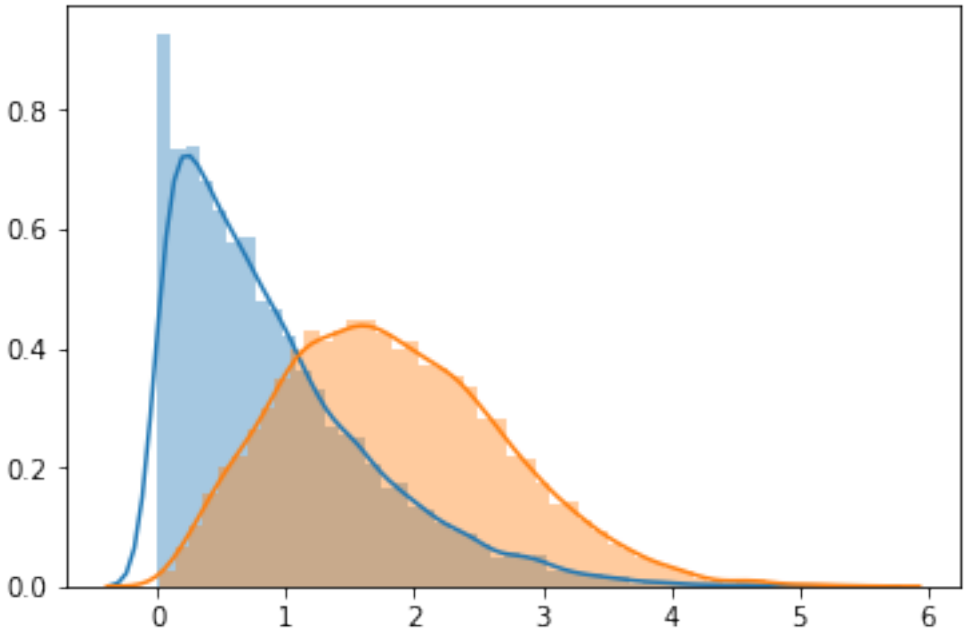

Figure 3: Simulated densities for rejection sampling block 1, *blue* and rejection sampling block 2, *orange*.

Stochastic Simulator	Rejection Sampler 1	Rejection Sampler 2

def f():
$z_1 \sim \mathcal{N}(0,1)$
$z_2 \leftarrow R_1(z_1, 1)$
$z_3 \sim \mathcal{U}(0,2)$
$z_4 \leftarrow R_2(z_2, z_3, 1)$
 	
return $g(z_{1:4})$
	def $R_1(z_1)$:
	$x \sim \mathcal{N}(z_1, 1)$
	if $x > 0$:
	return x
	else:
	return $R_1(z_1)$
		def $R_2(z_2, z_3)$:
		$y \leftarrow 0$
		while $y < z_2$:
		$y \sim \mathcal{N}(z_3, 1)$
		end
		return y

$P(\mathcal{T} = x_{1:n_x}\|\lambda)$	$q_1(z_2\|z_1)$	$q_2(z_4\|z_2, z_3)$

Here the rejection sampling functions take an additional argument 1, or 0, to specify whether or not we want to collect samples to train, 1, else replace with the learnt function, 0. P represents the program density and q_1 and q_2 represents the unknown densities that we wish to learn through an amortized approach.

However, in order to do this, we must correctly specify our variational objective.

5 An overview of Amortized Inference schemes

- Whilst the variational posterior we choose as our proposal has a functional form and can be sampled, the functional form of the rejection sampling loops is not explicitly known, which leads to problems when calculating the KL-divergence.
- There are two approaches that we can take to calculating some divergence metric.
- 1) Using something like a non-parametric divergence estimator that makes use of Renyi and L_2 divergences (see **?** for details)
- 2) Use some form of kernel density to estimation to approximate the functional form of the rejection sampling block, which can be used in place. From which we learn a variational approximation characterized by a trained network that we can sample from quickly.

