AIRS: Amortized Inference for Rejection Samplers for Fast Simulator Inference

Paper ID: 1010

Abstract

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

1 Introduction

Simulators arise in a myriad of scientific and industrial settings; epidemiology, physics, engineering, climate modelling and so forth [add citations]. One interesting class of simulators are population-based simulators, using in epidemiology, reinforcement learning and financial modelling [add citations]. Performing inference in such population-based simulators is challenging, because we have nested-programs, which come in the form of rejection sampling blocks. In addition to this, each population member can traverse a number of different paths and can interact with other members, in certain instances, which can lead to a combinatorial number of paths to explore. Equally, in many simulated settings we cannot correctly construct the functional form of the density of the desired program, since we do not have explicit access to the density of the *program*. Where program density refers to the joint density constructed by the program, which in this case is represented by the simulator. This inability to evaluate the joint density has spawned research into how to perform inference in this likelihood free setting [12] **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 [2] 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 [2], 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 [2].

Preprint. Under review.

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 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 Nested-Programs within Simulators

- Talk about how within simulators we have nested-structures which are *nested-programs*
- Define the notations used to define the processes within the simulator, in terms of the PPL framework
- Talk about the problems that such nested-programs cause to inference procedures
- Define the density of the simulator in terms of these sub-programs.

To construct the mathematical framework in this setting we employ the notation of [16], where *raw samples* are represented by $f_{a_i}(x_i|\eta)$ and can be thought of as prior **To finish - Linked back to figure 1** In order to correctly characterize this problem we first provide a useful diagram demonstrating the density of the system 1. **Lots to do**

3 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 [2]. 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.

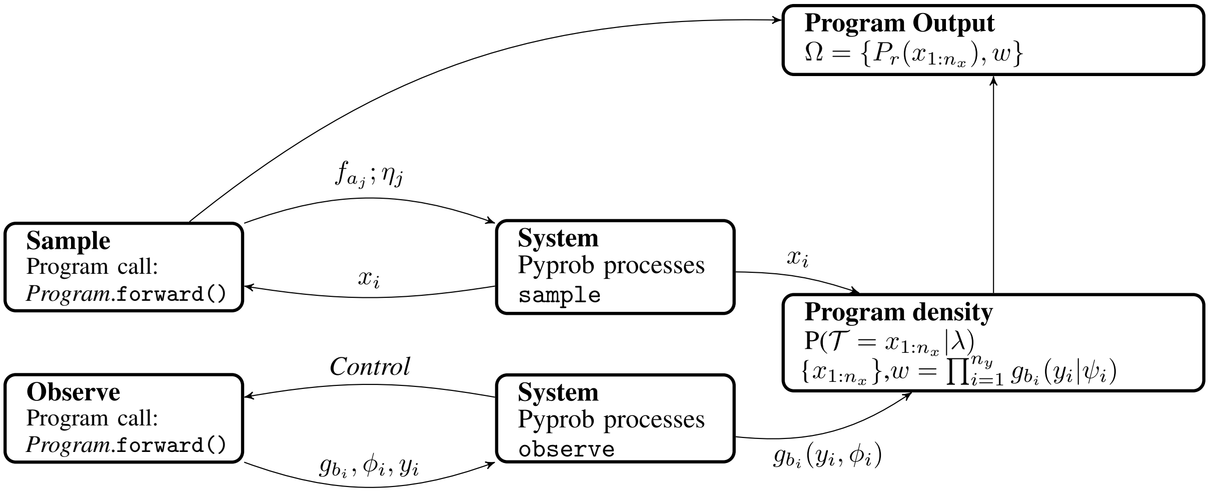


Figure 1: An overview of the hijacking system. x_i represent latent variables. f_{a_i} represent raw sample calls, g_{b_j} represent conditioning statements. $\mathcal{T}$ corresponds to the program trace. w represents the collective weights. r represents run number. n_x represents the number of latent 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()`.

