MCMC Methods for Non-linear State Space Models

by

Alexander Y. Shestopaloff

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Statistical Sciences
University of Toronto

Abstract

MCMC Methods for Non-linear State Space Models

Alexander Y. Shestopaloff
Doctor of Philosophy
Graduate Department of Statistical Sciences
University of Toronto
2016

This thesis is concerned with developing efficient MCMC (Markov Chain Monte Carlo) techniques for non-linear, non-Gaussian state space models. These models are widely applicable in many fields, such as population ecology and economics. However, Bayesian inference in such models remains challenging, since sampling the latent variables and parameters in them can be difficult.

We begin by showing how an efficient MCMC method can be developed for a queueing model, inference for which has previously been done with ABC (Approximate Bayesian Computation). This example is an illustration of how we can do inference with MCMC in a model in which inference is considered difficult and for which ABC was used. We do this by expanding the state to include variables involved in the process used to generate the observed data.

We then proceed to the main contributions of the thesis, which are developing general MCMC methods for non-linear, non-Gaussian state space models based on the ensemble MCMC and embedded HMM MCMC frameworks. The embedded HMM and ensemble MCMC methods we introduce provide a unified framework that is sufficiently flexible to often address difficult issues in MCMC sampling for non-linear, non-Gaussian state space models, without having to resort to specialized methods for different cases.

We start with the basic embedded HMM method of Neal (2003), and Neal, Beal and Roweis (2004), designed for sampling latent state sequences given known parameter values, and develop an efficient ensemble MCMC method for sampling parameters when the parameters and the latent sequence are strongly dependent. This method is tested

on a non-linear model of population dynamics. We then show how ensembles can be used to make parameter sampling more efficient in a stochastic volatility model by making use of a caching technique (Neal (2006)). Finally, we develop an ensemble MCMC method for sampling sequences in non-linear, non-Gaussian state space models when the state space is high-dimensional and test it on two sample models with high-dimensional state spaces, one of which is multimodal, and compare the performance of this method to a particle MCMC method.

Acknowledgements

My advisor Radford Neal, for teaching me a critical and inquisitive approach towards statistical methods, his infectious optimism and enthusiasm, his devotion to elegant and thorough research. And for never expressing any doubt that I would eventually be able to produce a good thesis, and supporting this research through all of its ups and downs.

Radu Craiu and Geoffrey Hinton, my thesis committee members, for their help and advice throughout the years. Jeff Rosenthal and Michael Evans for reading the thesis. Arnaud Doucet for being my external examiner and suggesting to clarify the proof in the last chapter.

My undergraduate statistics professors for laying a solid foundation for my future studies, most notably Philip McDunnough, Andrey Feurverger, Balint Virag and Jeremy Quastel.

The excellent staff of the Department who made it warm and welcoming and always ready to help with any sorts of issues: Andrea, Angela, Annette, Christine, Laura and Dermot.

My fellow students whom I've had a chance to meet: Muz, Becky (Wei), Shivon, Zeynep, Ramya, Avideh, Chunyi, Cody, Elif, Li Li and many others.

Finally, my dear parents Yuri and Valentina and my brother Konstantin for always being encouraging in innumerable ways throughout my school years.

The research in this thesis was funded by scholarships from Government of Ontario (OGSST and OGS), Government of Canada (NSERC PGS-D), Department of Statistics at the University of Toronto and the NSERC grants of Radford Neal.

Contents

1 Introduction **1**

1.1 Problem 1

1.2 Previous approaches 2

1.3 Outline of thesis 4

2 Queueing Model **6**

2.1 The model 6

2.2 MCMC sampling 7

2.2.1 Gibbs updates for arrival times 8

2.2.2 Simple Metropolis updates for parameters 9

2.2.3 Shift and scale updates 10

2.3 An empirical study 13

2.3.1 Simulated data from three scenarios 13

2.3.2 Experimental setup 14

2.3.3 Results 17

2.4 Conclusion 20

3 Ensemble MCMC for State Space Models **25**

3.1 Introduction 26

3.2 Ensemble MCMC 27

3.3 Embedded HMM MCMC as an ensemble MCMC method 29

3.4 The single-sequence embedded HMM MCMC method 31

3.5 An ensemble extension of the embedded HMM method 32

3.6 Staged ensemble MCMC sampling 34

3.7 Constructing pools of states 36

3.8 Performance comparisons on a population dynamics model 37

3.9 Conclusion 45

4 Stochastic Volatility **46**

4.1 Bayesian inference for the stochastic volatility model 47

4.1.1 A linear Gaussian approximation for sampling latent sequences . . 47

4.1.2 ASIS updates 49

4.2 Ensemble MCMC methods for stochastic volatility models 51

4.2.1 Choosing the pool distribution 55

4.3 Comparisons 56

4.3.1 Data 56

4.3.2 Sampling schemes and tuning 57

4.3.3 Results 59

4.4 Conclusion 60

5 Embedded HMMs for High Dimensions **65**

5.1 Introduction 65

5.2 Embedded HMM MCMC 68

5.3 Particle Gibbs with Backward Sampling MCMC 69

5.4 An embedded HMM sampler for high dimensions 71

5.4.1 Pool state distributions 71

5.4.2 Sampling pool states 72

5.4.3 Autoregressive updates 73

5.4.4 Shift updates 74

5.4.5 Flip updates 75

5.4.6 Relation to PGBS 76

5.5 Proof of correctness 76

5.6 Experiments 79

5.6.1 Test models 79

5.6.2 Single-state Metropolis Sampler 80

5.6.3 Particle Gibbs with Backward Sampling with Metropolis 82

5.6.4 Tuning the Baseline Samplers 83

5.6.5 Embedded HMM sampling 84

5.6.6 Comparisons 85

5.7 Conclusion 89

6 Conclusion **91**

7 References **93**

Chapter 1

Introduction

1.1 Problem

This thesis is motivated by the problem of developing efficient MCMC methods for Bayesian inference in non-linear, non-Gaussian state space models of time series. This is a class of statistical models that is used in economics, population ecology, engineering and other fields. For instance, the popular stochastic volatility model in economics is an example of a state space model where the observations depend non-linearly on the state variables. This model is used to study how the volatility of log-returns for some asset changes over time. Many efforts have been devoted to efficient Bayesian inference for this model; a recent example is the work of Kastner and Fruhwirth-Schnatter (2014). Another example is the Ricker model of population dynamics (see, for example, Wood (2010)). Both of these models are used as examples of non-linear, non-Gaussian state space models later in the thesis.

The general non-linear, non-Gaussian state space model describes the distribution of some observed sequence $y = (y_1, \ldots, y_n)$ via an unobserved Markov process $x = (x_1, \ldots, x_n)$. This Markov process has initial density $p(x_1|\theta)$ and transition densities $p(x_i|x_{i-1}, \theta)$. The observed data at time i is assumed to be drawn from some observation density $p(y_i|x_i, \theta)$. We are interested in inferring x and θ, or both of these. In a Bayesian approach to this problem, we could use MCMC (Markov Chain Monte Carlo) to draw a sample from the posterior of x and θ and perform inferences based on this sample.

There are two cases of state space models where exact and efficient sampling for the latent variables and parameters can be performed. These are hidden Markov models with a finite state space, for which we can use the forward-backward algorithm, and linear Gaussian models, for which we can use the Kalman filter. If the model is non-linear, or has non-Gaussian innovations or observations, or both, these straightforward

approaches no longer apply, and approximate methods such as Monte Carlo or Laplace approximations are needed.

When using MCMC, sampling can be challenging, since the dimensionality of the posterior distribution over the latent variables and the parameters is usually high, and complex dependencies amongst the different variables are usually present. In particular, sampling the latent variables x_i can be difficult if there are strong dependencies amongst the x_i, with the marginal distribution of each x_i much more diffuse than the distribution of x_i conditional on the other variables. Simple methods that update one variable at a time can work poorly in these cases. However, even if the latent states can be sampled efficiently given a fixed value of θ, sampling can still be inefficient if we are updating the parameters and latent variables by alternately sampling a value for θ and a value for x. This is because the distribution of θ given a specific value of x can be much more constrained than the marginal distribution of θ. In this case, the whole sampler can be inefficient due to slow exploration of the possible values of θ. MCMC sampling can also be slowed down if the amount of information that the observations contain varies. In this case, a sampler that uses a fixed proposal for all variables can end up sampling a subset of the variables inefficiently.

Additional complications arise if the state, x, is high-dimensional. When the dimensionality of the state is greater than one, sampling can be made difficult by dependencies amongst the latent variables at a single time, as well as different times. Designing good samplers, and in particular good proposal distributions for such samplers, can be considerably more challenging than in the one-dimensional case. In higher dimensions, handling multiple modes in the posterior distribution for x also becomes more difficult. In low dimensions, it is sometimes possible to construct global proposals for x, covering the entire space. In this case, we might then reasonably expect to efficiently cover all the different modes of the posterior distribution and have an efficient sampler. However, in high dimensions such a strategy is no longer feasible, and the sampler must take this into account, by, for example, using proposals that are both localized within a mode and able to switch between the different modes of the distribution.

1.2 Previous approaches

Numerous approaches to Bayesian inference in non-linear, non-Gaussian state space models via MCMC and other methods exist. Well-known sampling methods such as Metropolis or Hamiltonian Monte Carlo can be applied to state space models, often with good results. In addition to these well-known methods, specialized sampling methods for mod-

els with a state space structure have been developed. Some of these methods aim to solve the problem of sampling latent state sequences given the parameters, while other methods are designed to jointly sample the parameters and the latent variables.

The embedded HMM method of (Neal (2003), Neal, Beal and Roweis (2004)) is one such method, developed to specifically address the problem of sampling state sequences x in cases where there are strong dependencies amongst the consecutive states. This method was initially proposed and tested on one-dimensional models with known parameters, and was shown to be effective in improving the efficiency of sampling state sequences. The embedded HMM idea is an important building block of the methods we propose and develop in this thesis. Introduced as a general method, many open questions remained unanswered how to best tune this method, scale it to higher dimensions and reduce its computational cost.

Two other methods for state sampling in non-linear, non-Gaussian state space models are Particle Gibbs and Particle Gibbs with Backward Sampling (Andrieu, Doucet and Holenstein (2010), Lindsten and Schon (2012)). These are MCMC methods for sampling latent sequences in non-linear, non-Gaussian state space models based on Sequential Monte Carlo. They work well for models with low-dimensional state spaces, but do not scale well due to their reliance on importance sampling. The Particle Gibbs method can be applied even if the transition density of the model cannot be evaluated explicitly (but can be sampled from). The backward sampling modification (which does require transition density evaluation) can substantially improve the Particle Gibbs method. We show later how this method is related to the embedded HMM method.

Specific MCMC methods and updates exist for models that have a particular structure. For example, for the stochastic volatility model, efficient sampling methods have been developed which rely on being able to closely approximate the observation density by a mixture of Gaussians (Kastner and Fruhwirth-Schnatter (2014)). "Interweaving" updates (Yu and Meng (2001)) that are designed with the dependencies between different variables in mind — for example a mean parameter and the latent state x — have also been successfully applied to improve sampling efficiency in cases where there are strong dependencies between certain parameters and latent variables. However, such methods and updates are not universal, and do not apply with equal success to different non-linear, non-Gaussian state space models.

Another, non-MCMC approach to Bayesian inference in state space models is Approximate Bayesian Computation (ABC). This is a very general approach. ABC makes inference possible in models where densities for model parameters, given observed data, cannot be written down explicitly, but we can simulate data from the model. For state

space models, the density for model parameters is a marginal over latent states, and this density typically cannot be computed. However, simulating data from state space models is usually not hard. The basic procedure for ABC methods is as follows. First, a value for the model parameter, θ, is sampled from the prior. Then, data is simulated from the model with parameters set to θ. For state space models, simulating data is done by first simulating a sequence of latent states, and then observations given these latent states. Finally, the simulated observations are compared to the observed data, and a parameter θ is added to the sample if the simulated data is sufficiently close to the observed data. See, for example, Fearnhead and Prangle (2012) for a description of ABC and how it can be applied to a variety of models.

The advantage of the ABC method is its wide applicability. Its main disadvantage is that it is hard to make efficient, since specialized knowledge in designing summary statistics for a model is often required. Also, the performance of this method is hard to evaluate due to its approximate nature. This is in contrast to MCMC methods, which, at least in theory, should all produce the same answer in the end. The literature on ABC is extensive and growing, but it is not always clear if ABC is better than a well-designed MCMC method. The contributions in this thesis are relevant to this issue.

1.3 Outline of thesis

In Chapter 2, we start with a motivating example of a model that can be viewed in state space form, a model for a queue. We show how an efficient MCMC method can be designed for this model, in which inference is considered to be difficult, and for which other methods such as Approximate Bayesian Computation have previously been used. This chapter serves as an illustration of how an MCMC sampler can be designed for a model with strong dependencies between the parameters and the latent variables. The main idea is to design MCMC updates with proposals that take these dependencies into account. We show that using such updates can make sampling much more efficient compared to a basic Gibbs sampler.

We then move on to the main contributions. Throughout this thesis, another key method that we rely on is ensemble MCMC (Neal (2010)). An ensemble MCMC method simultaneously looks at an "ensemble" of multiple points when performing an update, with the ensemble chosen in a way that makes it possible to compute the density at all points of the ensemble in less time it would take to compute the density at all K points of the ensemble separately (else, it would likely be more efficient to perform K updates separately). The embedded HMM method is an example of an ensemble MCMC

method, where the ensemble is that of latent sequences. One of the overall goals we pursue is to show that the ensemble framework is sufficiently flexible to address many of the difficulties of sampling in state space models.

In particular, Chapter 3 focuses on developing an ensemble MCMC method for sampling parameters in non-linear, non-Gaussian state space models by considering an ensemble of latent sequences at once. We show that ensemble methods are particularly effective when there is a strong dependence between the current value of the latent sequence and the parameters. The key idea leading to improved performance is that a parameter proposal given a collection of sequences is at least as likely to be accepted as when given a single sequence. We also show how ensemble methods can be sped up by the introduction of a delayed acceptance step, in which a proposed set of parameters is accepted or rejected using a computationally cheap approximation to the likelihood.

Chapter 4 develops ensemble MCMC methods further, this time using a stochastic volatility model as an example. In this chapter we consider ensembles not only over latent sequences but also over parameters. We show that using an ensemble over parameter values, made possible by the use of a caching technique, can lead to improved sampling performance. The ensemble method is shown to perform better than a state of the art MCMC method specially designed for the stochastic volatility model, but without using the same restrictive assumptions that this comparison method does.

Finally, Chapter 5 shows how the embedded HMM method can be made to work in models with high-dimensional state spaces, by introducing a new way of selecting latent sequences that is related to how the Particle Gibbs method operates. This new method of selecting the ensemble also allows us to reduce the computational cost of the method. In this chapter we also consider an example when the posterior has multiple modes and show how sampling can be made efficient by using special updates to construct an ensemble of suitable latent sequences.

Chapter 2

Queueing Model

This chapter proposes a new approach to computation for Bayesian inference for the M/G/1 queue (Markovian arrival process/General service time distribution/1 server). Inference for this model using ABC (Approximate Bayesian Computation) was previously considered by Bonassi (2013), Fearnhead and Prangle (2012), and Blum and Francois (2010). ABC, in general, does not yield samples from the exact posterior distribution. We use the strategy of considering certain unobserved quantities as latent variables, allowing us to use Markov Chain Monte Carlo (MCMC), which converges to the exact posterior distribution.[1]

2.1 The model

In the M/G/1 queueing model, customers arrive at a single server with independent interarrival times, W_i, distributed according to the Exp(θ_3) distribution. Here, θ_3 is the arrival rate; hence W_i has density function $f(w_i) = \theta_3 \exp(-\theta_3 w_i)$ for $w_i \geq 0$ and 0 otherwise. They are served with independent service times U_i, which have a Uniform (θ_1, θ_2) distribution. (Our MCMC approach can be generalized to other service time distributions.) We do not observe the interarrival times, only the interdeparture times, Y_i. The goal is to infer the unknown parameters of the queueing model, $\theta = (\theta_1, \theta_2, \theta_3)$, using observed interdeparture times, $y = (y_1, \ldots, y_n)$. We assume that the queue is empty before the first arrival.

The process of interdeparture times can be written as

$$Y_i \;=\; U_i + \max\left(0, \sum_{j=1}^{i} W_j - \sum_{j=1}^{i-1} Y_j\right) \;=\; U_i + \max(0, V_i - X_{i-1}) \tag{2.1}$$

[1] Material in this chapter was first presented in Shestopaloff and Neal (2013b).

where W_j is the time from the arrival of customer $j-1$ (or from 0 when $j=1$) to the arrival of customer j, $V_i = \sum_{j=1}^{i} W_j$ is the arrival time of the i-th customer, and $X_i = \sum_{j=1}^{i} Y_j$ is the departure time of the i-th customer, with X_0 defined to be 0.

We take the arrival times, V_i, to be latent variables. The V_i evolve in time as a Markov process. Viewed this way, the queueing model can be summarized as follows:

$$\begin{aligned} V_1 &\sim \text{Exp}(\theta_3) && (2.2)\\ V_i|V_{i-1} &\sim V_{i-1} + \text{Exp}(\theta_3) && (2.3)\\ Y_i|X_{i-1}, V_i &\sim \text{Uniform}(\theta_1 + \max(0, V_i - X_{i-1}),\ \theta_2 + \max(0, V_i - X_{i-1})) && (2.4) \end{aligned}$$

where $i = 1, \ldots, n$. For convenience, set $v = (v_1, \ldots, v_n)$. The joint density of the V_i and the Y_i can be factorized as follows

$$P(v, y|\theta) = P(v_1|\theta) \prod_{i=2}^{n} P(v_i|v_{i-1}, \theta) \prod_{i=1}^{n} P(y_i|v_i, x_{i-1}, \theta) \quad (2.5)$$

Let $\pi(\theta)$ be a prior for θ. The posterior distribution $\pi(\theta|y)$ of θ is then

$$\pi(\theta|y) \propto \pi(\theta) \int \cdots \int P(v_1|\theta) \prod_{i=2}^{n} P(v_i|v_{i-1}, \theta) \prod_{i=1}^{n} P(y_i|v_i, x_{i-1}, \theta) dv_1 \cdots dv_n \quad (2.6)$$

The integral (2.6) cannot be computed analytically. Hence to sample from $\pi(\theta|y)$ we need to include the V_i in the MCMC state and sample from the joint posterior distribution of the V_i and θ given the data, which is

$$\pi(v, \theta|y) \propto \pi(\theta) P(v_1|\theta) \prod_{i=2}^{n} P(v_i|v_{i-1}, \theta) \prod_{i=1}^{n} P(y_i|v_i, x_{i-1}, \theta) \quad (2.7)$$

Taking the values of θ from each draw from (2.7) and ignoring the V_i will yield a sample from the posterior distribution of θ.

2.2 MCMC sampling

Our MCMC procedure will be based on combining five different updates. The first is a Gibbs update that draws new values for each V_i, given values of the other V_i, θ, and the data. The second is a simple Metropolis update for θ, given values of the V_i and the data. The last three updates are novel — one Metropolis "shift" update and two Metropolis-Hastings-Green "scale" updates that propose to simultaneously change components of θ

and all of the V_i. These updates are related to those proposed by Liu and Sabatti (2000).

