Modeling Selection Processes

Michael Betancourt

March 2024

Table of contents

We often take for granted our ability to observe *every* event that arises from a latent data generating process. Many observations, however, are compromised by selection effects, where the output of an initial, latent data generating process must pass through an intermediate selection process that can prevent events from being observed.

Inferences derived from the resulting data capture only a biased perspective of that latent data generating process. To derive faithful inferences we need to model both the latent data generating process and the subsequent selection process.

In this chapter we'll review the basic modeling techniques for incorporating selection processes and the computational challenges that arise when trying to implement the resulting models in practice.

1 Selection Models

Consider an initial data generating process which generates events that take values in the space Y. Introducing a selection process annotates Y with a binary variable $z \in \{0, 1\}$ that takes the value $z = 0$ when an event is rejected and $z = 1$ when an event is accepted and persists to the final observation.

A joint model of the latent data generating process and the selection process is defined by a joint probability density function over the product space $\{0, 1\} \times Y$,

$$p(z, y).$$

The complete data generating process is then captured by the conditional decomposition

$$p(z, y) = p(z \mid y)\, p(y),$$

where $p(y)$ models the latent data generating process and $p(z \mid y)$ models the selection process. Here the conditional probability of selection

$$p(z = 1 \mid y) \equiv S(y),$$

is known as a **selection function**.

Because we do not observe every event we cannot use the joint model $p(z, y)$ directly. Instead we have to *condition* it on a successful observation.

1.1 Modeling Accepted Events

Observations that survive the selection process are modeled with the conditional probability density function (Figure 1)

$$\begin{aligned} p(y \mid z = 1) &= \frac{p(z = 1, y)}{p(z = 1)} \\ &= \frac{p(y)\, S(y)}{\int \mathrm{d}y'\, p(y')\, S(y')}. \end{aligned}$$

The more the selection function deviates from $S(y) = 1$ across neighborhoods of high latent probability the more the observational model $p(y \mid z = 1)$ will deviate from the latent model $p(y)$ (Figure 2).

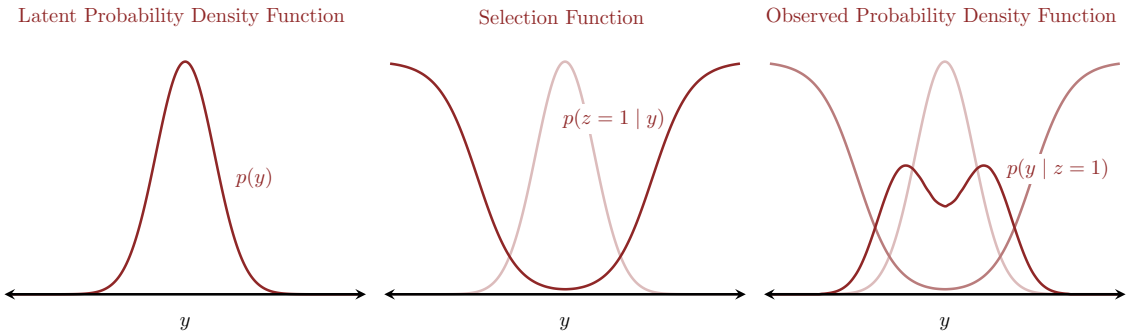


Figure 1: In a selection model events are generated from (a) a latent probability distribution before passing through a selection process defined by (b) a selection function. The observed events that are accepted by the selection process are modeled with a corresponding (c) observed probability distribution.

The denominator

$$Z \equiv p(z = 1) = \int \mathrm{d}y'\, p(y')\, S(y')$$

is referred to as the **normalization integral**, **normalization constant**, or simply **normalization** for short. Note that the normalization integral is also equal to the expectation value of the selection function with respect to the latent probability distribution,

$$Z = \int \mathrm{d}y'\, p(y')\, S(y') = \mathbb{E}_{\pi}[S].$$

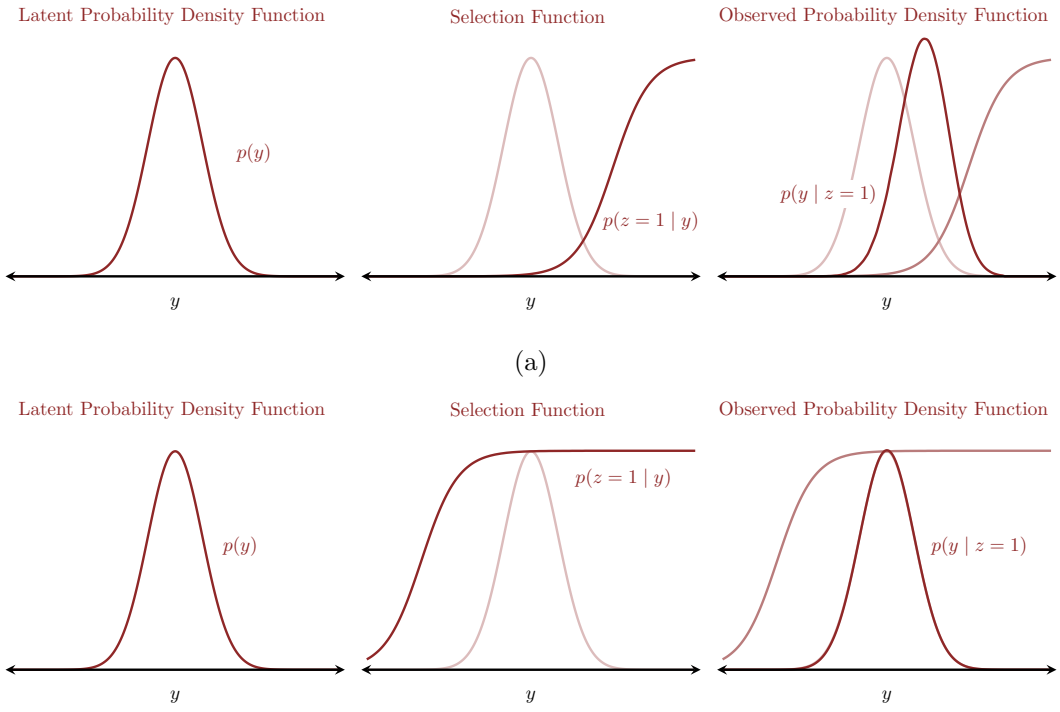


Figure 2: The overlap between the latent probability distribution and the selection function determines how strongly the selection process affects the observed data. (a) When a selection function strongly varies in regions of high latent probability the resulting observed probability distribution will be skewed away from the latent probability distribution. (b) On the other hand if the selection function is relatively uniform in regions of high latent probability then the selected events will be similar to the original latent events.

In most applications the latent probability density function and selection function are not known exactly and we have to infer their behaviors from observed data (Figure 3). When the latent probability density function is configured with the parameters ϕ and the selection function is configured with the parameters ψ the observational model becomes

$$p(y \mid z = 1, \phi, \psi) = \frac{p(y \mid \phi)\, S(y; \psi)}{\int \mathrm{d}y'\, p(y' \mid \phi)\, S(y'; \psi)}.$$

Critically the normalization is no longer a constant but varies with the possible parameter configurations,

$$Z(\phi, \psi) = \int \mathrm{d}y'\, p(y' \mid \phi)\, S(y'; \psi).$$

In order to implement this observational model in practice we need to be able to evaluate the normalization integral for arbitrarily values of ϕ and ψ. Unfortunately analytic evaluation of $Z(\phi, \psi)$ is rarely feasible for practical models. Because of this computational hurdle selection models are often referred to as **intractable** models.

We will discuss general numerical methods for evaluating $Z(\phi, \psi)$ in Section 2. The exercises in Section 3, however, will focus on exceptional cases where the normalization can be evaluated analytically so that we can directly quantify the error of these numerical methods.

1.2 Modeling The Number of Rejected Events

In some applications we may not be *completely* ignorant of the rejected events. The mere existence of rejected events can be surprisingly informative about the behavior of the latent probability distribution and selection function even if we don't know the exact values of those events.

We can model the presence of a single rejected event by evaluating the joint model at $z = 0$ and then marginalizing out the possible latent values (Figure 4),

$$\begin{aligned}
p(z = 0 \mid \phi, \psi) &= \int \mathrm{d}y'\, p(z = 0, y \mid \phi, \psi) \\
&= \int \mathrm{d}y'\, p(y \mid \phi)\, p(z = 0 \mid y, \psi) \\
&= \int \mathrm{d}y'\, p(y \mid \phi)\, (1 - p(z = 1 \mid y, \psi)) \\
&= \int \mathrm{d}y'\, p(y \mid \phi)\, (1 - S(y; \psi)) \\
&= \int \mathrm{d}y'\, p(y; \phi) - \int \mathrm{d}y'\, p(y \mid \phi)\, S(y; \psi) \\
&= 1 - \int \mathrm{d}y'\, p(y \mid \phi)\, S(y; \psi) \\
&= 1 - Z(\phi, \psi).
\end{aligned}$$

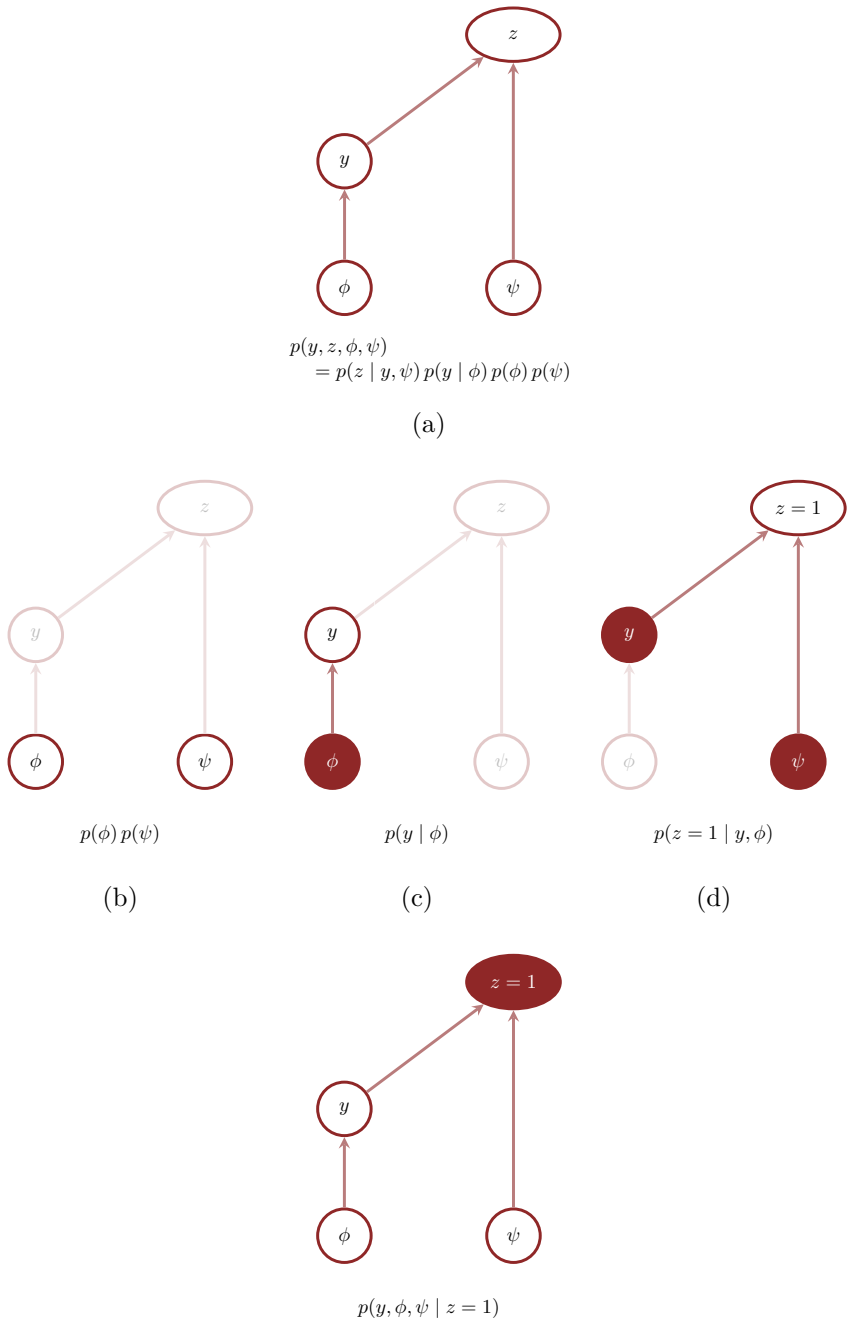


Figure 3: (a) A narratively generative full Bayesian model of a selection process is comprised of (b) a prior model for the parameters, (c) a latent model for the events before selection, and (d) a selection function. (e) When we observe only the selected events we have to condition this model on $z = 1$ which introduces a parameter-dependent normalization integral.

An observation with N independent accepted events,

$$\{(z_1 = 1, y_1), \ldots (z_N = 1, y_N)\},$$

and R independent rejected events,

$$\{z_{N+1} = 0, \ldots, z_{N+R} = 0\},$$

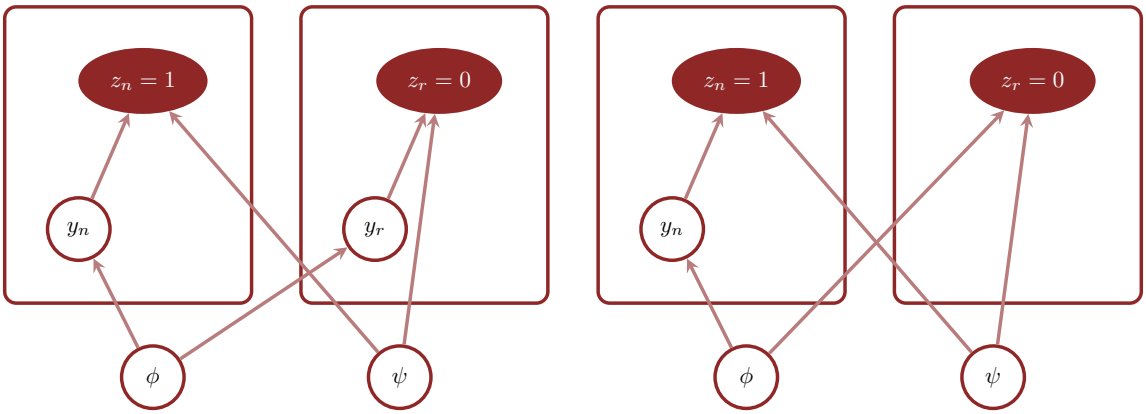
is then modeled as

$$\begin{aligned}
&p(z_1, y_1, \ldots, z_N, y_N, z_{N+1}, z_{N+R} \mid \phi, \psi) \\
&\qquad = \prod_{n=1}^{N} p(z_n = 1, y_n \mid \phi, \psi) \prod_{r=1}^{R} p(z_{N+r} = 0 \mid \phi, \psi) \\
&\qquad = \prod_{n=1}^{N} p(y_n \mid \phi, \psi)\, p(z_n = 1 \mid y_n, \phi, \psi) \prod_{r=1}^{R} p(z_{N+r} \mid \phi, \psi) \\
&\qquad = \prod_{n=1}^{N} p(y_n \mid \phi)\, S(y_n; \psi) \prod_{r=1}^{R} (1 - Z(\phi, \psi)) \\
&\qquad = \left(\prod_{n=1}^{N} p(y_n \mid \phi)\, S(y_n; \psi) \right) \left(1 - Z(\phi, \psi) \right)^R.
\end{aligned}$$

Note that because we are modeling *all* of the latent events, both accepted and rejected, we model the accepted events with the joint model $p(z = 1, y)$ instead of the conditional model $p(y \mid z = 1)$ and its possibly intractable normalization integral . That said the normalization still shows up in the model for the rejected events so that the implementation isn't actually any easier.

We can see the benefit of knowing the number of rejected events by rewriting this model in terms of the conditional model used for modeling accepted events,

$$\begin{aligned}
&p(z_1, y_1, \ldots, z_N, y_N, z_{N+1}, z_{N+R} \mid \phi, \psi) \\
&\qquad = \left(\prod_{n=1}^{N} p(y_n \mid \phi)\, S(y_n; \psi) \right) \left(1 - Z(\phi, \psi) \right)^R \\
&\qquad = \left(\prod_{n=1}^{N} \frac{p(y_n \mid \phi)\, S(y_n; \psi)}{Z(\phi, \psi)}\, Z(\phi, \psi) \right) \left(1 - Z(\phi, \psi) \right)^R \\
&\qquad = \left(\prod_{n=1}^{N} \frac{p(y_n \mid \phi)\, S(y_n; \psi)}{Z(\phi, \psi)} \right) \left(\prod_{n=1}^{N} Z(\phi, \psi) \right) \left(1 - Z(\phi, \psi) \right)^R \\
&\qquad = \left(\prod_{n=1}^{N} p(y \mid z = 1, \phi, \psi) \right) \left(\prod_{n=1}^{N} Z(\phi, \psi) \right) \left(1 - Z(\phi, \psi) \right)^R \\
&\qquad = \left(\prod_{n=1}^{N} p(y \mid z = 1, \phi, \psi) \right) Z(\phi, \psi)^N \left(1 - Z(\phi, \psi) \right)^R.
\end{aligned}$$

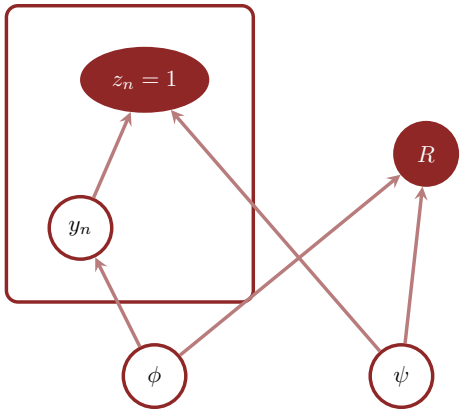


$\prod_{n=1}^{N} p(y_n, \phi, \psi \mid z_n = 1) \cdot \prod_{r=N+1}^{N+R} p(y_r, \psi \mid z_r = 0)$

(a)

$\prod_{n=1}^{N} p(y_n, \phi, \psi \mid z_n = 1) \cdot \prod_{r=N+1}^{N+R} p(\psi \mid z_r = 0)$

(b)



$\prod_{n=1}^{N} p(y_n, \phi, \psi \mid z_n = 1) \cdot Z(\phi, \psi)^R$

(c)

Figure 4: In order to incorporate the number of rejected events we (a) separate all events into those that are observed, $z_n = 1$, and those that are unobserved, $z_r = 0$ before (b) marginalizing out the value of the unobserved events and then finally (c) collapsing the unobserved indicators into the total number of rejected events, $R = \sum_{r=1}^{R} (1 - z_r)$.

Writing the total number of latent events as $M = N + R$ this becomes

$$\begin{aligned}
&p(z_1, y_1, \ldots, z_N, y_N, z_{N+1}, z_{N+R} \mid \phi, \psi) \\
&\quad = \left(\prod_{n=1}^{N} p(y \mid z = 1, \phi, \psi) \right) Z(\phi, \psi)^N \left(1 - Z(\phi, \psi) \right)^R \\
&\quad = \left(\prod_{n=1}^{N} p(y \mid z = 1, \phi, \psi) \right) Z(\phi, \psi)^N \left(1 - Z(\phi, \psi) \right)^{M-N} \\
&\quad = \left(\prod_{n=1}^{N} p(y \mid z = 1, \phi, \psi) \right) \text{binomial} \left(N \mid M, Z(\phi, \psi) \right).
\end{aligned}$$

This additional binomial probability mass function directly informs $Z(\phi, \psi)$, and hence indirectly informs ϕ and ψ, beyond what we would learn from the accepted events alone!

For example observing a large number of rejections relative to the number of acceptances, $R \gg N$, restricts $Z(\phi, \psi)$ to small values. Small values of $Z(\phi, \psi)$, however, require that the latent probability density function and selection function are misaligned. This misalignment doesn't directly inform ϕ and ψ, but it can be a powerful complement to the information provided by the accepted events.

1.3 Types of Selection Processes

The inferential consequences and implementation details of selection models depend on the nature of the selection process and the resulting structure of the selection function. In particular there are two main categories of selection functions.

1.3.1 Deterministic Selection

If the selection function outputs only extreme values,

$$S(y; \psi) : Y \to \{0, 1\},$$

then all latent events with a given value y are either *always* accepted or *always* rejected. In other words the selection process becomes completely **deterministic** once we know the value of the latent event.

Deterministic selection functions partition the observational space into the subset of latent values that are always accepted, $S^{-1}(1; \psi) \subset Y$, and the subset of latent values that are always

rejected, $S^{-1}(0; \psi) \subset Y$. Moreover the normalization reduces to the probability allocated to the acceptance subset,

$$\begin{aligned} Z(\phi, \psi) &= \int \mathrm{d}y' \, p(y' \mid \phi) \, S(y'; \psi) \\ &= \int \mathrm{d}y' \, p(y' \mid \phi) \, I_{S^{-1}(1;\psi)}(y') \\ &= \int_{S^{-1}(1;\psi)} \mathrm{d}y' p(y' \mid \phi) \\ &= \pi(S^{-1}(1; \psi)). \end{aligned}$$

Consequently a deterministic selection process is equivalent to restricting the latent probability distribution to the subset $S^{-1}(1; \psi)$,

$$\begin{aligned} p(y \mid \phi, \psi, z = 1) &= \frac{p(y \mid \phi) \, S(y; \psi)}{\int \mathrm{d}y' \, p(y' \mid \phi) \, S(y'; \psi)} \\ &= \frac{p(y \mid \phi) \, I_{S^{-1}(1;\psi)}(y)}{\pi(S^{-1}(1; \psi))}. \end{aligned}$$

When the observational space is one-dimensional and ordered the acceptance subset $S^{-1}(1; \psi)$ often decomposes into a union of continuous intervals. In this case we can evaluate $\pi(S^{-1}(1; \psi))$, and hence the normalization integral $Z(\phi, \psi)$, with various cumulative distribution function calculations. If we can evaluate the cumulative distribution function for the latent model analytically then the implementation of these particular deterministic selection models is straightforward.

That said implementing inference for deterministic selection models requires some care. Any observation $\tilde{y}$ by definition has been accepted by the selection process and has to fall in the acceptance subset,

$$\tilde{y} \in S^{-1}(1; \psi).$$

Consequently any model configuration with $S(\tilde{y}; \psi) = 0$ is completely inconsistent with the observation and must be excluded from our inferences. When deriving inferences from multiple observations

$$\{\tilde{y}_1, \ldots, \tilde{y}_N\}$$

we can include only those model configurations where the acceptance subset contains all of the observations,

$$\{\tilde{y}_1, \ldots, \tilde{y}_N\} \in S^{-1}(1; \psi).$$

Any computational method attempting to quantify a posterior distribution derived from a deterministic selection model will then have to avoid model configurations that yield $S(\tilde{y}_n; \psi) = 0$. For example when using Markov chain Monte Carlo we have to find initial states that satisfy this condition and then reject all Markov transitions to states that violate this condition.

Unfortunately the influence of ψ on the zero level set $S^{-1}(0; \psi)$ is subtle and difficult to quantify in many applications. Identifying the good model configurations with $S(\tilde{y}_n; \psi) > 0$ from the bad model configurations with $S(\tilde{y}_n; \psi) = 0$ is often much easier said than done.

1.3.2 Probabilistic Selection

When the selection function outputs an intermediate value for a given $y \in Y$,

$$0 < S(y; \psi) < 1,$$

then the acceptance of latent events with this value will no longer be deterministic and we cannot perfectly predict whether or not the event will be accepted or rejected. When

$$0 < S(y; \psi) < 1$$

for *all* $y \in Y$ then the selection function is described as **probabilistic** or, depending on the interpretation of the probabilistic selection, sometimes **stochastic** or even **random**.

Probabilistic selections functions that never output zero,

$$0 < S(y; \psi)$$

for all $y \in Y$, are particularly well-behaved when it comes to implementing inferences. In this case no model configurations are completely incompatible with observed data. An initial model configuration might imply that the observations are extremely unlikely, but because they are not outright impossible we will be able to follow the shape of the posterior distribution to more reasonable model configurations.

2 Computational Strategies For Evaluating Normalization Constants

In an ideal selection model the normalization integral

$$Z(\phi, \psi) = \int \mathrm{d}y' \, p(y' \mid \phi) \, S(y'; \psi)$$

can be evaluated in closed form, allowing us to analytically evaluate the normalization constant for any model configuration (ϕ, ψ). Unfortunately these ideal selection models are rare in practice.

When we cannot evaluate the normalization analytically we have to rely on numerical integration techniques that can *approximate* it,

$$\hat{Z}(\phi, \psi) \approx Z(\phi, \psi).$$

In this section we'll consider a few techniques that provide deterministic estimates for the normalization integral,

$$\hat{Z}(\phi, \psi) \approx Z(\phi, \psi),$$

and hence deterministic approximations to the full Bayesian probability density function,

$$\begin{aligned}
\hat{p}(y, \phi, \psi \mid z = 1) &= \frac{p(y' \mid \phi)\, S(y'; \psi)\, p(\psi, \phi)}{\hat{Z}(\phi, \psi)} \\
&\approx \frac{p(y' \mid \phi)\, S(y'; \psi)\, p(\psi, \phi)}{Z(\phi, \psi)} \\
&= p(y, \phi, \psi \mid z = 1),
\end{aligned}$$

and finally deterministic approximations to posterior density functions,

$$\hat{p}(\phi, \psi \mid \tilde{y}, \tilde{z} = 1) \propto \hat{p}(\tilde{y}, \phi, \psi \mid \tilde{z} = 1) \approx p(\tilde{y}, \phi, \psi \mid \tilde{z} = 1) \propto p(\phi, \psi \mid \tilde{y}, \tilde{z} = 1).$$

We can then use tools like Hamiltonian Monte Carlo to estimate expectation values with respect to these approximate posterior density functions. If the approximate posterior density function is close enough to the exact posterior density function then this can provide reasonably accurate estimates of the exact posterior expectation values,

$$\int \mathrm{d}\phi\, \mathrm{d}\psi\, \hat{p}(\phi, \psi \mid \tilde{y}, \tilde{z} = 1) f(\phi, \psi) \approx \int \mathrm{d}\phi\, \mathrm{d}\psi\, p(\phi, \psi \mid \tilde{y}, \tilde{z} = 1) f(\phi, \psi).$$

While we can often quantify the error in estimating the normalization integral, propagating that error through to the posterior density function estimate and then any resulting posterior expectation value estimates is much more difficult and outright infeasible for most practical problems. This makes it difficult to determine just how accurate our numerical integration need to be to ensure faithful posterior inferences.

In some cases we can verify that the consequences of the normalization estimator error are negligible by repeating the approximate posterior quantification with more accurate, although more expensive, estimators until the resulting posterior inferences appear to stabilize. The main limitation of this approach is that these consequences don't always vary smoothly with the normalization error itself: apparent stability doesn't always imply negligible error.

Another approach is to consider methods like Simulation-Based Calibration (Talts et al. 2018). Any normalization estimator error will introduce an inconsistency between the model used to simulate prior predictive data and the model used to construct posterior samples. This inconsistency will then bias the prior-posterior ranks which should manifest in the Simulation-Based Calibration rank histograms. That said we might need an excess number of replications, and hence an excess amount of computation, to resolve this bias.

2.1 Numerical Quadrature

When the observational space is one-dimensional numerical quadrature becomes a feasible tool for estimating the normalization integral.

Given a grid

$$\{y_0, \ldots, y_k, \ldots, y_K\} \in Y$$

we can construct a crude quadrature estimate of the normalization constant as

$$\begin{aligned} \hat{Z}(\phi, \psi) &= \sum_{k=1}^{K} (y_k - y_{k-1}) \, p(y_k \mid \phi) \, S(y_k; \psi) \\ &\approx \int \mathrm{d}y' \, p(y' \mid \phi) \, S(y'; \psi). \end{aligned}$$

The finer the grid the more we have to evaluate the latent probability density function and selection function, and the more computationally expensive the estimator becomes, but the smaller of an error we can achieve.

Modern numerical quadrature tools dynamically construct a grid based on the shape of the integrand, here $S(y_k; \psi) \, p(y_k \mid \phi)$, to guarantee extremely small errors with as few evaluations as possible.

Unfortunately the effectiveness of numerical quadrature decays rapidly with the dimension of the observational space. In order to maintain a fixed error the cost of numerical quadrature needs to increase exponentially with the dimension. At the same time if we fix the cost then the error will grow exponentially with dimension. Moreover as we go to higher and higher-dimensional observational spaces robust dynamic grid adaptation becomes more difficult. Ultimately numerical quadrature is most productive for one-dimensional problems, with exceptional applications for two and three-dimensional problems.

2.2 Monte Carlo Estimation

Recall that the normalization integral can be written as the expectation value of the selection function with respect to the latent probability distribution,

$$\begin{aligned} Z(\psi, \phi) &= \int \mathrm{d}y' \, p(y' \mid \phi) \, S(y'; \psi)) \\ &= \int \pi(\mathrm{d}y' \mid \phi) \, S(y'; \psi) \\ &= \int \pi_{\phi}(\mathrm{d}y') \, S(y'; \psi) \\ &= \mathbb{E}_{\pi_{\phi}}[S(\cdot; \psi)]. \end{aligned}$$

Consequently we can use expectation value estimation techniques to approximate the normalization integral.

For example we can use Monte Carlo estimation to approximate the normalization integral. Given an ensemble of exact samples

$$\{\tilde{y}_1, \ldots, \tilde{y}_j, \ldots, \tilde{y}_J\} \sim \pi_\phi$$

we can construct the Monte Carlo estimator

$$\frac{1}{J} \sum_{j=1}^{J} S(\tilde{y}_j; \psi) \approx \mathbb{E}_{\pi_\phi}[S(\cdot; \psi)] = Z(\psi, \phi).$$

We can then use the Monte Carlo central limit theorem to estimate the error of this estimator. Moreover if we ever need to decrease the error then we can increase the ensemble size J as needed.

As we consider higher and higher-dimensional observational spaces the cost of generating exact samples will in general increase, although the growth is often more linear than exponential. The Monte Carlo estimator error, however, will remain stable. This makes Monte Carlo particularly attractive for higher-dimensional selection problems.

That said any changes to the configuration of the latent probability distribution or selection function complicate the incorporation of Monte Carlo estimates into selection models. Varying ψ changes the expectand which requires computing a new Monte Carlo estimator. At the same time varying ϕ changes the latent probability distribution which requires generating a new ensemble of samples and then re-evaluating the Monte Carlo estimator.

To accommodate variations in the behavior of the latent probability distribution we would have to generating a new ensemble every time we evaluate the selection model. Consequently we would not longer be working with a fixed approximate model $\widehat{p}(y, \phi, \psi \mid z = 1)$ but rather a *stochastic* one. This, in turn, disrupts the scalable performance of computational tools like Hamiltonian Monte Carlo (Betancourt 2015).

When the configuration of the latent probability distribution is fixed, however, we can generate a *single* ensemble of latent samples and use it estimate the normalization integral for any given configuration of the selection function. We just have to be careful that the ensemble is large enough to ensure that the Monte Carlo estimator error is small for *all* selection function configurations of interest. In particular if we ever need to evaluate the selection model when the selection function is strongly misaligned with the latent probability distribution then we will need relatively large ensembles to ensure that Monte Carlo estimator error is always well-behaved.

One way to monitor for any problems is to compute and save the Monte Carlo estimator error at each evaluation of the model and check that the entire ensemble of errors is sufficiently small.

2.3 Importance Sampling Estimation

Unfortunately the convenience of Monte Carlo estimation is rarely of use in applied problems given that the behavior of the latent probability distribution is almost never known precisely. One way to infer the latent probability distribution at the same time as the selection function is to generate one ensemble of samples and adapt them to the varying behavior of the latent probability distribution. More formally we can replace our Monte Carlo estimator with an *importance sampling* estimator.

In this case we would generate an ensemble of samples from a convenient reference probability distribution ρ,

$$\{\tilde{y}_1, \ldots, \tilde{y}_j, \ldots, \tilde{y}_J\} \sim \rho(y)$$

and then estimate the normalization as

$$\frac{1}{J} \sum_{j=1}^{J} w(\tilde{y}_j; \phi)\, S(\tilde{y}_j; \psi) \approx \mathbb{E}_{\pi_\phi}[S(\cdot; \psi)] = Z(\psi, \phi),$$

where the importance weights account for the difference between the reference probability distribution and the current configuration of the latent probability distribution,

$$w(y, \phi) = \frac{\pi(y \mid \phi)}{\rho(y)}.$$

Unfortunately error quantification for importance sampling estimators is much more fragile than it is for Monte Carlo estimators. When the reference probability distribution is sufficiently similarly to the latent probability distribution then the importance weights will concentrate around 1 and the error in estimating the expectation value $\mathbb{E}_{\pi_\phi}[f]$ will be well-approximated by

$$\epsilon[f] = \sqrt{\frac{\mathrm{Var}_\rho[w\, f]}{J}}.$$

In practice we can estimate the variance $\mathrm{Var}_\rho[w\, f]$ with the empirical variance of the individual products

$$\{w(\tilde{y}_1; \phi)\, f(\tilde{y}_1), \ldots, w(\tilde{y}_j; \phi)\, f(\tilde{y}_j), \ldots, w(\tilde{y}_J; \phi)\, f(\tilde{y}_J)\}.$$

If the reference probability distribution is not similar enough to the latent probability distribution, however, then the distribution of the importance weights will exhibit heavy tails and the resulting importance sampling estimators, let alone the estimator error quantification, will be unreliable. In particular most of the importance weights will be very close to zero with only a few taking on extremely large values. The small weights will have negligible contributions to the importance sampling estimator, and the effective ensemble size will be very small.

When applying importance sampling estimates to the normalization integral in selection models we cannot tune the reference probability distribution to a single latent probability distribution.

Instead we need the reference probability distribution to span *all* of the latent probability distribution behaviors in neighborhoods of high posterior probability (Figure 5). The larger our posterior uncertainties the wider the reference probability distribution will need to be, and the larger of an ensemble of points we will need to ensure reasonable importance sampling estimator accuracy for any given configuration of the latent probability distribution.

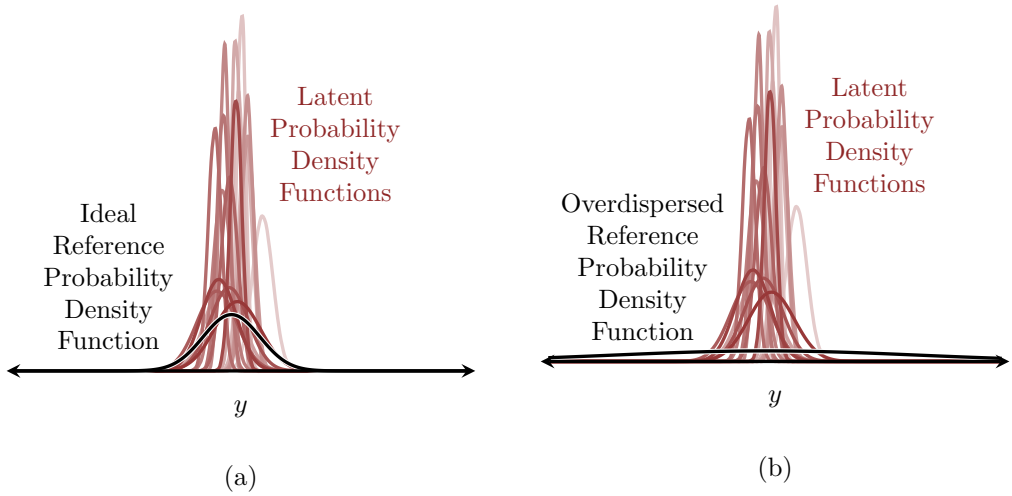


Figure 5: The choice of reference distribution is critical to the performance of importance sampling estimation of the normalization integral in selection models. (a) Ideally the reference distribution would span only the latent probability distribution configurations with high posterior probability. (b) In practical applications, however, we don't know what those configurations will be and instead have to use more conservative proxies such as the latent probability distribution configurations with high prior probability. The more strongly the posterior distribution contracts away from the prior model the worse the resulting importance sampling estimators will perform.

In most practical applications, however, we will not know what latent probability distribution behaviors will be spanned by the posterior distribution before we attempt posterior computation. All we can do in this case is adapt the reference probability distribution to the prior model; the more informative the prior model is the better this strategy will be. With enough effort we can even approach this problem iteratively, starting with a reference probability distribution adapted to the prior model and then tweaking it based on the results from each fit.

Unfortunately as the dimension of the observational space increases the engineering of a productive reference probability distribution that spans all of the relevant latent probability distribution behaviors becomes more and more difficult, and quickly becomes impractical altogether.

Because of their fragility we have to be very careful about the verification of importance

sampling estimators. For example we can start by examining the importance weights generated when evaluating the normalization integral at each Markov chain state for any indications of a heavy tail, such as excessive zeros or large but sparse deviations. When the reference probability distribution isn't too far off then diagnostics like the importance sampling effective sample size and the $\hat{k}$ statistic (Vehtari et al. 2015) can also be helpful.

Critically all of these diagnostics are limited by the size of the ensemble. If the ensemble is too small then we can easily miss large but rare weights that indicate unreliable estimation. In practice it can be useful to run with multiple ensemble sizes to verify that the estimator behavior is consistent.

3 Demonstrations

To contextualize these ideas let's explore their application in a variety of simulated data exercises that span different selection function scenarios.

3.1 Setup

Before we start with our first exercise, however, let's set up our local `R` environment.

```
par(family="serif", las=1, bty="l",
    cex.axis=1, cex.lab=1, cex.main=1,
    xaxs="i", yaxs="i", mar = c(5, 5, 3, 5))

c_light <- c("#DCBCBC")
c_light_highlight <- c("#C79999")
c_mid <- c("#B97C7C")
c_mid_highlight <- c("#A25050")
c_dark <- c("#8F2727")
c_dark_highlight <- c("#7C0000")

c_light_teal <- c("#6B8E8E")
c_mid_teal <- c("#487575")
c_dark_teal <- c("#1D4F4F")
```

In particular we'll need to load `RStan` and my recommended Hamiltonian Monte Carlo analysis suite.

```
library(rstan)
rstan_options(auto_write = TRUE)            # Cache compiled Stan programs
options(mc.cores = parallel::detectCores()) # Parallelize chains
parallel:::setDefaultClusterOptions(setup_strategy = "sequential")

util <- new.env()
source('stan_utility_rstan.R', local=util)
```

3.2 Deterministic Selection

For our first example let's consider a one-dimensional, real-valued observational space $Y = \mathbb{R}$ and a deterministic selection process that always rejects events with values above a certain threshold $\psi = \lambda \in Y$. Because this selection process limits the size of latent values that can be observed it is also known as **truncation**.

We can model this selection process with the selection function

$$S(y; \lambda) = \begin{cases} 1, & y \leq \lambda \\ 0, & y > \lambda \end{cases}$$

The resulting normalization integral is

$$\begin{aligned} Z(\phi, \lambda) &= \int \mathrm{d}y' \, p(y' \mid \phi) \, S(y'; \lambda) \\ &= \int \mathrm{d}y' \, p(y' \mid \phi) \, I_{(-\infty, \lambda]}(y') \\ &= \int_{-\infty}^{\lambda} \mathrm{d}y' p(y'; \phi), \\ &= \Pi(\lambda; \phi), \end{aligned}$$

where Π is the cumulative distribution function of the latent probability distribution.