3.1 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 [15, 14, 5]

There are two different forms of rejection samplers found with in simulators and program code. In both instances we we have X, a random variable whose density is $f(x)$ that we cannot easily sample from, but can evaluate. We then define a proposal using a random variable Z, whose density is $g(z)$ that we can efficiently sample from. Where the two methods diverge is in how the acceptance probability is defined and how the samples are generated. In the standard statistical setting, see Algorithm 1, we 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. Here M is a constant that has a finite bound, $M \in (1, \infty)$ and the likelihood ratio $\frac{f(x)}{g(x)}$ is defined over the support of X. M must also satisfy 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 $f(x) > 0$. We construct this acceptance probability by sampling from a uniform distribution defined on the interval $(0, 1)$, which the proof ultimately relies on to ensure that a sample generated from the rejection sampler directly corresponds to a sample from the true, normalized, target density $f(x)$. But, provided we can calculate $P_{\text{accept}}(\mathcal{D} \leq \mathcal{C})$ for the given density, or mass, function $\mathcal{D}$ and condition $\mathcal{C}$, we can also satisfy the standard proof with a little more work. However, as we will demonstrate, rejection samplers need not take this form and arise in many nested structures within probabilistic programs [17] and simulators [19, 4, 10]. More generally speaking we can write a rejection sampler in the form of Algorithm 2

Algorithm 1 ABC: 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
```

Algorithm 2 Simulator: Rejection Sampling