2.2.1 Gibbs updates for arrival times

We first work out how to apply standard Gibbs sampling updates for the V_i, for which we need to derive the full conditional density for each V_i given the other V_i and θ. We denote all of the V_j except V_i as V_{-i}. We consider three cases, when $i = 1$, when $2 \leq i \leq n-1$ and when $i = n$.

For the case $i = 1$, we have

$$\begin{aligned} P(v_1|v_{-1}, y, \theta) &\propto P(v_1)P(y_1|v_1, \theta)P(v_2|v_1, \theta) \\ &\propto \theta_3 e^{-\theta_3 v_1} I(0 \leq v_1) \frac{I(y_1 \in [\theta_1 + v_1, \theta_2 + v_1])}{\theta_2 - \theta_1} \\ &\quad \times \theta_3 e^{-\theta_3(v_2 - v_1)} I(v_1 \leq v_2) \\ &\propto I(v_1 \in [\max(0, x_1 - \theta_2), \min(v_2, x_1 - \theta_1)]) \end{aligned} \tag{2.8}$$

So, conditional on the parameters and the observed data, the distribution of V_1 is Uniform$(\max(0, x_1 - \theta_2), \min(v_2, x_1 - \theta_1))$.

When $2 \leq i \leq n-1$, we have

$$\begin{aligned} P(v_i|v_{-i}, y, \theta) &\propto P(v_i|v_{i-1}, \theta)P(y_i|x_{i-1}, v_i, \theta)P(v_{i+1}|v_i, \theta) \\ &\propto \theta_3 e^{-\theta_3(v_i - v_{i-1})} I(v_{i-1} \leq v_i) \\ &\quad \times \frac{I(y_i \in [\theta_1 + \max(0, v_i - x_{i-1}), \theta_2 + \max(0, v_i - x_{i-1})])}{\theta_2 - \theta_1} \\ &\quad \times \theta_3 e^{-\theta_3(v_{i+1} - v_i)} I(v_i \leq v_{i+1}) \\ &\propto I(v_i \in [v_{i-1}, v_{i+1}]) I(y_i \in [\theta_1 + \max(0, v_i - x_{i-1}), \theta_2 + \max(0, v_i - x_{i-1})]) \end{aligned} \tag{2.9}$$

To simplify this expression, note that $x_i = y_i + x_{i-1}$, and first consider the case $y_i > \theta_2$. When this is so, we must have $v_i > x_{i-1}$, hence, in this case V_i will have a Uniform distribution on $[x_i - \theta_2, \min(v_{i+1}, x_i - \theta_1)]$. Now consider the case $y_i \leq \theta_2$. We rewrite expression (2.10) as

$$\begin{aligned} & I(v_i \in [v_{i-1}, v_{i+1}]) I(y_i \in [\theta_1 + \max(0, v_i - x_{i-1}), \theta_2 + \max(0, v_i - x_{i-1})]) I(v_i \leq x_{i-1}) \\ &+ I(v_i \in [v_{i-1}, v_{i+1}]) I(y_i \in [\theta_1 + \max(0, v_i - x_{i-1}), \theta_2 + \max(0, v_i - x_{i-1})]) I(v_i > x_{i-1}) \\ &= I(v_i \in [v_{i-1}, x_{i+1}]) + I(v_i \in (x_{i-1}, \min(v_{i+1}, x_i - \theta_1)]) \\ &= I(v_i \in [v_{i-1}, \min(v_{i+1}, x_i - \theta_1)]) \end{aligned} \tag{2.10}$$

We see that for $y_i \leq \theta_2$, V_i will have a Uniform distribution on $[v_{i-1}, \min(v_{i+1}, x_i - \theta_1)]$.

Finally, for the case $i = n$, we have

$$\begin{aligned} P(v_n|v_{-n}, y, \theta) &\propto P(v_n|v_{n-1})P(y_n|v_n) \\ &\propto e^{-\theta_3(v_n - v_{n-1})} I(v_{n-1} \leq v_n) \\ &\quad \times I(y_n \in [\theta_1 + \max(0, v_n - x_{n-1}), \theta_2 + \max(0, v_n - x_{n-1})]) \end{aligned} \tag{2.11}$$

From this it follows that V_n will have an Exponential distribution, truncated to $[x_n - \theta_2, x_n - \theta_1]$ when $y_n > \theta_2$ and to $[v_{n-1}, x_n - \theta_1]$ when $y_n \leq \theta_2$.

All of the above distributions can be easily sampled from by using the inverse CDF method. The case $i = n$ deserves a small note. If we suppose the truncation interval of V_n is $[L, U]$, then the CDF and inverse CDF will be

$$F_{V_n}(v_n) = \frac{e^{-\theta_3 L} - e^{-\theta_3 v_n}}{e^{-\theta_3 L} - e^{-\theta_3 U}}, \quad F_{V_n}^{-1}(y) = \log((1-y)e^{-\theta_3 L} + ye^{-\theta_3 U}) \tag{2.12}$$

So we can sample V_n as $F_{V_n}^{-1}(R)$ where R is Uniform$(0, 1)$.

2.2.2 Simple Metropolis updates for parameters

We use simple Metropolis updates (Metropolis, et al (1953)) to sample from the conditional posterior distribution of θ given values of the V_i and the data, $\pi(\theta|v, y) \propto \pi(v, \theta|y)$.

When doing simple Metropolis updates of θ given the arrival times and the data, we used the 1-to-1 reparametrization $\eta = (\eta_1, \eta_2, \eta_3) = (\theta_1, \theta_2 - \theta_1, \log(\theta_3))$.

A simple Metropolis update for η that leaves $\pi(\eta|v, y)$ invariant proceeds as follows. We choose a symmetric proposal density $q(\cdot|\eta)$, for which $q(\eta^*|\eta) = q(\eta|\eta^*)$. Given the current value η, we generate a proposal $\eta^* \sim q(\eta^*|\eta)$. We compute the acceptance probability

$$a = \min\left(1, \frac{\pi(\eta^*|v, y)}{\pi(\eta|v, y)}\right) \tag{2.13}$$

and let the new state, η', be η^* with probability a and otherwise reject the proposal, letting $\eta' = \eta$.

We use a normal proposal with independent coordinates centered at the current value of η, updating all components of η at once. If the proposed value for η_1 or η_2 is outside its range, the proposal can be immediately rejected.

To prevent overflow or underflow, all MCMC computations use the logarithm of the

posterior (2.7), which (up to an additive constant) simplifies to

$$\log \pi(v, \theta|y) \quad = \quad \log(\pi(\theta)) + n \log(\theta_3) - \theta_3 v_n - n \log(\theta_2 - \theta_1) \tag{2.14}$$

whenever the following constraints are satisfied

$$\theta_2 - \theta_1 > 0 \tag{2.15}$$

$$\theta_1 \leq y_1 - v_1 \tag{2.16}$$

$$\theta_2 \geq y_1 - v_1 \tag{2.17}$$

$$\theta_1 \leq \min(y_i - \max(0, v_i - x_{i-1})) \quad \text{for all } i \geq 2 \tag{2.18}$$

$$\theta_2 \geq \max(y_i - \max(0, v_i - x_{i-1})) \quad \text{for all } i \geq 2 \tag{2.19}$$

Otherwise, $\log(\pi(v, \theta)|y)) = -\infty$.

The dominant operation when computing the log likelihood for a data set of length n is checking the constraints (2.15) to (2.19), which requires time proportional to n. Storing the constraints on θ_1 and θ_2 given by (2.16) to (2.19) during evaluation of the log likelihood at (θ, v_i) allows us to evaluate the log likelihood for another θ^* and the same v_i in constant time. This allows us to do K additional Metropolis updates for less than K times the computational cost it takes to do a single update, which is likely to make sampling more efficient. A similar improvement in sampling should be possible for models with other service time distributions that have low-dimensional sufficient statistics. As well, doing additional simple Metropolis updates makes an imperfect choice of proposal distribution have less of an effect on sampling efficiency.

2.2.3 Shift and scale updates

The Gibbs and simple Metropolis updates are sufficient to give an ergodic MCMC scheme. However, as we will see, these updates are sometimes very inefficient when used on their own. In this section, we introduce our novel shift and scale updates, which can make MCMC sampling much more efficient.

Shift updates and scale updates are used to sample from the joint posterior distribution of the parameters and the latent variables, $\pi(v, \theta|y)$. The shift update takes the form of a standard Metropolis update, while the scale updates are Metropolis-Hastings-Green (MHG) updates.

In general, an MHG update proceeds as follows. We introduce an extra variable $z \in \{-1, +1\}$. Given a current value (v, θ) and a density $q(\cdot|v, \theta)$ we generate $z \sim q(z|v, \theta)$.

We then propose $(v^*, \theta^*, z^*) = g(v, \theta, z)$. The function g is the inverse of itself, with a Jacobian $|\det(\nabla_{(v,\theta)} g(v, \theta, z))|$ which is nowhere zero or infinite. Here, $\nabla_{(v,\theta)} g(v, \theta, z)$ is a matrix of derivatives with respect to v and θ. We compute the acceptance probability

$$a = \min\left(1, \frac{\pi(\theta^*, v^*|y) q(z^*|v^*, \theta^*)}{\pi(\theta, v|y) q(z|v, \theta)} |\det(\nabla_{(v,\theta)} g(v, \theta, z))|\right) \tag{2.20}$$

and let the new state, (v', θ') be (v^*, θ^*) with probability a and let $(v', \theta') = (v, \theta)$ otherwise. For more on the MHG algorithm, see Geyer (2003).

The motivation for the shift updates and scale updates is that conditioning on given values of the arrival times constrains the range of parameter values for which the posterior density is non-zero, preventing us from changing the parameters by a significant amount. This can lead to inefficient sampling when θ is updated given the V_i. In contrast, our new shift and scale updates change the latent variables and the parameters simultaneously, in accordance with their dependence structure, thus allowing for much greater changes to the parameters.

Hard constraints on θ given v aren't necessary for these updates to be beneficial. If the service time density were non-zero for all positive values, the distribution of θ given v might still be much more concentrated than the marginal posterior for θ.

Shift updates

We first consider updates that shift both the minimum service time, θ_1, and all the arrival times, V_i, keeping the range of service times, $\theta_2 - \theta_1$, fixed. Note that for $i = 1$ and for all i when $X_{i-1} < V_i$, we have $\theta_1 < X_i - V_i$. Hence the values of one or more of the V_i will constrain how much we can propose to change θ_1 — any proposal to change θ_1 that violates these constraints must be rejected.

A shift update addresses the presence of these constraints as follows. We first draw a shift s from any distribution symmetric around 0. Then, we propose new values $V_i^* = V_i - s$ for all i and $\theta_1^* = \theta_1 + s$. So, a proposal to increase the minimum service time is coupled with a simultaneous proposal to decrease all of the arrival times in a way that keeps the constraints between the arrival times and the minimum service time satisfied.

A shift proposal will be rejected if it proposes a value for V_1 or θ_1 that is less than 0. Otherwise, the acceptance probability for a shift update is

$$a = \min\left(1, \frac{\pi(\theta^*, v^*|y)}{\pi(\theta, v|y)}\right) \tag{2.21}$$

Range scale updates

Next, we consider range scale updates that propose to simultaneously change the latent variables and the range of service times $\theta_2 - \theta_1$, which is also constrained by the latent variables — specifically, for V_1 and for all $i > 2$ where $V_i > X_{i-1}$, we have $\theta_2 - \theta_1 > X_i - \theta_1 - V_i$. These updates propose to scale the "gaps" $X_i - \theta_1 - V_i$ (gaps between the current arrival time and the latest possible arrival time), while at the same time proposing to scale $\theta_2 - \theta_1$ by the same amount, keeping θ_1 fixed.

In particular, we first fix (or draw from some distribution) a scale factor $c_{\text{range}} > 0$. We then draw $z \sim \text{Uniform}\{-1, 1\}$. The function $g(v, \theta, z)$ proposes to change V_i to $V_i^* = (X_1 - \theta_1) - c_{\text{range}}^z(X_i - \theta_1 - V_i)$ and $\theta_2 - \theta_1$ to $c_{\text{range}}^z(\theta_2 - \theta_1)$. We set $z^* = -z$. The Jacobian $|\det(\nabla_{(v,\theta)} g(v, \theta, z))| = c_{\text{range}}^{z(n+1)}$.

A range scale proposal will be rejected if it proposes a set of V_i for which some $V_i < V_{i-1}$ or for which $V_1 < 0$. Otherwise, the MHG acceptance probability for a range scale update is

$$a = \min\left(1, \frac{\pi(\theta^*, v^*|y)}{\pi(\theta, v|y)} c_{\text{range}}^{z(n+1)}\right) \tag{2.22}$$

Rate scale updates

Finally, we consider rate scale updates that propose to simultaneously change both the interarrival times $W_i = V_i - V_{i-1}$, and the arrival rate θ_3. We motivate these updates as follows. We expect the distribution of θ_3 given the interarrival times W_i to be concentrated around $1/\overline{W}$. Consequently, we would expect the joint posterior of θ_3 and the W_i to have a high density along a ridge with values cW_i and θ_3/c for some c, as long as cW_i and θ_3/c do not lie in a region with low probability. So, proposing to change W_i to cW_i and θ_3 to θ_3/c for some c potentially keeps us in a region of high density.

We perform these updates in terms of $\eta_3 = \log(\theta_3)$. In detail, this update proceeds as follows. We first fix (or draw from some distribution) a scale $c_{\text{rate}} > 0$. We then draw $z \sim \text{Uniform}\{-1, 1\}$. The function $g(v, \eta, z)$ proposes to change all W_i to $c_{\text{rate}}^z W_i$ and η_3 to $\eta_3 - \log(c_{\text{rate}}^z)$. We set $z^* = -z$. The Jacobian $|\det(\nabla_{(v,\eta)} g(v, \eta, z))| = c_{\text{rate}}^{zn}$.

We reject the proposal immediately if it violates the following constraints: for i where $y_i > \theta_2$ the proposed arrival time $V_i^* = \sum_{i=1}^n c_{\text{rate}}^z W_i$ must be in $[x_i - \theta_2, x_i - \theta_1]$, and for i where $y_i \leq \theta_2$, we must have $V_i^* \leq x_i - \theta_1$. Otherwise, the MHG acceptance probability for a rate scale update is

$$a = \min\left(1, \frac{\pi(\eta^*, v^*|y)}{\pi(\eta, v|y)} c_{\text{rate}}^{zn}\right) \tag{2.23}$$

Ergodicity

Although a scheme consisting of the shift, range scale, and rate scale updates changes all of the latent variables and parameters, it is not ergodic. To see this, consider updating the arrival times V_i, or equivalently, interarrival times $W_i = V_i - V_{i-1}$. Shift updates do not change the interarrival times. Range scale updates with scale factor c_{range} change each interarrival time as $W_i^* = y_i(1 - c_{\text{range}}^z) + c_{\text{range}}^z W_i$, and rate scale updates with scale factor c_{rate} change each interarrival time as $W_i^* = c_{\text{rate}}^z W_i$. If we are in a situation where $W_i = W_j$ and $y_i = y_j$ for one or more $j \neq i$, then all subsequent updates will keep $W_i = W_j$. Note that this is true even if c_{range} and c_{rate} are randomly selected at each update.

Hence, our novel updates must still be combined with simple Gibbs sampling updates of the V_i's to get an ergodic sampling scheme. We also still do simple Metropolis updates for θ. Although this is not essential for ergodicity, it makes sampling a lot more efficient.

2.3 An empirical study

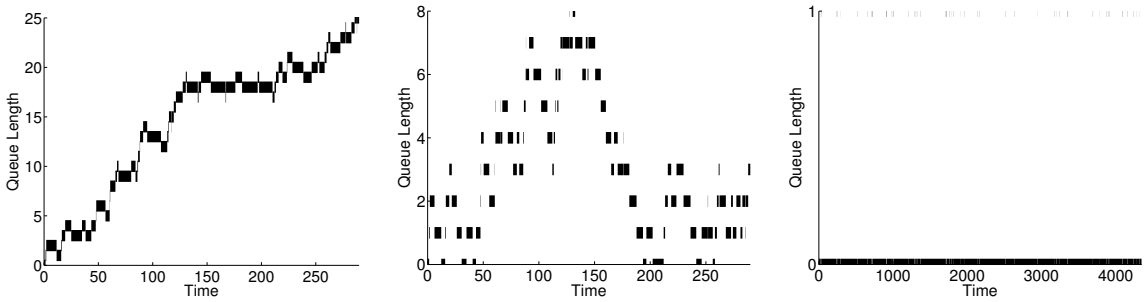
The goal of our empirical study is to determine when using the novel updates improves sampling efficiency. We generate three simulated data sets that are representative of a range of possible scenarios, sample from the posterior distributions of θ for each of these data sets using various sampling schemes, and compute autocorrelation times to compare sampling efficiency.

2.3.1 Simulated data from three scenarios

The sort of data arising from the observation of a queueing system can be roughly classified into one of three scenarios.

The first scenario is when the interarrival times are, on average, smaller than the service times, that is, arrivals are relatively frequent. In this case the queue is generally full, and empty only early on. Hence, most observed interdeparture times will tend to come from the service time distribution, and only a small number of interdeparture times will be relevant for inferring the arrival rate. Consequently, we expect there to be large uncertainty in θ_3, but less for θ_1 and θ_2.

The second scenario is when interarrival times are on average slightly larger than the service times. The queue is then sometimes empty, and sometimes contains a number of people. In this case, we expect the data to be informative about all parameters. This



(a) $\theta = (8, 16, 0.15)$ (b) $\theta = (4, 7, 0.15)$ (c) $\theta = (1, 2, 0.01)$

Figure 2.1: The three data sets.

is also the case of most practical interest, as it corresponds to how we may expect a queueing system to behave in the real world.

The third scenario is when arrivals happen rarely, while the service time is relatively small. In this case, the queue will usually be empty. Interarrival times are then informative for inferring θ_3, as most of them will come from the $\text{Exp}(\theta_3)$ distribution with a small amount of added $\text{Uniform}(\theta_1, \theta_2)$ noise. However, inference of the service time bounds θ_1 and θ_2 will now be based on how significantly the distribution of the interarrival times (for a given θ_3) deviates from the Exponential distribution. This difference may be rather small, and so we expect that the posterior for the service time bounds will be rather diffuse.

The three data sets we generated each consisted of $n = 50$ interdeparture times, corresponding to each of the three scenarios above. For the first scenario we take $\theta = (8, 16, 0.15)$, for the second, $\theta = (4, 7, 0.15)$, and for the third, $\theta = (1, 2, 0.01)$.

The plots in Figure 2.1 of the number of people in the queue up until the last arrival against time for each of the three data sets demonstrate that the data sets have the desired qualitative properties.

Numerical values of the simulated interdeparture times are presented in Table 2.1.

2.3.2 Experimental setup

We now compare the efficiency of five different MCMC schemes for performing Bayesian inference for θ by drawing samples from the posterior. These are

1) The basic scheme: Gibbs sampling for V_i with simple Metropolis updates for θ.
2) Basic scheme plus shift updates.
3) Basic scheme plus range scale updates.
4) Basic scheme plus rate scale updates.
5) Basic scheme plus shift, rate scale, and range scale updates.