In order to avoid selection function configurations that are inconsistent with the observed data, $S(\tilde{y}; \lambda) = 0$, we need to bound the selection threshold below by the observed data,

$$\tilde{y} < \lambda.$$

If we have multiple observations

$$\{\tilde{y}_1, \ldots, \tilde{y}_N\}$$

then we need to bound λ below by the largest observed value,

$$\max\left(\{\tilde{y}_1, \ldots, \tilde{y}_N\}\right) < \lambda.$$

Now let's assume that the latent probability distribution is specified by a one-dimensional, normal density function with the parameters $\phi = (\mu, \tau)$,

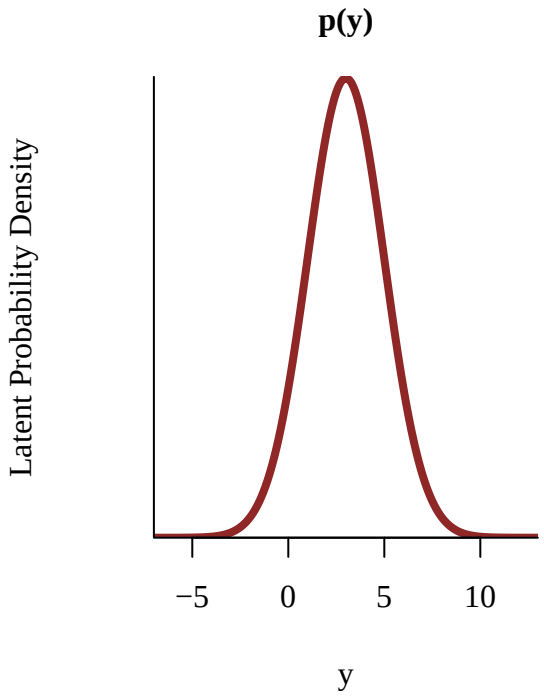
$$p(y \mid \mu, \tau) = \text{normal}(y \mid \mu, \tau)$$

In this case the normalization integral reduces to

$$\begin{aligned} Z(\mu, \tau, \lambda) &= \int \mathrm{d}y' \, p(y' \mid \mu, \tau) \, S(y'; \lambda) \\ &= \int \mathrm{d}y' \, \text{normal}(y' \mid \mu, \tau) \, I_{(-\infty, \lambda]}(y') \\ &= \int_{-\infty}^{\lambda} \mathrm{d}y' \, \text{normal}(y' \mid \mu, \tau) \\ &= \Phi_{\text{normal}}(\lambda \mid \mu, \tau). \end{aligned}$$

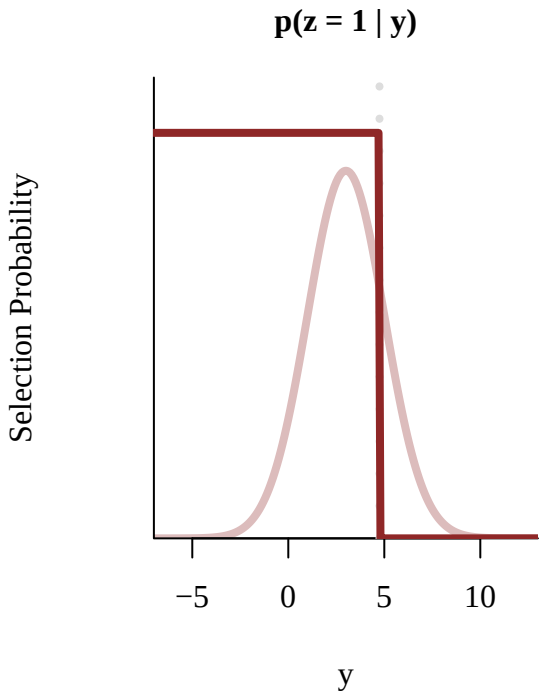
For example taking $\mu = 3$ and $\tau = 2$ gives a latent probability density function that concentrates around $y = 3$.

```
delta <- 0.1
ys <- seq(-7, 13, delta)

mu <- 3
tau <- 2

latent_pds <- dnorm(ys, mu, tau)

plot(ys, latent_pds, type="l", main ="p(y)", col=c_dark, lwd=4,
     xlab="y", xlim=c(-7, 13),
     ylab="Latent Probability Density", yaxt='n', ylim=c(0, 0.2))
```



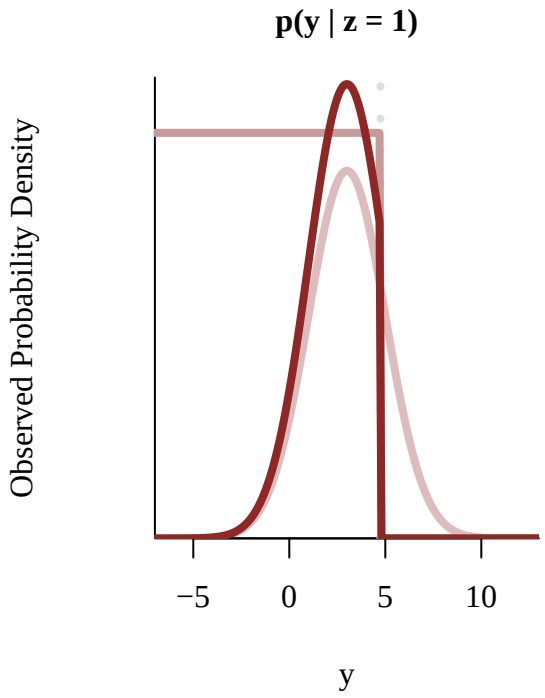
Taking $\lambda = 4.75$ defines a selection function that truncates all values above $y = 4.75$.

```
lambda <- 4.75

selection_probs <- ifelse(ys <= lambda, 1, 0)

plot(ys, latent_pds, type="l", main ="p(z = 1 | y)", col=c_light, lwd=4,
     xlab="y", xlim=c(-7, 13),
     ylab="Selection Probability", yaxt='n', ylim=c(0, 0.25))
abline(v = lambda, col="#DDDDDD", lty=3, lwd=4)
lines(ys, 0.22 * selection_probs, col=c_dark, lwd=4)
```



Together these components define an observed probability density function that abruptly cuts off at the truncation point.

```
observed_pds <- (selection_probs * latent_pds)
norm <- pnorm(lambda, mu, tau)
observed_pds <- observed_pds / norm

plot(ys, latent_pds, type="l", main ="p(y | z = 1)", col=c_light, lwd=4,
     xlab="y", xlim=c(-7, 13),
     ylab="Observed Probability Density", yaxt='n', ylim=c(0, 0.25))
abline(v = lambda, col="#DDDDDD", lty=3, lwd=4)
lines(ys, 0.22 * selection_probs, col=c_light_highlight, lwd=4)
lines(ys, observed_pds, col=c_dark, lwd=4)
```



Note that below the threshold the observed probability density function is larger than the latent probability density function because the probability that would be allocated above the threshold instead has to be reallocated to values below the threshold.

3.2.1 Simulating Data

The most efficient way to generate data is to sample directly from the observed probability distribution specified by the probability density function

$$p(y \mid \mu, \tau, \lambda, z = 1) = \frac{\text{normal}(y \mid \mu, \tau)\, I_{(-\infty, \lambda]}(y)}{\Pi_{\text{normal}}(\lambda \mid \mu, \tau)}.$$

Conveniently sampling from this particular observed probability distribution is straightforward with a small modification to the quantile function pushforward method.

First we compute the cumulative probability of the selection threshold,

$$r = \Pi_{\text{normal}}(\lambda \mid \mu, \tau).$$

Next we simulate a uniform variate in the interval $(0, r)$,

$$\tilde{q} \sim \text{uniform}(0, r).$$

Finally we apply the normal quantile function,

$$\tilde{y} = \Pi^{-1}_{\text{normal}}(\tilde{q} \mid \mu, \tau).$$

More generally we can also simulate the selection process directly. Here we generate a candidate sample from the latent probability distribution,

$$\tilde{y} \sim \pi_{\phi},$$

and then a binary acceptance variable

$$\tilde{z} \sim \text{Bernoulli}(S(\tilde{y}; \psi)).$$

If $\tilde{z} = 0$ then we reject $\tilde{y}$ and generate a new candidate sample. We then repeat these steps until $\tilde{z} = 1$.

Although straightforward this approach can also become extremely inefficient if the overlap between the latent probability distribution and selection function is weak. In this case we will need to generate many candidate samples, expend a lot of computation, and wait a long time for every acceptance.

The direct simulation approach is implemented in the following Stan program for the ground truth that we introduced above,

$$\begin{aligned} \mu &= 3 \\ \tau &= 2 \\ \lambda &= 4.75. \end{aligned}$$

In general the while loop here can be dangerous because there's no guarantee that it will terminate in a finite time. Because the overlap between the latent probability distribution and selection function is reasonably large here, however, this shouldn't be a problem.

To ensure a pretty substantial observation we'll take $N = 1000$.

```
N <- 1000

simu <- stan(file="stan_programs/simu_threshold.stan",
             iter=1, warmup=0, chains=1, data=list("N" = N),
             seed=4838282, algorithm="Fixed_param")
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

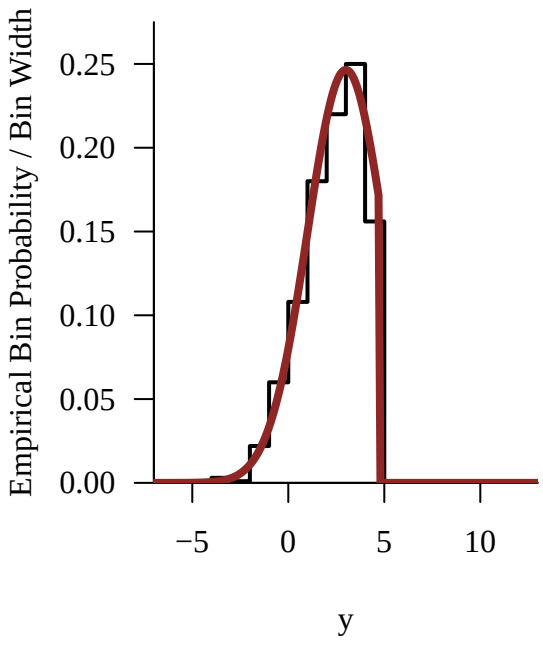
```
y <- extract(simu)$y[1,]
N_reject <- extract(simu)$N_reject[1]
```

To double check our simulation implementation we can compare a properly normalized histogram of the component observations to the true observed probability density function. Fortunately there don't appear to be any conflicts here.

```
plot_line_hist <- function(s, bin_min=NULL, bin_max=NULL, delta=NULL,
                           prob=FALSE, xlab="y", main="") {
  # Remove any NA values
  s <- s[!is.na(s)]

  # Construct binning configuration
  if (is.null(bin_min))
    bin_min <- min(s)
  if (is.null(bin_max))
    bin_max <- max(s)
  if (is.null(delta))
    delta <- (bin_max - bin_min) / 25

  # Construct bins
  bins <- seq(bin_min, bin_max, delta)
  B <- length(bins) - 1
  idx <- rep(1:B, each=2)
  x <- sapply(1:length(idx),
              function(b) if(b %% 2 == 1) bins[idx[b]] else bins[idx[b] + 1])
  x <- c(bin_min - 10, x, bin_max + 10)

  # Check bin containment
  N <- length(s)

  N_low <- sum(s < bin_min)
  if (N_low > 0)
    warning(sprintf('%s values (%.1f%%) fell below the histogram binning.',
                    N_low, 100 * N_low / N))

  N_high <- sum(bin_max < s)
  if (N_high > 0)
    warning(sprintf('%s values (%.1f%%) fell above the histogram binning.',
                    N_high, 100 * N_high / N))

  # Compute bin contents, including empty bounding bins
```

```
  counts <- hist(s[bin_min <= s & s <= bin_max], breaks=bins, plot=FALSE)$counts

  ylab <- "Counts"
  if (prob) {
    ylab <- "Empirical Bin Probability / Bin Width"
    # Normalize bin contents, if desired
    counts <- counts / (delta * sum(counts))
  }

  y <- counts[idx]
  y <- c(0, y, 0)

  # Plot
  ymax <- 1.1 * max(y)

  plot(x, y, type="l", main=main, col="black", lwd=2,
       xlab=xlab, xlim=c(bin_min, bin_max),
       ylab=ylab, ylim=c(0, ymax))
}
```

```
plot_line_hist(y, -7, 13, 1, prob=TRUE, xlab="y")
lines(ys, observed_pds, col=c_dark, lwd=4)
```



Additionally if our simulation is correct then the ratio of the total number of acceptances to

the expected number of rejections will be

$$\frac{\left\langle N_{\text{reject}} \right\rangle}{N} = \frac{1 - Z(\mu, \tau, \lambda)}{Z(\mu, \tau, \lambda)}.$$

Although we can't expect exact equality the ratio N_{reject}/N should be near the value on the right-hand side. Fortunately for us, it is.

```
N_reject / N; (1 - norm) / norm
```

```
[1] 0.233
```

```
[1] 0.2357685
```

With our simulation validated let's try to actually fit the simulated data.

3.2.2 Ignoring The Selection Process

To begin we'll be careless and ignore the selection process entirely, assuming that the data are drawn directly from the latent probability distribution.

```
data <- list("N" = N, "y" = y)

fit1 <- stan(file="stan_programs/fit_threshold1.stan",
             data=data, seed=8438338,
             warmup=1000, iter=2024, refresh=0)
```

There are no signs of computational problems.

```
diagnostics <- util$extract_hmc_diagnostics(fit1)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit1)
base_samples <- util$filter_expectands(samples,
                                       c('mu', 'tau'))
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

The retrodictive performance, however, leaves much to be desired.

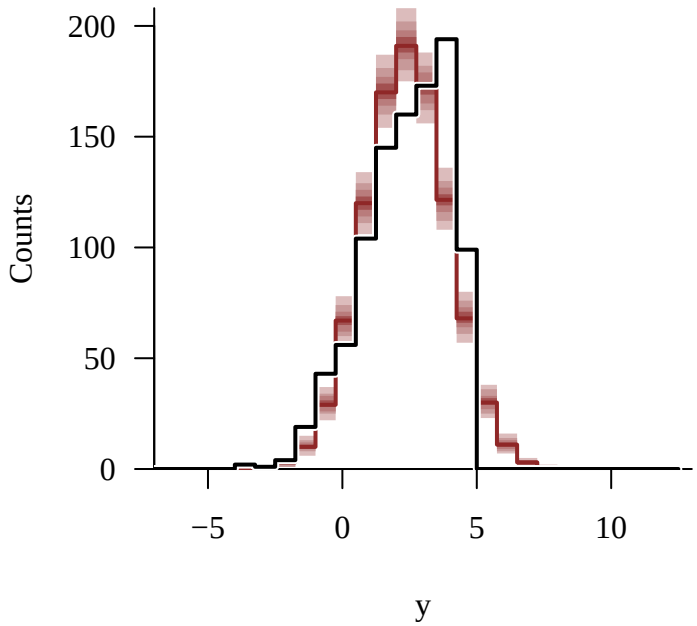
```
hist_retro <- function(obs, samples, pred_names,
                       bin_min=NULL, bin_max=NULL, delta=NULL,
                       xlab="", ylim=NA, title="") {
  # Check that pred_names are in samples
  all_names <- names(samples)

  bad_names <- setdiff(pred_names, all_names)
  if (length(bad_names) > 0) {
    warning(sprintf('The expectand names %s are not in the `samples` object and will be ignor
                    paste(bad_names, collapse=", ")))
  }

  good_names <- intersect(pred_names, all_names)
  if (length(good_names) == 0) {
    stop('There are no valid expectand names.')
  }
  pred_names <- good_names

  # Remove any NA values
  obs <- obs[!is.na(obs)]

  pred <- sapply(pred_names,
                 function(name) c(t(samples[[name]]), recursive=TRUE))
  pred_collapse <- c(pred)
  pred_collapse <- pred_collapse[!is.na(pred_collapse)]

  # Construct binning configuration
  if (is.null(bin_min))
    bin_min <- min(min(obs), min(pred_collapse))
  if (is.null(bin_max))
    bin_max <- max(max(obs), max(pred_collapse))
  if (is.null(delta))
    delta <- (bin_max - bin_min) / 25

  # Construct bins
  breaks <- seq(bin_min, bin_max, delta)
  B <- length(breaks) - 1
```

```
idx <- rep(1:B, each=2)
xs <- sapply(1:length(idx),
             function(b) if(b %% 2 == 0) breaks[idx[b] + 1]
             else            breaks[idx[b]] )

# Check bin containment for observed values
N <- length(obs)

N_low <- sum(obs < bin_min)
if (N_low > 0)
  warning(sprintf('%s data values (%.1f%%) fell below the histogram binning.',
                  N_low, 100 * N_low / N))

N_high <- sum(bin_max < obs)
if (N_high > 0)
  warning(sprintf('%s data values (%.1f%%) fell above the histogram binning.',
                  N_high, 100 * N_high / N))

# Compute observed bin contents
obs_counts <- hist(obs[bin_min <= obs & obs <= bin_max],
                   breaks=breaks, plot=FALSE)$counts
pad_obs_counts <- do.call(cbind,
                          lapply(idx, function(n) obs_counts[n]))

# Check bin containment for posterior predictive values
N <- length(pred_collapse)

N_low <- sum(pred_collapse < bin_min)
if (N_low > 0)
  warning(sprintf('%s predictive values (%.1f%%) fell below the histogram binning.',
                  N_low, 100 * N_low / N))

N_high <- sum(bin_max < pred_collapse)
if (N_high > 0)
  warning(sprintf('%s predictive values (%.1f%%) fell above the histogram binning.',
                  N_high, 100 * N_high / N))

# Construct ribbons for predictive bin contents
N <- dim(pred)[1]
pred_counts <- sapply(1:N,
                      function(n) hist(pred[n, bin_min <= pred[n,] &
                                               pred[n,] <= bin_max],
```

```
                                          breaks=breaks,
                                          plot=FALSE)$counts)
  probs = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)
  cred <- sapply(1:B,
                 function(b) quantile(pred_counts[b,], probs=probs))
  pad_cred <- do.call(cbind, lapply(idx, function(n) cred[1:9, n]))
  if (any(is.na(ylim))) {
    ylim <- c(0, max(c(obs_counts, cred[9,])))
  }

  # Plot
  plot(1, type="n", main=title,
       xlim=c(bin_min, bin_max), xlab=xlab,
       ylim=ylim, ylab="Counts")
  polygon(c(xs, rev(xs)), c(pad_cred[1,], rev(pad_cred[9,])),
          col = c_light, border = NA)
  polygon(c(xs, rev(xs)), c(pad_cred[2,], rev(pad_cred[8,])),
          col = c_light_highlight, border = NA)
  polygon(c(xs, rev(xs)), c(pad_cred[3,], rev(pad_cred[7,])),
          col = c_mid, border = NA)
  polygon(c(xs, rev(xs)), c(pad_cred[4,], rev(pad_cred[6,])),
          col = c_mid_highlight, border = NA)
  lines(xs, pad_cred[5,], col=c_dark, lwd=2)

  lines(xs, pad_obs_counts, col="white", lty=1, lw=4)
  lines(xs, pad_obs_counts, col="black", lty=1, lw=2)
}
```

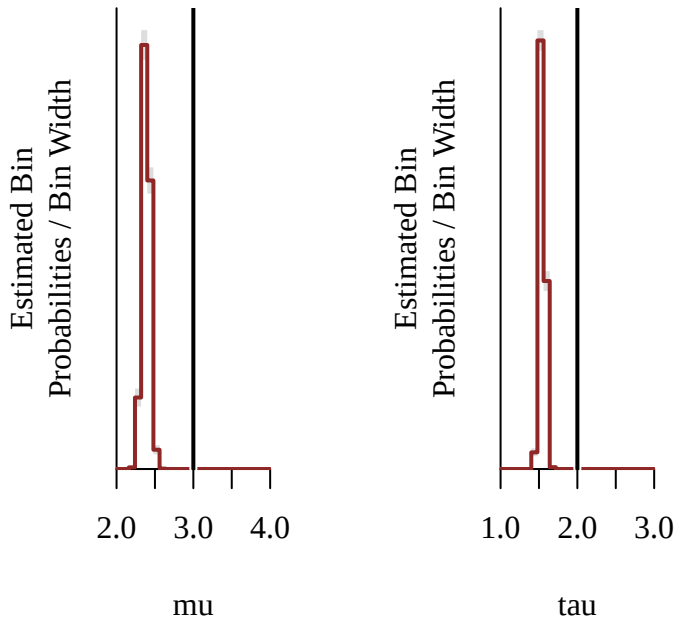
While the observed histogram exhibits a clear asymmetry, with a hard cut off around $y = 5$, the posterior predictive distribution concentrates on symmetric histograms that not only peak at smaller values of y abut also exhibit a tail that extends far above the hard cut off seen in the data.

```
par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))

pred_names <- grep('y_pred', names(samples), value=TRUE)
hist_retro(data$y, samples, pred_names, -7, 13, 0.75, 'y')
```



Moreover the resulting posterior inferences concentrate far away from the true values. Any decisions or predictions that we might derive from these inferences will perform poorly.

```
par(mfrow=c(1, 2))

util$plot_expectand_pushforward(samples[['mu']], 25, 'mu',
                                flim=c(2, 4), baseline=mu)
util$plot_expectand_pushforward(samples[['tau']], 25, 'tau',
                                flim=c(1, 3), baseline=tau)
```



3.2.3 Modeling The Selection Process

To obtain an accurate picture of the latent behavior we'll have to explicitly model the selection process. Fortunately this is straightforward given that the normalization integral can be evaluated as a normal cumulative distribution function.

In order to avoid model configurations with zero likelihood, and hence zero posterior density, we need to force the selection threshold λ to be larger than the largest observation.

Additionally to avoid excessive computation when the latent density function and selection function are poorly aligned we will cap the number of attempted latent simulations in the `generated quantities` block. Poor alignment is typically most problematic during the warmup phase, but we can readily diagnose any problems that persist into the sampling phase by looking for `NA` outputs.

```
fit2 <- stan(file="stan_programs/fit_threshold2.stan",
             data=data, seed=8438338,
             warmup=1000, iter=2024, refresh=0)
```

The diagnostics are clean.

```
diagnostics <- util$extract_hmc_diagnostics(fit2)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

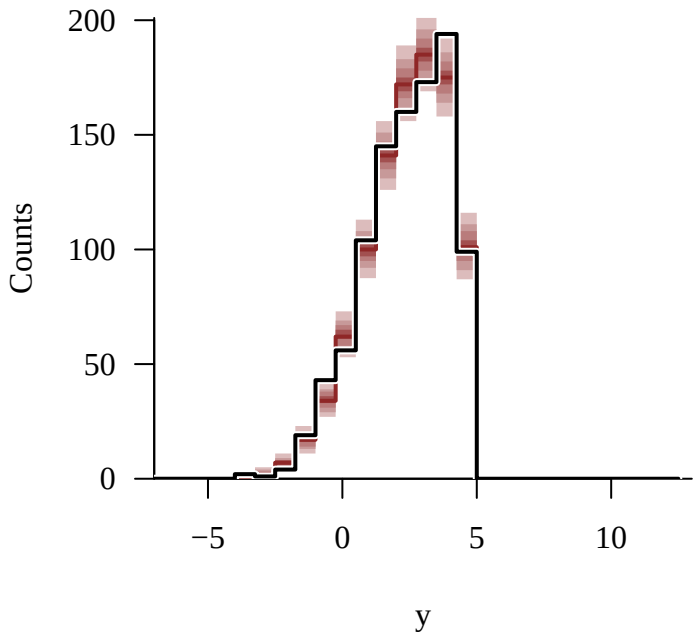
```
samples <- util$extract_expectands(fit2)
base_samples <- util$filter_expectands(samples,
                                       c('mu', 'tau', 'lambda'))
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

The lack of any retrodictive tension in the histogram summary statistic suggests that our selection model is accurately capturing the threshold behavior exhibited by the observed data.

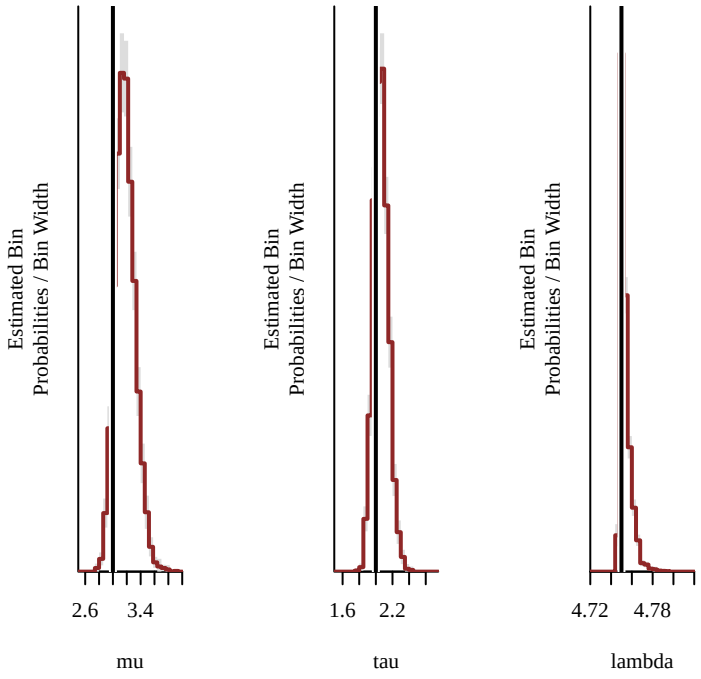
```
par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))

pred_names <- grep('y_pred', names(samples), value=TRUE)
hist_retro(data$y, samples, pred_names, -7, 13, 0.75, 'y')
```

```
Warning in hist_retro(data$y, samples, pred_names, -7, 13, 0.75, "y"): 6
predictive values (0.0%) fell below the histogram binning.
```



More importantly our posterior inferences are consistent with the true model configurations from which we simulated data.

```
par(mfrow=c(1, 3))

util$plot_expectand_pushforward(samples[['mu']], 25, 'mu',
                                flim=c(2.5, 4), baseline=mu)
util$plot_expectand_pushforward(samples[['tau']], 25, 'tau',
                                flim=c(1.5, 2.75), baseline=tau)
util$plot_expectand_pushforward(samples[['lambda']], 25, 'lambda',
                                flim=c(4.72, 4.82), baseline=lambda)
```



3.2.4 Incorporating The Number of Rejections

Finally let's go one step further and consider how our inferences might change if we knew not only that some latent events were rejected but also how many were rejected. Recall that in this case the accepted events no longer need to be modeled with a modified normalization, but that the normalization integral still shows up in the model for the rejected events.

```
data$N_reject <- N_reject

fit3 <- stan(file="stan_programs/fit_threshold3.stan",
             data=data, seed=8438338,
             warmup=1000, iter=2024, refresh=0)
```

Fortunately the computation remains unproblematic.

```
diagnostics <- util$extract_hmc_diagnostics(fit3)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit3)
base_samples <- util$filter_expectands(samples,
                                       c('mu', 'tau', 'lambda'))
util$check_all_expectand_diagnostics(base_samples)
```

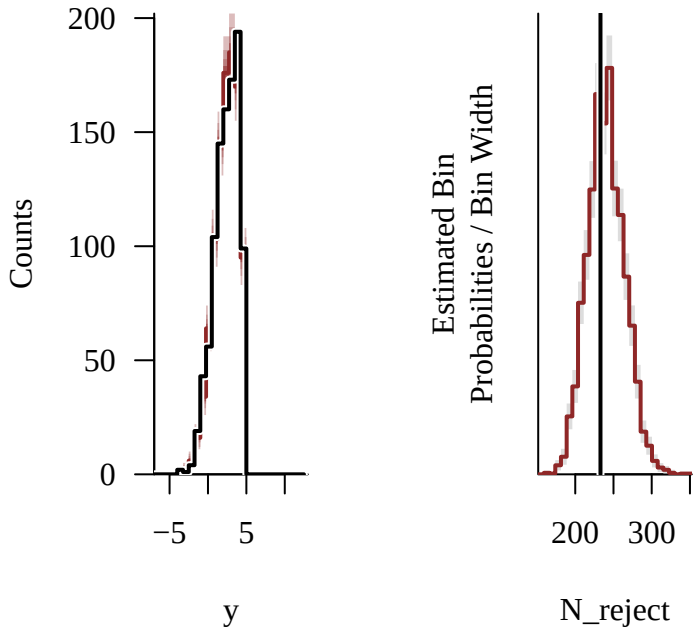
```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

In addition to the histogram summary statistic which is sensitive to the behavior of the accepted events we can also consider the number of rejected events as its own summary statistic. Our model is consistent with the observed behavior of both of these statistics, giving us confidence that we're capturing the relevant behavior.

```
par(mfrow=c(1, 2), mar = c(5, 5, 3, 1))

pred_names <- grep('y_pred', names(samples), value=TRUE)
hist_retro(data$y, samples, pred_names, -7, 13, 0.75, 'y')
```

```
Warning in hist_retro(data$y, samples, pred_names, -7, 13, 0.75, "y"): 1
predictive values (0.0%) fell below the histogram binning.
```

```
util$plot_expectand_pushforward(samples[['N_reject_pred']], 25, 'N_reject',
                                baseline=data$N_reject)
```

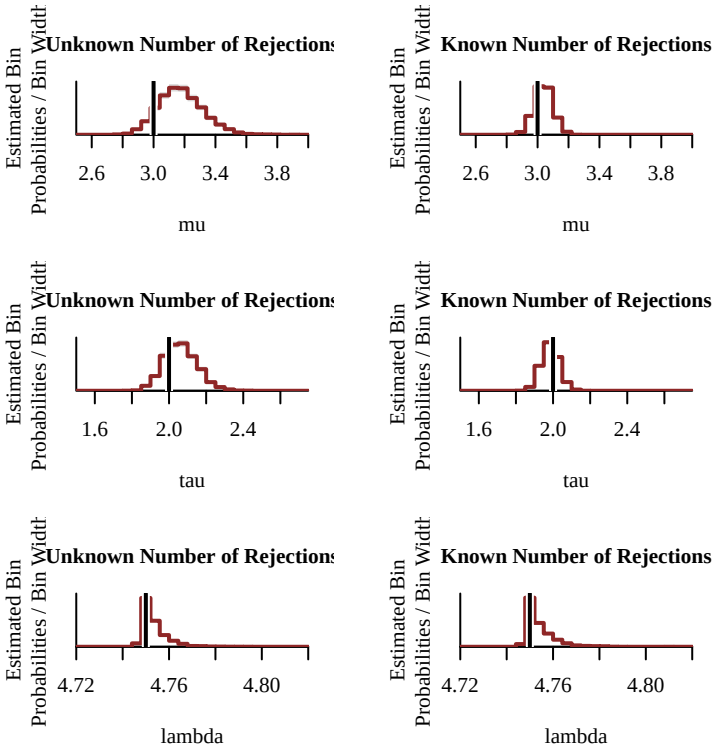


As in the previous fit the posterior inferences concentrate around the true model configuration. The concentration, however, is much stronger for the latent probability distribution parameters μ and τ than in that previous fit.

```
par(mfrow=c(3, 2))


samples2 <- util$extract_expectands(fit2)
samples3 <- util$extract_expectands(fit3)

util$plot_expectand_pushforward(samples2[['mu']], 25, 'mu',
                                flim=c(2.5, 4),
                                main="Unknown Number of Rejections",
                                baseline=mu)
util$plot_expectand_pushforward(samples3[['mu']], 25, 'mu',
                                flim=c(2.5, 4),
                                main="Known Number of Rejections",
                                baseline=mu)

util$plot_expectand_pushforward(samples2[['tau']], 25, 'tau',
                                flim=c(1.5, 2.75),
                                main="Unknown Number of Rejections",
                                baseline=tau)
util$plot_expectand_pushforward(samples3[['tau']], 25, 'tau',
                                flim=c(1.5 ,2.75),
```

```
                                  main="Known Number of Rejections",
                                  baseline=tau)

util$plot_expectand_pushforward(samples2[['lambda']], 25, 'lambda',
                                  flim=c(4.72, 4.82),
                                  main="Unknown Number of Rejections",
                                  baseline=lambda)
util$plot_expectand_pushforward(samples3[['lambda']], 25, 'lambda',
                                  flim=c(4.72, 4.82),
                                  main="Known Number of Rejections",
                                  baseline=lambda)
```

```
Warning in util$plot_expectand_pushforward(samples3[["lambda"]], 25, "lambda",
: 1 posterior samples (0.0%) fell above the histogram binning.
```



We shouldn't be surprised that we achieve our most precise inferences when we take advantage of all of the data available. What may be surprising is just how much more we learn when we incorporate only the total number of rejected events and not their latent values.

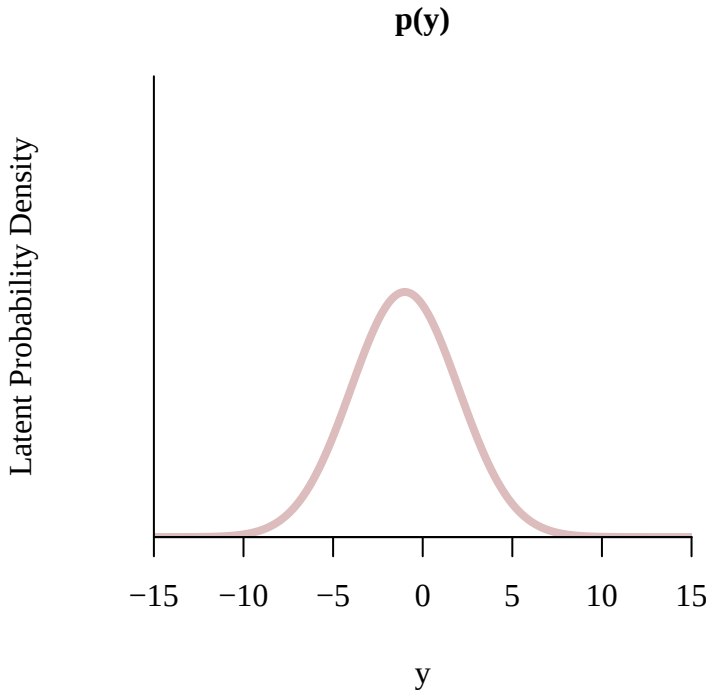
3.3 One-Dimensional Probabilistic Selection

As in the previous deterministic selection example let's consider a one-dimensional latent probability distribution specified by the normal probability density function,

$$p(y \mid \phi = (\mu, \tau)) = \text{normal}(y \mid \mu, \tau),$$

only this time with the location parameter $\mu = -1$ and scale parameter $\tau = 3$.

```
delta <- 0.1
ys <- seq(-20, 20, delta)

mu <- -1
tau <- 3

latent_pds <- dnorm(ys, mu, tau)

par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))
plot(ys, latent_pds, type="l", main ="p(y)", col=c_light, lwd=4,
     xlab="y", xlim=c(-15, 15),
     ylab="Latent Probability Density", yaxt='n', ylim=c(0, 0.25))
```



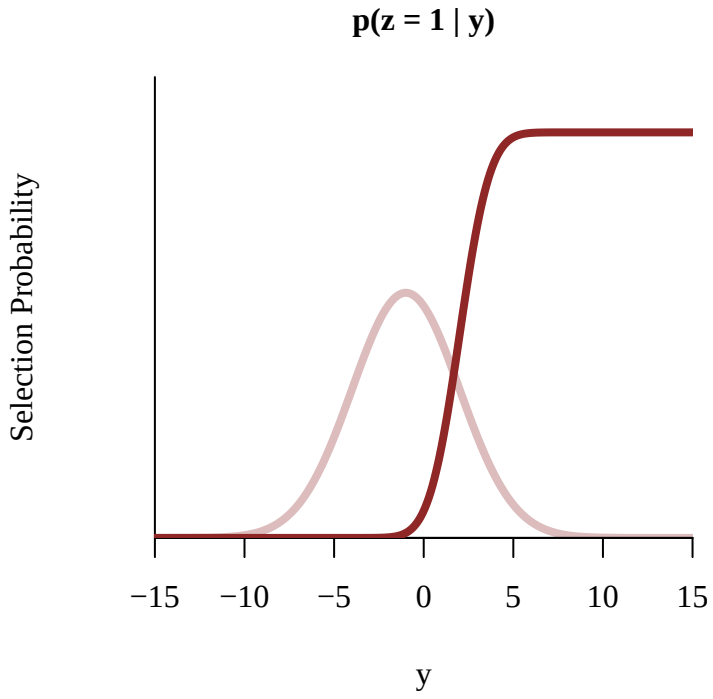
In this case, however, we'll consider a continuous selection function

$$S(y; \psi = (\chi, \gamma)) = \Phi(\gamma\,(y - \chi)),$$

where $\Phi(x) = \Pi_{\text{normal}}(x; 0, 1)$ is the standard normal cumulative distribution function. When $\gamma > 0$ this selection function gradually rises as the input increases, achieving a selection probability of $1/2$ at χ. Conversely when $\gamma < 0$ the selection function falls as the input increases.

Here we'll take $\gamma = 0.75$, so the selection function is monotonically increasing, and $\chi = 2$, so that the selection function is centered in the upper tail of the latent probability density function.

```
chi <- 2
gamma <- 0.75

selection_probs <- pnorm(gamma * (ys - chi))

plot(ys, latent_pds, type="l", main ="p(z = 1 | y)", col=c_light, lwd=4,
     xlab="y", xlim=c(-15, 15),
     ylab="Selection Probability", yaxt='n', ylim=c(0, 0.25))
lines(ys, 0.22 * selection_probs, col=c_dark, lwd=4)
```



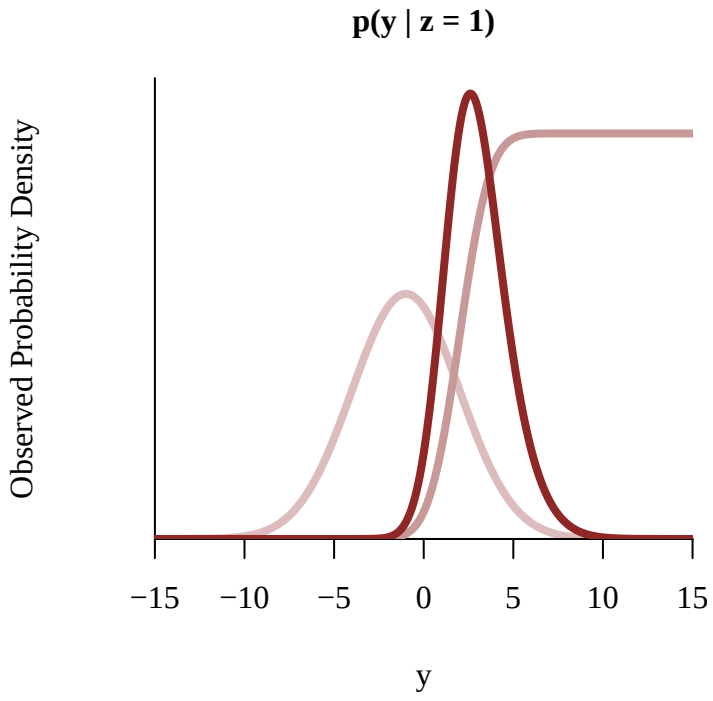
Conveniently the normalization integral resulting from this selection model does admit a closed-

form solution,

$$\begin{aligned} Z(\mu, \tau, \chi, \gamma) &= \int_{-\infty}^{+\infty} \mathrm{d}y \, S(y; \chi, \gamma) \, p(y \mid \mu, \tau) \\ &= \int_{-\infty}^{+\infty} \mathrm{d}y \, \Phi(\gamma \, (y - \chi)) \, \mathrm{normal}(y \mid \mu, \tau) \\ &= \Phi\left(\frac{\gamma \, (\mu - \chi)}{\sqrt{1 + (\gamma \, \tau)^2}} \right). \end{aligned}$$

More details on this integral are available in the Appendix.