```
At iteration $i$ s.t $i \geq 1$
$X_i \sim p(x)$
$\mathbb{I}(g(x) > 0)$
$Y \sim p(X = x|\mathbb{I}(g(x) > 0))$
accept $X_i$ with $p(\mathbb{I}(g(x) > 0))$
```

Thus, the probability of generating a sample $X = x$ from the true distribution can be defined, using Bayes rule, as:

$$p(X = x|\mathbb{I}) = \frac{p(\mathbb{I}|X = x)p(X = x)}{p(\mathbb{I})} \tag{1}$$

where $p(\mathbb{I}|X = x) = \{0, 1\}$.

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. **Bradley: Is this true?**

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 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.

We do this by replacing the rejection sampling blocks with learnt functions from an amortized inference procedure Figure 2. 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 [2], 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:

Stochastic Simulator	R1	R2	R3

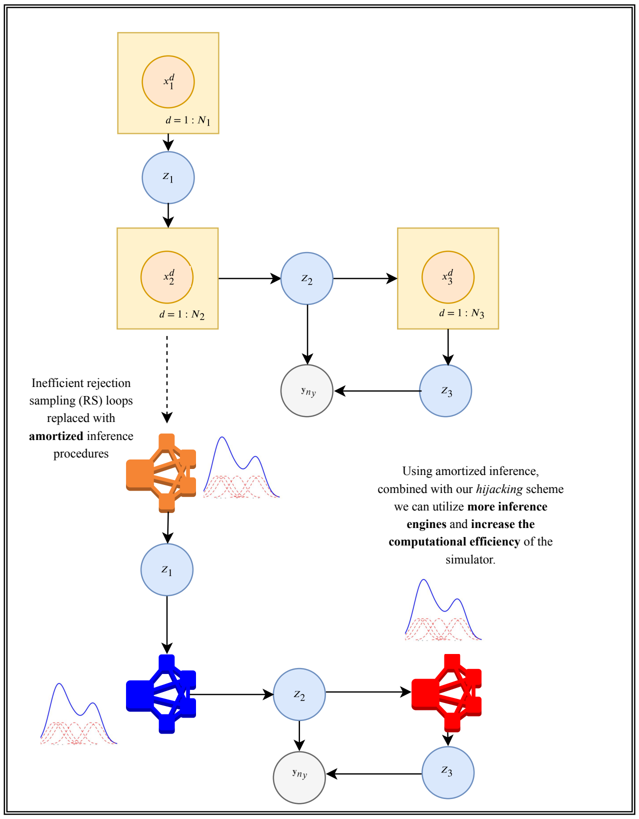
def f():
$z_1 \sim \mathcal{N}(0, 1)$
$z_2 \leftarrow R_1(z_1, 1)$
$z_3 \sim \mathcal{U}(0, 2)$
$z_4 \leftarrow R_2(z_2, z_3, 1)$
$z_5 \sim \mathcal{U}(50, 30^2)$
$z_6 \sim R_3(z_5, 1)$
return $g(z_{1:6})$
	def $R_1(z_1)$:
	$x \sim \mathcal{N}(z_1, 1)$
	if $x > 0$:
	return x
	else:
	return $R_1(z_1)$
		def $R_2(z_2, z_3)$:
		$y \leftarrow 0$
		while $y < z_2$:
		$y \sim \mathcal{N}(z_3, 1)$
		end
		return y
			def $R_3(z_5)$:
			$f(x) = \mathcal{N}(30, 10^2)+$
			$\mathcal{N}(80, 20^2)$
			$g(x) = \mathcal{N}(50, 30^2)$
			$u \sim \mathcal{U}(0, M * g(z_5))$
			if $u \leq f(z_5)$:
			return z_5
			else:
			return $R_3(z_5)$

$P(\mathcal{T} = x_{1:n_x}|\lambda)$	$q_1(z_2|z_1)$	$q_2(z_4|z_2, z_3)$	$q(z_5)$

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.

If we have direct access to a rejection sampling loop then it is indeed possible to construct the density, up to a constant of proportionality. That is:

$$\begin{aligned} z &\sim g(\theta) \\ &dosomething \\ \alpha &= p_{accept}(z, \theta) \\ \texttt{if} Bern(\alpha) &= 1 : \\ &returnz \end{aligned}$$



Inefficient rejection sampling (RS) loops replaced with **amortized** inference procedures

Using amortized inference, combined with our *hijacking* scheme we can utilize **more inference engines** and **increase the computational efficiency** of the simulator.

Figure 2: 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.

thus the density of $y, \theta \propto q(\theta)\alpha$. However, in order to do construct this we need the input θ, y, α and the distributions utilised for the rejection sampling loop. For any rejection sampling block, written in a generic code base, not in the form of R3, this is an incredibly hard task to do, because rejection sampling blocks within simulators are not necessarily written in a standard template. **Add examples in a supplement. Need to demonstrate to YW that this is challenging to do, even if you know where the rejection sampling loops are. Hypothesis, RS loops are not written in a templated form**. y leveraging the PPX protocols previously developed in [2, 11] we can however, extract all variables, other than α in an automated way. However, without being able to easily extract α this means that we are unable to correctly build the form of the density and thus are limited in our ability to leverage powerful inference methods such as MCMC and Variational inference. Thus, we propose another approach, automating the amortization of rejection sampling loops of type R1, R2 and R3. This enables us to learn functions that we can easily sample from and via the amortization process construct the density of the entire simulator.

The simulated densities for the rejection sampling block can be seen in Figure 3.1.

In order to replace such rejection sampling blocks with learnt functions that we can efficiently sample from, we must correctly specify our variational objective.

3.2 Constructing the Objective

The framework we present in this section is generally applicable to any stochastic simulator environment, where there exists any form of rejection sampling blocks. To perform the inferences that we

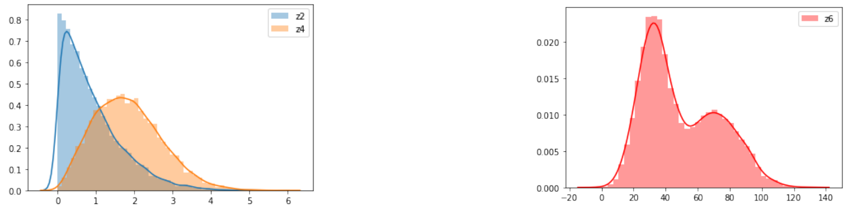


Figure 3: Simulated densities for rejection sampling block 1, *blue* and rejection sampling block 2, *orange* and rejection sampling block 3, *red*.