Frequent	Intermediate	Rare
11.57	6.19	21.77
13.44	6.04	10.30
13.24	9.52	206.34
9.30	4.49	8.57
8.95	4.36	45.79
11.99	9.86	233.13
15.68	9.91	128.30
10.72	5.02	59.73
12.68	5.76	4.59
9.79	4.67	3.21
14.01	6.25	185.29
10.04	4.77	2.49
12.05	5.52	4.63
13.59	6.10	72.48
15.13	6.67	22.47
15.67	6.88	195.34
12.38	5.64	85.92
9.11	4.42	8.39
9.19	4.45	23.30
10.06	4.77	4.24
14.73	6.52	42.78
10.03	4.76	332.64
14.51	6.44	16.91
9.95	4.73	6.26
15.43	6.79	39.44
10.80	5.05	27.16
9.57	4.59	29.53
10.01	4.75	93.65
12.93	5.85	42.60
11.79	5.42	176.36
10.81	5.05	34.69
14.65	6.49	345.20
12.68	5.76	128.16
12.40	8.67	307.50
15.34	16.65	233.54
10.29	4.86	18.79
14.06	6.27	36.88
14.03	6.26	114.85
11.04	5.14	4.73
12.54	10.60	337.02
8.61	4.23	81.89
8.43	6.15	96.33
12.25	5.59	27.20
14.23	6.34	23.16
15.47	6.80	167.89
9.04	4.39	70.58
12.55	5.71	81.28
11.76	5.41	43.55
8.10	4.04	33.88
10.70	5.01	28.47

Table 2.1: Simulated interdeparture times y_i

We put Uniform$(0, 10)$ priors on θ_1 and on $\theta_2 - \theta_1$, and a Uniform$(0, 1/3)$ prior on θ_3. These are the priors that were used by Fearnhead and Prangle (2012). The prior $\pi(\theta)$ for θ is then

$$\pi(\theta) \quad \propto \quad I(\theta_1 \in [0, 10])I(\theta_2 - \theta_1 \in [0, 10])I(\theta_3 \in [0, 1/3]) \tag{2.24}$$

In the η parametrization, the priors on η_1 and η_2 remain Uniform$(0, 10)$, while the prior for η_3, due to the log transformation, is now proportional to $\exp(\eta_3)$ on $(-\infty, \log(1/3))$ and 0 otherwise.

In order to compare MCMC methods fairly, we need to reasonably tune their parameters (such as the standard deviations of Metropolis proposals). In practice, tuning is usually done by guesswork and trial and error. But for these experiments, in order to avoid the influence of arbitrary choices, we use trial runs to ensure a reasonable choice of tuning parameters, even though in practice the time for such trial runs might exceed the gain compared to just making an educated guess.

We chose the standard deviations for the normal proposal in the simple Metropolis updates for η by performing a number of pilot runs (using the basic scheme plus shift, rate scale, and range scale updates), finding estimates of the marginal posterior standard deviations of each component of η, and taking scalings of these estimated standard deviations as the corresponding proposal standard deviations.

The rationale for this choice of proposal standard deviations is as follows. One extreme case is when knowing the latent variables will give little additional information beyond the data for inferring the parameters, so the standard deviations of the marginal posterior will correspond closely to the standard deviations of the posterior given the latent variables and the data. The other extreme case is when the posterior given the latent variables and the data is a lot more concentrated than the marginal posterior, perhaps for a subset of the parameters. In most cases, however, the situation will be between these two extremes, so the marginal posterior standard deviations will provide a reasonable guide to setting proposal standard deviations. The case of rare arrivals is closer to the second extreme, when knowing the latent variables will tend to strongly constrain η_1 and η_2, so we take a much smaller scaling for them than for η_3.

For each simulated data set, we chose the number of simple Metropolis updates to perform in each iteration of a scheme by looking at the performance of the basic scheme with $1, 2, 4, 8, 16, 32$ Metropolis updates.

For each shift update, we drew a shift s from a $\mathrm{N}(0, \sigma^2_{\text{shift}})$ distribution. For both scale updates we set fixed scales c_{range} and c_{rate}. The chosen settings of tuning parameters for

Scenario	Est. stdev. of (η_1, η_2, η_3)	Scaling	Met. prop. stdev.	Met. updates	σ^2_{shift}	c_{range}	c_{rate}
Frequent	(0.1701, 0.2399, 0.3051)	0.7	(0.1191, 0.1679, 0.2136)	1	0.3	1.008	1.7
Intermediate	(0.0764, 0.1093, 0.1441)	1	(0.0764, 0.1093, 0.1441)	16	0.2	1.03	1.004
Rare	(0.6554, 2.0711, 0.1403)	(0.1, 0.1, 1)	(0.0655, 0.2071, 0.1403)	16	2	1.4	1.00005

Table 2.2: Tuning settings for the different samplers.

Scenario	Basic	Basic + Shift	Basic + Range	Basic + Rate	Basic + All
Frequent	20	10.8	10.8	9.6	5.1
Intermediate	5.2	4.2	4.2	4	2.9
Rare	5.2	4.2	4.2	4	2.9

Table 2.3: Run lengths for different scenarios and samplers, in millions of iterations.

the different scenarios are presented in Table 2.2.

We initialized η_1 to $\min(y_i)$, η_2 and η_3 to their prior means, and the latent variables V_i to $x_i - \min(y_i)$, both for the pilot runs used to determine tuning settings and for the main runs used to compare the relative efficiency of different MCMC schemes.

2.3.3 Results

We compared the peformance of our different MCMC schemes with the tuning settings in Table 2.2 on the three data scenarios. Run lengths were chosen to take about the same amount of time for each MCMC scheme, a total of about 43 minutes per run using MATLAB on a Linux system with an Intel Xeon X5680 3.33 GHz CPU. Table 2.3 presents the run lengths. The acceptance rates of the various updates, across different data sets, were all in the range 17% to 34%.

We first verified that the different sampling schemes give answers which agree by estimating the posterior means of the η_h, as well as the standard errors of the posterior mean estimates. For each combination of method and data set, we estimated the posterior means by taking a grand mean over the five MCMC runs, after discarding 10% of each run as burn-in. (In all cases, visual inspection of trace plots showed that the chain appears to have reached equilibrium after the first 10% of iterations.) For each η_h, the standard errors of the posterior mean estimates were estimated by computing five posterior mean estimates using each of the five samples separately (discarding 10% of each run as burn-in), computing the standard deviation of these posterior mean estimates, and dividing this standard deviation by $\sqrt{5}$. The approximate confidence intervals were obtained by taking the posterior mean estimate and adding or subtracting twice the estimated standard error of the posterior mean estimate. The results are shown in Table 2.4. There is no significant disagreement for posterior mean estimates across different methods.

Having established that the different methods we want to compare give answers which

Parameter	η_1			η_2			η_3		
Estimates	Mean	CI	std. err.	Mean	CI	std. err.	Mean	CI	std. err.
Basic	7.9292	(7.9286, 7.9298)	0.00031	7.9101	(7.9092, 7.9110)	0.00045	-1.4817	(-1.4853, -1.4781)	0.00179
Basic + Shift	7.9291	(7.9287, 7.9296)	0.00022	7.9102	(7.9094, 7.9109)	0.00038	-1.4774	(-1.4816, -1.4733)	0.00207
Basic + Range	7.9292	(7.9288, 7.9296)	0.00019	7.9102	(7.9098, 7.9107)	0.00024	-1.4778	(-1.4796, -1.4760)	0.00090
Basic + Rate	7.9292	(7.9287, 7.9296)	0.00024	7.9102	(7.9094, 7.9110)	0.00041	-1.4835	(-1.4837, -1.4833)	0.00012
Basic + All	7.9293	(7.9286, 7.9301)	0.00037	7.9100	(7.9087, 7.9112)	0.00063	-1.4834	(-1.4836, -1.4832)	0.00011

(a) Frequent arrivals.

Parameter	η_1			η_2			η_3		
Estimates	Mean	CI	std. err.	Mean	CI	std. err.	Mean	CI	std. err.
Basic	3.9612	(3.9611, 3.9613)	0.00004	2.9866	(2.9865, 2.9867)	0.00006	-1.7317	(-1.7318, -1.7317)	0.00002
Basic + Shift	3.9611	(3.9610, 3.9612)	0.00004	2.9866	(2.9865, 2.9868)	0.00007	-1.7317	(-1.7318, -1.7316)	0.00006
Basic + Range	3.9612	(3.9611, 3.9613)	0.00004	2.9865	(2.9864, 2.9866)	0.00005	-1.7318	(-1.7318, -1.7317)	0.00004
Basic + Rate	3.9611	(3.9610, 3.9613)	0.00007	2.9866	(2.9864, 2.9868)	0.00010	-1.7317	(-1.7318, -1.7316)	0.00004
Basic + All	3.9612	(3.9611, 3.9612)	0.00003	2.9865	(2.9864, 2.9866)	0.00006	-1.7316	(-1.7317, -1.7316)	0.00003

(b) Intermediate case.

Parameter	η_1			η_2			η_3		
Estimates	Mean	CI	std. err.	Mean	CI	std. err.	Mean	CI	std. err.
Basic	1.6986	(1.6878, 1.7094)	0.00538	4.2875	(4.2330, 4.3420)	0.02725	-4.4549	(-4.4551, -4.4546)	0.00013
Basic + Shift	1.7012	(1.6984, 1.7039)	0.00138	4.3098	(4.2667, 4.3529)	0.02154	-4.4549	(-4.4550, -4.4548)	0.00005
Basic + Range	1.7038	(1.7002, 1.7074)	0.00181	4.2737	(4.2632, 4.2841)	0.00520	-4.4549	(-4.4551, -4.4547)	0.00010
Basic + Rate	1.7018	(1.6953, 1.7083)	0.00325	4.3077	(4.2631, 4.3523)	0.02230	-4.4549	(-4.4550, -4.4548)	0.00006
Basic + All	1.7003	(1.6996, 1.7010)	0.00036	4.2846	(4.2751, 4.2942)	0.00477	-4.4549	(-4.4552, -4.4546)	0.00013

(c) Rare arrivals.

Table 2.4: Posterior mean estimates with approximate CI's and standard errors

agree, we next compare their efficiency by looking at the autocorrelation times, τ_h, of the Markov chains used to sample η_h. The autocorrelation time is a common MCMC performance metric that can be roughly interpreted as the number of draws one needs to make with the MCMC sampler to get the equivalent of one independent point (Neal (1993)) and is defined as

$$\tau_h = 1 + 2\sum_{k=1}^{\infty} \rho_{h,k} \tag{2.25}$$

where $\rho_{h,k}$ is the autocorrelation at lag k of the chain used to sample η_h.

For each combination of method and data set, for each η_h, we estimate τ_h by

$$\hat{\tau}_h = 1 + 2\sum_{k=1}^{K_h} \hat{\rho}_{h,k} \tag{2.26}$$

where the truncation point K_h is such that for $k > K_h$, the estimate $\hat{\rho}_{h,k}$ is not appreciably different from 0.

Scenario	Frequent			Intermediate			Rare			Time (ms)	Frequent			Intermediate			Rare		
Parameter	η_1	η_2	η_3	η_1	η_2	η_3	η_1	η_2	η_3	Freq./Inter./Rare	η_1	η_2	η_3	η_1	η_2	η_3	η_1	η_2	η_3
Basic	99	98	7800	5.4	6.1	3.2	1400	4400	5.6	0.13/0.50/0.50	13	13	1000	2.7	3	1.6	700	2200	2.8
Basic + Shift	46	75	7400	4.4	5.8	3.2	130	2100	4.1	0.24/0.61/0.61	11	18	1800	2.7	3.5	2.0	79	1300	2.5
Basic + Range	95	86	5200	5.2	5.3	3.2	380	73	4.6	0.24/0.62/0.62	23	21	1200	3.2	3.3	2.0	240	45	2.9
Basic + Rate	95	94	13	5.4	6.0	3.2	1100	2800	4.5	0.27/0.65/0.65	26	25	3.5	3.5	3.9	2.1	720	1800	2.9
Basic + All	36	55	11	4.2	5.0	3.2	13	40	4.2	0.51/0.89/0.89	18	28	5.6	3.7	4.5	2.8	12	36	3.7

Table 2.5: Estimates of autocorrelation times for different methods. Unadjusted autocorrelation times are on the left, autocorrelation times multiplied by the average time per iteration are on the right.

To estimate $\rho_{h,k}$, we use estimates of the lag k autocovariances $\gamma_{h,k}$ for $k = 0, \ldots, K_h$. These are obtained as follows. For each of the five samples, indexed by $s = 1, \ldots, 5$ and drawn using some method, we first compute an autocovariance estimate

$$\hat{\gamma}^s_{h,k} \quad = \quad \frac{1}{M} \sum_{m=1}^{M-k} (\eta_h^{[m,s]} - \overline{\eta_h})(\eta_h^{[m+k,s]} - \overline{\eta_h}) \tag{2.27}$$

Here M the length of the sample (after discarding 10% of the run as burn-in) and $\eta_h^{[m,s]}$ the m-th value of η_h in the s-th sample. We take $\overline{\eta_h}$ to be the grand mean of η_h over all five samples (each of these samples has length M). We do this because it allows us to detect if the Markov chain explores different regions of the parameter and latent variable space for different random number generator seeds. (When the mean from one or more of the five samples differs substantially from the grand mean, autocovariance estimates will be much higher.)

We then estimate $\gamma_{h,k}$ by averaging autocovariance estimates from the five samples

$$\hat{\gamma}_{h,k} \quad = \quad \frac{1}{5} \sum_{s=1}^{5} \hat{\gamma}^s_{h,k} \tag{2.28}$$

and we estimate $\rho_{h,k}$ for $k = 1, \ldots, K_h$ with $\hat{\gamma}_{h,k}/\hat{\gamma}_{h,0}$. (In practice, for long runs, it is much more efficient to use the fast Fourier transform (FFT) to compute the autocovariance estimates.) Table 2.5 shows the estimated autocorrelation times for different sampling schemes. To compare the methods fairly, we multiply each autocorrelation time by the average time per iteration.

From Table 2.5, we see that using just the shift, or just a scale update (either a range or a rate scale update) improves performance for sampling parameters that are changed by one of these updates. For a scheme which uses all updates, the greatest improvement in performance for sampling η_3 is in the case of frequent arrivals (efficiency gain of 179 times), while perfomance improvement when sampling η_1 and η_2 is greatest in the case

of rare arrivals (efficiency gains of 58 and 61 times). In other cases, performance neither increases nor decreases significantly. These results are in approximate agreement with the estimates of the standard errors of the posterior mean estimates in Table 2.4.

In an additional run (not used to compute the autocorrelation times shown here) of the basic plus rate scheme, for the rare arrivals scenario, we found that the sampler got stuck for a while in a region of the parameter space. The basic and basic + shift methods (when initialized to a state from this "stuck" region) also stayed in this "stuck" region for a while before visting other regions. The basic plus range and basic plus all methods, when initialized with the stuck state, quickly returned to sampling other regions. So, autocorrelation time estimates for the rare arrivals scenario, for methods other than basic plus all and basic plus rate, probably underestimate actual autocorrelation times.

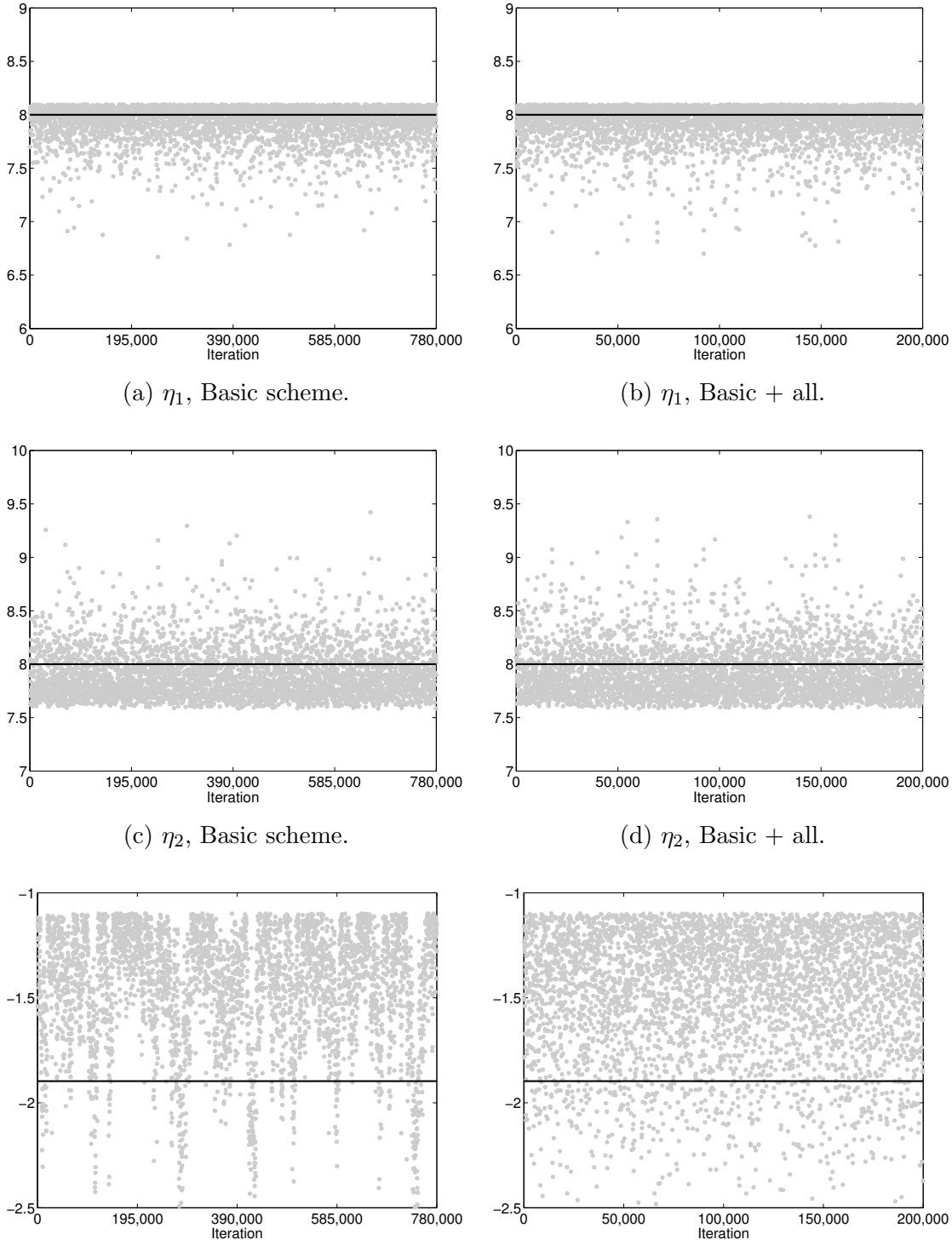
We illustrate the performance of the samplers with trace plots in Figures 2.2, 2.3, and 2.4. To produce these figures, we ran the samplers for an equal amount of computation time and thinned each run to $4,000$ points. The black line on each plot is the true parameter value.

In the case of frequent arrivals, the marginal posterior of η_3 is diffuse but concentrated given all v_i and the data, so simultaneously updating all v_i and η_3 makes sampling more efficient. In the case of rare arrivals, the marginal posteriors of both η_1 and η_2 are diffuse, while they are concentrated given all v_i and the data. Simultaneously updating all v_i and either η_1 or η_2 then leads to a noticeable gain in efficiency. In the other cases, the data is more informative about the parameters, so additional knowledge of the latent variables does not change the concentration of the posterior by as much. As a result, sampling efficiency is not affected as significantly by changing the latent variables and the parameters simultaneously.

