Bayesian Analysis (2018) **13**, Number 3, pp. 797–822

Sampling Latent States for High-Dimensional Non-Linear State Space Models with the Embedded HMM Method

Alexander Y. Shestopaloff* and Radford M. Neal†

Abstract. We propose a new scheme for selecting pool states for the embedded Hidden Markov Model (HMM) Markov Chain Monte Carlo (MCMC) method. This new scheme allows the embedded HMM method to be used for efficient sampling in state space models where the state can be high-dimensional. Previously, embedded HMM methods were only applicable to low-dimensional state-space models. We demonstrate that using our proposed pool state selection scheme, an embedded HMM sampler can have similar performance to a well-tuned sampler that uses a combination of Particle Gibbs with Backward Sampling (PGBS) and Metropolis updates. The scaling to higher dimensions is made possible by selecting pool states locally near the current value of the state sequence. The proposed pool state selection scheme also allows each iteration of the embedded HMM sampler to take time linear in the number of the pool states, as opposed to quadratic as in the original embedded HMM sampler.

MSC 2010 subject classifications: Primary 65C40; secondary 65C05.

Keywords: MCMC, non-linear, state space.

1 Introduction

Consider a non-linear, non-Gaussian state space model for an observed sequence $\mathbf{y} = (y_1, \ldots, y_n)$. This model, with parameters θ, assumes that the Y_i are drawn from an observation density $p(y_i|x_i, \theta)$, where X_i is an unobserved Markov process with initial density $p(x_1|\theta)$ and transition density $p(x_i|x_{i-1}, \theta)$. Here, the x_i might be either continuous or discrete. We may be interested in inferring both the realized values of the Markov process $\mathbf{x} = (x_1, \ldots, x_n)$ and the model parameters θ. In a Bayesian approach to this problem, this can be done by drawing a sample of values for $\mathbf{x}$ and θ using a Markov chain that alternately samples from the conditional posterior distributions $p(\mathbf{x}|\theta, \mathbf{y})$ and $p(\theta|\mathbf{x}, \mathbf{y})$. In this paper, we will only consider inference for $\mathbf{x}$ by sampling from $p(\mathbf{x}|\theta, \mathbf{y})$, taking the parameters θ to be known. As a result, we will omit θ in model densities for the rest of the paper. Except for linear Gaussian models and models with a finite state space, this sampling problem has no exact solution and hence approximate methods such as MCMC must be used.

One method for sampling state sequences in non-linear, non-Gaussian state space models is the embedded HMM method (Neal, 2003; Neal et al., 2004). An embedded

*Department of Statistical Sciences, University of Toronto, alexander@utstat.utoronto.ca
†Department of Statistical Sciences & Department of Computer Science, University of Toronto, radford@utstat.utoronto.ca

 https://doi.org/10.1214/17-BA1077

HMM update proceeds as follows. First, at each time i, a set of L "pool states" in the latent space is constructed. In this set, $L-1$ of the pool states are drawn from a chosen pool state density and one is the current value of x_i. This step can be thought of as temporarily reducing the state space model to an HMM with a finite set of L states, hence the name of the method. Then, using efficient forward-backward computations, which take time proportional to $L^2 n$, a new sequence $\mathbf{x}'$ is selected from the "ensemble" of L^n sequences passing through the sets of pool states, with the probability of choosing each sequence proportional to its posterior density divided by the probability of the sequence under the pool state density. At the next iteration of the sampler, new sets of pool states are constructed, so that the chain can sample all possible x_i, even when the state space is not finite.

Another method is the Particle Gibbs with Backward Sampling (PGBS) method. The Particle Gibbs (PG) method is an example of a particle MCMC method and was first introduced in Andrieu et al. (2010); Whiteley (2010) suggested the backward sampling modification. Lindsten and Schön (2012) implemented backward sampling and showed that it improves the efficiency of PG. Particle MCMC methods use Sequential Monte Carlo (SMC) (Doucet et al., 2001) to produce MCMC moves for parameter and state updates. Starting with a current sequence $\mathbf{x}$, PGBS first uses conditional SMC to construct a set of candidate sequences and then uses backward sampling to select a new sequence from the set of candidate ones. Here, conditional SMC works in the same way as ordinary SMC when generating a set of particles, except that one of the particles at time i is always set to the current x_i, similar to what is done in the embedded HMM method, which allows the sampler to remain at x_i if x_i lies in a high-density region. While this method works well for problems with low-dimensional state spaces, the reliance of the SMC procedure on choosing an appropriate importance density can make it challenging to make the method work in high dimensions. An important advantage of Particle Gibbs, however, is that each iteration has time complexity that is only linear in the number of particles.

Both the PGBS and embedded HMM methods can facilitate sampling of a latent state sequence, $\mathbf{x}$, when there are strong temporal dependencies amongst the x_i. In this case, using a method that samples x_i conditional on fixed values of x_{i-1} and x_{i+1} can be an inefficient way of producing a sample from $p(\mathbf{x}|\mathbf{y}, \theta)$, because the conditional density of x_i given x_{i-1} and x_{i+1} can be highly concentrated relative to the marginal density of x_i. In contrast, with the embedded HMM and PGBS methods it is possible to make changes to blocks of x_i's at once. This allows larger changes to the state in each iteration of the sampler, making updates more efficient. However, good performance of the embedded HMM and PGBS methods relies on appropriately choosing the set of pool states or particles at each time i.

In this paper, our focus will be on techniques for choosing pool states for the embedded HMM method. When the latent state space is one-dimensional, embedded HMMs work well when choosing pool states in a variety of ways. For example, in Shestopaloff and Neal (2013), we choose pool states at each time i by constructing a "pseudo-posterior" for each latent variable by taking the product of a "pseudo-prior" and the observation density, the latter treated as a "pseudo-likelihood" for the latent variable.

In Shestopaloff and Neal (2014), we choose pool states at each time i by sampling from the marginal prior density of the latent process.

Ways of choosing pool states that work well in one dimension begin to exhibit problems when applied to models with higher-dimensional state spaces. This is true even when the dimension of the state space is as small as three. Since these schemes are designed to produce sets of pool states without reference to the current point, as the dimension of the latent space grows, a higher proportion of the sequences in the ensemble ends up having low posterior density. Ensuring that performance does not degrade in higher dimensions thus requires a significant increase in the number of pool states. As a result, computation time may grow so large that any advantage that comes from using embedded HMMs is eliminated. One advantage of the embedded HMM method over PGBS is that the embedded HMM construction allows placing pool states locally near the current value of x_i, potentially allowing the method to scale better with the dimensionality of the state space. Switching to such a local scheme fixes the problem to some extent. However, local pool state schemes come with their own problems. For example, local schemes make it difficult to handle models with multiple posterior modes that are well-separated — the pool states might end up being placed near only some of the modes. Another issue with previously used pool state selection schemes is that they select pool states independently across time, which becomes increasingly inefficient in higher dimensions in the presence of strong temporal dependencies in $\mathbf{x}$.

In this paper, we propose an embedded HMM sampler suitable for models where the state space is high dimensional. This sampler uses a sequential approximation to the density $p(x_i|y_1, \ldots, y_i)$ or to the density $p(x_i|y_{i+1}, \ldots, y_n)$ as the pool state density. We show that by using this pool state density, together with an efficient MCMC scheme for sampling from it, we can reduce the cost per iteration of the embedded HMM sampler to be proportional to nL, as with PGBS. At the same time, we retain the ability to generate pool states locally, potentially allowing better scaling for high-dimensional state spaces. Our proposed scheme can thus be thought of as combining the best features of the PGBS and the embedded HMM methods, while overcoming the deficiencies of both. We use two sample state space models as examples. Both have Gaussian latent processes and Poisson observations, with one model having a unimodal posterior and the second a multimodal one. For the multimodal example, we introduce a "mirroring" technique that allows efficient movement between the different posterior modes. For these models, we show how our proposed embedded HMM method compares to a simple Metropolis sampler, a PGBS sampler, as well as a sampler that combines PGBS and simple Metropolis updates. Further details on ensemble methods are available in the PhD thesis of Shestopaloff (2016).

2 Embedded HMM MCMC

We review the embedded HMM method (Neal, 2003; Neal et al., 2004) here. We take the model parameters, θ, to be fixed, so we do not write them explicitly. Let $p(x)$ be the density from which the state at time 1 is drawn, let $p(x_i|x_{i-1})$ be the transition density between states at times i and $i-1$, and let $p(y_i|x_i)$ be the density of the observation y_i given x_i. For this and later methods, we assume that the initial and transition densities

can be feasibly computed at least up to some normalizing constant. We also assume that the observation density (given state) can be computed up to a normalizing constant. For each time, i, we choose a pool state density, κ_i, which we must be able to feasibly compute. This must be a density that assigns a positive probability to all possible state values, which lets the sampler explore the space of all possible latent sequences.

Suppose our current sequence is $\mathbf{x} = (x_1, \ldots, x_n)$. The embedded HMM sampler updates $\mathbf{x}$ to $\mathbf{x}'$ as follows.