require, we construct a variational approximation of the rejection sampling blocks via an amortization procedure and note that the variational approximation in the context of our setting is specified by three things:

1. Makes use of existing parts of the outer program including all deterministic operations.

2. Form of the $\gamma_{a_j}(x_j;\eta_{a_j},\psi_j)\forall a_j \in S_r$.

3. There exists variational parameters $\eta_{a_j}\forall a_j \in S_r$.

where $a_j \in \{1,\ldots,L\}$ corresponds to the set of all addresses, S_r denotes the set of addresses a_j that correspond directly to the locations of the rejection sampling blocks, that is $a_j \cup S_r = \{1,\ldots,L\}$, γ_{a_j} denotes the variational approximation at address a_j, for example, $\gamma_{a_j} = \mathcal{N}(x_j;\mu_{a_j}(\psi_j;\eta_{a_j}),\Sigma_{a_j}(\psi_j;\eta_{a_j}))$, with η_{a_j} representing the variational parameters and ψ_j representing all variables that input into the given program.

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

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

next, we state for a given program $\mathcal{P}$ we denote the proposal for the program as:

$$q(\mathbf{x};\kappa) = \begin{cases} \prod_{j=1,j\notin S_r}^{n_x} f_{a_j}(x_j;\psi_j)(\prod_{j=1,j\in S_r}^{n_x} \gamma_{a_j}(x_j;\eta_{a_j},\psi_j)) \texttt{ if } \mathcal{B}(x_{1:n_x} = 1) \\ 0 \end{cases} \tag{3}$$

where $\kappa = \{\eta_{a_j}; a_j \in S_r\}$ are the variational parameters. As we now have the density of the proposal and the density of the program we can construct a variational objective as follows:

using the information projection we construct the variational objective as follows:

$$\begin{aligned}
J(\eta) &= KL(p(x)||q(x;\kappa)) \\
&= \int p(x) \ln \left(\frac{p(x)}{q(x;\eta)} \right) dx \\
\text{Let } a_j &\in \{1, \ldots, L\}, S_R \subset \{1, \ldots, L\} \\
\kappa^* &= \underset{\kappa}{\operatorname{argmin}} \, \mathbb{E}_{X \sim p(x)}[- \log(q(X = x; \kappa))] \\
&= \underset{\kappa}{\operatorname{argmax}} \, \mathbb{E}_{X \sim p(x)}[\sum_{j=1, j \in S_r}^{n_x} \log(\gamma_{a_j}(X_j = x_j; \eta_{a_j}, \psi_j))] \\
\forall r \in S_r, \eta_r^* &= \underset{r}{\operatorname{argmax}} \, \mathbb{E}_{x \sim p(x)}[\sum_{j=1}^{n_x} \mathbb{I}(a_j = r) \log(\gamma_r(X_j = x_j; \eta_r, \psi_j))] \\
\nabla_\kappa J(\kappa) &= \nabla_\kappa \mathbb{E}_{X \sim p(x)}[\sum_{j=1}^{n_x} \mathbb{I}(a_j = r) \log(\gamma_r(X_j = x_j; \eta_r, \psi_j))] \\
&= \mathbb{E}_{X \sim p(x)}[\sum_{j=1, j \in S_r}^{n_x} \nabla_\kappa \log(\gamma_r(X_j = x_j; \eta_r, \psi_j)] \\
&\approx \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1, j \in S_r}^{n_x} \nabla_\kappa \log(\gamma_r(x_j^n; \eta_r, \psi_j))
\end{aligned}$$

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

$$\nabla_{\eta_r} J(\kappa) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{n_x} \mathbb{I}(r = a_j) \nabla_{\eta_r} \log(\gamma_r(x_j^n; \eta_r, \psi_j)) \tag{4}$$

where $x_{1:n_x}^n \overset{iid}{\sim} p(x)$.

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

4 Experiments: Amortizing Stochastic Processes

TODO list

- Implement baselines
- Implement plotting scripts and evaluation scripts
- testing code and infrastructure
- litrature review on any methods to learn rejection sampling methods, and conditional density estimation. See this paper for kernel approximation methods [8]. In addition to the methods listed in the rejection sampling subsection.

4.1 Ground truth

Because in this simple case we have direct access to α, the rejection sampler acceptance probability, we can calculate the true density up to a constant of proportionality. **I will add this tomorrow**. I had no idea about this, until my meeting with YW today.

4.2 Baseline

As a first approach, we train 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.