- An issue with approach 2) is that if I can approximate the density in functional form aren't there more efficient methods that we can use? (**Ask tom**)
- Or, if assuming that the rejection sampling block can be arbitrary dimension, are we first learning a flow to represent that density and then learn a variational approximation of something that we can easily sample from.
- But, this is very much the idea behind normalising flows, take a simple distribution that you can sample and then map it to the "complicated" density.
- Stein Variational inference may be quite nice to use here **?**, but it does make explicit assumptions in requiring the densities to be differentiable.
- For an overview of all variational methods, including the use of different divergence metrics see here **?**

New questions to ask Tom

- Isn't what we are doing just conditional density estimation? It seems like there is a lot of prior work here. I.e see this paper here for a good overview **?**
- It does seem like this field has a large extensive past.

5.1 Using a simple Neural Network regressor

As a first approach, we use a regressor to learn a function to map between the input / output pairs that are outputted from the rejection sampling blocks with in the simulator.

Architecture For each rejection sampling block we learn a proposal $q(z_1, \ldots, z_i|\eta)$

5.2 Using variants of VAEs

5.3 Density estimation using Normalising flows

Notes from models that define an explicit, tractable density are highly effective, because they permit the use of an optimization algorithm directly on the log-likelihood of the training data. However, the family of models that provide a tractable density is limited, with different families having different disadvantage. The rejection sampling loops, if proposed correctly, provide a way of generating samples from some density, but do not provide a way of constructing the target density (**Bradley: Ask Tom, is this true?**). Within event and population-based simulators, the rejection sampling loops have been designed with care, to ensure that they are representative to the events that they are simulating. So we can, with confidence, assume that they are representative of some density. However, to utilise powerful inference engines, we must be able to construct the full density of the program.

One recent technique for density estimation, in the amortized inference setting, is normalising flows combined with variation inference **?**. Flow-based methods work by leveraging a simple property of differential geometry applied to random variables. That being the change of variables. Let $z \sim r(z)$ where $r(z)$ represents the posterior defined by the rejection sampling loop, and define π to be a smooth and invertible function. When π acts z we transform the random variable from one domain, to another. Thus, if $z\prime = \pi(z)$ then we can construct the density of the probability density of the new random variable $z\prime$ under the flow initiated by π:

$$r(z\prime) = r(\pi^{-1}(z\prime)) \left| \det \frac{\partial \pi^{-1}}{\partial z'} \right| = r(\pi^{-1}(z\prime)) \left| \det \frac{\partial \pi}{\partial z\prime} \right|^{-1} \tag{1}$$

- Do we need to know the base distribution that the RS block is sampling from? i.e there maybe be multiple $r(z)$ in each RS block.
- Is it right to say here that $z\prime$ is the output of the RS block, we do not know π here, or is $r(z)$ the raw sample f_{a_i} and π the distribution specified by the rejection sampling loop?
- We need to learn $\pi(.)$

6 Constructing the Objective

The framework we present in this section is generally applicable to stochastic simulator environment, where there exists rejection sampling loops.

We start by noting that the prior, defined in the context of the outer samples is defined as:

$$p(x) = \begin{cases} \prod_{j=1}^{n_x} f_{a_j}(x_j; \psi_j) if \mathcal{B}(x) = 1 \\ 0 \end{cases} \tag{2}$$

We start with the information projection we construct the variational objective as follows:

$$\begin{aligned} J(\eta) &= KL(p(x)||q(x;\eta)) \\ &= \int p(x) \ln\left(\frac{p(x)}{q(x;\eta)}\right) dx \\ &\text{Let } a_j \in \{1,\dots,L\}, S_R \subset \{1,\dots,L\} \\ &= \mathbb{E}_{p(x)}[\sum_{j=1, a_j \notin S_R}^{n_x} \ln(p(x))] - \mathbb{E}_{p(x)}[\sum_{j=1, a_j \in S_R}^{n_x} \ln(q_{a_j}(x_j;\eta))] \end{aligned}$$

no dependence on η

Thus, to define the objective for each subproblem $r \in S_R$ as:

$$J_r(\eta_r) = -\mathbb{E}_{p(x)}[\sum_{j=1}^{n_x} \mathbb{I}(a_j = r) \ln(q_r(x_j; \eta_r, \phi_j))] \tag{3}$$

where during training we extract samples from each forward run and train each rejection sampler separately.