2.4 Conclusion

In this chapter, we have shown how Bayesian inference with MCMC can be performed for the M/G/1 queueing model. As mentioned earlier, Fearnhead and Prangle (2010) used ABC for inference in the M/G/1 queueing model. Fearnhead and Prangle (2010) also used ABC for inference in the Ricker model of population dynamics, in which we assume that a population process is observed with Poisson noise.

The basis of all ABC methods is the ability to simulate observations from a given stochastic model. The simulation process for a set of observations is always driven by a latent process of sampling and transforming certain random variables. The distributions of these random variables then determine the distribution of the final observed quantities.



(a) η_1, Basic scheme.

(b) η_1, Basic + all.

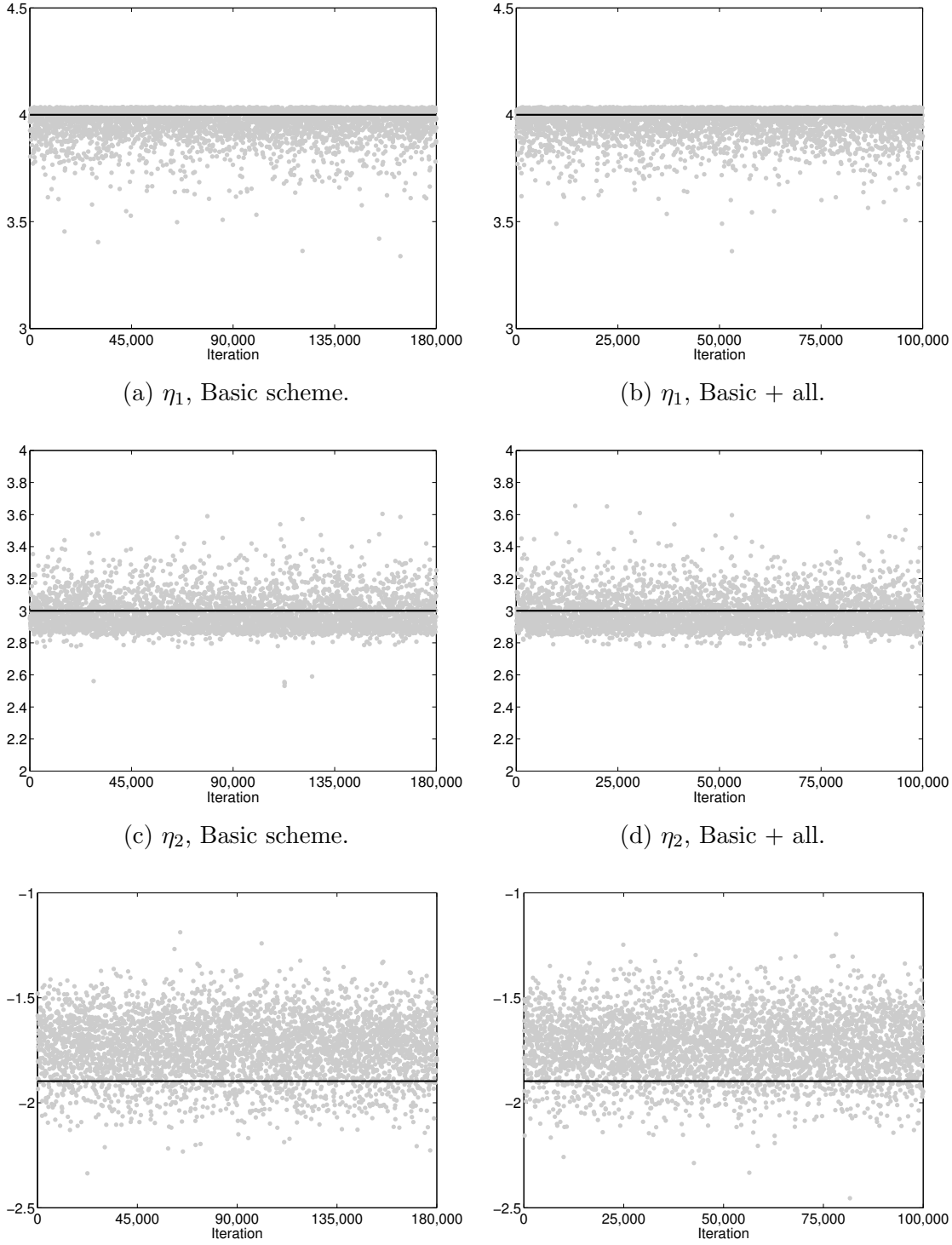
(c) η_2, Basic scheme.

(d) η_2, Basic + all.

(e) η_3, Basic scheme.

(f) η_3, Basic + all.

Figure 2.2: Comparison of performance for frequent arrivals.



(a) η_1, Basic scheme.

(b) η_1, Basic + all.

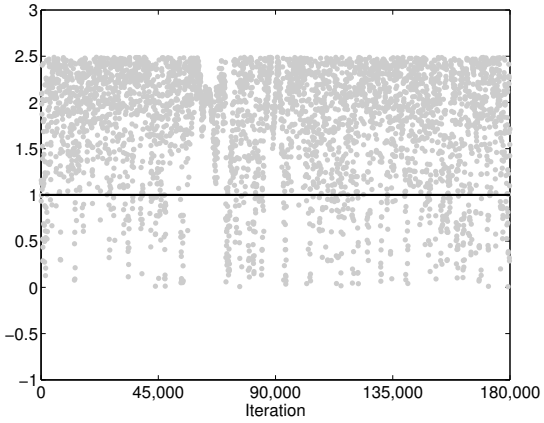
(c) η_2, Basic scheme.

(d) η_2, Basic + all.

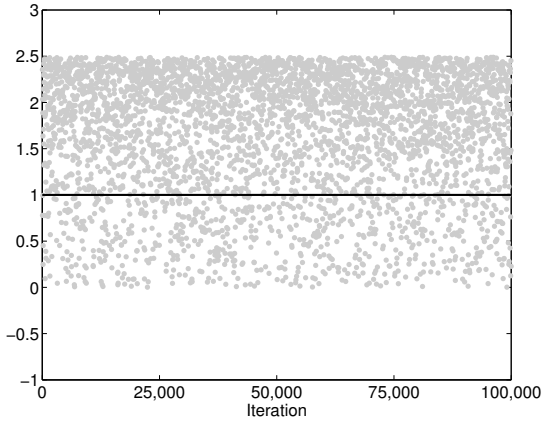
(e) η_3, Basic scheme.

(f) η_3, Basic + all.

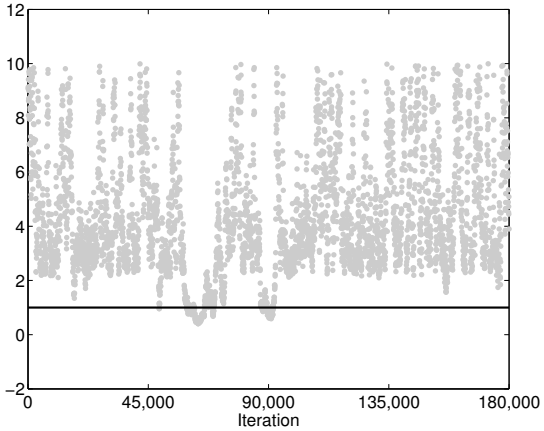
Figure 2.3: Comparison of performance for intermediate case.



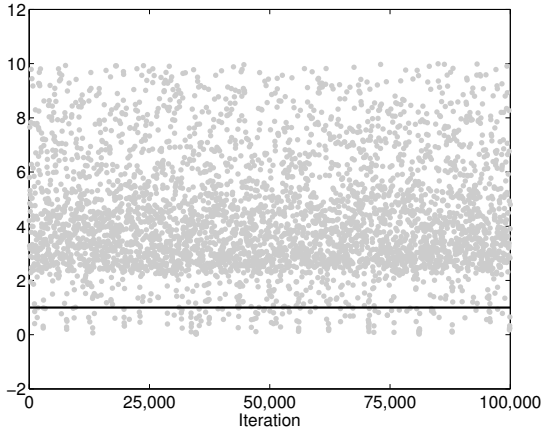
(a) η_1, Basic scheme.



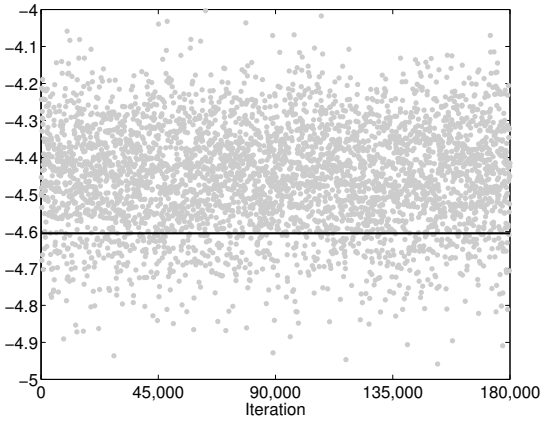
(b) η_1, Basic + all.



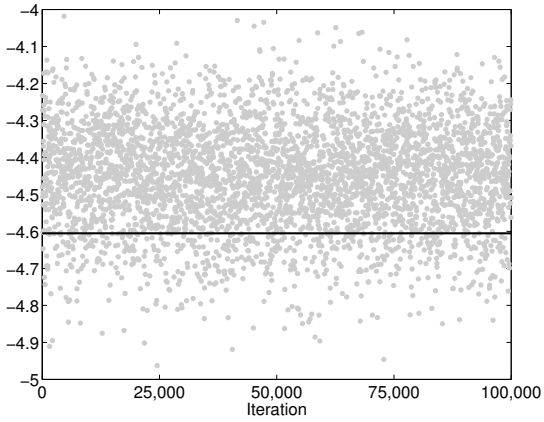
(c) η_2, Basic scheme.



(d) η_2, Basic + all.



(e) η_3, Basic scheme.



(f) η_3, Basic + all.

Figure 2.4: Comparison of performance for rare arrivals.

If we consider the latent variables which drive the simulation process jointly with the observations, then we can think of the observations as coming from a model with some latent structrure. These latent variables and the observed data will sometimes have a tractable joint density, which makes doing Bayesian inference with MCMC possible, at least in principle.

In the next chapter, we use the same approach as in this chapter to do Bayesian inference for the Ricker model, i.e. including additional latent variables in the MCMC state and sampling for them as well as model parameters. We compared a basic MCMC scheme with several "ensemble MCMC" schemes for Bayesian inference in the Ricker model, and showed that using the ensemble schemes leads to a significant improvement in efficiency when compared to the basic MCMC scheme. Like for the M/G/1 queueing model, we have shown that Bayesian inference with MCMC is possible for the Ricker model, but requires more sophisticated MCMC methods for sampling to be efficient.

It would be interesting to apply alternative inference methods for models with a time series structure to the M/G/1 queue, for example Particle Filters and Particle Markov Chain Monte Carlo (PMCMC) (Andrieu, Doucet, and Holenstein (2010)).

As mentioned earlier, it should be possible to extend the MCMC method in this chapter to service time distributions other than the Uniform one used in this chapter. The most direct extension would be to consider location-scale service time distributions. Besides the simple Metropolis updates, we can then do additional updates for the location parameter (or some 1-to-1 function of it) using a shift update and additional updates for the scale parameter (or some 1-to-1 function of it) using a range scale update. Computation would not be impacted so long as (2.14) can be written in terms of low-dimensional sufficient statistics.

Chapter 3

Ensemble MCMC for State Space Models

Non-linear state space models are a widely-used class of models for biological, economic, and physical processes. Fitting these models to observed data is a difficult inference problem that has no straightforward solution. We take a Bayesian approach to the inference of unknown parameters of a non-linear state space model. One way of performing Bayesian computations for such models is via Markov Chain Monte Carlo (MCMC) sampling methods for the latent (hidden) variables and model parameters. Using the ensemble technique of Neal (2010) and the embedded HMM technique of Neal (2003), we introduce a new Markov Chain Monte Carlo method for non-linear state space models. The key idea is to perform parameter updates conditional on an enormously large ensemble of latent sequences, as opposed to a single sequence, as with existing methods. We look at the performance of this ensemble method when doing Bayesian inference in the Ricker model of population dynamics. We show that for this problem, the ensemble method is vastly more efficient than a simple Metropolis method, as well as 1.9 to 12.0 times more efficient than a single-sequence embedded HMM method, when all methods are tuned appropriately. We also introduce a way of speeding up the ensemble method by performing partial backward passes to discard poor proposals at low computational cost, resulting in a final efficiency gain of 3.4 to 20.4 times over the single-sequence method.[1]

[1]Material in this chapter was first presented in Shestopaloff and Neal (2013a).

3.1 Introduction

Consider an observed sequence $z_1, \ldots, z_N$. In a state space model for $Z_1, \ldots, Z_N$, the distribution of the Z_i's is defined using a latent (hidden) Markov process $X_1, \ldots, X_N$. We can describe such a model in terms of a distribution for the first hidden state, $p(x_1)$, transition probabilities between hidden states, $p(x_i|x_{i-1})$, and emission probabilities, $p(z_i|x_i)$, with these distributions dependent on some unknown parameters θ.

While the state space model framework is very general, only two state space models, Hidden Markov Models (HMM's) and linear Gaussian models have efficient, exact inference algorithms. The forward-backward algorithm for HMM's and the Kalman filter for linear Gaussian models allow us to perform efficient inference of the latent process, which in turn allows us to perform efficient parameter inference, using an algorithm such as Expectation-Maximizaton for maximum likelihood inference, or various MCMC methods for Bayesian inference. No such exact and efficient algorithms exist for models with a continuous state space with non-linear state dynamics, non-Gaussian transition distributions, or non-Gaussian emission distributions, such as the Ricker model we consider later in this chapter.

In cases where we can write down a likelihood function for the model parameters conditional on latent and observed variables, it is possible to perform Bayesian inference for the parameters and the latent variables by making use of sampling methods such as MCMC. For example, one can perform inference for the latent variables and the parameters by alternately updating them according to their joint posterior.

Sampling of the latent state sequence $x = (x_1, \ldots, x_N)$ is difficult for state space models when the state space process has strong dependencies — for example, when the transitions between states are nearly deterministic. To see why, suppose we sample from $\pi(x_1, \ldots, x_N|z_1, \ldots, z_N)$ using Gibbs sampling, which samples the latent variables one at a time, conditional on values of other latent variables, the observed data, and the model parameters. In the presence of strong dependencies within the state sequence, the conditional distribution of a latent variable will be highly concentrated, and we will only be able to change it slightly at each variable update, even when the marginal distribution of this latent variable, given $z_1, \ldots, z_N$, is relatively diffuse. Consequently, exploration of the space of latent sequences will be slow.

The embedded HMM method of (Neal (2003), Neal, et al. (2004)) addresses this problem by updating the entire latent sequence at once. The idea is to temporarily reduce the state space of the model, which may be countably infinite or continuous, to a finite collection of randomly generated "pool" states at each time point. If the

transitions between states are Markov, this reduced model is an HMM, for which we can use the forward-backward algorithm to efficiently sample a sequence with values in the pools at each time point. Pool states are chosen from a distribution that assigns positive probability to all possible state values, allowing us to explore the entire space of latent sequences in accordance with their exact distribution. Neal (2003) showed that when there are strong dependencies in the state sequence, the embedded HMM method performs better than conventional Metropolis methods at sampling latent state sequences.

In this chapter, we first look at an MCMC method which combines embedded HMM updates of the hidden state sequence with random-walk Metropolis updates of the parameters. We call this method the 'single-sequence' method. We next reformulate the embedded HMM method as an ensemble MCMC method. Ensemble MCMC allows multiple candidate points to be considered simultaneously when a proposal is made. This allows us to consider an extension of the embedded HMM method for inference of the model parameters when they are unknown. We refer to this extension as the ensemble embedded HMM method. We then introduce and describe a "staged" method, which makes ensemble MCMC more efficient by rejecting poor proposals at low computational cost after looking at a part of the observed sequence. We use the single-sequence, ensemble, and ensemble with staging methods to perform Bayesian inference in the Ricker model of population dynamics, comparing the performance of these new methods to each other, and to a simple Metropolis sampler.

3.2 Ensemble MCMC

We first describe Ensemble MCMC, introduced by Neal (2010) as a general MCMC method, before describing its application to inference in state space models. Ensemble MCMC has some relationships with the multiple-try Metropolis sampler of Liu, Liang and Wong (2000) and the multiset sampler of Leman, Chen and Lavine (2009). For a detailed discussion, see Neal (2010).

Ensemble MCMC is based on the framework of MCMC using a temporary mapping. Suppose we want to sample from a distribution π on $\mathcal{X}$. This can be done using a Markov chain with transition probablity from x to x' given by $T(x'|x)$, for which π is an invariant distribution — that is, T must satisfy $\int \pi(x)T(x'|x)dx = \pi(x')$. The temporary mapping strategy defines T as a composition of three stochastic mappings. The current state x is stochastically mapped to a state $y \in \mathcal{Y}$ using the mapping $\hat{T}(y|x)$. Here, the space $\mathcal{Y}$ need not be the same as the space $\mathcal{X}$. The state y is then updated to y' using the

mapping $\bar{T}(y'|y)$, and finally a state x' is obtained using the mapping $\check{T}(x'|y')$. In this approach, we may choose whatever mappings we want, so long as the overall transition T leaves π invariant. In particular, if ρ is a density for y, T will leave π invariant if the following conditions hold.

$$\int \pi(x)\hat{T}(y|x)dx = \rho(y) \tag{3.1}$$

$$\int \rho(y)\bar{T}(y'|y)dy = \rho(y') \tag{3.2}$$

$$\int \rho(y')\check{T}(x'|y')dy' = \pi(x') \tag{3.3}$$

In the ensemble method, we take $\mathcal{Y} = \mathcal{X}^K$, with $y = (x^{(1)}, \ldots, x^{(K)})$ referred to as an "ensemble", with K being the number of ensemble elements. The three mappings are then constructed as follows. Consider an "ensemble base measure" over ensembles $(x^{(1)}, \ldots, x^{(K)})$ with density $\zeta(x^{(1)}, \ldots, x^{(K)})$, and with marginal densities $\zeta_k(x^{(k)})$ for each of the $k = 1, \ldots, K$ ensemble elements. We define $\hat{T}$ as

$$\hat{T}(x^{(1)}, \ldots, x^{(K)}|x) = \frac{1}{K}\sum_{k=1}^{K} \zeta_{-k|k}(x^{(-k)}|x)\delta_x(x^{(k)}) \tag{3.4}$$

Here, δ_x is a distribution that places a point mass at x, $x^{(-k)}$ is all of $x^{(1)}, \ldots, x^{(K)}$ except $x^{(k)}$, and $\zeta_{-k|k}(x^{(-k)}|x^{(k)}) = \zeta(x^{(1)}, \ldots, x^{(K)})/\zeta_k(x^{(k)})$ is the conditional density of all ensemble elements except the k-th, given the value $x^{(k)}$ for the k-th.

This mapping can be interpreted as follows. First, we select an integer k, from a uniform distribution on $\{1, \ldots, K\}$. Then, we set the ensemble element $x^{(k)}$ to x, the current state. Finally, we generate the remaining elements of the ensemble using the conditional density $\zeta_{-k|k}$.

