Hidden Markov Models

Michael Betancourt

June 2025

Table of contents

In the mixture modeling chapter we first considered how to model data that could have arisen from one of a number of component data generating processes. There we derived mixture observational models under the assumption that the activation of a component data generating process, and hence the behavior of the component assignment variables, is independent across individual observations.

Mixture observational models have broad applicability in practice, but they are by no means universal. In order to generalize this modeling technique we need to consider joint mixture observational models that *couple* the component assignment variables together. These generalizations, however, also introduce an immediate practical challenge: how, if at all, can be efficiently marginalize the component assignment variables?

From this perspective hidden Markov models are an exceptional generalization to mixture observational models. On one hand they allow for more sophisticated behavior by allowing the contribution of a component data generating process to *persist* across multiple observations before another data generating process takes over. On the other their mathematical structure is just cooperative enough to allow for practical marginalization.

In this chapter I will introduce hidden Markov models, carefully derive the algorithms needed for efficient marginalization, and demonstrate their robust implementation in Stan.

1 Joint Mixture Observational Models

Consider I component observations,

$$y_1, \ldots, y_i, \ldots, y_I,$$

each of which can arise from one of K component observational models,

$$p(y_i \mid i, k, \theta) = p_{i,k}(y_i \mid \theta).$$

Note that in general the y_i, and hence the precise form of the $p(y_i \mid i, k, \theta)$, do *not* have to homogeneous. For example some y_i might be discrete while others are continuous. Moreover the dimension of each y_i might vary, with some observations featuring more sub-components than others.

To quantify which component observational model is responsible for each component observation we introduce I categorical assignment variables,

$$z_i \in \{1, \ldots, K\}.$$

These assignment variables are also known as **states**, as in the state of the overall data generating process for each observation.

When the states z_i are unknown, equivalently *latent* or even *hidden*, we need to need to incorporate them into the observational model so that that we can infer their behavior. This requires a joint model of the form (Figure 1)

$$\begin{aligned}
p(z_1, \ldots, z_I, y_1, \ldots, y_I, \theta) &= p(y_1, \ldots, y_I \mid z_1, \ldots, z_I, \theta)\, p(z_1, \ldots, z_I \mid \theta)\, p(\theta) \\
&= \prod_{i=1}^{I} p_{i,z_i}(y_i \mid \theta) p(z_1, \ldots, z_I \mid \theta)\, p(\theta) \\
&= \prod_{i=1}^{I} p(y_i \mid i, z_i, \theta) p(z_1, \ldots, z_I \mid \theta)\, p(\theta).
\end{aligned}$$

To make the equations in this chapter a bit less overwhelming let's take the opportunity to introduce a more compact notation for sequences of observations and latent states,

$$\begin{aligned}
y_{i_i:i_2} &= (y_{i_1}, y_{i_1+1}, \ldots, y_{i_2-1}, y_{i_2}) \\
z_{i_i:i_2} &= (z_{i_1}, z_{i_1+1}, \ldots, z_{i_2-1}, z_{i_2}).
\end{aligned}$$

This allows us to write the general joint model as

$$p(z_{1:I}, y_{1:I}, \theta) = \prod_{i=1}^{I} p(y_i \mid i, z_i, \theta) p(z_1, \ldots, z_I \mid \theta)\, p(\theta).$$

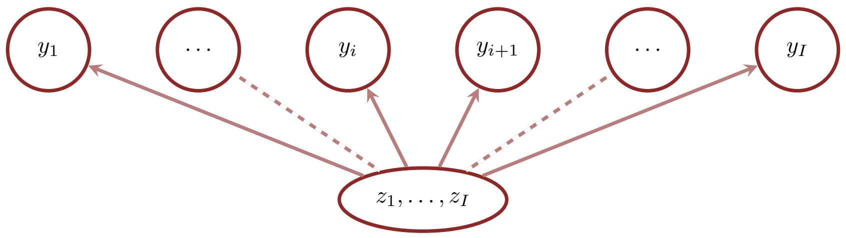


Figure 1: In general the behavior of the latent assignment variables for each mixture observational model, or latent states, are all coupled together.

In the mixture modeling chapter we derived mixture observational models by assuming that all of the latent states were independent and identically distributed (Figure 2),

$$p(z_{1:I} \mid \theta) = \prod_{i=1}^{I} p(z_i \mid \theta);$$

I will refer to this as a **basic mixture observational model**. With this assumption we were able to marginalize out each discrete latent state individually. This, in turn, allowed us to use high performance computational tools like Hamiltonian Monte Carlo.

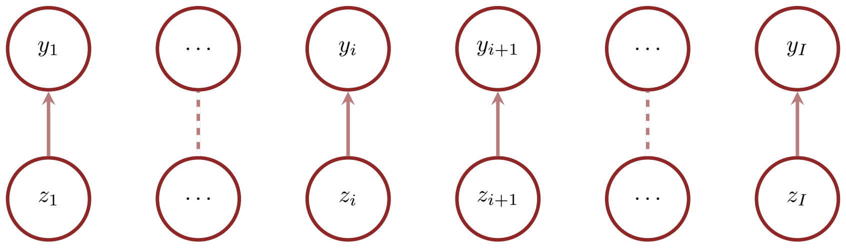


Figure 2: Basic mixture observational models assume that the latent states are independent.

An arbitrary joint mixture observational model, however, features K^I distinct configurations of the latent states. Consequently direct summation of the latent states,

$$p(y_{1:I} \mid \theta) = \sum_{z_1} \cdots \sum_{z_I} p(z_{1:I}, y_{1:I} \mid \theta),$$

is typically infeasible. There is a key exception, however, where the coupling between the latent states allows for efficient marginalization while still being rich enough to be useful for practical applications.

To motivate this coupling let's first assume that the latent states are ordered so that we have a well-defined notion of neighboring states. For example the observations might be indexed in the temporal order in which they were made, organized along a one-dimensional spatial direction, and the like.

Next we will assume that each latent state is conditionally dependent on only its preceding neighbor (Figure 3),

$$p(z_{1:I}, \mid \theta) = p(z_1) \prod_{i=2}^{I} p(z_i \mid z_{i-1}, \theta).$$

This particular dependency structure arises so often in both theoretical and applied probability theory that it has been given its own adjective: **Markovian**.

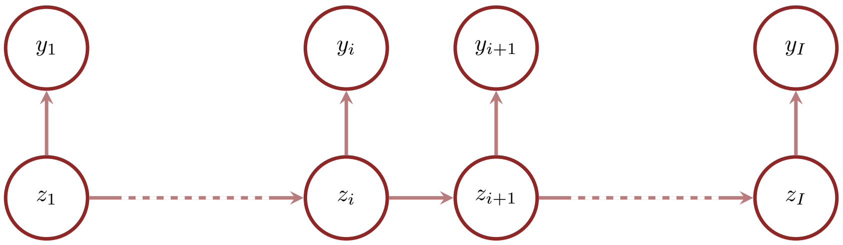


Figure 3: In a Markov model for the latent states the behavior of each latent state depends on only the behavior of its preceding neighbor.

Under this Markovian assumption the joint model becomes

$$p(z_{1:I}, y_{1:I}, \theta) = p(z_{1:I}, y_{1:I} \mid \theta) \, p(\theta)$$

where

$$\begin{aligned} p(z_{1:I}, y_{1:I} \mid \theta) &= \prod_{i=1}^{I} p_(y_i \mid i, z_i, \theta) p(z_1, \ldots, z_I \mid \theta) \\ &= \left[\prod_{i=1}^{I} p(y_i \mid i, z_i, \theta) \right] \left[p(z_1) \prod_{i=2}^{I} p(z_i \mid z_{i-1}, \theta) \right] \\ &= p(z_1) \, (y_1 \mid 1, z_1, \theta) \prod_{i=2}^{I} p(z_i \mid z_{i-1}, \theta) \, p(y_i \mid i, z_i, \theta). \end{aligned}$$

Because it assumes "Markovian" dependencies for a sequence of "hidden" states this model is known as a **hidden Markov model**. That said is it not unreasonable to refer to it as a **Markov mixture observational model** if we want to emphasize the assumed structure of the observations as well as that of the latent states.

In the context of a hidden Markov model the conditional probabilities $p(z_i \mid z_{i-1}, \theta)$ are known referred to as **transition probabilities**. A hidden Markov model with the same transition probabilities between all pairs of states, so that

$$p(z_i = k' \mid z_{i-1} = k, \theta)$$

depends on only k and k' but not i, is known as a **homogeneous hidden Markov model**. **Heterogeneous**, **non-homogeneous**, and **general** hidden Markov models are all terms used for emphasizing varying transition probabilities.

2 The Marginal Hidden Markov Model

In theory the joint observational model $p(z_{1:I}, y_{1:I} \mid \theta)$ is sufficient to infer which latent states $z_{1:I}$ and model configurations θ are consistent with any observed data. The discreteness of the latent states, however, frustrates practical implementation of those inferences.

Fortunately the Markovian structure of hidden Markov models allows us to efficiently marginalize out the discrete latent states. That said actually deriving an explicit marginalization algorithm from that structure is a pretty onerous task.

In this section I will motivate and then carefully derive the strategy for efficiently marginalizing the discrete hidden states of a hidden Markov model. The derivation requires some pretty brutal calculations and readers interested in only the final result can jump straight to Section 2.3 without missing anything.

2.1 Looking For A Pattern

First and foremost let's motivate the basic approach to efficient marginalization by carefully working out the direct summations for $I = 1$, $I = 2$, and then $I = 3$ and then looking for any useful patterns.

When $I = 1$ we have

$$p(z_1, y_1 \mid \theta) = p(z_1) \, p(y_1 \mid 1, z_1, \theta).$$

Summing over the initial latent state gives

$$\begin{aligned} p(y_1 \mid \theta) &= \sum_{z_1} p(z_1, y_1 \mid \theta) \\ &= \sum_{z_1} p(y_1 \mid 1, z_1, \theta) \, p(z_1). \end{aligned}$$

Note the similarity to the marginal form of a basic mixture observational model,

$$p(y \mid \theta) = \sum_{z} p(y \mid z, \theta) \, p(z).$$

For $I = 2$ we have

$$p(z_1, z_2, y_1, y_2 \mid \theta) = p(z_1) \, p(y_1 \mid 1, z_1, \theta) \, p(z_2 \mid z_1, \theta) \, p(y_2 \mid 2, z_2, \theta),$$

or, equivalently,

$$p(z_1, z_2, y_1, y_2 \mid \theta) = p(z_1, y_1 \mid \theta) \, p(z_2 \mid z_1, \theta) \, p(y_2 \mid 2, z_2, \theta).$$

Summing over the initial latent state gives

$$\begin{aligned} p(z_2, y_1, y_2 \mid \theta) &= \sum_{z_1} p(z_1, z_2, y_1, y_2 \mid \theta) \\ &= \sum_{z_1} p(z_1, y_1 \mid \theta) \, p(z_2 \mid z_1, \theta) \, p(y_2 \mid 2, z_2, \theta) \\ &= p(y_2 \mid 2, z_2, \theta) \sum_{z_1} p(z_1, y_1 \mid \theta) \, p(z_2 \mid z_1, \theta). \end{aligned}$$

We can then marginalize out both of the latent states by summing this result over z_2,

$$\begin{aligned} p(y_1, y_2 \mid \theta) &= \sum_{z_2} p(z_2, y_1, y_2 \mid \theta) \\ &= \sum_{z_2} p(y_2 \mid 2, z_2, \theta) \left[\sum_{z_1} p(z_1, y_1 \mid \theta) \, p(z_2 \mid z_1, \theta) \right] \\ &= \sum_{z_2} p(y_2 \mid 2, z_2, \theta) \, p(z_2 \mid y_1, \theta). \end{aligned}$$

Interestingly this once again this has the same form as a basic mixture observational model. Now, however, the component probabilities are informed by the previous observation,

$$p(z_2 \mid y_1, \theta) = \sum_{z_1} p(z_1, y_1 \mid \theta) \, p(z_2 \mid z_1, \theta).$$

Finally let's consider $I = 3$ where the joint model is given by

$$\begin{aligned} p(z_{1:3}, y_{1:3} \mid \theta) &= p(z_1) \, p(y_1 \mid 1, z_1, \theta) \prod_{i=2}^{3} p(z_i \mid z_{i-1}, \theta) \, p(y_i \mid i, z_i, \theta) \\ &= p(z_1, y_1 \mid \theta) \prod_{i=2}^{3} p(z_i \mid z_{i-1}, \theta) \, p(y_i \mid i, z_i, \theta). \end{aligned}$$

Here summing over the initial latent state gives

$$
\begin{aligned}
p(z_{2:3}, y_{1:3} \mid \theta) &= \sum_{z_1} p(z_{1:3}, y_{1:3} \mid \theta) \\
&= \sum_{z_1} p(z_1, y_1 \mid \theta) \prod_{i=2}^{3} p(z_i \mid z_{i-1}, \theta) \, p(y_i \mid i, z_i, \theta) \\
&= \; \Big[p(y_3 \mid 3, z_3, \theta) \, p(z_3 \mid z_2, \theta) \Big] \\
&\quad \cdot \Big[p(y_2 \mid 2, z_2, \theta) \, \sum_{z_1} p(z_1, y_1 \mid \theta) \, p(z_2 \mid z_1, \theta) \Big].
\end{aligned}
$$

The second term might look familiar; it's exactly $p(z_2, y_1, y_2 \mid \theta)$ which we just derived for the $I = 2$ case! Consequently we can write

$$
\begin{aligned}
p(z_{2:3}, y_{1:3} \mid \theta) &= \; \Big[p(y_3 \mid 3, z_3, \theta) \, p(z_3 \mid z_2, \theta) \Big] \\
&\quad \cdot \Big[p(y_2 \mid 2, z_2, \theta) \, \sum_{z_1} p(z_1, y_1 \mid \theta) \, p(z_2 \mid z_1, \theta) \Big] \\
&= \; \Big[p(y_3 \mid 3, z_3, \theta) \, p(z_3 \mid z_2, \theta) \Big] \\
&\quad \cdot \Big[p(z_2, y_1, y_2 \mid \theta) \Big] \\
&= p(y_3 \mid 3, z_3, \theta) \, p(z_2, y_1, y_2 \mid \theta) \, p(z_3 \mid z_2, \theta).
\end{aligned}
$$

Now we can sum over z_2 to give

$$
\begin{aligned}
p(z_3, y_{1:3} \mid \theta) &= \sum_{z_2} p(z_{2:3}, y_{1:3} \mid \theta) \\
&= \sum_{z_2} p(y_3 \mid 3, z_3, \theta) \, p(z_2, y_1, y_2 \mid \theta) \, p(z_3 \mid z_2, \theta) \\
&= p(y_3 \mid 3, z_3, \theta) \, \sum_{z_2} p(z_2, y_1, y_2 \mid \theta) \, p(z_3 \mid z_2, \theta).
\end{aligned}
$$

Notice how we derive $p(z_3, y_{1:3} \mid \theta)$ from $p(z_2, y_1, y_2 \mid \theta)$ in *exactly the same way* that we derived $p(z_2, y_1, y_2 \mid \theta)$ from $p(z_1, y_1 \mid \theta)$.

Lastly we sum over z_3 to give the full marginal probability density function,

$$\begin{aligned}
p(y_{1:3} \mid \theta) &= \sum_{z_3} p(z_3, y_{1:3} \mid \theta) \\
&= \sum_{z_3} p(y_3 \mid 3, z_3, \theta) \left[\sum_{z_2} p(z_2, y_1, y_2 \mid \theta) \, p(z_3 \mid z_2, \theta) \right] \\
&= \sum_{z_3} p(y_3 \mid 3, z_3, \theta) \, p(z_3 \mid y_{1:2}, \theta).
\end{aligned}$$

The final marginal model continues to mirror a basic mixture observational model, in this case with derived component probabilities

$$p(z_3 \mid y_{1:2}, \theta) = \sum_{z_2} p(z_2, y_1, y_2 \mid \theta) \, p(z_3 \mid z_2, \theta).$$

Let's summarize what we have seen in these first few cases.

Instead of deriving $p(y_{1:3} \mid \theta)$ by summing $p(z_{1:3}, y_{1:3} \mid \theta)$ over z_1, z_2, and z_3 at the same time, we can evaluate it by emulating a basic mixture observational model and summing $p(z_3, y_{1:3} \mid \theta)$ over just z_3.

Conveniently $p(z_3, y_{1:3} \mid \theta)$ can be derived directly from $p(z_2, y_{1:2} \mid \theta)$, and that term can be derived from $p(z_1, y_1 \mid \theta)$ which in turn can be derived from the initial state probabilities $p(z_1)$,

$$\begin{aligned}
p(z_1, y_1 \quad \mid \theta) &= p(y_1 \mid 1, z_1, \theta) \, p(z_1), \\
p(z_2, y_{1:2} \mid \theta) &= p(y_2 \mid 2, z_2, \theta) \sum_{z_1} p(z_1, y_1 \mid \theta) \, p(z_2 \mid z_1, \theta), \\
p(z_3, y_{1:3} \mid \theta) &= p(y_3 \mid 3, z_3, \theta) \, \sum_{z_2} p(z_2, y_{1:2} \mid \theta) \, p(z_3 \mid z_2, \theta).
\end{aligned}$$

At this point the important question is whether or not this recursive pattern over the first three cases holds more generally, with

$$p(z_i \mid y_{1:i}, \theta) = p(y_i \mid i, z_i, \theta) \left[\sum_{z_{i-1}} p(z_{i-1}, y_{1:(i-1)} \mid \theta) \, p(z_i \mid z_{i-1}, \theta) \right]$$

for any $1 < i \le I$. If it does then starting with the initial state probabilities $p(z_1)$ we can *iteratively* compute $p(z_I \mid y_{1:I} \mid \theta)$ and then sum over z_I to give the marginal model $p(y_{1:I} \mid \theta)$.

2.2 Verifying The Pattern

While not exactly straightforward, with enough mathematical effort we can derive a general relationship between $p(z_i, y_{1:i} \mid \theta)$ and $p(z_{i-1}, y_{1:(i-1)} \mid \theta)$ using conditional probability theory and the conditional structure of the joint hidden Markov model. The exact properties we need are pretty onerous to work out, and I will not attempt to do so in this chapter. Instead I will shamelessly defer to Bishop (2006) which reviews the necessary properties and strategies for deriving them.

Ultimately we want to reduce $p(z_i, y_{1:i} \mid \theta)$ to an object that depends on only the first $i - 1$ latent states and observations. To that end let's peel off the terms that depend z_i and y_i,

$$\begin{aligned}
p(z_i, y_{1:i} \mid \theta) &= p(z_i, y_i, y_{1:(i-1)} \mid \theta) \\
&= p(y_{1:(i-1)} \mid z_i, y_i, \theta) \, p(z_i, y_i \mid \theta) \\
&= p(y_{1:(i-1)} \mid z_i, y_i, \theta) \, p(y_i \mid i, z_i, \theta) \, p(z_i \mid \theta).
\end{aligned}$$

Because of the Markovian structure of the hidden Markov model the first term can be written as

$$p(y_{1:(i-1)} \mid z_i, y_i, \theta) = p(y_{1:(i-1)} \mid z_i, \theta)$$

Consequently we have

$$\begin{aligned}
p(z_i, y_{1:i} \mid \theta) &= p(y_{1:(i-1)} \mid z_i, y_i, \theta) \, p(y_i \mid i, z_i, \theta) \, p(z_i \mid \theta) \\
&= p(y_{1:(i-1)} \mid z_i, \theta) \, p(y_i \mid i, z_i, \theta) \, p(z_i \mid \theta) \\
&= p(y_i \mid i, z_i, \theta) \left[p(y_{1:(i-1)} \mid i, z_i, \theta) \, p(z_i \mid \theta) \right] \\
&= p(y_i \mid i, z_i, \theta) \, p(z_i, y_{1:(i-1)} \mid \theta).
\end{aligned}$$

The second term here is a bit awkward, but it can be simplified by writing it as a marginal probability density function,

$$\begin{aligned}
p(z_i, y_{1:(i-1)} \mid \theta) &= \sum_{z_{i-1}} p(z_{i-1}, z_i, y_{1:(i-1)} \mid \theta) \\
&= \sum_{z_{i-1}} p(z_i, y_{1:(i-1)} \mid z_{i-1}, \theta) \, p(z_{i-1} \mid \theta).
\end{aligned}$$

We can further simplify the first term in the sum to

$$p(z_i, y_{1:(i-1)} \mid z_{i-1}, \theta) = p(z_i \mid z_{i-1}, \theta) \, p(y_{1:(i-1)} \mid z_{i-1}, \theta);$$

intuitively conditioning on z_{i-1} blocks any coupling between the following latent state z_i and the previous observations $y_{1:(i-1)}$.

This allows us to write

$$\begin{aligned}
p(z_i, y_{1:(i-1)} \mid \theta) &= \sum_{z_{i-1}} p(z_i, y_{1:(i-1)} \mid z_{i-1}, \theta) \, p(z_{i-1} \mid \theta) \\
&= \sum_{z_{i-1}} p(z_i \mid z_{i-1}, \theta) \, p(y_{1:(i-1)} \mid z_{i-1}, \theta) \, p(z_{i-1} \mid \theta) \\
&= \sum_{z_{i-1}} p(z_i \mid z_{i-1}, \theta) \left[p(y_{1:(i-1)} \mid z_{i-1}, \theta) \, p(z_{i-1} \mid \theta) \right] \\
&= \sum_{z_{i-1}} p(z_i \mid z_{i-1}, \theta) \, p(z_{i-1}, y_{1:(i-1)} \mid \theta).
\end{aligned}$$

Substituting this back into the original equation for $p(z_i, y_{1:i} \mid \theta)$ finally gives

$$\begin{aligned}
p(z_i, y_{1:i} \mid \theta) &= p(y_i \mid i, z_i, \theta) \, p(z_i, y_{1:(i-1)} \mid \theta) \\
&= p(y_i \mid i, z_i, \theta) \sum_{z_{i-1}} p(z_i \mid z_{i-1}, \theta) \, p(z_{i-1}, y_{1:(i-1)} \mid \theta),
\end{aligned}$$

which is exactly the recursion equation that generalizes the pattern we found in the Section 2.1.

2.3 The Forward Algorithm

Implementing this recursive calculation given observations

$$\tilde{y}_1, \ldots, \tilde{y}_i, \ldots, \tilde{y}_I$$

is made a bit more straightforward with the introduction of some more compact notation.

First let's denote the initial state probabilities, observational model outputs, and transition probabilities as

$$\begin{aligned}
\rho_k &= p(z_0 = k) \\
\omega_{i,k} &= p(\tilde{y}_i \mid i, z_i = k, \theta) \\
\Gamma_{i,k'k} &= p(z_i = k \mid z_{i-1} = k', \theta)
\end{aligned}$$

Then we can define

$$\alpha_{i,k} = p(z_i = k, \tilde{y}_{1:i} \mid \theta)$$

with the recursion becoming

$$\begin{aligned}
\alpha_{i,k} &= p(\tilde{y}_i \mid i, z_i = k) \sum_{k'=1}^{K} p(z_i = k \mid z_{i-1} = k') \, \alpha_{i-1,k'} \\
&= \omega_{i,k} \sum_{k'=1}^{K} \Gamma_{i,k'k} \, \alpha_{i-1,k'}.
\end{aligned}$$

Conveniently we can also write this as a matrix equation,

$$\alpha_i = \omega_i \circ \left(\Gamma_i^T \cdot \alpha_{i-1} \right),$$

where $\circ$ denotes the Hadamard or element-wise, product. Alternatively we can use only matrix-vector products,

$$\alpha_i = \text{diag}(\omega_i) \cdot \left(\Gamma_i^T \cdot \alpha_{i-1} \right),$$

although we have to be careful to not actually evaluate $\text{diag}(\omega_i)$ as a dense matrix and waste precious computational resources.

In this matrix notation the evaluation of the marginal observational model is given by

$$\begin{aligned}
\alpha_0 &= \rho \\
&\ldots \\
\alpha_i &= \omega_i \circ \left(\Gamma_i^T \cdot \alpha_{i-1} \right) \\
&\ldots \\
p(\tilde{y}_{1:I} \mid \theta) &= \mathbf{1}^T \cdot \alpha_I.
\end{aligned}$$

This procedure is also known as the **forward algorithm** as it iterates through the latent states in sequential order.

The cost of each iteration of the forward algorithm is dominated by a matrix-vector product which requires $\mathcal{O}(K^2)$ operations. Consequently the cost of the entire forward algorithm scales as $\mathcal{O}(I\,K^2)$. For all but the smallest values of K and I this is drastically faster than the $\mathcal{O}(K^I)$ scaling that would be required of a brute force summation.

That said for sufficiently large K and/or I the cost of implementing the forward algorithm can still be prohibitive. In particular increasing the dimension of the latent states increases cost much faster than increasing the number of latent states.

2.4 Alternative Conventions

In the previous derivation of the forward algorithm I have made a few choices of notation that are not universal, and hence might clash with other references.

For example many references like Bishop (2006) consider an initial latent state z_0 that is not paired with a corresponding observation (Figure 4). One awkward consequence of this convention is that there will be a different number of latent states and observations which can be awkward to manage in practical implementations. Indexing is already hard enough!