For $R1$ we have one input $z^{r1} \in \mathbb{R}^{1\times1}$ and one output $y^{r1} \in \mathbb{R}^{1\times1}$. For $R2$ we have two inputs $z^{r2} \in \mathbb{R}^{2\times1}$ and one output $y^{r2} \in \mathbb{R}^{1\times1}$. For training purposes we generate n_b batches of n_t samples for training. This means that $z^{r1}_{batch}\mathbb{R}^{n_b\times1}$. For the loss function we use a simple mean squared error loss **We can try different loss functions and optimizers** and use the neural network, $\Gamma(z, y)$, to learn a $\hat{\mu}$ and $\hat{\sigma}^2$ to parametize a normal proposal. For $R1$, this means $\Gamma_{r1}(z, y) \rightarrow \hat{\mu_{r1}}, \hat{\sigma_{r1}}^2$ and so the proposal $q(z_1|z_2) = \mathcal{N}(\hat{\mu_{r1}}, \hat{\sigma_{r1}}^2)$. Once learned we can then efficiently sample from the parameterized normal distribution, which represents the density of the sub-program represented by $R1$. This is then replaced in the simulator, during inference time.

observations Using a standard $L2$-loss function $L_2 = (y_{pred} - y_{true})^2$ As the functions are relatively simple the loss functions it gets t to a loss of around 0.6 and stops decreasing. Although, this seems low, I don't think that it is actually good, because that would mean on average, for each data point there is a loss of ± 0.6 which per data point is quite bad. That may be for a number of reasons, using MSE loss, using a simple model architecture. It really struggles to model the tails. It may require a much larger batch input size. . Increasing the batch size helped to correct for the peaked density, however the behavior is still odd. The peaked behaviour still does not disappear, this, I believe is due to the loss function averaging over all points.

After my meeting with Tom, by using a loss function the problem of the network collapsing on a particular value was always going to happen as we should be optimizing our function via some objective, rather than looking at point-wise differences.

We had defined an objective above but it was not clear how this was supposed to be implemented in practice with the given notations.

We discussed at length how to correctly define the objective, I have now updated the objective with the one derivied in the revised discussion, along with additional notes on rejection sampling.

Add results from current experiments

4.3 2nd Baseline

Instead of using a single Gaussian, use a proposal that is a mixture of Gaussians. That is, for $R1$, $q(z_1|z_2) = \sum_{i=1}^{N} K_i \mathcal{N}(\hat{\mu^i_{r1}}, \hat{\sigma^i_{r1}}^2)$ under the constraint that $\sum_{i=1}^{N} K_i = 1$. Maybe N can be treated as a hyper-parameter. Or choose particular values.

4.4 More complicated: Conditional Density estimation

4.5 3rd Baseline

Use the variational objective constructed above.

Normalizing flows One recent technique for density estimation, in the amortized inference setting, is normalising flows combined with variation inference [18]. Flow-based methods work by leveraging a simple property of differential geometry applied to random variables. That being the change of variables. Using flow based methods we can construct our rejection sampling block density, by learning a flow to map from a simpler distribution to more complicated one. 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:

$$r(z\prime) = r(\pi^{-1}(z\prime)) \left| \det \frac{\partial \pi^{-1}}{\partial z'} \right| = r(\pi^{-1}(z\prime)) \left| \det \frac{\partial \pi}{\partial z\prime} \right|^{-1} \tag{5}$$

Flows available are Real-NVP [6] and Neural Spline flows [7]. There are several others, for other techniques ask Gunes.

- 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 $\pi(.)$

4.6 Using variants of VAEs

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.

5 Friday morning

The subprograms within the simulator should be treated as nested probabilistic programs.

There are three ways we can do these things, rather than just an amortization

5.1 Method 1: Learn the

$\gamma_{a_j}(\cdot;\eta_{a_j},\cdot)$

As per we are currently planning. Relies on generating exact samples, which we have access due to the rejection sampling loops.

In this setting we can approximate the output

5.2 Method 2: Proper amortized inference of sub-programs

Learn a $q_{a_j}(z_{1:j};\eta_{a_j},\phi_j)$ and uses this as a proposal for the inner inference. I.e. if we learn that q we could develop a more efficient rejection sampler by replacing the proposal.