The ensemble density ρ is determined by π and $\hat{T}$, and is given explicitly as

$$\begin{aligned}\rho(x^{(1)}, \ldots, x^{(K)}) &= \int \pi(x)\hat{T}(x^{(1)}, \ldots, x^{(K)}|x)dx \\ &= \zeta(x^{(1)}, \ldots, x^{(K)})\frac{1}{K}\sum_{k=1}^{K}\frac{\pi(x^{(k)})}{\zeta_k(x^{(k)})}\end{aligned} \tag{3.5}$$

$\bar{T}$ can be any update (or sequence of updates) that leaves ρ invariant. For example, $\bar{T}$ could be a Metropolis update for y, with a proposal drawn from some symmetrical proposal density. Finally, $\check{T}$ maps from y' to x' by randomly setting x' to $x^{(k)}$ with k chosen from $\{1, \ldots, K\}$ with probabilities proportional to $\pi(x^{(k)})/\zeta_k(x^{(k)})$.

The mappings described above satisfy the necessary properties to make them a valid update, in the sense of preserving the stationary distribution π. The proof can be found in Neal (2010).

3.3 Embedded HMM MCMC as an ensemble MCMC method

The embedded HMM method briefly described in the introduction was not initially introduced as an ensemble MCMC method, but it can be reformulated as one. We assume here that we are interested in sampling from the posterior distribution of the state sequences, $\pi(x_1, \ldots, x_N | z_1, \ldots, z_N)$, when the parameters of the model are known. Suppose the current state sequence is $x = (x_1, \ldots, x_N)$. We want to update this state sequence in a way that leaves π invariant.

The first step of the embedded HMM method is to temporarily reduce the state space to a finite number of possible states at each time, turning our model into an HMM. This is done by, for each time i, generating a set of L "pool" states, $P_i = \{x_i^{[1]}, \ldots, x_i^{[L]}\}$, as follows. We first set the pool state $x_i^{[1]}$ to x_i, the value of the current state sequence at time i. The remaining $L-1$ pool states $x_i^{[l]}$, for $l > 1$ are generated by sampling independently from some pool distribution with density κ_i. The collections of pool states at different times are selected independently of each other. The total number of sequences we can then construct using these pool states, by choosing one state from the pool at each time, is $K = L^N$.

The second step of the embedded HMM method is to choose a state sequence composed of pool states, with the probability of such a state sequence, x, being proportional to

$$q(x|z_1, \ldots, z_N) \;\propto\; p(x_1) \prod_{i=2}^{N} p(x_i|x_{i-1}) \prod_{i=1}^{N} \left[\frac{p(z_i|x_i)}{\kappa_i(x_i)} \right] \tag{3.6}$$

We can define $\gamma(z_i|x_i) = p(z_i|x_i)/\kappa_i(x_i)$, and rewrite (3.6) as

$$q(x|z_1, \ldots, z_N) \;\propto\; p(x_1) \prod_{i=2}^{N} p(x_i|x_{i-1}) \prod_{i=1}^{N} \gamma(z_i|x_i) \tag{3.7}$$

We now note that the distribution (3.7) takes the same form as the distribution over hidden state sequences for an HMM in which each $x_i \in P_i$ — the initial state distribution is proportional to $p(x_1)$, the transition probabilities are proportional to $p(x_i|x_{i-1})$, and the $\gamma(z_i|x_i)$ have the same role as emission probabilities. This allows us to use the well-

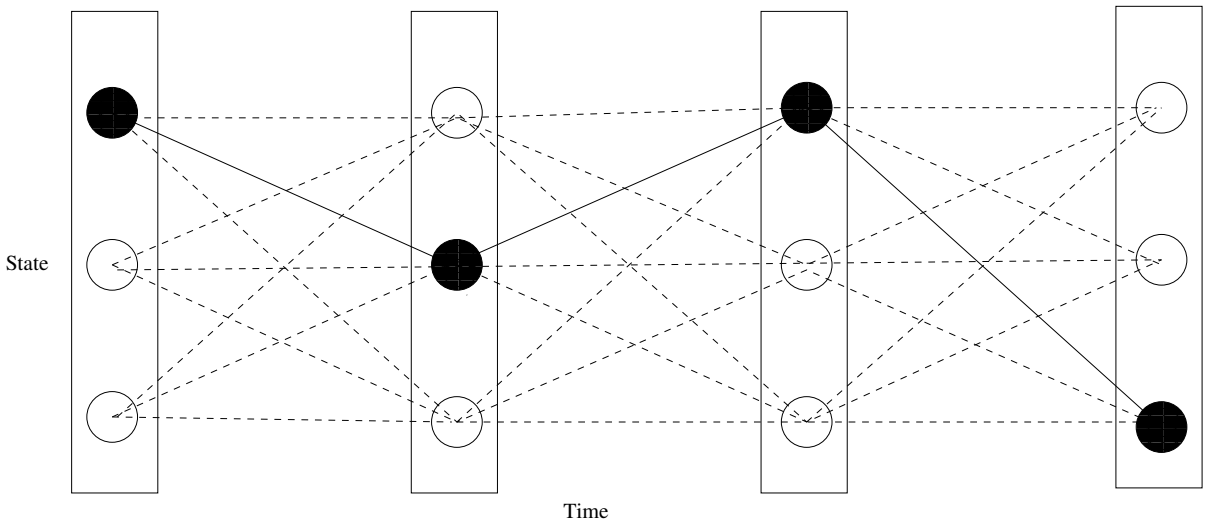


Figure 3.1: An illustration of the ensemble of latent sequences for $N = 4$ and $L = 3$.

known forward-backward algorithms for HMM's (reviewed by Scott (2002)) to efficiently sample hidden state sequences composed of pool states. To sample a sequence with the embedded HMM method, we first compute the "forward" probabilities. Then, using a stochastic "backwards" pass, we select a state sequence composed of pool states. (We can alternately compute backward probabilities and then do a stochastic forward pass). We emphasize that having an efficient algorithm to sample state sequences is crucial for the embedded HMM method. The number of possible sequences we can compose from the pool states, L^N, can be very large, and so naive sampling methods would be impractical. An illustration of the ensemble of state sequences passing through the chosen pool states is shown in Figure 3.1, with the current sequence denoted by the solid line.

In detail, for $x_i \in P_i$, the forward probabilities $\alpha_i(x_i)$ are computed using a recursion that goes forward in time, starting from $i = 1$. We start by computing $\alpha_1(x_1) = p(x_1)\gamma(z_1|x_1)$. Then, for $1 < i \leq N$, the forward probabilities $\alpha_i(x_i)$ are given by the recursion

$$\alpha_i(x_i) = \gamma(z_i|x_i) \sum_{l=1}^{L} p(x_i|x_{i-1}^{[l]})\alpha_{i-1}(x_{i-1}^{[l]}), \quad \text{for } x \in P_i \tag{3.8}$$

The stochastic backwards recursion samples a state sequence, one state at a time, beginning with the state at time N. First, we sample x_N from the pool P_N with probabilities proportional to $\alpha_N(x_N)$. Then, going backwards in time for i from $N-1$ to 1, we sample x_i from the pool P_i with probabilities proportional to $p(x_{i+1}|x_i)\alpha_i(x_i)$, where x_{i+1} is the variable just sampled at time $i+1$. Both of these recursions are commonly implemented using logarithms of probabilities to avoid numerical underflow.

The idea that the embedded HMM method is an example of an ensemble MCMC

method was first mentioned in (Neal (2010)). In detail, the relationship is as follows. The step of choosing the pool states can be thought of as performing a mapping $\hat{T}$ which takes a single hidden state sequence x and maps it to an ensemble of K state sequences $y = (x^{(1)}, \ldots, x^{(K)})$, with $x = x^{(k)}$ for some k chosen uniformly from $\{1, \ldots, K\}$. (However, we note that in this context, the order in the ensemble does not matter.)

Since the randomly chosen pool states are independent under κ_i at each time, and across time as well, the density of an ensemble of hidden state sequences in the ensemble base measure, ζ, is defined through a product of $\kappa_i(x_i^{[l]})$'s over the pool states and over time, and is non-zero for ensembles consisting of all sequences composed from the chosen set of pool states. The corresponding marginal density of a hidden state sequence $x^{(k)}$ in the ensemble base measure is

$$\zeta_k(x^{(k)}) = \prod_{i=1}^{N} \kappa_i(x_i^{(k)}) \tag{3.9}$$

Together, ζ and ζ_k define the conditional distribution $\zeta_{-k|k}$, which is used to define $\hat{T}$. The mapping $\bar{T}$ is taken to be a null mapping that keeps the ensemble fixed at its current value, and the mapping $\check{T}$ to a single state sequence is performed by selecting a sequence $x^{(k)}$ from the ensemble with probabilities given by (3.7), in the same way as in the embedded HMM method.

3.4 The single-sequence embedded HMM MCMC method

Let us assume that the parameters θ of our model are unknown, and that we want to sample from the joint posterior distribution of state sequences $x = (x_1, \ldots, x_N)$ and parameters $\theta = (\theta_1, \ldots, \theta_P)$, with density $\pi(x, \theta | z_1, \ldots, z_N)$. One way of doing this is by alternating embedded HMM updates of the state sequence with Metropolis updates of the parameters. Doing updates in this manner makes use of an ensemble to sample state sequences more efficiently in the presence of strong dependencies. However, this method only takes into account a single hidden state sequence when updating the parameters.

The update for the sequence is identical to that in the embedded HMM method, with initial, transition and emission densities dependent on the current value of θ. In our case, we only consider simple random-walk Metropolis updates for θ, updating all of the variables simultaneously.

Evaluating the likelihood conditional on x and z, as needed for Metropolis parameter updates, is computationally inexpensive relative to updates of the state sequence, which take time proportional to L^2. It may be beneficial to perform several Metropolis parameter updates for every update of the state sequence, since this will not greatly increase the overall computational cost, and allows us to obtain samples with lower autocorrelation time.

3.5 An ensemble extension of the embedded HMM method

When performing parameter updates, we can look at all possible state sequences composed of pool states by using an ensemble $((x^{(1)}, \theta), \ldots, (x^{(K)}, \theta))$ that includes a parameter value θ, the same for each element of the ensemble. The update $\bar{T}$ could change both θ and $x^{(k)}$, but in the method we will consider here, we only change θ.

To see why updating θ with an ensemble of sequences might be more efficient than updating θ given a single sequence, consider a Metropolis proposal in ensemble space, which proposes to change θ for all of the ensemble elements, from θ to θ^*. Such an update can be accepted whenever there are *some* elements $(x^{(k)}, \theta^*)$ in the proposed ensemble that make the ensemble density $\rho((x^{(1)}, \theta^*), \ldots, (x^{(K)}, \theta^*))$ relatively large. That is, it is possible to accept a move in ensemble space with a high probability as long as *some* elements of the proposed ensemble, with the new θ^*, lie in a region of high posterior density. This is at least as likely to happen as having a proposed θ^* together with a single state sequence x lie in a region of high posterior density.

Using ensembles makes sense when the ensemble density ρ can be computed in an efficient manner, in less than K times as long as it takes to compute $p(x, \theta | z_1, \ldots, z_N)$ for a single hidden state sequence x. Otherwise, one could make K independent proposals to change x, which would have approximately the same computational cost as a single ensemble update, and likely be a more efficient way to sample x. In the application here, $K = L^N$ is enormous for typical values of N when $L \geq 2$, while computation of ρ takes time proportional only to NL^2.

In detail, to compute the ensemble density, we need to sum $q(x^{(k)}, \theta | z_1, \ldots, z_N)$ over all ensemble elements $(x^{(k)}, \theta)$, that is, over all hidden sequences which are composed of the pool states at each time. The forward algorithm described above makes it possible to compute the ensemble density efficiently by summing the probabilities at the end of

the forward recursion $\alpha_N(x_N)$ over all $x_N \in P_N$. That is

$$\rho((x^{(1)},\theta),\ldots,(x^{(K)},\theta)) \;\propto\; \pi(\theta)\sum_{k=1}^{K} q(x^{(k)},\theta|z_1,\ldots,z_N) = \pi(\theta)\sum_{l=1}^{L}\alpha_N(x_N^{[l]}) \qquad (3.10)$$

where $\pi(\theta)$ is the prior density of θ.

The ensemble extension of the embedded HMM method can be thought of as using an approximation to the marginal posterior of θ when updating θ, since summing the posterior density over a large collection of hidden sequences approximates integrating over all such sequences. Larger updates of θ may be possible with an ensemble because the marginal posterior of θ, given the data, is more diffuse than the conditional distribution of θ given a fixed state sequence and the data. Note that since the ensemble of sequences is periodically changed, when new pool states are chosen, the ensemble method is a proper MCMC method that converges to the exact joint posterior, even though the set of sequences using pool states is restricted at each MCMC iteration.

The ensemble method is more computationally expensive than the single-sequence method — it requires two forward passes to evaluate the ensemble density for two values of θ and one backward pass to sample a hidden state sequence, whereas the single-sequence method requires only a single forward pass to compute the probabilities for every sequence in our ensemble, and a single backward pass to select a sequence.

Some of this additional computational cost can be offset by reusing the same pool states to do multiple updates of θ. Once we have chosen a collection of pool states, and performed a forward pass to compute the ensemble density at the current value of θ, we can remember it. Proposing an update of θ to θ^* requires us to compute the ensemble density at θ^* using a forward pass. Now, if this proposal is rejected, we can reuse the stored value of the ensemble density at θ when we make another proposal using the same collection of pool states. If this proposal is accepted, we can remember the ensemble density at the accepted value. Keeping the pool fixed, and saving the current value of the ensemble density therefore allows us to perform M ensemble updates with $M+1$ forward passes, as opposed to $2M$ if we used a new pool for each update.

With a large number of pool states, reusing the pool states for two or more updates has only a small impact on performance, since with any large collection of pool states we essentially integrate over the state space. However, pool states must still be updated occasionally, to ensure that the method samples from the exact joint posterior.

3.6 Staged ensemble MCMC sampling

Having to compute the ensemble density given the entire observed sequence for every proposal, even those that are obviously poor, is a source of inefficiency in the ensemble method. If poor proposals can be eliminated at a low computational cost, the ensemble method could be made more efficient. We could then afford to make our proposals larger, accepting occasional large proposals while rejecting others at little cost. This is an example of a "delayed acceptance" technique (Christen and Fox (2005)).

We propose to do this by performing "staged" updates. First, we choose a part of the observed sequence that we believe is representative of the whole sequence. Then, we propose to update θ to a θ^* found using an ensemble update that only uses the part of the sequence we have chosen. If the proposal found by this "first stage" update is accepted, we perform a "second stage" ensemble update given the entire sequence, with θ^* as the proposal. If the proposal at the first stage is rejected, we do not perform a second stage update, and add the current value of θ to our sequence of sampled values. This can be viewed as a second stage update where the proposal is the current state — to do such an update, no computations need be performed.

Suppose that ρ_1 is the ensemble density given the chosen part of the observed sequence, and $q(\theta^*|\theta)$ is the proposal density for constructing the first stage update. Then the acceptance probability for the first stage update is given by

$$\min\left(1, \frac{\rho_1((x^{(1)},\theta^*),\ldots,(x^{(K)},\theta^*))q(\theta|\theta^*)}{\rho_1((x^{(1)},\theta),\ldots,(x^{(K)},\theta))q(\theta^*|\theta)}\right) \tag{3.11}$$

If ρ is the ensemble density given the entire sequence, the acceptance probability for the second stage update is given by

$$\min\left(1, \frac{\rho((x^{(1)},\theta^*),\ldots,(x^{(K)},\theta^*))\min\left(1, \frac{\rho_1((x^{(1)},\theta),\ldots,(x^{(K)},\theta))q(\theta^*|\theta)}{\rho_1((x^{(1)},\theta^*),\ldots,(x^{(K)},\theta^*))q(\theta|\theta^*)}\right)}{\rho((x^{(1)},\theta),\ldots,(x^{(K)},\theta))\min\left(1, \frac{\rho_1((x^{(1)},\theta^*),\ldots,(x^{(K)},\theta^*))q(\theta|\theta^*)}{\rho_1((x^{(1)},\theta),\ldots,(x^{(K)},\theta))q(\theta^*|\theta)}\right)}\right) \tag{3.12}$$

Regardless of whether $\rho_1((x^{(1)},\theta^*),\ldots,(x^{(K)},\theta^*))q(\theta|\theta^*) < \rho_1((x^{(1)},\theta),\ldots,(x^{(K)},\theta))q(\theta^*|\theta)$ or vice versa, the above ratio simplifies to

$$\min\left(1, \frac{\rho((x^{(1)},\theta^*),\ldots,(x^{(K)},\theta^*))\rho_1((x^{(1)},\theta),\ldots,(x^{(K)},\theta))q(\theta|\theta^*)}{\rho((x^{(1)},\theta),\ldots,(x^{(K)},\theta))\rho_1((x^{(1)},\theta^*),\ldots,(x^{(K)},\theta^*))q(\theta^*|\theta)}\right) \tag{3.13}$$

Choosing a part of the observed sequence for the first stage update can be aided by looking

at the acceptance rates at the first and second stages. We need the moves accepted at the first stage to also be accepted at a sufficiently high rate at the second stage, but we want the acceptance rate at the first stage to be low so the method will have an advantage over the ensemble method in terms of computational efficiency. We can also look at the 'false negatives' for diagnostic purposes, that is, how many proposals rejected at the first step would have been accepted had we looked at the entire sequence when deciding whether to accept.

We are free to select any portion of the observed sequence to use for the first stage. We will look here at using a partial sequence for the first stage consisting of observations starting at n_1 and going until the end, at time N. This is appropriate for our example later in the paper, where we only have observations past a certain point in time.

For this scenario, to perform a first stage update, we need to perform a backward pass. As we perform a backward pass to do the first stage proposal, we save the vector of "backward" probabilities. Then, if the first stage update is accepted, we start the recursion for the full sequence using these saved backward probabilities, and compute the ensemble density given the entire sequence, avoiding recomputation of backward probabilities for the portion of the sequence used for the first stage.

To compute the backward probabilities $\beta_i(x_i)$, we perform a backward recursion, starting at time N. We first set $\beta_N(x_N)$ to 1 for all $x_N \in P_N$. We then compute, for $n_1 \leq i < N$

$$\beta_i(x_i) = \sum_{l=1}^{L} p(x_{i+1}^{[l]}|x_i)\beta_{i+1}(x_{i+1}^{[l]})\gamma(z_{i+1}|x_{i+1}^{[l]}) \tag{3.14}$$

We compute the first stage ensemble density ρ_1 as follows

$$\rho_1((x^{(1)},\theta),\ldots,(x^{(K)},\theta)) = \pi(\theta)\sum_{l=1}^{L} p(x_{n_1}^{[l]})p(y_{n_1}|x_{n_1}^{[l]})\beta_{n_1}(x_{n_1}^{[l]}) \tag{3.15}$$

We do not know $p(x_{n_1})$, but we can choose a substitute for it (which affects only performance, not correctness). One possibility is a uniform distrubution over the pool states at n_1, which is what we will use in our example below.

The ensemble density for the full sequence can be obtained by performing the backward recursion up to the beginning of the sequence, and then computing

$$\rho((x^{(1)},\theta),\ldots,(x^{(K)},\theta)) = \pi(\theta)\sum_{l=1}^{L} p(x_1^{[l]})p(y_1|x_1^{[l]})\beta_1(x_1^{[l]}) \tag{3.16}$$

To see how much computation time is saved with staged updates, we measure com-

putation time in terms of the time it takes to do a backward (or equivalently, forward) pass — generally, the most computationally expensive operation in ensemble MCMC — counting a backward pass to time n_1 as a partial backward pass.