```
observed_pds <- (selection_probs * latent_pds)
norm <- pnorm(gamma * (mu - chi) / sqrt(1 + (gamma * tau)**2), 0, 1)
observed_pds <- observed_pds / norm

plot(ys, latent_pds, type="l", main ="p(y | z = 1)", col=c_light, lwd=4,
     xlab="y", xlim=c(-15, 15),
     ylab="Observed Probability Density", yaxt='n', ylim=c(0, 0.25))
lines(ys, 0.22 * selection_probs, col=c_light_highlight, lwd=4)
lines(ys, observed_pds, col=c_dark, lwd=4)
```



The observed probability density function exhibits a noticeable skew, but not one so large that

it couldn't be approximated okay by a symmetric normal probability density function. Indeed we'll see this below.

3.3.1 Simulating Data

Although we can derive an analytic normalization integral, we cannot derive an analytic cumulative distribution function for the observed probability density function. Consequently we cannot simulate data with the inverse cumulative distribution function method.

We can, however, generate data by simulating the selection process directly.

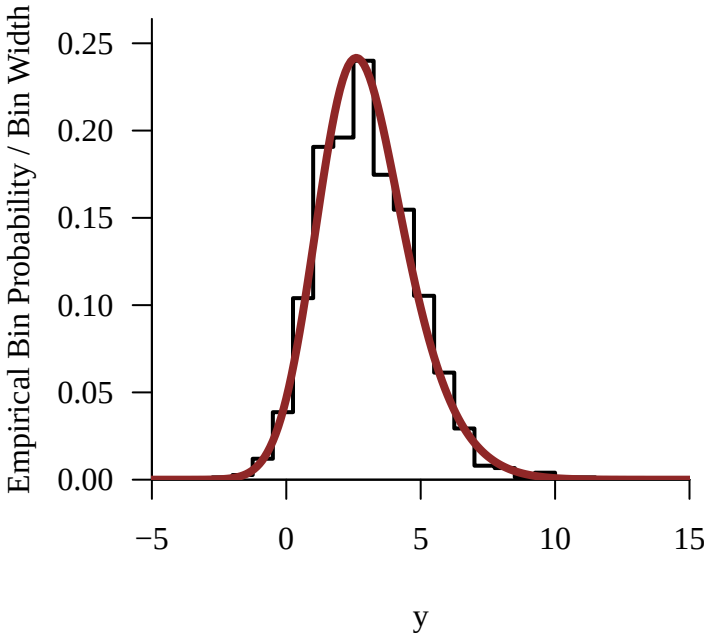
```
N <- 1000

simu <- stan(file="stan_programs/simu_selection_uni.stan",
             iter=1, warmup=0, chains=1, data=list("N" = N),
             seed=4838282, algorithm="Fixed_param")
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0 seconds (Sampling)
Chain 1:                0 seconds (Total)
Chain 1:
```

```
y <- extract(simu)$y[1,]
N_reject <- extract(simu)$N_reject[1]
```

To double check our implementation let's compare a normalized histogram of the simulated data to the true observed probability density function. Fortunately there are no signs on inconsistencies.

```
plot_line_hist(y, -5, 15, 0.75, prob=TRUE, xlab="y")
lines(ys, observed_pds, col=c_dark, lwd=4)
```



3.3.2 Ignoring The Selection Process

Let's begin as we did in the previous exercise by attempting to fit a model that ignores the selection process entirely.

```
data <- list("N" = N, "y" = y)

fit <- stan(file="stan_programs/fit_no_selection_uni.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

There are no indications of inaccurate posterior quantification.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

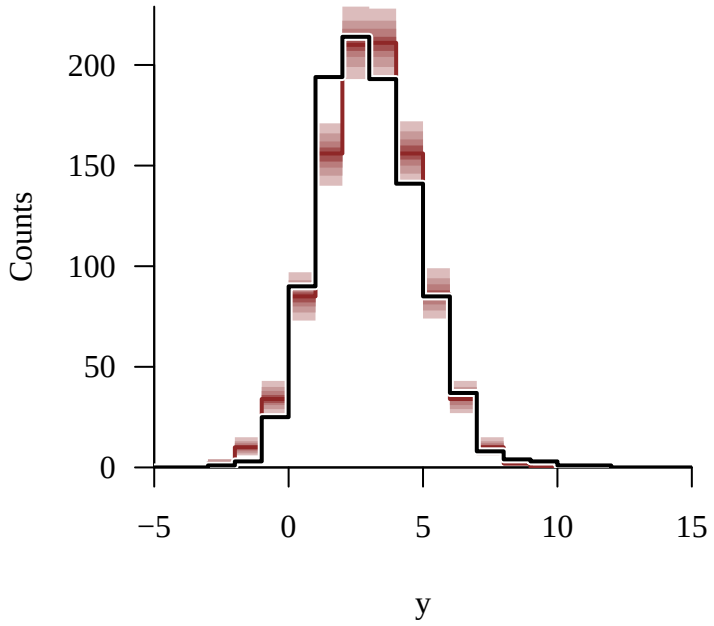
```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples,
                                       c('mu', 'tau'))
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

Perhaps surprisingly the retrodictive performance isn't terrible. If we look carefully we can see that the observed histogram skews slightly to smaller values more than in the posterior predictive histograms.

```
par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))

pred_names <- grep('y_pred', names(samples), value=TRUE)
hist_retro(data$y, samples, pred_names, -5, 15, 1, 'y')
```

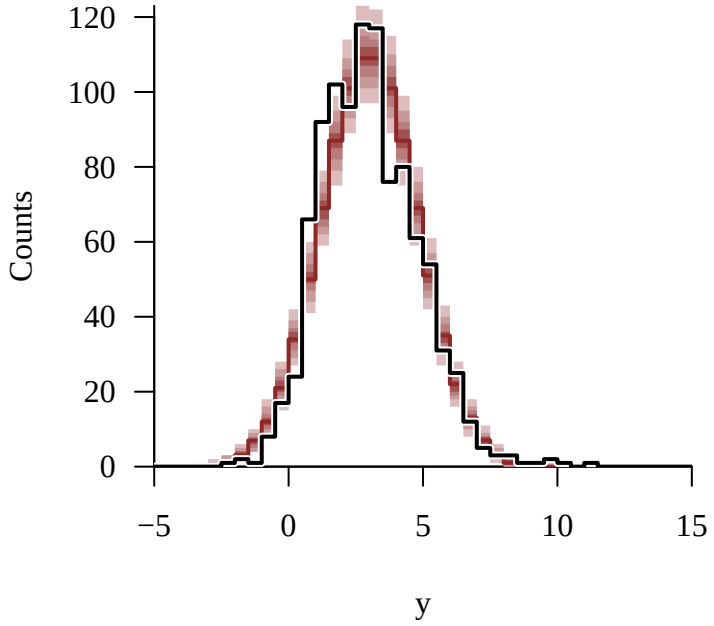
```
Warning in hist_retro(data$y, samples, pred_names, -5, 15, 1, "y"): 18
predictive values (0.0%) fell below the histogram binning.
```



We can see this skew a bit more clearly if we use a histogram with finer bins as our summary statistic. We can't use too fine a histogram summary statistic, however, as the bin occupancies will become too noisy and wash out the signal. Our ability to resolve the retrodictive tension is limited by the available data.

```
par(mfrow=c(1, 1))

hist_retro(data$y, samples, pred_names, -5, 15, 0.5, 'y')
```

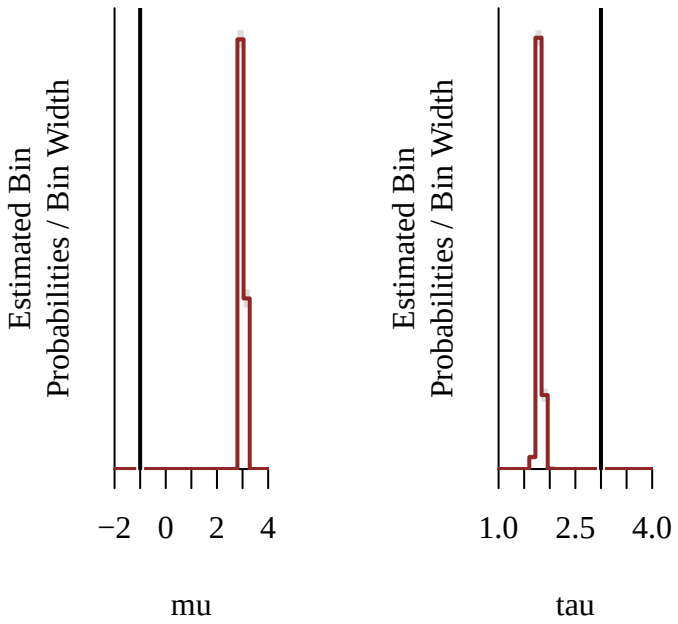
```
Warning in hist_retro(data$y, samples, pred_names, -5, 15, 0.5, "y"): 18
predictive values (0.0%) fell below the histogram binning.
```



If we wanted to investigate this retrodictive tension even further then we could consider summary statistics that target this shape more directly, such as an empirical skewness statistic.

What is not ambiguous is how extremely misaligned the posterior inferences of the latent probability distribution configuration are with the true model configuration.

```
par(mfrow=c(1, 2))

util$plot_expectand_pushforward(samples[['mu']], 25, 'mu',
                                flim=c(-2, 4), baseline=mu)
util$plot_expectand_pushforward(samples[['tau']], 25, 'tau',
                                flim=c(1, 4), baseline=tau)
```



Our model for the latent probability distribution is itself flexible enough to contort into an okay approximation of the observed behavior. This contortion, however, will not generalize to other circumstances where the behavior of the selection changes.

3.3.3 Modeling An Unknown Selection Behavior

In order to derive more robust inferences we'll need to model the selection process, which then requires evaluating the normalization integral. These exercises will then give us an opportunity to compare and contrast exact evaluations to numerical approximations.

Before confronting the entire model let's see what happens when the behavior of the latent probability distribution is known and we just need to infer the behavior of the selection function.

3.3.3.1 Exact

We'll begin with the exact evaluation.

To visualize the inferred selection function behaviors in this one-dimensional problem we can compute the inferred outputs along a grid of latent inputs.

```
delta <- 0.1
ys <- seq(-20, 20, delta)
```

```
data$y_grid <- ys
data$N_grid <- length(ys)
```

```
fit_exact <- stan(file="stan_programs/fit_unknown_selection_uni_exact.stan",
                  data=data, seed=8438338,
                  warmup=1000, iter=2024, refresh=0)
```

The diagnostics are clean.

```
diagnostics <- util$extract_hmc_diagnostics(fit_exact)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

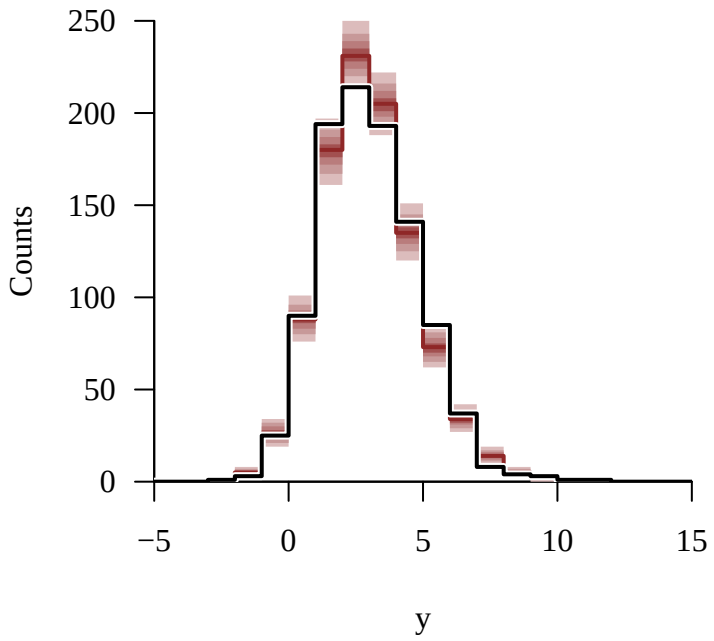
```
samples <- util$extract_expectands(fit_exact)
base_samples <- util$filter_expectands(samples,
                                       c('chi', 'gamma'))
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

The posterior predictions how exhibit a mild skewness similar to what we see in the observed data. With this retrodictive agreement we have nothing to criticize.

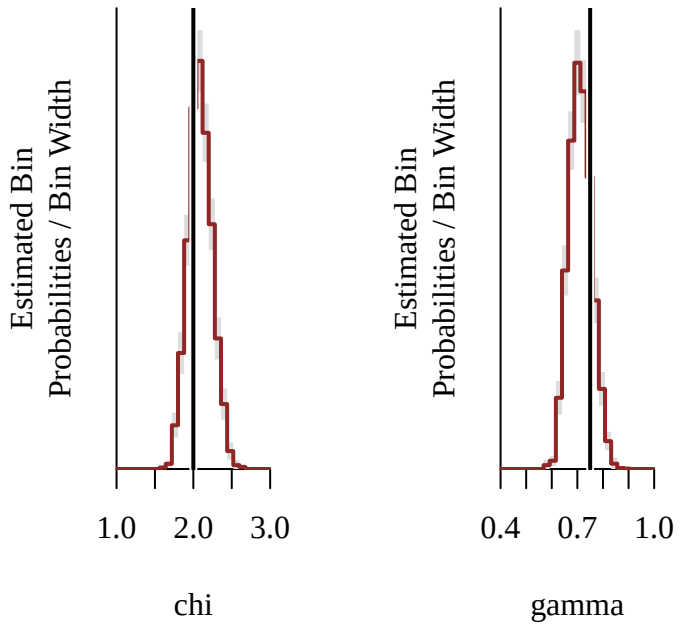
```
par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))

pred_names <- grep('y_pred', names(samples), value=TRUE)
hist_retro(data$y, samples, pred_names, -5, 15, 1, 'y')
```

```
Warning in hist_retro(data$y, samples, pred_names, -5, 15, 1, "y"): 2
predictive values (0.0%) fell above the histogram binning.
```



Even better the inferred configuration of the selection function is consistent with the true configuration.

```
par(mfrow=c(1, 2))

util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
                                flim=c(1, 3), baseline=chi)
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                flim=c(0.4, 1), baseline=gamma)
```

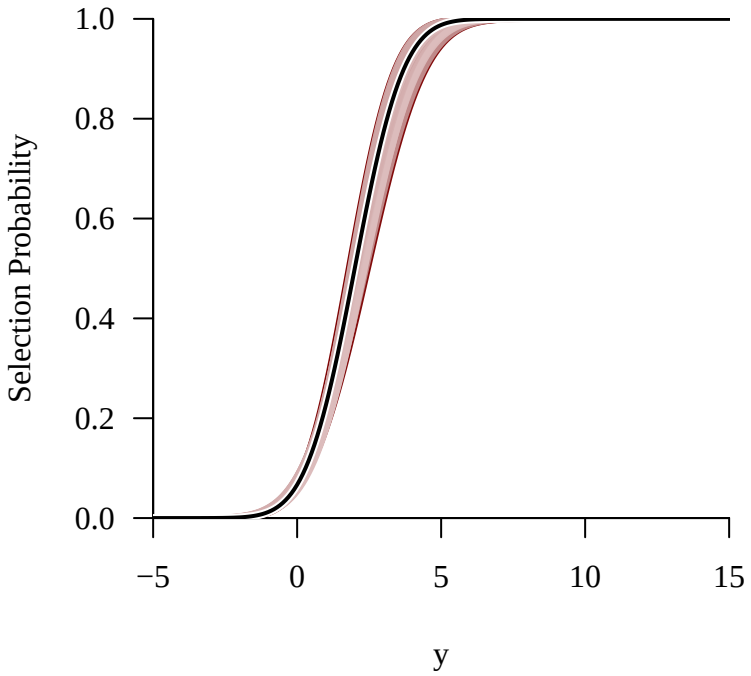


When the influence of the χ and γ parameters on the shape of the selection function isn't immediately obvious we can also visualize the inferred selection function behaviors directly.

```
plot_realizations <- function(xs, fs, N=50,
                              x_name="", display_xlims=NULL,
                              y_name="", display_ylims=NULL,
                              title="") {
  I <- dim(fs)[2]
  J <- min(N, I)

  plot_idx <- sapply(1:J, function(j) (I %/% J) * (j - 1) + 1)

  nom_colors <- c("#DCBCBC", "#C79999", "#B97C7C",
                  "#A25050", "#8F2727", "#7C0000")
  line_colors <- colormap(colormap=nom_colors, nshades=J)

  if (is.null(display_xlims)) {
    xlims <- range(xs)
  } else {
    xlims <- display_xlims
  }

  if (is.null(display_ylims)) {
    ylims <- range(fs)
  } else {
```

```
    ylims <- display_ylims
  }

  plot(1, type="n", main=title,
       xlab=x_name, xlim=xlims,
       ylab=y_name, ylim=ylims)
  for (j in 1:J) {
    r_fs <- fs[, plot_idx[j]]
    lines(xs, r_fs, col=line_colors[j], lwd=3)
  }
}

par(mfrow=c(1, 1), mar = c(5, 4, 2, 1))

fs <- t(sapply(1:data$N_grid,
               function(n) c(samples[[paste0('select_prob_grid[', n, ']')]],
                             recursive=TRUE)))
```

```
plot_realizations(data$y_grid, fs, 50,
                  x_name="y", display_xlims=c(-5, 15),
                  y_name="Selection Probability", display_ylims=c(0, 1))
lines(ys, selection_probs, type="l", col="white", lwd=4)
lines(ys, selection_probs, type="l", col="black", lwd=2)
```



3.3.3.2 Numerical Quadrature

Now we can repeat using a numerical quadrature estimate of the normalization integral. Conveniently `Stan` implements an adaptive quadrature method for one-dimensional integrals.

```
fit <- stan(file="stan_programs/fit_unknown_selection_uni_quad.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Large numerical error can often result in inconsistent model evaluations which in turn can compromise the performance of Markov chain Monte Carlo. Fortunately that doesn't seem to be the case here.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

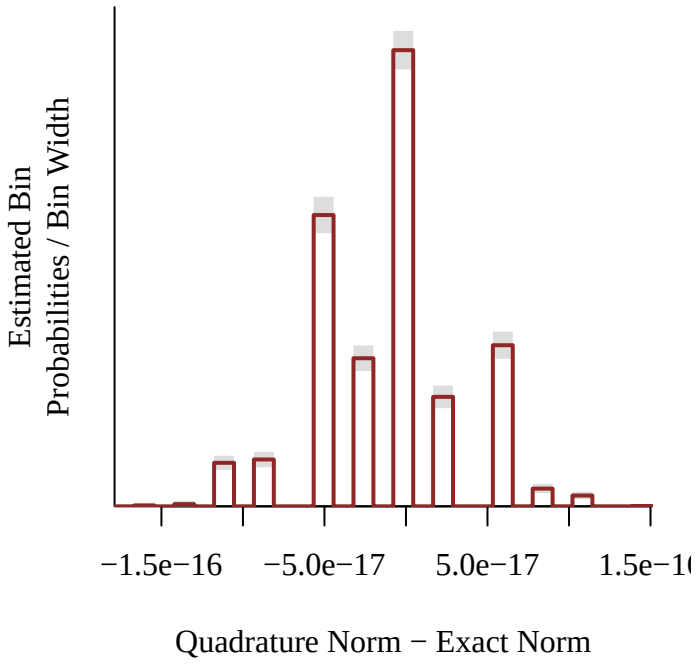
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples,
                                       c('chi', 'gamma'))
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

Indeed the error in the adaptive numerical quadrature is at the same level of double floating precision! We can even see the discretization that arises from subtracting two numbers whose difference is close to the double floating point precision threshold.

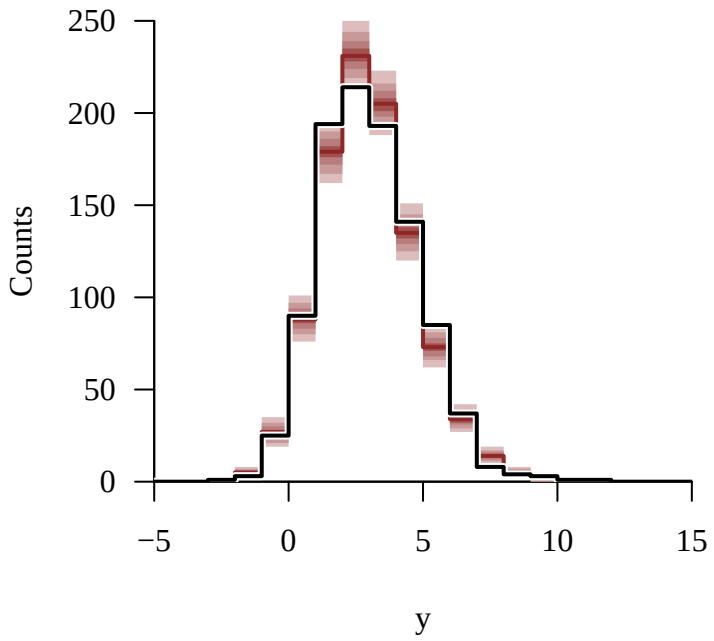
```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_expectand_pushforward(samples[['norm_error']], 25,
                                'Quadrature Norm - Exact Norm')
```



Because the error in evaluating the normalization integral is so small we see the same retrodictive behavior as in the exact fit.

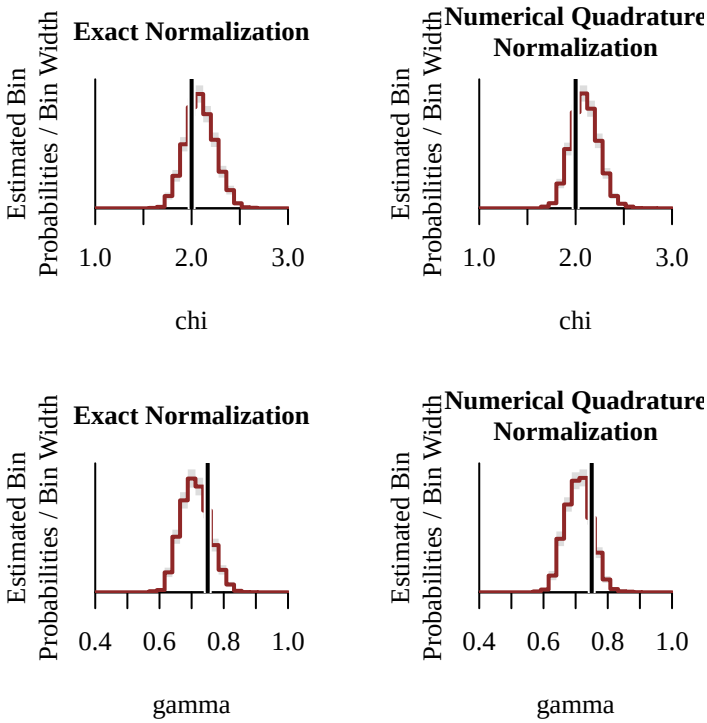
```
par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))

pred_names <- grep('y_pred', names(samples), value=TRUE)
hist_retro(data$y, samples, pred_names, -5, 15, 1, 'y')
```

```
Warning in hist_retro(data$y, samples, pred_names, -5, 15, 1, "y"): 2
predictive values (0.0%) fell above the histogram binning.
```



Similarly the posterior inferences appear to be the same.

```
par(mfrow=c(2, 2))

samples_exact <- util$extract_expectands(fit_exact)

util$plot_expectand_pushforward(samples_exact[['chi']], 25, 'chi',
                                flim=c(1, 3), baseline=chi,
                                main="Exact Normalization")
util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
                                flim=c(1, 3), baseline=chi,
                                main="Numerical Quadrature\nNormalization")

util$plot_expectand_pushforward(samples_exact[['gamma']], 25, 'gamma',
                                flim=c(0.4, 1), baseline=gamma,
                                main="Exact Normalization")
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                flim=c(0.4, 1), baseline=gamma,
                                main="Numerical Quadrature\nNormalization")
```



3.3.3.3 Monte Carlo

How do Monte Carlo estimates of the normalization integral behave? Let's start with a relatively small Monte Carlo ensemble.

Recall that we have to generate a single Monte Carlo ensemble, here in the `transformed data` block, which we then use to construct *all* Monte Carlo estimates of the normalization integral. Conditioned on this particular ensemble the Monte Carlo estimates define a fully deterministic model approximation, with evaluations at the same selection function configuration always returning the same output.

```
data$N_MC <- 100

fit <- stan(file="stan_programs/fit_unknown_selection_uni_mc.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

The diagnostics are clean.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

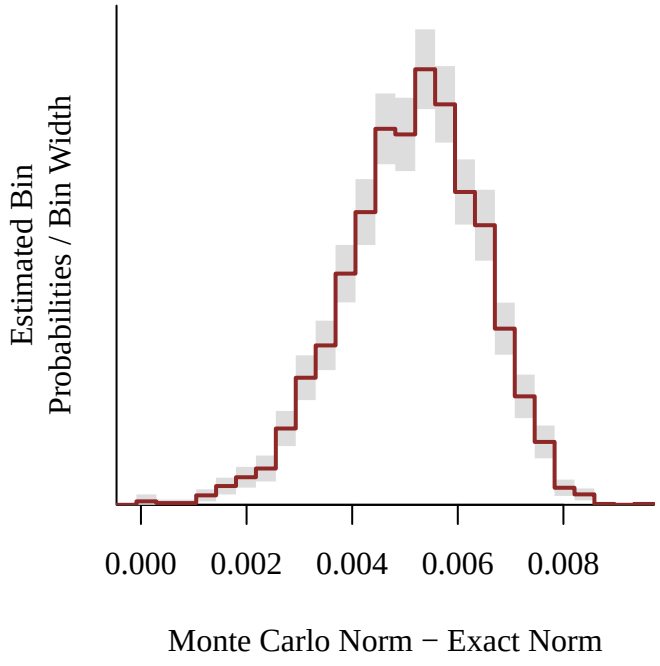
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples,
                                       c('chi', 'gamma'))
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

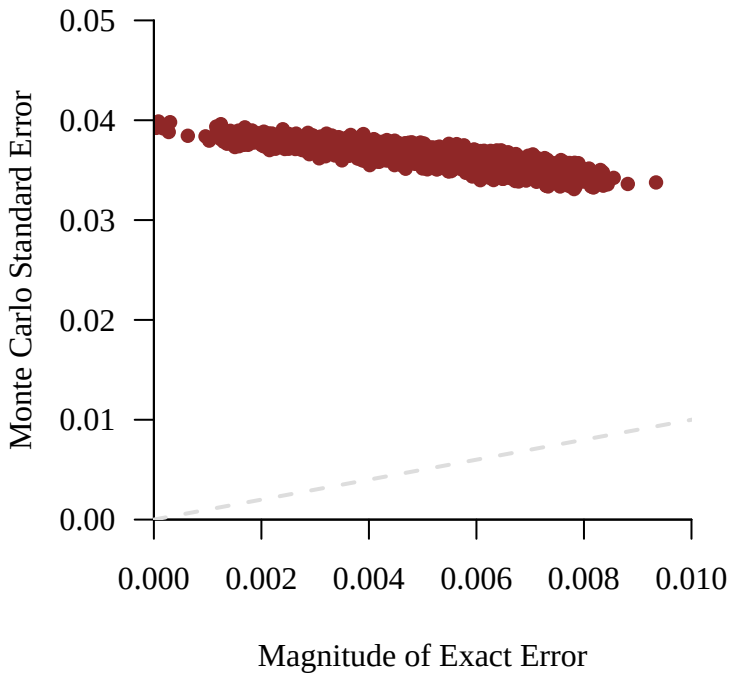
The exact normalization estimate error is much larger than it was for the adaptive numerical quadrature estimator, although not huge in absolute terms.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_expectand_pushforward(samples[['norm_error']], 25,
                                'Monte Carlo Norm - Exact Norm')
```



Perhaps most importantly the estimated Monte Carlo standard error provides a good estimate of the magnitude of the exact error. This makes the Monte Carlo standard error particularly important in applications where we don't know the exact normalization and hence cannot compute the exact error.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

plot(abs(c(samples[['norm_error']], recursive=TRUE)),
         c(samples[['MCSE']], recursive=TRUE),
     col=c_dark, pch=16,
     xlim=c(0, 0.01), xlab="Magnitude of Exact Error",
     ylim=c(0, 0.05), ylab="Monte Carlo Standard Error")
lines(c(0, 1), c(0, 1), col="#DDDDDD", lwd=2, lty=2)
```



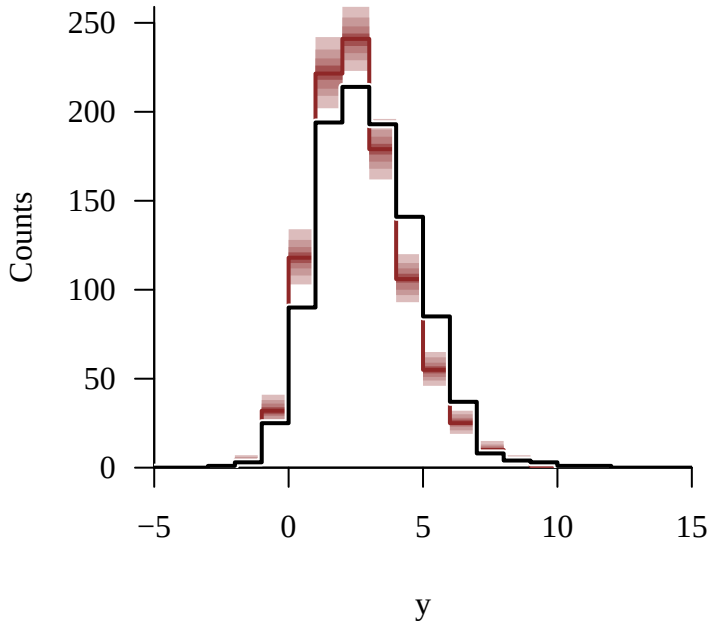
Note that the precise behavior we see here, where the Monte Carlo standard error conservatively upper bounds the exact error, is an accident of the particular Monte Carlo ensemble we drew. Changing the `seed` argument in the `stan` call will result in a different Monte Carlo ensemble whose Monte Carlo estimators could both overshoot or undershoot the exact error. That said the Monte Carlo standard error will usually be within a factor of five of the exact error.

The retrodictive performance has degraded substantially. Because we've changed only how we estimate the normalization integral this must be due to the Monte Carlo estimator error being too large.

```
par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))
```

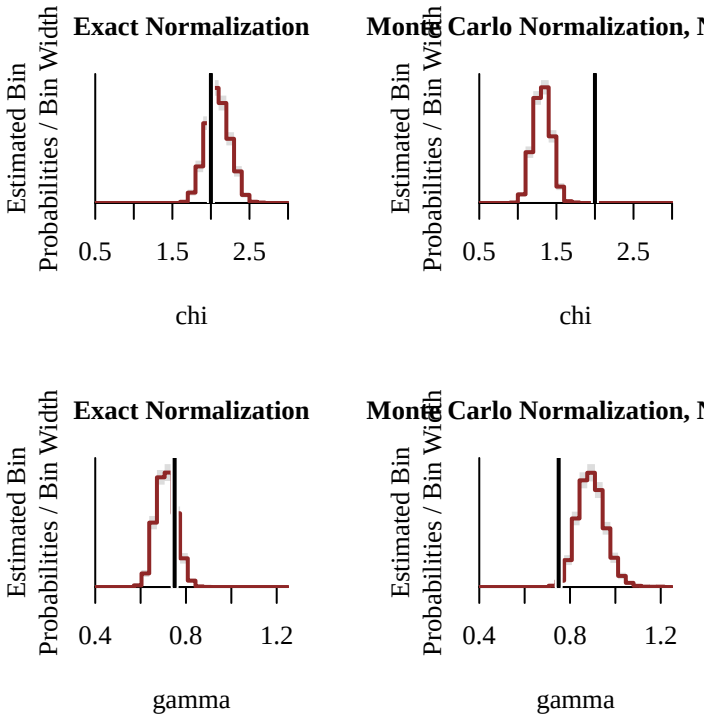
```
pred_names <- grep('y_pred', names(samples), value=TRUE)
hist_retro(data$y, samples, pred_names, -5, 15, 1, 'y')
```

```
Warning in hist_retro(data$y, samples, pred_names, -5, 15, 1, "y"): 1
predictive values (0.0%) fell above the histogram binning.
```



We can confirm this by examining the marginal posterior distributions which have both shifted away from the true model configuration.

```
par(mfrow=c(2, 2))
util$plot_expectand_pushforward(samples_exact[['chi']], 25, 'chi',
                                flim=c(0.5, 3), baseline=chi,
                                main="Exact Normalization")
util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
                                flim=c(0.5, 3), baseline=chi,
                                main="Monte Carlo Normalization, N = 100")

util$plot_expectand_pushforward(samples_exact[['gamma']], 25, 'gamma',
                                flim=c(0.4, 1.25), baseline=gamma,
                                main="Exact Normalization")
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                flim=c(0.4, 1.25), baseline=gamma,
                                main="Monte Carlo Normalization, N = 100")
```



One of the reasons why Monte Carlo estimation is so powerful is that we can somewhat control the error. In particular increasing the Monte Carlo ensemble will, at least in expectation, decrease the Monte Carlo estimator error. Here let's try going from 100 samples to 10000 samples.

```
data$N_MC <- 10000

fit <- stan(file="stan_programs/fit_unknown_selection_uni_mc.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

The Markov chain Monte Carlo diagnostics remain clean.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

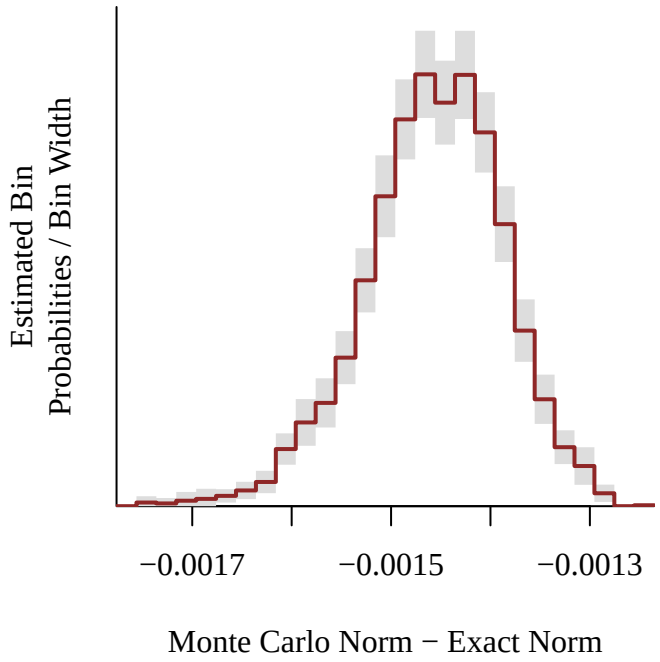
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples,
                                       c('chi', 'gamma'))
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

With the larger ensemble the error in the Monte Carlo estimates of the normalization integral have shrunk by almost a factor of ten.

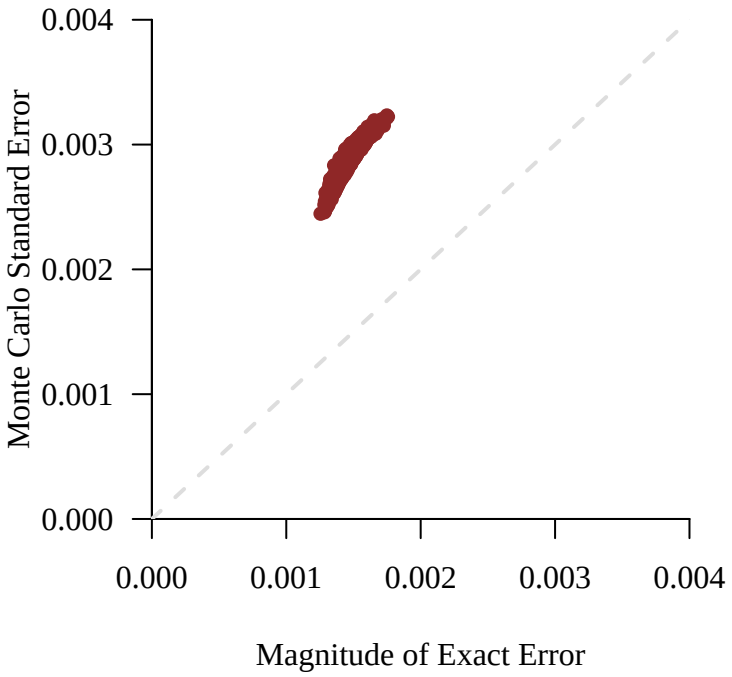
```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_expectand_pushforward(samples[['norm_error']], 25,
                                'Monte Carlo Norm - Exact Norm')
```



Moreover the Monte Carlo standard error continues to well-approximate the exact error.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

plot(abs(c(samples[['norm_error']], recursive=TRUE)),
         c(samples[['MCSE']], recursive=TRUE),
```

```
     col=c_dark, pch=16,
     xlim=c(0, 0.004), xlab="Magnitude of Exact Error",
     ylim=c(0, 0.004), ylab="Monte Carlo Standard Error")
lines(c(0, 1), c(0, 1), col="#DDDDDD", lwd=2, lty=2)
```

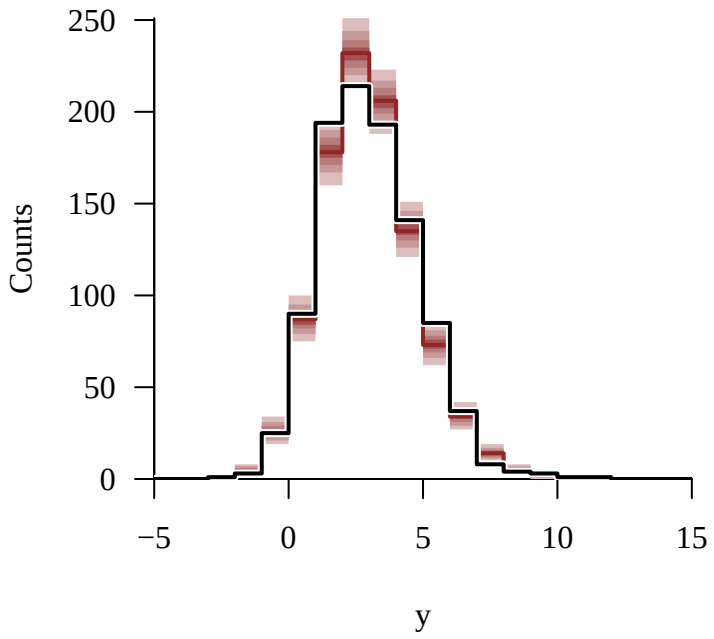


With the error in evaluating the normalization integral so much smaller the resulting model implementation much better approximates the exact model, and the retrodictive performance recovers what we saw with the exact model.