I.e we are calibrating the proposal to make it more amenable to accepting samples. We can do this by using self normalized importance sampling on the inner program.

The likelihood is just a $\{0,1\}$ likelihood, ie Bernoulli.

5.3 Method 3: Learn marginal likelihood approximation

With $p(\cdot)$ representing the marginal likelihood.

$$p(x) = \begin{cases} \prod_{j=1,a_j\in S_R}^{n_x} f_{a_j}(x_j;\psi_j) \prod_{j=1,a_j\in S_R}^{n_x} \frac{P_{a_j}^{in}(x_j,\psi_j)}{P_{a_j}^{in}(\psi_j)} \\ 0 \end{cases} \tag{6}$$

The normalising constant inside (in) the rejection sampler is intractable.

The things inside of the rejection sampler are proportional to the following:

It is really important to ensure that these pairs of samples are unbiased.

If you minimise an L_2 norm the optimal

6 Questions for Tom

New questions to ask Tom

Old questions for Tom

- I don't understand how the objective function is comparing the generated samples from the density estimator to the ground truth, at least in the way it is currently defined? Answered - we updated the objective defined and added more comments
- All RS references I look at state that we must sample from a uniform distribution and then check if our proposed value is less than that uniformly sampled point, yet our RS do not have this. Answered - Rejection samplers we have are valid rejection samplers, this just might not define the density we expect them to. ..? nor what is the acceptance probability within our rejection sampling statements? The proof for rejection sampling seems to rely on the fact that you are drawing from a uniform distribution.
- 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 - reparametrize a with a function from a family of easy to sample distribution.
- 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 [1] for details) - No point, more difficult to implement
- 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? - There are other methods out there and it would maybe make sense to compare some of those methods. However, it may still be difficult to sample.
- 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 - Question not well defined.
- Stein Variational inference may be quite nice to use here [9], but it does make explicit assumptions in requiring the densities to be differentiable. - Yes this method is good, but only works in the continuous domain.
- For an overview of all variational methods, including the use of different divergence metrics see here [24] - Answered - Quite finicky to set up.

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.

References

[1] Michele Basseville. Divergence measures for statistical data processing—an annotated bibliography. *Signal Processing*, 93(4):621–633, 2013.

[2] Atilim Gunes Baydin, Lukas Heinrich, Wahid Bhimji, Bradley Gram-Hansen, Gilles Louppe, Lei Shao, Kyle Cranmer, Frank Wood, et al. Efficient probabilistic inference in the quest for physics beyond the standard model. *arXiv preprint arXiv:1807.07706*, 2018.

[3] John C Beier, Charles N Oster, Fred K Onyango, James D Bales, James A Sherwood, Peter V Perkins, David K Chumo, Davy V Koech, Richard E Whitmire, Clifford R Roberts, et al. Plasmodium falciparum incidence relative to entomologic inoculation rates at a site proposed for testing malaria vaccines in western kenya. *The American journal of tropical medicine and hygiene*, 50(5):529–536, 1994.

[4] 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.

[5] George Casella, Christian P. Robert, and Martin T. Wells. *Generalized Accept-Reject sampling schemes*, volume Volume 45 of *Lecture Notes–Monograph Series*, pages 342–347. Institute of Mathematical Statistics, Beachwood, Ohio, USA, 2004.

[6] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[7] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. *arXiv preprint arXiv:1906.04032*, 2019.

[8] Akram Erraqabi, Michal Valko, Alexandra Carpentier, and Odalric Maillard. Pliable rejection sampling. In *International Conference on Machine Learning*, pages 2121–2129, 2016.

[9] Yihao Feng, Dilin Wang, and Qiang Liu. Learning to draw samples with amortized stein variational gradient descent. *arXiv preprint arXiv:1707.06626*, 2017.

[10] 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.