The convention used in this chapter is more general in the sense that the alternative convention can be recovered by setting

$$\omega_{1,k} = p(\tilde{y}_1 \mid 1, z_i = k, \theta) = 1$$

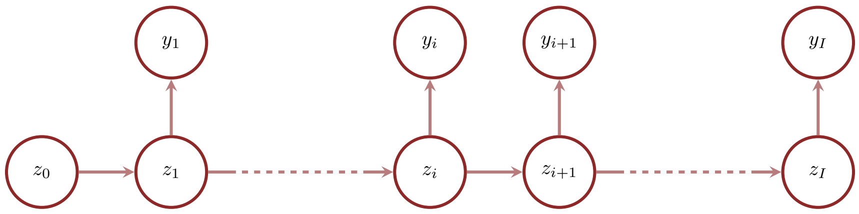


Figure 4: One common convention for hidden Markov models assumes that the initial latent state is not complemented with any observations.

for all k.

Another potential divergence in notation concerns the transition matrix Γ. Some references like Bishop (2006) define the elements of the transition matrix not as

$$\Gamma_{i,k'k} = p(z_i = k \mid z_{i-1} = k', \theta)$$

but rather the transpose

$$\Gamma_{i,kk'} = p(z_i = k \mid z_{i-1} = k', \theta).$$

Neither convention has any substantial practical advantage over the other, we just have to make sure that we're using one convention consistently.

3 Inferring Latent State Behavior

After marginalization we can still recover inferences for the latent states with an application of Bayes' Theorem. Specifically the joint posterior density function for all of the latent states is given, up to normalization, by partially evaluating the joint model,

$$p(z_{1:I} \mid \tilde{y}_{1:I}, \theta) \propto p(z_{1:I}, \tilde{y}_{1:I}, \theta).$$

When I is large, however, this joint posterior density function will be much too ungainly to manipulate or communicate directly.

If we are interested in the behavior of the latent states then we will need to be able to isolate meaningful, low-dimensional summaries of these inferences.

As in Section 2 we will make heavy use of some of the properties of the hidden Markov model detailed in Bishop (2006). Moreover, readers interested in only the final results can safely jump directly to the end of each section.

3.1 Marginal Posterior Probabilities For Individual Latent States

In some applications the marginal posterior probabilities for a single latent state,

$$p(z_i \mid y_{1:I}, \theta),$$

is useful. We can derive these marginal posterior probabilities with an application of Bayes' Theorem,

$$p(z_i \mid y_{1:I}, \theta) \propto p(y_{1:I} \mid z_i, \theta) \, p(z_i \mid \theta).$$

Fortunately deriving the terms on the right hand side of this equation is relatively straightforward if we take advantage of the structure of the joint model. For example the first term simplifies to,

$$p(y_{1:I} \mid z_i, \theta) = p(y_{1:i} \mid z_i, \theta) \, p(y_{(i+1):I} \mid z_i, \theta).$$

In words conditioning on z_i breaks any coupling between the proceeding observations and the following observations.

Substituting this result gives

$$\begin{aligned} p(z_i \mid y_{1:I}, \theta) &\propto p(y_{1:I} \mid z_i, \theta) \, p(z_i \mid \theta) \\ &\propto p(y_{1:i} \mid z_i, \theta) \, p(y_{(i+1):I} \mid z_i, \theta) \, p(z_i \mid \theta) \\ &\propto \bigg[p(y_{1:i} \mid z_i, \theta) \, p(z_i \mid \theta) \bigg] \, p(y_{(i+1):I} \mid z_i, \theta) \\ &\propto p(z_i, y_{1:i} \mid \theta) \, p(y_{(i+1):I} \mid z_i, \theta). \end{aligned}$$

As we saw in Section 2, when we run the forward algorithm for a given observation we compute $p(z_i, \tilde{y}_{1:i} \mid \theta)$ for each i. Consequently in order to compute the marginal posterior probabilities for each latent state we just need to save these intermediate values and then compute $p(\tilde{y}_{(i+1):I} \mid z_i, \theta)$.

Conveniently this term can be evaluated with a recursion of its own. To parallel the notation of the forward algorithm let's denote

$$\beta_{i,z_i} = p(\tilde{y}_{(i+1):I} \mid z_i, \theta).$$

In order to compute a particular β_{i,z_i} we need to introduce the latent state z_{i+1} and then decompose,

$$\begin{aligned} \beta_{i,z_i} &= p(\tilde{y}_{(i+1):I} \mid z_i, \theta) \\ &= \sum_{z_{i+1}} p(\tilde{y}_{(i+1):I}, z_{i+1} \mid z_i, \theta) \\ &= \sum_{z_{i+1}} p(\tilde{y}_{(i+1):I} \mid z_i, z_{i+1}, \theta) \, p(z_{i+1} \mid z_i, \theta). \end{aligned}$$

Now the first term in each summand simplifies due to, you guessed it, the structure of the joint model,

$$p(y_{(i+1):I}, \mid z_i, z_{i+1}, \theta) = p(y_{(i+1):I}, \mid z_{i+1}, \theta).$$

Once we know z_{i+1} the following observations $y_{(i+1):I}$ do not depend on any of the previous latent states.

Substituting this simplification gives

$$\begin{aligned}
\beta_{i,z_i} &= \sum_{z_{i+1}} p(\tilde{y}_{(i+1):I}, \mid z_i, z_{i+1}, \theta) \, p(z_{i+1} \mid z_i, \theta) \\
&= \sum_{z_{i+1}} p(\tilde{y}_{(i+1):I}, \mid z_{i+1}, \theta) \, p(z_{i+1} \mid z_i, \theta) \\
&= \sum_{z_{i+1}} p(\tilde{y}_{i+1} \mid i+1, z_{i+1}, \theta) \, p(\tilde{y}_{(i+2):I}, \mid z_{i+1}, \theta) \, p(z_{i+1} \mid z_i, \theta).
\end{aligned}$$

That middle term, however, is just the β for the subsequent latent state,

$$p(\tilde{y}_{(i+2):I}, \mid z_{i+1}, \theta) = \beta_{i+1,z_{i+1}}!$$

Consequently we have a *backwards* recursion formula for the β_{i,z_i},

$$\begin{aligned}
\beta_{i,z_i} &= \sum_{z_{i+1}} p(\tilde{y}_{i+1} \mid i+1, z_{i+1}, \theta) \, p(\tilde{y}_{(i+2):I}, \mid z_{i+1}, \theta) \, p(z_{i+1} \mid z_i, \theta) \\
&= \sum_{z_{i+1}} p(\tilde{y}_{i+1} \mid i+1, z_{i+1}, \theta) \, \beta_{i+1,z_{i+1}} \, p(z_{i+1} \mid z_i, \theta) \\
&= \sum_{z_{i+1}} \omega_{i+1,z_{i+1}} \, \beta_{i+1,z_{i+1}} \, \Gamma_{i+1,z_i,z_{i+1}},
\end{aligned}$$

which we can also write in matrix notation,

$$\beta_i = \Gamma_{i+1} \cdot (\omega_{i+1} \circ \beta_{i+1}) = \Gamma_{i+1} \cdot (\mathrm{diag}(\omega_{i+1}) \cdot \beta_{i+1}).$$

The only remaining issue is how to compute β_{I,z_I} and initialize the backwards recursion. At the last step we have

$$\begin{aligned}
p(y_I \mid z_{I-1}, \theta) &= \sum_{z_I} p(y_I, z_I \mid z_{I-1}, \theta) \\
&= \sum_{z_I} p(y_I \mid z_I, z_{I-1}, \theta) \, p(z_I \mid z_{I-1}, \theta).
\end{aligned}$$

Recall, however, that conditioning on z_I blocks any coupling between z_{I-1} and y_I. Consequently

$$p(y_I \mid z_I, z_{I-1}, \theta) = p(y_I \mid I, z_I, \theta)$$

and

$$
\begin{aligned}
p(y_I \mid z_{I-1}, \theta) &= \sum_{z_I} p(y_I \mid z_I, z_{I-1}, \theta) \, p(z_I \mid z_{I-1}, \theta) \\
&= \sum_{z_I} p(y_I \mid I, z_I, \theta) \, p(z_I \mid z_{I-1}, \theta).
\end{aligned}
$$

Comparing this to the backwards recursion formula

$$
\beta_{I-1, z_{I-1}} = \sum_{z_I} p(\tilde{y}_I \mid I, z_I, \theta) \, \beta_{I, z_I} \, p(\tilde{y}_I \mid z_{I-1}, \theta)
$$

we see that we will get consistent results if and only if we set

$$
\beta_{I, z_I} = 1
$$

for all values of z_I.

Putting everything together we can evaluate the marginal posterior probabilities for the configuration of any latent state by first scanning forward through the latent states,

$$
\begin{aligned}
\alpha_0 &= \rho \\
&\ldots \\
\alpha_i &= \omega_i \circ \left(\Gamma_i^T \cdot \alpha_{i-1} \right) \\
&\ldots,
\end{aligned}
$$

scanning backwards through the latent states,

$$
\begin{aligned}
\beta_I &= \mathbf{1} \\
&\ldots \\
\beta_{i-1} &= \Gamma_i \cdot (\omega_i \circ \beta_i),
\end{aligned}
$$

and then finally multiplying and normalizing,

$$
p(z_i = k \mid \tilde{y}_{1:I}, \theta) = \frac{\alpha_{i,k} \, \beta_{i,k}}{\sum_{k'=1}^{K} \alpha_{i,k'} \, \beta_{i,k'}}.
$$

This combined procedure is known as the **forward-backward algorithm**. Comparing operations we can see the the forward-backward algorithm exhibits the same computational scaling as the forward algorithm alone. Once we've run the forward algorithm running the backward algorithm through all of the latent states will only double the computational cost.

3.2 Sampling Hidden States

The main limitation of marginal posterior probabilities for individual latent states is that, by construction, they project out any information about the couplings between different latent states. In particular sampling latent state configurations from individual marginal posterior probabilities will not in general give a realistic *trajectory* through the latent states. Without realistic latent state trajectories we cannot properly simulate joint observations.

We can always construct more useful posterior summaries that are sensitive to multiple latent states if we can generate exact samples from the joint latent state posterior distribution. In practice this requires an ancestral sampling procedure where we sample one latent state at a time given all of the previously sampled latent states.

Ancestral sampling then raises the question of the sequence in which we should sample the latent states. For example we might naively consider trying to sample the latent states in order,

$$
\begin{aligned}
\tilde{z}_1 &\sim p(z_1, \tilde{y}_{1:I}) \\
\tilde{z}_2 &\sim p(z_2 \mid \tilde{z}_1, \tilde{y}_{1:I} \theta) \\
&\ldots \\
\tilde{z}_i &\sim p(z_i \mid \tilde{z}_{i-1}, \tilde{y}_{1:I}, \theta).
\end{aligned}
$$

The immediate issue with this approach, however, is that each latent state needs to be informed by not just the proceeding observations but rather *all* of the observations. In particular we won't be able to generate proper posterior samples without first accumulating the needed information from subsequent latent states.

This suggests that we'll need to sweep through the latent states at least twice, once to aggregating information from all of the observations and then a second time to sample latent states. Inspired by the structure of the forward-backward algorithm let's consider sweeping forward first and then sampling latent states in reverse.

In order to sample posterior latent states in reverse we'll need to construct the conditional posterior distributions

$$
\begin{aligned}
p(z_{i-1} \mid z_i, y_{1:I}, \theta) &\propto p(z_{i-1}, z_i \mid y_{1:I}, \theta) \\
&\propto p(y_{1:I} \mid z_{i-1}, z_i, \theta) \, p(z_{i-1}, z_i \mid \theta) \\
&\propto p(y_{1:I} \mid z_{i-1}, z_i, \theta) \, p(z_i \mid z_{i-1}, \theta) \, p(z_{i-1} \mid \theta).
\end{aligned}
$$

We can simplify this by exploiting one last property of the joint model,

$$
p(y_{1:I} \mid z_{i-1}, z_i, \theta) = p(y_{1:(i-1)} \mid z_{i-1}, \theta) \, p(y_i \mid i, z_i, \theta) \, p(y_{(i+1):I} \mid z_i, \theta).
$$

This allows us to write the reverse conditional posterior probabilities as

$$\begin{aligned}
p(z_{i-1} \mid z_i, y_{1:I}, \theta) \propto\ & p(y_{1:I} \mid z_{i-1}, z_i, \theta) \\
& \cdot p(z_i \mid z_{i-1}, \theta)\, p(z_{i-1} \mid \theta) \\
\propto\ & p(y_{1:(i-1)} \mid z_{i-1}, \theta)\, p(y_i \mid i, z_i, \theta) \\
& \cdot p(y_{(i+1):I} \mid z_i, \theta)\, p(z_i \mid z_{i-1}, \theta)\, p(z_{i-1} \mid \theta) \\
\propto\ & \Big[p(y_{1:(i-1)} \mid z_{i-1}, \theta)\, p(z_{i-1} \mid \theta) \Big] \\
& \cdot p(y_i \mid i, z_i, \theta)\, p(y_{(i+1):I} \mid z_i, \theta)\, p(z_i \mid z_{i-1}, \theta) \\
\propto\ & p(y_{1:(i-1)}, z_{i-1} \mid \theta)\, p(y_i \mid i, z_i, \theta) \\
& \cdot p(y_{(i+1):I} \mid z_i, \theta)\, p(z_i \mid z_{i-1}, \theta).
\end{aligned}$$

In terms of the forward-backward algorithm notation this becomes

$$\begin{aligned}
p(z_{i-1} \mid z_i, \tilde{y}_{1:I}, \theta) \propto\ & p(\tilde{y}_{1:(i-1)}, z_{i-1} \mid \theta)\, p(\tilde{y}_i \mid i, z_i, \theta) \\
& \cdot p(\tilde{y}_{(i+1):I} \mid z_i, \theta)\, p(z_i \mid z_{i-1}, \theta) \\
\propto\ & \alpha_{i-1, z_{i-1}}\, \omega_{i, z_i} \\
& \cdot \beta_{i, z_i}\, \Gamma_{i, z_i z_{i-1}},
\end{aligned}$$

or in matrix notation,

$$\mathbf{q}_{i-1} \propto \omega_{i,z_i}\, \beta_{i,z_i} \left(\alpha_{i-1} \circ \gamma_{i,z_i} \right),$$

where γ_{i,z_i} is the z_ith column of the transition matrix Γ_i.

Critically the conditional posterior probabilities for a given observation,

$$p(z_{i-1} \mid z_i, \tilde{y}_{1:I}, \theta)$$

are straightforward to compute once we've run the forward algorithm to completion and the backwards algorithm backwards to the active latent state. After completing the forward algorithm we can sample a final latent state,

$$\begin{aligned}
\mathbf{r} &= \alpha_I \\
\lambda &= \frac{\mathbf{r}}{\sum_k r_k} \\
\tilde{z}_I &\sim \text{categorical}\left(\lambda\right).
\end{aligned}$$

At this point we start running the backwards algorithm, using each new β_i and the previously calculated α_{i-1} to sample a new latent state $\tilde{z}_{i-1}$ given the previously sampled latent state $\tilde{z}_i$,

$$\begin{aligned}
\mathbf{r} &= \omega_{i,z_i}\, \beta_{i,z_i} \left(\alpha_{i-1} \circ \gamma_{i,z_i} \right) \\
\lambda &= \frac{\mathbf{r}}{\sum_k r_k} \\
\tilde{z}_{i-1} &\sim \text{categorical}\left(\lambda\right).
\end{aligned}$$

Compared to evaluating α_{i-1} and β_i the cost of multiplying and normalizing is negligible. Consequently sampling joint latent states introduces very little additional cost once if we're already running the forward-backward algorithm.

4 Robust Implementations of Hidden Markov Models

As elegant as the forward and forward-backward algorithms are they are unfortunately not quite as straightforward to implement in practice as we might hope. The problem is that the forward and backward recursions are *multiplicative*; each α_i and β_i is given by the product of many terms and consequently are prone to numerical underflow on computers.

One way to avoid numerical issues is to work on the log scale with the `log-sum-exp` function. For example instead of computing

$$\alpha_{i,k} = \omega_{i,k} \sum_{k'=1}^{K} \Gamma_{i,k'k} \, \alpha_{i-1,k'}$$

we could compute

$$\begin{aligned}
\log(\alpha_{i,k}) &= \log\left(\omega_{i,k} \sum_{k'=1}^{K} \Gamma_{i,k'k} \, \alpha_{i-1,k'} \right) \\
&= \log\left(\omega_{i,k}\right) + \log\left(\sum_{k'=1}^{K} \Gamma_{i,k'k} \, \alpha_{i-1,k'} \right) \\
&= \log\left(\omega_{i,k}\right) + \log\left(\sum_{k'=1}^{K} \exp\left(\log(\Gamma_{i,k'k}) + \log(\alpha_{i-1,k'}) \right) \right) \\
&= \quad \log\left(\omega_{i,k}\right) \\
&\quad + \text{log-sum-exp}\Big(\log(\Gamma_{i,1k}) + \log(\alpha_{i-1,1}), \\
&\qquad\qquad \ldots, \\
&\qquad\qquad \log(\Gamma_{i,Kk}) + \log(\alpha_{i-1,K}) \Big)
\end{aligned}$$

Similarly in the backwards pass we could compute

$$\begin{aligned}
\log(\beta_{i,k}) &= \log \left(\sum_{k'=1}^{K} \omega_{i+1,k'} \, \beta_{i+1,k'} \, \Gamma_{i+1,kk'} \right) \\
&= \log \left(\sum_{k'=1}^{K} \exp \bigg(\log(\omega_{i+1,k'}) + \log(\beta_{i+1,k'}) + \log(\Gamma_{i+1,kk'}) \bigg) \right) \\
&= \text{log-sum-exp}\bigg(\log(\omega_{i+1,1}) + \log(\beta_{i+1,1}) + \log(\Gamma_{i+1,k1}), \\
&\qquad \ldots, \\
&\qquad \log(\omega_{i+1,K}) + \log(\beta_{i+1,K}) + \log(\Gamma_{i+1,kK}) \bigg)
\end{aligned}$$

While numerically stable, the `log-sum-exp` function is unfortunately expensive. Introducing a `log-sum-exp` function at each iteration of the forward-backward algorithm can substantially increase computational cost.

Another way that we can avoid numerical underflow is to repeatedly *rescale* the α_i and β_i at each iteration and then correct for the rescaling whenever we use them in a calculation.

Consider, for example, the scaled forward recursion formula

$$\begin{aligned}
a_i &= \omega_i \circ \left(\Gamma_i^T \cdot \alpha'_{i-1} \right) \\
\nu_i &= \max(a_i) \\
\alpha'_i &= \frac{a_i}{\nu_i}.
\end{aligned}$$

After i iterations the unscaled α_i and scaled α'_i will be proportional to each other,

$$\alpha_i = \left(\prod_{i'=1}^{i} \nu_{i'} \right) \alpha'_i.$$

Consequently we can always recover α_i from the more numerically stable α'_i.

That said the product

$$\prod_{i'=1}^{i} \nu_{i'}$$

will itself be prone to numerical underflow. Fortunately we can avoid this by working on the log scale,

$$\alpha_i = \exp \left(\sum_{i'=1}^{i} \log(\nu_{i'}) \right) \alpha'_i.$$

Note that this requires evaluating only one logarithm function at each forward iteration and a single exponential function after all I iterations, much less expensive than repeated `log-sum-exp` evaluations.

Dynamic rescaling is also useful for ensuring numerical stability when computing each β_i in the backward recursion. Here, however, we don't even have to keep track of the running normalization so long as we always normalize the marginal and conditional latent probabilities at each iteration.

The `log-sum-exp` implementation of hidden Markov models is popular in many `Stan` tutorials, but the dedicated hidden Markov model functions in the `Stan` modeling language are implemented using dynamic rescaling, which was shown to be just as robust but substantially more efficient. We will implement both approaches in Section 7.

5 Hidden Markov Modeling Techniques

The basic construction and implementation of hidden Markov models is more general than they might first appear to be. With some care we can incorporate a diversity of interesting structure into hidden Markov models.

5.1 Modeling The Initial State

When marginalizing out the latent states all of the latent state probabilities are derived except for the probabilities of the initial latent state. These have to be modeled and inferred along with the transition matrices and configuration of the component observational models.

In some applications we might use hidden Markov models to capture the dynamics of a system evolving from an explicit initialization. Here we can use our domain expertise about that initialization to directly inform an appropriate prior model for the initial state probabilities. Absent any readily available domain expertise a uniform prior model over the simplex of possible initial state probabilities is not unreasonable, especially as a starting default to be improved upon as needed.

For some applications, however, we might be interested in modeling dynamics that have evolved for so long that the initialization is at best ambiguous. In these applications we need the initial state probabilities to model **equilibrium** behavior. To formally define equilibrium dynamics we'll need to take a little detour into linear algebra.

The elements of each transition matrix are defined by the individual transition probabilities between each pair of latent states,

$$\Gamma_{i,k'k} = p(z_i = k \mid z_{i-1} = k', \theta).$$

In particular the elements of Γ_i are all bounded between zero and one,

$$0 \le \Gamma_{i,k'k} \le 1,$$

and the elements in each row sum to one,

$$\sum_{k=1}^{K} \Gamma_{i,k'k} = \sum_{k=1}^{K} p(z_i = k \mid z_{i-1} = k', \theta) = 1.$$

Matrices with these properties are more generally known as **row stochastic matrices** or **right stochastic matrices**

These properties endow the transpose of stochastic matrices with particularly interesting *eigenstructure* (Papoulis and Pillai (2002)). For example the magnitude of the eigenvalues of any right stochastic matrix are always bounded by one, and at least one left eigenvector will saturate this bound,

$$\mathbf{v}^T \cdot S = \mathbf{v}^T \cdot 1 = \mathbf{v}^T,$$

or equivalently

$$S^T \cdot \mathbf{v} = 1 \cdot \mathbf{v} = \mathbf{v}.$$

Consequently the repeated application of the transpose of a fixed right stochastic matrix to *any* vector $\mathbf{u}$ will always converge to one of those leading eigenvectors,

$$\lim_{i \rightarrow \infty} \left(S^T \right)^i \cdot \mathbf{u} = \mathbf{v}.$$

If the elements of S^m are all non-zero for some finite $m \in \mathbb{N}$ then the leading eigenvector, and hence this limiting behavior, will be unique. Moreover the elements of this unique leading eigenvector will be real-valued and positive so that they can normalized into probabilities.

All of this linear algebra implies that any latent dynamics modeled with a well-behaved, homogeneous transition probabilities will eventually converge to some stable, equilibrium behavior regardless of the precise initial state. The equilibrium probabilities of the latent states are completely determined up to normalization by the leading left eigenvector of the transition matrix.

In practice we can model a system already in equilibrium by *deriving* the initial state probabilities from the eigendecomposition of the transition matrix. If the configuration of the transition matrix is uncertain then so to will be its eigendecomposition, and hence the derived initial states.

Our ability to implement these calculations in practice, however, will depend on the available linear algebraic tools. For example `Stan` currently implements eigendecompositions for only general complex matrices and symmetric real-valued matrices, but not stochastic matrices. In theory we can use a general eigensolver, but in practice identifying an eigenvector with *exactly* unit eigenvalue and removing any complex number artifacts is at best awkward for floating point arithmetic.

Finally it's worth emphasizing that equilibrium is a well-defined concept for only homogeneous transition matrices. If the transition probabilities change between different pairs of neighboring latent states then the realized dynamics will always depend on the behavior of the initial latent states.

5.2 Modeling Multiple Latent Sequences

Some systems are best modeled not with a single sequence of latent states,

$$z_1, \ldots, z_i, \ldots, z_I,$$

but rather multiple sequences of latent states (Figure 5),

$$\begin{aligned} z^1 &= (z^1_1, \ldots, z^1_i, \ldots, z^1_I) \\ z^2 &= (z^2_1, \ldots, z^2_i, \ldots, z^2_I), \end{aligned}$$

with coupled dynamics

$$p(z^m_i \mid z^1_{i-1}, \ldots, z^M_{i-1}).$$

In theory we could derive generalized forward and forward-backward algorithms for multiple latent states. Alternatively we can combine multiple latent states together into a single joint latent state and use the standard forward and forward-backward algorithms.

One *systematic* way to combine two finite latent states together requires some more linear algebra, this time in the form of an operation known as a **Kronecker product**, **matrix direct product**, or **tensor product** (Curtis (1993)). The tensor product maps M latent states z^m_i into a single joint state consisting of all possible n-tuples of the component state elements,

$$z_i = z^1_i \otimes \ldots \otimes z^M_i = \otimes^M_{m=1} z^m_i.$$

For example the tensor product of the two latent states

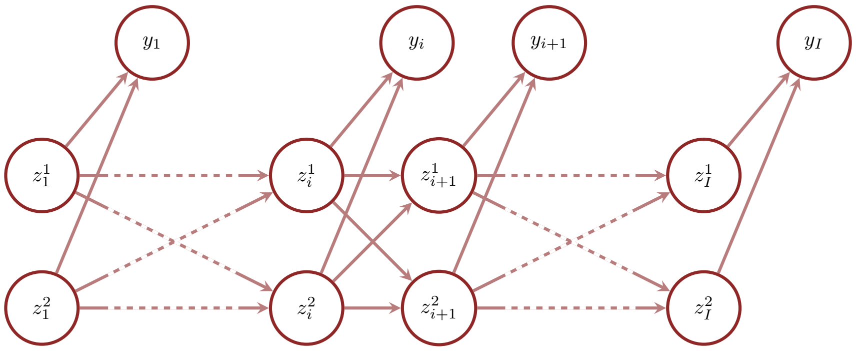
$$z^1_i \in (1, 2, 3)$$

and

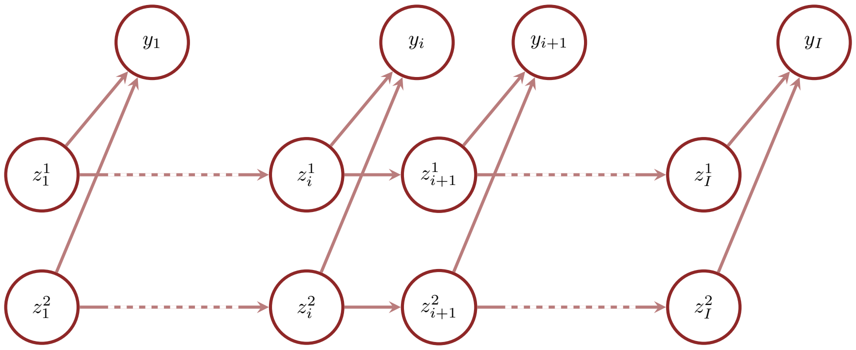
$$z^2_i \in (1, 2)$$

is given by the ordered pairs

$$z_i = z^1_i \otimes z^2_i \in (1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2).$$



(a)



(b)

Figure 5: Hidden Markov models can accommodate multiple sequences of latent states. (a) In general the latent states evolve together but in some applications they might (b) evolve independently of each other.