```
par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))

pred_names <- grep('y_pred', names(samples), value=TRUE)
hist_retro(data$y, samples, pred_names, -5, 15, 1, 'y')
```

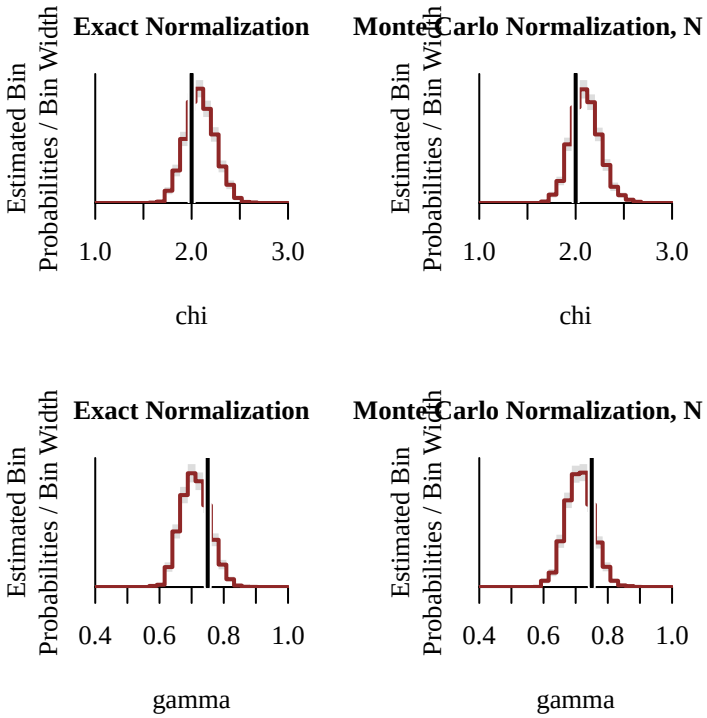
```
Warning in hist_retro(data$y, samples, pred_names, -5, 15, 1, "y"): 1
predictive values (0.0%) fell below the histogram binning.
```

```
Warning in hist_retro(data$y, samples, pred_names, -5, 15, 1, "y"): 2
predictive values (0.0%) fell above the histogram binning.
```



Similarly the posterior inferences derived from this approximate model are now indistinguishable from the posterior inferences derived from the exact model.

```
par(mfrow=c(2, 2))
util$plot_expectand_pushforward(samples_exact[['chi']], 25, 'chi',
                                flim=c(1, 3), baseline=chi,
                                main="Exact Normalization")
util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
                                flim=c(1, 3), baseline=chi,
                                main="Monte Carlo Normalization, N = 10000")

util$plot_expectand_pushforward(samples_exact[['gamma']], 25, 'gamma',
                                flim=c(0.4, 1), baseline=gamma,
                                main="Exact Normalization")
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                flim=c(0.4, 1), baseline=gamma,
                                main="Monte Carlo Normalization, N = 10000")
```



In a real application of Monte Carlo normalization integral estimation we won't know the exact errors nor the exact posterior behaviors. Consequently we won't be able to validate our inferences using these comparisons.

We will, however, be able to consult the Monte Carlo standard errors and compare posterior inferences across different ensemble sizes. In particular we can always increase the ensemble size until the posterior inferences stabilize.

3.3.4 Modeling Unknown Selection and Latent Behavior

In most applications we won't know the behavior of the latent probability distribution. Instead we will have to infer it jointly with the behavior of the selection function.

3.3.4.1 Exact

Following the same progression as in the previous section we'll start with a model implementing the exact normalization integral.

```
fit_exact <- stan(file="stan_programs/fit_unknown_both_uni_exact.stan",
                  data=data, seed=8438338,
                  warmup=1000, iter=2024, refresh=0)
```

Fortunately our computational diagnostics continue to be clean.

```
diagnostics <- util$extract_hmc_diagnostics(fit_exact)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

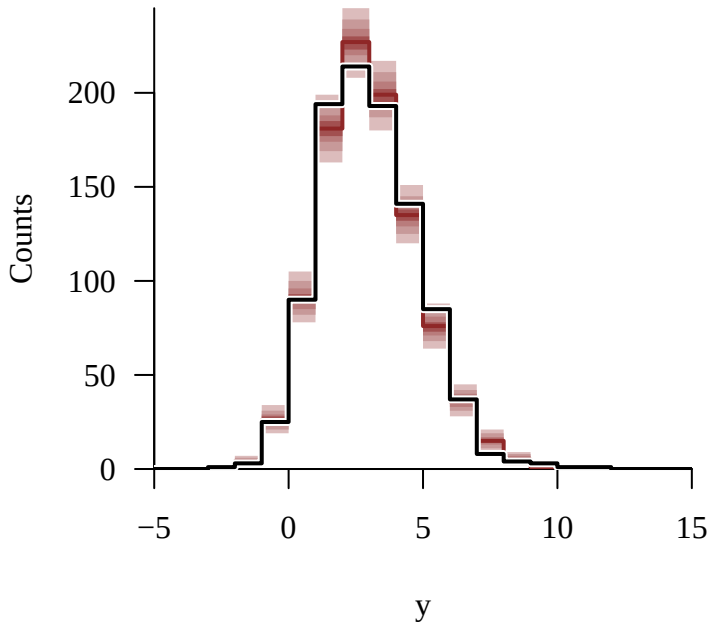
```
samples <- util$extract_expectands(fit_exact)
base_samples <- util$filter_expectands(samples,
                                       c('mu', 'tau', 'chi', 'gamma'))
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

As expected the exact model allows us to recover all of the features of the observed data, including the mild skewness.

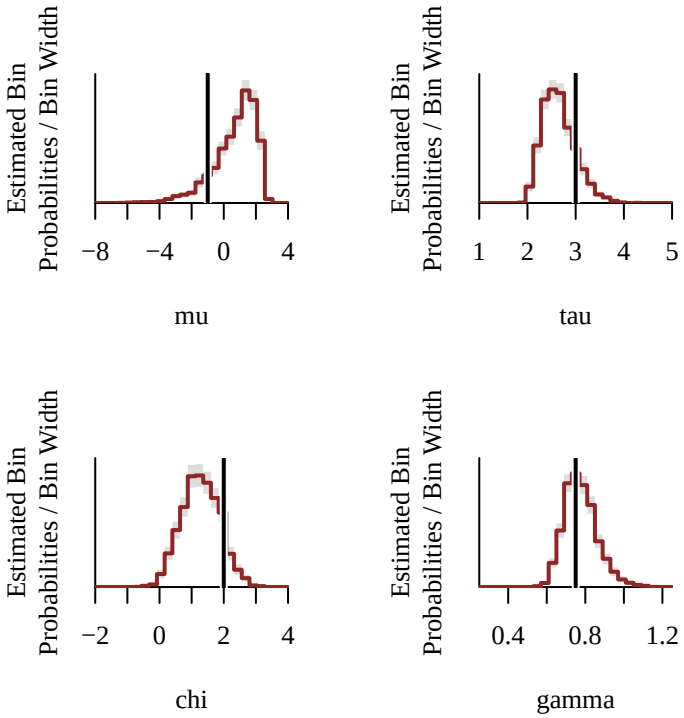
```
par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))

pred_names <- grep('y_pred', names(samples), value=TRUE)
hist_retro(data$y, samples, pred_names, -5, 15, 1, 'y')
```

```
Warning in hist_retro(data$y, samples, pred_names, -5, 15, 1, "y"): 1
predictive values (0.0%) fell above the histogram binning.
```



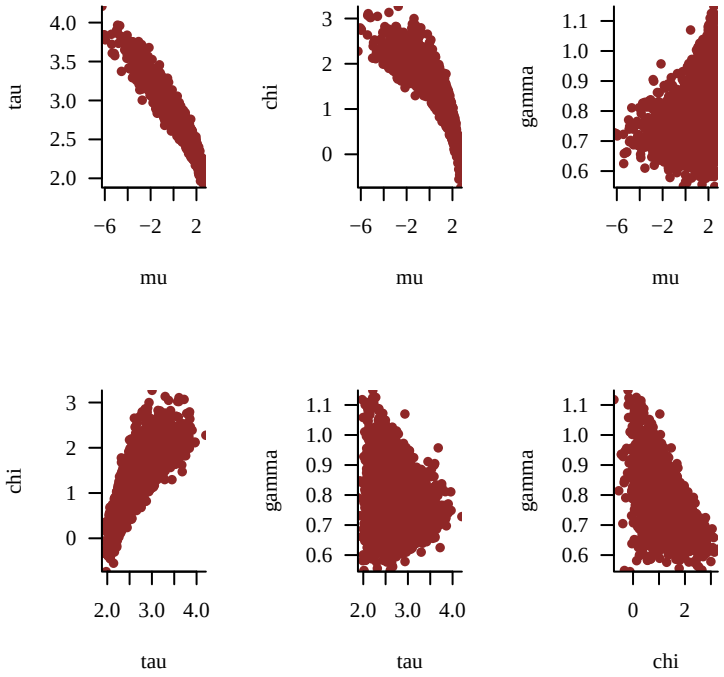
One immediate difference from the previous exercise is that our marginal posterior uncertainties are much wider.

```
par(mfrow=c(2, 2))

util$plot_expectand_pushforward(samples[['mu']], 25, 'mu',
                                flim=c(-8, 4), baseline=mu)
util$plot_expectand_pushforward(samples[['tau']], 25, 'tau',
                                flim=c(1, 5), baseline=tau)
util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
                                flim=c(-2, 4), baseline=chi)
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                flim=c(0.25, 1.25), baseline=gamma)
```



Indeed the pairs plots between the four parameters reveal some nontrivial degeneracies.

```
par(mfrow=c(2, 3))

names <- c('mu', 'tau', 'chi', 'gamma')
for (n in 1:4) {
  for (m in tail(1:4, 4 - n)) {
    plot(c(samples[[names[n]]], recursive=TRUE),
         c(samples[[names[m]]], recursive=TRUE),
         col=c_dark, pch=16,
         xlab=names[n], ylab=names[m])
  }
}
```



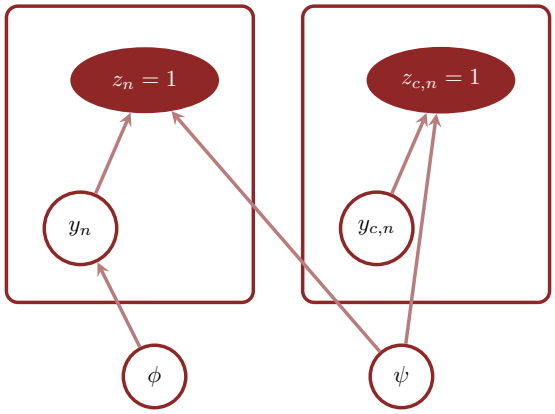
Strong degeneracies like these are not uncommon when fitting full selection models. Many different configurations of the latent probability distribution and selection function will typically be able to accommodate any particular set of observations.

With a powerful tool like Hamiltonian Monte Carlo we may be able to accurately quantify these complex uncertainties, but they may leave much to be desired when it comes to decision making and prediction. In order to improve the situation we need to reduce our uncertainties, which means incorporating additional information.

A more informative prior model often helps, but additional channels of data are often most informative. For example knowledge of the total number of rejected events is often quite informative. So too are *calibration* observations where we can probe the configuration of the selection function using an auxiliary, but fully determined, latent probability distribution (Figure 6). Which strategy is most productive in any given applications will depend on the particular details of that application.

3.3.4.2 Numerical Quadrature

One of the complications of large uncertainties is that our numerical approximations will have to be accurate over a wider range of inputs. How does our adaptive numerical quadrature hold up?



$$\prod_{n=1}^{N} p(y_n, \phi, \psi \mid z_n = 1) \cdot \prod_{n=1}^{N_c} p(y_{c,n}, \psi \mid z_{c,n} = 1)$$

Figure 6: Observations of selected events are often not informative enough to constrain the behavior of the latent probability distribution from the behavior of the selection function. The behavior of the selection function, however, can often be informed directly with data from a calibration process where latent events are generated from a auxiliary but known latent probability distribution. A joint Bayesian model of both sources of data is straightforward to implement in practice.

```
fit <- stan(file="stan_programs/fit_unknown_both_uni_quad.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

The diagnostics are clean. So far so good.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples,
                                       c('mu', 'tau', 'chi', 'gamma'))
util$check_all_expectand_diagnostics(base_samples)
```

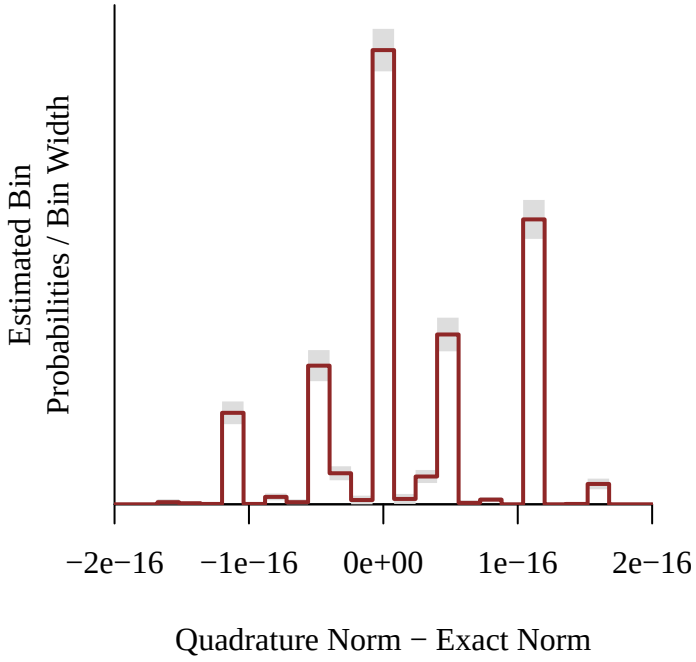
```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

Despite the wider range of inputs that we have to explore the adaptive numerical quadrature algorithm has been able to maintain similarly small errors, save for a few exceptional evaluations.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_expectand_pushforward(samples[['norm_error']], 25,
                                'Quadrature Norm - Exact Norm',
                                flim=c(-2e-16, +2e-16))
```

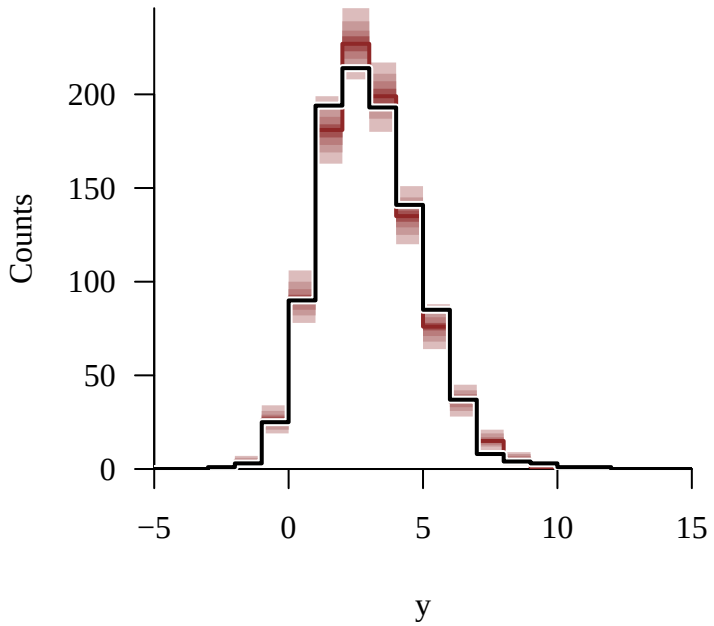
```
Warning in util$plot_expectand_pushforward(samples[["norm_error"]], 25, : 21
posterior samples (0.5%) fell below the histogram binning.
```

```
Warning in util$plot_expectand_pushforward(samples[["norm_error"]], 25, : 569
posterior samples (13.9%) fell above the histogram binning.
```



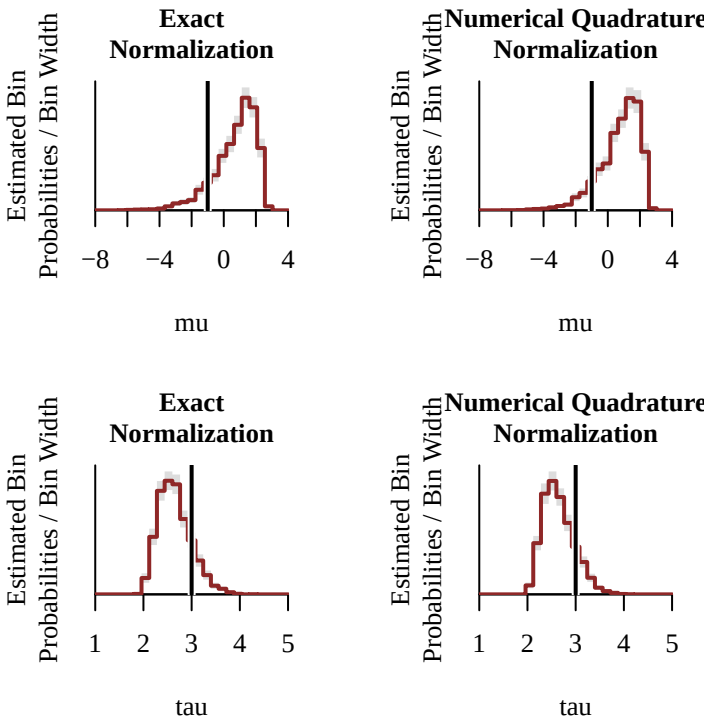
Consequently the retrodictive performance remains the same.

```
par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))

pred_names <- grep('y_pred', names(samples), value=TRUE)
hist_retro(data$y, samples, pred_names, -5, 15, 1, 'y')
```

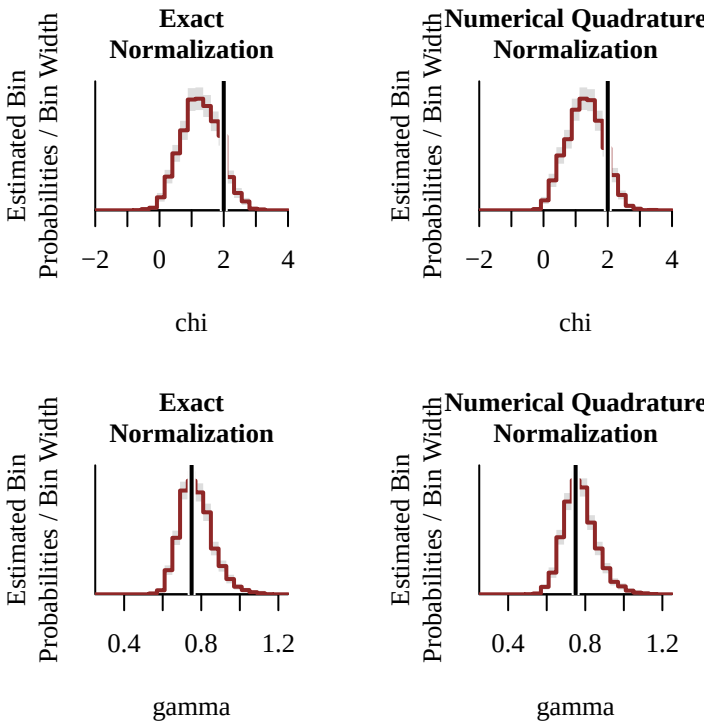


Moreover we cannot distinguish between the posterior inferences derived from the exact model and the approximate model.

```
samples_exact <- util$extract_expectands(fit_exact)

par(mfrow=c(2, 2))


util$plot_expectand_pushforward(samples_exact[['mu']], 25, 'mu',
                                flim=c(-8, 4), baseline=mu,
                                main="Exact\nNormalization")
util$plot_expectand_pushforward(samples[['mu']], 25, 'mu',
                                flim=c(-8, 4), baseline=mu,
                                main="Numerical Quadrature\nNormalization")

util$plot_expectand_pushforward(samples_exact[['tau']], 25, 'tau',
                                flim=c(1, 5), baseline=tau,
                                main="Exact\nNormalization")
util$plot_expectand_pushforward(samples[['tau']], 25, 'tau',
                                flim=c(1, 5), baseline=tau,
                                main="Numerical Quadrature\nNormalization")
```




```
par(mfrow=c(2, 2))

util$plot_expectand_pushforward(samples_exact[['chi']], 25, 'chi',
                                flim=c(-2, 4), baseline=chi,
                                main="Exact\nNormalization")
util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
                                flim=c(-2, 4), baseline=chi,
                                main="Numerical Quadrature\nNormalization")

util$plot_expectand_pushforward(samples_exact[['gamma']], 25, 'gamma',
                                flim=c(0.25, 1.25), baseline=gamma,
                                main="Exact\nNormalization")
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                flim=c(0.25, 1.25), baseline=gamma,
                                main="Numerical Quadrature\nNormalization")
```



3.3.4.3 Importance Sampling

Because the latent probability distribution is no longer fixed we cannot utilize Monte Carlo estimation using a fixed ensemble. We can, however, utilize importance sampling estimation with a fixed ensemble drawn from a fixed reference probability distribution.

Ideally the reference probability distribution would span all of the latent probability distributions in neighborhoods of high posterior probability. In practice, however, we will not know the shape of the posterior distribution. As a proxy we can tune the reference probability distribution to the shape of the prior model.

For example our prior model roughly contains the location of the latent probability density function to the interval

$$-5 \lessapprox \mu \lessapprox 5$$

and the scale to the interval

$$0 < \tau \lessapprox 5.$$

The largest Y interval allowed by these constraints is

$$\begin{aligned} -5 - 2.32 \cdot 5 &\lessapprox y \lessapprox +5 + 2.32 \cdot 5 \\ -16.6 &\lessapprox y \lessapprox +16.6. \end{aligned}$$

This interval is in turn spanned by a normal probability density function with location parameter 0 and scale parameter of about 7.2.

Note that much of the Stan program is dedicated to computing importance sampling diagnostics.

As with our previous Monte Carlo experiment we'll start with a relatively small ensemble.

```
data$N_IS <- 100

fit <- stan(file="stan_programs/fit_unknown_both_uni_is.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

```
Info: Found int division at 'string', line 90, column 22 to column 26:
  N_IS / 2
Values will be rounded towards zero. If rounding is not desired you can write
the division as
  N_IS / 2.0
```

For the first time in this chapter we have encountered some diagnostic warnings. In addition to large autocorrelations across many parameters it appears that γ might be exhibiting heavy tails.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples,
                                       c('mu', 'tau', 'chi', 'gamma'))
util$check_all_expectand_diagnostics(base_samples)
```

```
tau:
  Chain 3: hat{ESS} (94.353) is smaller than desired (100)!

chi:
  Chain 4: hat{ESS} (79.693) is smaller than desired (100)!
```

```
gamma:
  Chain 2: Left tail hat{xi} (0.309) exceeds 0.25!
  Chain 3: Left tail hat{xi} (0.437) exceeds 0.25!
  Chain 4: Left tail hat{xi} (0.352) exceeds 0.25!
  Chain 3: hat{ESS} (73.131) is smaller than desired (100)!
  Chain 4: hat{ESS} (69.685) is smaller than desired (100)!


Large tail hat{xi}s suggest that the expectand might not be
sufficiently integrable.

Small empirical effective sample sizes indicate strong empirical
autocorrelations in the realized Markov chains. If the empirical
effective sample size is too small then Markov chain Monte Carlo
estimation may be unreliable even when a central limit theorem holds.
```

The sudden change in posterior quantification suggests that the importance sampling error may be large enough to be causing problems. To investigate further let's consult the importance sampling diagnostics.

```
plot_histogram_pushforward <- function(samples, var_names,
                                       bin_min=NULL, bin_max=NULL, delta=NULL,
                                       xlab="", ylim=NA, title="") {
  # Check that pred_names are in samples
  all_names <- names(samples)

  bad_names <- setdiff(var_names, all_names)
  if (length(bad_names) > 0) {
    warning(sprintf('The expectand names %s are not in the `samples` object and will be ignor
                    paste(bad_names, collapse=", ")))
  }

  good_names <- intersect(var_names, all_names)
  if (length(good_names) == 0) {
    stop('There are no valid expectand names.')
  }
  var_names <- good_names

  # Remove any NA values
  var <- sapply(var_names,
                function(name) c(t(samples[[name]]), recursive=TRUE))
  var_collapse <- c(var)
```

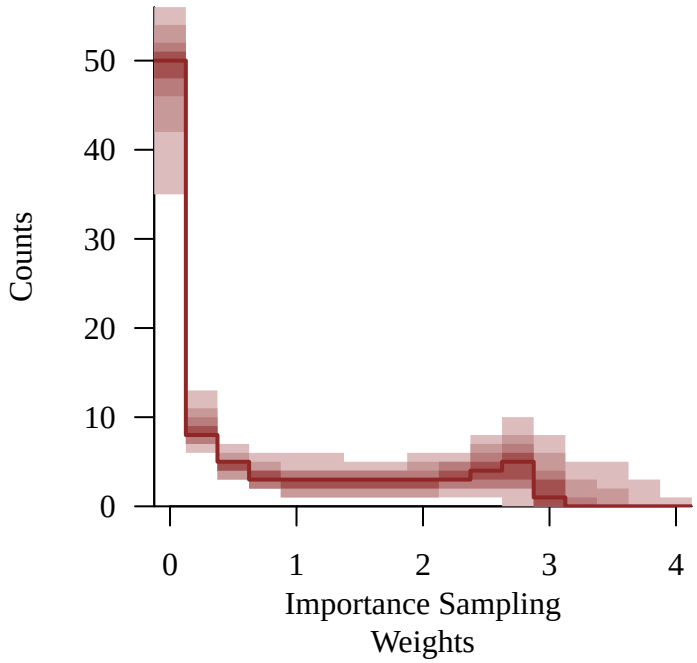
```
var_collapse <- var_collapse[!is.na(var_collapse)]

# Construct binning configuration
if (is.null(bin_min))
  bin_min <- min(var_collapse)
if (is.null(bin_max))
  bin_max <- max(var_collapse)
if (is.null(delta))
  delta <- (bin_max - bin_min) / 25

# Construct bins
breaks <- seq(bin_min, bin_max, delta)
B <- length(breaks) - 1
idx <- rep(1:B, each=2)
xs <- sapply(1:length(idx),
             function(b) if(b %% 2 == 0) breaks[idx[b] + 1]
             else            breaks[idx[b]] )

# Check bin containment of values
N <- length(var_collapse)

N_low <- sum(var_collapse < bin_min)
if (N_low > 0)
  warning(sprintf('%s values (%.1f%%) fell below the histogram binning.',
                  N_low, 100 * N_low / N))

N_high <- sum(bin_max < var_collapse)
if (N_high > 0)
  warning(sprintf('%s values (%.1f%%) fell above the histogram binning.',
                  N_high, 100 * N_high / N))

# Construct ribbons for  bin contents
N <- dim(var)[1]
var_counts <- sapply(1:N,
                     function(n) hist(var[n, bin_min <= var[n,] &
                                              var[n,] <= bin_max],
                                      breaks=breaks,
                                      plot=FALSE)$counts)
probs = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)
cred <- sapply(1:B,
               function(b) quantile(var_counts[b,], probs=probs))
var_cred <- do.call(cbind, lapply(idx, function(n) cred[1:9, n]))
```

```
  if (any(is.na(ylim))) {
    ylim <- c(0, max(cred[9,]))
  }

  # Plot
  plot(1, type="n", main=title,
       xlim=c(bin_min, bin_max), xlab=xlab,
       ylim=ylim, ylab="Counts")
  polygon(c(xs, rev(xs)), c(var_cred[1,], rev(var_cred[9,])),
          col = c_light, border = NA)
  polygon(c(xs, rev(xs)), c(var_cred[2,], rev(var_cred[8,])),
          col = c_light_highlight, border = NA)
  polygon(c(xs, rev(xs)), c(var_cred[3,], rev(var_cred[7,])),
          col = c_mid, border = NA)
  polygon(c(xs, rev(xs)), c(var_cred[4,], rev(var_cred[6,])),
          col = c_mid_highlight, border = NA)
  lines(xs, var_cred[5,], col=c_dark, lwd=2)
}
```

The distribution importance weights appears to be particularly well-behaved; the weights not only show no signs of heavy tails but also appear to be bounded below by 3 or so. This suggests that our reference probability distribution has been well-chosen, boding well for importance sampling estimation.

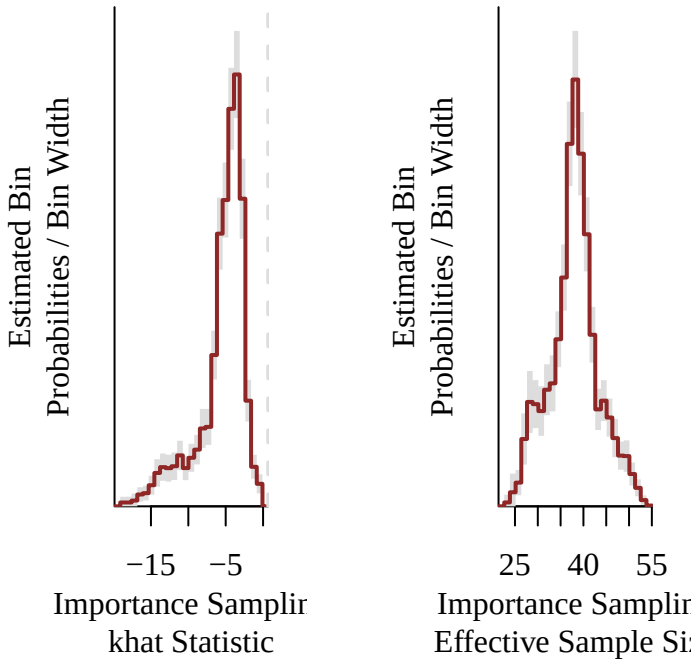
```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

weight_names <- grep('weights', names(samples), value=TRUE)
plot_histogram_pushforward(samples, weight_names, -0.125, 4.125, 0.25,
                           "Importance Sampling\nWeights")
```

```
Warning in plot_histogram_pushforward(samples, weight_names, -0.125, 4.125, :
1015 values (0.2%) fell above the histogram binning.
```



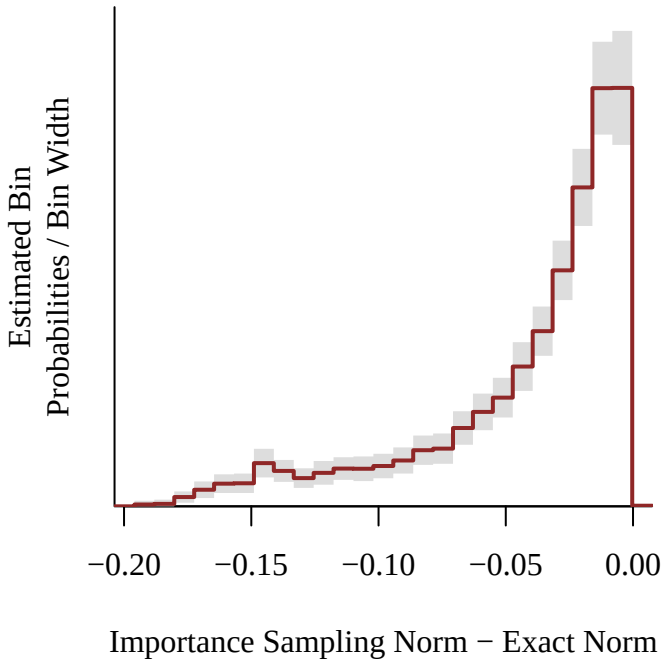
The negative values of the $\hat{k}$ statistic support the good behavior of the importance weights. That said the importance sampling effective sample size is quite low, suggesting imprecise normalization integral estimates.

```
par(mfrow=c(1, 2), mar = c(5, 5, 2, 1))

util$plot_expectand_pushforward(samples[['khat']], 25,
                                'Importance Sampling\nkhat Statistic')
abline(v=0.7, col="#DDDDDD", lwd=2, lty=2)

util$plot_expectand_pushforward(samples[['ISESS']], 25,
                                'Importance Sampling\nEffective Sample Size')
```



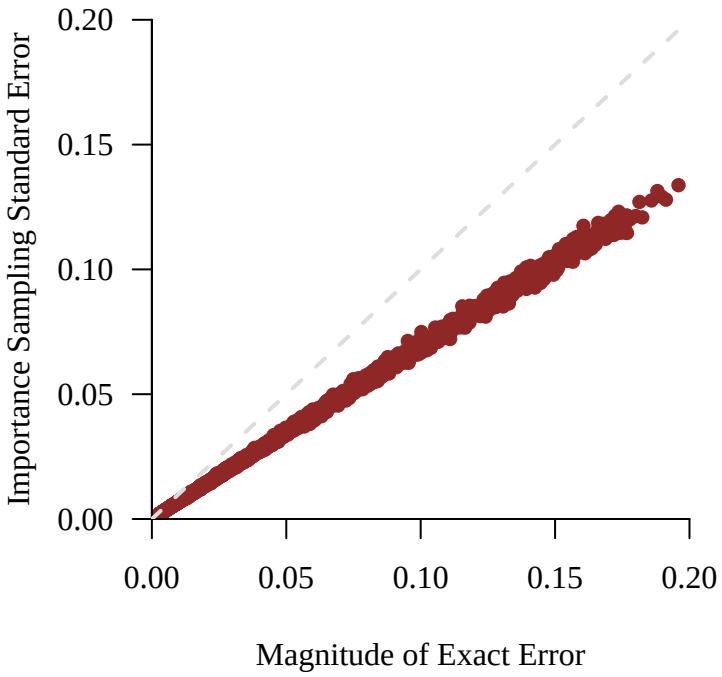
Indeed the exact errors are quite large.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_expectand_pushforward(samples[['norm_error']], 25,
                                'Importance Sampling Norm - Exact Norm')
```



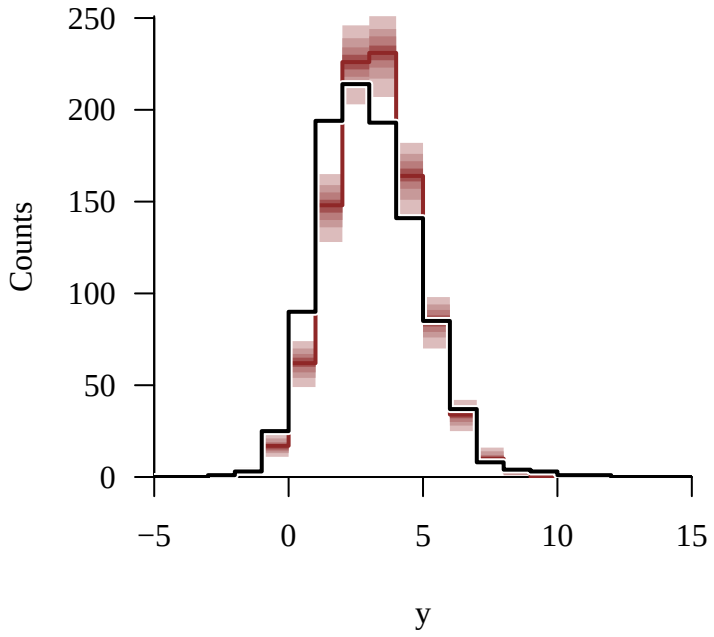
That said the importance sampling standard errors provides a reasonably accurate estimate of the magnitude of the exact error. The fact that the importance sampling standard errors all underestimate the exact error is an accident of the particular importance sampling ensemble that we generated here, similarly to what we saw with the Monte Carlo method.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

plot(abs(c(samples[['norm_error']], recursive=TRUE)),
         c(samples[['ISSE']], recursive=TRUE),
     col=c_dark, pch=16,
     xlim=c(0, 0.2), xlab="Magnitude of Exact Error",
     ylim=c(0, 0.2), ylab="Importance Sampling Standard Error")
lines(c(0, 1), c(0, 1), col="#DDDDDD", lwd=2, lty=2)
```



These large normalization integral errors compromise the retrodictive performance.

```
par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))

pred_names <- grep('y_pred', names(samples), value=TRUE)
hist_retro(data$y, samples, pred_names, -5, 15, 1, 'y')
```

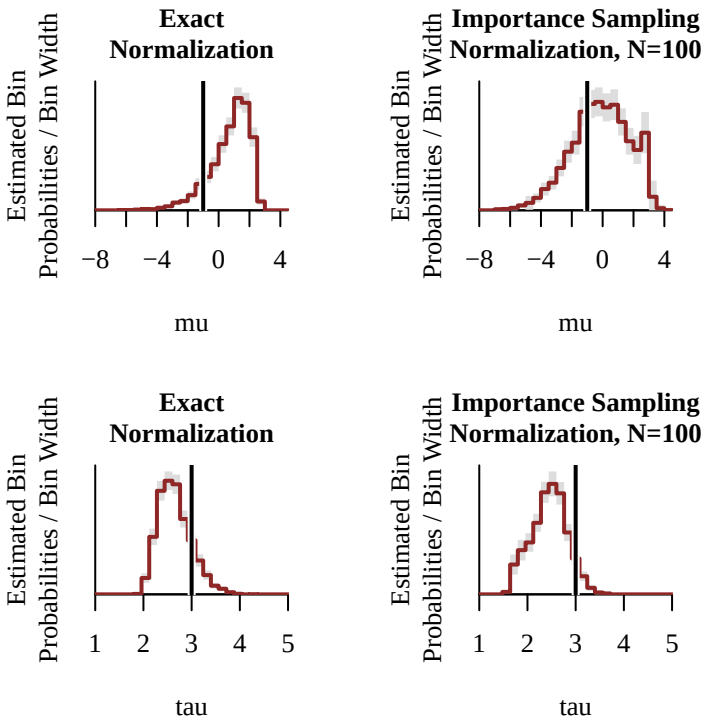


Similarly the posterior inferences no longer conform to the posterior inferences derived from the exact model.

```
samples_exact <- util$extract_expectands(fit_exact)


par(mfrow=c(2, 2))

util$plot_expectand_pushforward(samples_exact[['mu']], 25, 'mu',
                                flim=c(-8, 4.5), baseline=mu,
                                main="Exact\nNormalization")
util$plot_expectand_pushforward(samples[['mu']], 25, 'mu',
                                flim=c(-8, 4.5), baseline=mu,
                                main="Importance Sampling\nNormalization, N=100")
```

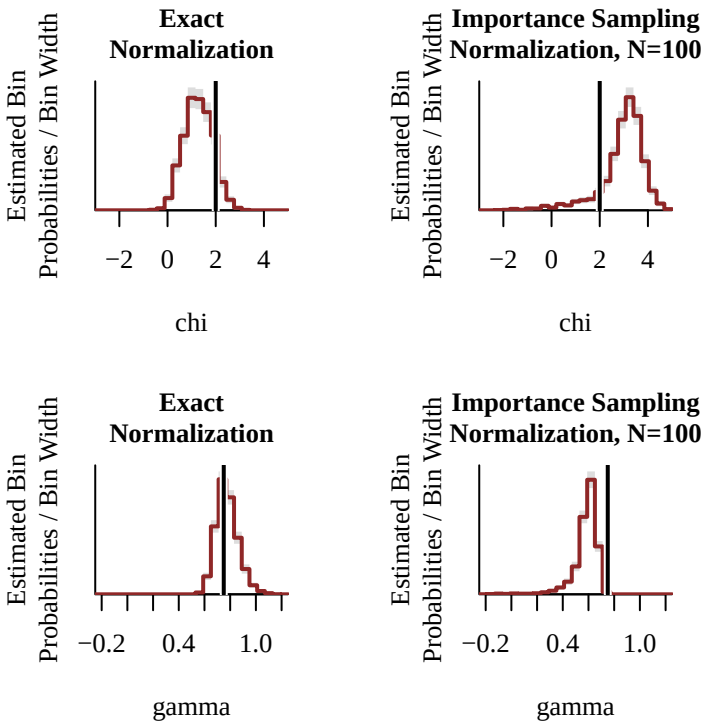
```
Warning in util$plot_expectand_pushforward(samples[["mu"]], 25, "mu", flim =
c(-8, : 5 posterior samples (0.1%) fell above the histogram binning.
```

```
util$plot_expectand_pushforward(samples_exact[['tau']], 25, 'tau',
                                flim=c(1, 5), baseline=tau,
                                main="Exact\nNormalization")
util$plot_expectand_pushforward(samples[['tau']], 25, 'tau',
                                flim=c(1, 5), baseline=tau,
                                main="Importance Sampling\nNormalization, N=100")
```




```
par(mfrow=c(2, 2))

util$plot_expectand_pushforward(samples_exact[['chi']], 25, 'chi',
                                flim=c(-3, 5), baseline=chi,
                                main="Exact\nNormalization")
util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
                                flim=c(-3, 5), baseline=chi,
                                main="Importance Sampling\nNormalization, N=100")

util$plot_expectand_pushforward(samples_exact[['gamma']], 25, 'gamma',
                                flim=c(-0.25, 1.25), baseline=gamma,
                                main="Exact\nNormalization")
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                flim=c(-0.25, 1.25), baseline=gamma,
                                main="Importance Sampling\nNormalization, N=100")
```

```
Warning in util$plot_expectand_pushforward(samples[["gamma"]], 25, "gamma", : 5
posterior samples (0.1%) fell below the histogram binning.
```



If the importance weights are as well-behaved as they appear then we should be able to reduce the error of the importance sampling normalization integral evaluations by increasing the ensemble size.