First, at each time $i = 1, \ldots, n$, we generate a set of L pool states, denoted by $\mathcal{P}_i = \{x_i^{[1]}, \ldots, x_i^{[L]}\}$. The pool states are sampled independently across the different times i. We choose $l_i \in \{1, \ldots, L\}$ uniformly at random and set $x_i^{[l_i]}$ to x_i. We sample the remaining $L-1$ pool states $x_i^{[1]}, \ldots, x_i^{[l_i-1]}, x_i^{[l_i+1]}, \ldots, x_n^{[L]}$ using a Markov chain that leaves κ_i invariant, as follows. Let $R_i(x'|x)$ be the transitions of this Markov chain with $\tilde{R}_i(x|x')$ the transitions for this Markov chain reversed (i.e. $\tilde{R}_i(x|x') = R_i(x'|x)\kappa_i(x)/\kappa_i(x')$), so that

$$\kappa_i(x) R_i(x'|x) = \kappa_i(x') \tilde{R}_i(x|x') \tag{1}$$

for all x and x'. Then, starting at $j = l_i - 1$, use reverse transitions $\tilde{R}_i(x_i^{[j]}|x_i^{[j+1]})$ to generate $x_i^{[l_i-1]}, \ldots, x_i^{[1]}$ and starting at $j = l_i + 1$ use forward transitions $R_i(x_i^{[j]}|x_i^{[j-1]})$ to generate $x_i^{[l_i+1]}, \ldots, x_n^{[L]}$. As examples, both transitions R_i and $\tilde{R}_i$ can be those of a Metropolis sampler (which is reversible), or R_i can be a systematic Gibbs sampling scan, and $\tilde{R}_i$ the corresponding Gibbs sampling scan updating components in the reverse order. We only need to be able to sample from these transition distributions, not compute their densities.

At each $i = 1, \ldots, n$, we then compute the (unnormalized) forward probabilities $\alpha_i(x)$, with x taking values in $\mathcal{P}_i$. At time $i = 1$, we have

$$\alpha_1(x) = \frac{p(x) p(y_1|x)}{\kappa_1(x)} \tag{2}$$

and at times $i = 2, \ldots, n$, we have

$$\alpha_i(x) = \frac{p(y_i|x)}{\kappa_i(x)} \sum_{l=1}^{L} p(x|x_{i-1}^{[l]}) \alpha_{i-1}(x_{i-1}^{[l]}). \tag{3}$$

Finally, we sample a new state sequence $\mathbf{x}'$ using a stochastic backwards pass. This is done by first selecting l'_n with probabilities proportional to $\alpha_n(x_n^{[l'_n]})$ and setting $x'_n = x_n^{[l'_n]}$. We then go backwards from $i = n-1$ to $i = 1$, selecting l'_i with probabilities proportional to $\alpha_i(x_i^{[l'_i]}) p(x'_{i+1}|x_i^{[l'_i]})$ and setting $x'_i = x_i^{[l'_i]}$. Note that only the relative values of the $\alpha_i(x)$ will be required, so the α_i may be computed up to some constant factor, which may be convenient for avoiding floating point overflow.

Alternatively, given sets of pool states, embedded HMM updates can be done by first computing the backward probabilities. For the original embedded HMM method of (Neal, 2003; Neal et al., 2004), the backward probability formulation is probabilistically equivalent to the forward one. Setting $\beta_n(x) = 1$ for all $x \in \mathcal{P}_n$, we compute for $i < n$

$$\beta_i(x) = \frac{1}{\kappa_i(x)} \sum_{l=1}^{L} p(y_{i+1}|x_{i+1}^{[l]}) p(x_{i+1}^{[l]}|x) \beta_{i+1}(x_{i+1}^{[l]}). \tag{4}$$

A new state sequence $\mathbf{x}'$ is then sampled using a stochastic forward pass. We first select l'_1 with probabilities proportional to $\beta_1(x_1^{[l'_1]}) p(x_1^{[l'_1]}) p(y_1|x_1^{[l'_1]})$ and set $x'_1 = x_1^{[l'_1]}$. We then go forward, selecting l'_i for $i = 2, \ldots, n$ with probabilities proportional to $\beta_i(x_i^{[l'_i]}) p(x_i^{[l'_i]}|x'_{i-1}) p(y_i|x_i^{[l'_i]})$ and setting $x'_i = x_i^{[l'_i]}$.

Computing the α_i or β_i at each time $i > 1$ takes time proportional to L^2, since for each of the L pool states it takes time proportional to L to compute the sums in (3) or (4). Hence each iteration of the embedded HMM sampler takes time proportional to $L^2 n$.

Each iteration of the sampler takes memory proportional to nL. This is because to sample a new sequence, we need to perform a forward pass followed by a backwards pass, which requires the stored forward probabilities for each of the L pool states for each time i.

Algorithm 1 Embedded HMM MCMC Sampler

Require: $\mathbf{x}$, L, and $\tilde{R}_i$, R_i, κ_i for $i = 1, \ldots, n$

for $i = 1$ to n **do**
  $l_i \sim \text{Unif}\{1, \ldots, L\}$
  $x_i^{[l_i]} \leftarrow x_i$
  for $j = l_i - 1$ down to 1 **do**
    Use $\tilde{R}_i(x_i^{[j]}|x_i^{[j+1]})$ to sample $x_i^{[j]}$ from κ_i
  end for
  for $j = l_i + 1$ to L **do**
    Use $R_i(x_i^{[j]}|x_i^{[j-1]})$ to sample $x_i^{[j]}$ from κ_i
  end for
  for $j = 1$ to L **do**
    Compute $\alpha_i(x_i^{[j]})$
  end for
end for
Sample l'_n with probabilities $P(l'_n = k) \propto \alpha_n(x_n^{[k]})$ with $k \in \{1, \ldots, L\}$
$x'_n \leftarrow x_n^{[l'_n]}$
for $i = n - 1$ to 1 **do**
  Sample l'_i with probabilities $P(l'_i = k) \propto \alpha_i(x_i^{[k]}) p(x'_{i+1}|x_i^{[k]})$ with $k \in \{1, \ldots, L\}$
  $x'_i \leftarrow x_i^{[l'_i]}$
end for

3 Particle Gibbs with Backward Sampling MCMC

We review the Particle Gibbs with Backward Sampling (PGBS) sampler here. For full details, see the articles by Andrieu et al. (2010) and Lindsten and Schön (2012).

Let $q_1(x|y_1)$ be the importance density from which we sample particles at time 1, and let $q_i(x|y_i, x_{i-1})$ be the importance density for sampling particles at times $i > 1$. These may depend on the current value of the parameters, θ, which we suppressed in this notation. Suppose we start with a current sequence $\mathbf{x}$. We set the first particle $x_1^{[1]}$ to the current state x_1. We then sample $L-1$ particles $x_1^{[2]}, \ldots, x_1^{[L]}$ from q_1 and compute and normalize the weights of the particles:

$$w_1^{[l]} = \frac{p(x_1^{[l]})p(y_1|x_1^{[l]})}{q_1(x_1^{[l]}|y_1)}, \tag{5}$$

$$W_1^{[l]} = \frac{w_1^{[l]}}{\sum_{m=1}^{L} w_1^{[m]}} \tag{6}$$

for $l = 1, \ldots, L$.

For $i > 1$, we proceed sequentially. We first set $x_i^{[1]} = x_i$. We then sample a set of $L-1$ ancestor indices for particles at time i, defined by $a_{i-1}^{[l]} \in \{1, \ldots, L\}$, for $l = 2, \ldots, L$, with probabilities proportional to $W_{i-1}^{[l]}$. The ancestor index for the first state, $a_{i-1}^{[1]}$, is 1. We then sample each of the $L-1$ particles, $x_i^{[l]}$, from $q_i(x|y_i, x_{i-1}^{[a_{i-1}^{[l]}]})$, for $l = 2, \ldots, L$, and compute and normalize the weights of the particles:

$$w_i^{[l]} = \frac{p(x_i^{[l]}|x_{i-1}^{[a_{i-1}^{[l]}]})p(y_i|x_i^{[l]})}{q_i(x_i^{[l]}|y_i, x_{i-1}^{[a_{i-1}^{[l]}]})}, \tag{7}$$

$$W_i^{[l]} = \frac{w_i^{[l]}}{\sum_{m=1}^{L} w_i^{[m]}}. \tag{8}$$

A new sequence taking values in the set of particles at each time is then selected using a backwards sampling pass. This is done by first selecting x'_n from the set of particles at time n with probabilities $W_n^{[l]}$ and then selecting the rest of the sequence going backward in time to time 1, setting x'_i to $x_i^{[l]}$ with probability

$$\frac{w_i^{[l]} p(x'_{i+1}|x_i^{[l]})}{\sum_{m=1}^{L} w_i^{[m]} p(x'_{i+1}|x_i^{[m]})}. \tag{9}$$

A common choice for q_1 is the model's initial density. For q_i where $i > 1$, a common choice is the model's transition density. We use these choices in this paper.

Note that each iteration of the PGBS sampler takes time proportional to nL, since it takes time proportional to L to create the set of particles at each time i, and to do one step of backward sampling. Since we need to store the entire set of particles to sample a new sequence, each iteration of PGBS takes memory proportional to nL.