When working with these joint states in practice it is often helpful to relabel them with sequential integers,

$$
\begin{aligned}
(1,1) &\to 1 \\
(1,2) &\to 2 \\
(2,1) &\to 3 \\
(2,2) &\to 4 \\
(3,1) &\to 5 \\
(3,2) &\to 6.
\end{aligned}
$$

In general if each component state features K_m elements then then tensor product will feature

$$\prod_{m=1}^{M} K_m$$

total elements.

Implementing a hidden Markov model with respect to this joint latent state then requires an expanded transition matrix. Sometimes it is more productive to interpret the transition matrix with respect to the joint latent state,

$$\Gamma_{i,k'k} = p(z_i = k \mid z_{i-1} = k', \theta),$$

and sometimes it is more useful to reason about the transition matrix in terms of the component latent states,

$$
\begin{aligned}
&\Gamma_{i,k'_1 \cdots k'_M k_1 \cdots k_M} \\
&\qquad = p(z_i^1 = k_1, \ldots, z_i^M = k_M \mid z_{i-1}^1 = k'_1, \ldots, z_{i-1}^M = k'_M, \theta).
\end{aligned}
$$

When the two states evolve independently of each other,

$$
\begin{aligned}
&p(z_i^1 = k_1, \ldots, z_i^M = k_M \mid z_{i-1}^1 = k'_1, \ldots, z_{i-1}^M = k'_M, \theta) \\
&\qquad\qquad = \prod_{m=1}^{M} p(z_i^m = k_m \mid z_{i-1}^m = k'_m, \theta),
\end{aligned}
$$

we can write the joint transition matrix in terms of component transition matrices. More formally if

$$\Gamma_{i,k'k}^m = p(z_i^m = k \mid z_{i-1}^m = k, \theta)$$

then the joint transition matrix for the joint latent states is given by the tensor product

$$\Gamma_i = \Gamma_i^1 \otimes \ldots \otimes \Gamma_i^M = \otimes_{m=1}^{M} \Gamma_i^m.$$

For example consider two component latent states with only two elements each,

$$z_i^1 \in (1, 2)$$

and

$$z_i^2 \in (1, 2).$$

Their tensor product is given by four ordered pairs

$$z_i = z_i^1 \otimes z_i^2 \in (1,1), (1,2), (2,1), (2,2),$$

which we can encode as

$$\begin{aligned} (1,1) &\rightarrow 1 \\ (1,2) &\rightarrow 2 \\ (2,1) &\rightarrow 3 \\ (2,2) &\rightarrow 4. \end{aligned}$$

If each of these component latent states evolves independently of each other with the transition matrices

$$\begin{aligned} \Gamma^1_{i,k'k} &= p(z_i^1 = k \mid z_{i-1}^1 = k, \theta) \\ \Gamma^2_{i,k'k} &= p(z_i^2 = k \mid z_{i-1}^2 = k, \theta). \end{aligned}$$

then we can write the joint transition matrix as

$$\begin{aligned} \Gamma_i &= \Gamma_i^1 \otimes \Gamma_i^2 \\ &= \begin{pmatrix} \Gamma^1_{i,11} \, \Gamma^2 & \Gamma^1_{i,12} \, \Gamma^2 \\ \Gamma^1_{i,21} \, \Gamma^2 & \Gamma^1_{i,22} \, \Gamma^2 \end{pmatrix} \\ &= \begin{pmatrix} \Gamma^1_{i,11} \, \Gamma^2_{i;11} & \Gamma^1_{i,11} \, \Gamma^2_{i;12} & \Gamma^1_{i,12} \, \Gamma^2_{i;11} & \Gamma^1_{i,12} \, \Gamma^2_{i;12} \\ \Gamma^1_{i,11} \, \Gamma^2_{i;21} & \Gamma^1_{i,11} \, \Gamma^2_{i;22} & \Gamma^1_{i,12} \, \Gamma^2_{i;21} & \Gamma^1_{i,12} \, \Gamma^2_{i;22} \\ \Gamma^1_{i,21} \, \Gamma^2_{i;11} & \Gamma^1_{i,21} \, \Gamma^2_{i;12} & \Gamma^1_{i,22} \, \Gamma^2_{i;11} & \Gamma^1_{i,22} \, \Gamma^2_{i;12} \\ \Gamma^1_{i,21} \, \Gamma^2_{i;21} & \Gamma^1_{i,21} \, \Gamma^2_{i;22} & \Gamma^1_{i,22} \, \Gamma^2_{i;21} & \Gamma^1_{i,22} \, \Gamma^2_{i;22} \end{pmatrix}. \end{aligned}$$

Note that the tensor product of vectors and matrices is not strictly commutative. Changing the order of the components changes the organization of the ordered pairs which in turn permutes the elements of the joint states and joint transition matrix. In practice this just means that we have to be careful to organize everything consistently.

More generally we have to reason through all

$$\left(\prod_{m=1}^{M} K_m \right)^2$$

of the possible couplings between the neighboring component states,

$$\begin{aligned}
&p(z_i = k \mid z_{i-1} = k', \theta) \\
&\qquad \equiv p(z_i^1 = k_1, \dots, z_i^M = k_M \mid z_{i-1}^1 = k_1', \dots, z_{i-1}^M = k_M', \theta).
\end{aligned}$$

Any lack of conditional dependence, with the evolution of any component latent states depending on only some but not all of the component latent states, makes this task much more manageable.

5.3 Modeling Partially Observed Latent States

Unobserved latent states are not uncommon in hidden Markov models, especially when the models span many latent states. How best to model partially observed latent states depends on why the observations are incomplete.

For example if the probability that a particular latent state is unobserved is independent of any of the other model configuration variables then we can model partial observations with a straightforward marginalization. Formally if the ith latent state is unobserved then we can model the remaining observations with

$$\begin{aligned}
p(z_{1:I}, &y_{1:(i-1)}, y_{(i+1):I} \mid \theta) \\
&= \int \mathrm{d}y_i p(z_{1:I}, y_{1:I} \mid \theta) \\
&= \int \mathrm{d}y_i p(z_1) \, p(y_1 \mid 1, z_1, \theta) \prod_{i'=2}^{I} p(z_{i'} \mid z_{i'-1}, \theta) \, p(y_{i'} \mid i', z_{i'}, \theta) \\
&= \quad p(z_1) \, p(y_1 \mid 1, z_1, \theta) \\
&\quad \cdot \left[\prod_{i'=1}^{i-1} p(z_{i'} \mid z_{i'-1}, \theta) \, p(y_{i'} \mid i', z_{i'}, \theta) \right] \\
&\quad \cdot p(z_i \mid z_{i-1}, \theta) \int \mathrm{d}y_i p(y_i \mid i, z_i, \theta) \\
&\quad \cdot \left[\prod_{i'=i+1}^{I} p(z_{i'} \mid z_{i'-1}, \theta) \, p(y_{i'} \mid i', z_{i'}, \theta) \right] \\
&= \quad p(z_1) \, p(y_1 \mid 1, z_1, \theta) \\
&\quad \cdot \left[\prod_{i'=1}^{i-1} p(z_{i'} \mid z_{i'-1}, \theta) \, p(y_{i'} \mid i', z_{i'}, \theta) \right] \\
&\quad \cdot p(z_i \mid z_{i-1}, \theta) \\
&\quad \cdot \left[\prod_{i'=i+1}^{I} p(z_{i'} \mid z_{i'-1}, \theta) \, p(y_{i'} \mid i', z_{i'}, \theta) \right].
\end{aligned}$$

Conveniently this is equivalent to setting the corresponding observational model evaluations to one,

$$p(\tilde{y}_i \mid i, z_i, \theta) = 1,$$

or equivalently the log observational model evaluations to zero,

$$\log \circ p(\tilde{y}_i \mid i, z_i, \theta) = 0,$$

for each z_i. In particular we can account for any unobserved latent states without having to modify the forward and forward-backward algorithms by simply replacing the component observational model evaluations with ones wherever necessary.

If we are not interested in inferring the behavior of unobserved latent states then we can also model the partially observed data by marginalizing out the unobserved latent states entirely. For example we can model a gap of observations along the latent states

$$z_i, z_{i+1}, \ldots, z_{i+\delta}$$

by removing them and then replacing the transition matrix Γ_i with the matrix product (Figure 6)

$$\prod_{i'=i}^{i+\delta+1} \Gamma_i.$$

After re-indexing the remaining latent states we can run the forward and forward-backward algorithms using the collapsed transition matrix. Really the only downside to this approach is that the collapsed hidden Markov model will be non-homogeneous even if the full hidden Markov model is homogeneous.

Modeling more complicated mechanisms for unobserved latent states is, unsurprisingly, more complicated. A general approach that can be useful in practice is to introduce a auxiliary sequence of known, binary states

$$w_1, \ldots, w_i, \ldots, w_I$$

that indicates whether or not a latent state is complemented with any observations. For example we might define $w_i = 0$ when there are no observations at the ith state, and $w_i = 1$ when there are.

When $w_i = 1$ the component observational models behave as they would without any missingness,

$$p(y_i \mid i, z_i, w_i = 0, \theta) = p(y_i \mid i, z_i, \theta)$$

and when $w_i = 0$ they all return one,

$$p(y_i \mid i, z_i, w_i = 1, \theta) = 1.$$

Because the w_i are known the implementation of the forward and forward-backward algorithms proceed as before with these modified observational model outputs for each i.

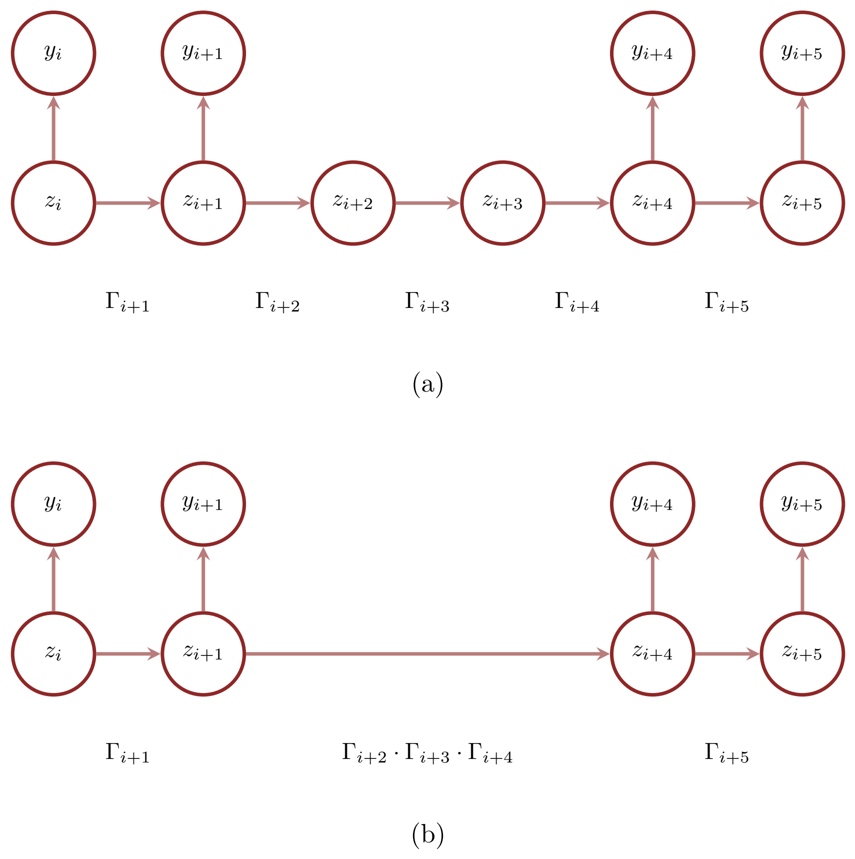


Figure 6: (a) Gaps of latent states without any observations can be (b) collapsed into a single transition given by the product of the transitions between the observed states.

The mechanism for which latent states are observed and which are not is then encoded in the structure of the transition probabilities for these auxiliary states. In general the transition from any w_{i-1} to any w_i can depend on not only i and w_{i-1} but also z_{i-1} as well as any other model configuration variables,

$$\Omega_{i,j'jk} = p(w_i = j' \mid w_{i-1} = j, z_{i-1} = k, \theta).$$

Specifically the coupling of w_i to z_{i-1} and θ allows our inferences to account for any selection bias in partially observed data.

6 Degeneracies of Hidden Markov Models

As discussed in the mixture modeling chapter basic mixture observational models are vulnerable to degenerate inferences when the component observational models include redundant behaviors. By coupling the latent states together hidden Markov models can amplify these problems if we are not careful.

For example the redundancy of hidden Markov models with fully exchangeable component observational models at each state prevents data from unambiguously informing individual latent states. Consequently the latent state dynamics will be wildly uncertain.

That said the coupling between neighboring latent states can actually reduce uncertainties is less redundant component observational models.

For example if one latent state is complemented by only weakly informative observations then inferences for the corresponding component probabilities using that local data alone will be poor. More informative observations at the previous and following latent states, however, can provide less uncertain inferences for those neighboring component probabilities. When the behavior of the transition matrices is also well informed these neighboring inferences will inform the behavior of the intermediate state far beyond what the local observations could do on their own.

In general the more rigid the latent state dynamics are the more the constraints from each local observation will propagate across the entire sequence of latent states. We can take advantage of this rigidity, however, only when we have strong inferences for the transition probabilities.

Without informative observations at enough pairs of neighboring states we may not be able to learn much about the behavior of the transition matrices. This is especially problematic for non-homogeneous hidden Markov models models which are particularly data hungry.

7 Demonstrations

To anchor all of this abstract math into a more practical context let's work through some implementations of hidden Markov models that demonstrate not only the basic features but also some of the more sophisticated directions that we can take the methodology.

7.1 Setup

First and foremost we have to setup our local `R` environment.

```r
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, 1))

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

```r
util <- new.env()
source('mcmc_analysis_tools_rstan.R', local=util)
source('mcmc_visualization_tools.R', local=util)
```

7.2 Comparing and Contrasting Implementations

Let's start with a relatively simple exercise that compares some of the basic implementation strategies.

7.2.1 Simulate Data

We'll simulate data assuming `K = 3` component observational models across `I = 100` states.

```r
I <- 100
K <- 3
```

To avoid too much complexity we'll assume a homogeneous transition matrix between all of the neighboring latent states and the same component observational models at each latent state.

```
simu <- stan(file="stan_programs/simu_data.stan",
             algorithm="Fixed_param", seed=194838,
             data=list("I" = I), iter=1, chains=1, refresh=0)
```

```
simu_samples <- util$extract_expectand_vals(simu)

y_names <- sapply(1:I, function(i) paste0('y[', i, ']'))
y <- sapply(y_names, function(name) simu_samples[[name]][1,1])

z_names <- sapply(1:I, function(i) paste0('z[', i, ']'))
z <- sapply(z_names, function(name) simu_samples[[name]][1,1])
```

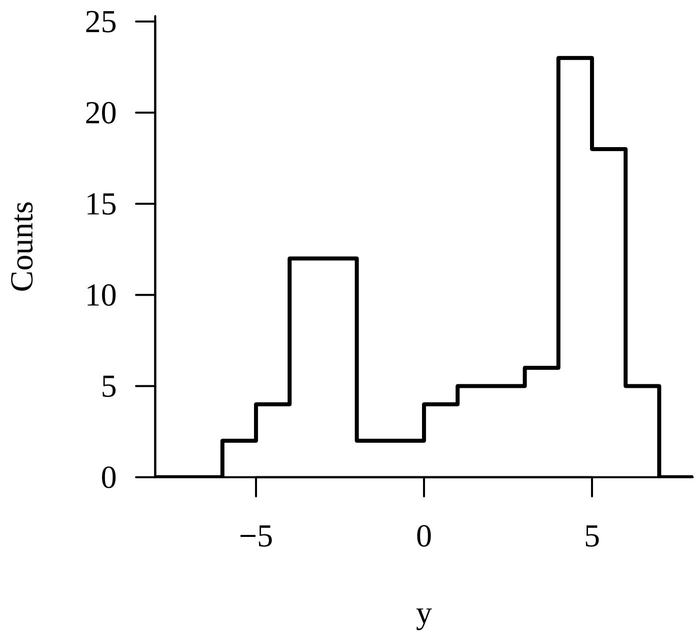
```
data <- list("I" = I, "y" = y, "K" = K)
```

7.2.2 Exploratory Data Analysis

There are a few summary statistics that are particularly natural to the structure assumed by a hidden Markov model.

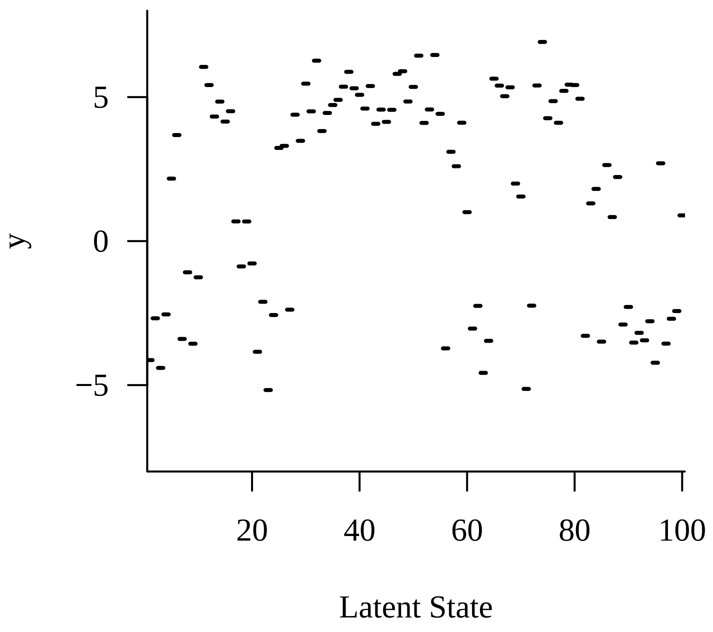
A histogram of all observations collapses the latent dynamics, allowing us to focus on the aggregate behavior of the component observational models. This summary is particularly informative when the component observational models are the same for all latent states.

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

util$plot_line_hist(data$y, -8, 8, 1, xlab="y")
```



We can also isolate the latent dynamics as much as possible by plotting the observations against the corresponding latent state indices. This is often referred to as an **empirical trajectory**.

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

plot(1, type="n",
     xlab="Latent State", xlim=c(0.5, data$I + 0.5),
     ylab="y", ylim=c(-8, 8))

for (i in 1:data$I) {
  lines(c(i - 0.5, i + 0.5), rep(data$y[i], 2),
        col="black", lwd=2)
}
```



7.2.3 Hidden Markov Model Log-Sum-Exp Implementation

Let's first consider a hidden Markov model using the `log-sum-exp` implementation of the forward and forward-background algorithms.

```
fit <- stan(file="stan_programs/hmm_log_sum_exp.stan",
            data=data, seed=4938483,
            warmup=1000, iter=2024, refresh=0)
```

Hamiltonian Monte Carlo doesn't appear to have any issues quantifying the posterior distribution.

```
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.
```

```
samples1 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples1,
                                       c('mu', 'sigma',
                                         'rho', 'gamma'),
                                               check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

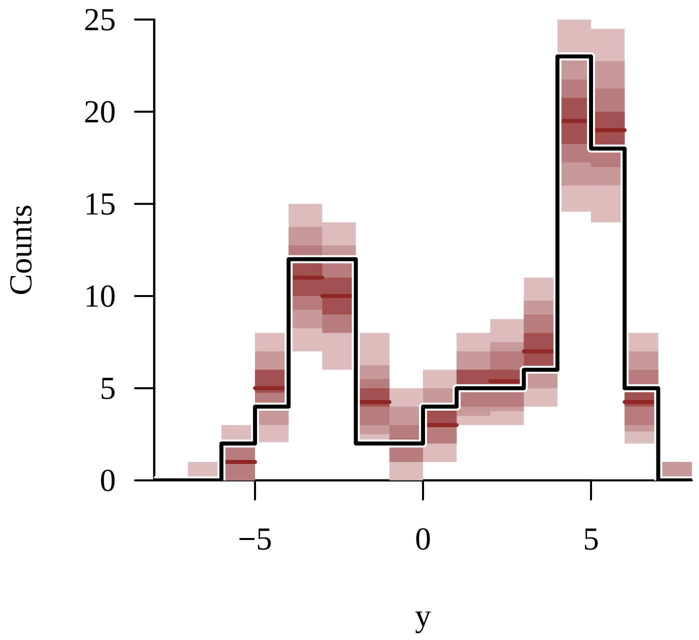
```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

Having modeled latent dynamics we can consider not only the histogram summary statistic but also the empirical trajectory. In both cases there is no appreciable retrodictive tension that would indicate any inadequacy of our modeling assumptions.

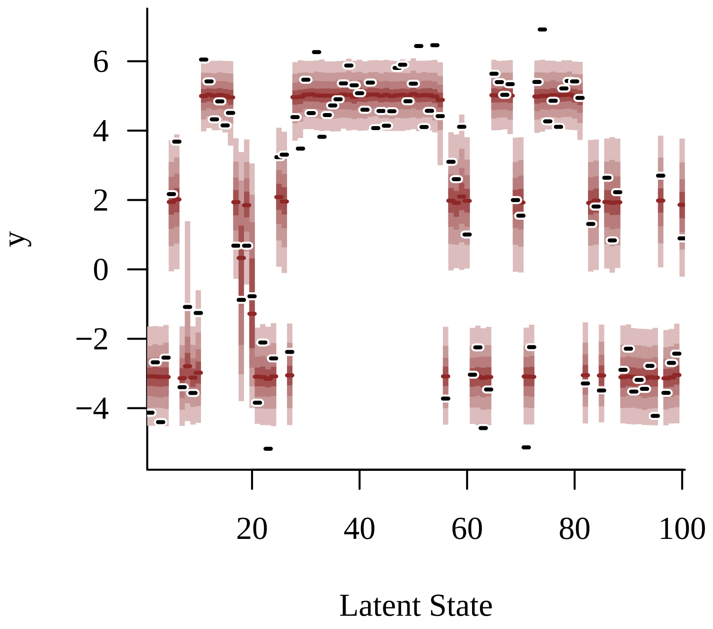
```
par(mfrow=c(1, 1))

util$plot_hist_quantiles(samples1, 'y_pred', -8, 8, 1,
                         baseline_values=data$y, xlab='y')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 8 predictive values (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 75 predictive values (0.0%) fell above the binning.
```

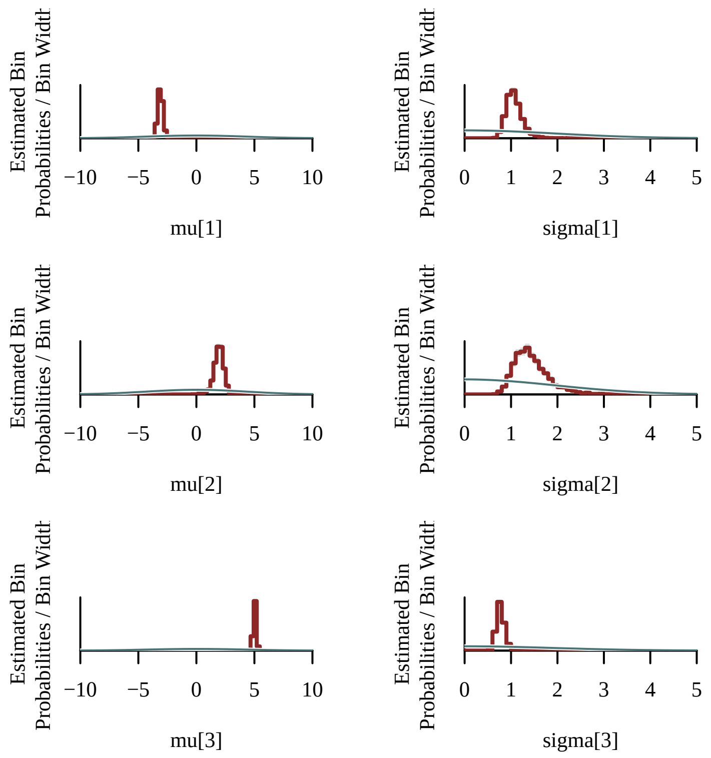


```r
par(mfrow=c(1, 1))

names <- sapply(1:data$I, function(i) paste0('y_pred[', i, ']'))
util$plot_disc_pushforward_quantiles(samples1, names,
                                     baseline_values=data$y,
                                     xlab="Latent State", ylab="y")
```



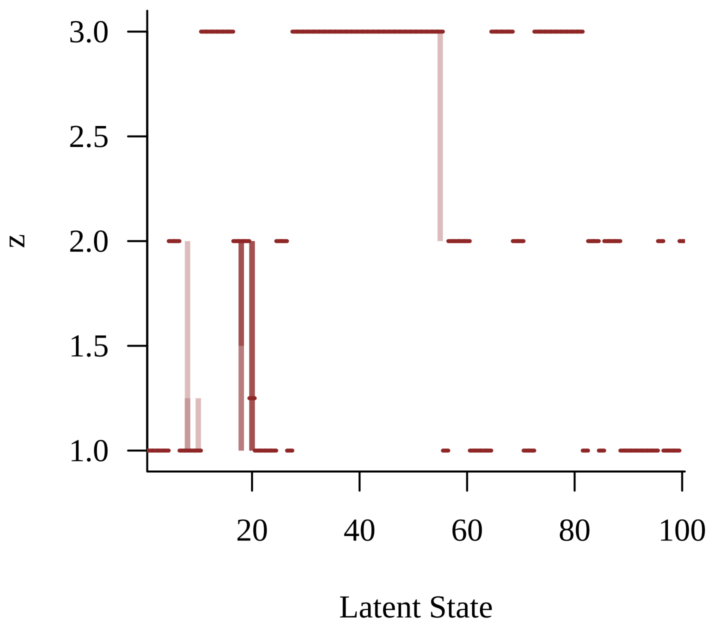
Consequently we can have some confidence in the faithfulness of our posterior inferences. This includes the inferred behavior of the component observational models.

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

for (k in 1:data$K) {
  name <- paste0('mu[', k, ']')
  util$plot_expectand_pushforward(samples1[[name]], 75,
                                  display_name=name, flim=c(-10, 10))

  xs <- seq(-10, 10, 0.1)
  ys <- dnorm(xs, 0, 10 / 2.32);
  lines(xs, ys, lwd=2, col="white")
  lines(xs, ys, lwd=1, col=util$c_mid_teal)

  name <- paste0('sigma[', k, ']')
  util$plot_expectand_pushforward(samples1[[name]], 50,
                                  display_name=name, flim=c(0, 5))

  xs <- seq(0, 5, 0.1)
  ys <- 2 * dnorm(xs, 0, 5 / 2.57);
  lines(xs, ys, lwd=2, col="white")
  lines(xs, ys, lwd=1, col=util$c_mid_teal)
}
```



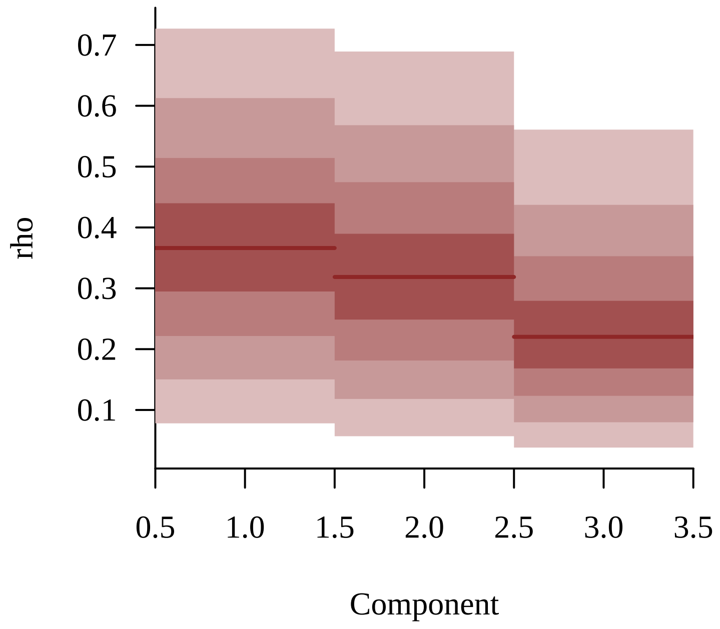
We can also make inferences about the behavior of the latent states.