[11] 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. *arXiv preprint arXiv:1905.12432*, 2019.

[12] David S Greenberg, Marcel Nonnenmacher, and Jakob H Macke. Automatic posterior transformation for likelihood-free inference. *arXiv preprint arXiv:1905.07488*, 2019.

[13] Nicolas Maire, Thomas Smith, Amanda Ross, Seth Owusu-Agyei, Klaus Dietz, and Louis Molineaux. A model for natural immunity to asexual blood stages of plasmodium falciparum malaria in endemic areas. *The American journal of tropical medicine and hygiene*, 75(2_suppl):19–31, 2006.

[14] Luca Martino, Jesse Read, and David Luengo. Improved adaptive rejection metropolis sampling algorithms. *arXiv preprint arXiv:1205.5494*, 2012.

[15] Renate Meyer, Bo Cai, and François Perron. Adaptive rejection metropolis sampling using lagrange interpolation polynomials of degree 2. *Computational Statistics and Data Analysis*, 52(7):3408 – 3423, 2008.

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

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

[18] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

[19] 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.

[20] Thomas Smith, Gerry F Killeen, Nicolas Maire, Amanda Ross, Louis Molineaux, Fabrizio Tediosi, Guy Hutton, Jürg Utzinger, Klaus Dietz, and Marcel Tanner. Mathematical modeling of the impact of malaria vaccines on the clinical epidemiology and natural history of plasmodium falciparum malaria: Overview. *The American journal of tropical medicine and hygiene*, 75(2_suppl):1–10, 2006.

[21] Thomas Smith, Nicolas Maire, Klaus Dietz, Gerry F Killeen, Penelope Vounatsou, Louis Molineaux, and Marcel Tanner. Relationship between the entomologic inoculation rate and the force of infection for plasmodium falciparum malaria. *The American journal of tropical medicine and hygiene*, 75(2_suppl):11–18, 2006.

[22] Thomas A Smith. Estimation of heterogeneity in malaria transmission by stochastic modelling of apparent deviations from mass action kinetics. *Malaria journal*, 7(1):12, 2008.

[23] Howard G Tucker. A generalization of the glivenko-cantelli theorem. *The Annals of Mathematical Statistics*, 30(3):828–830, 1959.

[24] C. Zhang, J. Bütepage, H. Kjellström, and S. Mandt. Advances in variational inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):2008–2026, Aug 2019.

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 [23] we can define the empirical distribution function for $x \in \mathbb{R}$ as:

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{[X_i,\infty)}(x) \tag{7}$$

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

$$\mathbb{E}[\mathbb{I}_{[X_i,\infty)}(x)] = P[\mathbb{I}_{X_i,\infty)}(x) = 1] = P[X_i \leq x] = F_(X)(x) \tag{8}$$

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

$$\text{Prob}(U \leq \frac{p_X(z)}{Mg_Z(z)} = \mathbb{E}_() \tag{9}$$

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:

$$\mathbb{E}[X] = \mathbb{E}_{p(z)}[\mathbb{E}_{p(x|z)}[X|Z]] \tag{10}$$

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:

$$q(x_{1:n_x}) = \begin{cases} \prod_{i=1}^{n_x} f_{a_i}(x_i|\eta) \; \texttt{if} \; \mathcal{B}(x_{1:n_x}) = 1 \\ 0 \end{cases} \tag{11}$$

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:

$$p(\mathcal{T} = x_{1:n_x}|\lambda) = \begin{cases} \prod_{i=1}^{n_x} f_{a_i}(x_i|\eta) \prod_{j=1}^{n_y} g_{b_j}(y_j|\phi) \texttt{ if}\mathcal{B}(x_{1:n_x}) = 1 \\ 0 \end{cases} \tag{12}$$

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:

$$\mathbb{E}_{p(\Omega|\lambda)}[h(\Omega)] = \mathbb{E}_{p(x_{1:n_x}|\lambda)}[h(P_r(x_{1:n_x}))] \tag{13}$$

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:

$$w(x_{1:n_x}) = \prod_{i=1}^{n_x} \frac{f_{a_i}(x_i|\eta_i)}{q_i(x_i|f_{a_i}, \eta_i)} \prod_{j=1}^{n_y} g_{b_j}(y_j|\phi_j) \tag{14}$$

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:

$$w_j(x1 : K(j)) = g_{b_j}(y_j|\psi_j) \prod_{K(j-1)+1}^{K(j)} v_i \tag{15}$$

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:

$$\begin{aligned} w(x_{1:t}) = \prod_{j=1}^{t} w_j(x_{1:K(j)}) = \\ \frac{p_t(x_{1:K(t)})}{q(x_{1:K(t)})} = \begin{cases} \prod_{j=1}^{t} g_{b_j}(y_j|w_i) \prod^K(j)_{K(j-1)+1} v_i \texttt{ if } B_t(x_1 : K(t)) = 1 \\ 0 \end{cases} \end{aligned} \tag{16}$$