Algorithm 2 Particle Gibbs with Backwards Sampling MCMC

Require: $\mathbf{x}$, L, and q_i for $i = 1, \ldots, n$

$x_1^{[1]} \leftarrow x_1$

Sample $x_1^{[2]}, \ldots, x_1^{[L]}$ from q_1

$w_1^{[l]} \leftarrow \frac{p(x_1^{[l]})p(y_1|x_1^{[l]})}{q_1(x_1^{[l]}|y_1)}$ for $l = 1, \ldots, L$

$W_1^{[l]} \leftarrow \frac{w_1^{[l]}}{\sum_{m=1}^{L} w_1^{[m]}}$ for $l = 1, \ldots, L$

for $i = 2$ to n **do**

  $x_i^{[1]} \leftarrow x_i$

  $a_{i-1}^{[1]} \leftarrow 1$

  for $l = 2$ to L **do**

    Sample $a_{i-1}^{[l]}$ with probabilities $P(a_{i-1}^{[l]} = k) \propto W_{i-1}^{[k]}$ with $k \in \{1, \ldots, L\}$

    Sample $x_i^{[l]} \sim q_i(x|y_i, x_{i-1}^{[a_{i-1}^{[l]}]})$

    $w_i^{[l]} \leftarrow \frac{p(x_i^{[l]}|x_{i-1}^{[a_{i-1}^{[l]}]})p(y_i|x_i^{[l]})}{q_i(x_i^{[l]}|y_i, x_{i-1}^{[a_{i-1}^{[l]}]})}$

  end for

  $W_i^{[l]} \leftarrow \frac{w_i^{[l]}}{\sum_{m=1}^{L} w_i^{[m]}}$ for $l = 1, \ldots, L$

end for

Sample l with probabilities $P(l = k) \propto W_n^{[k]}$ with $k \in \{1, \ldots, L\}$

$x'_n \leftarrow x_n^{[l]}$.

for $i = n - 1$ to 1 **do**

  Sample l with probabilities $P(l = k) \propto \frac{w_i^{[l]} p(x'_{i+1}|x_i^{[l]})}{\sum_{m=1}^{L} w_i^{[m]} p(x'_{i+1}|x_i^{[m]})}$ with $k \in \{1, \ldots, L\}$

  $x'_i \leftarrow x_i^{[l]}$

end for

4 An embedded HMM sampler for high dimensions

We propose two new ways, denoted f and b, of generating pool states for the embedded HMM sampler. Unlike previously-used pool state selection schemes, where pool states are selected independently at each time, our new schemes select pool states sequentially, with pool states at time i selected conditional on pool states at time $i-1$, or alternatively at time $i + 1$. As a result, the forward and backward formulations of an embedded HMM sampler using these pool state selection schemes are not equivalent, and can have different performance.

4.1 Pool state distributions

The first way to generate pool states is to use a forward pool state selection scheme, with a sequential approximation to $p(x_i|y_1, \ldots, y_i)$ as the pool state density. In particular, at time 1, we set the pool state distribution of our proposed embedded HMM sampler to

$$\kappa_1^f(x) \propto p(x)p(y_1|x). \tag{10}$$

As a result of (2), $\alpha_1(x)$ is constant (independent of x). At time $i > 1$, we set the pool state distribution to

$$\kappa_i^f(x|\mathcal{P}_{i-1}) \propto p(y_i|x) \sum_{a=1}^{L} p(x|x_{i-1}^{[a]}), \tag{11}$$

which makes $\alpha_i(x)$ constant for $i > 1$ as well (see (3)).

We then draw a sequence composed of these pool states with the forward probability implementation of the embedded HMM method, with the $\alpha_i(x)$'s all set to 1.

The second way is to instead use a backward pool state selection scheme, with a sequential approximation of $p(x_i|y_{i+1}, \ldots, y_n)$ as the pool state density. We begin by creating the pool $\mathcal{P}_n$, consisting of the current state x_n and the remaining $L-1$ pool states sampled from $p_n(x)$, the marginal density at time n, which is the same as $p(x)$ if the latent process is stationary. The backward probabilities $\beta_n(x)$, for x in $\mathcal{P}_n$, are then set to 1. At time $i < n$ we set the pool state densities to

$$\kappa_i^b(x|\mathcal{P}_{i+1}) \propto \sum_{a=1}^{L} p(y_{i+1}|x_{i+1}^{[a]}) p(x_{i+1}^{[a]}|x) \tag{12}$$

so that $\beta_i(x)$ is constant for all $i = 1, \ldots, n$ (see (4)).

We then draw a sequence composed of these pool states as in the backward probability implementation of the embedded HMM method, with the $\beta_i(x)$'s all set to 1.

If the latent process is Gaussian, and the latent state at time 1 is sampled from the stationary distribution of the latent process, it is possible to update the latent variables by applying the forward scheme to the reversed sequence $(y_n, \ldots, y_1)$ by making use of time reversibility, since X_n is also sampled from the stationary distribution, and the latent process evolves backward in time according to the same transition density as it would going forward. We then use the forward pool state selection scheme along with a stochastic backward pass to sample a sequence $(x_n, \ldots, x_1)$, starting with x_1 and going to x_n.

It can sometimes be advantageous to alternate between using forward and backward (or, alternatively, forward applied to the reversed sequence) embedded HMM updates, since this can improve sampling of certain x_i. The sequential pool state selection schemes use only part of the observed sequence in generating the pool states. By alternating update directions, the pool states can depend on different parts of the observed data, potentially allowing us to better cover the region where x_i has high posterior density. For example, at time 1, the pool state density may disperse the pool states too widely, leading to poor sampling for x_1, but sampling x_1 using a backwards scheme can be much better, since we are now using all of the data in the sequence when sampling pool states at time 1.

4.2 Sampling pool states

To sample from κ_i^f or κ_i^b, we can use any Markov transitions R_i that leave this distribution invariant. The validity of the method does not depend on the Markov transitions for sampling from κ_i^f or κ_i^b reaching equilibrium or even on them being ergodic.

Directly using these pool state densities in an MCMC routine leads to a computational cost per iteration that is proportional to nL^2, like in the original embedded HMM method, since at times $i > 1$ we need at least L updates to produce L pool states, and the cost of computing an acceptance probability is proportional to L.

However, it is possible to reduce the cost per iteration of the embedded HMM method to be proportional to nL when we use κ_i^f or κ_i^b as the pool state densities. To do this, we start by thinking of the pool state densities at each time $i > 1$ as marginal densities summing over the variable $a = 1, \ldots, L$ that indexes a pool state at the previous time. Specifically, κ_i^f can be viewed as a marginal of the density

$$\lambda_i(x, a) \propto p(y_i|x)p(x|x_{i-1}^{[a]}) \tag{13}$$

while κ_i^b is a marginal of the density

$$\gamma_i(x, a) \propto p(y_{i+1}|x_{i+1}^{[a]})p(x_{i+1}^{[a]}|x). \tag{14}$$

Both of these densities are defined given a pool $\mathcal{P}_{i-1}$ at time $i-1$ or pool $\mathcal{P}_{i+1}$ at time $i+1$.

For the embedded HMM sampler using λ_i (or γ_i) as the pool state density, the sets of pool states at times $i > 1$ are defined as consisting of values of both x and a, hence $\mathcal{P}_1 = \{x_1^{[1]}, \ldots, x_1^{[L]}\}$ and $\mathcal{P}_i = \{(x_i^{[1]}, a_i^{[1]}), \ldots, (x_i^{[L]}, a_i^{[L]})\}$. We then use Markov transitions, R_i, to sample a set of values of x and a, with probabilities proportional to λ_i for the forward scheme, or probabilities proportional to γ_i for the backward scheme. This technique is reminiscent of the auxiliary particle filter of Pitt and Shephard (1999).

The chain is started at x set to the current x_i, and the initial value of a is chosen randomly with probabilities proportional to $p(x_i|x_{i-1}^{[a]})$ for the forward scheme or $p(y_{i+1}|x_{i+1}^{[a]})p(x_{i+1}^{[a]}|x_i)$ for the backward scheme. This stochastic initialization of a is needed to make the algorithm valid when we use λ_i or γ_i to generate the pool states.

Sampling values of x and a from λ_i or γ_i can be done by updating each of x and a separately, alternately sampling values of x conditional on a, and values of a conditional on x, or by updating x and a jointly, or by a combination of these.

Updating x given a can be done with any appropriate sampler, such as Metropolis. To update a given x we can also use Metropolis updates, proposing $a' = a + k$, with a drawn from some proposal distribution on $\{-K, \ldots, -1, 1, \ldots, K\}$. Alternatively, we can simply propose a' uniformly at random from $\{1, \ldots, L\}$.

Another possibility is to simultaneously update x and a at time $i > 1$ by proposing to update (x, a) to (x', a') where a' is proposed in any valid way while x' is chosen in a

way such that x' and $x_{i-1}^{[a']}$ are linked in the same way as x and $x_{i-1}^{[a]}$. This "shift" update makes it easier to generate a set of pool states at time i with different predecessor states at time $i-1$, helping to ensure that the pool states are well-dispersed. This update is accepted with the usual Metropolis probability.

For a concrete example of a shift update we use in our Experiments section, suppose that the latent process is an autoregressive Gaussian process of order 1, with the model being that $X_i|x_{i-1} \sim N(\Phi x_{i-1}, \Sigma)$. In this case, given a', we propose $x_i' = x_i + \Phi(x_{i-1}^{[a']} - x_{i-1}^{[a]})$. This update is accepted with probability

$$\min\left(1, \frac{p(y_i|x_i')}{p(y_i|x_i)}\right) \tag{15}$$

as a result of the transition densities in the acceptance ratio cancelling out, since

$$\begin{aligned} x_i' - \Phi x_{i-1}^{[a']} &= x_i + \Phi(x_{i-1}^{[a']} - x_{i-1}^{[a]}) - \Phi x_{i-1}^{[a']} \\ &= x_i - \Phi x_{i-1}^{[a]}. \end{aligned} \tag{16}$$

To be useful, shift updates normally need to be combined with other updates for generating pool states. When combining shift updates with other updates, tuning of acceptance rates for both updates needs to be done carefully in order to ensure that the shift updates actually improve sampling performance. In particular, if the pool states at time $i-1$ are spread out too widely, then the shift updates may have a low acceptance rate and not be very useful. Therefore, jointly optimizing proposals for x and for (x, a) may lead to a relatively high acceptance rate on updates of x, in order to ensure that the acceptance rate for the shift updates is not low.

We can also design specialized updates that are tailored to the model we are using. As an example of such an update, we propose a "flip" update to handle multimodality in one of the models in our Experiments section.

Like with the original embedded HMM sampler, each iteration of this new embedded HMM sampler takes memory proportional to nL. This is because we always need to store the entire set of pool states — since they are generated sequentially, there is no short-cut to compute the pool states at time i when performing a backward pass to sample a new sequence.