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

names <- sapply(1:data$I, function(i) paste0('z[', i, ']'))
util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Latent State", ylab="z")
```

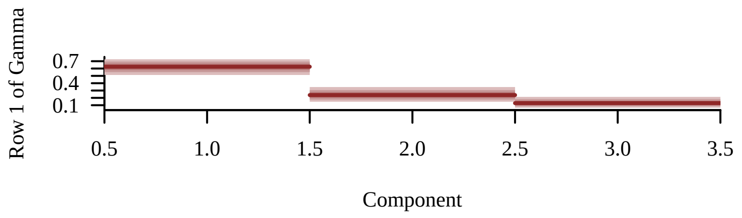


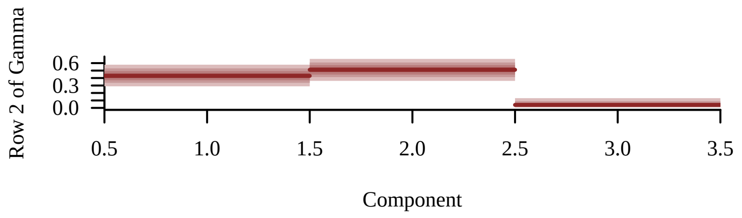
Alternatively we can look at the inferred initial state and transition probabilities that drive the evolution of the latent states.

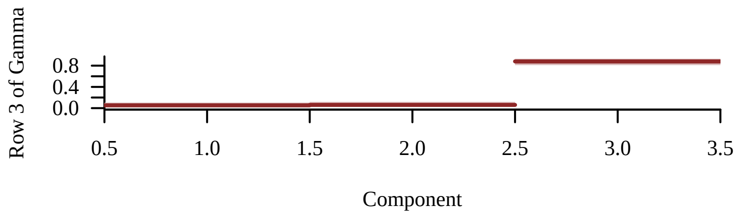
```
par(mfrow=c(1, 1))

names <- sapply(1:data$K, function(kk) paste0('rho[', kk, ']'))
util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Component",
                                     ylab="rho")
```



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

for (k in 1:data$K) {
  names <- sapply(1:data$K,
                  function(kk) paste0('gamma[', k, ',', kk, ']'))
  util$plot_disc_pushforward_quantiles(samples1, names,
                                       xlab="Component",
                                       ylab=paste("Row", k, "of Gamma"))
}
```







7.2.4 Hidden Markov Model Dynamic Rescaling Implementation

We can also implement hidden Markov models with dynamical rescaling.

```r
fit <- stan(file="stan_programs/hmm_rescaled.stan",
            data=data, seed=4938483,
            warmup=1000, iter=2024, refresh=0)
```

No computational diagnostics have arisen with this new implementation.

```r
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.
```

```r
samples2 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples2,
                                       c('mu', 'sigma',
                                         'rho', 'gamma'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

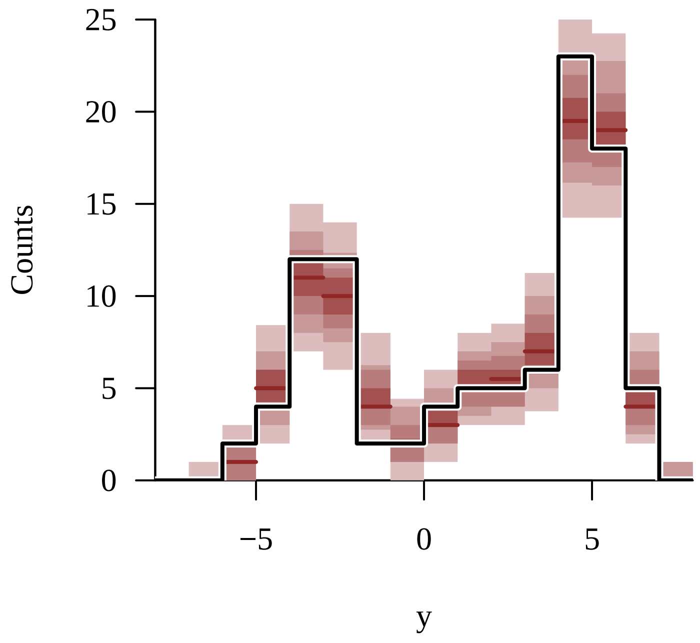
```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

Moreover the posterior retrodictive performance not only continues to be great but also is consistent with the performance we saw with the `log-sum-exp` model implementation.

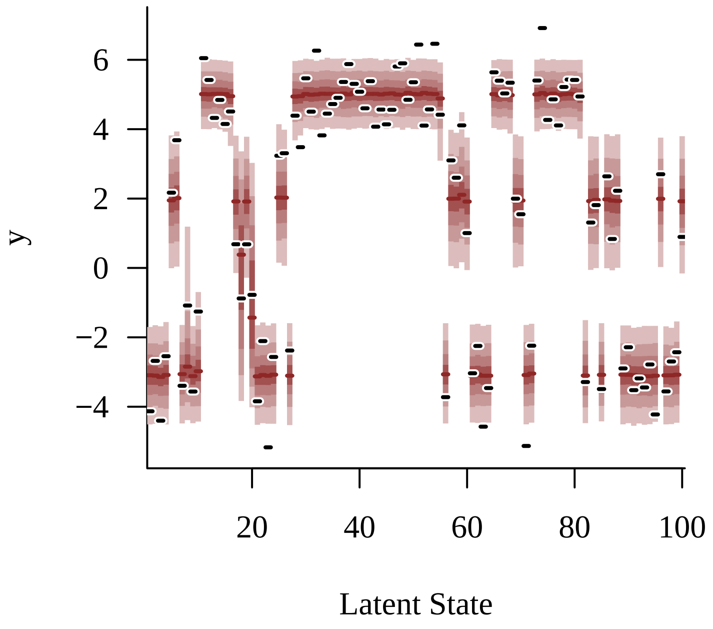
```
par(mfrow=c(1, 1))

util$plot_hist_quantiles(samples2, 'y_pred', -8, 8, 1,
                         baseline_values=data$y, xlab='y')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 5 predictive values (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 70 predictive values (0.0%) fell above the binning.
```

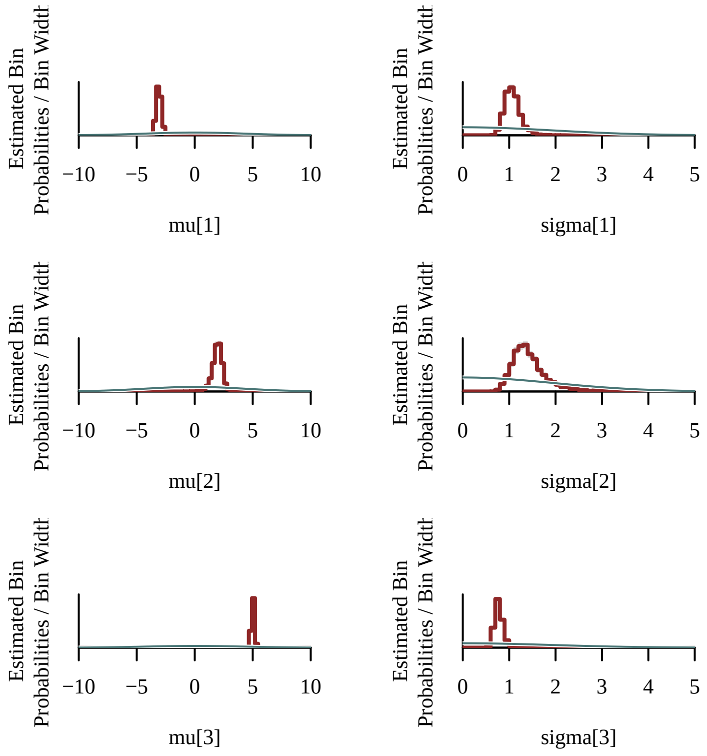


```
par(mfrow=c(1, 1))

names <- sapply(1:data$I, function(i) paste0('y_pred[', i, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     baseline_values=data$y,
                                     xlab="Latent State", ylab="y")
```

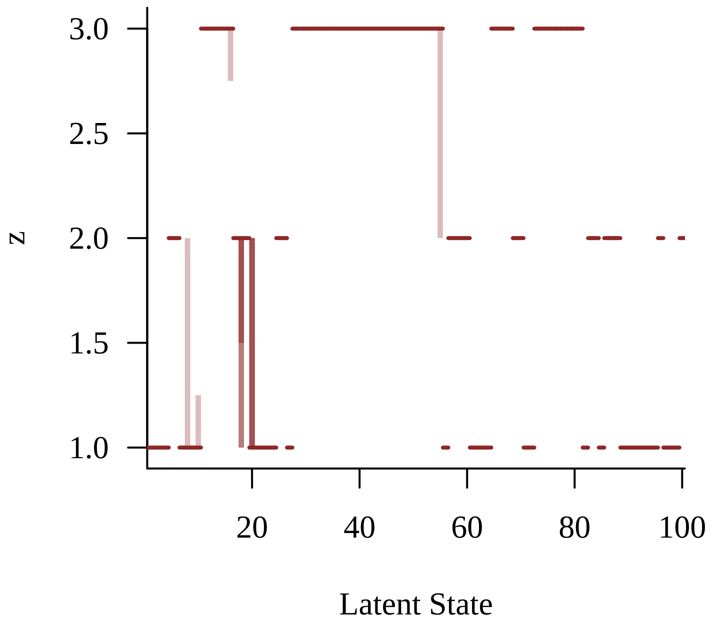


This is not surprising given the similarly of the posterior inferences.

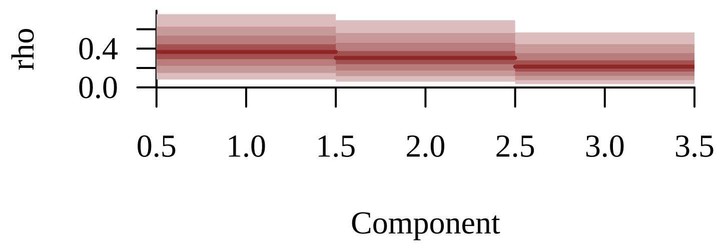
```
par(mfrow=c(3, 2))

for (k in 1:data$K) {
  name <- paste0('mu[', k, ']')
  util$plot_expectand_pushforward(samples2[[name]], 75,
                                  display_name=name, flim=c(-10, 10))

  xs <- seq(-10, 10, 0.1)
  ys <- dnorm(xs, 0, 10 / 2.32);
  lines(xs, ys, lwd=2, col="white")
  lines(xs, ys, lwd=1, col=util$c_mid_teal)

  name <- paste0('sigma[', k, ']')
  util$plot_expectand_pushforward(samples2[[name]], 50,
                                  display_name=name, flim=c(0, 5))
  xs <- seq(0, 5, 0.1)
  ys <- 2 * dnorm(xs, 0, 5 / 2.57);
  lines(xs, ys, lwd=2, col="white")
  lines(xs, ys, lwd=1, col=util$c_mid_teal)
}
```

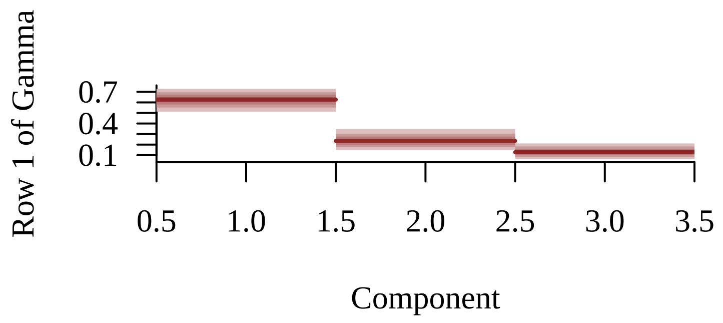


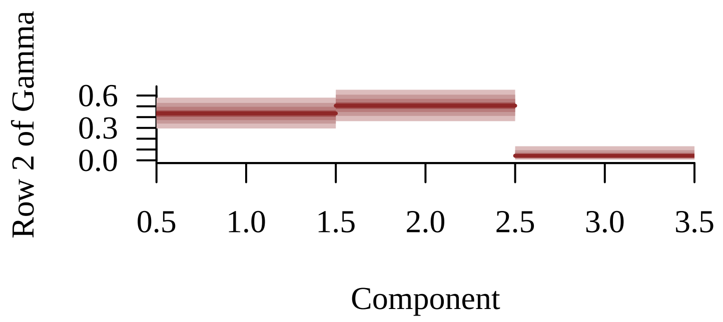
```
par(mfrow=c(1, 1))

names <- sapply(1:data$I, function(i) paste0('z[', i, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Latent State", ylab="z")
```

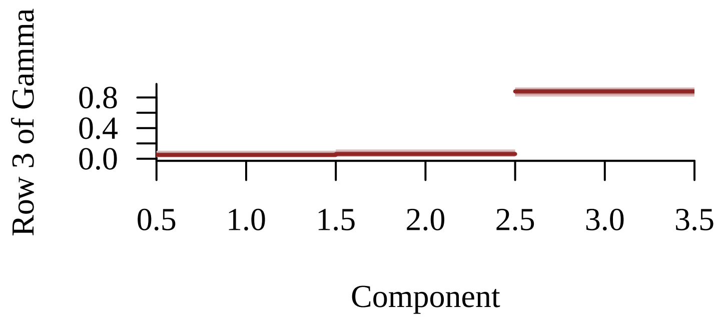


```
par(mfrow=c(2, 1))

names <- sapply(1:data$K, function(kk) paste0('rho[', kk, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Component",
                                     ylab="rho")

for (k in 1:data$K) {
  names <- sapply(1:data$K,
                  function(kk) paste0('gamma[', k, ',', kk, ']'))
  util$plot_disc_pushforward_quantiles(samples2, names,
                                       xlab="Component",
                                       ylab=paste("Row", k, "of Gamma"))
}
```









7.2.5 Hidden Markov Model Built-In Implementation

Finally we can use the hidden Markov model functions that are provided in the `Stan` Modeling Language. These functions take as inputs a matrix of component observational model evaluations, a homogeneous transition matrix, and initial state probabilities. They then run the forward and forward-backward algorithms using dynamic rescaling.

Using these built-in functions should give us equivalent results to the hand-coded implementation in the previous section. The advantage of the built-in functions is that because they are written in `C++` they should be a bit faster. On the other hand the limitation to homogeneous hidden Markov models can limit their scope.

```
fit <- stan(file="stan_programs/hmm_builtin.stan",
            data=data, seed=4938483,
            warmup=1000, iter=2024, refresh=0)
```

Welcomingly the computational 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.
```

```
samples3 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples3,
                                       c('mu', 'sigma',
                                         'rho', 'gamma'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

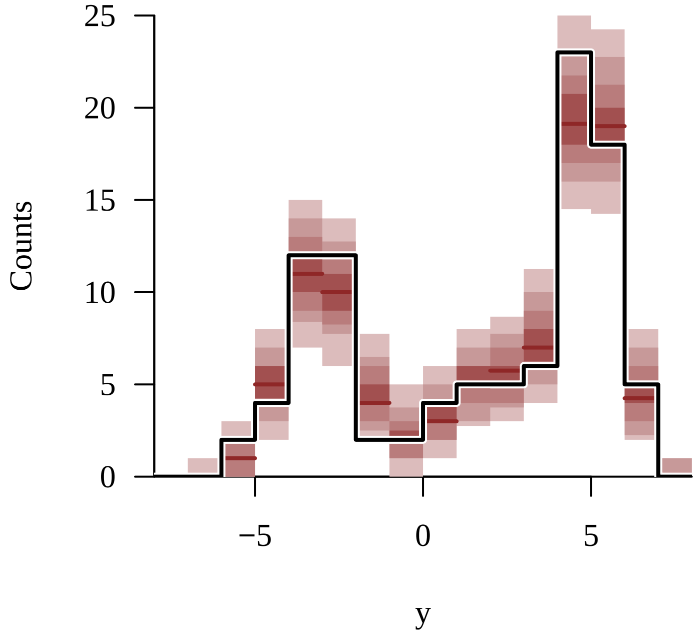
```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

We then can examine the posterior retrodictive and posterior inferential behaviors as before. As expected everything appears to be consistent with the previous model implementations.

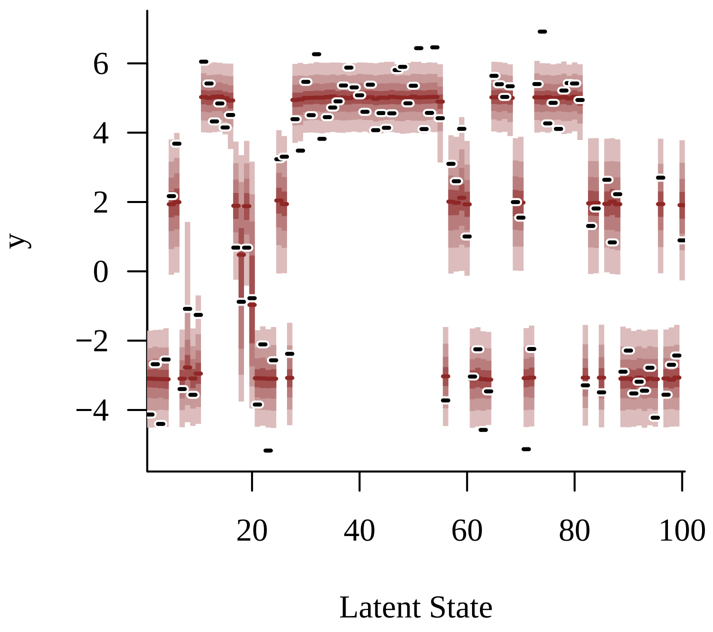
```
par(mfrow=c(1, 1))

util$plot_hist_quantiles(samples3, 'y_pred', -8, 8, 1,
                         baseline_values=data$y, xlab='y')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 11 predictive values (0.0%) fell below the binning.
```

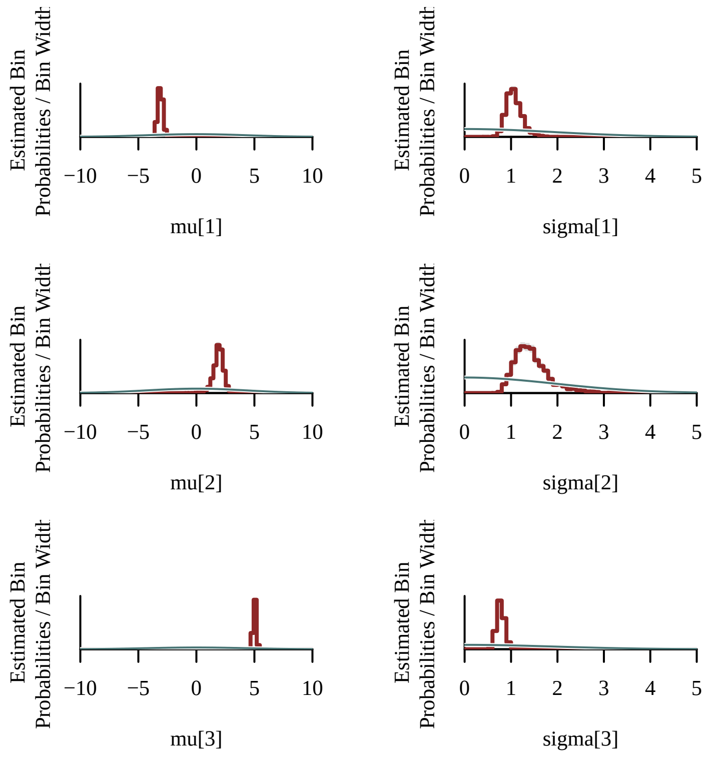
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 77 predictive values (0.0%) fell above the binning.
```



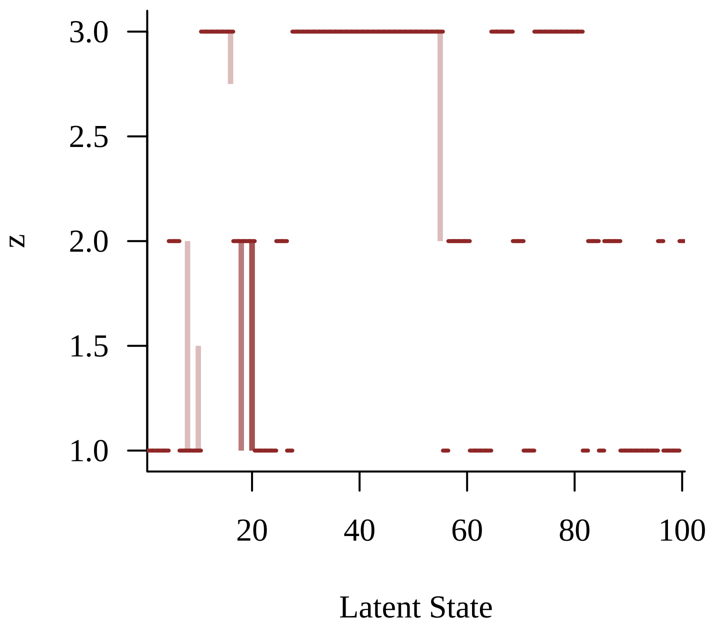
```
par(mfrow=c(1, 1))

names <- sapply(1:data$I, function(i) paste0('y_pred[', i, ']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     baseline_values=data$y,
                                     xlab="Latent State", ylab="y")
```