```
data$N_IS <- 10000

fit <- stan(file="stan_programs/fit_unknown_both_uni_is.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

```
If rounding is intended please use the integer division operator %/%.
Info: Found int division at 'string', line 90, column 22 to column 26:
  N_IS / 2
Values will be rounded towards zero. If rounding is not desired you can write
the division as
  N_IS / 2.0
```

The diagnostics warnings have gone away.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples,
                                       c('chi', 'gamma', 'mu', 'tau'))
util$check_all_expectand_diagnostics(base_samples)
```

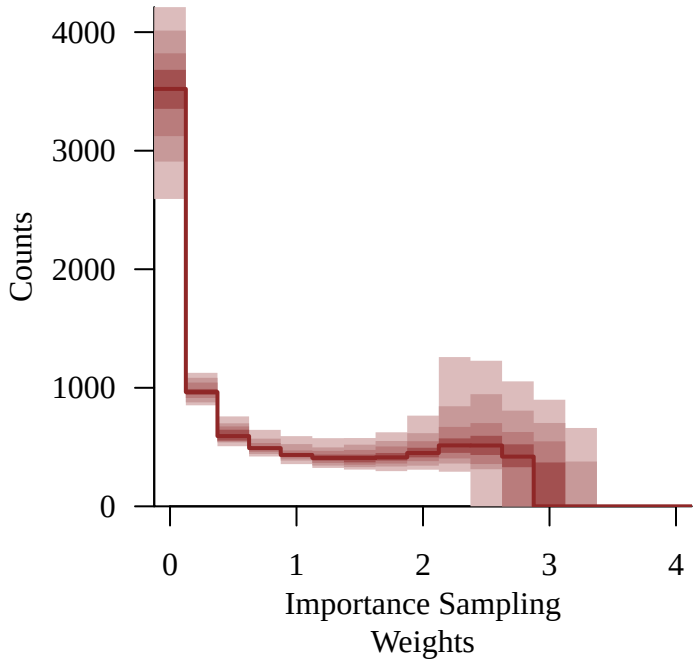
```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

The distribution of the importance weights across evaluations is determined by the reference probability distribution and posterior distribution over the parameter inputs. Because the reference probability distribution is the same the only way that the importance weight distributions could change is if the larger ensemble allows us to explore the exact posterior distribution more accurately.

Here there doesn't seem to be any obvious difference in the behavior of the importance weights using the larger ensemble relative to the smaller ensemble.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

weight_names <- grep('weights', names(samples), value=TRUE)
plot_histogram_pushforward(samples, weight_names, -0.125, 4.125, 0.25,
                           "Importance Sampling\nWeights")
```

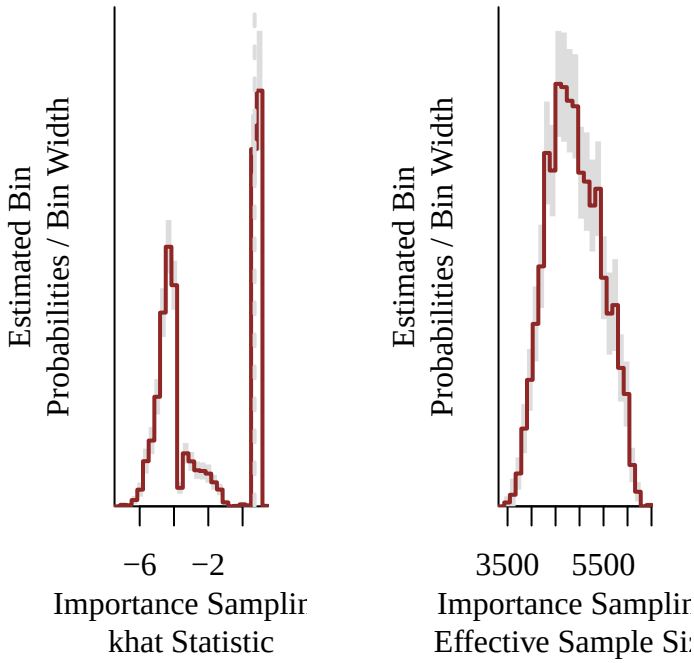
```
Warning in plot_histogram_pushforward(samples, weight_names, -0.125, 4.125, :
25 values (0.0%) fell above the histogram binning.
```



The $\hat{k}$-statistics, however, now concentrate at larger values and even extend to the 0.7 threshold that is suggested as a diagnostic of unreliable importance sampling estimates. On the other hand the $\hat{k}$-statistic does not work particularly well when the importance weights are bounded, as they are here, so we shouldn't be too worried.

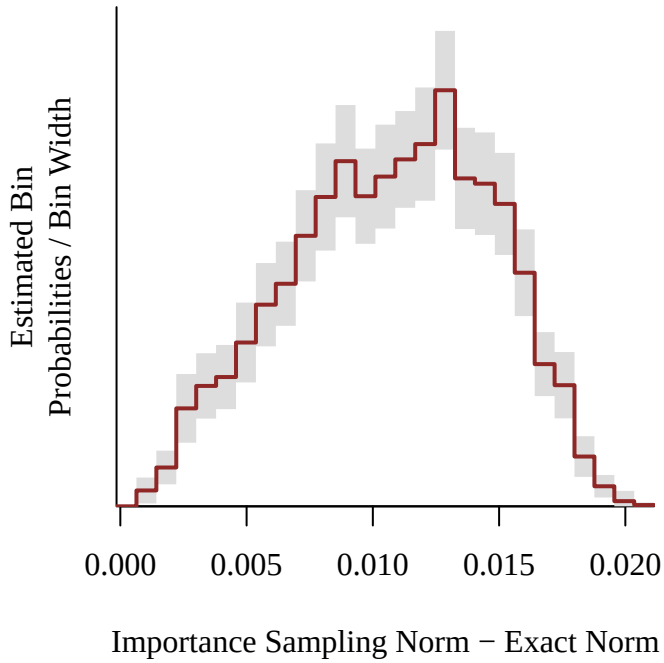
Indeed the importance sampling effective sample sizes are *much* larger, suggesting more accurate normalization integral estimates.

```
par(mfrow=c(1, 2), mar = c(5, 5, 2, 1))

util$plot_expectand_pushforward(samples[['khat']], 25,
                                'Importance Sampling\nkhat Statistic')
abline(v=0.7, col="#DDDDDD", lwd=2, lty=2)

util$plot_expectand_pushforward(samples[['ISESS']], 25,
                                'Importance Sampling\nEffective Sample Size')
```



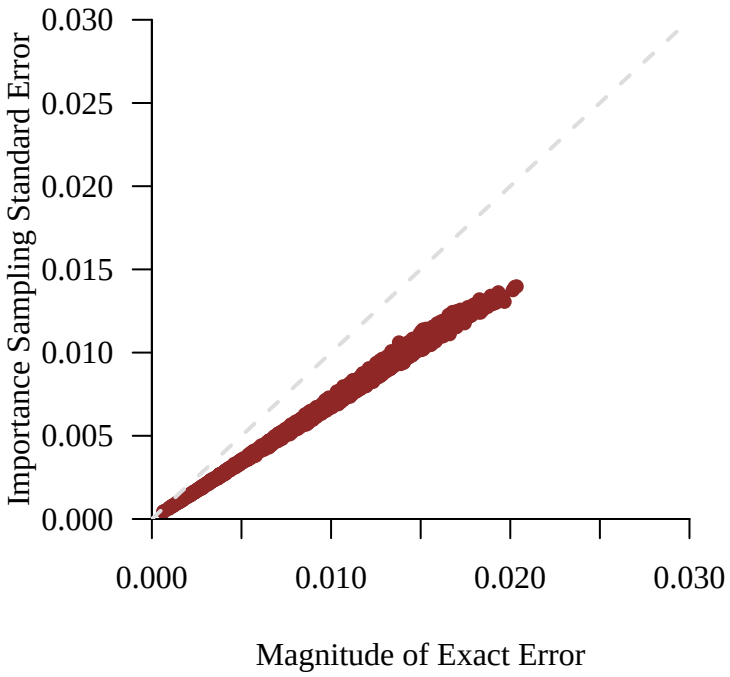
Indeed the exact errors are over an order of magnitude smaller than they were with the smaller ensemble.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_expectand_pushforward(samples[['norm_error']], 25,
                                'Importance Sampling Norm - Exact Norm')
```



The accuracy of the importance sampling standard errors persists.

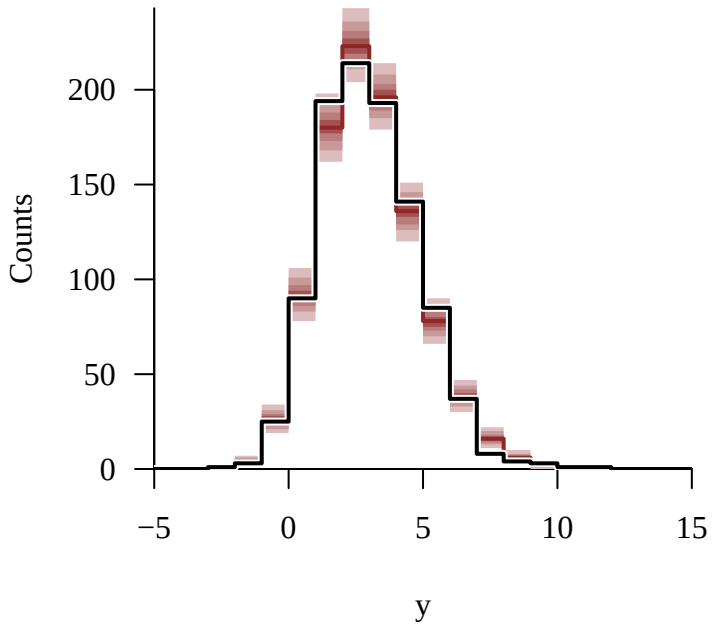
```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

plot(abs(c(samples[['norm_error']], recursive=TRUE)),
        c(samples[['ISSE']], recursive=TRUE),
     col=c_dark, pch=16,
     xlim=c(0, 0.03), xlab="Magnitude of Exact Error",
     ylim=c(0, 0.03), ylab="Importance Sampling Standard Error")
lines(c(0, 1), c(0, 1), col="#DDDDDD", lwd=2, lty=2)
```



On a more positive note our previous retrodictive performance is back.

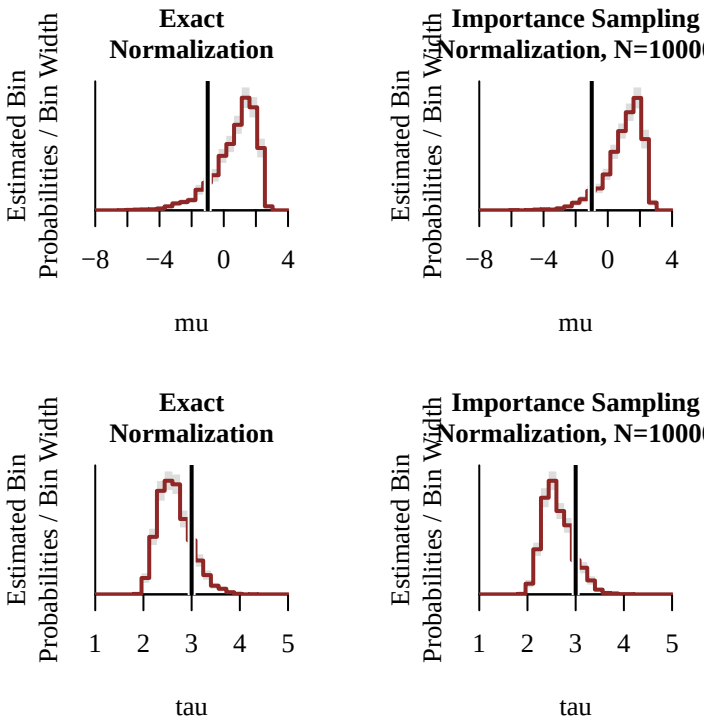
```
par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))

pred_names <- grep('y_pred', names(samples), value=TRUE)
hist_retro(data$y, samples, pred_names, -5, 15, 1, 'y')
```

```
Warning in hist_retro(data$y, samples, pred_names, -5, 15, 1, "y"): 1
predictive values (0.0%) fell below the histogram binning.
```

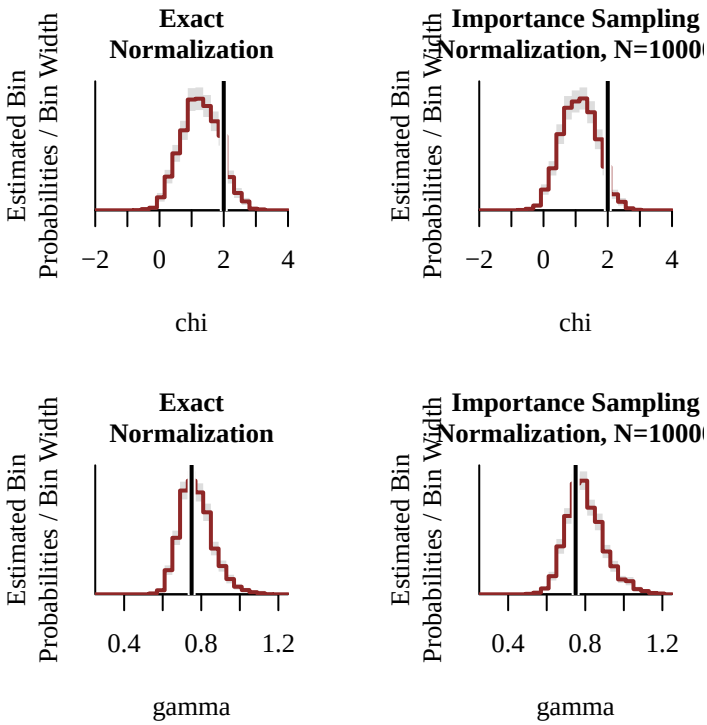


At the same time the posterior inferences are now indistinguishable from those of the exact model.

```
samples_exact <- util$extract_expectands(fit_exact)

par(mfrow=c(2, 2))


util$plot_expectand_pushforward(samples_exact[['mu']], 25, 'mu',
                                flim=c(-8, 4), baseline=mu,
                                main="Exact\nNormalization")
util$plot_expectand_pushforward(samples[['mu']], 25, 'mu',
                                flim=c(-8, 4), baseline=mu,
                                main="Importance Sampling\nNormalization, N=10000")

util$plot_expectand_pushforward(samples_exact[['tau']], 25, 'tau',
                                flim=c(1, 5), baseline=tau,
                                main="Exact\nNormalization")
util$plot_expectand_pushforward(samples[['tau']], 25, 'tau',
                                flim=c(1, 5), baseline=tau,
                                main="Importance Sampling\nNormalization, N=10000")
```




```
par(mfrow=c(2, 2))

util$plot_expectand_pushforward(samples_exact[['chi']], 25, 'chi',
                                flim=c(-2, 4), baseline=chi,
                                main="Exact\nNormalization")
util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
                                flim=c(-2, 4), baseline=chi,
                                main="Importance Sampling\nNormalization, N=10000")

util$plot_expectand_pushforward(samples_exact[['gamma']], 25, 'gamma',
                                flim=c(0.25, 1.25), baseline=gamma,
                                main="Exact\nNormalization")
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                flim=c(0.25, 1.25), baseline=gamma,
                                main="Importance Sampling\nNormalization, N=10000")
```



Although more general than Monte Carlo estimation, importance sampling estimation is also more fragile. If the importance sampling ensemble is too small then the importance sampling diagnostics can be unreliable, leaving us ignorant to large errors. When using importance sampling estimation in practice it becomes especially important to consider a sequence of larger ensembles to verify that the importance sampling estimates and the associated diagnostics are actually well-behaved.

3.4 Multi-Dimensional Probabilistic Selection

To see how the performance of our normalization integral estimation methods scales to higher dimensions let's consider a selection process that spans a real-valued latent space with K dimensions, $Y = \mathbb{R}^K$.

In particular let's consider a latent probability distribution specified by a K-dimensional multivariate normal density function,

$$p(\mathbf{y} \mid \mu, \Sigma)$$

where the covariance matrix is parameterized by a Cholesky matrix Ψ and vector of scales τ,

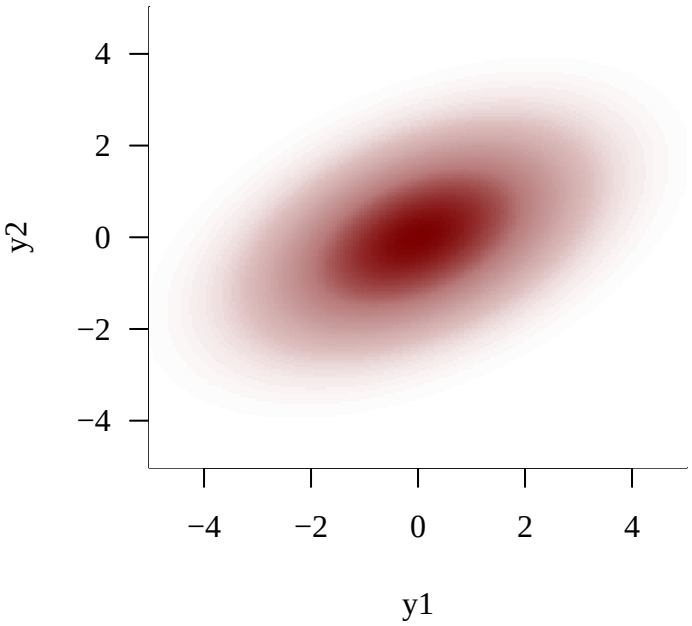
$$\Sigma = \text{diag}(\tau) \cdot \Psi \cdot \Psi^T \cdot \text{diag}(\tau).$$

Altogether the latent probability distribution parameters are

$$\phi = (\mu, \tau, \Psi).$$

To facilitate visualization let's first consider an example with $K = 2$.

```
disc_colors <- c("#FFFFFF", "#DCBCBC", "#C79999", "#B97C7C",
                 "#A25050", "#8F2727", "#7C0000")
cont_colors <- colormap(colormap=disc_colors, nshades=100)

N <- 200
y1s <- seq(-5, 5, 10 / (N - 1))
y2s <- seq(-5, 5, 10 / (N - 1))


K <- 2

mu <- c(0, 0)

sigma1 <- 1.5
sigma2 <- 1.5
rho <- 0.5
Sigma <- matrix(0, nrow=K, ncol=K)
Sigma[1, 1] <- sigma1**2
Sigma[1, 2] <- rho * sigma1 * sigma2
Sigma[2, 1] <- rho * sigma1 * sigma2
Sigma[2, 2] <- sigma2**2
L <- chol(Sigma)

latent_pds <- matrix(0, nrow=N, ncol=N)

for (n in 1:N) {
  for (m in 1:N) {
    y <- c(y1s[n], y2s[m])
    z <- backsolve(L, y)
    Q <- t(z) %*% z
    latent_pds[n, m] <- exp(-0.5 * Q - (K / 2) * log(2 * pi) - sum(log(diag(L))) )
  }
}

par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))
image(y1s, y2s, latent_pds, col=rev(cont_colors),
      main="Latent Probability Density Function p(y1, y2)",
      xlab="y1", ylab="y2")
```



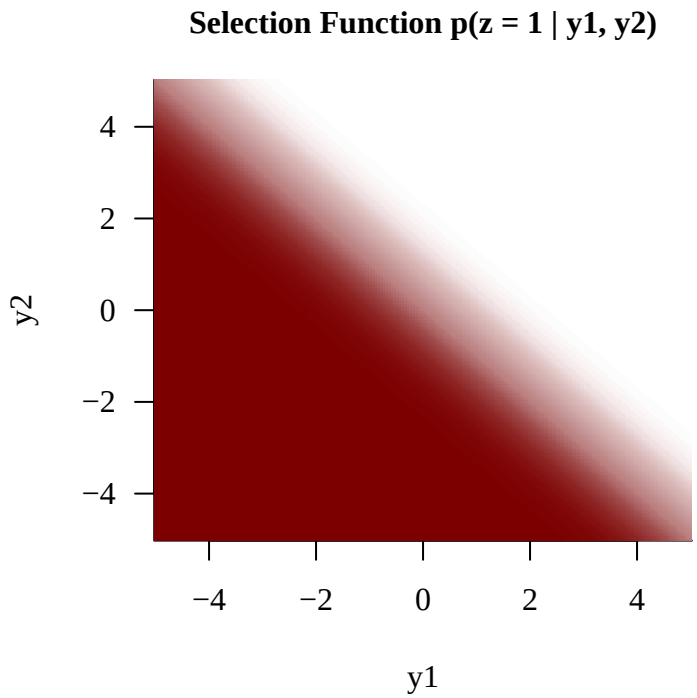
A higher-dimensional observational space allows for all kinds of interesting selection functions. Here we'll consider a selection function that begins to suppress latent events when the *sum* of the component values becomes too small or too large, depending on the sign of the parameter γ,

$$S(\mathbf{y}; \psi = (\chi, \gamma)) = \Pi_{\text{normal}}\left(\sum_{k=1}^{K} y_k; \chi, \gamma\right).$$

For example in two dimensions this selection function rejects latent events whose summed components fall above 0.

```
chi <- 0
gamma <- -1

selection_probs <- matrix(0, nrow=N, ncol=N)

for (n in 1:N) {
  for (m in 1:N) {
    y <- c(y1s[n], y2s[m])
    selection_probs[n, m] <- pnorm(gamma * (sum(y) - chi))
  }
}

par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))
```

```
image(y1s, y2s, selection_probs, col=rev(cont_colors),
      main="Selection Function p(z = 1 | y1, y2)",
      xlab="y1", ylab="y2")
```



Somewhat miraculously we can actually derive an analytic normalization integral for this particular normalization model,

$$Z(\mu, \tau, \Psi, \chi, \gamma) = \Phi\left(\frac{\gamma \cdot (\mu_S - \chi)}{\sqrt{1 + (\gamma\,\tau_S)^2}}\right)$$

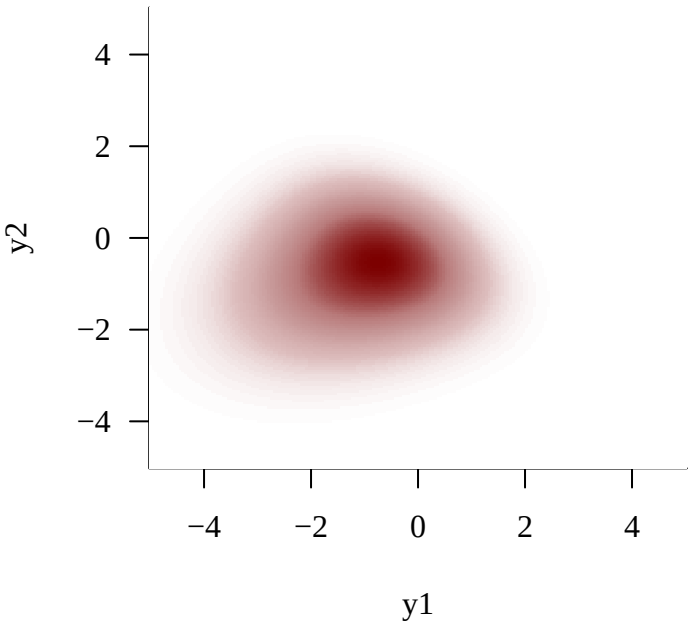
where

$$\mu_S = \sum_{k=1}^{K} \mu_k$$

and τ_S is a pretty complicated function of the covariance matrix defined by τ and Ψ. As my favor to you I have relegated the details of this derivation to the Appendix.

```
observed_pds <- (selection_probs * latent_pds)

mu_sum <- sum(mu)

Lambda <- solve(Sigma)
```

```
k <- 1
idxs <- setdiff(1:K, k)

l <- rep(0, K - 1)
A <- matrix(0, nrow=(K - 1), ncol=(K - 1))

for (i in 1:(K - 1))
  for (j in 1:(K - 1))
    A[i, j] = Lambda[idxs[i], idxs[j]] -
              Lambda[k, idxs[j]] -
              Lambda[idxs[i], k] +
              Lambda[k, k]

for (i in 1:(K - 1))
  l[i] = Lambda[k, idxs[i]] - Lambda[k, k]

omega <- ( Lambda[k, k] - t(l) %*% solve(A, l) )[1, 1]
tau_sum <- 1 / sqrt(omega)

norm <- pnorm(gamma * (mu_sum - chi) / sqrt(1 + (gamma * tau_sum)**2), 0, 1)

observed_pds <- observed_pds / norm

par(mfrow=c(1, 1), mar = c(5, 5, 3, 1))
image(y1s, y2s, observed_pds, col=rev(cont_colors),
      main="Observed Probability Density Function p(y1, y2 | z = 1)",
      xlab="y1", ylab="y2")
```



3.4.1 Simulating Data

To push our normalization integral estimation methods let's go beyond the two dimensions we used for visualization all the way to five entire dimensions. We'll also sample the multivariate normal density function locations, scales, and Cholesky factor so that we don't have to assign all of those values one by one.

```
K <- 5
N <- 1000

chi <- 2
gamma <- -1
```

```
simu <- stan(file="stan_programs/simu_selection_multi.stan",
             iter=1, warmup=0, chains=1,
             data=list("N" = N, "K" = K, "chi" = chi, "gamma" = gamma),
             seed=4838282, algorithm="Fixed_param")
```

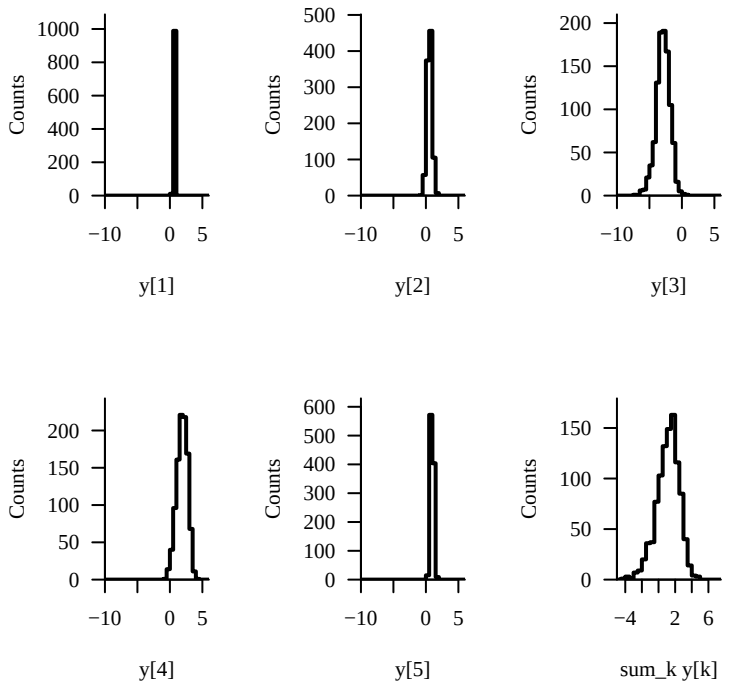
```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
```

```
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                0.002 seconds (Sampling)
Chain 1:                0.002 seconds (Total)
Chain 1:
```

```
mu <- extract(simu)$mu[1,]
tau <- extract(simu)$tau[1,]
Psi <- extract(simu)$Psi[1,,]
y <- extract(simu)$y[1,,]
N_reject <- extract(simu)$N_reject[1]
```

The individual components exhibit a wide range of behaviors, some concentrated and some diffuse but most exhibiting a weak skew. We also see a substantial skew in the sum of the components, which we expect from the selection process.

```
par(mfrow=c(2, 3))

for (k in 1:K)
  plot_line_hist(y[,k], -10, 6, 0.5, xlab=paste0("y[", k, "]"))

plot_line_hist(sapply(1:N, function(n) sum(y[n,])),
               -5, 7.5, 0.5, xlab="sum_k y[k]")
```



3.4.2 Ignoring The Selection Process

Let's once again see what happens when we ignore the selection process entirely and try to fit the latent model directly to the observed data.

```
data <- list("N" = N, "K" = K, "y" = y)

fit <- stan(file="stan_programs/fit_no_selection_multi.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

We have no signs of computational issues.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples,
                                       c('mu', 'tau', 'Psi'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples,
                                     exclude_zvar=TRUE)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

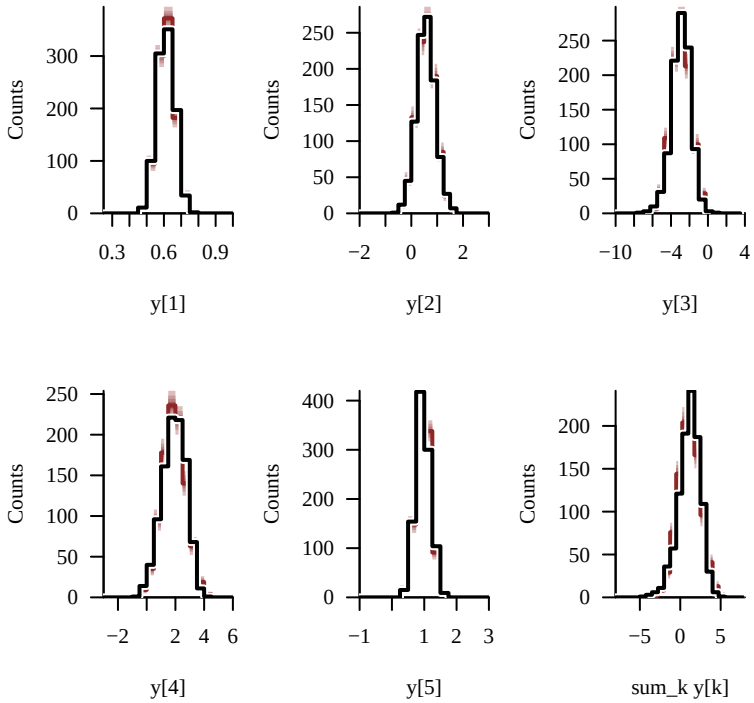
Someone surprisingly the retrodictive checks don't look terrible within each individual component. That said there is some retrodictive tension in the histogram of the summed components, with the observed data exhibiting a slight skew that the model doesn't seem to be able to accommodate.

```
par(mfrow=c(2, 3), mar = c(5, 4, 2, 1))

lower <- c(0.25, -2, -10, -3, -1)
upper <- c(1, 3, 4, 6, 3)
delta <- c(0.05, 0.25, 0.75, 0.5, 0.25)
for (k in 1:data$K) {
  pred_names <- grep(paste0('y_pred\\[[0-9]*,', k),
                     names(samples), value=TRUE)
  hist_retro(data$y[,k], samples, pred_names, lower[k], upper[k], delta[k],
             xlab=paste0('y[', k, ']'))
}
```

```
Warning in hist_retro(data$y[, k], samples, pred_names, lower[k], upper[k], : 4
predictive values (0.0%) fell above the histogram binning.
```

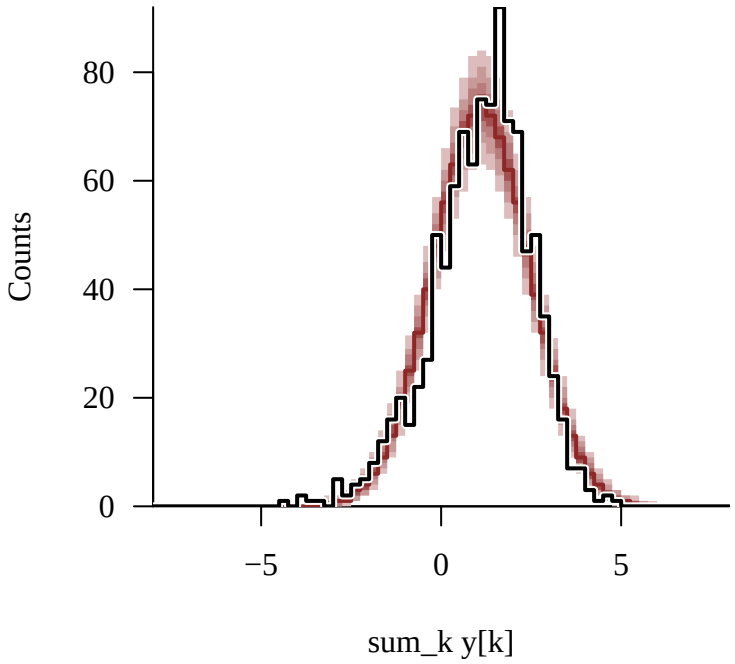
```
sum_y <- sapply(1:N, function(n) sum(data$y[n,]))

names <- c()
sum_samples <- list()
for (n in 1:data$N) {
  name <- paste0('sum_y[', n, ']')
  names <- c(names, name)

  summands <- lapply(1:data$K,
                     function(k) samples[[paste0('y_pred[', n, ',', k, ']')]])
```

```
  sum_samples[[name]] <- Reduce("+", summands)
}

hist_retro(sum_y, sum_samples, names, -8, 8, 0.75, xlab="sum_k y[k]")
```



We can probe this tension more carefully by using a finer histogram as our summary statistic. In a practical application we might use the weak tension that we see here to motivate more targeted summary statistics, such as an empirical skewness statistic.

```
par(mfrow=c(1, 1))

hist_retro(sum_y, sum_samples, names, -8, 8, 0.25, xlab="sum_k y[k]")
```



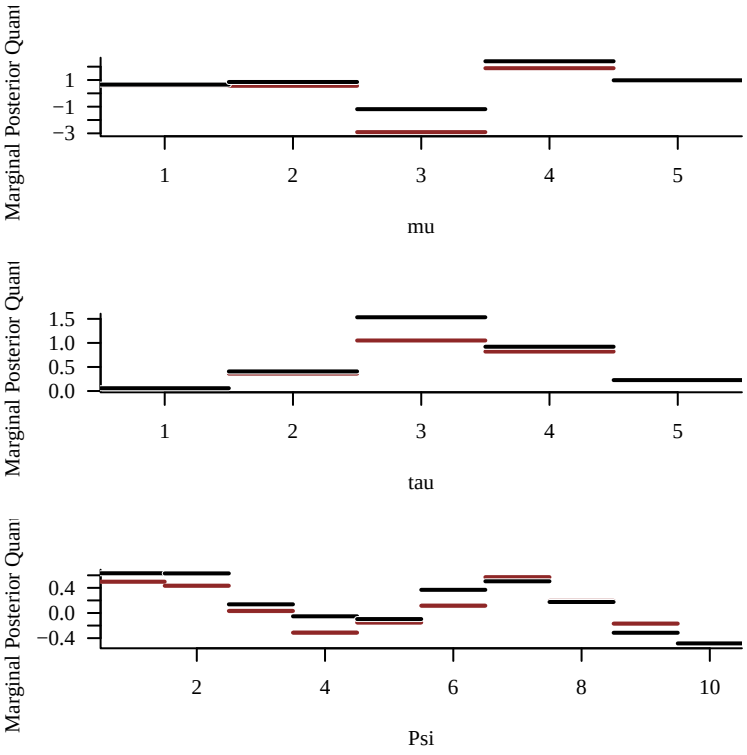
This passable retrodictive performance, however, comes only because the model is substantially pulled away from the true model configuration, especially in the location and scale of the third component and the Cholesky factors.

```
plot_disc_pushforward_quantiles <- function(samples, names,
                                            x_name="", display_ylim=NULL,
                                            main="", baselines=NULL) {
  # Check that names are in samples
  all_names <- names(samples)

  bad_names <- setdiff(names, all_names)
  if (length(bad_names) > 0) {
    warning(sprintf('The expectand names %s are not in the `samples` object and will be ignor
                    paste(bad_names, collapse=", ")))
  }

  good_names <- intersect(names, all_names)
  if (length(good_names) == 0) {
    stop('There are no valid expectand names.')
  }
  names <- good_names

  # Compute x-axis gridding
  N <- length(names)
```

```
idx <- rep(1:N, each=2)
xs <- sapply(1:length(idx), function(k) if(k %% 2 == 0) idx[k] + 0.5
             else idx[k] - 0.5)

# Compute pushforward quantiles
probs = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)
cred <- sapply(1:N, function(n) quantile(c(t(samples[[names[n]]]),
                                           recursive=TRUE),
                                         probs=probs))
pad_cred <- do.call(cbind, lapply(idx, function(n) cred[1:9,n]))

# Plot
if (is.null(display_ylim)) {
  if (is.null(baselines)) {
    display_ylim <- c(min(cred[1,]), max(cred[9,]))
  } else {
    display_ylim <- c(min(c(cred[1,], baselines)),
                      max(c(cred[9,], baselines)))
  }
  delta <- 0.05 * (display_ylim[2] - display_ylim[1])
  display_ylim[1] <- display_ylim[1] - delta
  display_ylim[2] <- display_ylim[2] + delta
}

plot(1, type="n", main=main,
     xlim=c(0.5, N + 0.5), xlab=x_name,
     ylim=display_ylim, ylab="Marginal Posterior Quantiles")

polygon(c(xs, rev(xs)), c(pad_cred[1,], rev(pad_cred[9,])),
        col = c_light, border = NA)
polygon(c(xs, rev(xs)), c(pad_cred[2,], rev(pad_cred[8,])),
        col = c_light_highlight, border = NA)
polygon(c(xs, rev(xs)), c(pad_cred[3,], rev(pad_cred[7,])),
        col = c_mid, border = NA)
polygon(c(xs, rev(xs)), c(pad_cred[4,], rev(pad_cred[6,])),
        col = c_mid_highlight, border = NA)
for (n in 1:N) {
  lines(xs[(2 * n - 1):(2 * n)], pad_cred[5,(2 * n - 1):(2 * n)],
        col=c_dark, lwd=2)
}

# Plot baseline values, if applicable
```

```
  if (!is.null(baselines)) {
    if (length(baselines) != N) {
      warning('The list of baseline values has the wrong dimension and baselines will not be
    } else {
      for (n in 1:N) {
        lines(xs[(2 * n - 1):(2 * n)], rep(baselines[n], 2),
              col="white", lwd=3)
        lines(xs[(2 * n - 1):(2 * n)], rep(baselines[n], 2),
              col="black", lwd=2)
      }
    }
  }
}
```

```
par(mfrow=c(3, 1), mar = c(5, 4, 2, 1))

mu_names <- sapply(1:data$K, function(k) paste0('mu[', k, ']'))
plot_disc_pushforward_quantiles(samples, mu_names, "mu", baselines=mu)

tau_names <- sapply(1:data$K, function(k) paste0('tau[', k, ']'))
plot_disc_pushforward_quantiles(samples, tau_names, "tau", baselines=tau)

Psi_lt <- c(sapply(2:K, function(k)
  sapply(1:(k - 1), function(kk) Psi[k, kk])),
  recursive=TRUE)
Psi_lt_names <- c(sapply(2:K, function(k)
  sapply(1:(k - 1), function(kk)
    paste0('Psi[', k, ',', kk, ']'))),
  recursive=TRUE)
plot_disc_pushforward_quantiles(samples, Psi_lt_names, "Psi", baselines=Psi_lt)
```



3.4.3 Modeling An Unknown Selection Behavior

Next let's see how difficult it is to infer the selection function configuration in the somewhat-unrealistic circumstance where we know the exact configuration of the latent probability distribution.

3.4.3.1 Exact

First let's take advantage of the analytic result to evaluate the normalization integral exactly.