It is possible to reduce this memory requirement at the cost of recomputing pool states. For example, we can store pool states only at every k-th time step, and recompute intermediate pool states starting from the nearest set of pool states as needed. Such a choice of memory-time tradeoff would depend on the application. Similar considerations apply to the original embedded HMM and the PGBS methods.

4.3 Relation to particle MCMC

Finke et al. (2016) build on an earlier preprint (Shestopaloff and Neal, 2016) of the current article and study the connections between embedded HMM and particle MCMC

Algorithm 3 Embedded HMM MCMC for High Dimensions

Require: $\mathbf{x}$, L, and $\tilde{R}_i$, R_i for $i = 1, \ldots, n$
1: $l_1 \sim \text{Unif}\{1, \ldots, L\}$
2: $x_1^{[l_1]} \leftarrow x_1$
3: **for** $j = l_i - 1$ down to 1 **do**
4: Use $\tilde{R}_1(x_1^{[j]} | x_1^{[j+1]})$ to sample $x_1^{[j]}$ from κ_1^f
5: **end for**
6: **for** $j = l_i + 1$ to L **do**
7: Use $R_1(x_1^{[j]} | x_1^{[j-1]})$ to sample $x_1^{[j]}$ from κ_1^f
8: **end for**
9: **for** $i = 2$ to n **do**
10: $l_i \sim \text{Unif}\{1, \ldots, L\}$
11: $x_i^{[l_i]} \leftarrow x_i$
12: Sample $a_i^{[l_i]}$ with probabilities $P(a_i^{[l_i]} = k) \propto p(x_i | x_{i-1}^{[k]})$ with $k \in \{1, \ldots, L\}$
13: **for** $j = l_i - 1$ down to 1 **do**
14: Use $\tilde{R}_i(x_i^{[j]}, a_i^{[j]} | x_i^{[j+1]}, a_i^{[j+1]})$ to sample $(x_i^{[j]}, a_i^{[j]})$ from $\lambda_i(x, a)$
15: **end for**
16: **for** $j = l_i + 1$ to L **do**
17: Use $R_i(x_i^{[j]}, a_i^{[j]} | x_i^{[j-1]}, a_i^{[j-1]})$ to sample $(x_i^{[j]}, a_i^{[j]})$ from $\lambda_i(x, a)$
18: **end for**
19: **end for**
20: $l'_n \leftarrow \text{Unif}\{1, \ldots, L\}$
21: $x'_n \leftarrow x_n^{[l'_n]}$
22: **for** $i = n - 1$ to 1 **do**
23: Sample l'_i with probabilities $P(l'_i = k) \propto p(x'_{i+1} | x_i^{[l'_i]})$ with $k \in \{1, \ldots, L\}$
24: $x'_i \leftarrow x_i^{[l'_i]}$
25: **end for**

methods in detail. They show how the embedded HMM sampler introduced in this section can be viewed as sampling from an extended target distribution that includes the auxiliary variables a. This provides an alternative way of establishing the validity of the embedded HMM sampler introduced here, which is analogous to how the validity of particle MCMC algorithms is established.

5 Proof of correctness

We will prove the correctness of the embedded HMM sampler described in Section 4 — that is, that it leaves $p(\mathbf{x}|\mathbf{y})$ invariant. Whether the transition produces an ergodic chain will depend on the model, the selection of R, and whether any other transitions are also being used. Many models and transitions used in practice have non-zero density everywhere in the state space, and then ergodicity will not be a problem, in theory at least. In particular, as long as at each time i we pick a transition R_i that can sample

any value of x_i that has non-zero probability density under λ_i, we will be able to create sets of pool states that contain any $\mathbf{x}$ that has non-zero probability density given $\mathbf{y}$.

We assume that we use the forward pool state selection scheme as described in Section 4.1 and the corresponding auxiliary variable construction as described in Section 4.2. A similar proof can be done for the backward pool state selection scheme. This sampler targets the distribution of state sequences for a non-linear, non-Gaussian state space model as described in Section 1.

Our proof is based on the original proof of Neal (2003), modifying it to not assume that the sets of pool states $\mathcal{P}_1, \ldots, \mathcal{P}_n$ are selected independently at each time. Another change in the proof is to account for generating the pool states by sampling them from λ_i instead of κ_i^f for $i > 2$.

We show that the probability of starting at $\mathbf{x}$ and moving to $\mathbf{x}'$ with given sets of pool states $\mathcal{P}_i$ (consisting of values of x and a at each time i), pool indices l_i of x_i, and pool indices l'_i of x'_i is the same as the probability of starting at $\mathbf{x}'$ and moving to $\mathbf{x}$ with the same set of pool states $\mathcal{P}_i$, pool indices l'_i of x'_i, and pool indices l_i of x_i. This in turn implies, by summing/integrating over $\mathcal{P}_i$ and l_i, that the embedded HMM method with the sequential pool state scheme satisfies the detailed balance condition with respect to $p(\mathbf{x}|\mathbf{y})$, which is that

$$p(\mathbf{x}|\mathbf{y})Q(\mathbf{x}'|\mathbf{x}) = p(\mathbf{x}'|\mathbf{y})Q(\mathbf{x}|\mathbf{x}') \tag{17}$$

with Q the transition of the embedded HMM sampler. As is well-known, this implies that Q leaves $p(\mathbf{x}|\mathbf{y})$ invariant.

The probability of starting at $\mathbf{x}$ and moving to $\mathbf{x}'$ decomposes into the product of the probability of starting at $\mathbf{x}$, which is $p(\mathbf{x}|\mathbf{y})$, the probability of choosing a set of pool state indices l_i, which is $\frac{1}{L^n}$, the probability, given the current sequence, of selecting the sets of pool states $\mathcal{P}_i$, $P(\mathcal{P}_1, \ldots, \mathcal{P}_n|\mathbf{x})$, and finally the probability of choosing $\mathbf{x}'$.

The probability of choosing given sets of pool states is

$$P(\mathcal{P}_1, \ldots, \mathcal{P}_n|\mathbf{x}) = P(\mathcal{P}_1|x_1) \prod_{i=2}^{n} P(\mathcal{P}_i|\mathcal{P}_{i-1}, x_i). \tag{18}$$

At time 1, we use a Markov chain with invariant density κ_1^f to select pool states in $\mathcal{P}_1$ with the chain started at $x_1^{[l_1]} = x_1$. Therefore

$$\begin{aligned}
P(\mathcal{P}_1|x_1) &= \prod_{j=l_1+1}^{L} R_1(x_1^{[j]}|x_1^{[j-1]}) \prod_{j=l_1-1}^{1} \tilde{R}_1(x_1^{[j]}|x_1^{[j+1]}) \\
&= \prod_{j=l_1+1}^{L} R_1(x_1^{[j]}|x_1^{[j-1]}) \prod_{j=l_1-1}^{1} R_1(x_1^{[j+1]}|x_1^{[j]}) \frac{\kappa_1^f(x_1^{[j]})}{\kappa_1^f(x_1^{[j+1]})} \\
&= \prod_{j=l_1}^{L-1} R_1(x_1^{[j+1]}|x_1^{[j]}) \prod_{j=l_1-1}^{1} R_1(x_1^{[j+1]}|x_1^{[j]}) \frac{\kappa_1^f(x_1^{[j]})}{\kappa_1^f(x_1^{[j+1]})}
\end{aligned}$$

$$= \frac{\kappa_1^f(x_1^{[1]})}{\kappa_1^f(x_1)} \prod_{j=1}^{L-1} R_1(x_1^{[j+1]}|x_1^{[j]}). \tag{19}$$

At times $i > 1$, we first stochastically select an initial value of a_i and then use a Markov chain with invariant density λ_i to sample the pool states, with the chain started at $x_i^{[l_i]} = x_i$ and $a_i^{[l_i]} = a_i$. Therefore

$$\begin{aligned}
P(\mathcal{P}_i|\mathcal{P}_{i-1}, x_i) &= \frac{p(x_i|x_{i-1}^{[a_i]})}{\sum_{m=1}^{L} p(x_i|x_{i-1}^{[m]})} \prod_{j=l_i+1}^{L} R_i(x_i^{[j]}, a_i^{[j]}|x_i^{[j-1]}, a_i^{[j-1]}) \\
&\quad \times \prod_{j=l_i-1}^{1} \tilde{R}_i(x_i^{[j]}, a_i^{[j]}|x_i^{[j+1]}, a_i^{[j+1]}) \\
&= \frac{p(x_i|x_{i-1}^{[a_i]})}{\sum_{m=1}^{L} p(x_i|x_{i-1}^{[m]})} \prod_{j=l_i+1}^{L} R_i(x_i^{[j]}, a_i^{[j]}|x_i^{[j-1]}, a_i^{[j-1]}) \\
&\quad \times \prod_{j=l_i-1}^{1} R_i(x_i^{[j+1]}, a_i^{[j+1]}|x_i^{[j]}, a_i^{[j]}) \frac{\lambda_i(x_i^{[j]}, a_i^{[j]})}{\lambda_i(x_i^{[j+1]}, a_i^{[j+1]})} \\
&= \frac{p(x_i|x_{i-1}^{[a_i]})}{\sum_{m=1}^{L} p(x_i|x_{i-1}^{[m]})} \prod_{j=l_i}^{L-1} R_i(x_i^{[j+1]}, a_i^{[j+1]}|x_i^{[j]}, a_i^{[j]}) \\
&\quad \times \prod_{j=l_i-1}^{1} R_i(x_i^{[j+1]}, a_i^{[j+1]}|x_i^{[j]}, a_i^{[j]}) \frac{\lambda_i(x_i^{[j]}, a_i^{[j]})}{\lambda_i(x_i^{[j+1]}, a_i^{[j+1]})} \\
&= \frac{p(x_i|x_{i-1}^{[a_i]})}{\sum_{m=1}^{L} p(x_i|x_{i-1}^{[m]})} \frac{\lambda_i(x_i^{[1]}, a_i^{[1]})}{\lambda_i(x_i, a_i)} \\
&\quad \times \prod_{j=1}^{L-1} R_i(x_i^{[j+1]}, a_i^{[j+1]}|x_i^{[j]}, a_i^{[j]}).
\end{aligned} \tag{20}$$