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 3 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
        $w_t^m \leftarrow w_t^m g_{b_t^m}(y_t^m | \psi_t^m)$
        $\{x_{1:t}^{1:m}\} \leftarrow \texttt{resample}( \{x_{1:t}^{1:m}\}, \{w_{1:t}^{1:m}\} )$
    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.

D Stochastic Variational Inference

Using the standard notations we define the joint as follows:

$$p(x, z; \theta) = p_\theta(x, z) = p_\theta(x|z)p_\theta(z) \tag{17}$$

We assume that the distributions that create the *model* $p_\theta(x, z)$ can be:

1. We can sample from each p_i
2. We can compute the pointwise log pdf of p_i
3. p_i is differentiable w.r.t the parameters θ

D.1 Learning the model

Our objective will either want to maximize, or minimize, depending on what we want to discover.

$$\theta_{\text{min}\backslash\text{max}} = \text{argmax}\backslash\text{min}_\theta \log p_\theta(x) \tag{18}$$

where the log evidence $p_\theta(x)$ is given by:

$$\log p_\theta(x) = \log \int dz p_\theta(x, z) \tag{19}$$

This is a doubly hard problem as even if θ is fixed the integral over the set of latent variables defined by z is intractable. Even if we know how to calculate the log evidence for all θ, maximising or minimising, the log evidence as a function of θ is in general a difficult non-convex problem.

In addition to finding the parameters θ we also require the posterior over the latent variables z:

$$p_{\theta_{\text{max}\setminus\text{min}}}(z|x) = \frac{p_\theta(x,z)}{\int dz p_{\theta_{\text{max}\setminus\text{min}}}(x,z)} \tag{20}$$

To this in the variational inference setting we learn an approximation to $p_{\theta_{\text{max}\setminus\text{min}}}(z|x)$ denoted by $q_\phi(z)$ where ϕ are the variational parameters.

E The Objective

In many of these settings the ELBO loss characterises the problem that we aim to solve.

$$ELBO = \mathbb{E}_{q_\phi(z)}[\log p_\theta(x,z) - \log q_\phi(z)] \tag{21}$$

and we have for all choices of θ and ϕ that:

$$\log p_\theta(x) \geq ELBO \tag{22}$$

F An Example: Modelling Parasite Densities in Simulation

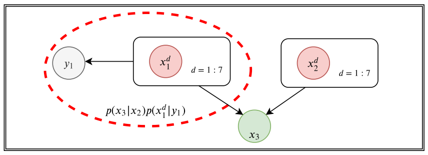


Figure 4: The DAG of the Parasite model.

The parasite model by Smith el al. [21] is a simple stochastic model for determining the number of infections within a given population demographic. The models hypothesized in [21] 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:

$$h(i,t) \sim \mathit{Pos}(\lambda(i,t)) \tag{23}$$

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 [21] 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:

$$S(i,t) = (S_\infty + \frac{1-S_\infty}{1+\frac{E_a(i,t)}{E^*}})(S_{imm} + \frac{1-S_{imm}}{1+(\frac{X_p(i,t)}{X_p^*})^{\gamma_p}}) \tag{24}$$

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 [3] 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 [13]. In the presence of previous exposure and co-infection, the expected log density for each concurrent infection is:

$$E(\ln(y(i,j,\tau))) = D_y(i,t)D_m(i,t)D_h(i,t)\ln(y_0(i,j,\tau)) + \ln(\frac{D_x}{M(i,t)} + 1 - D_x)) \tag{25}$$

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

$$D_y(i,t) = \frac{1}{1+\frac{X_y(i,j,t)}{X_y^*}} \tag{26}$$

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 [20, 13] and hence measures the cumulative parasite load, where

$$D_y(i,t) = \frac{1}{1+\frac{X_h(i,t)}{X_h^*}} \tag{27}$$

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.

$$D_m(i,t) = 1 - \alpha_m \exp(-\frac{0.693a(i,t)}{a_m^*}) \tag{28}$$

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 [22]. 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:

$$\log(A_b(i)) \sim \mathcal{N}(-\frac{\sigma_b^2}{2}, \sigma_b^2) \tag{29}$$

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:

$$E_b(i,t) = E_{max}(t)A_b(t)\frac{A(a(i,t))}{A_{max}} \tag{30}$$

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 :

$$\ln(E_a(i,t)) \sim Normal(\ln(E_b(i,t) - \frac{\sigma_n^2, \sigma_n^2}{)}) \tag{31}$$

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

$$S(i,t(= S_\infty(S_{imm} + \frac{1 - S_{imm}}{1 + (\frac{X_p(i,t)}{X_p^*})^{\gamma_p}}) \tag{32}$$

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 [3]. 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