```
data$mu <- mu
data$tau <- tau
data$Psi <- Psi

fit_exact <- stan(file="stan_programs/fit_unknown_selection_multi_exact1.stan",
                  data=data, seed=8438338,
                  warmup=1000, iter=2024, refresh=0)
```

Even in this ideal case we encounter some diagnostic warnings. In addition to the small number of divergent transitions the $\hat{R}$-warnings suggest strong multi-modality in the posterior distribution.

```
diagnostics <- util$extract_hmc_diagnostics(fit_exact)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  Chain 3: 4 of 1024 transitions (0.4%) diverged.

  Chain 4: 17 of 1024 transitions (1.7%) diverged.

  Divergent Hamiltonian transitions result from unstable numerical
trajectories.  These instabilities are often due to degenerate target
geometry, especially "pinches".  If there are only a small number of
divergences then running with adept_delta larger than 0.801 may reduce
the instabilities at the cost of more expensive Hamiltonian
transitions.
```

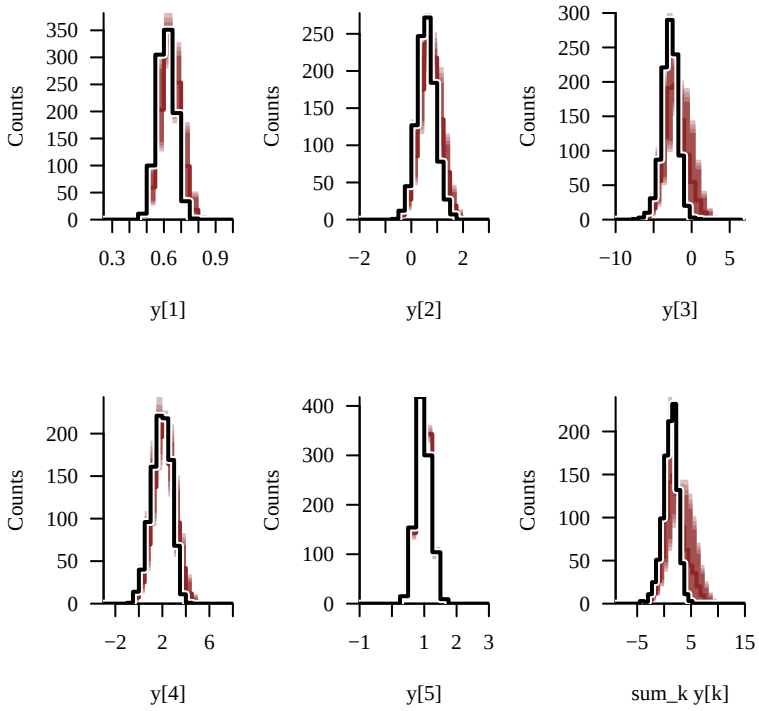
```
samples <- util$extract_expectands(fit_exact)
base_samples <- util$filter_expectands(samples,
                                       c('chi', 'gamma'))
util$check_all_expectand_diagnostics(base_samples)
```

```
chi:
  Split hat{R} (10.912) exceeds 1.1!

gamma:
  Split hat{R} (3.158) exceeds 1.1!


Split Rhat larger than 1.1 suggests that at least one of the Markov
chains has not reached an equilibrium.
```

Overall the retrodictive performance looks reasonable, although the wide posterior predictive quantile ribbons are also consistent with posterior multi-modality. This is especially true for the histograms of third components and the summed components which both exhibit a shoulder in the posterior predictive distribution.

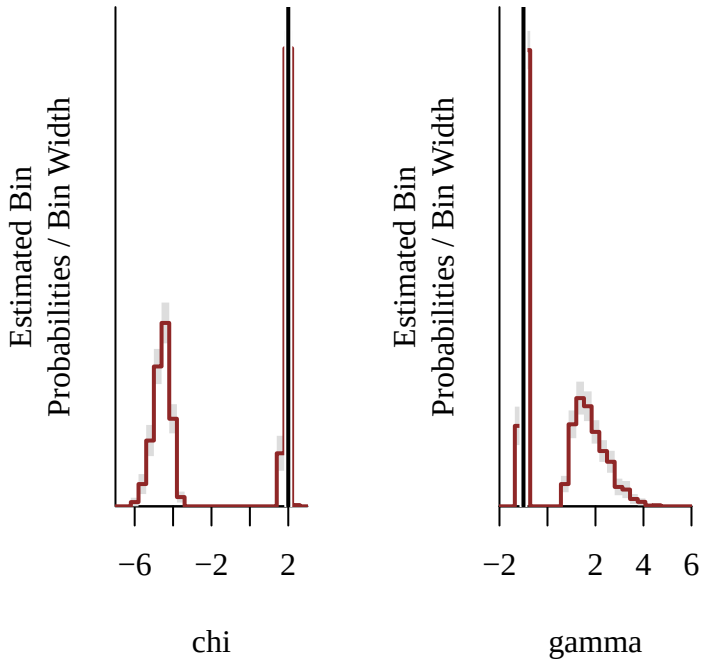
```
par(mfrow=c(2, 3), mar = c(5, 4, 2, 1))

lower <- c(0.25, -2, -10, -3, -1)
```

```
upper <- c(1, 3, 7, 8, 3)
delta <- c(0.05, 0.25, 0.75, 0.5, 0.25)
for (k in 1:data$K) {
  pred_names <- grep(paste0('y_pred\\[[0-9]*,', k),
                     names(samples), value=TRUE)
  hist_retro(data$y[,k], samples, pred_names, lower[k], upper[k], delta[k],
             xlab=paste0('y[', k, ']'))
}


sum_y <- sapply(1:N, function(n) sum(data$y[n,]))

names <- c()
sum_samples <- list()
for (n in 1:data$N) {
  name <- paste0('sum_y[', n, ']')
  names <- c(names, name)

  summands <- lapply(1:data$K,
                     function(k) samples[[paste0('y_pred[', n, ',', k, ']')]])
  sum_samples[[name]] <- Reduce("+", summands)
}

hist_retro(sum_y, sum_samples, names, -9, 15, 0.75, xlab="sum_k y[k]")
```



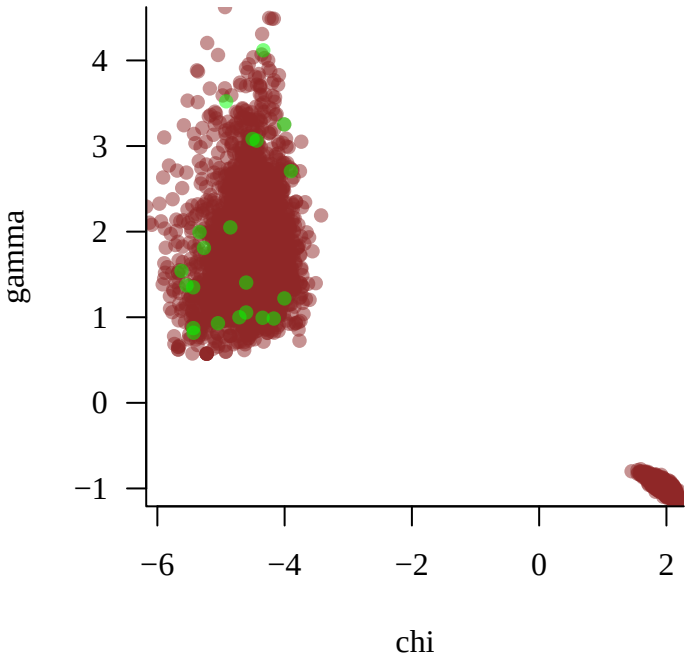
Indeed our posterior inferences for both χ and γ concentrate into two peaks.

```
par(mfrow=c(1, 2))

util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
                                flim=c(-7, 3), baseline=chi)
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                flim=c(-2, 6), baseline=gamma)
```



This is easier to see with a pairs plot of the two parameters. The top-left posterior mode corresponds to a selection function that sharply *rises* once the summed components surpass a negative threshold while the bottom-right posterior corresponds to a selection function that moderately *falls* once the summed components surpass a positive threshold.

```
util$plot_div_pairs(c('chi'), c('gamma'), samples, diagnostics)
```



Let's say that in this case our domain expertise excludes a rising selection function. This is often possible, for example, when the selection function models an experimental apparatus. We can then use that domain expertise to motivate a hard upper bound on γ which should immediately eliminate the spurious top-left mode.

```
fit_exact <- stan(file="stan_programs/fit_unknown_selection_multi_exact2.stan",
                  data=data, seed=8438338,
                  warmup=1000, iter=2024, refresh=0)
```

Encouragingly the previous diagnostics warnings have vanished.

```
diagnostics <- util$extract_hmc_diagnostics(fit_exact)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

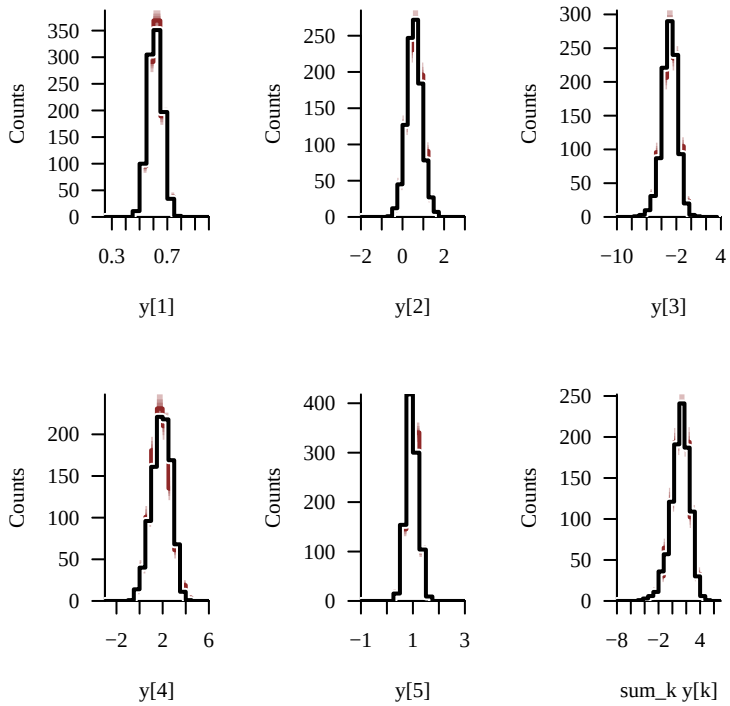
```
samples <- util$extract_expectands(fit_exact)
base_samples <- util$filter_expectands(samples,
                                       c('chi', 'gamma'))
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

Moreover the posterior predictive distribution now concentrates much more strongly around the observed histogram summary statistics.

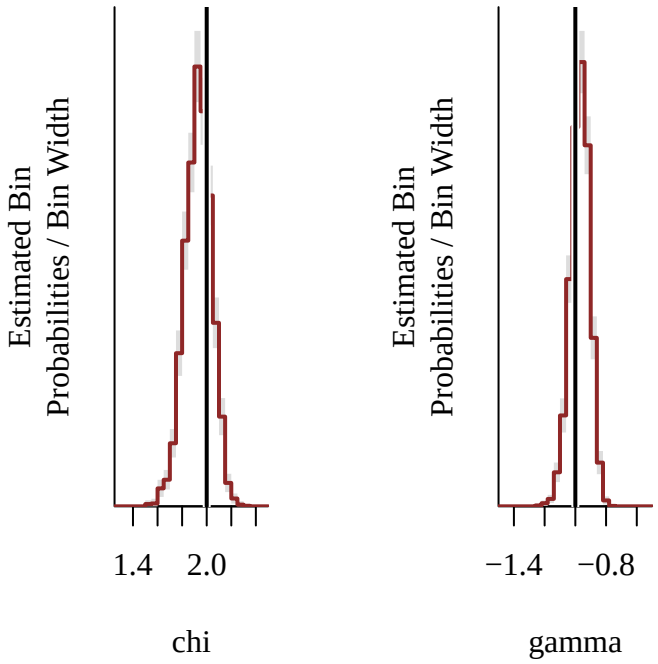
```
par(mfrow=c(2, 3))

lower <- c(0.25, -2, -10, -3, -1)
upper <- c(1, 3, 4, 6, 3)
delta <- c(0.05, 0.25, 0.75, 0.5, 0.25)
for (k in 1:data$K) {
  pred_names <- grep(paste0('y_pred\\[[0-9]*,', k),
                     names(samples), value=TRUE)
  hist_retro(data$y[,k], samples, pred_names, lower[k], upper[k], delta[k],
             xlab=paste0('y[', k, ']'))
}


sum_y <- sapply(1:N, function(n) sum(data$y[n,]))

names <- c()
sum_samples <- list()
for (n in 1:data$N) {
  name <- paste0('sum_y[', n, ']')
  names <- c(names, name)

  summands <- lapply(1:data$K,
                     function(k) samples[[paste0('y_pred[', n, ',', k, ']')]])
  sum_samples[[name]] <- Reduce("+", summands)
}

hist_retro(sum_y, sum_samples, names, -8, 7, 0.75, xlab="sum_k y[k]")
```

```
Warning in hist_retro(sum_y, sum_samples, names, -8, 7, 0.75, xlab = "sum_k
y[k]"): 2 predictive values (0.0%) fell below the histogram binning.
```



Our posterior inferences now concentrate not into a single mode but also around the true model configuration from which we simulated our data.

```
par(mfrow=c(1, 2))

util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
                                flim=c(1.25, 2.5), baseline=chi)
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                flim=c(-1.5, -0.5), baseline=gamma)
```



3.4.3.2 Monte Carlo

Confident that we can fit the exact model let's see how our approximation methods fare. Outside of a single latent dimension we can no longer rely on numerical quadrature, but we can consider Monte Carlo estimates of the normalization integral. Here we'll use a moderate ensemble, in between the smaller and large ensembles that we considered in the previous Monte Carlo exercise.

```
data$N_MC <- 1000

fit <- stan(file="stan_programs/fit_unknown_selection_multi_mc.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

All quiet on the computational front.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

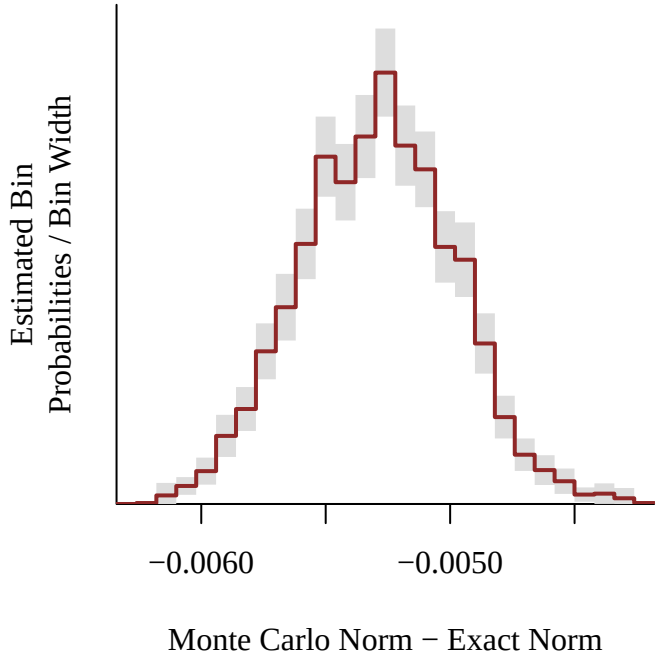
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples,
                                       c('chi', 'gamma'))
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

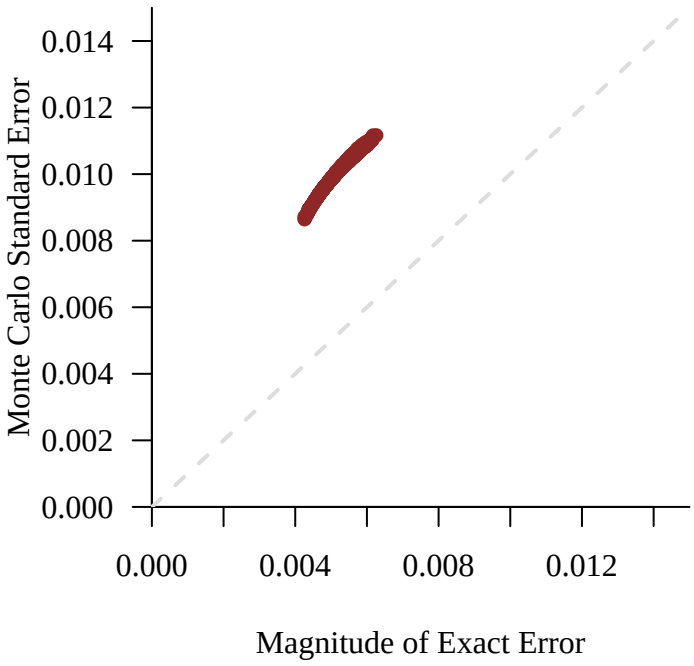
Despite the higher-dimension the exact errors in the normalization integral estimates have not grown relative to what we saw in the one-dimensional example. Robustness to dimensionality is one of the most powerful features of the Monte Carlo method.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_expectand_pushforward(samples[['norm_error']], 25,
                                'Monte Carlo Norm - Exact Norm')
```



The Monte Carlo standard errors continue to provide a good approximation to the exact errors, giving us confidence that we can apply the method in more realistic settings where we are not blessed with the exact normalization integral for comparison.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

plot(abs(c(samples[['norm_error']], recursive=TRUE)),
         c(samples[['MCSE']], recursive=TRUE),
     col=c_dark, pch=16,
     xlim=c(0, 0.015), xlab="Magnitude of Exact Error",
     ylim=c(0, 0.015), ylab="Monte Carlo Standard Error")
lines(c(0, 1), c(0, 1), col="#DDDDDD", lwd=2, lty=2)
```

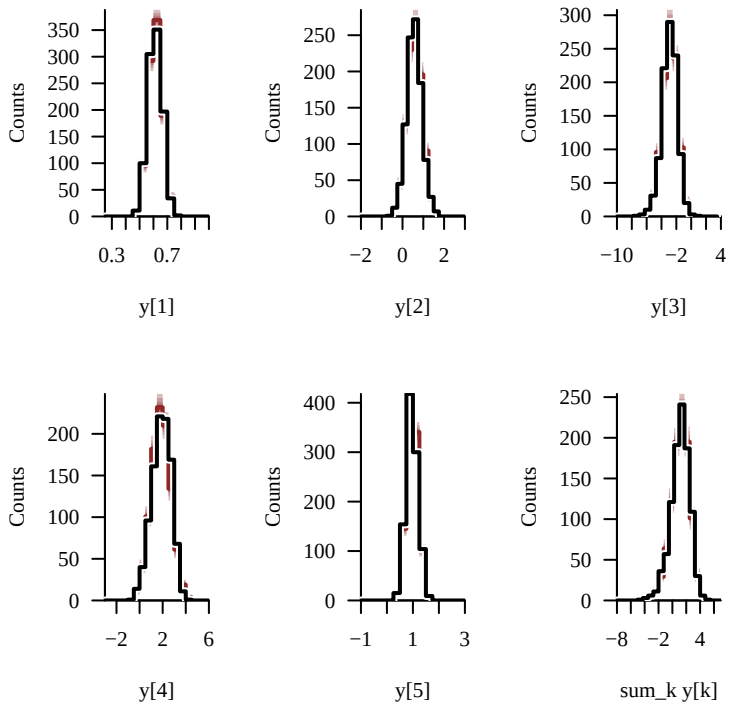


There don't seem to be any changes to the retrodictive performance.

```
par(mfrow=c(2, 3))

lower <- c(0.25, -2, -10, -3, -1)
upper <- c(1, 3, 4, 6, 3)
delta <- c(0.05, 0.25, 0.75, 0.5, 0.25)
for (k in 1:data$K) {
  pred_names <- grep(paste0('y_pred\\[[0-9]*,', k),
                     names(samples), value=TRUE)
  hist_retro(data$y[,k], samples, pred_names, lower[k], upper[k], delta[k],
             xlab=paste0('y[', k, ']'))
}
```

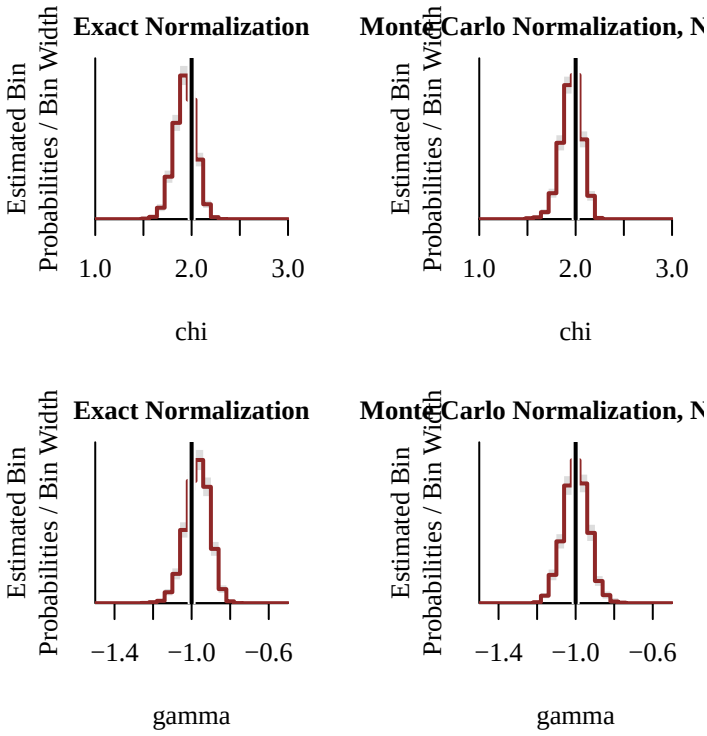
```
sum_y <- sapply(1:N, function(n) sum(data$y[n,]))

names <- c()
sum_samples <- list()
for (n in 1:data$N) {
  name <- paste0('sum_y[', n, ']')
  names <- c(names, name)

  summands <- lapply(1:data$K,
                     function(k) samples[[paste0('y_pred[', n, ',', k, ']')]])
  sum_samples[[name]] <- Reduce("+", summands)
}

hist_retro(sum_y, sum_samples, names, -8, 7, 0.75, xlab="sum_k y[k]")
```

```
Warning in hist_retro(sum_y, sum_samples, names, -8, 7, 0.75, xlab = "sum_k
y[k]"): 2 predictive values (0.0%) fell below the histogram binning.
```



Most importantly the posterior inferences are consistent with those derived from the exact model.

```
par(mfrow=c(2, 2))

samples_exact <- util$extract_expectands(fit_exact)

util$plot_expectand_pushforward(samples_exact[['chi']], 25, 'chi',
                                flim=c(1, 3), baseline=chi,
                                main="Exact Normalization")
util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
                                flim=c(1, 3), baseline=chi,
                                main="Monte Carlo Normalization, N = 1000")

util$plot_expectand_pushforward(samples_exact[['gamma']], 25, 'gamma',
                                flim=c(-1.5, -0.5), baseline=gamma,
                                main="Exact Normalization")
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                flim=c(-1.5, -0.5), baseline=gamma,
                                main="Monte Carlo Normalization, N = 1000")
```



3.4.4 Modeling Unknown Selection and Latent Behavior

We are now ready for our final challenge of jointly fitting the configurations of the latent probability distribution and selection function. Note that we'll maintain the negativity constraint on γ to avoid the selection function degeneracy that we encountered above.

3.4.4.1 Exact

First up is the exact implementation of the normalization integral.

```
fit_exact <- stan(file="stan_programs/fit_unknown_both_multi_exact.stan",
                  data=data, seed=8438338,
                  warmup=1000, iter=2024, refresh=0)
```

Diagnostics are copacetic.

```
diagnostics <- util$extract_hmc_diagnostics(fit_exact)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit_exact)
base_samples <- util$filter_expectands(samples,
                                       c('chi', 'gamma', 'mu', 'tau', 'Psi'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples,
                                     exclude_zvar=TRUE)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

There doesn't seem to be any retrodictive tension across the behavior of the individual components or their sum.

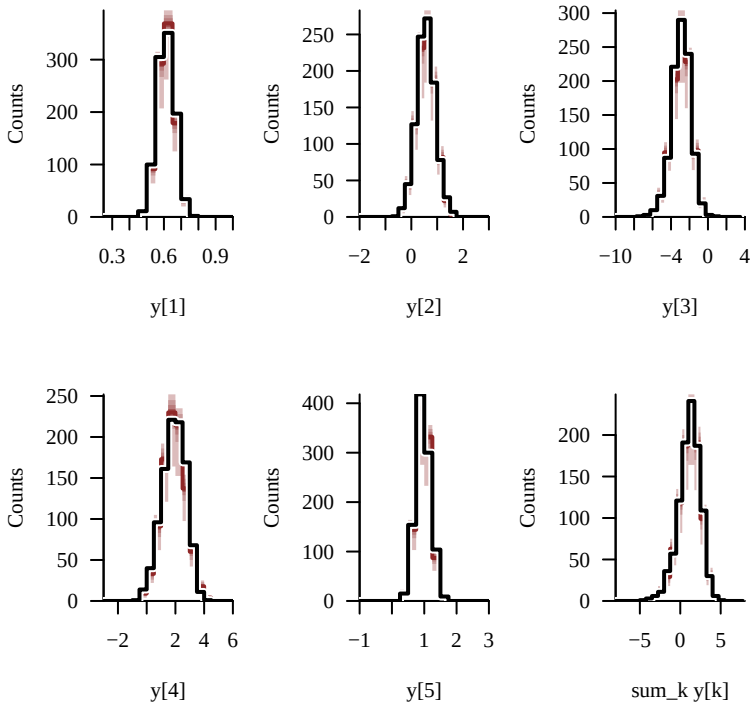
```
par(mfrow=c(2, 3), mar = c(5, 4, 2, 1))

lower <- c(0.25, -2, -10, -3, -1)
upper <- c(1, 3, 4, 6, 3)
delta <- c(0.05, 0.25, 0.75, 0.5, 0.25)
```

```
for (k in 1:data$K) {
  pred_names <- grep(paste0('y_pred\\[[0-9]*,', k),
                     names(samples), value=TRUE)
  hist_retro(data$y[,k], samples, pred_names, lower[k], upper[k], delta[k],
             xlab=paste0('y[', k, ']'))
}
```

```
Warning in hist_retro(data$y[, k], samples, pred_names, lower[k], upper[k], : 1
predictive values (0.0%) fell above the histogram binning.
```

```
sum_y <- sapply(1:N, function(n) sum(data$y[n,]))

names <- c()
sum_samples <- list()
for (n in 1:data$N) {
  name <- paste0('sum_y[', n, ']')
  names <- c(names, name)

  summands <- lapply(1:data$K,
                     function(k) samples[[paste0('y_pred[', n, ',', k, ']')]])
  sum_samples[[name]] <- Reduce("+", summands)
}

hist_retro(sum_y, sum_samples, names, -8, 8, 0.75, xlab="sum_k y[k]")
```

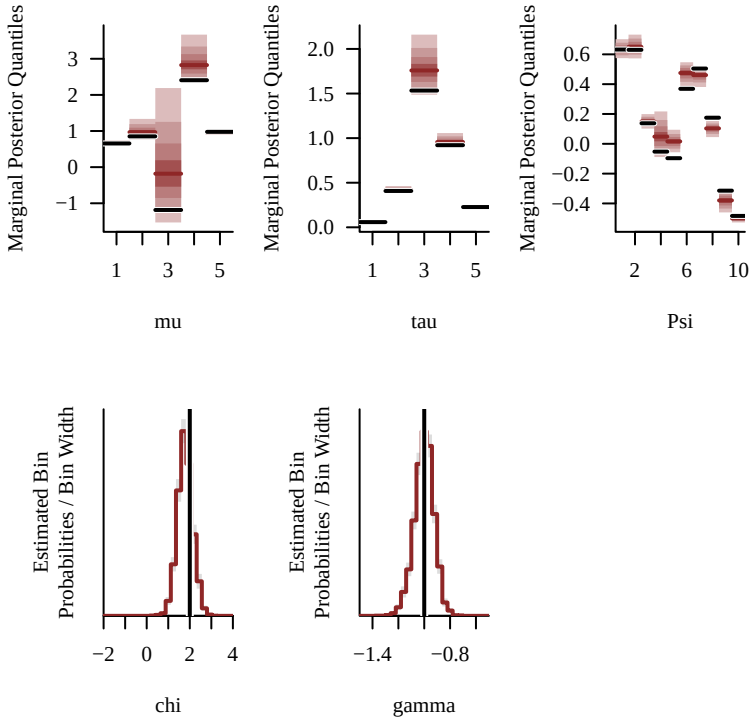
```
Warning in hist_retro(sum_y, sum_samples, names, -8, 8, 0.75, xlab = "sum_k
y[k]"): 3 predictive values (0.0%) fell below the histogram binning.
```



As a cherry on top our posterior inferences all concentrate around the true model configuration, although the uncertainties are quite large for some of the parameters. Even with the sign of γ fixed we still encounter strong degeneracies when trying to infer the latent probability distribution and selection function as the same time.

```
par(mfrow=c(2, 3))

mu_names <- sapply(1:data$K, function(k) paste0('mu[', k, ']'))
plot_disc_pushforward_quantiles(samples, mu_names, "mu", baselines=mu)

tau_names <- sapply(1:data$K, function(k) paste0('tau[', k, ']'))
plot_disc_pushforward_quantiles(samples, tau_names, "tau", baselines=tau)

Psi_lt <- c(sapply(2:K, function(k)
  sapply(1:(k - 1), function(kk) Psi[k, kk])),
  recursive=TRUE)
Psi_lt_names <- c(sapply(2:K, function(k)
  sapply(1:(k - 1), function(kk)
    paste0('Psi[', k, ',', kk, ']'))),
  recursive=TRUE)
plot_disc_pushforward_quantiles(samples, Psi_lt_names, "Psi", baselines=Psi_lt)

util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
```

```
                                    flim=c(-2, 4), baseline=chi)
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                    flim=c(-1.5, -0.5), baseline=gamma)
plot.new()
```



3.4.4.2 Importance Sampling

With numerical quadrature out of commission we'll have to jump straight to importance sampling estimation of the normalization integral. As we did in the multi-dimensional application of Monte Carlo we'll utilize a moderate importance sampling ensemble.

```
data$N_IS <- 1000

fit <- stan(file="stan_programs/fit_unknown_both_multi_is1.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

```
If rounding is intended please use the integer division operator %/%.
Info: Found int division at 'string', line 121, column 22 to column 26:
  N_IS / 2
```

```
Values will be rounded towards zero. If rounding is not desired you can write
the division as
  N_IS / 2.0
```

Uh, oh. The Hamiltonian Monte Carlo diagnostics look bad. Really bad.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  Chain 1: 1024 of 1024 transitions (100.0%) diverged.
  Chain 1: Averge proxy acceptance statistic (0.721) is
           smaller than 90% of the target (0.801).

  Chain 2: 1024 of 1024 transitions (100.0%) diverged.
  Chain 2: E-FMI = 0.026.

  Chain 3: 1024 of 1024 transitions (100.0%) diverged.
  Chain 3: E-FMI = 0.017.

  Chain 4: 1024 of 1024 transitions (100.0%) diverged.

  Divergent Hamiltonian transitions result from unstable numerical
trajectories.  These instabilities are often due to degenerate target
geometry, especially "pinches".  If there are only a small number of
divergences then running with adept_delta larger than 0.801 may reduce
the instabilities at the cost of more expensive Hamiltonian
transitions.

  E-FMI below 0.2 arise when a funnel-like geometry obstructs how
effectively Hamiltonian trajectories can explore the target
distribution.

  A small average proxy acceptance statistic indicates that the
adaptation of the numerical integrator step size failed to converge.
This is often due to discontinuous or imprecise gradients.
```

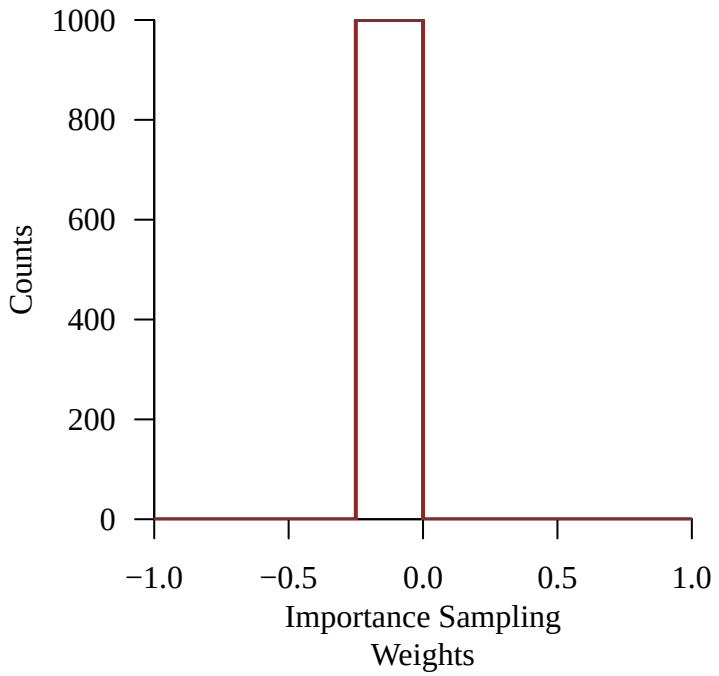
```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples,
                                       c('chi', 'gamma', 'mu', 'tau', 'Psi'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples,
                                     exclude_zvar=TRUE)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

Because we've already successfully fit the exact model we know that these computational issues have to be due to poor importance sampling estimation. In a more realistic application of importance sampling, however, we would not have the luxury of an exact fit for comparison. Instead we would have to rely on the importance sampling diagnostics themselves.

The importance weights appear to completely concentrate at zero across *every* evaluation of the normalization integral.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

weight_names <- grep('weights', names(samples), value=TRUE)
plot_histogram_pushforward(samples, weight_names, -1, 1, 0.25,
                           "Importance Sampling\nWeights")
```



We can confirm this by examining some of the importance weights directly.

```
samples[['weights[1]']][1,1:25]
```

```
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
samples[['weights[500]']][1,1:25]
```

```
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
samples[['weights[1000]']][1,1:25]
```

```
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Because the importance weights have all collapsed to zero the remaining importance sampling diagnostics become ill-defined.

```
samples[['khat']][1,1:25]
```

```
 [1] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
[20] NaN NaN NaN NaN NaN NaN
```

```
samples[['ISESS']][1,1:25]
```

```
 [1] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
[20] NaN NaN NaN NaN NaN NaN
```

An importance weight falls to zero when the corresponding reference sample is nowhere near the target probability distribution. This in turn is due to the the reference probability distribution being poorly aligned with the target probability distribution. For example the reference probability distribution might concentrate in the tails of the target probability distribution. Alternatively it might be too diffuse, making the target probability distribution too small of a target.

Either way a larger importance sampling ensemble should resolve the issue *in theory*. In practice, however, the size of the importance sampling ensemble needed to ensure reasonable estimation might be far beyond our limited computational resources.

Consequently the most effective practical strategy is usually to engineer a reference probability distribution that is better aligned with the target probability distribution. The feasibility of this strategy, however, is limited by our knowledge of the target probability distribution.

For example we might try to tune the reference probability distribution iteratively. If we can find an initial reference probability distribution where at least *some* of the importance weights are non-zero we can use the resulting approximate posterior inferences to inform a better reference probability distribution. That said these iterations are often time consuming, and

finding even a mediocre initialization can be extremely difficult especially when we're working in higher-dimensional spaces.