Finally, we choose a new sequence $\mathbf{x}'$ amongst the collection of sequences consisting of the pool states with a backward pass. This is done by first choosing l'_n uniformly at random and then setting $x'_n = x_n^{[l'_n]}$. We select the remaining $x_i^{[l'_i]}$ by selecting $l'_1, \ldots, l'_{n-1}$ with probability

$$\prod_{i=2}^{n} \frac{p(x'_i|x_{i-1}^{[l'_{i-1}]})}{\sum_{m=1}^{L} p(x'_i|x_{i-1}^{[m]})}. \tag{21}$$

Thus, the probability of starting at $\mathbf{x}$ and going to $\mathbf{x}'$, with given $\mathcal{P}_1, \ldots, \mathcal{P}_n, l_1, \ldots, l_n$ and $l'_1, \ldots, l'_n$ is

$$p(\mathbf{x}|\mathbf{y}) \times \frac{1}{L^n} \times \frac{\kappa_1^f(x_1^{[1]})}{\kappa_1^f(x_1)} \prod_{j=1}^{L-1} R_1(x_1^{[j+1]}|x_1^{[j]})$$

$$\times \prod_{i=2}^{n}\left[\frac{p(x_i|x_{i-1}^{[a_i]})}{\sum_{m=1}^{L} p(x_i|x_{i-1}^{[m]})}\frac{\lambda_i(x_i^{[1]},a_i^{[1]})}{\lambda_i(x_i,a_i)}\prod_{j=1}^{L-1} R_i(x_i^{[j+1]},a_i^{[j+1]}|x_i^{[j]},a_i^{[j]})\right]$$
$$\times \frac{1}{L}\times\prod_{i=2}^{n}\frac{p(x_i'|x_{i-1}^{[l'_{i-1}]})}{\sum_{m=1}^{L} p(x_i'|x_{i-1}^{[m]})}. \tag{22}$$

This expression can be rewritten as

$$\kappa_1^f(x_1^{[1]})\prod_{j=1}^{L-1} R_1(x_1^{[j+1]}|x_1^{[j]})\times\prod_{i=2}^{n}\left[\lambda_i(x_i^{[1]},a_i^{[1]})\prod_{j=1}^{L-1} R_i(x_i^{[j+1]},a_i^{[j+1]}|x_i^{[j]},a_i^{[j]})\right]$$
$$\times\frac{p(\mathbf{x}|\mathbf{y})}{\kappa_1^f(x_1)\prod_{i=2}^{n}\lambda_i(x_i,a_i)}\times\prod_{i=2}^{n}\frac{p(x_i|x_{i-1}^{[a_i]})}{\sum_{m=1}^{L} p(x_i|x_{i-1}^{[m]})}\prod_{i=2}^{n}\frac{p(x_i'|x_{i-1}^{[l'_{i-1}]})}{\sum_{m=1}^{L} p(x_i'|x_{i-1}^{[m]})}\times\frac{1}{L^{n+1}}. \tag{23}$$

Note that $x_{i-1}^{[l'_{i-1}]}=x'_{i-1}$. Also $\kappa_1^f(x)=p(x)p(y_1|x)/\int p(x)p(y_1|x)dx$ and

$$\prod_{i=2}^{n}\lambda_i(x,a)=\prod_{i=2}^{n}\frac{p(y_i|x)p(x|x_{i-1}^{[a]})}{\int\sum_{m=1}^{L} p(y_i|x)p(x|x_{i-1}^{[m]})dx}. \tag{24}$$

Also,

$$p(\mathbf{x}|\mathbf{y})=\frac{p(x_1)\prod_{i=2}^{n}p(x_i|x_{i-1})\prod_{i=1}^{n}p(y_i|x_i)}{p(\mathbf{y})}. \tag{25}$$

Therefore (23) can be rewritten as

$$\frac{1}{p(\mathbf{y})}\kappa_1^f(x_1^{[1]})\prod_{j=1}^{L-1} R_1(x_1^{[j+1]}|x_1^{[j]})\times\prod_{i=2}^{n}\left[\lambda_i(x_i^{[1]},a_i^{[1]})\prod_{j=1}^{L-1} R_i(x_i^{[j+1]},a_i^{[j+1]}|x_i^{[j]},a_i^{[j]})\right]$$
$$\times\prod_{i=2}^{n}p(x_i|x_{i-1})\times\prod_{i=2}^{n}p(x_i'|x_{i-1}')\times\prod_{i=2}^{n}\frac{1}{\sum_{m=1}^{L} p(x_i|x_{i-1}^{[m]})}\times\prod_{i=2}^{n}\frac{1}{\sum_{m=1}^{L} p(x_i'|x_{i-1}^{[m]})}$$
$$\times\int p(x)p(y_1|x)dx\prod_{i=2}^{n}\int\sum_{m=1}^{L} p(y_i|x)p(x|x_{i-1}^{[m]})dx\times\frac{1}{L^{n+1}}. \tag{26}$$

The last factor in the product only depends on the selected sets of pool states and on the observations. By exchanging $\mathbf{x}$ and $\mathbf{x}'$ we see that the probability of starting at $\mathbf{x}'$ and then going to $\mathbf{x}$, with given sets of pool states $\mathcal{P}_i$, pool indices l_i of x_i and pool indices l_i' of x_i' is the same.

Finally, by integrating (26) over $\mathcal{P}_i$ and summing over l_i we conclude that our new embedded HMM method satisfies detailed balance with respect to $p(\mathbf{x}|\mathbf{y})$.

6 Experiments

6.1 Test models

To demonstrate the performance of our new pool state scheme, we use two different state space models. The latent process for both models is a vector autoregressive process, with

$$X_1 \sim N(0, \Sigma_{\text{init}}), \tag{27}$$

$$X_i | x_{i-1} \sim N(\Phi x_{i-1}, \Sigma), \quad i = 2, \ldots, n, \tag{28}$$

where $X_i = (X_{i,1}, \ldots, X_{i,P})'$ and

$$\Phi = \begin{pmatrix} \phi_1 & 0 & \ldots & 0 \\ 0 & \phi_2 & \ddots & \vdots \\ \vdots & \ddots & \phi_{P-1} & 0 \\ 0 & \ldots & 0 & \phi_P \end{pmatrix}, \tag{29}$$

$$\Sigma = \begin{pmatrix} 1 & \rho & \ldots & \rho \\ \rho & 1 & \ddots & \vdots \\ \vdots & \ddots & 1 & \rho \\ \rho & \ldots & \rho & 1 \end{pmatrix}, \tag{30}$$

$$\Sigma_{\text{init}} = \begin{pmatrix} \frac{1}{1-\phi_1^2} & \frac{\rho}{\sqrt{1-\phi_1^2}\sqrt{1-\phi_2^2}} & \cdots & \frac{\rho}{\sqrt{1-\phi_1^2}\sqrt{1-\phi_P^2}} \\ \frac{\rho}{\sqrt{1-\phi_2^2}\sqrt{1-\phi_1^2}} & \frac{1}{1-\phi_2^2} & \ddots & \vdots \\ \vdots & \ddots & \frac{1}{1-\phi_{P-1}^2} & \frac{\rho}{\sqrt{1-\phi_{P-1}^2}\sqrt{1-\phi_P^2}} \\ \frac{\rho}{\sqrt{1-\phi_P^2}\sqrt{1-\phi_1^2}} & \cdots & \frac{\rho}{\sqrt{1-\phi_P^2}\sqrt{1-\phi_{P-1}^2}} & \frac{1}{1-\phi_P^2} \end{pmatrix}. \tag{31}$$

Note that Σ_{init} is the covariance of the stationary distribution for this process.

For model 1, the observations are given by

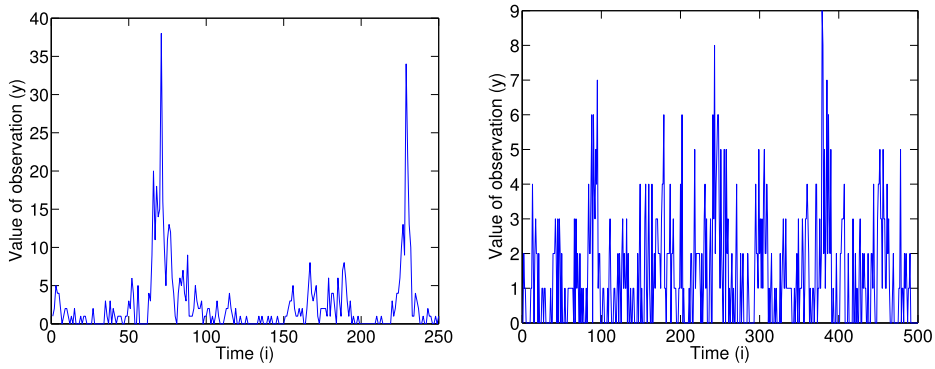
$$Y_{i,j} | x_{i,j} \sim \text{Poisson}(\exp(c_j + \sigma_j x_{i,j})), \quad i = 1, \ldots, n, \quad j = 1, \ldots, P. \tag{32}$$

For model 2, the observations are given by

$$Y_{i,j} | x_{i,j} \sim \text{Poisson}(\sigma_j |x_{i,j}|), \quad i = 1, \ldots, n, \quad j = 1, \ldots, P. \tag{33}$$

For model 1, we use a 10-dimensional latent state and a sequence length of $n = 250$, setting parameter values to $\rho = 0.7$, and $c_j = -0.4$, $\phi_j = 0.9$, $\sigma_j = 0.6$ for $j = 1, \ldots, P$, with $P = 10$.