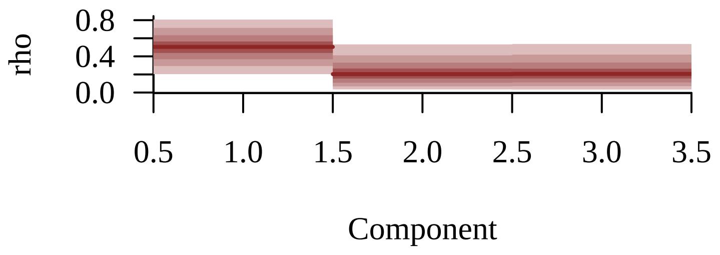
```r
par(mfrow=c(3, 2))

for (k in 1:data$K) {
  name <- paste0('mu[', k, ']')
  util$plot_expectand_pushforward(samples3[[name]], 75,
                                  display_name=name, flim=c(-10, 10))

  xs <- seq(-10, 10, 0.1)
  ys <- dnorm(xs, 0, 10 / 2.32);
  lines(xs, ys, lwd=2, col="white")
  lines(xs, ys, lwd=1, col=util$c_mid_teal)

  name <- paste0('sigma[', k, ']')
  util$plot_expectand_pushforward(samples3[[name]], 50,
                                  display_name=name, flim=c(0, 5))

  xs <- seq(0, 5, 0.1)
  ys <- 2 * dnorm(xs, 0, 5 / 2.57);
  lines(xs, ys, lwd=2, col="white")
  lines(xs, ys, lwd=1, col=util$c_mid_teal)
}
```

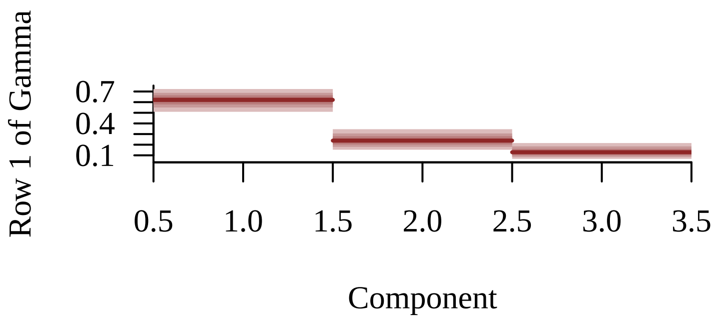


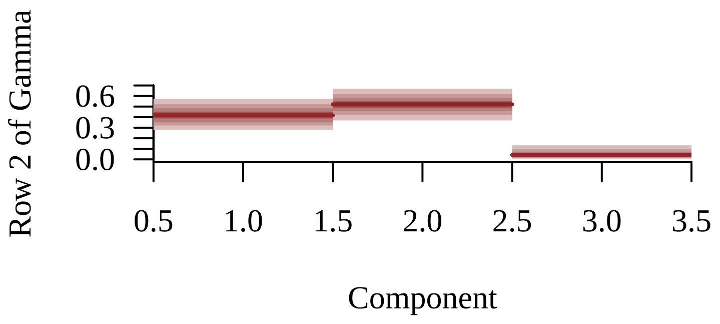
```
par(mfrow=c(1, 1))

names <- sapply(1:data$I, function(i) paste0('z[', i, ']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Latent State", ylab="z")
```

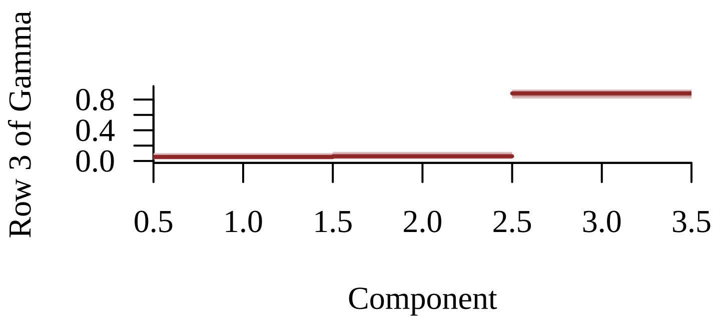


```r
par(mfrow=c(2, 1))

names <- sapply(1:data$K, function(kk) paste0('rho[', kk, ']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Component",
                                     ylab="rho")

for (k in 1:data$K) {
  names <- sapply(1:data$K,
                  function(kk) paste0('gamma[', k, ',', kk, ']'))
  util$plot_disc_pushforward_quantiles(samples3, names,
                                       xlab="Component",
                                       ylab=paste("Row", k, "of Gamma"))
}
```







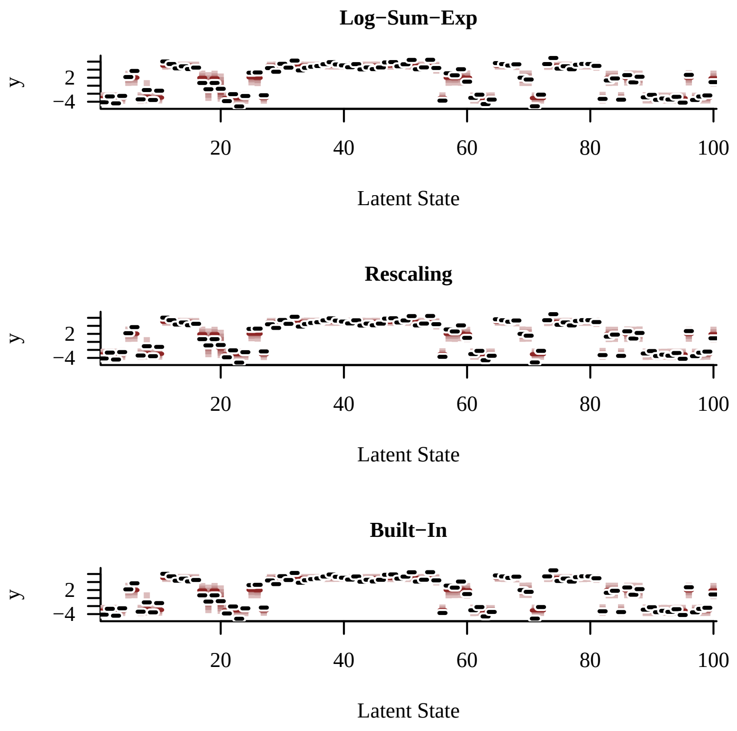


7.2.6 Implementation Consistency Check

To properly compare the outputs of these three model implementations we really need to visualize them next to each other.

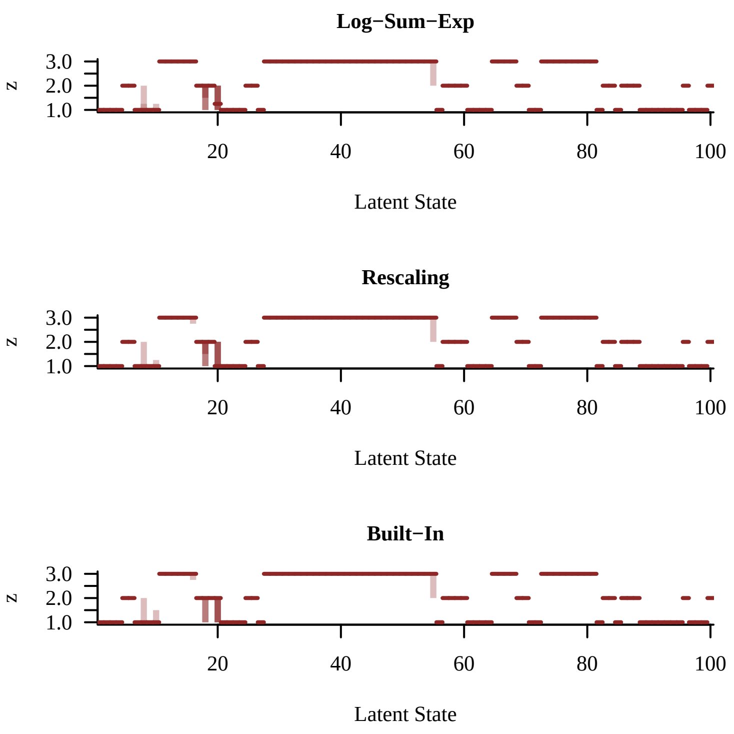
As expected, or at least for what we hoped, the posterior retrodictive performance for the empirical trajectories is consistent across all three model implementations.

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

names <- sapply(1:data$I, function(i) paste0('y_pred[', i, ']'))

util$plot_disc_pushforward_quantiles(samples1, names,
                                     baseline_values=data$y,
                                     xlab="Latent State", ylab="y",
                                     main="Log-Sum-Exp")

util$plot_disc_pushforward_quantiles(samples2, names,
                                     baseline_values=data$y,
                                     xlab="Latent State", ylab="y",
                                     main="Rescaling")

util$plot_disc_pushforward_quantiles(samples3, names,
                                     baseline_values=data$y,
                                     xlab="Latent State", ylab="y",
                                     main="Built-In")
```



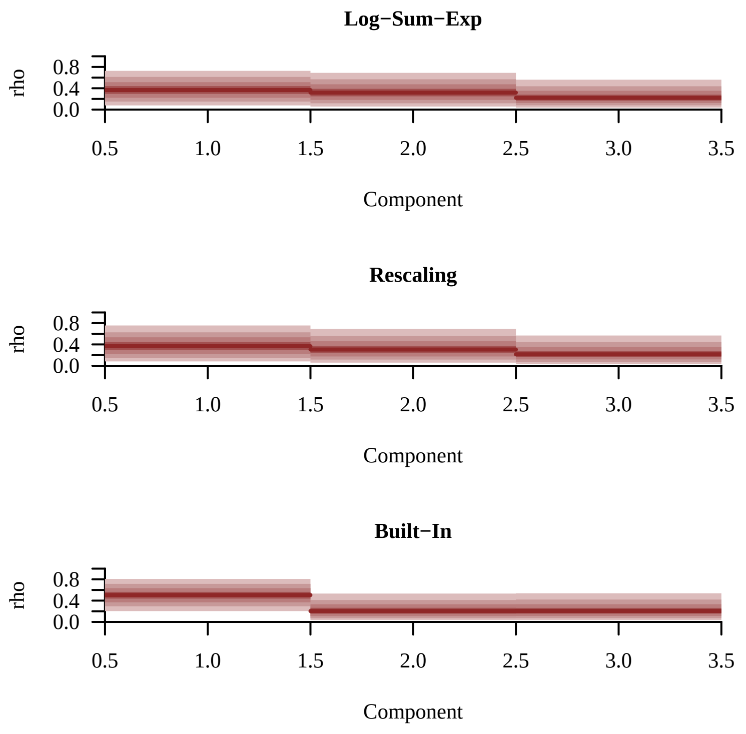
So too is the inferred behavior of the latent states.

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

names <- sapply(1:data$I, function(i) paste0('z[', i, ']'))

util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Latent State", ylab="z",
                                     main="Log-Sum-Exp")

util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Latent State", ylab="z",
                                     main="Rescaling")

util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Latent State", ylab="z",
                                     main="Built-In")
```

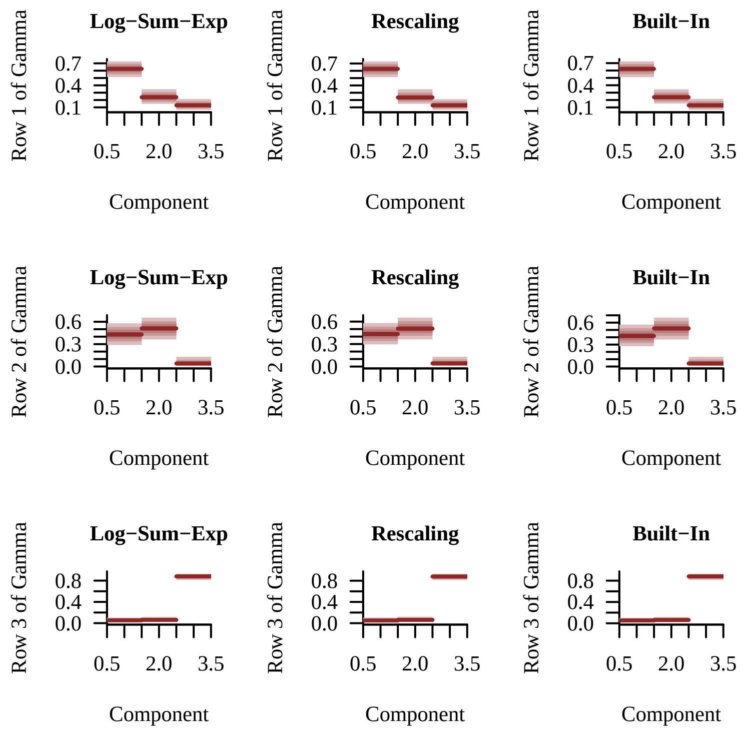


The consistency continues into the inferences for the dynamical parameters.

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

names <- sapply(1:data$K, function(kk) paste0('rho[', kk, ']'))

util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Component",
                                     ylab="rho",
                                     display_ylim=c(0, 1),
                                     main="Log-Sum-Exp")

util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Component",
                                     ylab="rho",
                                     display_ylim=c(0, 1),
                                     main="Rescaling")

util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Component",
                                     ylab="rho",
                                     display_ylim=c(0, 1),
                                     main="Built-In")
```

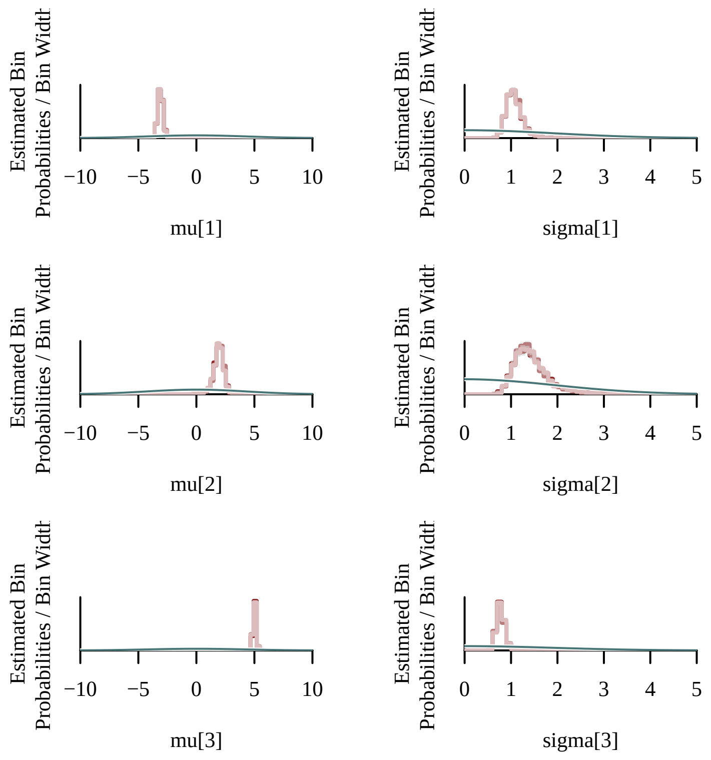


```
par(mfrow=c(3, 3))

for (k in 1:data$K) {
  names <- sapply(1:data$K,
                  function(kk) paste0('gamma[', k, ',', kk, ']'))

  util$plot_disc_pushforward_quantiles(samples1, names,
                                       xlab="Component",
                                       ylab=paste("Row", k, "of Gamma"),
                                       main="Log-Sum-Exp")

  util$plot_disc_pushforward_quantiles(samples2, names,
                                       xlab="Component",
                                       ylab=paste("Row", k, "of Gamma"),
                                       main="Rescaling")

  util$plot_disc_pushforward_quantiles(samples3, names,
                                       xlab="Component",
                                       ylab=paste("Row", k, "of Gamma"),
                                       main="Built-In")
}
```



Finally the inferred behavior of the component observational models is the same across all three model implementations.

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

for (k in 1:data$K) {
  name <- paste0('mu[', k, ']')
  util$plot_expectand_pushforward(samples1[[name]],
                                  75, flim=c(-10, 10),
                                  display_name=name)
  util$plot_expectand_pushforward(samples2[[name]],
                                  75, flim=c(-10, 10),
                                  col=util$c_mid,
                                  border="#DDDDDD88",
                                  add=TRUE)
  util$plot_expectand_pushforward(samples3[[name]],
                                  75, flim=c(-10, 10),
                                  col=util$c_light,
                                  border="#DDDDDD88",
                                  add=TRUE)

  xs <- seq(-10, 10, 0.1)
  ys <- dnorm(xs, 0, 10 / 2.32);
  lines(xs, ys, lwd=2, col="white")
  lines(xs, ys, lwd=1, col=util$c_mid_teal)

  name <- paste0('sigma[', k, ']')
  util$plot_expectand_pushforward(samples1[[name]],
                                  50, flim=c(0, 5),
                                  display_name=name)
  util$plot_expectand_pushforward(samples2[[name]],
                                  50, flim=c(0, 5),
                                  col=util$c_mid,
                                  border="#DDDDDD88",
                                  add=TRUE)
  util$plot_expectand_pushforward(samples3[[name]],
                                  50, flim=c(0, 5),
                                  col=util$c_light,
                                  border="#DDDDDD88",
                                  add=TRUE)

  xs <- seq(0, 5, 0.1)
  ys <- 2 * dnorm(xs, 0, 5 / 2.57);
  lines(xs, ys, lwd=2, col="white")
  lines(xs, ys, lwd=1, col=util$c_mid_teal)
}
```



7.2.7 Static/Dynamic Mixture Model Comparison

Finally to see if there is any advantage to modeling the dynamics of the latent states let's compare the hidden Markov model inferences to inferences from a static mixture model.

```
fit <- stan(file="stan_programs/static_mixture.stan",
            data=data, seed=4938483,
            warmup=1000, iter=2024, refresh=0)
```

The computational 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.
```

```
samples4 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples4,
                                       c('mu', 'sigma',
                                         'lambda'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

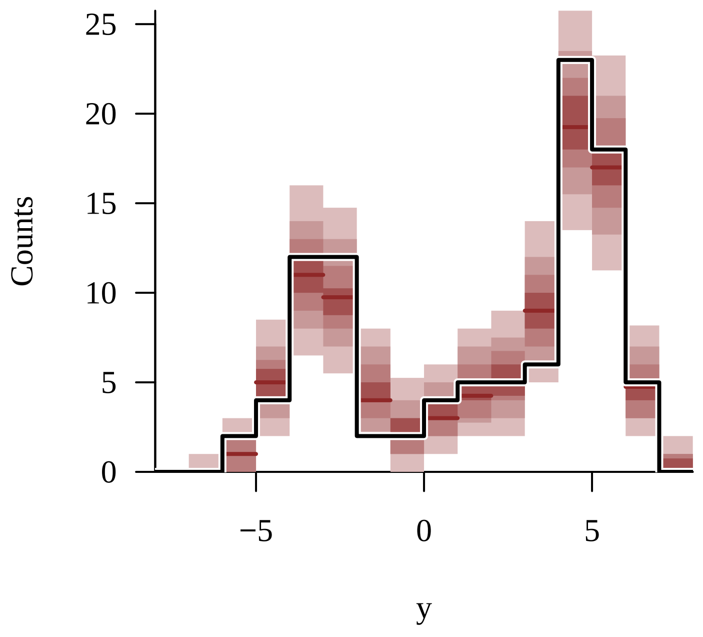
```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

We also see excellent posterior retrodictive performance with respect to the histogram summary statistic. When the component observational models are homogeneous across the latent states static mixture models often perform reasonably well in aggregate even though they ignore the latent dynamics.

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

util$plot_hist_quantiles(samples4, 'y_pred', -8, 8, 1,
                         baseline_values=data$y, xlab='y')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 48 predictive values (0.0%) fell below the binning.
```

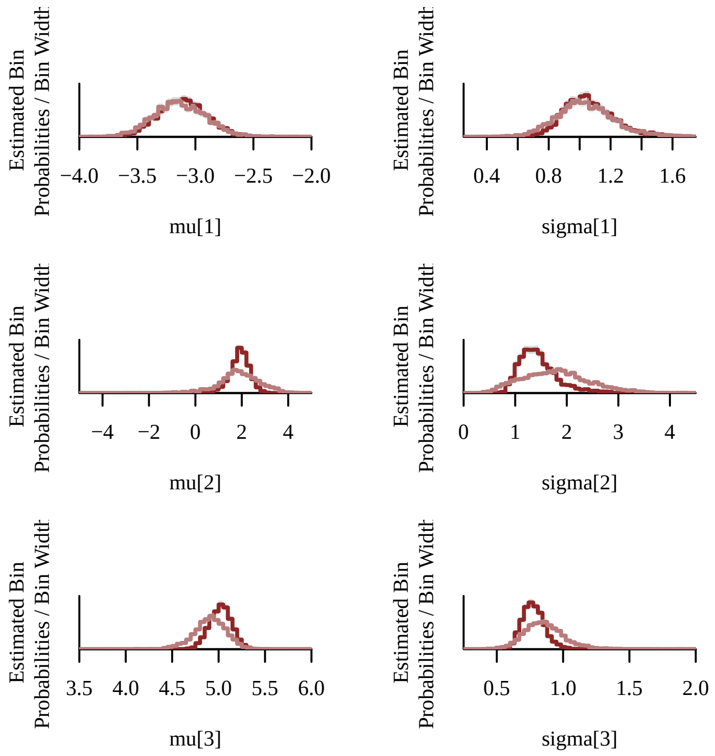
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 647 predictive values (0.2%) fell above the binning.
```



Inferences for the location and scale of the first component observational model are nearly identical between the dynamic and static models, but the inferences for the second and third component observational model configurations are more strongly informed with the hidden Markov model.

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

mu_lims <- list(c(-4, -2), c(-5, 5), c(3.5, 6))
mu_ylims <- list(c(0, 2.75), c(0, 1.25), c(0, 4))

sigma_lims <- list(c(0.25, 1.75), c(0, 4.5), c(0.25, 2))
sigma_ylims <- list(c(0, 3.5), c(0, 1.5), c(0, 5.5))

for (k in 1:data$K) {
  name <- paste0('mu[', k, ']')
  util$plot_expectand_pushforward(samples3[[name]], 50,
                                  display_name=name,
                                  flim=mu_lims[[k]],
                                  ylim=mu_ylims[[k]])
  util$plot_expectand_pushforward(samples4[[name]], 50,
                                  display_name=name,
                                  flim=mu_lims[[k]],
                                  ylim=mu_ylims[[k]],
                                  col=util$c_mid,
                                  border="#DDDDDD88",
                                  add=TRUE)

  name <- paste0('sigma[', k, ']')
  util$plot_expectand_pushforward(samples3[[name]], 50,
                                  display_name=name,
                                  flim=sigma_lims[[k]],
                                  ylim=sigma_ylims[[k]])
  util$plot_expectand_pushforward(samples4[[name]], 50,
                                  display_name=name,
                                  flim=sigma_lims[[k]],
                                  ylim=sigma_ylims[[k]],
                                  col=util$c_mid,
                                  border="#DDDDDD88",
                                  add=TRUE)
}
```

```
Warning in util$plot_expectand_pushforward(samples4[[name]], 50, display_name =
name, : 1 value (0.0%) fell below the histogram binning.

Warning in util$plot_expectand_pushforward(samples4[[name]], 50, display_name =
name, : 1 value (0.0%) fell above the histogram binning.
```



Even when a static mixture model adequately captures the aggregate behavior of the observed data the ignorance of any latent dynamics generally results in worse inferential uncertainties.

7.3 Modeling Unobserved Latent States

Now let's explore some of the ways that we can accommodate latent states that have been only partially observed.

For this exercise we'll use a data set where there are fewer observations than latent states. Consequently at least some of the latent states have to be missing complementary data.

```
data <- read_rdump("data/miss.data.R")

cat(sprintf('%i latent states', data$I))
```

```
100 latent states
```

```
cat(sprintf('%i observational model components', data$K))
```

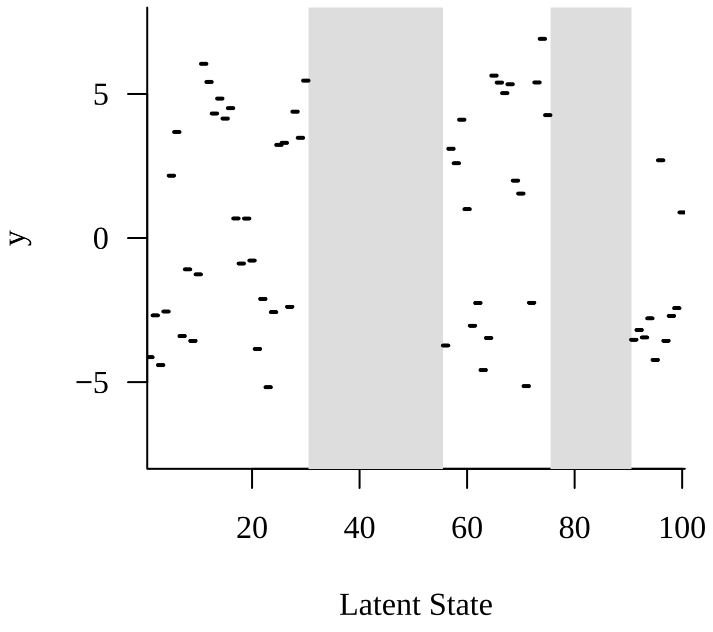
```
3 observational model components
```

```
cat(sprintf('%i observations', data$N))
```

```
60 observations
```

If we look at the empirical trajectory we can see two prominent gaps in the observations.

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

plot(1, type="n",
     xlab="Latent State", xlim=c(0.5, data$I + 0.5),
     ylab="y", ylim=c(-8, 8))

for (miss_win in list(c(31, 55), c(76, 90))) {
  polygon(c(miss_win[1] - 0.5, miss_win[2] + 0.5,
            miss_win[2] + 0.5, miss_win[1] - 0.5),
          c(-8, -8, 8, 8), col="#DDDDDD", border=NA)
}

for (n in 1:data$N) {
  lines(c(data$state[n] - 0.5, data$state[n] + 0.5),
        rep(data$y[n], 2),
        col="black", lwd=2)
}
```



The most straightforward way to unobserved latent states is to just replace the log observational model outputs with zeroes whenever a latent state is unobserved. In particular this allows us to use hidden Markov model functions in `Stan`.

For convenience this Stan program fills in the unobserved latent states with proxy observations and then constructs a binary variable indicating whether or not each latent state is observed. This just makes it easier to work sequentially through the latent states.