Let us suppose that the acceptance rate for stage 1 updates is a_1. An ensemble update that uses the full sequence requires us to perform two backwards passes if we update the pool at every iteration, whereas a staged ensemble update will require us to perform

$$1 + \frac{N - n_1}{N - 1} + a_1 \frac{n_1 - 1}{N - 1} \tag{3.17}$$

backward passes on average (counting a pass back only to n_1 as $(N-n_1)/(N-1)$ passes). The first term in the above formula represents the full backward pass for the initial value of θ that is always needed — either for a second stage update, or for mapping to a new set of pool states. The second term represents the partial backward pass we need to perform to complete the first stage proposal. The third term accounts for having to compute the remainder of a backwards pass if we accept the first stage proposal — hence it is weighted by the first stage acceptance rate.

We can again save computation time by updating the pool less frequently. Suppose we decide to update the pool every M iterations. Without staging, the ensemble method would require a total of $M + 1$ forward (or equivalently, backward) passes. With staged updates, the expected number of backward passes we would need to perform is

$$1 + M \frac{N - n_1}{N - 1} + M a_1 \frac{n_1 - 1}{N - 1} \tag{3.18}$$

as can be seen by generalizing the expression above to $M > 1$.

3.7 Constructing pools of states

An important aspect of these embedded HMM methods is the selection of pool states at each time, which determines the mapping $\hat{T}$ from a single state sequence to an ensemble. In this chapter, we will only consider sampling pool states independently from some distribution with density κ_i (though dependent pool states are considered in (Neal, 2003)).

One choice of κ_i is the conditional density of x_i given z_i, based on some pseudo-prior distribution η for x_i — that is, $\kappa_i(x_i) \propto \eta(x_i)p(z_i|x_i)$. This distribution approximates, to some extent, the marginal posterior of x_i given all observations. We hope that such a pool distribution will produce sequences in our ensemble which have a high posterior

density, and as a result make sampling more efficient.

To motivate how we might go about choosing η, consider the following. Suppose we sample values of θ from the model prior, and then sample hidden state sequences, given the sampled values of θ. We can then think of η as a distribution that is consistent with this distribution of hidden state sequences, which is in turn consistent with the model prior. In this chapter, we choose η heuristically, setting it to what we believe to be reasonable, and not in violation of the model prior.

Note that the choice of η affects only sampling efficiency, not correctness (as long as it is non-zero for all possible x_i). However, when we choose pool states in the ensemble method, we cannot use current values of θ, since this would make an ensemble update non-reversible. This restriction does not apply to the single-sequence method since in this case $\bar{T}$ is null.

3.8 Performance comparisons on a population dynamics model

We test the performance of the proposed methods on the Ricker population dynamics model, described by Wood (2003). This model assumes that a population of size N_i (modeled as a real number) evolves as $N_{i+1} = rN_i \exp(-N_i + e_i)$, with e_i independent with a Normal$(0, \sigma^2)$ distribution, with $N_0 = 1$. We don't observe this process directly, but rather we observe Y_i's whose distribution is Poisson(ϕN_i). The goal is to infer $\theta = (r, \sigma, \phi)$. This is considered to be a fairly complex inference scenario, as evidenced by the application of recently developed inference methods such as Approximate Bayesian Computation (ABC) to this model. (See Fearnhead, Prangle (2012) for more on the ABC approach.) This model can be viewed as a non-linear state space model, with N_i as our state variable.

MCMC inference in this model can be inefficient for two reasons. First, when the value of σ^2 in the current MCMC iteration is small, consecutive N_i's are highly dependent, so the distribution of each N_i, conditional on the remaining N_i's and the data, is highly concentrated, making it hard to efficiently sample state sequences one state at a time. An MCMC method based on embedding an HMM into the state space, either the single-sequence method or the ensemble method, can potentially make state sequence sampling more efficient, by sampling whole sequences at once. The second reason is that the distribution of θ, given a single sequence of N_i's and the data, can be concentrated as well, so we may not be able to efficiently explore the posterior distribution of θ by

alternately sampling θ and the N_i's. By considering an ensemble of sequences, we may be able to propose and accept larger changes to θ. This is because the posterior distribution of θ summed over an ensemble of state sequences is less concentrated than it is given a single sequence.

To test our MCMC methods, we consider a scenario similar to those considered by Wood (2010) and Fearnhead and Prangle (2012). The parameters of the Ricker model we use are $r = \exp(3.8), \sigma = 0.15, \phi = 2$. We generated 100 points from the Ricker model, with y_i only observed from time 51 on, mimicking a situation where we do not observe a population right from its inception.

We put a Uniform$(0, 10)$ prior on $\log(r)$, a Uniform$(0, 100)$ prior on ϕ, and we put a Uniform$[\log(0.1), 0]$ prior on $\log(\sigma)$. Instead of N_i, we use $M_i = \log(\phi N_i)$ as our state variables, since we found that doing this makes MCMC sampling more efficient. With these state variables, our model can be written as

$$M_1 \sim N(\log(r) + \log(\phi) - 1, \sigma^2) \tag{3.19}$$

$$M_i | M_{i-1} \sim N(\log(r) + M_{i-1} - \exp(M_{i-1})/\phi, \sigma^2), \quad i = 2, \ldots, 100 \tag{3.20}$$

$$Y_i | M_i \sim \text{Poisson}(\exp(M_i)), \quad i = 51, \ldots, 100 \tag{3.21}$$

Furthermore, the MCMC state uses the logarithms of the parameters, $(\log(r), \log(\sigma), \log(\phi))$.

For parameter updates in the MCMC methods compared below, we used independent normal proposals for each parameter, centered at the current parameter values, proposing updates to all parameters at once. To choose appropriate proposal standard deviations, we did a number of runs of the single sequence and the ensemble methods, and used these trial runs to estimate the marginal standard deviations of the logarithm of each parameter. We got standard deviation estimates of 0.14 for $\log(r)$, 0.36 for $\log(\sigma)$, and 0.065 for $\log(\phi)$. We then scaled each estimated marginal standard deviation by the same factor, and used the scaled estimates as the proposal standard deviations for the corresponding parameters. The maximum scaling we used was 2, since beyond this our proposals would often lie outside the high probability region of the marginal posterior density.

We first tried a simple Metropolis sampler on this problem. This sampler updates the latent states one at a time, using Metropolis updates to sample from the full conditional

density of each state M_i, given by

$$p(m_1|y_{51},\ldots,y_{100},m_{-1}) \propto p(m_1)p(m_2|m_1) \tag{3.22}$$

$$p(m_i|y_{51},\ldots,y_{100},m_{-i}) \propto p(m_i|m_{i-1})p(m_{i+1}|m_i), \quad 2 \le i \le 50 \tag{3.23}$$

$$p(m_i|y_{51},\ldots,y_{100},m_{-i}) \propto p(m_i|m_{i-1})p(y_i|m_i)p(m_{i+1}|m_i), \quad 51 \le i \le 99 \tag{3.24}$$

$$p(m_{100}|y_{51},\ldots,y_{100},m_{-100}) \propto p(m_{100}|m_{99})p(y_{100}|m_{100}) \tag{3.25}$$

We started the Metropolis sampler with parameters set to prior means, and the hidden states to randomly chosen values from the pool distributions we used for the embedded HMM methods below. After we perform a pass over the latent variables, and update each one in turn, we perform a Metropolis update of the parameters, using a scaling of 0.25 for the proposal density.

The latent states are updated sequentially, starting from 1 and going up to 100. When updating each latent variable M_i, we use a Normal proposal distribution centered at the current value of the latent variable, with the following proposal standard deviations. When we do not observe y_i, or $y_i = 0$, we use the current value of σ from the state times 0.5. When we observe $y_i > 0$, we use a proposal standard deviation of $1/\sqrt{1/\sigma^2 + y_i}$ (with σ from the state). This choice can be motivated as follows. An estimate of precision for M_i given M_{i-1} is $1/\sigma^2$. Furthermore, Y_i is Poisson(ϕN_i), so that $\mathrm{Var}(\phi N_i) \approx y_i$ and $\mathrm{Var}(\log(\phi N_i)) = \mathrm{Var}(M_i) \approx 1/y_i$. So an estimate for the precision of M_i given y_i is y_i. We combine these estimates of precisions to get a proposal standard deviation for the latent variables in the case when $y_i > 0$.

We ran the Metropolis sampler for $6,000,000$ iterations from five different starting points. The acceptance rate for parameter updates was between about 10% and 20%, depending on the initial starting value. The acceptance rate for latent variable updates was between 11% and 84%, depending on the run and the particular latent variable being sampled.

We found that the simple Metropolis sampler does not perform well on this problem. This is most evident for sampling σ. The Metropolis sampler appears to explore different regions of the posterior for σ when it is started from different initial hidden state sequences. That is, the Metropolis sampler can get stuck in various regions of the posterior for extended periods of time. An example of the behaviour of the Metropolis sampler be seen on Figure 3.2. The autocorrelations for the parameters are so high that accurately estimating them would require a much longer run.

This suggests that more sophisticated MCMC methods are necessary for this problem. We next looked at the single-sequence method, the ensemble method, and the staged

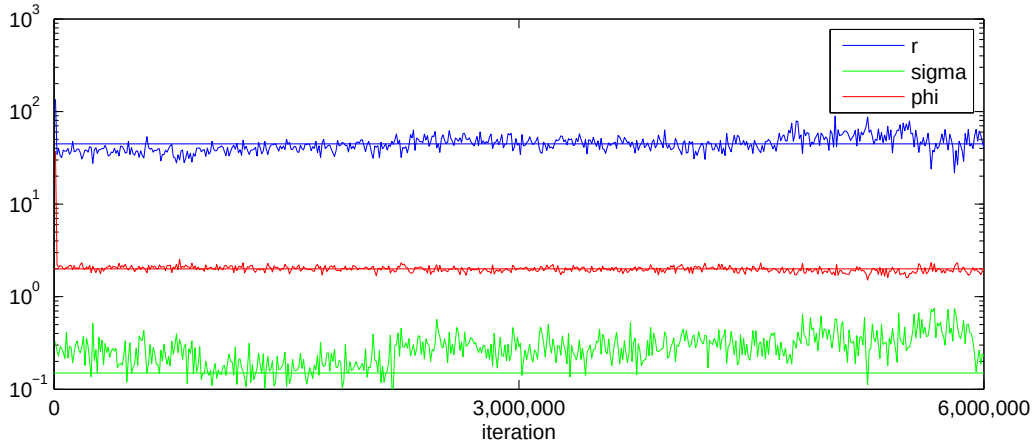


Figure 3.2: An example run of the simple Metropolis method.

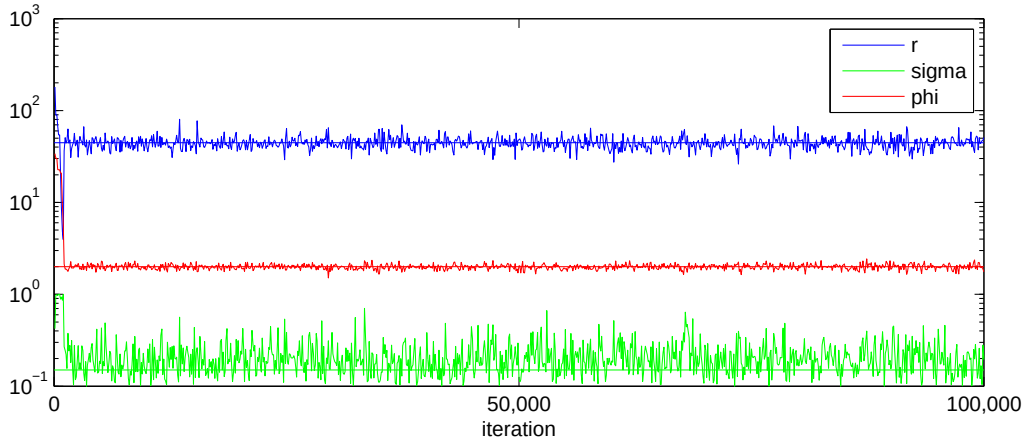


Figure 3.3: An example ensemble method run, with 120 pool states and proposal scaling 1.4.

ensemble method.

All embedded MCMC-based samplers require us to choose pool states for each time i. For the i's where no y_i's are observed, we choose our pool states by sampling values of $\exp(M_i)$ from a pseudo-prior η for the Poisson mean $\exp(M_i)$ — a Gamma(k, θ) distribution with $k = 0.15$ and $\theta = 50$ — and then taking logarithms of the sampled values. For the i's where we observe y_i's we choose our pool states by sampling values of $\exp(M_i)$ from the conditional density of $\exp(M_i)|y_i$ with η as the pseudo-prior. Since $Y_i|M_i$ is Poisson$(\exp(M_i))$, the conditional density of $\exp(M_i)|y_i$ has a Gamma$(k + y_i, \theta/(1 + \theta))$ distribution.

It should be noted that since we choose our pool states by sampling $\exp(M_i)$'s, but our model is written in terms of M_i, the pool density κ_i must take this into account. In particular, when y_i is unobserved, we have

$$\kappa_i(m_i) = \frac{1}{\Gamma(k)\theta^k} \exp(km_i - \exp(m_i)/\theta), \quad -\infty < m_i < \infty \tag{3.26}$$

and when y_i is observed we replace k with $k + y_i$ and θ with $\theta/(1 + \theta)$. We use the same way of choosing the pool states for all three methods we consider.

The staged ensemble MCMC method also requires us to choose a portion of the sequence to use for the first stage update. On the basis of several trial runs, we chose to use the last 20 points, i.e. $n_1 = 81$.

As we mentioned earlier, when likelihood evaluations are computationally inexpensive relative to sequence updates, we can do multiple parameter updates for each update of the hidden state sequence, without incurring a large increase in computation time. For the single-sequence method, we do ten Metropolis updates of the parameters for each update of the hidden state sequence. For the ensemble method, we do five updates of the parameters for each update of the ensemble. For staged ensemble MCMC, we do ten parameter updates for each update of the ensemble. These choices were based on numerous trial runs.

We thin each of the single-sequence runs by a factor of ten when computing autocorrelation times — hence the time per iteration for the single-sequence method is the time it takes to do a single update of the hidden sequence and ten Metropolis updates. We do not thin the ensemble and staged ensemble runs.

To compare the performance of the embedded HMM methods, and to tune each method, we looked at the autocorrelation time, τ, of the sequence of parameter samples, for each parameter. The autocorrelation time can be thought of as the number of samples we need to draw using our Markov chain to get the equivalent of one independent point

(Neal (1993)). It is defined as

$$\tau = 1 + 2\sum_{k=1}^{\infty} \rho_k \tag{3.27}$$

where ρ_k is the autocorrelation at lag k for a function of state that is of interest. Here, $\hat{\rho}_k = \hat{\gamma}_k/\hat{\gamma}_0$, where $\hat{\gamma}_k$ is the estimated autocovariance at lag k. We estimate each $\hat{\gamma}_k$ by estimating autocovariances using each of the five runs, using the overall mean from the five runs, and then averaging the resulting autocovariance estimates. We then estimate τ by

$$\hat{\tau} = 1 + 2\sum_{k=1}^{K} \hat{\rho}_k \tag{3.28}$$

Here, the truncation point K is where the remaining $\hat{\rho}_k$'s are nearly zero.

For our comparisons to have practical validity, we need to multiply each estimate of autocorrelation time estimate by the time it takes to perform a single iteration. A method that produces samples with a lower autocorrelation time is often more computationally expensive than a method that produces samples with a higher autocorrelation time, and if the difference in computation time is sufficiently large, the computationally cheaper method might be more efficient. Computation times per iteration were obtained with a program written in MATLAB on a Linux system with an Intel Xeon X5680 3.33 GHz CPU.

For each number of pool states considered, we started the samplers from five different initial states. Like with the Metropolis method, the parameters were initialized to prior means, and the hidden sequence was initialized to states randomly chosen from the pool distribution at each time. When estimating autocorrelations, we discarded the first 10% of samples of each run as burn-in.

An example ensemble run is shown in Figure 3.3. Comparing with Figure 3.2, one can see that the ensemble run has an enormously lower autocorrelation time. Autocorrelation time estimates for the various ensemble methods, along with computation times, proposal scaling, and acceptance rates, are presented in Table 3.1. For the staged ensemble method, the acceptance rates are shown for the first stage and second stage. We also estimated the parameters of the model by averaging estimates of the posterior means from each of the five runs for the single-sequence method with 40 pool states, the ensemble method with 120 pool states, and the staged ensemble method with 120 pool states. The results are presented in Table 3.2. The estimates using the three different methods do not differ significantly.

From these results, one can see that for our Ricker model example, the single-sequence method is less efficient than the ensemble method, when both methods are well-tuned and

Method	Pool states	Scaling	Acc. Rate	Iter-ations	Time / iteration	ACT r	ACT σ	ACT ϕ	ACT $\times$ time r	ACT $\times$ time σ	ACT $\times$ time ϕ
Single-sequence	10	0.25	12%	400,000	0.09	4345	2362	7272	391	213	654
	20			400,000	0.17	779	1875	1849	132	319	314
	40			200,000	0.31	427	187	1317	132	58	408
	60			200,000	0.47	329	155	879	155	73	413
	80			200,000	0.61	294	134	869	179	82	530
Ensemble	40	0.6	13%	100,000	0.23	496	335	897	114	77	206
	60	1	11%	100,000	0.34	187	115	167	64	39	57
	80	1	16%	100,000	0.47	107	55	90	50	26	42
	120	1.4	14%	100,000	0.76	52	41	45	40	31	34
	180	1.8	12%	100,000	1.26	35	27	29	44	34	37
Staged Ensemble	40	1	30, 15%	100,000	0.10	692	689	1201	69	69	120
	60	1.4	27, 18%	100,000	0.14	291	373	303	41	52	42
	80	1.4	30, 25%	100,000	0.21	187	104	195	39	22	41
	120	1.8	25, 29%	100,000	0.29	75	59	70	22	17	20
	180	2	23, 32%	100,000	0.45	48	44	52	22	20	23

Table 3.1: Comparison of autocorrelation times.

computation time is taken into account. We also found that the staged ensemble method allows us to further improve performance of the ensemble method. In detail, depending on the parameter one looks at, the best tuned ensemble method without staging (with 120 pool states) is between 1.9 and 12.0 times better than the best tuned single-sequence method (with 40 pool states). The best tuned ensemble method with staging (120 pool states) is between 3.4 and 20.4 times better than the best single-sequence method.

The large drop in autocorrelation time for σ for the single-sequence method between 20 and 40 pool states is due to poor mixing in one of the five runs. To confirm whether this is systematic, we did five more runs of the single sequence method, from another set of starting values, and found that the same problem is again present in one of the five runs. This is indicative of a risk of poor mixing when using the single-sequence sampler with a small number of pool states. We did not observe similar problems for larger numbers of pool states.

The results in Table 3.1 show that performance improvement is greatest for the parameter ϕ. One reason for this may be that the posterior distribution of ϕ given a sequence is significantly more concentrated than the marginal posterior distribution of ϕ. Since for a sufficiently large number of pool states, the posterior distribution given an ensemble approximates the marginal posterior distribution, the posterior distribution of

Method	r	σ	ϕ
Single-Sequence	44.46 ($\pm$ 0.09)	0.2074 ($\pm$ 0.0012)	1.9921 ($\pm$ 0.0032)
Ensemble	44.65 ($\pm$ 0.09)	0.2089 ($\pm$ 0.0009)	1.9853 ($\pm$ 0.0013)
Staged Ensemble	44.57 ($\pm$ 0.04)	0.2089 ($\pm$ 0.0010)	1.9878 ($\pm$ 0.0015)

Table 3.2: Estimates of posterior means, with standard errors of posterior means shown in brackets.