For model 2, we increase the dimensionality of the latent space to 15 and the sequence length to 500. We set $\rho = 0.7$ and $\phi_j = 0.9$, $\sigma_j = 0.8$ for $j = 1, \ldots, P$, with $P = 15$.



(a) Model 1 (b) Model 2

Figure 1: Observations from model 1 and model 2 along dimension $j = 1$.

We generated one random sequence from each model to test our samplers on. These observations from model 1 and model 2 are shown in Figure 1. Note that we are testing only sampling of the latent variables, with the parameters set to their true values. All samplers were implemented in MATLAB on a Linux system with a 2.60 GHz Intel i7-3720QM CPU. The code for all experiments in this paper is available at `http://arxiv.org/abs/1602.06030`.

6.2 Autoregressive updates

For sampling pool states in our embedded HMM MCMC schemes, as well as for comparison MCMC schemes, we will make use of Neal's (1998) "autoregressive" Metropolis-Hastings update, which we review here. This update is designed to draw samples from a distribution of the form $p(w)p(y|w)$ where $p(w)$ is multivariate Gaussian with mean μ and covariance Σ and $p(y|w)$ is typically a density for some observed data. These proposals are known to work better than ordinary random walk proposals for distributions of this form (Neal, 1998). They have also been further developed into the elliptical slice sampler by Murray et al. (2010).

This autoregressive update proceeds as follows. Let M be the lower triangular Cholesky decomposition of Σ, so $\Sigma = MM^T$, and z be a vector of i.i.d. normals with mean zero and variance one. Let $\epsilon \in [-1, 1]$ be a tuning parameter that determines the scale of the proposal. Starting at w, we propose

$$w' = \mu + \sqrt{1 - \epsilon^2}(w - \mu) + \epsilon M z. \tag{34}$$

Because these autoregressive proposals are reversible with respect to $p(w)$, the proposal density and $p(w)$ cancel in the Metropolis-Hastings acceptance ratio. This update is therefore accepted with probability

$$\min\left(1, \frac{p(y|w')}{p(y|w)}\right). \tag{35}$$

Note that for this update, the same value of ϵ is used for scaling along every dimension. It would be of independent interest to develop a version of this update where ϵ can be different for each dimension of w.

6.3 Single-state Metropolis sampler

A simple scheme for sampling the latent state sequence is to use Metropolis-Hastings updates that sample each x_i in sequence, conditional on $x_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ and the data, starting at time 1 and going to time n. We sample all dimensions of x_i at once using autoregressive updates (see Section 6.2). In both of our test models, the posterior standard deviation of the latent variables $x_{i,j}$ varies depending on the value of the observed $y_{i,j}$. To address this, we alternately use a larger or a smaller proposal scaling, ϵ, for the autoregressive updates when performing a scan with the Metropolis sampler, using the same ϵ for all times. For details on the Metropolis scheme, see the Appendix provided in the online supplemental materials (Shestopaloff and Neal, 2017).

6.4 PGBS combined with Metropolis

We implement the PGBS method as described in Section 3, using the initial density $p(x)$ and the transition densities $p(x_i|x_{i-1})$ as importance densities to generate particles. We combine PGBS updates with single-state Metropolis updates from Section 6.2. This way, we combine the strengths of the two samplers in targeting different parts of the posterior distribution. In particular, we expect the Metropolis updates to do better for the x_i with highly informative y_i, and the PGBS updates to do better for the x_i where y_i is not as informative.

6.5 Flip updates

Generating pool states locally can be helpful when applying embedded HMMs to models with high-dimensional state spaces but it also makes sampling difficult if the posterior is multimodal. Consider the case of model 2 when the observation probability depends on $|x_i|$ instead of x_i, so that many modes with different signs for some x_i exist. We propose to handle this problem by adding an additional specialized flip update that creates a "mirror" set of pool states, in which $-x_i$ will be in the pool if x_i is. By having a mirror set of pool states, we are able to flip large segments of the sequence in a single update, allowing efficient exploration of different posterior modes.

To generate a mirror set of pool states, we must correctly use the flip updates when sampling the pool states. Since we want each pool state to have a negative counterpart, we choose the number of pool states L to be even. The chain used to sample pool states then alternates two types of updates, a usual update to generate a pool state and a flip update to generate its negated version. The usual update can be a combination of any updates, such as those we consider above. So that each state will have a flipped version, we start with a flip transition between $x^{[1]}$ and $x^{[2]}$, a usual transition between $x^{[2]}$ and $x^{[3]}$, and so on up to a flip transition between $x^{[L-1]}$ to $x^{[L]}$.

At time 1, we start with the current state x_1 and randomly assign it to some index l_1 in the chain used to generate pool states. Then, starting at x_1 we generate pool states by reversing the Markov chain transitions back to 1 and going forward up to L. Each flip update is then a Metropolis update proposing to generate a pool state $-x_1$ given that the chain is at some pool state x_1. Note that if the observation probability depends on x_1 only through $|x_1|$ and $p(x)$ is symmetric around zero then this update is always accepted.

At time $i > 1$, a flip update proposes to update a pool state (x, a) to $(-x, a')$ such that $x_{i-1}^{[a']} = -x_{i-1}^{[a]}$. Here, since the pool states at each time are generated by alternating flip and usual updates, starting with a flip update to $x_i^{[1]}$, the proposal to move from a to a' can be viewed as follows. Suppose that instead of labelling our pool states from 1 to L we instead label them 0 to $L-1$. The pool states at times 0 and 1, then 2 and 3, and so on will then be flipped pairs, and the proposal to change a to a' can be seen as proposing to flip the lower order bit in a binary representation of a'. For example, a proposal to move from $a = 3$ to $a = 2$ can be seen as proposing to change a from 11 to 10 (in binary). Such a proposal will always be accepted assuming a transition density for which $p(x_i|x_{i-1}) = p(-x_i|-x_{i-1})$ and an observation probability which depends on x_i only via $|x_i|$.

6.6 Setup for baseline sampling

For model 1, we implemented the simple single-state Metropolis sampler, the PGBS sampler, and the combination of PGBS with Metropolis. For model 2, we implemented the simple single-state Metropolis sampler and the PGBS with Metropolis sampler. For both models and all samplers, we ran the sampler five times using five different random number generator seeds.

Model 1

For the single-state Metropolis sampler, we initialized all $x_{i,j}$ to 0. Every iteration alternately used a scaling factor, ϵ, of either 0.2 or 0.8, which resulted in an average acceptance rate of between 30% and 90% for the different x_i over the sampler run. We ran the sampler for $1,000,000$ iterations, and prior to analysis, the resulting sample was thinned by a factor of 10, to $100,000$. The thinning was done due to the difficulty of working with all samples at once, and after thinning the samples still possessed autocorrelation times significantly greater than 1. Each of the $100,000$ samples took about 0.17 seconds to draw.

For the PGBS sampler and the sampler combining PGBS and Metropolis updates, we also initialized all $x_{i,j}$ to 0. We used 250 particles for the PGBS updates. For the Metropolis updates, we alternated between scaling factors of 0.2 and 0.8, which also gave acceptance rates between 30% and 90%. For the standalone PGBS sampler, we performed a total of $70,000$ iterations. Each iteration produced two samples for a total of $140,000$ samples and consisted of a PGBS update using the forward sequence and

a PGBS update using the reversed sequence. Each sample took about 0.12 seconds to draw.

For the PGBS with Metropolis sampler, we performed a total of 30,000 iterations of the sampler. Each iteration was used to produce four samples, for a total of 120,000 samples, and consisted of a PGBS update using the forward sequence, ten Metropolis updates (of which only the value after the tenth update was retained), a PGBS update using the reversed sequence, and another ten Metropolis updates, again only keeping the value after the tenth update. The average time to draw each of the 120,000 samples was about 0.14 seconds.

Model 2

For model 2, we were unable to make the single-state Metropolis sampler converge to anything resembling the actual posterior in a reasonable amount of time. In particular, we found that for $x_{i,j}$ sufficiently far from 0, the Metropolis sampler tended to be stuck in a single mode, never visiting values with the opposite sign.

For the PGBS with Metropolis sampler, we set the initial values of $x_{i,j}$ to 1. We set the number of particles for PGBS to 80,000, which was nearly the maximum possible for the memory capacity of the computer we used. For the Metropolis sampler, we alternated between scaling factors of 0.3 and 1, which resulted in acceptance rates ranging between 29% and 72%. We performed a total of 250 iterations of the sampler. As for model 1, each iteration produced four samples, for a total of 1,000 samples, and consisted of a PGBS update with the forward sequence, 50 Metropolis updates (of which we only keep the value after the last one), a PGBS update using the reversed sequence, and another 50 Metropolis updates (again only keeping the last value). It took about 26 seconds to draw each sample.

6.7 Setup for embedded HMM sampling

For both model 1 and model 2, we implemented the proposed embedded HMM method using the forward pool state selection scheme, alternating between updates that use the original and the reversed sequence. Like the baseline samplers, we ran the embedded HMM samplers five times for both models, using five different random number generator seeds.

We generate pool states at time 1 using autoregressive updates to sample from κ_1^f. At times $i \geq 2$, we sample each pool state from $\lambda_i(x, a)$ by combining an autoregressive and shift update. The autoregressive update proposes to only change x, keeping the current a fixed. The shift update samples both x and a, with a new a proposed from a Uniform$\{1, \ldots, L\}$ distribution. For model 2, we also add a flip update to generate a negated version of each pool state.

Since the shift and flip updates are also Metropolis updates, using them together with Metropolis or autoregressive updates, for each of x and a separately, allows embedded HMM updates to be performed in time proportional to nL.