```
fit <- stan(file="stan_programs/hmm_builtin_partial.stan",
            data=data, seed=4938483,
            warmup=1000, iter=2024, refresh=0)
```

There are some indications of mild autocorrelations but no indications of inaccurate computation.

```
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.
```

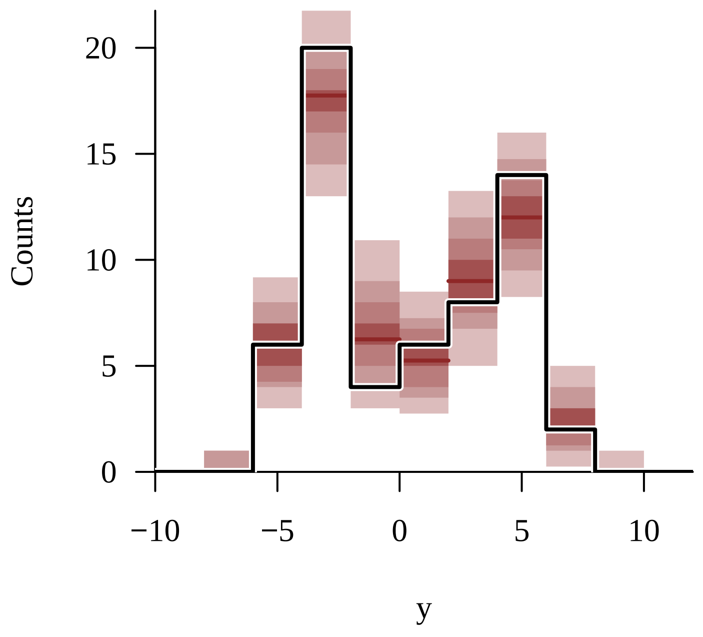
```
samples1 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples1,
                                       c('mu', 'sigma',
                                         'rho', 'gamma'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
mu[2]:
  Chain 1: hat{ESS} (73.717) is smaller than desired (100).
  Chain 2: hat{ESS} (74.010) is smaller than desired (100).
  Chain 3: hat{ESS} (52.720) is smaller than desired (100).

mu[3]:
  Chain 1: hat{ESS} (94.008) is smaller than desired (100).

gamma[1,1]:
  Chain 1: hat{ESS} (82.813) is smaller than desired (100).
  Chain 3: hat{ESS} (52.942) is smaller than desired (100).

gamma[1,2]:
  Chain 1: Right tail hat{xi} (0.280) exceeds 0.25.
  Chain 1: hat{ESS} (84.315) is smaller than desired (100).
  Chain 3: hat{ESS} (52.267) is smaller than desired (100).

gamma[2,3]:
  Chain 3: hat{ESS} (97.845) is smaller than desired (100).


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

Small empirical effective sample sizes result in imprecise Markov chain
Monte Carlo estimators.
```

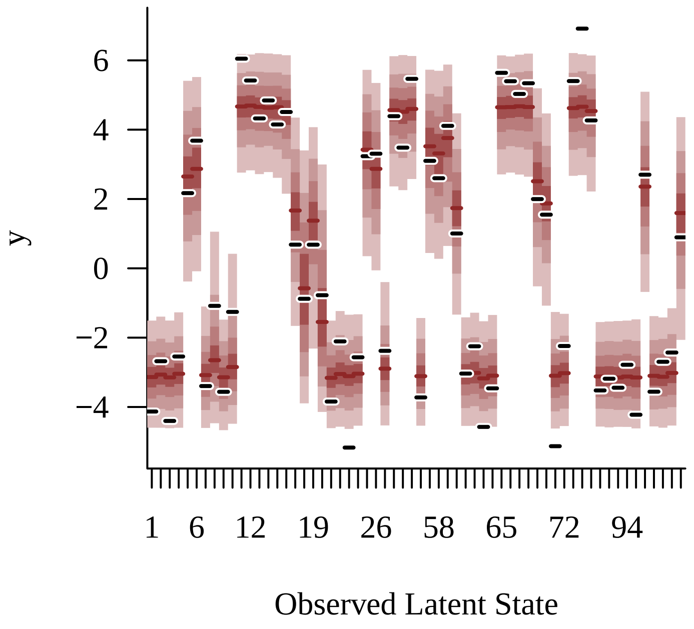
The posterior retrodictive performance is solid.

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

util$plot_hist_quantiles(samples1, 'y_pred', -10, 12, 2,
                         baseline_values=data$y, xlab='y')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 8 predictive values (0.0%) fell below the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 4 predictive values (0.0%) fell above the binning.
```

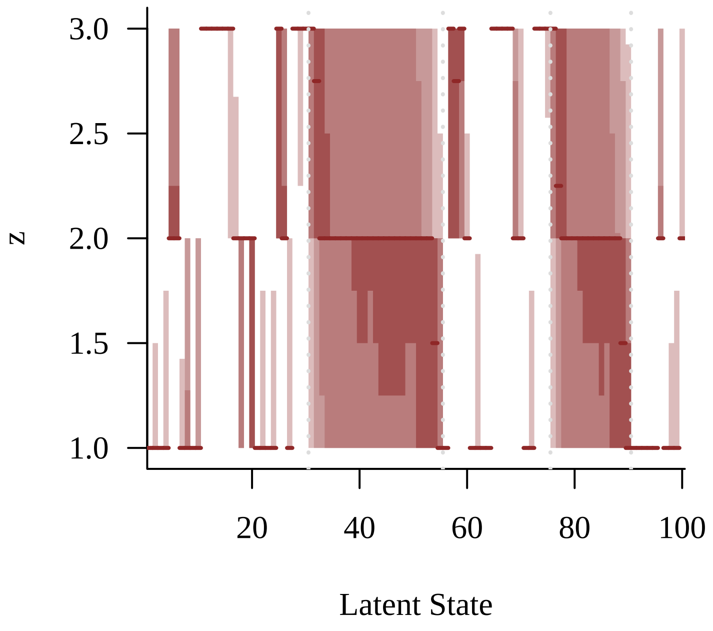


```
par(mfrow=c(1, 1))

names <- sapply(1:data$N, function(n) paste0('y_pred[', n, ']'))
util$plot_disc_pushforward_quantiles(samples1, names,
                                     baseline_values=data$y,
                                     xlab="Observed Latent State",
                                     ylab="y",
                                     xticklabs=data$state)
```



Unsurprisingly inferences for the latent states become increasingly uncertain within the unobserved gaps.

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

names <- sapply(1:data$I, function(i) paste0('z[', i, ']'))
util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Latent State", ylab="z")
abline(v=30.5, lwd=2, lty=3, col="#DDDDDD")
abline(v=55.5, lwd=2, lty=3, col="#DDDDDD")
abline(v=75.5, lwd=2, lty=3, col="#DDDDDD")
abline(v=90.5, lwd=2, lty=3, col="#DDDDDD")
```



The other way that we can take unobserved latent states into account is to remove them and update the transition matrices between the observed latent states. Because this results in a non-homogeneous hidden Markov model we can no longer use the `Stan` hidden Markov model functions. Fortunately this non-homogeneous model is not too difficult to implement by hand.

There are a few ways that we could implement the heterogeneous transition matrices. For example we could pre-compute and then store each transition matrix, but this would introduce a pretty large memory burden. A more effective approach is to compute each transition matrix only when they are used. That said the matrix-matrix products needed for this can be pretty expensive, especially across long gaps of unobserved latent states.

In this Stan program I don't compute the collapsed transition matrices directly but rather iteratively compute their action on the α_i and β_i in the forward and backward algorithms. This requires only matrix-vector products which are less expensive than matrix-matrix products, especially if K is large.

```
fit <- stan(file="stan_programs/hmm_rescaled_partial.stan",
            data=data, seed=4938483,
            warmup=1000, iter=2024, refresh=0)
```

We see the same hints are moderate autocorrelations in the computational diagnostics but nothing that suggests we shouldn't trust our posterior computation.

```r
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.
```

```r
samples2 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples2,
                                       c('mu', 'sigma',
                                         'rho', 'gamma'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
mu[1]:
  Chain 1: hat{ESS} (90.239) is smaller than desired (100).
  Chain 4: hat{ESS} (76.016) is smaller than desired (100).

mu[2]:
  Chain 1: hat{ESS} (50.050) is smaller than desired (100).
  Chain 3: hat{ESS} (95.651) is smaller than desired (100).
  Chain 4: hat{ESS} (52.991) is smaller than desired (100).

mu[3]:
  Chain 4: hat{ESS} (80.004) is smaller than desired (100).

sigma[3]:
  Chain 4: hat{ESS} (85.254) is smaller than desired (100).

gamma[1,1]:
  Chain 1: hat{ESS} (41.724) is smaller than desired (100).
  Chain 4: hat{ESS} (38.617) is smaller than desired (100).

gamma[1,2]:
  Chain 4: Right tail hat{xi} (0.322) exceeds 0.25.
  Chain 1: hat{ESS} (37.214) is smaller than desired (100).
  Chain 4: hat{ESS} (36.432) is smaller than desired (100).

gamma[2,3]:
  Chain 1: hat{ESS} (87.830) is smaller than desired (100).
  Chain 4: hat{ESS} (67.000) is smaller than desired (100).
```

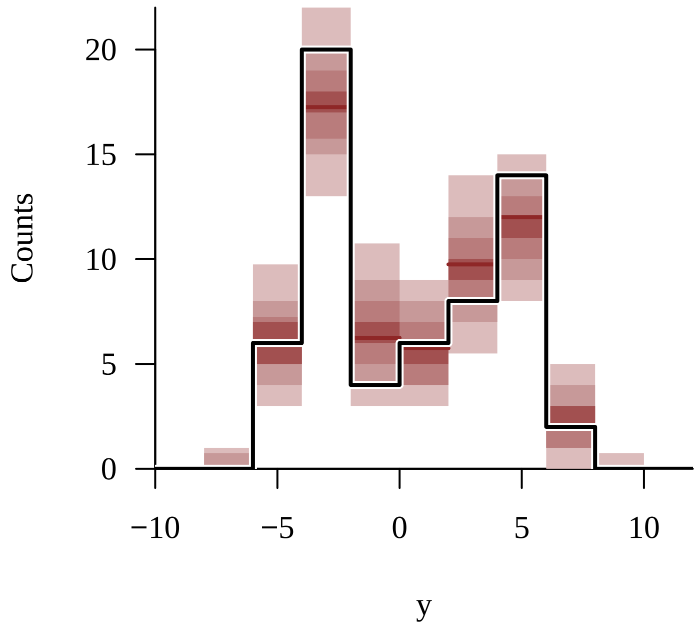
```
Large tail hat{xi}s suggest that the expectand might not be
sufficiently integrable.

Small empirical effective sample sizes result in imprecise Markov chain
Monte Carlo estimators.
```

We don't see any indications of poor retrodictive performance.

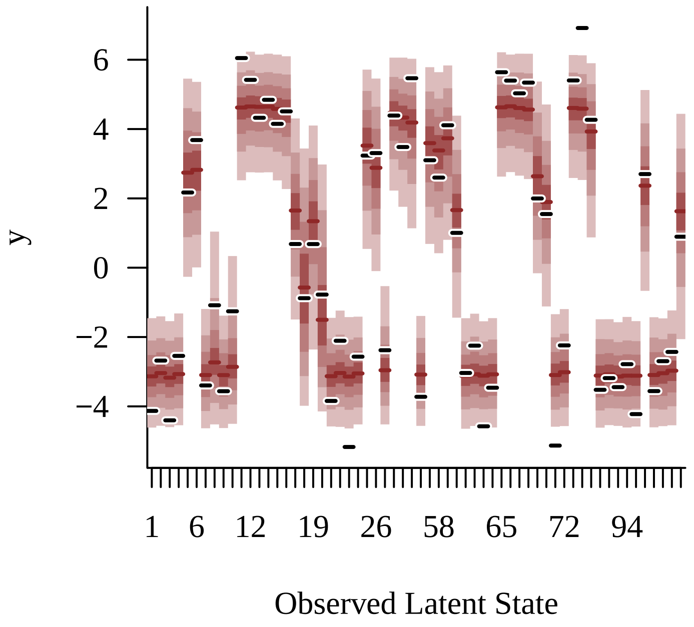
```
par(mfrow=c(1, 1))

util$plot_hist_quantiles(samples2, 'y_pred', -10, 12, 2,
                         baseline_values=data$y, xlab='y')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 9 predictive values (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 10 predictive values (0.0%) fell above the binning.
```



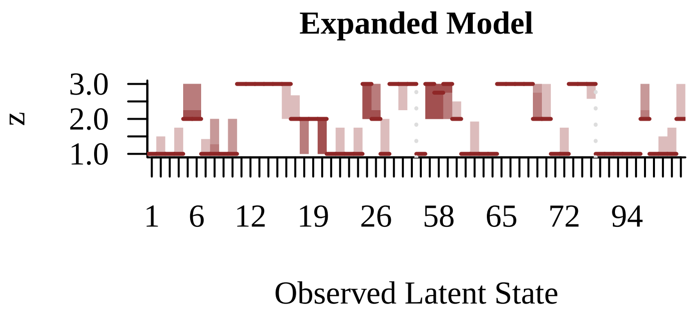
```r
par(mfrow=c(1, 1))

names <- sapply(1:data$N, function(n) paste0('y_pred[', n, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     baseline_values=data$y,
                                     xlab="Observed Latent State",
                                     ylab="y",
                                     xticklabs=data$state)
```

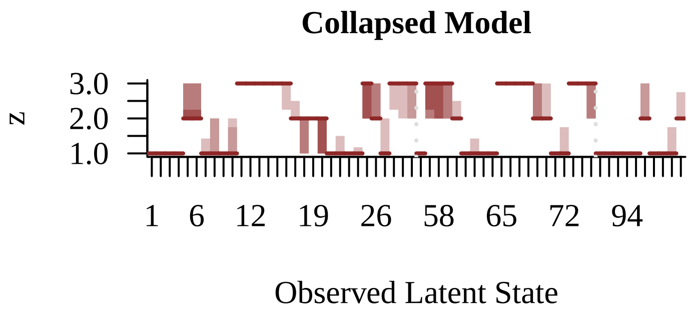


Either way we approach this problem our posterior inferences are consistent with each other. For example inferred behavior for the the observed latent states is equivalent.

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

names <- sapply(data$state, function(i) paste0('z[', i, ']'))
util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Observed Latent State",
                                     ylab="z",
                                     xticklabs=data$state,
                                     main="Expanded Model")
abline(v=30.5, lwd=2, lty=3, col="#DDDDDD")
abline(v=50.5, lwd=2, lty=3, col="#DDDDDD")

names <- sapply(1:data$N, function(n) paste0('z[', n, ']'))
```

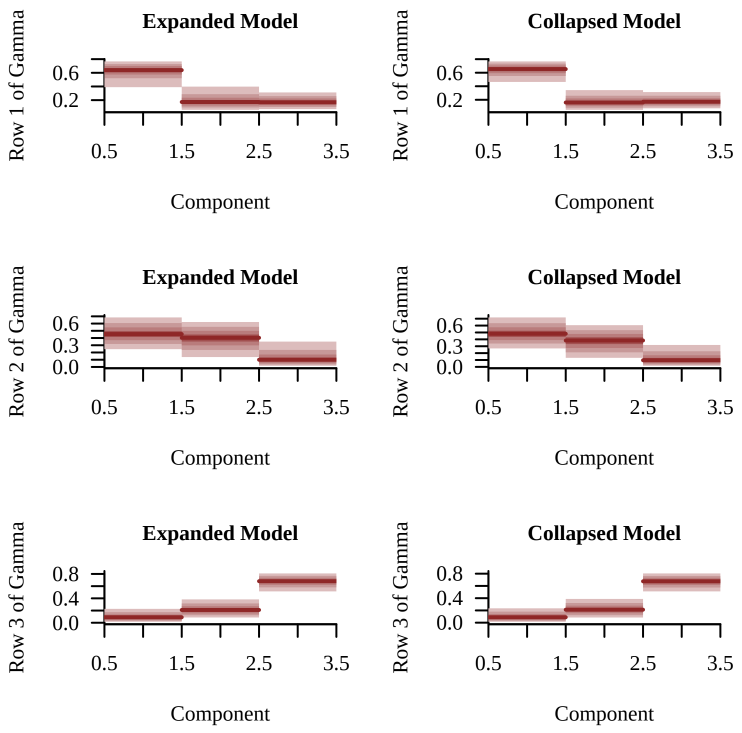
```
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Observed Latent State",
                                     ylab="z",
                                     xticklabs=data$state,
                                     main="Collapsed Model")
abline(v=30.5, lwd=2, lty=3, col="#DDDDDD")
abline(v=50.5, lwd=2, lty=3, col="#DDDDDD")
```





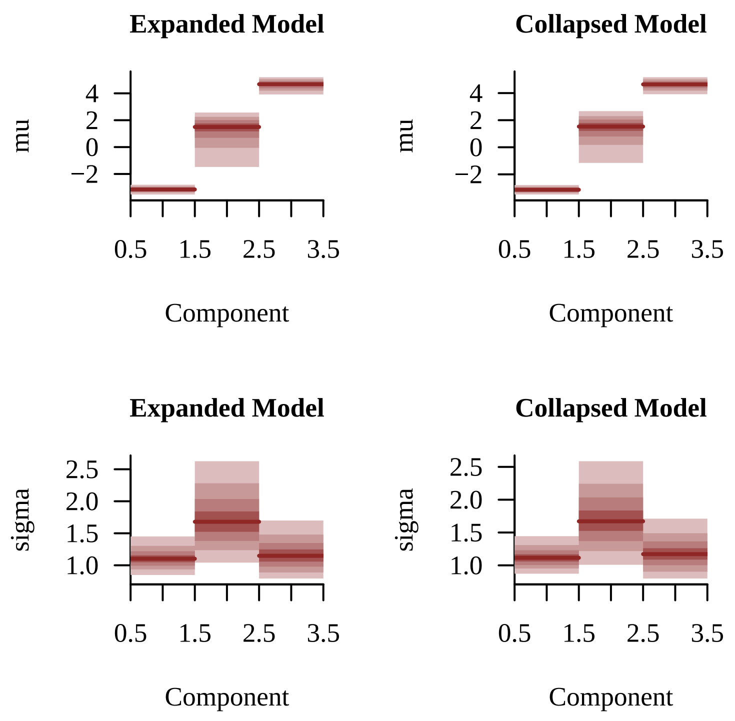
So too is the inferred behavior for the base transition matrix.

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

for (k in 1:K) {
  names <- sapply(1:data$K,
                  function(kk) paste0('gamma[', k, ',', kk, ']'))
  ylab <- paste("Row", k, "of Gamma")

  util$plot_disc_pushforward_quantiles(samples1, names,
                                       xlab="Component", ylab=ylab,
                                       main="Expanded Model")

  util$plot_disc_pushforward_quantiles(samples2, names,
                                       xlab="Component", ylab=ylab,
                                       main="Collapsed Model")
}
```



As well as the component observational model configurations.

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

names <- sapply(1:data$K, function(k) paste0('mu[', k, ']'))

util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Component", ylab="mu",
                                     main="Expanded Model")

util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Component", ylab="mu",
                                     main="Collapsed Model")

names <- sapply(1:data$K, function(k) paste0('sigma[', k, ']'))

util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Component", ylab="sigma",
                                     main="Expanded Model")

util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Component", ylab="sigma",
                                     main="Collapsed Model")
```



7.4 Hidden Markov Models In Practice

Let's conclude with an analysis that at least approximates some of the messiness that we might encounter when using hidden Markov models in practice.

The data for this analysis were collected every month from detectors that record how often a certain phenomena appeared locally in a particular environment. For example these counts could be the number of appearances of a certain animal species in field cameras or the number of insects caught in traps.

7.4.1 Exploratory Data Analysis

For this new data set we have more observations than months.

```
data <- read_rdump("data/analysis.data.R")

cat(sprintf('%i months', data$I))
```

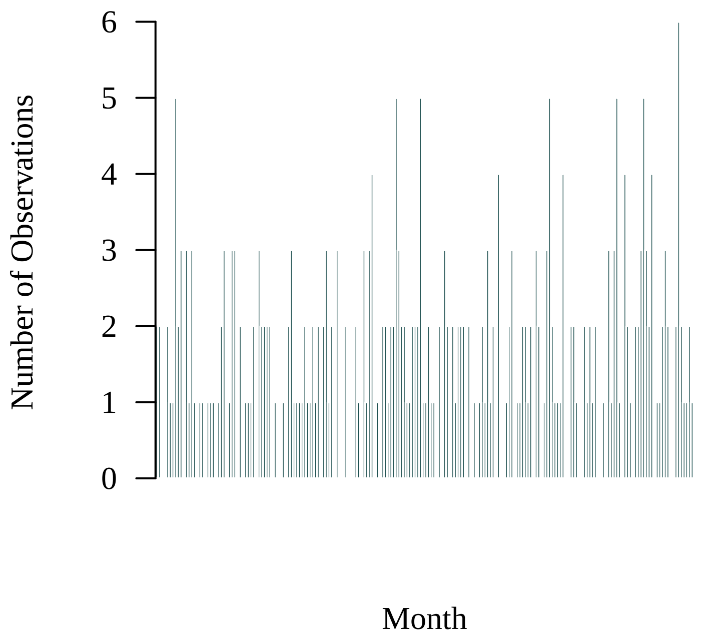
```
200 months
```

```
cat(sprintf('%i observations', data$N))
```

```
300 observations
```

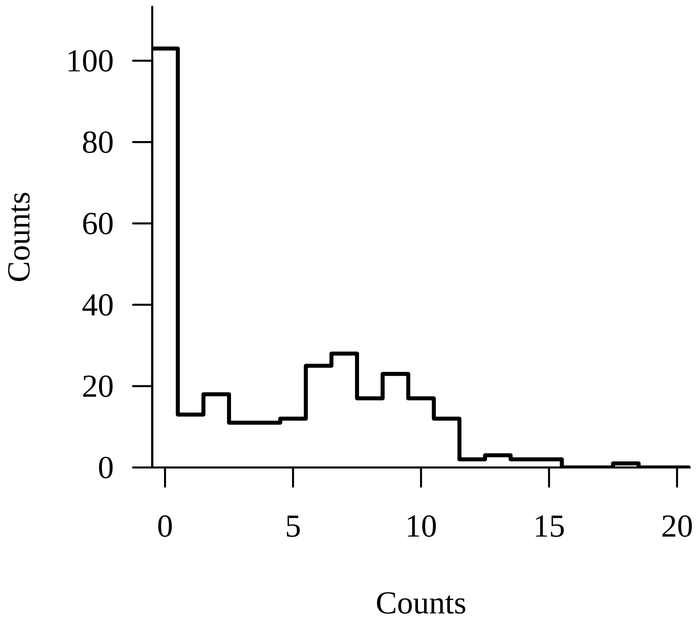
In fact some months are accompanied by observations from multiple detectors and some are completely unobserved. This heterogeneity in the data collection is not at all uncommon in practice.

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

barplot(data$state_N,
        space=0, col=util$c_dark_teal, border="white",
        xlab="Month", ylab="Number of Observations")
```



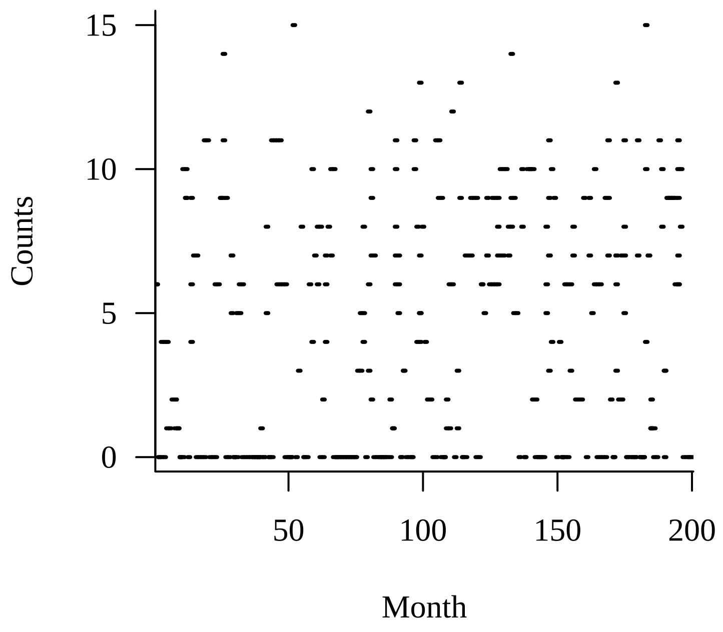
There definitely seem to be multiple, distinct features when we aggregate the data together.

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

util$plot_line_hist(data$y, -0.5, 20.5, 1, xlab="Counts")
```



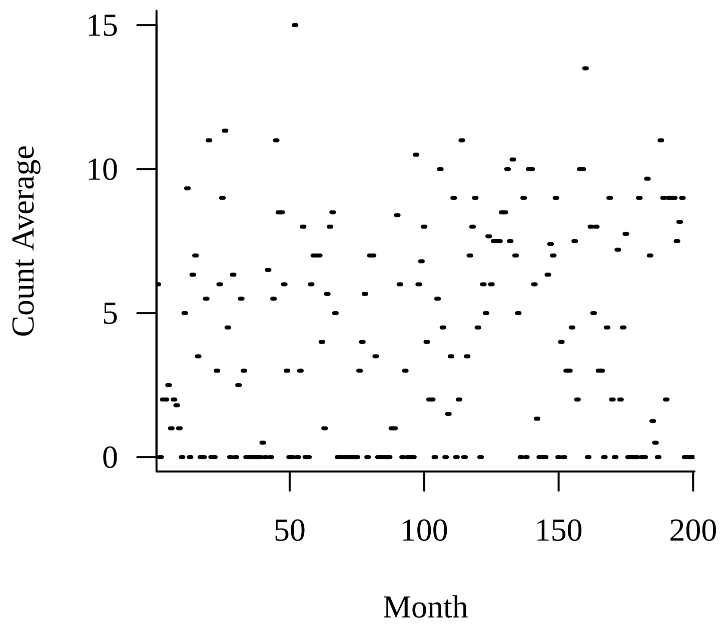
We can still plot the observed data against their corresponding latent states, but because there are multiple observations for some latent states we can no longer interpret this as a single empirical trajectory.