7 Additional TODOs

- Before implementing such tools into the simulator, spend time looking for simpler stochastic simulators that can be hijacked but have target densities that can be constructed analytically.
- If we are not utilizing the additional information stored in the trace, which is only used during the inference compilation training procedure, then we do not need it all, as for importance sampling we only require the weights after each `forward()` update.
- If we cannot find any pre-existing simulators that are amenable to this approach then, we need to create our own, with a rejection sampling loop, that can be exploited by the new machinery. This would enable us to debug, provide a simple example for the paper and understand the feasibility of the entire approach.

A The unconditional acceptance probability

Given a collection of i.i.d random variables $\{X_i\}_{i=1:n}$ with cumulative density function (CDF) F_X. Then by the Glivenko-Cantelli theorem **?** we can define the empirical distribution function for $x \in \mathbb{R}$ as:

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}_{[X_i,\infty)}(x) \tag{4}$$

And for any fixed $x \in \mathbb{R}$, the Strong law of large numbers ensures that $F_n(x) \to F_X(x)$ as $n \to \infty$ as

$$\mathbb{E}[\mathbb{I}_{[X_i,\infty)}(x)] = P[\mathbb{I}_{X_i,\infty)}(x) = 1] = P[X_i \le x] = F_(X)(x) \tag{5}$$

where $\mathbb{I}_A(x)$ is the indicator function for a set A.

$$\text{Prob}(U \le \frac{p_X(z)}{M g_Z(z)} = \mathbb{E}_(\tag{6}$$

B Law of total expectations

If X is a random variable whose expected value is $\mathbb{E}[X]$ and Z is a random variable on the same probability space then:

$$\mathbb{E}[X] = \mathbb{E}_{p(z)}[\mathbb{E}_{p(x|z)}[X|Z]] \tag{7}$$

That is, the expected value of X given Z is the same as the expectation of X.

C Limitations with different inference engines

C.1 Prior based sampling

In prior based sampling we take our existing program and look directly at the product of all the raw sample calls, denoted $f_{a_i}(x_i|\eta)$ and construct a proposal that is is dependent on those. In this sampling scheme our proposal is defined as:

$$q(x_{1:n_x}) = \begin{cases} \prod_{i=1}^{n_x} f_{a_i}(x_i|\eta) \texttt{ if } \mathcal{B}(x_{1:n_x}) = 1 \\ 0 \end{cases} \tag{8}$$

where $\mathcal{B}(\cdot)$ is satisfied by the simulator and $f_{a_i}(\cdot)$ represents a raw random sample, η is a subset of the variables in-scope at the point of sampling and n_x is the number of latent variables. This is true in all inference engines used in this hijacking framework. However, this makes use of no conditioning and is not sufficient for performing inference in complex, population-based simulations.

C.2 Non-prior Based Sampling

In non-prior based sampling we utilise conditioning too, which enables us to explore spaces more efficiently. In the current hijacking set-up, we can only do conditioning at the end of each `forward()` run of the simulator. This does inhibit certain inference engines and makes aspects of performing inference within this setting more complicated. Nonetheless, there are several inference schemes that we can exploit and utilise. We shall now give an Introduction to each of these schemes.