Note that the chain used to produce the pool states now uses a sequence of updates. Therefore, if our forward transition first does an autoregressive update and then a shift update, the reverse transitions must first do a shift update and then an autoregressive update.

In the process of tuning the embedded HMM sampler, we found that is beneficial to choose at random the proposal scaling, ϵ, for each single-state Metropolis update within the Markov chain used to generate pool states at time i. This allows generation of sets of pool states which are more concentrated when y_i is informative and more dispersed when y_i holds little information. In our example data sets, there is substantial difference in how informative the different y_i are, and as a result this technique helps improve performance over using a fixed proposal scaling.

Model 1

For model 1, we initialized all $x_{i,j}$ to 0. We used 50 pool states for the embedded HMM updates. For each Metropolis update to sample a pool state, we used a different scaling ϵ, chosen at random from a Uniform$(0.1, 0.4)$ distribution. The acceptance rates ranged between 55% and 95% for the Metropolis updates and between 20% and 70% for the shift updates. We performed a total of $9,000$ iterations of the sampler, with each iteration consisting of an embedded HMM update using the forward sequence and an embedded HMM update using the reversed sequence, for a total of $18,000$ samples. Each sample took about 0.81 seconds to draw.

Model 2

For model 2, we initialized the $x_{i,j}$ to 1. We used a total of 80 pool states for the embedded HMM sampler (i.e. 40 positive-negative pairs due to flip updates). Each Metropolis update used to sample a pool state used a scaling, ϵ, randomly drawn from the Uniform$(0.05, 0.2)$ distribution. The acceptance rates ranged between 75% and 90% for the Metropolis updates and between 20% and 40% for the shift updates. We performed a total of $9,000$ iterations of the sampler, producing two samples per iteration with an embedded HMM update using the forward sequence and an embedded HMM update using the reversed sequence. Each of the $18,000$ samples took about 1.4 seconds to draw.

6.8 Comparisons

As a way of comparing the performance of the two methods, we use an estimate of autocorrelation time[1] for each of the latent variables $x_{i,j}$. Autocorrelation time is a measure of how many draws need to be made using the sampling chain to produce

[1]Technically, when we alternate updates with the forward and reversed sequence or mix PGBS and single-state Metropolis updates, we cannot use autocorrelation times to measure how well the chain explores the space. While the sampling scheme leaves the correct target distribution invariant, the flipping of the sequence makes the sampling chain for a given variable non-homogeneous. However, suppose that instead of deterministically flipping the sequence at every step, we add an auxiliary

the equivalent of one independent sample. The autocorrelation time is defined as $\tau = 1 + 2\sum_{i=1}^{\infty} \rho_k$, where ρ_k is the autocorrelation at lag k. It is commonly estimated as

$$\hat{\tau} = 1 + 2\sum_{i=1}^{K} \hat{\rho}_k, \tag{36}$$

where $\hat{\rho}_k$ are estimates of lag-k autocorrelations and the cutoff point K is chosen so that $\hat{\rho}_k$ is negligibly different from 0 for $k > K$. Here

$$\hat{\rho}_k = \frac{\hat{\gamma}_k}{\hat{\gamma}_0}, \tag{37}$$

where $\hat{\gamma}_k$ is an estimate of the lag-k autocovariance

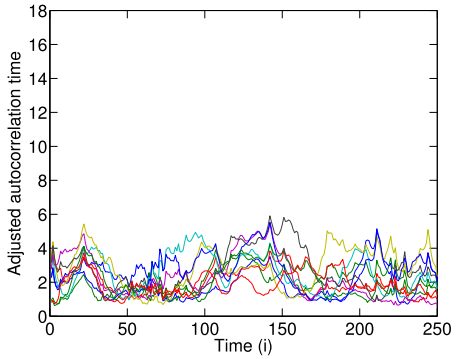
$$\hat{\gamma}_k = \frac{1}{n}\sum_{l=1}^{n-k}(x_l - \bar{x})(x_{k+l} - \bar{x}). \tag{38}$$

When estimating autocorrelation time, we remove the first 10% of the sample as burn-in. Then, to estimate $\hat{\gamma}_k$, we first estimate autocovariances for each of the five runs, taking $\bar{x}$ to be the overall mean over the five runs. We then average these five autocovariance estimates to produce $\hat{\gamma}_k$. To speed up autocovariance computations, we use the Fast Fourier Transform. The autocorrelation estimates are then adjusted for computation time, by multiplying the estimated autocorrelation time by the time it takes to draw a sample, to ensure that the samplers are compared fairly.

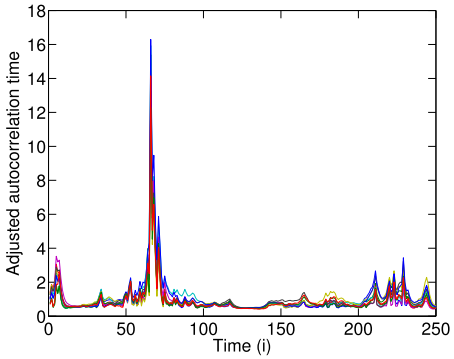
The computation time-adjusted autocorrelation estimates for model 1, for all the latent variables, plotted over time, are presented in Figure 2. We found that the combination of single-state Metropolis and PGBS works best for the unimodal model. The other samplers work reasonably well too. We note that the spike in autocorrelation time for the PGBS and to a lesser extent for the PGBS with Metropolis sampler occurs at the point where the data is very informative. This in turn makes PGBS sampling inefficient, due to the use of a diffuse transition distribution. As a result, much of the sampling in that region is due to the Metropolis updates. Here, we also note that the computation time adjustment is sensitive to the particularities of the implementation, in this case done in MATLAB, where performance is strongly influenced by how well vectorization can be exploited. Implementing the samplers in a different language might change the relative comparisons.

We now look at how the samplers perform on the more challenging model 2. We first did a preliminary check of whether the samplers do indeed explore the different modes of the distribution by looking at variables far apart in the sequence, where we expect to see four modes (with all possible combinations of signs). This is indeed the case for both the PGBS with Metropolis and Embedded HMM samplers. Next, we look at how efficiently the latent variables are sampled. Of particular interest are the latent

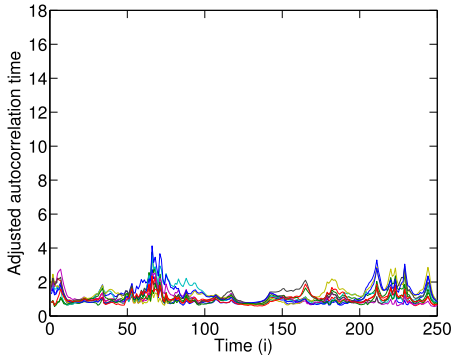
indicator variable that determines (given the current state) whether the forward or the reversed sequence is used, and that the probability of flipping this indicator variable is nearly one. With this auxiliary variable the sampling chain becomes homogeneous, with its behaviour nearly identical to that of our proposed scheme. Using autocorrelation time estimates to evaluate the performance of our sampler is therefore valid, for all practical purposes.



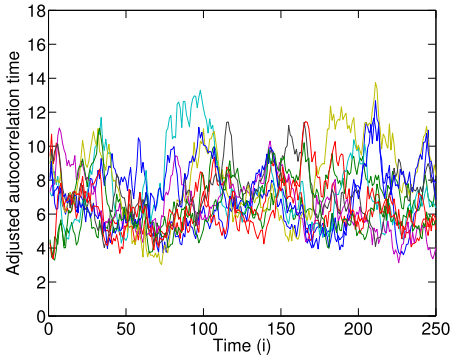
(a) Metropolis (0.17 seconds/sample)



(b) PGBS (0.12 seconds/sample)



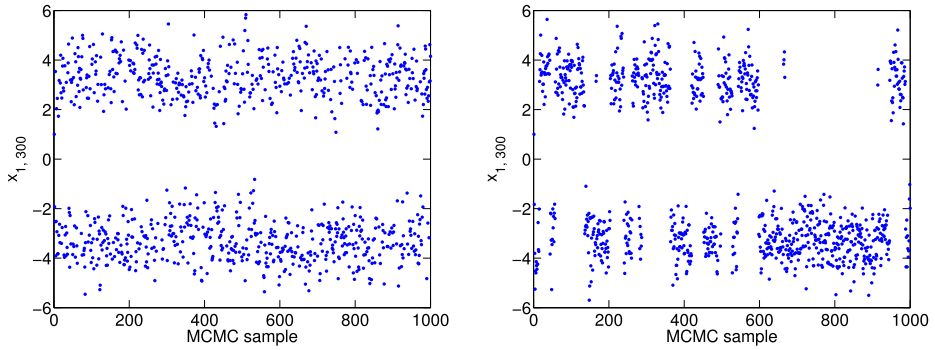
(c) PGBS+Metropolis (0.14 seconds/sample)



(d) Embedded HMM (0.81 seconds/sample)

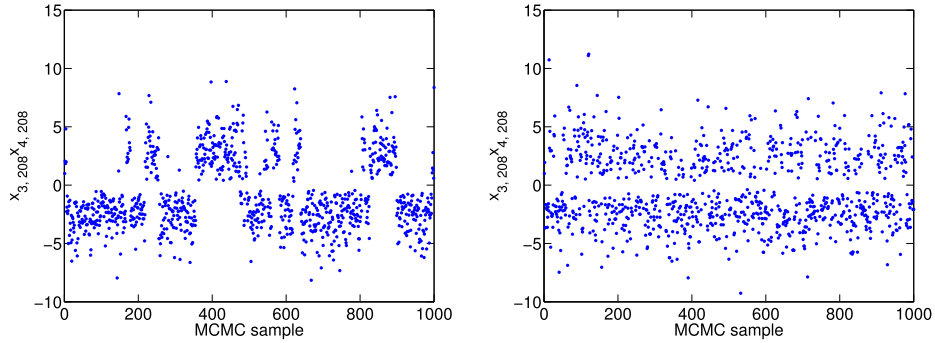
Figure 2: Estimated autocorrelation times for each latent variable for model 1, adjusted for computation time. The colour of the lines corresponds to different components of the latent state and the x-axis corresponds to different times.