To conclude this exercise let's see what happens in an idealized setting where we can set the reference probability distribution to the true configuration of the latent probability distribution. If the exact posterior distribution concentrates around that true configuration then it may be good enough to ensure reasonably accurate importance sampling estimation for all of the normalization integral evaluations we need to fully explore the posterior distribution.

```
data$N_IS <- 1000

data$mu_true <- mu
data$tau_true <- tau
data$Psi_true <- Psi

fit <- stan(file="stan_programs/fit_unknown_both_multi_is2.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

```
If rounding is intended please use the integer division operator %/%.
Info: Found int division at 'string', line 124, column 22 to column 26:
  N_IS / 2
Values will be rounded towards zero. If rounding is not desired you can write
the division as
  N_IS / 2.0
```

The Hamiltonian Monte Carlo diagnostics have mostly cleared up, with the single tree depth warning not a serious concern.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  Chain 3: 1 of 1024 transitions (0.09765625%) saturated the maximum treedepth of 10.

  Numerical trajectories that saturate the maximum treedepth have
terminated prematurely.  Increasing max_depth above 10 should result in
more expensive, but more efficient, Hamiltonian transitions.
```

```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples,
                                       c('chi', 'gamma', 'mu', 'tau', 'Psi'),
```

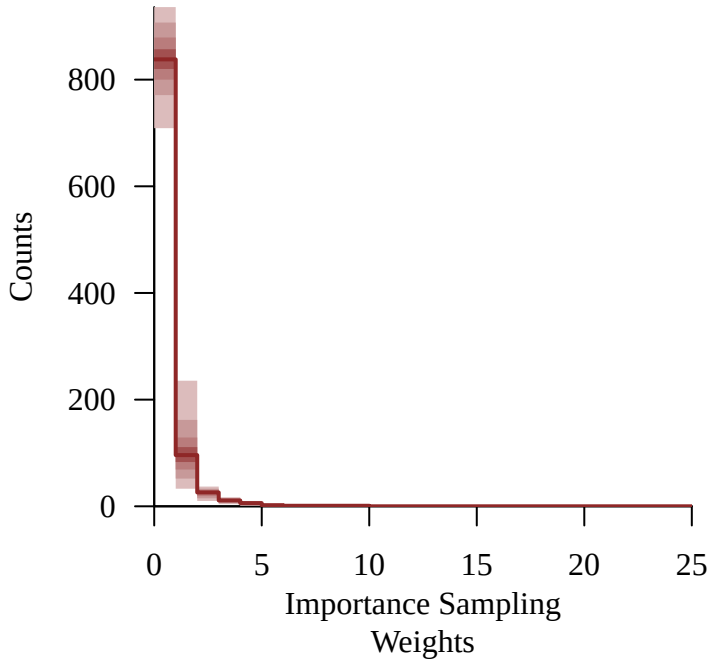
```
                                          check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples,
                                     exclude_zvar=TRUE)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

Welcomingly the importance weights are no longer all zero. Many are close to zero, however, with a long tail of larger values.

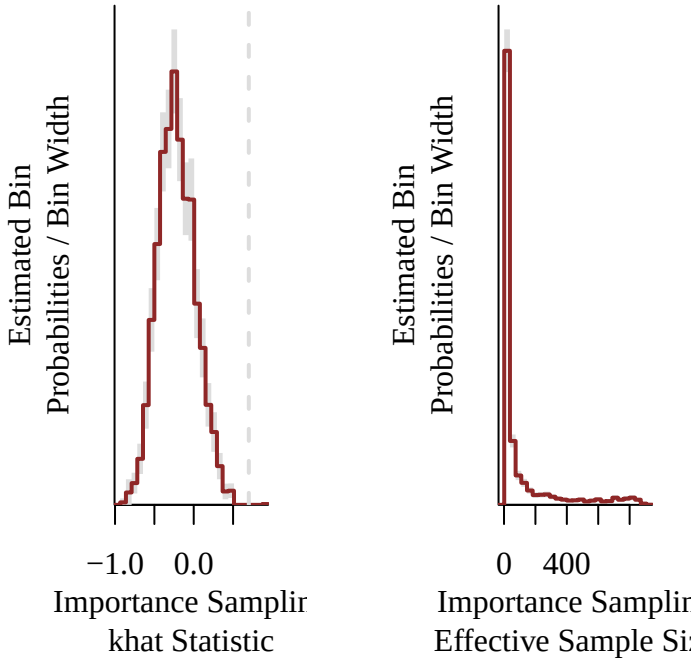
```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

weight_names <- grep('weights', names(samples), value=TRUE)
plot_histogram_pushforward(samples, weight_names, 0, 25, 1,
                           "Importance Sampling\nWeights")
```

```
Warning in plot_histogram_pushforward(samples, weight_names, 0, 25, 1,
"Importance Sampling\nWeights"): 7968 values (0.2%) fell above the histogram
binning.
```



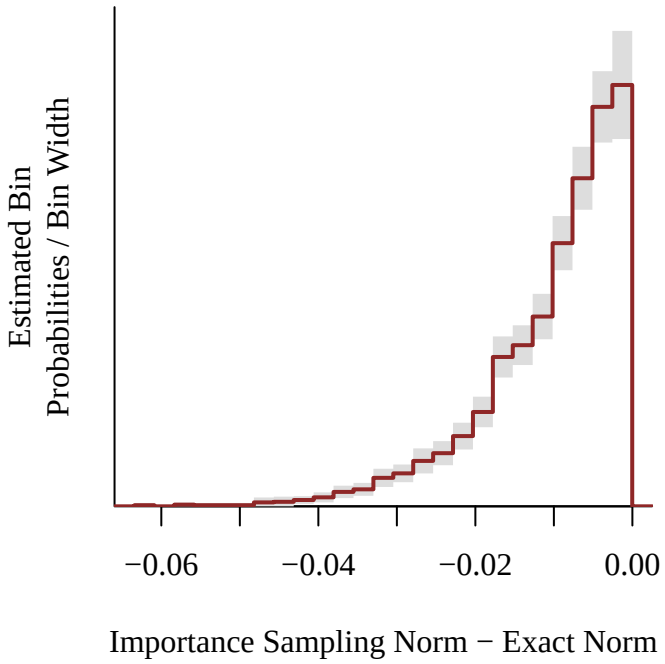
This importance weight behavior is not uncommon in importance sampling. The precision of the resulting importance sampling weights will depend on just how heavy that tail is. Fortunately the $\hat{k}$-statistic suggests that importance sampling estimation will be stable. That said the wide distribution of importance sampling effective sample sizes *also* suggests that the estimator precision varies strongly across normalization integral evaluations.

```
par(mfrow=c(1, 2), mar = c(5, 5, 2, 1))

util$plot_expectand_pushforward(samples[['khat']], 25,
                                'Importance Sampling\nkhat Statistic')
abline(v=0.7, col="#DDDDDD", lwd=2, lty=2)

util$plot_expectand_pushforward(samples[['ISESS']], 25,
                                'Importance Sampling\nEffective Sample Size')
```



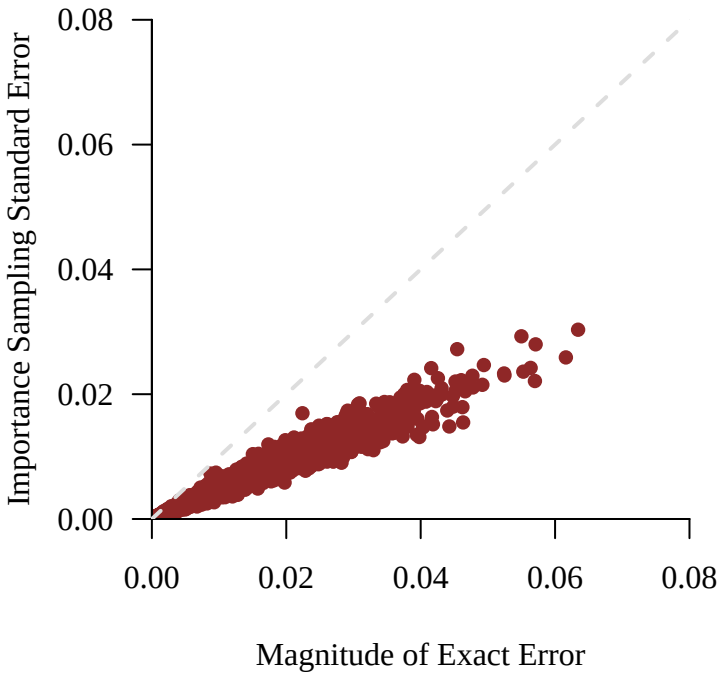
This suspicion is corroborated in the distribution of the exact errors which peaks at small values but exhibits a long tail of larger values. The importance sampling estimates are most precise for those latent probability distribution configurations near the true latent probability distribution, but that precision decays rapidly as we explore the posterior tails.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_expectand_pushforward(samples[['norm_error']], 25,
                                'Importance Sampling Norm - Exact Norm')
```



Now that the importance weights are better behaved we can well-approximate the exact error with the importance sampling standard error.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

plot(abs(c(samples[['norm_error']], recursive=TRUE)),
     c(samples[['ISSE']], recursive=TRUE),
     col=c_dark, pch=16,
     xlim=c(0, 0.08), xlab="Magnitude of Exact Error",
     ylim=c(0, 0.08), ylab="Importance Sampling Standard Error")
lines(c(0, 1), c(0, 1), col="#DDDDDD", lwd=2, lty=2)
```



The retrodictive performance appears to be consistent with that from the exact model.

```
par(mfrow=c(2, 3), mar = c(5, 4, 2, 1))

lower <- c(0.25, -2, -10, -3, -1)
upper <- c(1, 3, 4, 6, 3)
delta <- c(0.05, 0.25, 0.75, 0.5, 0.25)
for (k in 1:data$K) {
  pred_names <- grep(paste0('y_pred\\[[0-9]*,', k),
                     names(samples), value=TRUE)
  hist_retro(data$y[,k], samples, pred_names, lower[k], upper[k], delta[k],
             xlab=paste0('y[', k, ']'))
}
```

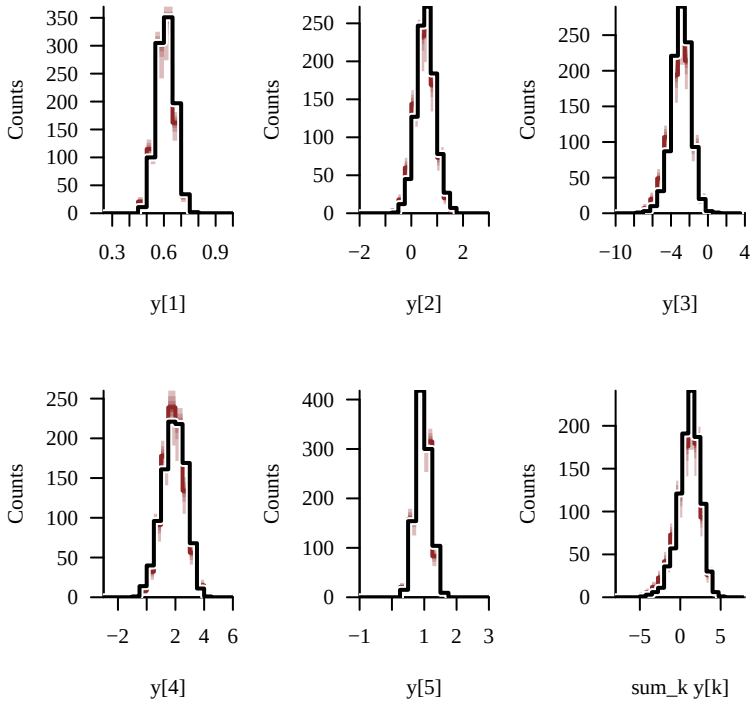
```
Warning in hist_retro(data$y[, k], samples, pred_names, lower[k], upper[k], :
129 predictive values (0.0%) fell below the histogram binning.
```

```
Warning in hist_retro(data$y[, k], samples, pred_names, lower[k], upper[k], : 1
predictive values (0.0%) fell above the histogram binning.
```

```
sum_y <- sapply(1:N, function(n) sum(data$y[n,]))
```

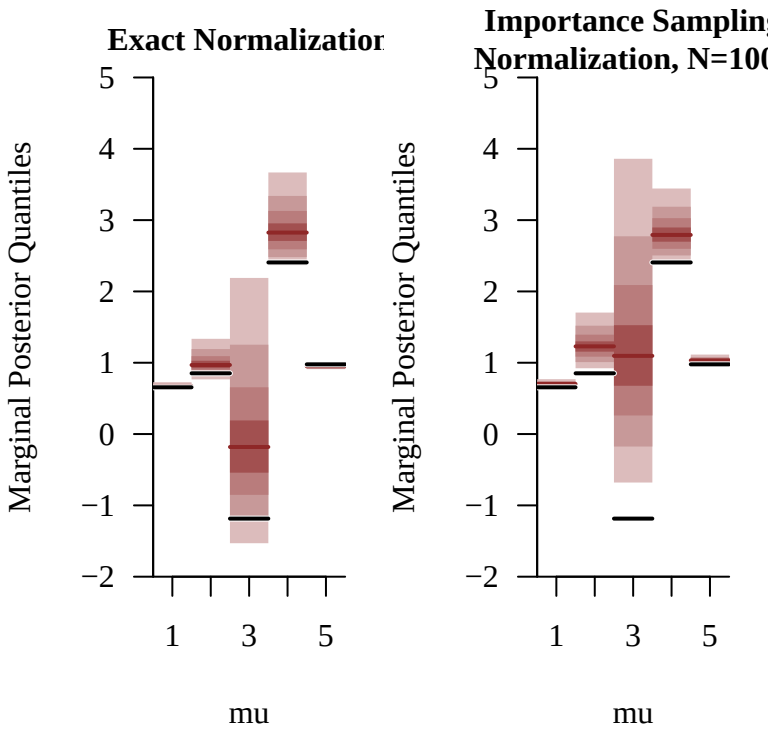
```
names <- c()
sum_samples <- list()
for (n in 1:data$N) {
  name <- paste0('sum_y[', n, ']')
  names <- c(names, name)

  summands <- lapply(1:data$K,
                     function(k) samples[[paste0('y_pred[', n, ',', k, ']')]])
  sum_samples[[name]] <- Reduce("+", summands)
}

hist_retro(sum_y, sum_samples, names, -8, 8, 0.75, xlab="sum_k y[k]")
```

```
Warning in hist_retro(sum_y, sum_samples, names, -8, 8, 0.75, xlab = "sum_k
y[k]"): 270 predictive values (0.0%) fell below the histogram binning.
```

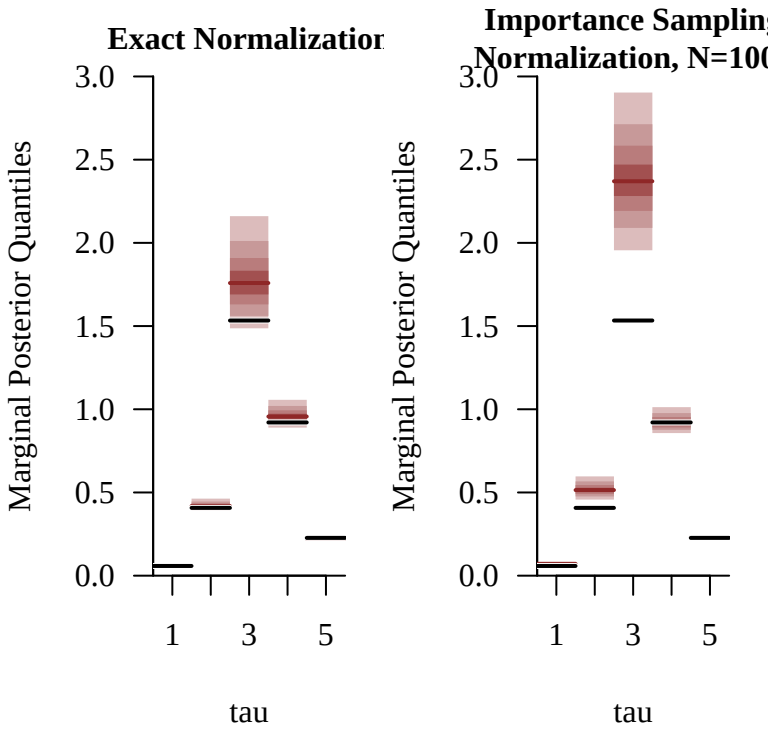


Sadly even in this idealized circumstance the importance sampling estimation of the normalization integral evaluations is not sufficiently precise to ensure an faithful recovery of the exact posterior distribution. Instead we recover an approximate posterior distribution that is substantially biased away from the exact posterior distribution, and in some cases underestimates uncertainties.

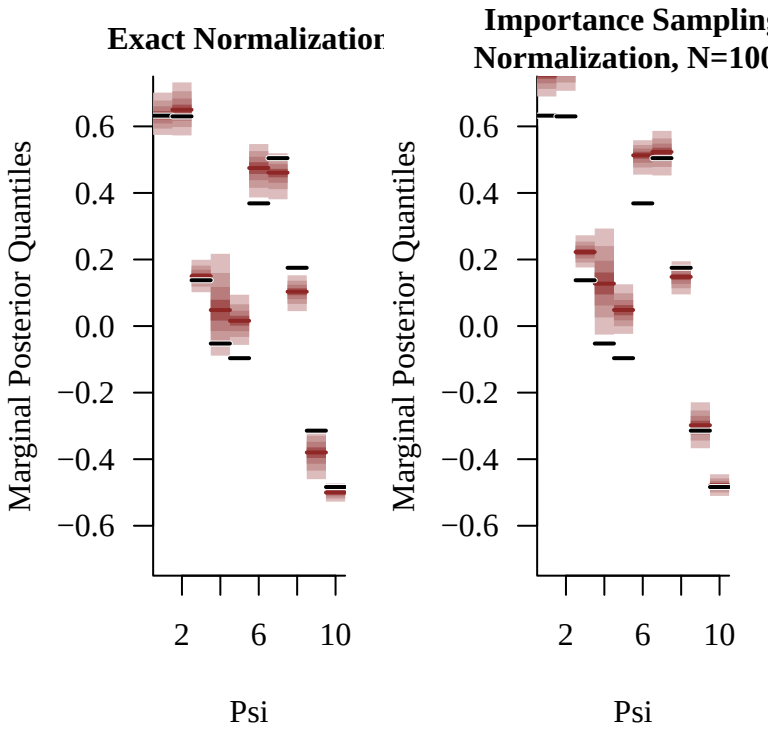
```
par(mfrow=c(1, 2))

exact_samples <- util$extract_expectands(fit_exact)

mu_names <- sapply(1:data$K, function(k) paste0('mu[', k, ']'))
plot_disc_pushforward_quantiles(exact_samples, mu_names, "mu",
                                baselines=mu, display_ylim=c(-2, 5),
                                main="Exact Normalization")
plot_disc_pushforward_quantiles(samples, mu_names, "mu",
                                baselines=mu, display_ylim=c(-2, 5),
                                main="Importance Sampling\nNormalization, N=1000")
```




```
par(mfrow=c(1, 2))
tau_names <- sapply(1:data$K, function(k) paste0('tau[', k, ']'))
plot_disc_pushforward_quantiles(exact_samples, tau_names, "tau",
                                baselines=tau, display_ylim=c(0, 3),
                                main="Exact Normalization")
plot_disc_pushforward_quantiles(samples, tau_names, "tau",
                                baselines=tau, display_ylim=c(0, 3),
                                main="Importance Sampling\nNormalization, N=1000")
```



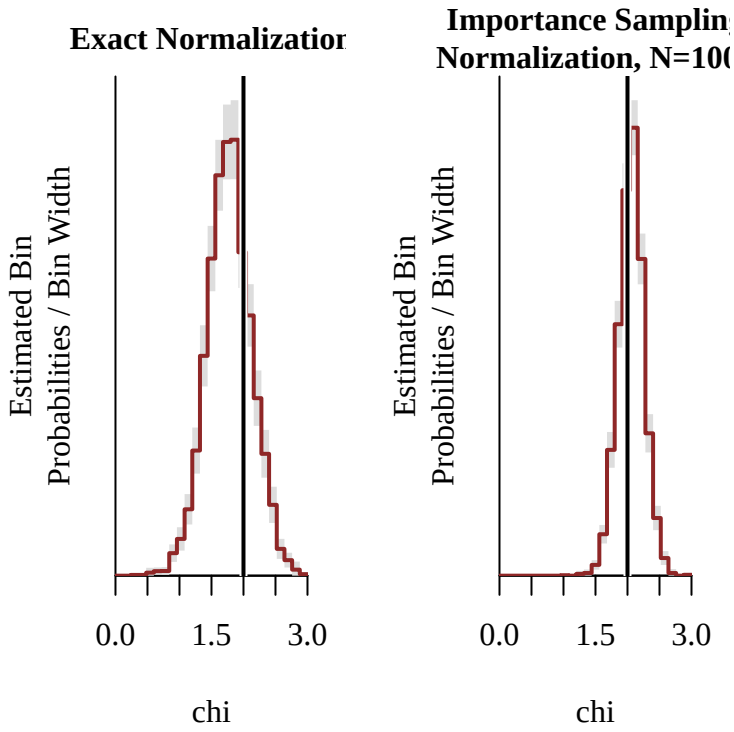

```
par(mfrow=c(1, 2))
Psi_lt <- c(sapply(2:K, function(k)
  sapply(1:(k - 1), function(kk) Psi[k, kk])),
  recursive=TRUE)
Psi_lt_names <- c(sapply(2:K, function(k)
  sapply(1:(k - 1), function(kk)
    paste0('Psi[', k, ',', kk, ']'))),
  recursive=TRUE)
plot_disc_pushforward_quantiles(exact_samples, Psi_lt_names, "Psi",
                                baselines=Psi_lt, display_ylim=c(-0.75, 0.75),
                                main="Exact Normalization")
plot_disc_pushforward_quantiles(samples, Psi_lt_names, "Psi",
                                baselines=Psi_lt, display_ylim=c(-0.75, 0.75),
                                main="Importance Sampling\nNormalization, N=1000")
```




```
par(mfrow=c(1, 2))
util$plot_expectand_pushforward(exact_samples[['chi']], 25, 'chi',
                                flim=c(0, 3), baseline=chi,
                                main="Exact Normalization")
```

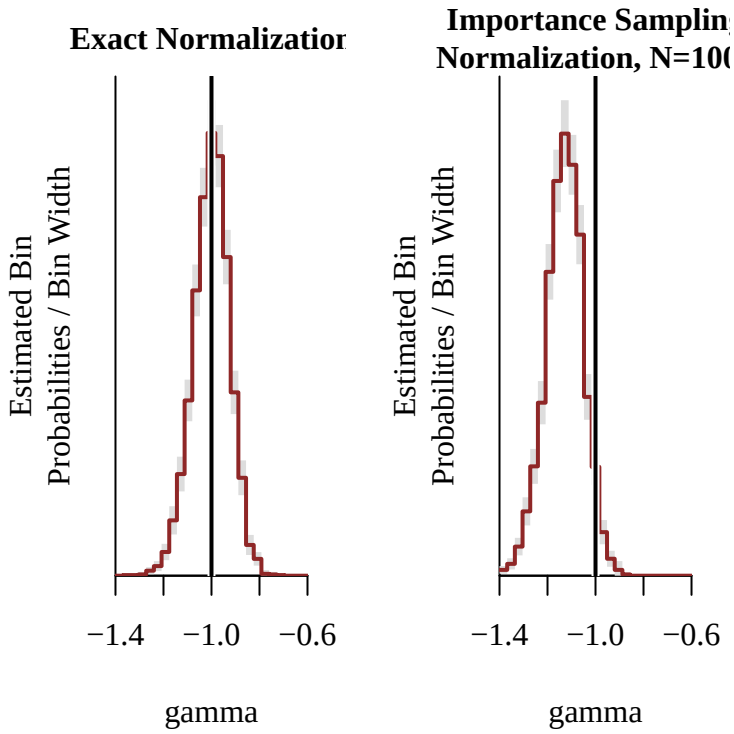
```
Warning in util$plot_expectand_pushforward(exact_samples[["chi"]], 25, "chi", :
3 posterior samples (0.1%) fell above the histogram binning.
```

```
util$plot_expectand_pushforward(samples[['chi']], 25, 'chi',
                                flim=c(0, 3), baseline=chi,
                                main="Importance Sampling\nNormalization, N=1000")
```




```
par(mfrow=c(1, 2))
util$plot_expectand_pushforward(exact_samples[['gamma']], 25, 'gamma',
                                flim=c(-1.4, -0.6), baseline=gamma,
                                main="Exact Normalization")
util$plot_expectand_pushforward(samples[['gamma']], 25, 'gamma',
                                flim=c(-1.4, -0.6), baseline=gamma,
                                main="Importance Sampling\nNormalization, N=1000")
```

```
Warning in util$plot_expectand_pushforward(samples[["gamma"]], 25, "gamma", : 7
posterior samples (0.2%) fell below the histogram binning.
```



Because importance sampling estimation is actually working now, however, we should be able to resolve this underestimation by increasing the size of the importance sampling ensemble. That said the ensemble size we need to ensure faithful posterior inferences may not be within our computational budget.

Like so many methods importance sampling is just really hard to implement well in high-dimensional spaces.

3.5 Further Directions

Readers interested in exploring selection models further have many interesting directions in which they can expand these exercises.

For example one might investigate what happens when the number of rejections are incorporated into the probabilistic selection exercises, similar to how they were incorporated into the deterministic selection model in Section 3.2. Perhaps this additional information can resolve the sign of γ without having to impose a hard constraint.

Similarly one could introduce calibration data from an auxiliary, but known, latent probability distribution. Incorporating this new data generating process into the existing models should help to inform the configuration of the selection function, allowing the original data to better inform the configuration of the unknown latent probability distribution.

More adversarially one could explore how the exact and approximate models perform as the number of observations are decreased or as the selection function is shifted away from the latent probability distribution. These more difficult scenarios will challenge both the normalization integral estimation strategies and Hamiltonian Monte Carlo.

4 Conclusion

Selection is a common feature of many data generating processes that arise in practical applications. Unfortunately *modeling* selection, and deriving inferences from those models, is not trivial.

Accurately estimating the normalization integral is a challenging hurdle for practical implementations to overcome, especially when we have to deal with strong inferential degeneracies between the behavior of the latent probability distribution and the selection function. In some circumstances we can take advantage of robust methods, such as numerical quadrature in low-dimensional problems and Monte Carlo estimation when only the selection function is unknown, but more general methods such as importance sampling have to be very carefully adapted to the particular problem at hand.

Ultimately successfully learning from data corrupted by selection effects requires careful prior modeling, experimental design, and computational methodology.

Appendix: Integral Calculations

Per tradition I've sequestered the more gory integral calculations that are used in the exercises into this appendix for safe keeping.

Appendix A: Univariate Calculations

Let

$$\Phi(x) = \int_{-\infty}^{x} \mathrm{d}x' \, \mathrm{normal}(x' \mid 0, 1)$$

denote the standard normal cumulative distribution function. The expectation value of a shifted and scaled Φ with respect to a probability distribution specified by a normal probability density function admits a closed-form solution (Geller and Ng 1971; Owen 1980),

$$\int_{-\infty}^{+\infty} \mathrm{d}x \, \mathrm{normal}(x \mid 0, 1) \, \Phi(a + b \cdot x) = \Phi\left(\frac{a}{\sqrt{1 + b^2}}\right).$$

This result then allows us to compute the normalization integral for the selection model defined by the latent probability density function

$$p(y) = \text{normal}(y \mid \mu, \tau)$$

and selection function

$$S(y) = \Phi(\gamma\,(y - \chi)).$$

We begin with the definition of the normalization integral,

$$\begin{aligned} Z(\mu, \tau, \chi, \gamma) &= \int_{-\infty}^{+\infty} \mathrm{d}y\, p(y \mid \mu, \tau)\, S(y; \chi, \gamma) \\ &= \int_{-\infty}^{+\infty} \mathrm{d}y\, \text{normal}(y \mid \mu, \tau)\, \Phi(\gamma\,(y - \chi)). \end{aligned}$$

Making the change of variables to

$$u = \frac{y - \mu}{\tau}$$

then gives

$$\begin{aligned} Z(\mu, \tau, \chi, \gamma) &= \int_{-\infty}^{+\infty} \mathrm{d}y\, \text{normal}(y \mid \mu, \tau)\, \Phi(\gamma\,(y - \chi)) \\ &= \int_{-\infty}^{+\infty} \mathrm{d}u\, \left[\tau\, \text{normal}(\mu + \tau\, u \mid \mu, \tau)\right] \Phi(\gamma\,(\mu + \tau\, u - \chi)) \\ &= \int_{-\infty}^{+\infty} \mathrm{d}u\, \left[\text{normal}(u \mid 0, 1) \qquad\quad\right] \Phi(\gamma\,(\mu + \tau\, u - \chi)) \\ &= \int_{-\infty}^{+\infty} \mathrm{d}u\, \text{normal}(u \mid 0, 1)\, \Phi(\gamma\,(\mu - \chi) + (\gamma\,\tau)\, u)) \\ &= \int_{-\infty}^{+\infty} \mathrm{d}u\, \text{normal}(u \mid 0, 1)\, \Phi(a + b\, u)) \\ &= \Phi\left(\frac{a}{\sqrt{1 + b^2}}\right), \end{aligned}$$

where

$$\begin{aligned} a &= \gamma\,(\mu - \chi) \\ b &= \gamma\,\tau. \end{aligned}$$

Substituting the values of the intermediate variables a and b finally gives

$$\begin{aligned} Z(\mu, \tau, \chi, \gamma) &= \int_{-\infty}^{+\infty} \mathrm{d}u\, \text{normal}(u \mid 0, 1)\, \Phi(\gamma\,(\mu - \chi) + (\gamma\,\tau)\, u)) \\ &= \Phi\left(\frac{\gamma\,(\mu - \chi)}{\sqrt{1 + (\gamma\,\tau)^2}}\right). \end{aligned}$$

Consequently the observed probability density function is given by

$$\begin{aligned} p(y \mid \mu, \tau, \chi, \gamma, z = 1) &= \frac{p(y; \mu, \tau)\, S(y; \chi, \gamma)}{Z(\mu, \tau, \chi, \gamma)} \\ &= \frac{\text{normal}(y \mid \mu, \tau)\, \Phi(\gamma\,(y - \chi))}{\Phi\left(\frac{\gamma\,(\mu - \chi)}{\sqrt{1 + (\gamma\,\tau)^2}}\right)}. \end{aligned}$$

Appendix B: Multivariate Calculations

Our goal is to compute the normalizing integral of the selection model where the selection function depends on only the sum of the component variables,

$$Z = \int \mathrm{d}y_1 \cdot \ldots \cdot \mathrm{d}y_K\, p(y_1, \ldots, y_K)\, S\left(\sum_{k=1}^{K} y_k\right).$$

Consider a change of variables where the first component is mapped into the sum while the remaining components map are unchanged,

$$\begin{aligned} z_1 &= \sum_{k=1}^{K} y_k \\ z_2 &= y_2 \\ \ldots &= \ldots \\ z_K &= y_K. \end{aligned}$$

Note that the Jacobian determinant of this transformation is just 1.

Under this change of variables the normalization integral becomes

$$\begin{aligned} Z &= \int \mathrm{d}y_1 \cdot \ldots \cdot \mathrm{d}y_K\, p(y_1, \ldots, y_K)\, S\left(\sum_{k=1}^{K} y_k\right) \\ &= \int \mathrm{d}z_1 \cdot \ldots \cdot \mathrm{d}z_K\, p(z_1, \ldots, z_K)\, S\left(z_1\right). \end{aligned}$$

If we can decompose the latent probability density function into

$$p(y_1, \ldots, y_K) = p(z_2, \ldots, z_K \mid z_1)\, p(z_1)$$

then this becomes

$$\begin{aligned}
Z &= \int \mathrm{d}z_1 \cdot \ldots \cdot \mathrm{d}z_K \, p(z_1, \ldots, z_K) \, S(z_1) \\
&= \int \mathrm{d}z_1 \cdot \ldots \cdot \mathrm{d}z_K \, p(z_2, \ldots, z_K \mid z_1) \, p(z_1) \, S(z_1) \\
&= \int \mathrm{d}z_1 \left[\int \mathrm{d}z_2 \ldots \cdot \mathrm{d}z_K \, p(z_2, \ldots, z_K \mid z_1) \right] p(z_1) \, S(z_1) \\
&= \int \mathrm{d}z_1 \left[\qquad\qquad 1 \qquad\qquad \right] p(z_1) \, S(z_1) \\
&= \int \mathrm{d}z_1 \, p(z_1) \, S(z_1) \, .
\end{aligned}$$

In other words if we can isolate the pushforward probability density function $p(z_1) = p\left(\sum_{k=1}^{K} z_k\right)$ then we can reduce the normalization integral into a one-dimensional problem.

Conveniently we can do this when the latent probability density function is from the multivariate normal family,

$$\begin{aligned}
p(y_1, \ldots, y_K) &= \text{multinormal}(\mathbf{y} \mid \mu, \Sigma) \\
&= \sqrt{\frac{1}{(2\pi)^K \det \Sigma}} \exp\left(-\frac{1}{2} (\mathbf{y} - \mu)^T \cdot \Sigma^{-1} \cdot (\mathbf{y} - \mu) \right) \\
&= \sqrt{\frac{\det \Lambda}{(2\pi)^K}} \exp\left(-\frac{1}{2} (\mathbf{y} - \mu)^T \cdot \Lambda \cdot (\mathbf{y} - \mu) \right) \\
&= \sqrt{\frac{\det \Lambda}{(2\pi)^K}} \exp\left(-\frac{1}{2} Q \right) .
\end{aligned}$$

To achieve this we begin by decomposing the full quadratic form Q into a quadratic form for

only $y_2, \ldots, y_K$ and a collection of residual terms,

$$\begin{aligned}
Q = \quad & (\mathbf{y}-\mu)^T \cdot \Lambda \cdot (\mathbf{y}-\mu) \\
= \quad & \sum_{i=1}^{K}\sum_{j=1}^{K}(z_i-\mu_i)\,\Lambda_{ij}\,(y_j-\mu_j) \\
= \quad & \sum_{i=2}^{K}\sum_{j=1}^{K}(z_i-\mu_i)\,\Lambda_{ij}\,(y_j-\mu_j) \\
& + (y_1-\mu_1)\left[\sum_{j=1}^{K}\Lambda_{1j}\,(y_j-\mu_j)\right] \\
= \quad & \sum_{i=2}^{K}\sum_{j=2}^{K}(z_i-\mu_i)\,\Lambda_{ij}\,(y_j-\mu_j) \\
& + (y_1-\mu_1)\left[\sum_{j=1}^{K}\Lambda_{1j}\,(y_j-\mu_j)\right] \\
& + \left[\sum_{i=2}^{K}(z_i-\mu_i)\,\Lambda_{i1}\right](y_1-\mu_1) \\
= \quad & \sum_{i=2}^{K}\sum_{j=2}^{K}(z_i-\mu_i)\,\Lambda_{ij}\,(y_j-\mu_j) \\
& + (y_1-\mu_1)\left[\sum_{j=2}^{K}\Lambda_{1j}\,(y_j-\mu_j)\right] \\
& + \left[\sum_{i=2}^{K}(z_i-\mu_i)\,\Lambda_{i1}\right](y_1-\mu_1) \\
& + (y_1-\mu_1)\,\Lambda_{11}\,(y_1-\mu_1) \\
= \quad & \sum_{i=2}^{K}\sum_{j=2}^{K}(z_i-\mu_i)\,\Lambda_{ij}\,(z_j-\mu_j) \\
& + (y_1-\mu_1)\left[\sum_{j=2}^{K}\Lambda_{1j}\,(z_j-\mu_j)\right] \\
& + \left[\sum_{i=2}^{K}(z_i-\mu_i)\,\Lambda_{i1}\right](y_1-\mu_1) \\
& + (y_1-\mu_1)\,\Lambda_{11}\,(y_1-\mu_1).
\end{aligned}$$

To remove the dependence on y_1 we introduce the variable

$$\mu_S = \sum_{i=1}^{K}\mu_i$$

which allows us to write

$$\begin{aligned} y_1 - \mu_1 &= \left(\sum_{i=1}^{K} z_i - \sum_{i=2}^{K} z_i \right) - \left(\sum_{i=1}^{K} \mu_i - \sum_{i=2}^{K} \mu_i \right) \\ &= \left(z_1 - \sum_{i=2}^{K} z_i \right) - \left(\mu_S - \sum_{i=2}^{K} \mu_i \right) \\ &= (z_1 - \mu_S) - \sum_{i=2}^{K} (z_i - \mu_i). \end{aligned}$$

Substituting this into the decomposition gives

$$\begin{aligned} Q = \quad & \sum_{i=2}^{K} \sum_{j=2}^{K} (z_i - \mu_i) \, \Lambda_{ij} \, (z_j - \mu_j) \\ & + \left((z_1 - \mu_S) - \sum_{i=2}^{K} (z_i - \mu_i) \right) \left[\sum_{j=2}^{K} \Lambda_{1j} \, (z_j - \mu_j) \right] \\ & + \left[\sum_{i=2}^{K} (z_i - \mu_i) \, \Lambda_{i1} \right] \left((z_1 - \mu_S) - \sum_{j=2}^{K} (z_j - \mu_j) \right) \\ & + \left((z_1 - \mu_S) - \sum_{i=2}^{K} (z_i - \mu_i) \right) \Lambda_{11} \left((z_1 - \mu_S) - \sum_{j=2}^{K} (z_j - \mu_j) \right) \\ = \quad & \sum_{i=2}^{K} \sum_{j=2}^{K} (z_i - \mu_i) \, \Lambda_{ij} \, (z_j - \mu_j) \\ & + (z_1 - \mu_S) \left[\sum_{j=2}^{K} \Lambda_{1j} \, (z_j - \mu_j) \right] - \sum_{i=2}^{K} (z_i - \mu_i) \sum_{j=2}^{K} \Lambda_{1j} \, (z_j - \mu_j) \\ & + \left[\sum_{i=2}^{K} (z_i - \mu_i) \, \Lambda_{i1} \right] (z_1 - \mu_S) - \sum_{i=2}^{K} \Lambda_{i1} \, (z_j - \mu_j) \sum_{j=2}^{K} (z_i - \mu_i) \\ & + (z_1 - \mu_S) \, \Lambda_{11} \, (z_1 - \mu_S) \\ & - (z_1 - \mu_S) \, \Lambda_{11} \left[\sum_{j=2}^{K} (z_j - \mu_j) \right] - \left[\sum_{i=2}^{K} (z_i - \mu_i) \right] \Lambda_{11} \, (z_1 - \mu_S) \\ & + \sum_{i=2}^{K} \sum_{j=2}^{K} (z_i - \mu_i) \, \Lambda_{11} \, (z_j - \mu_j). \end{aligned}$$

Collecting common terms together yields

$$\begin{aligned}
Q = & \sum_{i=2}^{K}\sum_{j=2}^{K}(z_i-\mu_i)\Big[\Lambda_{ij}-\Lambda_{1j}-\Lambda_{i1}+\Lambda_{11}\Big](z_j-\mu_j) \\
& + (z_1-\mu_S)\left[\sum_{j=2}^{K}\Lambda_{1j}(z_j-\mu_j)\right] - (z_1-\mu_S)\Lambda_{11}\left[\sum_{j=2}^{K}(z_j-\mu_j)\right] \\
& + \left[\sum_{i=2}^{K}(z_i-\mu_i)\Lambda_{i1}\right](z_1-\mu_S) - \left[\sum_{i=2}^{K}(z_i-\mu_i)\right]\Lambda_{11}(z_1-\mu_S) \\
& + (z_1-\mu_S)\Lambda_{11}(z_1-\mu_S) \\
= & \sum_{i=2}^{K}\sum_{j=2}^{K}(z_i-\mu_i)\Big[\Lambda_{ij}-\Lambda_{1j}-\Lambda_{i1}+\Lambda_{11}\Big](z_j-\mu_j) \\
& + 2(z_1-\mu_S)\left[\sum_{j=2}^{K}\Lambda_{1j}(z_j-\mu_j)\right] - 2(z_1-\mu_S)\Lambda_{11}\left[\sum_{j=2}^{K}(z_j-\mu_j)\right] \\
& + (z_1-\mu_S)\Lambda_{11}(z_1-\mu_S) \\
= & \sum_{i=2}^{K}\sum_{j=2}^{K}(z_i-\mu_i)\Big[\Lambda_{ij}-\Lambda_{1j}-\Lambda_{i1}+\Lambda_{11}\Big](z_j-\mu_j) \\
& + \sum_{j=2}^{K}2(z_1-\mu_S)\Big[\Lambda_{1j}-\Lambda_{11}\Big](z_j-\mu_j) \\
& + (z_1-\mu_S)\Lambda_{11}(z_1-\mu_S).
\end{aligned}$$

In order to write this in a more compact matrix form let's introduce the $(K-1)$-dimensional vector $\tilde{\mathbf{z}}$ with components

$$\tilde{z}_i = z_{i+1},$$

the $(K-1)$-dimensional vector ν with components

$$\nu_i = \mu_{i+1},$$

the $(K-1)$-dimensional vector α with components

$$\alpha_i = 2(z_1-\mu_S)\Big[\Lambda_{1,i+1}-\Lambda_{11}\Big],$$

and the $(K-1,K-1)$-dimensional matrix Γ with components

$$\Gamma_{ij} = \Lambda_{i+1,j+1}-\Lambda_{1,j+1}-\Lambda_{i+1,1}+\Lambda_{11}$$

so that the full quadratic form becomes

$$Q = (\tilde{\mathbf{z}}-\nu)^T\cdot\Gamma\cdot(\tilde{\mathbf{z}}-\nu) + \alpha^T\cdot(\tilde{\mathbf{z}}-\nu) + (z_1-\mu_S)\Lambda_{11}(z_1-\mu_S).$$

At this point we can complete the square with respect to $\tilde{\mathbf{z}}$,

$$Q = (\tilde{\mathbf{z}} - \nu - \mathbf{m})^T \cdot \Gamma \cdot (\tilde{\mathbf{z}} - \nu - \mathbf{m}) + R$$

where

$$\mathbf{m} = -\frac{1}{2}\Gamma^{-1} \cdot \alpha$$

and

$$R = (z_1 - \mu_S)\,\Lambda_{11}\,(z_1 - \mu_S) - \frac{1}{4}\alpha^T \cdot \Gamma^{-1} \cdot \alpha.$$

Defining the vector $\mathbf{l}$ with components

$$\lambda_j = \Lambda_{1,j+1} - \Lambda_{11}$$

the second term simplifies to

$$\begin{aligned} R &= (z_1 - \mu_S)\,\Lambda_{11}\,(z_1 - \mu_S) - \frac{1}{4}2\,(z_1 - \mu_S)\,\mathbf{l}^T \cdot \Gamma^{-1} \cdot 2\,(z_1 - \mu_S)\,\mathbf{l} \\ &= (z_1 - \mu_S)\,\Lambda_{11}\,(z_1 - \mu_S) - (z_1 - \mu_S)\,\mathbf{l}^T \cdot \Gamma^{-1} \cdot \mathbf{l}\,(z_1 - \mu_S) \\ &= (z_1 - \mu_S)\Big[\Lambda_{11} - \mathbf{l}^T \cdot \Gamma^{-1} \cdot \mathbf{l}\Big](z_1 - \mu_S). \end{aligned}$$

Exponentiating this quadratic form finally gives

$$\begin{aligned} p(y_1, \ldots, y_K) &\propto \exp\left(-\frac{1}{2}Q\right) \\ &\propto \exp\Bigg(-\frac{1}{2}(\tilde{\mathbf{z}} - \nu - \mathbf{m})^T \cdot \Gamma \cdot (\tilde{\mathbf{z}} - \nu - \mathbf{m}) \\ &\qquad -\frac{1}{2}(z_1 - \mu_S)\Big[\Lambda_{11} - \mathbf{l}^T \cdot \Gamma^{-1} \cdot \mathbf{l}\Big](z_1 - \mu_S)\Bigg) \\ &\propto \exp\left(-\frac{1}{2}(\tilde{\mathbf{z}} - \nu - \mathbf{m})^T \cdot \Gamma \cdot (\tilde{\mathbf{z}} - \nu - \mathbf{m})\right) \\ &\quad \cdot \exp\left(-\frac{1}{2}(z_1 - \mu_S)\Big[\Lambda_{11} - \mathbf{l}^T \cdot \Gamma^{-1} \cdot \mathbf{l}\Big](z_1 - \mu_S)\right) \\ &\propto \text{multinormal}\left(z_2, \ldots, z_K \mid \nu + \mathbf{m}, \Gamma^{-1}\right) \\ &\quad \cdot \text{normal}\,(z_1 \mid \mu_S, \tau_S) \end{aligned}$$

where

$$\tau_S = \frac{1}{\sqrt{\Lambda_{11} - \mathbf{l}^T \cdot \Gamma^{-1} \cdot \mathbf{l}}}.$$

In other words the marginal probability density function for the sum of the components is given by normal probability density function with location parameter

$$\mu_S = \sum_{k=1}^{K} \mu_k$$

and scale parameter

$$\tau_S = \frac{1}{\sqrt{\Lambda_{11} - \mathbf{l}^T \cdot \Gamma^{-1} \cdot \mathbf{l}}}.$$

Technically this derivation is consistent regardless of which component we replace by the sum of all components. Consequently we can use any index k to compute the marginal scale parameter,

$$\begin{aligned}
\Gamma_{ij} &= \Lambda_{i+1,j+1} - \Lambda_{k,j+1} - \Lambda_{i+1,k} + \Lambda_{kk} \\
\lambda_j &= \Lambda_{k,j+1} - \Lambda_{kk} \\
\tau_S &= \frac{1}{\sqrt{\Lambda_{kk} - \mathbf{l}^T \cdot \Gamma^{-1} \cdot \mathbf{l}}}.
\end{aligned}$$

In matrix form this becomes

$$\begin{aligned}
\Gamma &= \Lambda - \lambda_k \cdot \mathbf{1}^T - \mathbf{1} \cdot \lambda_k^T + \Lambda_{kk}\, \mathbf{1} \cdot \mathbf{1}^T \\
\mathbf{l} &= \lambda_k - \Lambda_{kk}\, \mathbf{1} \\
\tau_S &= \frac{1}{\sqrt{\Lambda_{kk} - \mathbf{l}^T \cdot \mathbf{A}^{-1} \cdot \mathbf{l}}}.
\end{aligned}$$

Note that these linear algebra operations, which require inverting a dense matrix, decomposing it, and then inverting a dense submatrix again, are prone to computational inefficiency and numerical instability when implemented in practice. There may very well be a more effective way to computing τ_S!

If the shape of the selection function is given by a normal cumulative distribution function then applying the result from Appendix A gives

$$\begin{aligned}
Z(\mu, \tau, \Psi, \chi, \gamma) &= \int \mathrm{d}z_1\, p(z_1 \mid \mu, \tau, \Psi)\, S\left(z_1; \chi, \gamma\right) \\
&= \int \mathrm{d}z_1\, \mathrm{normal}(z_1 \mid \mu_S, \tau_S)\, \Phi(\gamma\,(z_1 - \chi)) \\
&= \Phi\left(\frac{\gamma\,(\mu_S - \chi)}{\sqrt{1 + (\gamma\,\tau_S)^2}}\right).
\end{aligned}$$

Acknowledgements

The importance sampling estimator approach was first introduced to me by Will Farr. I thank Alexander Noll for helpful comments.

Adam Fleischhacker, Adriano Yoshino, Alessandro Varacca, Alexander Noll, Alexander Petrov, Alexander Rosteck, Andrea Serafino, Andrew Mascioli, Andrew Rouillard, Andrew Vigotsky, Ara Winter, Austin Rochford, Avraham Adler, Ben Matthews, Ben Swallow, Benoit Essiambre, Bradley Kolb, Brandon Liu, Brendan Galdo, Brynjolfur Gauti Jónsson, Cameron Smith, Canaan Breiss, Cat Shark, Charles Naylor, Charles Shaw, Chase Dwelle, Chris Jones, Christopher Mehrvarzi, Colin Carroll, Colin McAuliffe, Damien Mannion, dan mackinlay, Dan W Joyce, Dan Waxman, Dan Weitzenfeld, Daniel Edward Marthaler, Darshan Pandit, Darthmaluus , Dav Pe, David Galley, David Wurtz, Denis Vlašiček, Doug Rivers, Dr. Jobo, Dr. Omri Har Shemesh, Dylan Maher, Ed Cashin, Edgar Merkle, Eric LaMotte, Ero Carrera, Eugene O'Friel, Felipe González, Fergus Chadwick, Finn Lindgren, Florian Wellmann, Geoff Rollins, Guido Biele, Håkan Johansson, Hamed Bastan-Hagh, Haonan Zhu, Hauke Burde, Hector Munoz, Henri Wallen, hs, Hugo Botha, Ian, Ian Costley, idontgetoutmuch, Ignacio Vera, Ilaria Prosdocimi, Isaac Vock, J, J Michael Burgess, jacob pine, Jair Andrade, James C, James Hodgson, James Wade, Janek Berger, Jason Martin, Jason Pekos, Jason Wong, jd, Jeff Burnett, Jeff Dotson, Jeff Helzner, Jeffrey Erlich, Jesse Wolfhagen, Jessica Graves, Joe Wagner, John Flournoy, Jonathan H. Morgan, Jonathon Vallejo, Joran Jongerling, JU, Justin Bois, Kádár András, Karim Naguib, Karim Osman, Kejia Shi, Kristian Gårdhus Wichmann, Lars Barquist, lizzie , LOU ODETTE, Luís F, Marcel Lüthi, Marek Kwiatkowski, Mark Donoghoe, Markus P., Márton Vaitkus, Matt Moores, Matthew, Matthew Kay, Matthieu LEROY, Mattia Arsendi, Maurits van der Meer, Michael Colaresi, Michael DeWitt, Michael Dillon, Michael Lerner, Mick Cooney, N Sanders, N.S. , Name, Nathaniel Burbank, Nic Fishman, Nicholas Clark, Nicholas Cowie, Nick S, Octavio Medina, Ole Rogeberg, Oliver Crook, Olivier Ma, Patrick Kelley, Patrick Boehnke, Pau Pereira Batlle, Peter Johnson, Pieter van den Berg , ptr, Ramiro Barrantes Reynolds, Raúl Peralta Lozada, Ravin Kumar, Rémi , Rex Ha, Riccardo Fusaroli, Richard Nerland, Robert Frost, Robert Goldman, Robert kohn, Robin Taylor, Ryan Grossman, S Hong, Saleem Huda, Sean Wilson, Sergiy Protsiv, Seth Axen, shira, Simon Duane, Simon Lilburn, sssz, Stan_user, Stephen Lienhard, Stew Watts, Stone Chen, Susan Holmes, Svilup, Tao Ye, Tate Tunstall, Tatsuo Okubo, Teresa Ortiz, Theodore Dasher, Thomas Kealy, Thomas Siegert, Thomas Vladeck, Tim Radtke, Tobychev , Tom McEwen, Tomáš Frýda, Tony Wuersch, Virginia Fisher, Vladimir Markov, Wil Yegelwel, Will Farr, woejozney, Xianda Sun, yolhaj , yureq , Zach A, Zad Rafi, and Zhengchen Cai.

License

Betancourt, Michael. 2015. "The Fundamental Incompatibility of Scalable Hamiltonian Monte Carlo and Naive Data Subsampling." In *Proceedings of the 32nd International Conference on Machine Learning*, edited by Francis Bach and David Blei, 37:533–40. Proceedings of Machine Learning Research. Lille, France: PMLR.

Geller, Murray, and Edward W. Ng. 1971. "A Table of Integrals of the Error Function. II. Additions and Corrections." *Journal of Research of the National Bureau of Standards, Section B: Mathematical Sciences* 75B (3 and 4): 149.

Owen, D. B. 1980. "A Table of Normal Integrals." *Communications in Statistics - Simulation and Computation* 9 (4): 389–419.

Talts, Sean, Michael Betancourt, Daniel Simpson, Aki Vehtari, and Andrew Gelman. 2018. "Validating Bayesian Inference Algorithms with Simulation-Based Calibration," April.

Vehtari, Aki, Daniel Simpson, Andrew Gelman, Yuling Yao, and Jonah Gabry. 2015. "Pareto Smoothed Importance Sampling," July.

Stan
Program 1 `simu\_threshold.stan`