ϕ given an ensemble will become relatively more diffuse than the posterior distributions of r and σ. This leads to a larger relative performance improvement when sampling values of ϕ.

Evidence of this can be seen on Figure 3.4. To produce it, we took the hidden state sequence and parameter values from the end of one of our ensemble runs, and performed Metropolis updates for the parameters, while keeping the hidden state sequence fixed. We also took the same hidden state sequence and parameter values, mapped the hidden sequence to an ensemble of sequences by generating a collection of pool states (we used 120) and peformed Metropolis updates of the parameter part of the ensemble, keeping the pool states fixed. We drew $50,000$ samples given a single fixed sequence and $2,000$ samples given an ensemble.

Visually, we can see that the posterior of θ given a single sequence is significantly more concentrated than the marginal posterior, and the posterior given a fixed ensemble of sequences. Comparing the standard deviations of the posterior of θ given a fixed sequence, and the marginal posterior of θ, we find that the marginal posterior of ϕ has a standard deviation about 21 times larger than the posterior of ϕ given a single sequence. The marginal posteriors for r and σ have a posterior standard deviation larger by a factor of 5.2 and 6.0. The standard deviation of the posterior given our fixed ensemble of sequences is greater for ϕ by a factor of 11, and by factors of 4.3 and 3.2 for r and σ. This is consisent with our explanation above.

We note that the actual timings of the runs are different from what one may expect. In theory, the computation time should scale as nL^2, where L is the number of pool states and n is the number of observations. However, the implementation we use, in MATLAB, implements the forward pass as a nested loop over n and over L, with another inner loop over L vectorized. MATLAB is an interpreted language with slow loops, and vectorizing

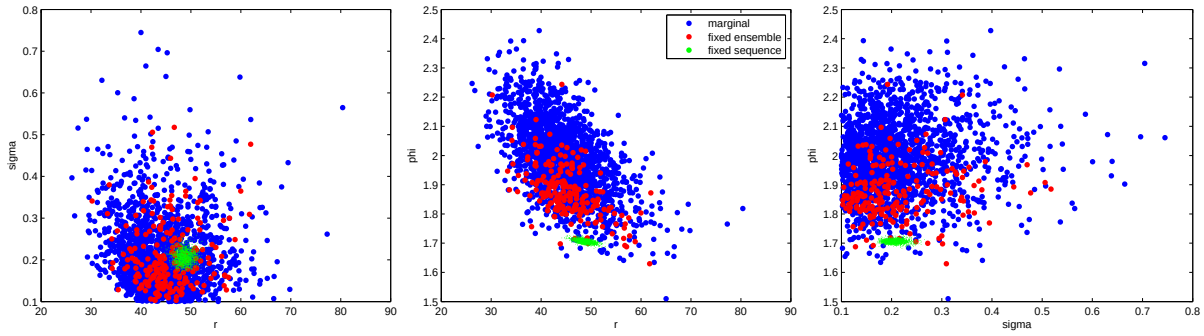


Figure 3.4: An illustration to aid explanation of relative performance. Note that the fixed sequence dots are smaller, to better illustrate the difference in the spread between the fixed sequence and two other distributions.

loops generally leads to vast performance improvements. For the numbers of pool states we use, the computational cost of the vector operation corresponding to the inner loop over L is very low compared to that of the outer loop over L. As a result, the total computational cost scales approximately linearly with L, in the range of values of L we considered. An implementation in a different language might lead to different optimal pool state settings.

The original examples of Wood (2010) and Fearnhead and Prangle (2012) used $\phi = 10$ instead of $\phi = 2$. We found that for $\phi = 10$, the ensemble method still performs better than the single sequence method, when computation time is taken into account, though the difference in performance is not as large. When ϕ is larger, the observations y_i are larger on average as well. As a result, the data is more informative about the values of the hidden state variables, and the marginal posteriors of the model parameters are more concentrated. As a result of this, though we would still expect the ensemble method to improve performance, we would not expect the improvement to be as large as when $\phi = 2$.

Finally, we would like to note that if the single-sequence method is implemented already, implementing the ensemble method is very simple, since all that is needed is an additional call of the forward pass function and summing of the final forward probabilities. So, the performance gains from using an ensemble for parameter updates can be obtained with little additional programming effort.

3.9 Conclusion

We found that both the embedded HMM MCMC method and its ensemble extension perform significantly better than the ordinary Metropolis method for doing Bayesian inference in the Ricker model. This suggests that it would be promising to investigate other state space models with non-linear state dynamics, and see if it is possible to use the embedded HMM methods we described to perform inference in these models.

Our results also show that using staged proposals further improves the performance of the ensemble method. It would be worthwhile to look at other scenarios where this technique might be applied.

Most importantly, however, our results suggest that looking at multiple hidden state sequences at once can make parameter sampling in state space models noticeably more efficient, and so indicate a direction for further research in the development of MCMC methods for non-linear, non-Gaussian state space models.

Chapter 4

Stochastic Volatility

In this chapter, we introduce efficient ensemble Markov Chain Monte Carlo (MCMC) sampling methods for Bayesian computations in the univariate stochastic volatility model. We compare the performance of our ensemble MCMC methods with an improved version of a recent sampler of Kastner and Fruhwirth-Schnatter (2014). We show that ensemble samplers are more efficient than this state of the art sampler by a factor of about 3.1, on a data set simulated from the stochastic volatility model. This performance gain is achieved without the ensemble MCMC sampler relying on the assumption that the latent process is linear and Gaussian, unlike the sampler of Kastner and Fruhwirth-Schnatter.[1]

The stochastic volatility model is a widely-used example of a state space model with non-linear or non-Gaussian transition or observation distributions. It models observed log-returns $y = (y_1, \ldots, y_N)$ of a financial time series with time-varying volatility, as follows:

$$\begin{aligned} Y_i|x_i &\sim N(0, \exp(c + \sigma x_i)), \quad i = 1, \ldots, N && (4.1) \\ X_1 &\sim N(0, 1/(1-\phi^2)) && (4.2) \\ X_i|x_{i-1} &\sim N(\phi x_{i-1}, 1) && (4.3) \end{aligned}$$

Here, the latent process x_i determines the unobserved log-volatility of y_i. Because the relation of the observations to the latent state is not linear and Gaussian, this model cannot be directly handled by efficient methods based on the Kalman filter.

In a Bayesian approach to this problem, we estimate the unknown parameters $\theta = (c, \phi, \sigma)$ by sampling from their marginal posterior distribution $p(\theta|y)$. This distribution cannot be written down in closed form. We can, however, write down the joint posterior of θ and the log-volatilities $x = (x_1, \ldots, x_N)$, $p(\theta, x|y)$, and draw samples of (θ, x) from

[1] Material in this chapter was first presented in Shestopaloff and Neal (2014).

it. Discarding the x coordinates in each draw will give us a sample from the marginal posterior distribution of θ.

To sample from the posterior distribution of the stochastic volatility model, we develop two new MCMC samplers within the framework of ensemble MCMC, introduced by Neal (2010). The key idea underlying ensemble MCMC is to simultaneously look at a collection of points (an "ensemble") in the space we are sampling from, with the K ensemble elements chosen in such a way that the density of interest can be simultaneously evaluated at all of the ensemble elements in less time than it would take to evaluate the density at all K points separately.

In the previous chapter, we developed an ensemble MCMC sampler for non-linear, non-Gaussian state space models, with ensembles over latent state sequences, using the embedded HMM (Hidden Markov Model) technique of Neal (2003), Neal et al. (2004). This ensemble MCMC sampler was used for Bayesian inference in a population dynamics model and shown to be more efficient than methods which only look at a single sequence at a time. In this chapter we consider ensemble MCMC samplers that look not only at ensembles over latent state sequences as in Chapter 3 but also over a subset of the parameters. We see how well both of these methods work for the widely-used stochastic volatility model.

4.1 Bayesian inference for the stochastic volatility model

Bayesian inference for the stochastic volatility model has been extensively studied. In this chapter, we focus on comparisons with the method of Kastner and Fruhwirth-Schnatter (2014). This state-of-the-art method combines the method of Kim et al. (1998) with the ASIS (Ancillary Sufficiency Interweaving Strategy) technique of Yu and Meng (2011). Kastner and Fruhwirth-Schnatter's method consists of two parts. The first is an update of the latent variables x and the second is a joint update of θ and the latent variables x. We improve this method here by saving and re-using sufficient statistics to do multiple parameter updates at little additional computational cost.

4.1.1 A linear Gaussian approximation for sampling latent sequences

The non-linear relationship between the latent and the observation process prohibits the direct use of Kalman filters for sampling the latent variables x_i. Kim et al. (1998)

introduced an approximation to the stochastic volatility model that allows using Kalman filters to draw samples of x_i which can be later reweighed to give samples from their exact posterior distribution. This approximation proceeds as follows. First, the observation process for the stochastic volatility model is written in the form

$$\log(y_i^2) = c + \sigma x_i + \zeta_i \tag{4.4}$$

where ζ_i has a $\log(\chi_1^2)$ distribution.

Next, the distribution of ζ_i is approximated by a ten-component mixture of Gaussians with mixture weights π_k, means m_k and variances $\tau_k^2, k = 1, \ldots, 10$. The values of these mixture weights, means and variances can be found in Omori (2007). At each step of the sampler, at each time i, a single component of the mixture is chosen to approximate the distribution of ζ_i by drawing a mixture component indicator $r_i \in \{1, \ldots, 10\}$ with probabilities proportional to

$$P(r_i = k | y_i, x_i, c, \sigma) \propto \pi_k \frac{1}{\tau_k} \exp((\log(y_i^2) - c - \sigma x_i)^2 / \tau_k^2) \tag{4.5}$$

Conditional on r_i, the observation process is now linear and Gaussian, as is the latent process:

$$\begin{aligned} \log(Y_i^2) | x_i, r_i, c, \sigma &\sim N(m_{r_i} + c + \sigma x_i, \tau_{r_i}^2), \quad i = 1, \ldots, N & (4.6) \\ X_1 | \phi &\sim N(0, 1/(1-\phi^2)) & (4.7) \\ X_i | x_{i-1} &\sim N(\phi x_{i-1}, 1) & (4.8) \end{aligned}$$

Kalman filtering followed by a backward sampling pass can now be used to sample a latent sequence x. For a description of this sampling procedure, see Petris et al. (2009).

The mis-specification of the model due to the approximation can be corrected using importance weights

$$w^{(l)} = \frac{\prod_{i=1}^N f(y_i | x_i^{(l)}, c^{(l)}, \sigma^{(l)})}{\prod_{i=1}^N (\sum_{k=1}^{10} \pi_k g(\log(y_i^2) | x_i^{(l)}, c^{(l)}, \sigma^{(l)}, m_k, \tau_k))} \tag{4.9}$$

where f is the $N(0, \exp(c + \sigma x_i))$ density, g is the $N(m_k + c + \sigma x_i, \tau_k)$ density and the index l refers to a draw. Posterior expectations of functions of θ can then be computed as $\sum w^{(l)} f(\theta^{(l)})$, with $\theta^{(l)}$ the draws.

We note that, when doing any other updates affecting x in combination with this approximate scheme, we need to continue to use the same mixture of Gaussians approx-

imation to the observation process. If an update drawing from an approximate distribution is combined with an update drawing from an exact distribution, neither update will draw samples from their target distribution, since neither update has a chance to reach equilibrium before the other update disturbs things. We would then be unable to compute the correct importance weights to estimate posterior expectations of functions of θ.

4.1.2 ASIS updates

ASIS methods (Yu and Meng (2011)) are based on the idea of interweaving two parametrizations. For the stochastic volatility model, these are the so-called non-centered (NC) and centered (C) parametrizations. The NC parametrization is the one in which the stochastic volatility model was originally presented above. The C parametrization for the stochastic volatility model is

$$\begin{aligned} Y_i|\tilde{x}_i &\sim N(0, \exp(\tilde{x}_i)), \quad i = 1, \ldots, N && (4.10)\\ \tilde{X}_1 &\sim N(c, \sigma^2/(1-\phi^2)) && (4.11)\\ \tilde{X}_i|\tilde{x}_{i-1} &\sim N(c + \phi(\tilde{x}_{i-1} - c), \sigma^2) && (4.12) \end{aligned}$$

The mixture of Gaussians approximation for C is the same as for NC.

Kastner and Fruhwirth-Schnatter (2014) propose two new sampling schemes, GIS-C and GIS-NC, in which they interweave these two parametrizations, using either the NC or C parameterization as the baseline. The authors report a negiligible performance difference between using NC or C as the baseline. For the purposes of our comparisons, we use the method with NC as the baseline, GIS-NC, which proceeds as follows.

1. Draw x given θ, r, y using the linear Gaussian approximation update (NC)
2. Draw θ given x, r, y using a Metropolis update (NC)
3. Move to C by setting $\tilde{x} = c + \sigma x$
4. Draw θ given $\tilde{x}, r, y$ using a Metropolis update (C)
5. Move back to NC by setting $x = \frac{\tilde{x}-c}{\sigma}$
6. Redraw the mixture component indicators r given θ, x, y.

Theorem 4 of Yu and Meng (2011) establishes a link between ASIS and the PX-DA (Parameter Expansion-Data Augmentation) method of Liu and Wu (1999). In the

case of the stochastic volatility model, this means that we can view the ASIS scheme for updating x and θ as a combination of two updates, both done in the NC parametrization. The first of these draws new values for θ conditional on x. The second draws new values for both x and θ, such that when we propose to update c to c^* and σ to σ^*, we also propose to update the sequence x to $x^* = ((c+\sigma x) - c^*)/\sigma^*$. For this second update, the Metropolis acceptance probability needs to be multiplied by a Jacobian factor $(\sigma/\sigma^*)^N$ to account for scaling σ. A joint translation update for c and x has been previously considered by Liu and Sabatti (2000) and successfully applied to to stochastic volatility model. Scale updates are considered by Liu and Sabatti (2000) as well, though they do not apply them to the stochastic volatility model.

The view of ASIS updates as joint updates to θ and x makes it easier to see why ASIS updates improve efficiency. At first glance, they look like they only update the parameters, but they actually end up proposing to change both θ and x in a way that preserves dependence between them. This means that moves proposed in ASIS updates are more likely to end up in a region of high posterior density, and so be accepted.

Kastner and Fruhwirth-Schnatter (2014) do a single Metropolis update of the parameters for every update of the latent sequence. However, we note that given values for the mixture indices r, y and x, low-dimensional sufficient statistics exist for all parameters in the centered parametrization. In the non-centered parametrization, given r, y and x, low-dimensional sufficient statistics exist for ϕ. We propose doing multiple Metropolis updates given saved values of these sufficient statistics (for all parameters in the case of C and for ϕ in the case of NC). This allows us to reach equilibrium given a fixed latent sequence at little computational cost since additional updates have small cost, not dependent on N. Also, this eliminates the need to construct complex proposal schemes, since with these repeated samples the algorithm becomes less sensitive to the particular choice of proposal density.

The sufficient statistics in the case of NC are

$$t_1 = \sum_{i=1}^{N} x_i^2, \quad t_2 = \sum_{i=2}^{N} x_{i-1}x_i, \quad t_3 = x_1^2 + x_N^2 \tag{4.13}$$

with the log-likelihood of ϕ as a function of the sufficient statistics being

$$\log(L(\phi|t)) = (1/2)\log(1-\phi^2) - (1/2)(\phi^2(t_1 - t_3) - 2\phi t_2 + t_1)$$

In the case of C the sufficient statistics are

$$\tilde{t}_1 = \sum_{i=1}^{N} \tilde{x}_i^2, \quad \tilde{t}_2 = \sum_{i=2}^{N-1} \tilde{x}_i^2 \quad \tilde{t}_3 = \sum_{i=2}^{N} \tilde{x}_{i-1}\tilde{x}_i \quad \tilde{t}_4 = \sum_{i=2}^{N-1} \tilde{x}_i \quad \tilde{t}_5 = \tilde{x}_1 + \tilde{x}_N \tag{4.14}$$

with the log-likelihood as a function of the sufficient statistics being

$$\begin{aligned} \log(L(c,\phi,\sigma^2|\tilde{t})) &= -(N/2)\log(\sigma^2) + (1/2)\log(1-\phi^2) \\ &\quad -(1/2)(\tilde{t}_1 + \phi^2\tilde{t}_2 - 2\phi\tilde{t}_3 - 2c\phi^2\tilde{t}_4 - 2c(\tilde{t}_4 + \tilde{t}_5) \\ &\quad +4c\phi\tilde{t}_4 + 2c\phi\tilde{t}_5 + (N-1)(c(\phi-1))^2 + c^2(1-\phi^2))/\sigma^2 \end{aligned} \tag{4.15}$$

The details of the derivations are given in the Appendix.

4.2 Ensemble MCMC methods for stochastic volatility models

The general framework underlying ensemble MCMC methods was introduced by Neal (2010). An ensemble MCMC method using embedded HMMs for parameter inference in non-linear, non-Gaussian state space models was introduced in Chapter 3. We briefly review ensemble methods for non-linear, non-Gaussian state space models here.

Ensemble MCMC builds on the idea of MCMC using a temporary mapping. Suppose we are interested in sampling from a distribution with density $\pi(z)$ on $\mathcal{Z}$. We can do this by constructing a Markov chain with transition kernel $T(z'|z)$ with invariant distribution π. The temporary mapping strategy takes T to be a composition of three stochastic mappings. The first mapping, $\hat{T}$, takes z to an element w of some other space $\mathcal{W}$. The second, $\bar{T}$, updates w to w'. The last, $\check{T}$, takes w' back to some $z' \in \mathcal{Z}$. The idea behind this strategy is that doing updates in an intermediate space $\mathcal{W}$ may allow us to make larger changes to z, as opposed to doing updates directly in $\mathcal{Z}$.

In the ensemble method, the space $\mathcal{W}$ is taken to be the K-fold Cartesian product of $\mathcal{Z}$. First, z mapped to an ensemble $w = (z^{(1)}, \ldots, z^{(K)})$, with the current value z assigned to $z^{(k)}$, with $k \in \{1, \ldots, K\}$ chosen uniformly at random. The remaining elements $z^{(j)}$ for $j \neq k$ are chosen from their conditional distribution under an ensemble base measure ζ, given that $z^{(k)} = z$. The marginal density of an ensemble element $z^{(k)}$ in the ensemble base measure ζ is denoted by $\zeta_k(z^{(k)})$. Next, w is updated to w' using any update that

leaves invariant the *ensemble density*

$$\rho(w) = \rho((z^{(1)}, \ldots, z^{(K)})) = \zeta((z^{(1)}, \ldots, z^{(K)}))\frac{1}{K}\sum_{i=1}^{K}\frac{\pi(z^{(k)})}{\zeta_k(z^{(k)})} \tag{4.16}$$

Finally, a new value z' is chosen by selecting an element $z^{(k)}$ from the ensemble with probabilities proportional to $\pi(z^{(k)})/\zeta_k(z^{(k)})$. The benefit of doing, say Metropolis, updates in the space of ensembles is that a proposed move is more likely to be accepted, since for the ensemble density to be large it is enough that the proposed ensemble contains at least some elements with high density under π.