variables with well-separated modes, since sampling performance for such variables is illustrative of how well the samplers explore the different posterior modes. Consider the variable $x_{1,300}$, which has true value -1.99. Figure 3 shows how the different samplers explore the two modes for this variable, with equal computation times used to produced the samples for the trace plots. We can see that the embedded HMM sampler with flip updates performs significantly better for sampling a variable with well-separated modes. Experiments showed that the performance of the embedded HMM sampler on model 2 without flip updates is much worse. We can also look at the product of the two variables $x_{3,208}x_{4,208}$, with true value -4.45. The trace plot is given in Figure 4. In this case, we can see that the PGBS with Metropolis sampler performs better. Since the flip updates change the signs of all dimensions of x_i at once, we do not expect them to be as useful for improving sampling of this function of state. The vastly greater number of particles used by PGBS, $80,000$, versus 80 for the embedded HMM method, works to the advantage



(a) Embedded HMM (b) PGBS with Metropolis

Figure 3: Comparison of samplers for model 2. Trace plots of $x_{1,300}$ drawn using the embedded HMM and Particle Gibbs with Metropolis samplers.

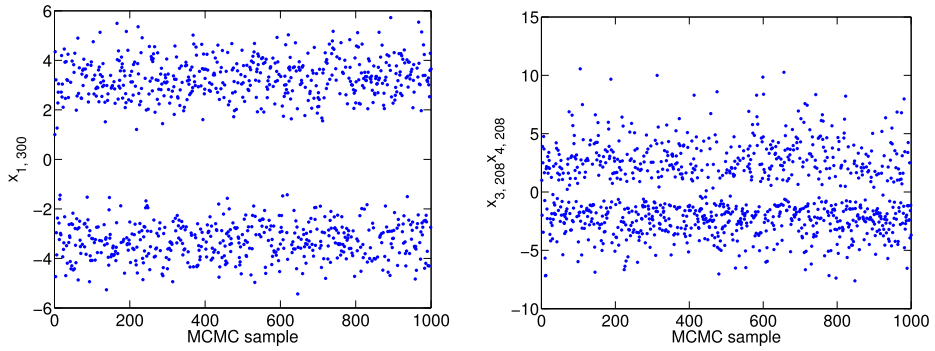


(a) Embedded HMM (b) PGBS with Metropolis

Figure 4: Comparison of samplers for model 2. Trace plots of $x_{3,208}x_{4,208}$ drawn using the embedded HMM and Particle Gibbs with Metropolis samplers.

of PGBS, and explains the performance difference. Looking at these results, we might expect that we can get a good sampler for both $x_{1,300}$ and $x_{3,208}x_{4,208}$ by alternating embedded HMM and PGBS with Metropolis updates. This is indeed the case, which can be seen in Figure 5. For producing these plots, we used an embedded HMM sampler with the same settings as in the experiment for model 2 and a PGBS with Metropolis sampler with $10,000$ particles and Metropolis updates using the same settings as in the experiment for Model 2.

This example of model 2 demonstrates another advantage of the embedded HMM viewpoint, which is that it allows us to design updates for sampling pool states to handle certain properties of the density. This is arguably easier than designing importance densities in high dimensions.



(a) Trace plot for $x_{1,300}$

(b) Trace plot for $x_{3,208}x_{4,208}$

Figure 5: Comparison of samplers for model 2. Combining the embedded HMM and PGBS with Metropolis samplers.

7 Conclusion

We have demonstrated that it is possible to use embedded HMM's to efficiently sample state sequences in models with higher dimensional state spaces. The novel methods introduced in this work are most useful for high-dimensional models with strong temporal dependencies in the state sequence and where the data can be very informative. For simpler models with low-dimensional state spaces or where the data is less informative, methods which select pool states independently may work better. We have also shown how embedded HMMs can improve sampling efficiency in an example model with a multimodal posterior, by introducing a new pool state selection scheme. There are several directions in which this research can be further developed.

The most obvious extension is to treat the model parameters as unknown and add a step to sample parameters given a value of the latent state sequence as in Andrieu et al. (2010). In the unknown parameter context, it would also be interesting to see how the proposed sequential pool state selection schemes can be used together with ensemble MCMC updates of Shestopaloff and Neal (2013). For example, one approach is to have the pool state distribution depend on the average of the current and proposed parameter values in an ensemble Metropolis update, as in Shestopaloff and Neal (2014).

One might also wonder whether it is possible to use the latent state at all times in constructing the pool state density at a given time. It is not obvious how to do this. For example, for the forward scheme, using the current value of the state sequence at some time $k > i$ to construct pool states at time i means that the pool states at time k will end up depending on the current value of x_k, which would lead to an invalid sampler.

A similar point is that at each time $i < n$, the pool state generation procedure does not depend on the observed data after time i, which may cause some difficulties in scaling this method further. On one hand, this allows for greater dispersion in the pool states than if we were to impose a constraint from the other direction as with the single-

state Metropolis method, potentially allowing us to make larger moves. On the other hand, not having this constraint as in the current algorithm also means that the pool states can become too dispersed. In higher dimensions, one way in which this can be controlled is by using a Markov chain that samples pool states close to the current x_i — that is, a Markov chain that is deliberately slowed down in order not to overdisperse the pool states, which could lead to a collection of sequences with low posterior density.

Supplementary Material

Supplementary Material for "Sampling latent states for high-dimensional non-linear state space models with the embedded HMM method" (DOI: 10.1214/17-BA1077SUPP; .pdf).

Computing code for "Sampling latent states for high-dimensional non-linear state space models with the embedded HMM method" (DOI: 10.1214/17-BA1077SUPPB; .zip).

References

Andrieu, C., Doucet, A. and Holenstein, R. (2010) "Particle Markov chain Monte Carlo methods", *Journal of the Royal Statistical Society B*, vol. 72, pp. 269–342. 798, 801, 820

Doucet, A., de Freitas, N. and Gordon, N. (2001) *Sequential Monte Carlo Methods in Practice*, Springer: New York. MR1847784. doi: https://doi.org/10.1007/978-1-4757-3437-9_1. 798

Finke, A., Doucet, A. and Johansen, A. M. (2016) "On embedded hidden Markov models and particle Markov chain Monte Carlo methods", Technical Report, http://arxiv.org/abs/1610.08962. 806

Lindsten, F., Schön, T. B. (2012) "On the use of backward simulation in the particle Gibbs sampler", in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pp. 3845–3848. 798, 801

Murray, I., Adams, R. P. and MacKay, D. J. C. (2010) "Elliptical slice sampling", in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), JMLR W&CP*, vol. 9, pp. 541–548. 812

Neal, R. M. (1998) "Regression and classification using Gaussian process priors", in J.M. Bernardo et al (editors) *Bayesian Statistics 6*, Oxford University Press, pp. 475–501. MR1723510. 812

Neal, R. M. (2003) "Markov Chain Sampling for Non-linear State Space Models using Embedded Hidden Markov Models", Technical Report No. 0304, Department of Statistics, University of Toronto, http://arxiv.org/abs/math/0305039. 797, 799, 800, 808

Neal, R. M., Beal, M. J., and Roweis, S. T. (2004) "Inferring state sequences for non-linear systems with embedded hidden Markov models", in S. Thrun, et al (editors), *Advances in Neural Information Processing Systems 16*, MIT Press. 797, 799, 800

Pitt, M. K. and Shephard, N. (1999) "Filtering via Simulation: Auxiliary Particle Filters", *Journal of the American Statistical Association*, vol. 94, no. 446, pp. 590–599. MR1702328. doi: https://doi.org/10.2307/2670179. 805

Shestopaloff, A. Y. and Neal, R. M. (2013) "MCMC for non-linear state space models using ensembles of latent sequences", Technical Report, http://arxiv.org/abs/1305.0320. 798, 820

Shestopaloff, A. Y. and Neal, R. M. (2014) "Efficient Bayesian inference for stochastic volatility models with ensemble MCMC methods", Technical Report, http://arxiv.org/abs/1412.3013. 799, 820

Shestopaloff, A. Y. and Neal, R. M. (2016) "Sampling latent states for high-dimensional non-linear state space models with the embedded HMM method", Technical Report, https://arxiv.org/abs/1602.06030. 806

Shestopaloff, A. Y. and Neal, R. M. (2017) "Supplementary Material for "Sampling latent states for high-dimensional non-linear state space models with the embedded HMM method"," *Bayesian Analysis*. dois: http://dx.doi.org/10.1214/17-BA1077SUPP, http://dx.doi.org/10.1214/17-BA1077SUPPB 813

Shestopaloff, A. Y. (2016) "MCMC Methods for Non-Linear State Space Models". PhD Thesis, University of Toronto, https://tspacenew.library.utoronto.ca/handle/1807/73175. MR3564012. 799

Whiteley, N. (2010) Discussion of: "Particle Markov chain Monte Carlo methods" by Andrieu, C., Doucet, A. and Holenstein, R. (2010), *Journal of the Royal Statistical Society B*, vol. 72, pp. 269–342. MR2758115. doi: https://doi.org/10.1111/j.1467-9868.2009.00736.x. 798

Acknowledgments

We thank Arnaud Doucet for helpful comments. This research was supported by the Natural Sciences and Engineering Research Council of Canada. A. S. is in part funded by an NSERC Postgraduate Scholarship. R. N. holds a Canada Research Chair in Statistics and Machine Learning.