```
data {
  int<lower=1> N; // Number of observations
}

transformed data {
  real lambda = 4.75;    // Selection threshold
  real mu = 3;           // Location of latent probability density function
  real<lower=0> tau = 2; // Shape of latent probability density function
}

generated quantities {
  // Simulated data
  real y[N];
  real N_reject = 0;

  for (n in 1:N) {
    // Sample latent events until one survives the selection process
    while (1) {
      y[n] = normal_rng(mu, tau);
      if (y[n] <= lambda) {
        break;
      } else {
        N_reject += 1;
      }
    }
  }
}
```

Stan
Program 2 `fit\_threshold1.stan`

```
data {
  int<lower=1> N; // Number of observations
  real y[N];      // Observations
}

parameters {
  real mu;           // Location of latent probability density function
  real<lower=0> tau; // Shape of latent probability density function
}

model {
  // Prior model
  mu ~ normal(0, 5 / 2.32);     // ~ 99% prior mass between -5 and +5
  tau ~ normal(0, 5 / 2.57);    // ~ 99% prior mass between  0 and +5

  // Observational model
  target += normal_lpdf(y | mu, tau);
}

generated quantities {
  real y_pred[N]; // Posterior predictive data

  for (n in 1:N) {
    y_pred[n] = normal_rng(mu, tau);
  }
}
```

Stan
Program 3 `fit\_threshold2.stan`

```
data {
  int<lower=1> N; // Number of observations
  real y[N];      // Observations
}

transformed data {
  real lambda_lower_bound = max(y);
}

parameters {
  real<lower=lambda_lower_bound> lambda; // Selection threshold
  real mu;           // Location of latent probability density function
  real<lower=0> tau; // Shape of latent probability density function
}

model {
  real log_norm = normal_lcdf(lambda | mu, tau);

  // Prior model
  lambda ~ normal(5, 5 / 2.32); // ~ 99% prior mass between  0 and +10
  mu ~ normal(0, 5 / 2.32);     // ~ 99% prior mass between -5 and +5
  tau ~ normal(0, 5 / 2.57);    // ~ 99% prior mass between  0 and +5

  // Observational model
  for (n in 1:N) {
    target += normal_lpdf(y[n] | mu, tau) - log_norm;
  }
}

generated quantities {
  real y_pred[N]; // Posterior predictive data

  for (n in 1:N) {
    // Sample latent intensities until one survives the selection process
    // Use fixed iterations instead of while loop to avoid excessive trials
    // when the selection function and latent density function are not aligned
    for (m in 1:100) {
      real y_sample = normal_rng(mu, tau);
      if (y_sample <= lambda) {
        y_pred[n] = y_sample;
        break;
      }
    }
  }
}
```

Stan
Program 4 `fit\_threshold3.stan`

```
data {
  int<lower=1> N; // Number of observations
  real y[N];      // Observations

  int N_reject; // Number of rejected latent events
}

transformed data {
  real lambda_lower_bound = max(y);
}

parameters {
  real<lower=lambda_lower_bound> lambda; // Selection threshold
  real mu;           // Location of latent probability density function
  real<lower=0> tau; // Shape of latent probability density function
}

model {
  real log_norm = normal_lcdf(lambda | mu, tau);

  // Prior model
  lambda ~ normal(5, 5 / 2.32); // ~ 99% prior mass between  0 and +10
  mu ~ normal(0, 5 / 2.32);     // ~ 99% prior mass between -5 and +5
  tau ~ normal(0, 5 / 2.57);    // ~ 99% prior mass between  0 and +5

  // Observational model
  for (n in 1:N) {
    target += normal_lpdf(y[n] | mu, tau);
  }

  target += N_reject * log1m_exp(log_norm);
}

generated quantities {
  real y_pred[N];        // Posterior predictive data
  int N_reject_pred = 0; // Posterior predictive data

  for (n in 1:N) {
    // Sample latent intensities until one survives the selection process
    // Use fixed iterations instead of while loop to avoid excessive trials
    // when the selection function and latent density function are not aligned
    for (m in 1:100) {
      real y_sample = normal_rng(mu, tau);
      if (y_sample <= lambda) {
        y_pred[n] = y_sample;
        break;
      } else {
        N_reject_pred += 1;
      }
    }
```

Stan
Program 5 `simu\_selection\_uni.stan`

```
data {
  int<lower=1> N; // Number of observations
}

transformed data {
  real mu = -1;           // Location of latent probability density function
  real<lower=0> tau = 3; // Shape of latent probability density function
  real chi = 2;           // Location of selection function
  real gamma = 0.75;      // Shape of selection function
}

generated quantities {
  // Simulate filtered data
  real y[N];
  real N_reject = 0;

  for (n in 1:N) {
    // Sample latent points until one survives the selection process
    while (1) {
      y[n] = normal_rng(mu, tau);
      if ( bernoulli_rng( Phi(gamma * (y[n] - chi)) ) ) {
        break;
      } else {
        N_reject += 1;
      }
    }
  }
}
```

Stan
Program 6 `fit\_no\_selection\_uni.stan`

```
data {
  int<lower=1> N; // Number of observations
  real y[N];      // Observations
}

parameters {
  real mu;           // Location of latent probability density function
  real<lower=0> tau; // Shape of latent probability density function
}

model {
  // Prior model
  mu ~ normal(0, 5 / 2.32);   // ~ 99% prior mass between -5 and +5
  tau ~ normal(0, 5 / 2.57);  // ~ 99% prior mass between 0 and +5

  // Observational model
  for (n in 1:N) {
    target += normal_lpdf(y[n] | mu, tau);
  }
}

generated quantities {
  real y_pred[N]; // Posterior predictive data

  for (n in 1:N) {
    y_pred[n] = normal_rng(mu, tau);
  }
}
```

Stan
Program 7 `fit\_unknown\_selecton\_uni\_exact.stan`

```
data {
  int<lower=1> N; // Number of observations
  real y[N];      // Observations

  int<lower=1> N_grid; // Number of latent value grid points
  real y_grid[N_grid]; // Latent value grid points
}

transformed data {
  // Exact latent probability distribution configuration
  real mu = -1;          // Location
  real<lower=0> tau = 3; // Shape
}

parameters {
  real chi;   // Location of selection function
  real gamma; // Shape of selection function
}

model {
  // Filtered normalization
  real norm = Phi( gamma * (mu - chi) / sqrt(1 + square(gamma * tau)) );

  // Prior model
  chi ~ normal(0, 3 / 2.32);   // ~ 99% prior mass between -3 and +3
  gamma ~ normal(0, 3 / 2.32); // ~ 99% prior mass between -3 and +3

  // Observational model
  for (n in 1:N) {
    real log_select_prob = log( Phi(gamma * (y[n] - chi)) );
    real lpdf = normal_lpdf(y[n] | mu, tau);
    target += log_select_prob + lpdf - log(norm);
  }
}

generated quantities {
  real select_prob_grid[N_grid]; // Selection function values
  real y_pred[N];                // Posterior predictive data

  for (n in 1:N_grid)
    select_prob_grid[n] = Phi(gamma * (y_grid[n] - chi));

  for (n in 1:N) {
    // Sample latent intensities until one survives the selection
    // process.  Use fixed iterations instead of while loop to avoid
    // excessive trials when the selection function and latent density
    // function are not aligned.
    for (m in 1:100) {
      real y_sample = normal_rng(mu, tau);
      if ( bernoulli_rng( Phi(gamma * (y_sample - chi)) ) ) {
```

Stan
Program 8 `fit\_unknown\_selecton\_uni\_quad.stan`

```
functions {
  // Unnormalized observed probability density function
  real observed_updf(real x, real xc, real[] theta, real[] x_r, int[] x_i) {
    real chi = theta[1];
    real gamma = theta[2];
    real mu = x_r[1];
    real tau = x_r[2];

    return exp(normal_lpdf(x | mu, tau)) * Phi(gamma * (x - chi));
  }
}

data {
  int<lower=1> N; // Number of observations
  real y[N];      // Observations

  int<lower=1> N_grid; // Number of latent value grid points
  real y_grid[N_grid]; // Latent value grid points
}

transformed data {
  // Exact latent probability distribution configuration
  real mu = -1;          // Location
  real<lower=0> tau = 3; // Shape

  // Empty arrays for `integrate_1d` function
  real x_r[2] = {mu, tau};
  int x_i[0];
}

parameters {
  real chi;   // Location of selection function
  real gamma; // Shape of selection function
}

model {
  // Numerical quadrature estimator of normalization integral
  real theta[2] = {chi, gamma};
  real norm = integrate_1d(observed_updf,
                           negative_infinity(), positive_infinity(),
                           theta, x_r, x_i, 1e-8);

  // Prior model
  chi ~ normal(0, 3 / 2.32);   // ~ 99% prior mass between -3 and +3
  gamma ~ normal(0, 3 / 2.32); // ~ 99% prior mass between -3 and +3

  // Observational model
  for (n in 1:N) {
    real log_select_prob = log( Phi(gamma * (y[n] - chi)) );
    real lpdf = normal_lpdf(y[n] | mu, tau);
```

Stan
Program 9 `fit\_unknown\_selecton\_uni\_mc.stan`

```
functions {
  real compute_mc_norm(real[] y_MC, real chi, real gamma) {
    int N_MC = size(y_MC);
    real select_probs[N_MC];
    for (n in 1:N_MC)
      select_probs[n] = Phi(gamma * (y_MC[n] - chi));
    return mean(select_probs);
  }
  real compute_MCSE(real[] y_MC, real chi, real gamma) {
    int N_MC = size(y_MC);
    real select_probs[N_MC];
    for (n in 1:N_MC)
      select_probs[n] = Phi(gamma * (y_MC[n] - chi));
    return sqrt(variance(select_probs) / N_MC);
  }
}

data {
  int<lower=1> N; // Number of observations
  real y[N];      // Observations

  int<lower=1> N_grid; // Number of latent value grid points
  real y_grid[N_grid]; // Latent value grid points

  int<lower=1> N_MC; // Size of Monte Carlo ensemble
}

transformed data {
  // Exact latent probability distribution configuration
  real mu = -1;          // Location
  real<lower=0> tau = 3; // Shape

  // Generate Monte Carlo ensemble
  real y_MC[N_MC];
  for (n in 1:N_MC)
    y_MC[n] = normal_rng(mu, tau);
}

parameters {
  real chi;   // Location of selection function
  real gamma; // Shape of selection function
}

model {
  // Monte Carlo estimator of normalization integral
  real norm = compute_mc_norm(y_MC, chi, gamma);

  // Prior model
  chi ~ normal(0, 3 / 2.32);   // ~ 99% prior mass between -3 and +3
  gamma ~ normal(0, 3 / 2.32); // ~ 99% prior mass between -3 and +3
```

Stan
Program 10 `fit\_unknown\_both\_uni\_exact.stan`

```
data {
  int<lower=1> N; // Number of observations
  real y[N];      // Observations

  int<lower=1> N_grid; // Number of latent value grid points
  real y_grid[N_grid]; // Latent value grid points
}

parameters {
  real mu;           // Location of latent probability density function
  real<lower=0> tau; // Shape of latent probability density function
  real chi;          // Location of selection function
  real gamma;        // Shape of selection function
}

model {
  // Filtered normalization
  real norm = Phi( gamma * (mu - chi) / sqrt(1 + square(gamma * tau)) );

  // Prior model
  mu ~ normal(0, 5 / 2.32);   // ~ 99% prior mass between -5 and +5
  tau ~ normal(0, 5 / 2.57);  // ~ 99% prior mass between 0 and +5
  chi ~ normal(0, 3 / 2.32);   // ~ 99% prior mass between -3 and +3
  gamma ~ normal(0, 3 / 2.32); // ~ 99% prior mass between -3 and +3

  // Observational model
  for (n in 1:N) {
    real log_select_prob = log( Phi(gamma * (y[n] - chi)) );
    real lpdf = normal_lpdf(y[n] | mu, tau);
    target += log_select_prob + lpdf - log(norm);
  }
}

generated quantities {
  real select_prob_grid[N_grid]; // Selection function values
  real y_pred[N];                // Posterior predictive data

  for (n in 1:N_grid)
    select_prob_grid[n] = Phi(gamma * (y_grid[n] - chi));

  for (n in 1:N) {
    // Sample latent intensities until one survives the selection
    // process.  Use fixed iterations instead of while loop to avoid
    // excessive trials when the selection function and latent density
    // function are not aligned.
    for (m in 1:100) {
      real y_sample = normal_rng(mu, tau);
      if ( bernoulli_rng( Phi(gamma * (y_sample - chi)) ) ) {
        y_pred[n] = y_sample;
        break;
```

Stan
Program 11 `fit\_unknown\_both\_uni\_quad.stan`

```
functions {
  // Unnormalized observed probability density function
  real observed_updf(real x, real xc, real[] theta, real[] x_r, int[] x_i) {
    real chi = theta[1];
    real gamma = theta[2];
    real mu = theta[3];
    real tau = theta[4];

    return exp(normal_lpdf(x | mu, tau)) * Phi(gamma * (x - chi));
  }
}

data {
  int<lower=1> N; // Number of observations
  real y[N];      // Observations

  int<lower=1> N_grid; // Number of latent value grid points
  real y_grid[N_grid]; // Latent value grid points
}

transformed data {
  // Empty arrays for `integrate_1d` function
  real x_r[0];
  int x_i[0];
}

parameters {
  real mu;           // Location of latent probability density function
  real<lower=0> tau; // Shape of latent probability density function
  real chi;   // Location of selection function
  real gamma; // Shape of selection function
}

model {
  // Numerical quadrature estimator of normalization integral
  real theta[4] = {chi, gamma, mu, tau};
  real norm = integrate_1d(observed_updf,
                           negative_infinity(), positive_infinity(),
                           theta, x_r, x_i, 1e-8);

  // Prior model
  mu ~ normal(0, 5 / 2.32);   // ~ 99% prior mass between -5 and +5
  tau ~ normal(0, 5 / 2.57);  // ~ 99% prior mass between 0 and +5
  chi ~ normal(0, 3 / 2.32);   // ~ 99% prior mass between -3 and +3
  gamma ~ normal(0, 3 / 2.32); // ~ 99% prior mass between -3 and +3

  // Observational model
  for (n in 1:N) {
    real log_select_prob = log( Phi(gamma * (y[n] - chi)) );
    real lpdf = normal_lpdf(y[n] | mu, tau);
```

Stan
Program 12 `fit\_unknown\_both\_uni\_is.stan`

```
functions {
real compute_is_norm(real[] y_IS, real[] ref_lpdfs,
                     real chi, real gamma, real mu, real tau) {
    int N_IS = size(y_IS);
    vector[N_IS] weights;
    vector[N_IS] select_probs;
    for (n in 1:N_IS) {
      weights[n] = exp(normal_lpdf(y_IS[n] | mu, tau) - ref_lpdfs[n]);
      select_probs[n] = Phi(gamma * (y_IS[n] - chi));
    }
    return mean(weights .* select_probs);
  }

  // First returned component is the importance sampling standard error
  // Second returned component is the importance sampling effective sample size
  real[] compute_is_diagnostics(real[] y_IS, real[] ref_lpdfs,
                                real chi, real gamma,
                                real mu, real tau) {
    int N_IS = size(y_IS);
    vector[N_IS] weights;
    vector[N_IS] select_probs;
    for (n in 1:N_IS) {
      weights[n] = exp(normal_lpdf(y_IS[n] | mu, tau) - ref_lpdfs[n]);
      select_probs[n] = Phi(gamma * (y_IS[n] - chi));
    }
    return {sqrt(variance(weights .* select_probs) / N_IS),
            square(sum(weights)) / dot_self(weights)};
  }

  real compute_khat(vector fs) {
    int N = num_elements(fs);
    vector[N] sorted_fs = sort_asc(fs);

    real R;
    real q;
    real M;
    vector[N] b_hat_vec;
    vector[N] log_w_vec;

    real khat;
    real max_log_w;
    real b_hat_denom = 0;
    real b_hat_numer = 0;
    real b_hat;

    if (sorted_fs[1] == sorted_fs[N]) return not_a_number();
    if (sorted_fs[1] < 0) return not_a_number();

    // Estimate 25% quantile
    R = floor(0.25 * N + 0.5);
```

Stan
Program 13 simu_selection_multi.stan

```
data {
  int<lower=1> N; // Number of multivariate observations
  int<lower=1> K; // Number of components in each observation

  real chi;   // Location of selection function
  real gamma; // Shape of selection function
}

generated quantities {
  // Locations of latent probability density function
  vector[K] mu;

  // Scales of latent probability density function
  vector<lower=0>[K] tau;

  // Cholesky factor of probability density function correlation matrix
  cholesky_factor_corr[K] Psi;

  vector[K] y[N];
  real N_reject = 0;

  // Simulate location and scales
  for (k in 1:K) {
    mu[k] = normal_rng(0, 5 / 2.32);
    tau[k] = fabs(normal_rng(0, 3 / 2.57));
  }

  // Simulate correlation matrix and compute its Cholesky decomposition
  {
    matrix[K, K] Gamma = lkj_corr_rng(K, 4 / sqrt(K));
    Psi = cholesky_decompose(Gamma);
  }

  for (n in 1:N) {
    // Sample latent points until one survives the selection process
    while (1) {
      vector[K] y_sample
        = multi_normal_cholesky_rng(mu, diag_pre_multiply(tau, Psi));
      real s = sum(y_sample);
      if ( bernoulli_rng( Phi(gamma * (s - chi)) ) ) {
        y[n] = y_sample;
        break;
      } else {
        N_reject += 1;
      }
    }
  }
}
```

Stan
Program 14 `fit\_no_\\selection\_multi.stan`

```
data {
  int<lower=1> N; // Number of multivariate observations
  int<lower=1> K; // Number of components in each observation
  vector[K] y[N]; // Multivariate observations
}

parameters {
  // Locations of latent probability density function
  vector[K] mu;

  // Scales of latent probability density function
  vector<lower=0>[K] tau;

  // Cholesky factor of probability density function correlation matrix
  cholesky_factor_corr[K] Psi;
}

model {
  // Prior model
  mu ~ normal(0, 10 / 2.32);  // ~ 99% prior mass between -10 and +10
  tau ~ normal(0, 5 / 2.57);  // ~ 99% prior mass between 0 and +5
  Psi ~ lkj_corr_cholesky(4 / sqrt(K));

  // Observational model
  for (n in 1:N) {
    target += multi_normal_cholesky_lpdf(y[n] | mu,
                                         diag_pre_multiply(tau, Psi));
  }
}

generated quantities {
  vector[K] y_pred[N]; // Posterior predictive data

  for (n in 1:N) {
    y_pred[n] = multi_normal_cholesky_rng(mu, diag_pre_multiply(tau, Psi));
  }
}
```

Stan
Program 15 `fit\_unknown_\\selection\_multi_\\exact1.stan`

```
functions {
  real compute_norm(real chi, real gamma,
                    vector mu, vector tau, matrix Psi) {
    int K = num_elements(mu);
    real mu_s = sum(mu);
    real tau_s;

    matrix[K, K] L_Sigma = diag_pre_multiply(tau, Psi);
    matrix[K, K] Sigma = L_Sigma * L_Sigma';
    matrix[K, K] Lambda = inverse(Sigma);

    matrix[K - 1, K - 1] Gamma;
    vector[K - 1] l;

    for (i in 1:(K - 1)) {
      l[i] = Lambda[1, i + 1] - Lambda[1, 1];
      for (j in 1:(K - 1)) {
        Gamma[i, j] =  Lambda[i + 1, j + 1]
                     - Lambda[1, j + 1] - Lambda[i + 1, 1]
                     + Lambda[1, 1];
      }
    }
    tau_s = 1 / sqrt(Lambda[1, 1] - quad_form(inverse(Gamma), l));
    return Phi(  gamma * (mu_s - chi)
               / sqrt(1 + square(gamma * tau_s)) );
  }
}

data {
  int<lower=1> N; // Number of multivariate observations
  int<lower=1> K; // Number of components in each observation
  vector[K] y[N]; // Multivariate observations

  // Locations of latent probability density function
  vector[K] mu;

  // Scales of latent probability density function
  vector<lower=0>[K] tau;

  // Cholesky factor of probability density function correlation matrix
  cholesky_factor_corr[K] Psi;
}

parameters {
  real chi;   // Location of selection function
  real gamma; // Shape of selection function
}

model {
  real norm = compute_norm(chi, gamma, mu, tau, Psi);
```

Stan
Program 16 `fit\_unknown_\\selection\_multi_\\exact2.stan`

```
functions {
  real compute_norm(real chi, real gamma,
                    vector mu, vector tau, matrix Psi) {
    int K = num_elements(mu);
    real mu_s = sum(mu);
    real tau_s;

    matrix[K, K] L_Sigma = diag_pre_multiply(tau, Psi);
    matrix[K, K] Sigma = L_Sigma * L_Sigma';
    matrix[K, K] Lambda = inverse(Sigma);

    matrix[K - 1, K - 1] Gamma;
    vector[K - 1] l;

    for (i in 1:(K - 1)) {
      l[i] = Lambda[1, i + 1] - Lambda[1, 1];
      for (j in 1:(K - 1)) {
        Gamma[i, j] =  Lambda[i + 1, j + 1]
                     - Lambda[1, j + 1] - Lambda[i + 1, 1]
                     + Lambda[1, 1];
      }
    }
    tau_s = 1 / sqrt(Lambda[1, 1] - quad_form(inverse(Gamma), l));
    return Phi(  gamma * (mu_s - chi)
               / sqrt(1 + square(gamma * tau_s)) );
  }
}

data {
  int<lower=1> N; // Number of multivariate observations
  int<lower=1> K; // Number of components in each observation
  vector[K] y[N]; // Multivariate observations

  // Locations of latent probability density function
  vector[K] mu;

  // Scales of latent probability density function
  vector<lower=0>[K] tau;

  // Cholesky factor of probability density function correlation matrix
  cholesky_factor_corr[K] Psi;
}

parameters {
  real chi;            // Location of selection function
  real<upper=0> gamma; // Shape of selection function
}

model {
  real norm = compute_norm(chi, gamma, mu, tau, Psi);
```

Stan
Program 17 `fit\_unknown_\\selection\_multi_\\mc.stan`

```
functions {
    real compute_exact_norm(real chi, real gamma,
                            vector mu, vector tau, matrix Psi) {
    int K = num_elements(mu);
    real mu_s = sum(mu);
    real tau_s;

    matrix[K, K] L_Sigma = diag_pre_multiply(tau, Psi);
    matrix[K, K] Sigma = L_Sigma * L_Sigma';
    matrix[K, K] Lambda = inverse(Sigma);

    matrix[K - 1, K - 1] Gamma;
    vector[K - 1] l;

    for (i in 1:(K - 1)) {
      l[i] = Lambda[1, i + 1] - Lambda[1, 1];
      for (j in 1:(K - 1)) {
        Gamma[i, j] =  Lambda[i + 1, j + 1]
                     - Lambda[1, j + 1] - Lambda[i + 1, 1]
                     + Lambda[1, 1];
      }
    }
    tau_s = 1 / sqrt(Lambda[1, 1] - quad_form(inverse(Gamma), l));
    return Phi(  gamma * (mu_s - chi)
               / sqrt(1 + square(gamma * tau_s)) );
  }
  real compute_mc_norm(vector[] y_MC, real chi, real gamma) {
    int N_MC = size(y_MC);
    real select_probs[N_MC];
    for (n in 1:N_MC)
      select_probs[n] = Phi(gamma * (sum(y_MC[n]) - chi));
    return mean(select_probs);
  }
  real compute_MCSE(vector[] y_MC, real chi, real gamma) {
    int N_MC = size(y_MC);
    real select_probs[N_MC];
    for (n in 1:N_MC)
      select_probs[n] = Phi(gamma * (sum(y_MC[n]) - chi));
    return sqrt(variance(select_probs) / N_MC);
  }
}

data {
  int<lower=1> N; // Number of multivariate observations
  int<lower=1> K; // Number of components in each observation
  vector[K] y[N]; // Multivariate observations

  // Locations of latent probability density function
  vector[K] mu;
```

Stan
Program 18 `fit\_unknown_\\both\_multi_\\exact.stan`

```
functions {
  real compute_norm(real chi, real gamma,
                    vector mu, vector tau, matrix Psi) {
    int K = num_elements(mu);
    real mu_s = sum(mu);
    real tau_s;

    matrix[K, K] L_Sigma = diag_pre_multiply(tau, Psi);
    matrix[K, K] Sigma = L_Sigma * L_Sigma';
    matrix[K, K] Lambda = inverse(Sigma);

    matrix[K - 1, K - 1] Gamma;
    vector[K - 1] l;

    for (i in 1:(K - 1)) {
      l[i] = Lambda[1, i + 1] - Lambda[1, 1];
      for (j in 1:(K - 1)) {
        Gamma[i, j] =  Lambda[i + 1, j + 1]
                     - Lambda[1, j + 1] - Lambda[i + 1, 1]
                     + Lambda[1, 1];
      }
    }
    tau_s = 1 / sqrt(Lambda[1, 1] - quad_form(inverse(Gamma), l));
    return Phi(  gamma * (mu_s - chi)
               / sqrt(1 + square(gamma * tau_s)) );
  }
}

data {
  int<lower=1> N; // Number of multivariate observations
  int<lower=1> K; // Number of components in each observation
  vector[K] y[N]; // Multivariate observations
}

parameters {
  // Locations of latent probability density function
  vector[K] mu;

  // Scales of latent probability density function
  vector<lower=0>[K] tau;

  // Cholesky factor of probability density function correlation matrix
  cholesky_factor_corr[K] Psi;

  real chi;            // Location of selection function
  real<upper=0> gamma; // Shape of selection function
}

model {
  real norm = compute_norm(chi, gamma, mu, tau, Psi);
```

Stan
Program 19 `fit\_unknown_\\both\_multi_\\is1.stan`

```
functions {
  real compute_exact_norm(real chi, real gamma,
                          vector mu, vector tau, matrix Psi) {
    int K = num_elements(mu);
    real mu_s = sum(mu);
    real tau_s;

    matrix[K, K] L_Sigma = diag_pre_multiply(tau, Psi);
    matrix[K, K] Sigma = L_Sigma * L_Sigma';
    matrix[K, K] Lambda = inverse(Sigma);

    matrix[K - 1, K - 1] Gamma;
    vector[K - 1] l;

    for (i in 1:(K - 1)) {
      l[i] = Lambda[1, i + 1] - Lambda[1, 1];
      for (j in 1:(K - 1)) {
        Gamma[i, j] =  Lambda[i + 1, j + 1]
                     - Lambda[1, j + 1] - Lambda[i + 1, 1]
                     + Lambda[1, 1];
      }
    }
    tau_s = 1 / sqrt(Lambda[1, 1] - quad_form(inverse(Gamma), l));
    return Phi(  gamma * (mu_s - chi)
               / sqrt(1 + square(gamma * tau_s)) );
  }

  real compute_is_norm(vector[] y_IS, real[] ref_lpdfs,
                       real chi, real gamma,
                       vector mu, vector tau, matrix Psi) {
    int N_IS = size(y_IS);
    int K = size(y_IS[1]);

    vector[N_IS] weights;
    vector[N_IS] select_probs;
    matrix[K, K] L_Sigma = diag_pre_multiply(tau, Psi);
    for (n in 1:N_IS) {
      real active_lpdf =
        multi_normal_cholesky_lpdf(y_IS[n] | mu, L_Sigma);
      real s = sum(y_IS[n]);

      weights[n] = exp(active_lpdf - ref_lpdfs[n]);
      select_probs[n] = Phi(gamma * (s - chi));
    }
    return mean(weights .* select_probs);
  }

  // First component is the importance sampling standard error
  // Second component is the importance sampling effective sample size
  real[] compute_is_diagnostics(vector[] y_IS, real[] ref_lpdfs,
```

Stan
Program 20 `fit\_unknown_\\both\_multi_\\is2.stan`

```
functions {
  real compute_exact_norm(real chi, real gamma,
                          vector mu, vector tau, matrix Psi) {
    int K = num_elements(mu);
    real mu_s = sum(mu);
    real tau_s;

    matrix[K, K] L_Sigma = diag_pre_multiply(tau, Psi);
    matrix[K, K] Sigma = L_Sigma * L_Sigma';
    matrix[K, K] Lambda = inverse(Sigma);

    matrix[K - 1, K - 1] Gamma;
    vector[K - 1] l;

    for (i in 1:(K - 1)) {
      l[i] = Lambda[1, i + 1] - Lambda[1, 1];
      for (j in 1:(K - 1)) {
        Gamma[i, j] =  Lambda[i + 1, j + 1]
                     - Lambda[1, j + 1] - Lambda[i + 1, 1]
                     + Lambda[1, 1];
      }
    }
    tau_s = 1 / sqrt(Lambda[1, 1] - quad_form(inverse(Gamma), l));
    return Phi(  gamma * (mu_s - chi)
               / sqrt(1 + square(gamma * tau_s)) );
  }

  real compute_is_norm(vector[] y_IS, real[] ref_lpdfs,
                       real chi, real gamma,
                       vector mu, vector tau, matrix Psi) {
    int N_IS = size(y_IS);
    int K = size(y_IS[1]);

    vector[N_IS] weights;
    vector[N_IS] select_probs;
    matrix[K, K] L_Sigma = diag_pre_multiply(tau, Psi);
    for (n in 1:N_IS) {
      real active_lpdf =
        multi_normal_cholesky_lpdf(y_IS[n] | mu, L_Sigma);
      real s = sum(y_IS[n]);

      weights[n] = exp(active_lpdf - ref_lpdfs[n]);
      select_probs[n] = Phi(gamma * (s - chi));
    }
    return mean(weights .* select_probs);
  }

  // First component is the importance sampling standard error
  // Second component is the importance sampling effective sample size
  real[] compute_is_diagnostics(vector[] y_IS, real[] ref_lpdfs,
```