C.3 Importance Sampling

In the generalised probabilistic programming setting an importance sampling scheme over the program raw samples and observations can be defined as follows:

$$p(\mathcal{T} = x_{1:n_x}|\lambda) = \begin{cases} \prod_{i=1}^{n_x} f_{a_i}(x_i|\eta) \prod_{j=1}^{n_y} g_{b_j}(y_j|\phi) \texttt{ if} \mathcal{B}(x_{1:n_x}) = 1 \\ 0 \end{cases} \tag{9}$$

where the observe statements are defined by $g_{b_j}(.)$, ϕ plays the same role as η and n_y represents the number of observations. The expectation over the whole program can be defined as:

$$\mathbb{E}_{p(\Omega|\lambda)}[h(\Omega)] = \mathbb{E}_{p(x_{1:n_x}|\lambda)}[h(P_r(x_{1:n_x}))] \tag{10}$$

Note: It is a little weird how we define this, as the the program output Ω should also include the product of the weights

where r represents a run of the program, $P_r(\cdot)$ is the entirety of the program output, $h(\cdot)$ is just a deterministic function.

As the importance sampler updates the weights according to the following formula:

$$w(x_{1:n_x}) = \prod_{i=1}^{n_x} \frac{f_{a_i}(x_i|\eta_i)}{q_i(x_i|f_{a_i}, \eta_i)} \prod_{j=1}^{n_y} g_{b_j}(y_j|\phi_j) \tag{11}$$

When the number of observations is fixed, the combined weight is equal to $w(x_{1:n_x}) = \prod_{j=1}^{n_y} w_j$. To generate the weight up to each observe we can define $K(j)$ to be a function that returns the least i before hitting an observe j. This enables us to write the j-th weight as follows:

$$w_j(x1 : K(j)) = g_{b_j}(y_j|\psi_j) \prod_{K(j-1)+1}^{K(j)} v_i \tag{12}$$

where we define $v_j = \frac{f_{a_j}(x_j|\eta_j)}{q_j(x_j|f_{a_j},\eta_j)}$

Taking these definitions we can construct a target density over the latents that is proportional to the products of `sample` and `observe` statements **Note: Add update posterior here**. Thus, given the proposal $q(x_{1:K(t)})$ we define the importance weight as:

$$w(x_{1:t}) = \prod_{j=1}^{t} w_j(x_{1:K(j)}) = \frac{p_t(x_{1:K(t)})}{q(x_{1:K(t)})} = \begin{cases} \prod_{j=1}^{t} g_{b_j}(y_j|w_i) \prod^{K}(j)_{K(j-1)+1} v_i \texttt{ if } B_t(x_1 : K(t)) = 1 \\ 0 \end{cases} \tag{13}$$

One problem with importance sampling is in its ability to scale and condition upon multiple observations. | Choosing a *good* prior remains challenging.

C.4 Sequential Monte Carlo

Given multiple observations, we can ake the inference more computationally efficient by employing sequential monte carlo (SMC), although as we can only condition at the end of one full run of the simulation, computational speed increases are minimal. SMC allows for conditioning on multiple observations, but the number of observations must be fixed. **Note: If we could condition on observations within the simulator, we could get dramatic speed increases over standard importance sampling.We can actually do this in pyprob, but it requires knowing where to condition. That can be quite challenging for arbitrary code.**

Algorithm 2 SMC for Population Based Simulators - well any general program - not new

procedure SMC(T, M,**x**, **w**,**v**, **g**)
  for $t = 1 : T$ **do**
    for $m = 1 : M$ **do**
      $w^m = 1$
      for $k = K(t-1)+1 : K(t)$ **do**
        $x_k^m \sim q_k(x_k^m|f_{a_k^m}, \eta_j^m)$
        $w_t^m \leftarrow w_m^t v_j^m(x_k^m)$
      end for
    end for
    $w_t^m \leftarrow w_t^m g_{b_t^m}(y_t^m|\psi_t^m)$
    $\{x_{1:t}^{1:m}\} \leftarrow$ `resample(` $\{x_{1:t}^{1:m}\}, \{w_{1:t}^{1:m}\}$)
  end for
end procedure