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

plot(1, type="n",
     xlab="Month", xlim=c(0.5, data$I + 0.5),
     ylab="Counts", ylim=c(-0.5, 15.5))

for (i in 1:data$I) {
  for (n in data$state_start_idx[i]:data$state_end_idx[i])
    lines(c(i - 0.5, i + 0.5), rep(data$y[n], 2),
          col="black", lwd=2)
}
```



In order to construct a single trajectory we need to reduce the observations at each latent state to a single value. For example we might look at the evolution of the empirical average along the latent states.

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

plot(1, type="n",
     xlab="Month", xlim=c(0.5, data$I + 0.5),
     ylab="Count Average", ylim=c(-0.5, 15.5))

for (i in 1:data$I) {
  ave_y <- mean(data$y[data$state_start_idx[i]:data$state_end_idx[i]])
  lines(c(i - 0.5, i + 0.5), rep(ave_y, 2),
        col="black", lwd=2)
}
```



7.4.2 Model 1

For this analysis let's say that our available domain expertise suggests that the environmental activity cycles between low and high periods, and we are interested in learning about rates of transition between these two behaviors. This is an excellent setting for a hidden Markov model, using two Poisson observational models to capture the environmental behaviors.

In other to accommodate the heterogeneous number of observations we just have to be careful in how we evaluate the component observational model outputs at each iteration.

```
fit <- stan(file="stan_programs/hmm_analysis1.stan",
            data=data, seed=4938483,
            warmup=1000, iter=2024, refresh=0)
```

Nothing of concern in the computational diagnostics.

```
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.
```

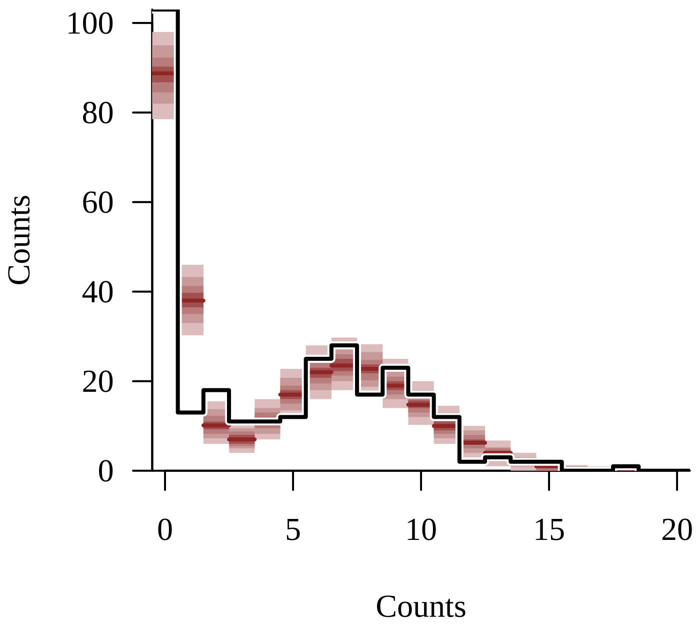
```
samples1 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples1,
                                       c('lambda1', 'lambda2',
                                         'rho', 'tau11', 'tau22'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

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

Unfortunately the posterior retrodictive performance leaves much to be desired. In order to capture the observed peak at zero counts as best as possible the posterior predictive distribution introduces an excess of probability at small but non-zero counts. Even with that contortion the model still struggles to produce as many zero counts.

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

util$plot_hist_quantiles(samples1, 'y_pred', -0.5, 20.5, 1,
                         baseline_values=data$y, xlab='Counts')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 34 predictive values (0.0%) fell above the binning.
```

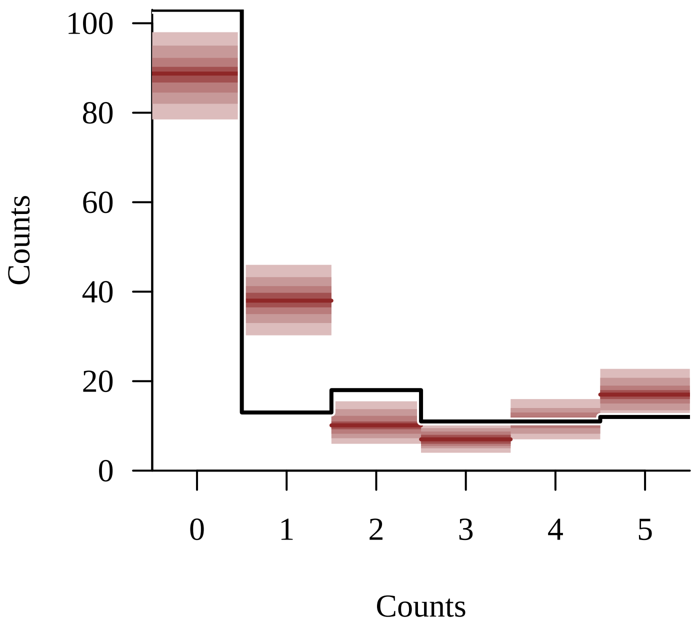


This tension is more evident if we zoom in.

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

util$plot_hist_quantiles(samples1, 'y_pred', -0.5, 5.5, 1,
                         baseline_values=data$y, xlab='Counts')
```

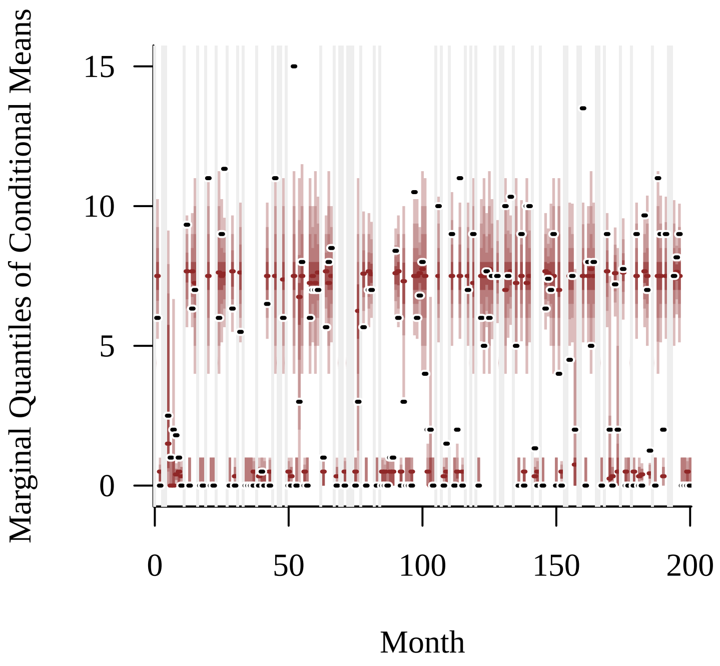
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 521482 predictive values (42.4%) fell above the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, baseline_values, "observed
value"): 132 observed values (44.0%) fell above the binning.
```



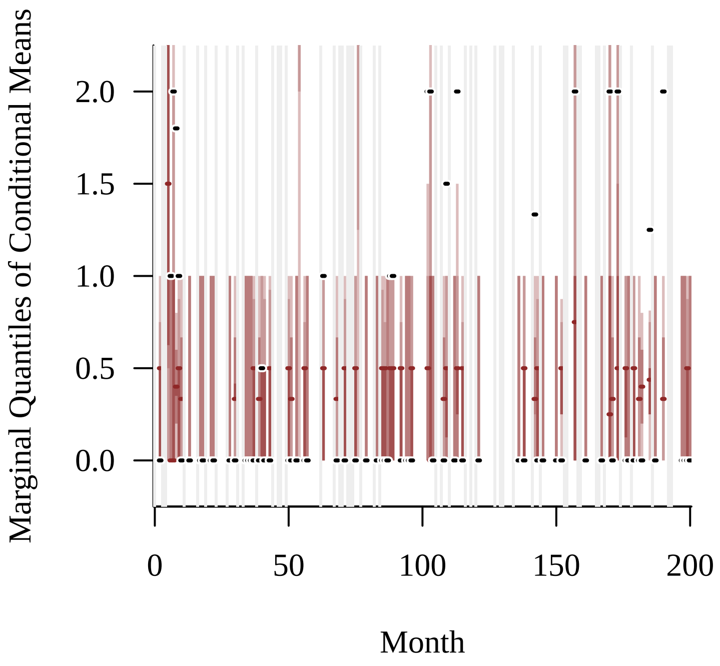
Comparing conditional averages we can see that the observed data concentrates much more strongly at zero than the posterior predictive distribution.

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

names <- sapply(1:data$N, function(n) paste0('y_pred[', n, ']'))
util$plot_conditional_mean_quantiles(samples1, names, data$state,
                                     -0.5, data$I + 0.5, 1, data$y,
                                     xlab="Month")
```



Again it can help to zoom in to emphasize the clashing behavior.

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

names <- sapply(1:data$N, function(n) paste0('y_pred[', n, ']'))
util$plot_conditional_mean_quantiles(samples1, names, data$state,
                                     -0.5, data$I + 0.5, 1, data$y,
                                     xlab="Month",
                                     display_ylim=c(-0.25, 2.25))
```



7.4.3 Model 2

In the process of complaining about our frustrations with this poor retrodictive performance one of our colleagues asks us if we took the operation of the detectors into account. Seeing the confused look on our face they explain that sometimes the detectors at a site all break and record zero counts regardless of how the local environment behaved that month. The only problem is that the status of the detectors were not themselves deemed important enough to record.

With the benefit of hindsight the poor predictive performance of our initial model now makes a lot more sense. In order more adequately model these observations we need to incorporate a third component observational model into our hidden Markov model to accommodate broken detectors.

To do this let's introduce an auxiliary sequence of latent states that indicates whether the detectors were working or not. Assuming that the the probability of the detectors breaking each month is the same regardless of their operation in the previous month the homogeneous transition matrix for this auxiliary latent state becomes

$$\Gamma^1 = \begin{pmatrix} \tau_d & 1 - \tau_d \\ \tau_d & 1 - \tau_d. \end{pmatrix}$$

Combined with the two possible values of the environmental latent state this gives us four elements of each joint latent state,

- `z = 1`: Working detector, low environmental activity
- `z = 2`: Working detector, high environmental activity
- `z = 3`: Broken detector, low environmental activity
- `z = 4`: Broken detector, high environmental activity.

When `z = 1` the local observations are modeled with the low environmental activity Poisson model. For `z = 2` the local observations are modeled the high environmental activity Poisson model. Finally for both `z = 3` and `z = 4` the local observations are modeled with a Dirac model concentrating at zero.

The transition matrix of these joint latent states is then given by the tensor product of the auxiliary transition matrix with our initial environmental transition matrix. By working with multiple latent sequences we can maintain the initial environmental transition probabilities and their interpretations.

```
fit <- stan(file="stan_programs/hmm_analysis2.stan",
            data=data, seed=4938483,
            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.
```

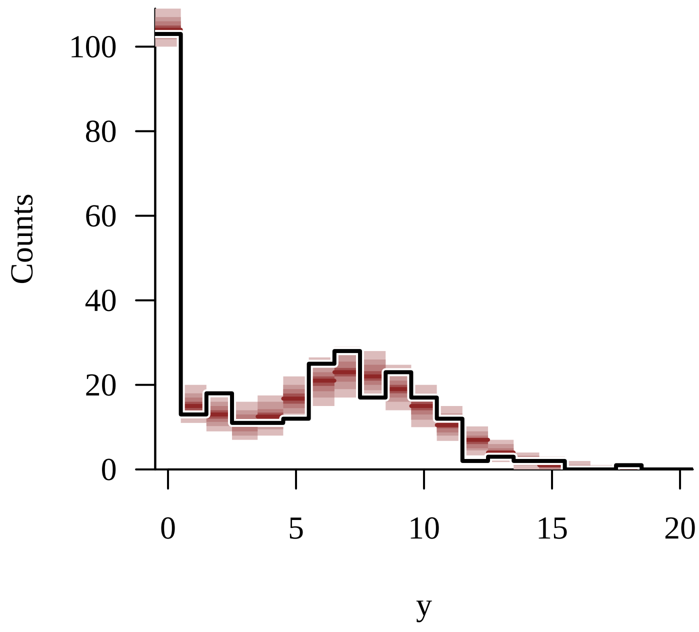
```
samples2 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples2,
                                       c('lambda1', 'lambda2',
                                         'rho', 'tau11', 'tau22',
                                         'taud'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

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

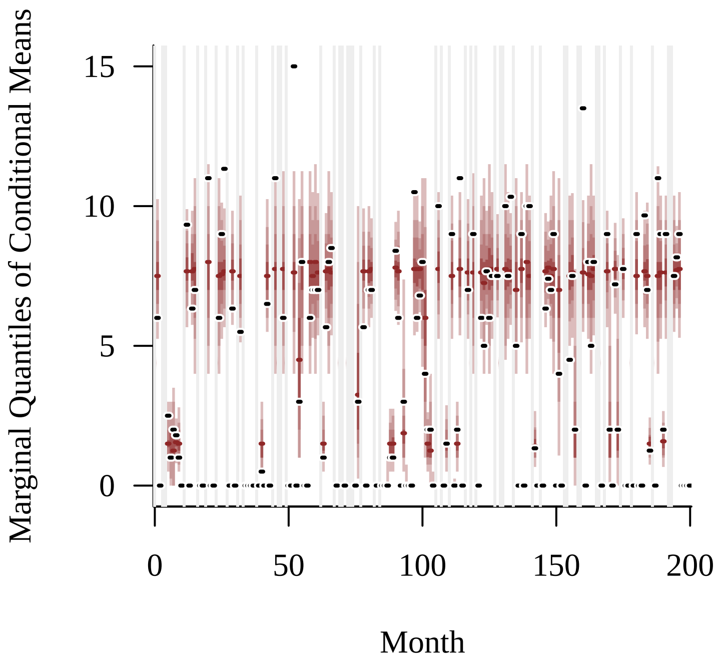
The posterior retrodictive performance has drastically improved.

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

util$plot_hist_quantiles(samples2, 'y_pred', -0.5, 20.5, 1,
                         baseline_values=data$y, xlab='y')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 56 predictive values (0.0%) fell above the binning.
```



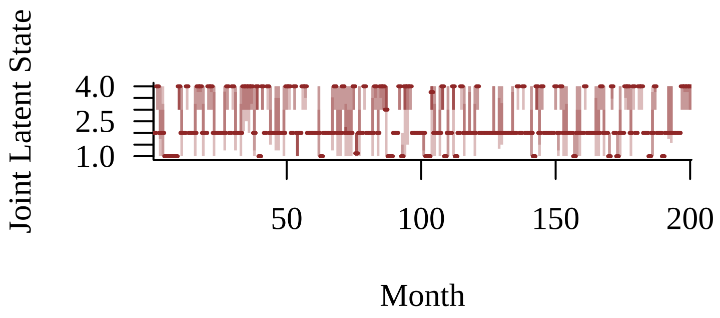
```
par(mfrow=c(1, 1))

state <- c(sapply(1:data$I, function(i) rep(i, data$state_N[i])),
           recursive=TRUE)

names <- sapply(1:data$N, function(n) paste0('y_pred[', n, ']'))
util$plot_conditional_mean_quantiles(samples2, names, data$state,
                                     -0.5, data$I + 0.5, 1, data$y,
                                     xlab="Month")
```

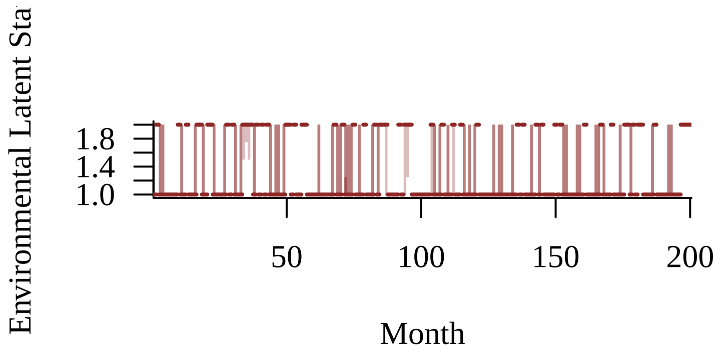


Now we can examine our posterior inferences with confidence.

For example we can investigate either the joint latent state behaviors or just the environmental latent state behaviors.

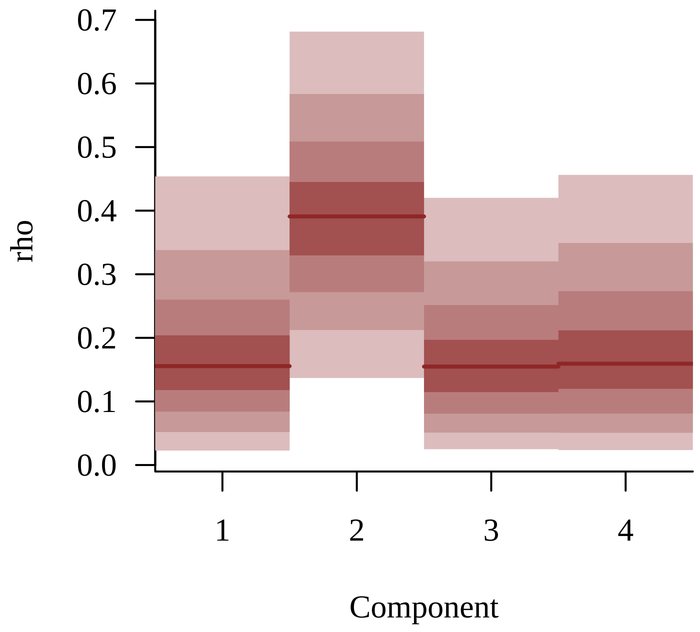
```
par(mfrow=c(2, 1))

names <- sapply(1:data$I, function(i) paste0('z[', i, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Month",
                                     ylab="Joint Latent State")

names <- sapply(1:data$I, function(i) paste0('z_env[', i, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Month",
                                     ylab="Environmental Latent State")
```



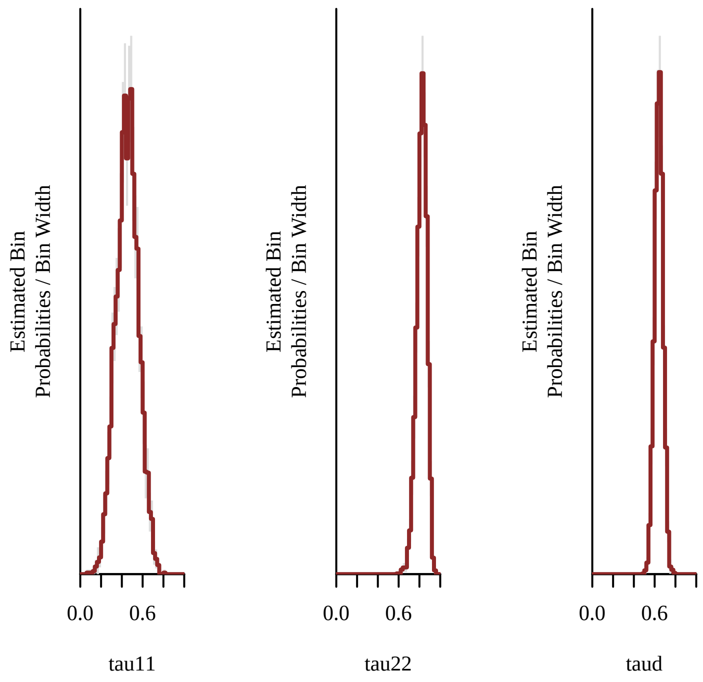


We can examine the dynamics of the latent states. Note that the site detectors seem to work only about two-thirds of the time. Moreover the high activity state appear to be more persistent than the low activity state.

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

names <- sapply(1:4, function(kk) paste0('rho[', kk, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Component",
                                     ylab="rho")
```

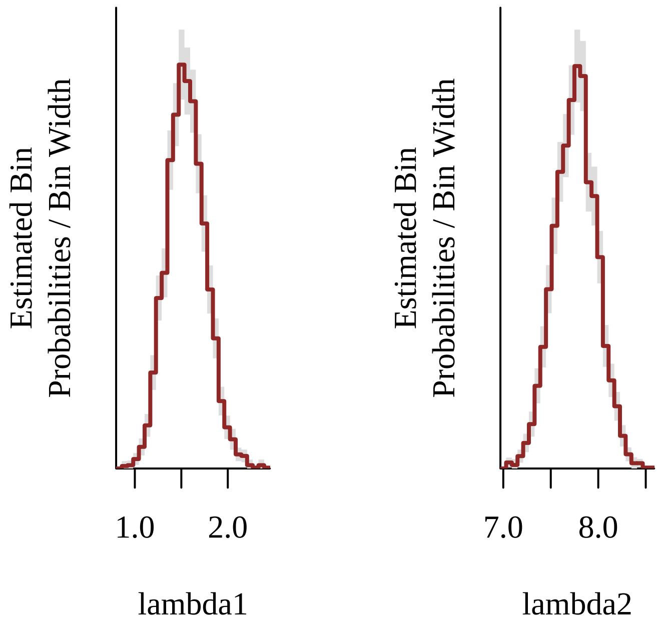


```
par(mfrow=c(1, 3))

for (name in c('tau11', 'tau22', 'taud')) {
  util$plot_expectand_pushforward(samples2[[name]], 50,
                                  display_name=name,
                                  flim=c(0, 1))
}
```



Finally we can consider the main behavior of interest: the two environmental activity intensity parameters.

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

for (name in c('lambda1', 'lambda2')) {
  util$plot_expectand_pushforward(samples2[[name]], 25,
                                  display_name=name)
}
```



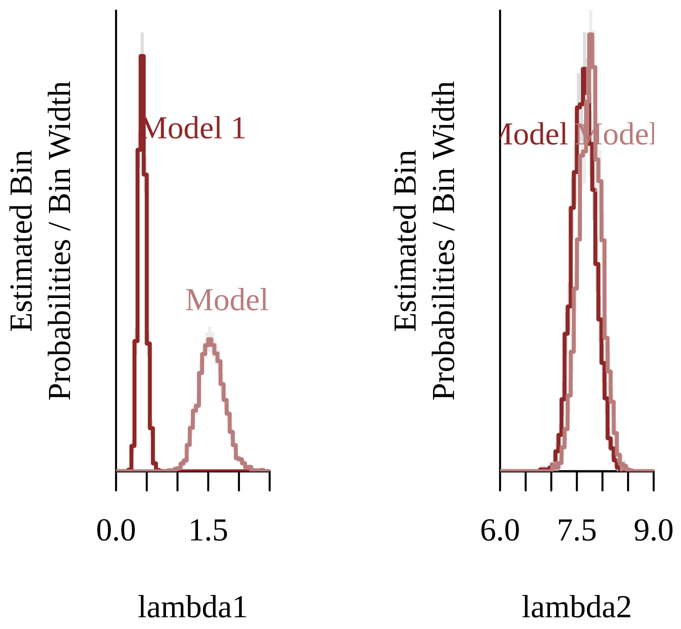
By isolating the environmental behaviors from the detector behaviors we not only advance our immediate scientific goals but also put ourselves in a position to make predictions under various hypotheses for future detector behavior. This in turn helps us motivate improved experimental procedures moving forwards.

7.4.4 Inferential Comparison

What would happen if we had been careless and ignored the poor retrodictive performance exhibited by our initial model?

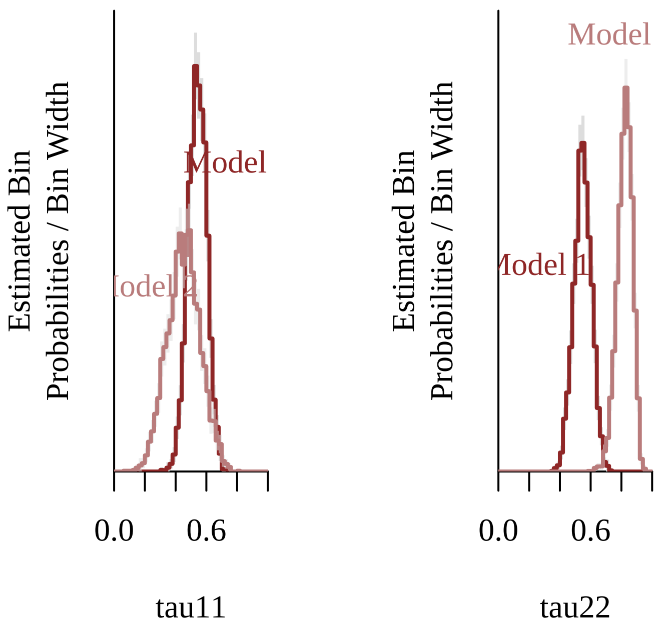
Firstly our inferences for the low activity behavior would substantially underestimate the more realistic behavior. The high activity behavior, which is far less easy to confuse with the behavior of broken detectors, is less compromised.