To implement SMC in the current set-up we round require a forking method and the

C.5 Random-walk Metropolis Hastings

Due to the current set-up performing Random-walk Metropolis Hastings (RMH) is not feasible as the rejection sampling blocks within the simulator mean that we cannot correctly perform RMH as we do not have a functional form for all parts of the simulator to construct a target density that we can evaluate. **Note: Ask tom what: one can sometimes achieve improvements by instead using the type of the distribution object associated with x_m (what does this first part mean?) to automatically construct a valid random walk proposal that allows for improved hill climbing behavior on the individual updates.** .

TODO: Add some of the plots generated from this model and an accompanying analysis to them.

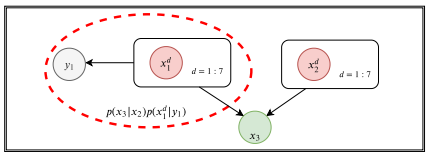

Figure 4: The DAG of the Parasite model.

D An Example: Modelling Parasite Densities in Simulation

The parasite model by Smith el al. **?** is a simple stochastic model for determining the number of infections within a given population demographic. The models hypothesized in **?** were found, after extensive testing and development, to not characterise well the number of infections among different demographics, especially those under 15 years old. By using probabilistic programming we can immediately understand the expressiveness of the proposed model and can exact important insights for extending the proposed models, which increases ones ability to develop improved models to estimate parasite densities, given our observations of the seasonal entomological inoculation rate (EIR). In addition to this we can greatly reduce the computational costs, whilst improving model comprehension and expressivity.

Common features of the models The models used are all discrete time micro-simulations of malaria in humans, based on the published model used in[1], which we refer to as the base model. The model for infection of the vector as a function of parasite densities[2] (and hence for the effect of vaccination on onward transmission), the case-management model[3], the model of vaccination by pre-erythrocytic vaccines[4], the models for clinical outcomes and mortality[5,6], and the basic design of the scenarios used for predicting impacts[3,7] all correspond to the previous descriptions, which are accompanied by detailed rationales for the formulations used.

For all models, the number of infections introduced in individual i in five-day time step t, is distributed as:

$$h(i,t) \sim Pos(\lambda(i,t)) \tag{14}$$

where the expected number $\lambda(i,t) = S(i,t)E_a(i,t)$ and $S(i,t)$ is the suseptability (proportion of inoculations that result in infections) and and $E_a(i,t)$ is the expected number of entomological inoculations, adjusted for age and individual factors.

Base model Although a statistical model is not formally defined within **?** we define the unknown *disease dynamics* parameters with a set of latent variables$\{x_i\}^N$ and construct the model as follows:

In the base model $E_a(i,t) = E_{max}(t)\frac{A(a(i,t))}{A_{max}}$ where $E_{max}(t)$ refers to the usual measure of the EIR computed from human bait collections on adults $A(\cdot)$ and , the availability to mosquitoes, is assumed to be proportional to average body surface area and to depend only on age $a(i,t)$. The function used to compute $A(a(i,t))$ increases with age up to age 20 years where it reaches a value of A_{max}.

To capture observed relationships between infection and measured entomological exposure, the susceptibility is then:

$$S(i,t) = (S_\infty + \frac{1-S_\infty}{1+\frac{E_a(i,t)}{E^*}})(S_{imm} + \frac{1-S_{imm}}{1+(\frac{X_p(i,t)}{X_p^*})^{\gamma_p}}) \tag{15}$$

where $S_{imm}, X_p^* E^*, \gamma_p, S_\infty$ are latent variables and $X_p(i,t) = \int_{-a(i,t)}^{t} E_a(i,\tau)d\tau$ Fitting this model to data of **?** using their method learns the values $S_\infty = 0.049$ and $E^* = 0.032$ inoculations/person-night. The model for each individual infection j in host i comprises a time series of parasite densities. In a naïve host the expectation of the logarithm of the density of the infection at time point τ in the course of the infection, $\ln(y_0(i,j,\tau))$, is taken from a statistical description of parasite densities in malaria therapy patients **?**. In the presence of previous exposure and co-infection, the expected log density for each concurrent infection is:

$$E(\ln(y(i,j,\tau))) = D_y(i,t)D_m(i,t)D_h(i,t)\ln(y_0(i,j,\tau)) + \ln(\frac{D_x}{M(i,t)} + 1 - D_x)) \tag{16}$$

where $M(i,t)$ is the total multiplicity of infection and

$$D_y(i,t) = \frac{1}{1 + \frac{X_y(i,j,t)}{X_y^*}} \tag{17}$$

where $X_y(i,j,t) = \sum_{t-a}^{t} Y(i,t) - \sum_{t_{0,j}}^{t} y(i,j,\tau)$ (note that a continuous time approximation to this is given in the original publications **??** and hence measures the cumulative parasite load, where

$$D_y(i,t) = \frac{1}{1 + \frac{X_h(i,t)}{X_h^*}} \tag{18}$$

where $X_h(i,t) = \sum_{t-a}^{t} h(i,\tau) - 1$, the number of inoculations since birth, excluding the one under consideration, which measures the inocula experienced by the host up to the time point under consideration.

$$D_m(i,t) = 1 - \alpha_m \exp(-\frac{0.693a(i,t)}{a_m^*}) \tag{19}$$

which measures the effect of maternal immunity. X_y^*, X_h^*, D_x, a_m^* and α_m are essentially latent variables to be learnt given the field data.

Mass action model of the force of infection The base model assumes that susceptibility is a sigmoidal function of the exposure, which conflicts with the usual mass action assumption for infectious disease transmission. However, if entomological exposure is over-dispersed, infection-exposure relationships similar to those observed in the field can be reconciled with mass action principles, thus providing explanation for the observed non-linear relationship between infection and entomological exposure and hence a more justifiable representation of the infection process than that used in the base model **?**. We simulate this by including random variation in the availability of the human host to mosquitoes. This variation comprises an individual-specific component which is captured by defining for each simulated human a baseline availability, $A_b(i)$, which is sampled at birth using:

$$\log(A_b(i)) \sim \mathcal{N}(-\frac{\sigma_b^2}{2}, \sigma_b^2) \tag{20}$$

so that $A_b(i)$ is log-normally distributed with arithmetic mean 1. The age adhusted EIR at time t, adjusted to the individual is then:

$$E_b(i,t) = E_{max}(t) A_b(t) \frac{A(a(i,t))}{A_{max}} \tag{21}$$

Additional log normal variation is introduced at each time step in the simulation to make the expected number of entomological inoculations a function both of the individual and of log-normal noise measured by the variance parameter, σ_n^2 :

$$\ln(E_a(i,t)) \sim Normal(\ln(E_b(i,t) - \frac{\sigma_n^2, \sigma_n^2}{)}) \tag{22}$$

The susceptibility in these models is independent of EIR, i.e.:

$$S(i,t(= S_\infty(S_{imm} + \frac{1 - S_{imm}}{1 + (\frac{X_p(i,t)}{X_p^*})^{\gamma_p}}) \tag{23}$$

Three different parameterisations, XML files, of this model were considered in [8] (E0063,R0065 and R0068). In each case, the overall variability in $E_a(i,t)$ was constrained to the same total, estimated by fitting to the data of **?**. A probabilistic programming approach to determining these paramteters would have provided a way to leverage state of the art inference algorithms that enable conditioning on arbitrary models. The parameters learnt from the original approach taken by the Malaria community generated values of $\sigma_b^2 = 0.5, 0.3$ and 0.05 for each model respectively, corresponding to $\sigma_n^2 = 0.187, 0.567$ and 0.704. **TODO: To finish - Add prob prog example + plots**

Models of decay of natural immunity To complete