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

name <- 'lambda1'
util$plot_expectand_pushforward(samples1[[name]], 50,
                                display_name=name,
                                flim=c(0, 2.5))
text(1.25, 5, 'Model 1', col=util$c_dark)

util$plot_expectand_pushforward(samples2[[name]], 50,
                                display_name=name,
                                flim=c(0, 2.5),
                                col=util$c_mid,
                                border="#DDDDDD88",
                                add=TRUE)
text(2.0, 2.5, 'Model 2', col=util$c_mid)


name <- 'lambda2'
util$plot_expectand_pushforward(samples1[[name]], 50,
                                display_name=name,
                                flim=c(6, 9))
text(6.75, 1.5, 'Model 1', col=util$c_dark)

util$plot_expectand_pushforward(samples2[[name]], 50,
                                display_name=name,
                                flim=c(6, 9),
                                col=util$c_mid,
                                border="#DDDDDD88",
                                add=TRUE)
text(8.5, 1.5, 'Model 2', col=util$c_mid)
```



The inferences derived from the first model are also inaccurate for both environmental transition probabilities, especially for the high activity persistence probability τ_{22}.

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

name <- 'tau11'
util$plot_expectand_pushforward(samples1[[name]], 50,
                                display_name=name,
                                flim=c(0, 1))
text(0.8, 5, 'Model 1', col=util$c_dark)

util$plot_expectand_pushforward(samples2[[name]], 50,
                                display_name=name,
                                flim=c(0, 1),
                                col=util$c_mid,
                                border="#DDDDDD88",
                                add=TRUE)
text(0.2, 3, 'Model 2', col=util$c_mid)


name <- 'tau22'
util$plot_expectand_pushforward(samples1[[name]], 50,
                                display_name=name,
                                flim=c(0, 1), ylim=c(0, 10))
text(0.25, 4.5, 'Model 1', col=util$c_dark)

util$plot_expectand_pushforward(samples2[[name]], 50,
                                display_name=name,
                                flim=c(0, 1),
                                col=util$c_mid,
                                border="#DDDDDD88",
                                add=TRUE)
text(0.8, 9.5, 'Model 2', col=util$c_mid)
```



Without accounting for the full structure of the data generating process we cannot extract accurate inferences for the phenomenology of interest!

8 Conclusion

Hidden Markov models generalize mixture observational modeling by allowing for persistent dynamics in the behavior of the data generating process. What really makes hidden Markov models so powerful in practice is that the forward and backward algorithms allow us to efficiently marginalize out the component assignment variables. With care in how exactly we define the latent states and the component observational model outputs these marginalization algorithms allow hidden Markov models to be used in a wide range of sophisticated applications.

Appendix: State-Space Models

The construction of hidden Markov models is actually quite a bit more general than what we have considered in this chapter (Cappé, Moulines, and Rydén (2005)). In particular the mathematical results generalize pretty immediately to any mixture observational model, including not only discrete mixture models but also continuous mixture models. This is useful if, for example, we want to model the evolution of a continuous latent state such as the position and velocity of a moving object.

Most references reserve hidden Markov model for finite latent states, using the term **state space model** to refer to models with the same conditional structure but more general latent states. That said this convention is not universal and it never hurts to use redundant language, such as "discrete hidden Markov model" or even "finite hidden Markov model" when there is any possibility of confusion.

Generalizing the hidden Markov model construction presented in this chapter uses the same joint model,

$$p(z_{1:I}, y_{1:I} \mid \theta) = p(z_1) \, p(y_1 \mid 1, z_1, \theta) \prod_{i=2}^{I} p(z_i \mid z_{i-1}, \theta) \, p_{z_i}(y_i \mid 1, z_i, \theta).$$

In order to marginalize the latent states, however, we have to replace summations with more general expectations. For example in order to accommodate a real-valued latent state we need to replace the marginal model

$$p(y_{1:I} \mid \theta) = \sum_{z_1} \cdots \sum_{z_I} p(z_{1:I}, y_{1:I} \mid \theta)$$

with

$$p(y_{1:I} \mid \theta) = \int \mathrm{d}z_1 \cdots \mathrm{d}z_1 \, p(z_{1:I}, y_{1:I} \mid \theta).$$

In this case the forward algorithm becomes

$$p(z_i, y_{1:i} \mid \theta) = p(y_i \mid i, z_i, \theta) \int \mathrm{d}z_{i-1} \, p(z_i \mid z_{i-1}, \theta) \, p(z_{i-1}, y_{1:(i-1)} \mid \theta)$$

while the forward-backward algorithm becomes

$$p(y_{(i+1):I}, \mid z_i, \theta) = \int \mathrm{d}z_{i+1} \, p(y_{i+1} \mid i+1, z_{i+1}, \theta) \, p(y_{(i+2):I}, \mid z_i, \theta) \, p(z_{i+1} \mid z_i, \theta)$$

with

$$p(z_i \mid y_{1:I}, \theta) \propto p(z_i, y_{1:i} \mid \theta) \, p(y_{(i+1):I} \mid z_i, \theta).$$

The practical implementation of these more general forward and backward algorithms is not always feasible. In addition to needing all of the integrals to admit analytic solutions we also need the intermediate conditional probability density functions to all fall into the same family of probability density functions so that we have to keep track of only finite-dimensional parameters and not arbitrary, infinite-dimensional functions.

One exceptional circumstance where this implementation is feasible is when the observational and transition probability density functions are all multivariate normal. In this case each $p(z_i, y_{1:i} \mid \theta)$, $p(y_{(i+1):I}, \mid z_i, \theta)$, and $p(z_i \mid y_{1:I}, \theta)$ also become multivariate normal probability density functions that we can completely characterized with location and covariance parameters. The forward and backward algorithms then reduce to recursive updates of these parameters.

Even better the recursive updates all become linear operations on these parameters. Consequently this model is known as a **linear state space model**. Maximum likelihood estimation of the location parameters is known as a **Kalman filter**.

Acknowledgements

A very special thanks to everyone supporting me on Patreon: Adam Fleischhacker, Alejandro Navarro-Martínez, Alessandro Varacca, Alex D, Alexander Noll, Amit, Andrea Serafino, Andrew Mascioli, Andrew Rouillard, Ara Winter, Ari Holtzman, Austin Rochford, Aviv Keshet, Avraham Adler, Ben Matthews, Ben Swallow, Benoit Essiambre, boot, Bradley Kolb, Brendan Galdo, Bryan Chang, Brynjolfur Gauti Jónsson, Cameron Smith, Canaan Breiss, Cat Shark, Cathy Oliveri, Charles Naylor, Chase Dwelle, Chris Jones, Christina Van Heer, Christopher Mehrvarzi, Colin Carroll, Colin McAuliffe, Damien Mannion, dan mackinlay, Dan W Joyce, Dan Waxman, Dan Weitzenfeld, Daniel Edward Marthaler, Daniel Saunders, Danny Van Nest, Darshan Pandit, Darthmaluus , David Burdelski, David Wurtz, Doug Rivers, Dr. Jobo, Dr. Omri Har Shemesh, Dylan Maher, Dylan Spielman, Ed Cashin, Edgar Merkle, Eli Witus, Eric LaMotte, Ero Carrera, Eugene O'Friel, Felipe González, Fergus Chadwick, Finn Lindgren, Francesco Corona, Geoff Rollins, Granville Matheson, Gregor Gorjanc, Guilherme Marthe, Håkan Johansson, Hamed Bastan-Hagh, haubur, Hector Munoz, Henri Wallen, hs, Hugo Botha, Ian, Ian Costley, idontgetoutmuch, Ignacio Vera, Ilaria Prosdocimi, Isaac Vock, Isidor Belic, jacob pine, Jair Andrade, James C, James Hodgson, James Wade, Janek Berger, Jarrett Byrnes, Jason Martin, Jason Pekos, Jason Wong, jd, Jeff Burnett, Jeff Dotson, Jeff Helzner, Jeffrey Erlich, Jerry Lin , Jessica Graves, Joe Sloan, John Flournoy, Jonathan H. Morgan, Jonathon Vallejo, Joran Jongerling, Josh Knecht, Joshua Miller, JU, Julian Lee, June, Justin Bois, Kádár András, Karim Naguib, Karim Osman, Kristian Gårdhus Wichmann, Lars Barquist, lizzie , LOU ODETTE, Luís F, Mads Christian Hansen, Marek Kwiatkowski, Mariana Carmona, Mark Donoghoe, Markus P., Márton Vaitkus, Matthew, Matthew Kay, Matthew Mulvahill, Matthieu LEROY, Mattia Arsendi, Matěj, Maurits van der Meer, Max, Michael Colaresi, Michael DeWitt, Michael Dillon, Michael Lerner, Mick Cooney, Mike Lawrence, Mister-Mentat, N Sanders, N.S. , Name, Nathaniel Burbank, Nicholas Clark, Nicholas Cowie, Nick S, Nikita Karetnikov, Octavio Medina, Ole Rogeberg, Oliver Crook, Olivier Ma, Patrick Kelley, Patrick Boehnke, Pau Pereira Batlle, Peter Johnson, Pieter van den Berg , ptr, quasar, Ramiro Barrantes Reynolds, Raúl Peralta Lozada, Ravin Kumar, Rémi , Rex, Riccardo Fusaroli, Richard Nerland, Robert Frost, Robert Goldman, Robert kohn, Robin Taylor, Ryan Gan, Ryan Grossman, Ryan Kelly, Sean Wilson, Seth Axen, shira, Simon Duane, Simon Lilburn, Simon Steiger, Simone Sebben, sssz, Stefan Lorenz, Stephen Lienhard, Steve Harris, Steven Forrest, Stew Watts, Stone Chen, Susan Holmes, Svilup, Tate Tunstall, Tatsuo Okubo, Teresa Ortiz, Theodore Dasher, Thomas Kealy, Thomas Siegert, Thomas Vladeck, Tobychev , Tony

Wuersch, Tyler Burch, Virginia Fisher, Vitalie Spinu, Vladimir Markov, Wil Yegelwel, Will Farr, Will Lowe, Will Wen, woejozney, yolhaj, yureq, Zach A, and Zhengchen Cai.

References

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York.

Cappé, Olivier, Eric Moulines, and Tobias Rydén. 2005. *Inference in Hidden Markov Models*. Springer Series in Statistics. Springer, New York.

Curtis, Charles W. 1993. *Linear Algebra*. Fourth. Undergraduate Texts in Mathematics. Springer-Verlag, New York.

Papoulis, Athanasios, and S. Unnikrishna Pillai. 2002. *Probability, Random Variables, and Stochastic Processes*. Fourth. McGraw-Hill Book Co.

License

Original Computing Environment

```r
writeLines(readLines(file.path(Sys.getenv("HOME"), ".R/Makevars")))
```

```
CC=clang

CXXFLAGS=-O3 -mtune=native -march=native -Wno-unused-variable -Wno-unused-function -Wno-macr
CXX=clang++ -arch x86_64 -ftemplate-depth-256

CXX14FLAGS=-O3 -mtune=native -march=native -Wno-unused-variable -Wno-unused-function -Wno-ma
CXX14=clang++ -arch x86_64 -ftemplate-depth-256
```

```r
sessionInfo()
```

```
R version 4.3.2 (2023-10-31)
Platform: x86_64-apple-darwin20 (64-bit)
Running under: macOS Sonoma 14.7.5

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRlapack.dylib;

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/New_York
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] colormap_0.1.4     rstan_2.32.6       StanHeaders_2.32.7

loaded via a namespace (and not attached):
 [1] gtable_0.3.4       jsonlite_1.8.8     compiler_4.3.2     Rcpp_1.0.11
 [5] parallel_4.3.2     gridExtra_2.3      scales_1.3.0       yaml_2.3.8
 [9] fastmap_1.1.1      ggplot2_3.4.4      R6_2.5.1           curl_5.2.0
[13] knitr_1.45         tibble_3.2.1       munsell_0.5.0      pillar_1.9.0
[17] rlang_1.1.2        utf8_1.2.4         V8_4.4.1           inline_0.3.19
[21] xfun_0.41          RcppParallel_5.1.7 cli_3.6.2          magrittr_2.0.3
[25] digest_0.6.33      grid_4.3.2         lifecycle_1.0.4    vctrs_0.6.5
[29] evaluate_0.23      glue_1.6.2         QuickJSR_1.0.8     codetools_0.2-19
[33] stats4_4.3.2       pkgbuild_1.4.3     fansi_1.0.6        colorspace_2.1-0
[37] rmarkdown_2.25     matrixStats_1.2.0  tools_4.3.2        loo_2.6.0
[41] pkgconfig_2.0.3    htmltools_0.5.7
```

Stan
Program 1 `simu\_data.stan`

```
data {
  // Number of latent states
  int<lower=1> I;
}

transformed data {
  // True model configuration
  int<lower=1> K = 3;
  array[K] real mu = {-3, 2, 5};
  array[K] real<lower=0> sigma = {1.0, 1.5, 0.75};

  simplex[K] rho = [0.8, 0.2, 0.0 ]';

  matrix[K, K] Gamma = [ [ 0.60, 0.30, 0.10 ],
                         [ 0.40, 0.50, 0.10 ],
                         [ 0.05, 0.05, 0.90 ] ];
}

generated quantities {
  array[I] int<lower=1, upper=K> z; // Simulated latent states
  array[I] real y;                  // Simulated observations

  //  Initial state
  z[1] = categorical_rng(rho);
  y[1] = normal_rng(mu[z[1]], sigma[z[1]]);

  for (i in 2:I) {
    // Following state
    vector[K] lambda = Gamma[z[i - 1],]';
    z[i] = categorical_rng(lambda);

    // Observation
    y[i] = normal_rng(mu[z[i]], sigma[z[i]]);
  }
}
```

Stan
Program 2 hmm_log_sum_exp.stan

```
data {
  int<lower=0> K;  // Number of component observational models
  int<lower=0> I;  // Number of latent states
  array[I] real y; // Observations of each latent state
}

parameters {
  // Component observational model configurations
  ordered[K] mu;
  array[K] real<lower=0> sigma;

  // Latent state dynamics
  simplex[K] rho;
  array[K] simplex[K] gamma;
}

model {
  // Prior model
  target += normal_lpdf(mu | 0, 10 / 2.32);   // -10 <~ mu[k] <~ +10
  target += normal_lpdf(sigma | 0, 5 / 2.57); // 0 <~ sigma[k] <~ 5

  target += dirichlet_lpdf(rho | rep_vector(1, K));

  for (k in 1:K)
    target += dirichlet_lpdf(gamma[k] | rep_vector(1, K));

  // Observational model
  {
    // Construct transition matrix
    matrix[K, K] log_Gamma;
    for (k in 1:K) log_Gamma[k,] = log(gamma[k]');

    // Forward algorithm
    vector[K] log_alpha = log(rho);

    for (i in 1:I) {
      vector[K] log_alpha_prev = log_alpha;
      for (k in 1:K) {
        real log_lambda = log_sum_exp(log_Gamma[,k] + log_alpha_prev);
        log_alpha[k] =   log_lambda
                       + normal_lpdf(y[i] | mu[k], sigma[k]);
      }
    }
```

```
    // Marginal observational model
    target += log_sum_exp(log_alpha);
  }
}

generated quantities {
```

Stan
Program 3 hmm_rescaled.stan

```
data {
  int<lower=0> K;  // Number of component observational models
  int<lower=0> I;  // Number of latent states
  array[I] real y; // Observations of each latent state
}

parameters {
  // Component observational model configurations
  ordered[K] mu;
  array[K] real<lower=0> sigma;

  // Latent state dynamics
  simplex[K] rho;
  array[K] simplex[K] gamma;
}

model {
  // Prior model
  target += normal_lpdf(mu | 0, 10 / 2.32);   // -10 <~ mu[k] <~ +10
  target += normal_lpdf(sigma | 0, 5 / 2.57); // 0 <~ sigma[k] <~ 5

  target += dirichlet_lpdf(rho | rep_vector(1, K));

  for (k in 1:K)
    target += dirichlet_lpdf(gamma[k] | rep_vector(1, K));

  // Observational model
  {
    // Construct transition matrix
    matrix[K, K] Gamma;
    for (k in 1:K) Gamma[k,] = gamma[k]';

    // Forward algorithm
    real norm;
    real log_norm = 0;

    vector[K] alpha = rho;

    norm = max(alpha);
    log_norm += log(norm);
    alpha /= norm;

    for (i in 1:I) {
      vector[K] omega;
      for (k in 1:K)
        omega[k] = exp(normal_lpdf(y[i] | mu[k], sigma[k]));

      alpha = omega .* (Gamma' * alpha);

      norm = max(alpha);
```

Stan
Program 4 hmm_builtin.stan

```
data {
  int<lower=0> K;  // Number of component observational models
  int<lower=0> I;  // Number of latent states
  array[I] real y; // Observations of each latent state
}

parameters {
  // Component observational model configurations
  ordered[K] mu;
  array[K] real<lower=0> sigma;

  // Latent state dynamics
  simplex[K] rho;
  array[K] simplex[K] gamma;
}

model {
  // Prior model
  target += normal_lpdf(mu | 0, 10 / 2.32);   // -10 <~ mu[k] <~ +10
  target += normal_lpdf(sigma | 0, 5 / 2.57); // 0 <~ sigma[k] <~ 5

  target += dirichlet_lpdf(rho | rep_vector(1, K));

  for (k in 1:K)
    target += dirichlet_lpdf(gamma[k] | rep_vector(1, K));

  // Observational model
  {
    // Construct transition matrix
    matrix[K, K] Gamma;
    for (k in 1:K) Gamma[k,] = gamma[k]';

    // Construct component observational model evaluations
    matrix[K, I] log_omega;
    for (i in 1:I)
      for (k in 1:K)
        log_omega[k, i] = normal_lpdf(y[i] | mu[k], sigma[k]);

    // Marginal observational model
    target += hmm_marginal(log_omega, Gamma, rho);
  }
}
```

```
generated quantities {
  // Marginal latent state posterior probabilities
  array[I] simplex[K] z_prob;

  array[I] int z;       // Posterior latent states
  array[I] real y_pred; // Posterior predictive observations
```

Stan
Program 5 `static\_mixure.stan`

```
data {
  int<lower=0> K;  // Number of component observational models
  int<lower=0> I;  // Number of latent states
  array[I] real y; // Observations at each step
}

parameters {
  // Component observational model configurations
  ordered[K] mu;
  array[K] real<lower=0> sigma;

  // Homogeneous mixture probabilities
  simplex[K] lambda;
}

model {
  // Prior model
  target += normal_lpdf(mu | 0, 10 / 2.32);   // -10 <~ mu[k] <~ +10
  target += normal_lpdf(sigma | 0, 5 / 2.57); // 0 <~ sigma[k] <~ 5
  target += dirichlet_lpdf(lambda | rep_vector(1, K));

  // Observational model
  for (n in 1:I) {
    array[K] real lpds;
    for (k in 1:K) {
      lpds[k] =   log(lambda[k])
                + normal_lpdf(y[n] | mu[k], sigma[k]);
    }
    target += log_sum_exp(lpds);
  }
}

generated quantities {
  array[I] real y_pred; // Posterior predictive observations

  for (i in 1:I) {
    int z = categorical_rng(lambda);
    y_pred[i] = normal_rng(mu[z], sigma[z]);
  }
}
```

Stan
Program 6 hmm_builtin_miss.stan

```
data {
  int<lower=0> K;  // Number of component observational models
  int<lower=0> I;  // Number of latent states
  int<lower=1, upper=I> N; // Number of observations

  array[N] int<lower=1, upper=I> state; // Observed latent states
  array[N] real y;                      // Observations
}

transformed data {
  // Surrogate value for unobserved latent states
  real miss_val = -25.0;

  // Buffer observations with surrogate
  // values for unobserved latent states
  array[I] real y_buff = rep_array(miss_val, I);
  y_buff[state] = y;

  // Binary variable indicating whether or
  // not each latent state is observed
  array[I] int<lower=0, upper=1> obs_flag = rep_array(0, I);
  obs_flag[state] = rep_array(1, N);
}

parameters {
  // Component observational model configurations
  ordered[K] mu;
  array[K] real<lower=0> sigma;

  // Latent state dynamics
  simplex[K] rho;
  array[K] simplex[K] gamma;
}

model {
  // Prior model
  target += normal_lpdf(mu | 0, 10 / 2.32);   // -10 <~ mu[k] <~ +10
  target += normal_lpdf(sigma | 0, 5 / 2.57); // 0 <~ sigma[k] <~ 5

  target += dirichlet_lpdf(rho | rep_vector(1, K));

  for (k in 1:K)
    target += dirichlet_lpdf(gamma[k] | rep_vector(1, K));

  // Observational model
  {
    // Construct transition matrix
    matrix[K, K] Gamma;
    for (k in 1:K) Gamma[k,] = gamma[k]';
```

Stan
Program 7 hmm_rescaled_partial.stan

```
data {
  int<lower=0> K;  // Number of component observational models
  int<lower=0> I;  // Number of latent states
  int<lower=1, upper=I> N; // Number of observations

  array[N] int<lower=1, upper=I> state; // Observed latent states
  array[N] real y;                      // Observations
}

parameters {
  // Component observational model configurations
  ordered[K] mu;
  array[K] real<lower=0> sigma;

  // Latent state dynamics
  simplex[K] rho;
  array[K] simplex[K] gamma;
}

model {
  // Prior model
  target += normal_lpdf(mu | 0, 10 / 2.32);   // -10 <~ mu[k] <~ +10
  target += normal_lpdf(sigma | 0, 5 / 2.57); // 0 <~ sigma[k] <~ 5

  target += dirichlet_lpdf(rho | rep_vector(1, K));

  for (k in 1:K)
    target += dirichlet_lpdf(gamma[k] | rep_vector(1, K));

  // Observational model
  {
    // Construct baseline transition matrix
    matrix[K, K] Gamma;
    for (k in 1:K) Gamma[k,] = gamma[k]';

    // Forward algorithm, jumping across any gaps of unobserved
    // latent states with a heterogeneous transition matrix
    real norm;
    real log_norm = 0;

    vector[K] alpha = rho;

    norm = max(alpha);
    log_norm += log(norm);
    alpha /= norm;

    for (n in 1:N) {
      vector[K] omega;
      for (k in 1:K)
        omega[k] = exp(normal_lpdf(y[n] | mu[k], sigma[k]));
```

Stan
Program 8 hmm_analysis1.stan

```
data {
  int<lower=0> I;  // Number of latent states
  int<lower=0> N;  // Number of observations

  // All observations
  array[N] int<lower=0> y;

  // Organization of observations across latent states
  array[I] int<lower=0, upper=N> state_N;
  array[I] int<lower=1, upper=N> state_start_idx;
  array[I] int<lower=1, upper=N> state_end_idx;
}

transformed data {
  int<lower=0> K = 2; // Number of component observational models
}

parameters {
  // Component observational model configurations
  real<lower=0>       lambda1; // Low environmental activity
  real<lower=lambda1> lambda2; // High environmental activity

  // Hidden state dynamics
  simplex[K] rho;

  // Low activity to low activity transition probability
  real<lower=0, upper=1> tau11;

  // High activity to high activity transition probability
  real<lower=0, upper=1> tau22;
}

model {
  // Prior model
  target += normal_lpdf(lambda1 | 0,  5 / 2.57); // 0 <~ lambda1 <~ 5
  target += normal_lpdf(lambda2 | 0, 15 / 2.57); // 0 <~ lambda2 <~ 15

  target += dirichlet_lpdf(rho | rep_vector(1, K));

  target += beta_lpdf(tau11 | 1, 1);
  target += beta_lpdf(tau22 | 1, 1);

  // Observational model
  {
```

```
    // Construct transition matrix
    matrix[K, K] Gamma
      = [ [ tau11,     1 - tau11 ],
          [ 1 - tau22, tau22     ] ];

    // Construct component observational model evaluations
```

Stan
Program 9 hmm_analysis2.stan

```
data {
  int<lower=0> I;  // Number of latent states
  int<lower=0> N;  // Number of observations

  // All observations
  array[N] int<lower=0> y;

  // Organization of observations across latent states
  array[I] int<lower=0, upper=N> state_N;
  array[I] int<lower=1, upper=N> state_start_idx;
  array[I] int<lower=1, upper=N> state_end_idx;
}

transformed data {
  int<lower=0> K = 4;  // Number of component observational models

  // z1: Detector status
  // z2: Active component observational model
  // z = 1: (z1 = 1, z2 = 1) Working detector, low environmental activity
  // z = 2: (z1 = 1, z2 = 2) Working detector, high environmental activity
  // z = 3: (z1 = 2, z2 = 1) Broken detector,  low environmental activity
  // z = 4: (z1 = 2, z2 = 2) Broken detector,  high environmental activity
}

parameters {
  // Component observational model configurations
  real<lower=0>       lambda1; // Low environmental activity
  real<lower=lambda1> lambda2; // High environmental activity

  // Hidden state dynamics
  simplex[K] rho;

  // Low activity to low activity transition probability
  real<lower=0, upper=1> tau11;

  // High activity to high activity transition probability
  real<lower=0, upper=1> tau22;

  // Working detector to working detector transition probability
  real<lower=0, upper=1> taud;
}

model {
  // Prior model
  target += normal_lpdf(lambda1 | 0,  5 / 2.57); // 0 <~ lambda1 <~ 5
  target += normal_lpdf(lambda2 | 0, 15 / 2.57); // 0 <~ lambda2 <~ 15

  target += dirichlet_lpdf(rho | rep_vector(1, K));

  target += beta_lpdf(tau11 | 1, 1);
```