In Chapter 3, we consider an ensemble over latent state sequences x. Specifically, the current state, (x, θ), consisting of the latent states x and the parameters θ is mapped to an ensemble $y = ((x^{(1)}, \theta), \ldots, (x^{(K)}, \theta))$ where the ensemble contains all distinct sequences $x^{(k)}$ passing through a collection of pool states chosen at each time i. The ensemble is then updated to $y' = ((x^{(1)}, \theta'), \ldots, (x^{(K)}, \theta'))$ using a Metropolis update that leaves ρ invariant. At this step, only θ is changed. We then map back to a new $x' = (x', \theta')$, where x' is now potentially different from the original x. We show that this method considerably improves sampling efficiency in the Ricker model of population dynamics.

As in the original Neal (2010) paper, we emphasize here that applications of ensemble methods are worth investigating when the density at each of the K elements of an ensemble can be computed in less time than it takes to do K separate density evaluations. For the stochastic volatility model, this is possible for ensembles over latent state sequences, and over the parameters c and σ^2. In this chapter, we will only consider joint ensembles over x and over σ. Since we will use $\eta = \log(\sigma^2)$ in the MCMC state, we will refer to ensembles over η below.

We propose two ensemble MCMC sampling schemes for the stochastic volatility model. The first, ENS1, updates the latent sequence, x, and η by mapping to an ensemble composed of latent sequences x and values of η, then immediately mapping back to new values of x and η. The second, ENS2, maps to an ensemble of latent state sequences x and values of η, like ENS1, then updates ϕ using an ensemble density summing over x and η, and finally maps back to new values of x and η.

For both ENS1 and ENS2, we first create a pool of η values with L_η elements, and at each time, i, a pool of values for the latent state x_i, with L_x elements. The current value of η is assigned to the pool element $\eta^{[1]}$ and for each time i, the current x_i is assigned to the pool element $x_i^{[1]}$. (Since the pool states are drawn independently, we don't need to randomly assign an index to the current η and the current x_i's in their pools.) The

remaining pool elements are drawn independently from some distribution having positive probability for all possible values of x_i and η, say κ_i for x_i and λ for η.

The total number of ensemble elements that we can construct using the pools over x_i and over η is $L_\eta L_x^N$. Naively evaluating the ensemble density presents an enormous computational burden for $L_x > 1$, taking time on the order of $L_\eta L_x^N$. By using the forward algorithm, together with a "caching" technique, we can evaluate the ensemble density much more efficiently, in time on the order of $L_\eta L_x^2 N$. The forward algorithm is used to efficiently evaluate the densities for the ensemble over the x_i. The caching technique is used to efficiently evaluate the densities for the ensemble over η, which gives us a substantial constant factor speed-up in terms of computation time.

In detail, we do the following. Let $p(x_1)$ be the initial state distribution, $p(x_i|x_{i-1})$ the transition density for the latent process and $p(y_i|x_i, \eta)$ the observation probabilities. We begin by computing and storing the initial latent state probabilities — which do not depend on η — for each pool state $x_1^{[k]}$ at time 1.

$$P_1 = (p(x_1^{[1]}), \ldots, p(x_1^{[L_x]})) \tag{4.17}$$

For each $\eta^{[l]}$ in the pool and each pool state $x_1^{[k]}$ we then compute and store the initial forward probabilities

$$\alpha_1^{[l]}(x_1^{[k]}|\eta^{[l]}) \;=\; p(x_1^{[k]})\frac{p(y_1|x_1^{[k]}, \eta^{[l]})}{\kappa_1(x_1^{[k]})} \tag{4.18}$$

Then, for $i > 1$, we similarly compute and store the matrix of transition probabilities

$$P_i = \begin{pmatrix} p(x_i^{[1]}|x_{i-1}^{[1]}) & \ldots & p(x_i^{[L_x]}|x_{i-1}^{[1]}) \\ \vdots & \ddots & \vdots \\ p(x_i^{[1]}|x_{i-1}^{[L_x]}) & \ldots & p(x_i^{[L_x]}|x_{i-1}^{[L_x]}) \end{pmatrix}$$

where

$$p(x_i^{[k_1]}|x_{i-1}^{[k_2]}) \propto \exp(-(x_i^{[k_1]} - \phi x_{i-1}^{[k_2]})^2/2) \tag{4.19}$$

are transition probabilities between pool states $x_{i-1}^{[k_2]}$ and $x_i^{[k_1]}$ for $k_1, k_2 \in \{1, \ldots, L_x\}$. We then use the stored values of the transition probabilities P_i to efficiently compute the

vector of forward probabilities for all values of $\eta^{[l]}$ in the pool

$$\alpha_i^{[l]}(x_i|\eta^{[l]}) = \frac{p(y_i|x_i,\eta^{[l]})}{\kappa_i(x_i)} \sum_{k=1}^{L_x} p(x_i|x_{i-1}^{[k]})\alpha_{i-1}^{[l]}(x_{i-1}^{[k]}|\eta^{[l]}), \quad i = 1,\ldots,N \tag{4.20}$$

with $x_i \in \{x_i^{[1]},\ldots,x_i^{[L_x]}\}$.

At each time i, we divide the forward probabilities $\alpha_i^{[l]}$ by $c_i^{[l]} = \sum_{k=1}^{L_x} \alpha_i^{[l]}(x_i^{[k]}|\eta^{[l]})$, storing the $c_i^{[l]}$ values and using the normalized $\alpha_i^{[l]}$'s in the next step of the recursion. This is needed to prevent underflow and for ensemble density computations. In the case of all the forward probabilities summing to 0, we set the forward probabilities at all subsequent times to 0. Note that we won't get underflows for all values of $\eta^{[l]}$, since we are guaranteed to have a log-likelihood that is not $-\infty$ for the current value of η in the MCMC state.

For each $\eta^{[l]}$, the ensemble density can then be computed as

$$\rho^{[l]} = \prod_{i=1}^{N} c_i^{[l]} \tag{4.21}$$

To avoid overflow or underflow, we work with the logarithm of $\rho^{[l]}$.

Even with caching, computing the forward probabilities for each $\eta^{[l]}$ in the pool is still an order L_x^2 operation since we multiply the vector of forward probabilities from the previous step by the transition matrix. However, if we do not cache and re-use the transition probabilities P_i when computing the forward probabilities for each value of $\eta^{[l]}$ in the pool, the computation of the ensemble densities $\rho^{[l]}$, for all $l = 1,\ldots,L_\eta$, would be about 10 times slower. This is because computing forward probabilities for a value of η given saved transition probabilities only involves multiplications and additions, and not exponentiations, which are comparatively more expensive.

In ENS1, after mapping to the ensemble, we immediately sample new values of η and x from the ensemble. We first sample a $\eta^{[l]}$ from the marginal ensemble distribution, with probabilities proportional to $\rho^{[l]}$. After we have sampled an $\eta^{[l]}$, we sample a latent sequence x conditional on $\eta^{[l]}$, using a stochastic backwards recursion. The stochastic backwards recursion first samples a state x_N from the pool at time N with probabilities proportional to $\alpha_N^{[l]}(x_N|\eta^{[l]})$. Then, given the sampled value of x_i, we sample x_{i-1} from the pool at time $i-1$ with probabilities proportional to $p(x_i|x_{i-1})\alpha_{i-1}^{[l]}(x_{i-1}|\eta^{[l]})$, going back to time 1.

In the terminology of Chapter 3, this is a "single sequence" update combined with an ensemble update for η (which is a "fast" variable in the terminology of Neal (2010)

since recomputation of the likelihood function after changes to this variable is fast given the saved transition probabilities).

In ENS2, before mapping back to a new η and a new x as in ENS1, we perform a Metropolis update for ϕ using the ensemble density summing over all $\eta^{[l]}$ and all latent sequences in the ensemble, $\sum_{l=1}^{L_\eta} \rho^{[l]}$. This approximates updating ϕ using the posterior density of θ with x and η integrated out, when the number of pool states is large. The update nevertheless leaves the correct distribution exactly invariant, even if the number of pool states is not large.

4.2.1 Choosing the pool distribution

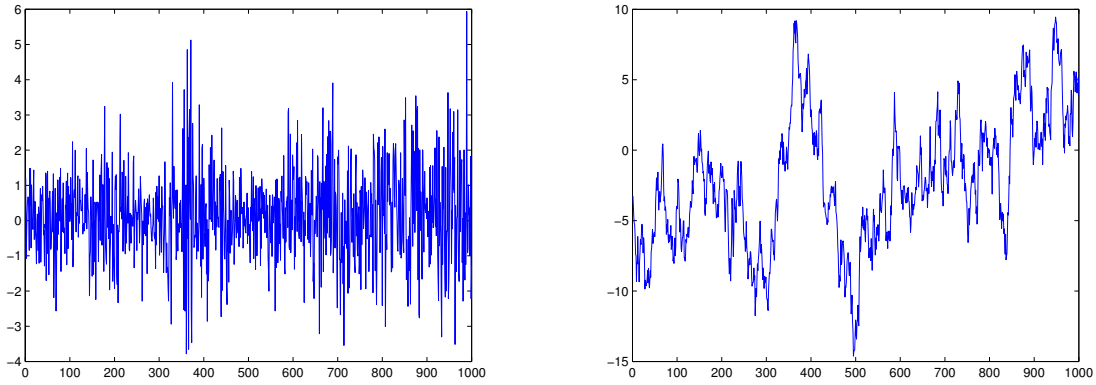
A good choice for the pool distribution is crucial for the efficient performance of the ensemble MCMC method.

For a pool distribution for x_i, a good candidate is the stationary distribution of x_i in the AR(1) latent process, which is $N(0, 1/\sqrt{1-\phi^2})$. The question here is how to choose ϕ. For ENS1, which does not change ϕ, we can simply use the current value of ϕ from the MCMC state, call it ϕ_{cur}, and draw pool states from $N \sim (0, c/\sqrt{1-\phi_{\text{cur}}^2})$ for some scaling factor c. Typically, we would choose $c > 1$ in order to ensure that for different values of ϕ, we produce pool states that cover the region where x_i has high probability density.

We cannot use this pool selection scheme for ENS2 because the reverse transition after a change in ϕ would use different pool states, undermining the proof via reversibility that the ensemble transitions leave the posterior distribution invariant. However, we can choose pool states that depend on both the current and the proposed values of ϕ, say ϕ and ϕ^*, in a symmetric fashion. For example, we can propose a value ϕ^*, and draw the pool states from $N \sim (0, c/\sqrt{1-\phi_{\text{avg}}^2})$ where ϕ_{avg} is the average of ϕ and ϕ^*. The validity of this scheme can be seen by considering ϕ^* to be an additional variable in the model; proposing to update ϕ to ϕ^* can then be viewed as proposing to swap ϕ and ϕ^* within the MCMC state.

We choose pool states for η by sampling them from the model prior. Alternative schemes are possible, but we do not consider them here. For example, it is possible to draw local pool states for η which stay close to the current value of η by running a Markov chain with some desired stationary distribution J steps forwards and $L_\eta - J - 1$ steps backwards, starting at the current value of η. For details, see Neal (2003).

In Chapter 3, one recommendation we made was to consider pool states that depend on the observed data y_i at a given point, constructing a "pseudo-posterior" for x_i using



(a) y (b) x

Figure 4.1: Data set used for testing.

data observed at time i or in a small neighbourhood around i. For the ensemble updates ENS1 and ENS2 presented here, we cannot use this approach, as we would then need to make the pool states also depend on the current values of c and η, the latter of which is affected by the update. We could switch to the centered parametrization to avoid this problem, but that would prevent us from making η a fast variable.

4.3 Comparisons

The goal of our computational experiments is to determine how well the introduced variants of the ensemble method compare to our improved version of the Kastner and Fruhwirth-Schnatter (2014) method. We are also interested in understanding when using a full ensemble update is helpful or not.

4.3.1 Data

We use a series simulated from the stochastic volatility model with parameters $c = 0.5, \phi = 0.98, \sigma^2 = 0.0225$ with $N = 1000$. A plot of the data is presented in Figure 4.1.

We use the following priors for the model parameters.

$$c \sim N(0, 1) \tag{4.22}$$

$$\phi \sim \text{Unif}[0, 1] \tag{4.23}$$

$$\sigma^2 \sim \text{Inverse-Gamma}(2.5, 0.075) \tag{4.24}$$

We use the parametrization in which the Inverse-Gamma(α, β) has probability density

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta/x}, \quad x > 0 \tag{4.25}$$

For $\alpha = 2.5, \beta = 0.075$ the 2.5% and 97.5% quantiles of this distribution are approximately $(0.0117, 0.180)$.

In the MCMC state, we transform ϕ and σ^2 to

$$\eta = \log(\sigma^2) \tag{4.26}$$

$$\gamma = \log((1+\phi)/(1-\phi)) \tag{4.27}$$

with the priors transformed correspondingly.

4.3.2 Sampling schemes and tuning

We compare three sampling schemes — the Kastner and Fruhwirth-Schnatter (KF) method, and our two ensemble schemes, ENS1, in which we map to an ensemble of η and x values and immediately map back, and ENS2, in which we additionally update γ with an ensemble update before mapping back.

We combine the ensemble scheme with the computationally cheap ASIS Metropolis updates. It is sensible to add cheap updates to a sampling scheme if they are available. Note that the ASIS (or translation and scale) updates we use in this chapter are generally applicable to location-scale models and are not restricted by the linear and Gaussian assumption.

Pilot runs showed that 80 updates appears to be the point at which we start to get diminishing returns from using more Metropolis updates (given the sufficient statistics) in the KF scheme. This is the number of Metropolis updates we use with the ensemble schemes as well.

The KF scheme updates the state as follows:

1. Update x, using the Kalman filter-based update, using the current mixture indicators r.

2. Update the parameters using the mixture approximation to the observation density. This step consists of 80 Metropolis updates to γ given the sufficient statistics for NC, followed by one joint update of c and η.

3. Change to the C parametrization.

4. Update all three parameters simultaneously using 80 Metropolis updates, given the sufficient statistics for C. Note that this update does not depend on the observation density and is therefore exact.

5. Update the mixture indicators r.

The ENS1 scheme proceeds as follows:

1. Map to an ensemble of η and x.

2. Map back to a new value of η and x.

3. Do steps 2) - 4) as for KF, but with the exact observation density.

The ENS2 scheme proceeds as follows:

1. Map to an ensemble of η and x.

2. Update γ using an ensemble Metropolis update.

3. Map back to a new value of η and x.

4. Do steps 2) - 4) as for KF, but with the exact observation density.

The Metropolis updates use a normal proposal density centered at the current parameter values. Proposal standard deviations for the Metropolis updates in NC were set to estimated marginal posterior standard deviations, and to half of that in C. This is because in C, we update all three parameters at once, whereas in NC we update c and η jointly and γ separately. The marginal posterior standard deviations were estimated using a pilot run of the ENS2 method. The tuning settings for the Metropolis updates are presented in Table 4.1.

For ensemble updates of γ, we also use a normal proposal density centered at the current value of γ, with a proposal standard deviation of 1, which is double the estimated marginal posterior standard deviation of γ. The pool states over x_i are selected from the stationary distribution of the AR(1) latent process, with standard deviation $2/\sqrt{1-\phi_{\text{cur}}^2}$ for the ENS1 scheme and $2/\sqrt{1-\phi_{\text{avg}}^2}$ for the ENS2 scheme. We used the prior density of η to select pool states for η.

For each method, we started the samplers from 5 randomly chosen points. Parameters were initalized to their prior means (which were 0 for c, 1.39 for γ and -3.29 for η), and each $x_i, i = 1, \ldots, N$, was initialized independently to a value randomly drawn from the stationary distribution of the AR(1) latent process, given γ set to the prior mean. For

the KF updates, the mixture indicators r where all initialized to 5's, this corresponds to the mixture component whose median matches the median of the $\log(\chi^2_1)$ distribution most closely. All methods were run for approximately the same amount of computational time.

4.3.3 Results

Before comparing the performance of the methods, we verified that the methods give the same answer up to expected variation by looking at the 95% confidence intervals each produced for the posterior means of the parameters. These confidence intervals were obtained from the standard error of the average posterior mean estimate over the five runs. The KF estimates were adjusted using the importance weights that compensate for the use of the approximate observation distribution. No significant disagreement between the answers from the different methods was apparent. We then evaluated the performance of each method using estimates of autocorrelation time, which measures how many MCMC draws are needed to obtain the equivalent of one independent draw.

To estimate autocorrelation time, we first estimated autocovariances for each of the five runs, discarding the first 10% of the run as burn-in, and plugging in the overall mean of the five runs into the autocovariance estimates. (This allows us to detect if the different runs for each method are exploring different regions of the parameter/latent variable space). We then averaged the resulting autocovariance estimates and used this average to get autocorrelation estimates $\hat{\rho}_k$. Finally, autocorrelation time was estimated as $1 + 2\sum_{k=1}^{K} \hat{\rho}_i$, with K chosen to be the point beyond which the ρ_k become approximately 0. All autocovariances were estimated using the Fast Fourier Transform for computational efficiency.

The results are presented in Tables 4.2 and 4.3. The timings for each sampler represent an average over 100 iteratons (each iteration consisting of the entire sequence of updates), with the samplers started from a point taken after the sampler converged to equilibrium. The program was written in MATLAB and run on a Linux system with an Intel Xeon

Method	Prop. Std. (NC)			Acc. Rate	Acc. Rate	Prop. Std. (C)			Acc. Rate
	c	γ	η	for γ (NC)	for (c,η) (NC)	c	γ	η	for (c,γ,η)
KF					0.12				
ENS1	0.21	0.5	0.36	0.52	0.12	0.105	0.25	0.18	0.22
ENS2									

Table 4.1: Metropolis proposal standard deviations with associated acceptance rates.

X5680 3.33 GHz CPU. For a fair comparison, we multiply estimated autocorrelation times by the time it takes to do one iteration and compare these estimates.

We ran the KF method for $140,000$ iterations, with estimated autocorrelation times using the original (unweighed) sequence for (c, γ, η) of $(2.1, 37, 73)$, which after adjusting by computation time of 0.16 seconds per iteration are $(0.34, 5.9, 12)$. It follows that the ENS1 method with L_x set to 50 and L_η set to 10 is better than the KF method by a factor of about 3.1 for the parameter η. For ENS2, the same settings $L_x = 50$ and $L_\eta = 10$ appears to give the best results, with ENS2 worse by a factor of about 1.7 than ENS1 for sampling η. We also see that the ENS1 and ENS2 methods aren't too sensitive to the particular tuning parameters, so long at there is a sufficient number of ensemble elements both for x_i and for η.

The results show that using a small ensemble (10 or so pool states) over η is particularly helpful. One reason for this improvement is the ability to use the caching technique to make these updates computationally cheap. A more basic reason is that updates of η consider the entire collection of latent sequences, which allows us to make large changes to η, compared to the Metropolis updates.

Even though the ENS2 method in this case is outperformed by the ENS1 method, we have only applied it to one data set and there is much room for further tuning and improvement of the methods. A possible explanation for the lack of substantial performance gain with the ensemble method is that conditional on a single sequence, the distribution of ϕ has standard deviation comparable to its marginal standard deviation, which means that we can't move too much further with an ensemble update than we do with our Metropolis updates. An indication of this comes from the acceptance rate for ensemble updates of γ in ENS2, which we can see isn't improved by much as more pool states are added.

Parameter estimates for the best performing KF, ENS1 and ENS2 settings are presented in Table 4.4. These estimates were obtained by averaging samples from all 5 runs with 10% of the sample discarded as burn-in. We see that the differences between the standard errors are in approximate agreement with the differences in autocorrelation times for the different methods.

4.4 Conclusion

We found that noticeable performance gains can be obtained by using ensemble MCMC based sampling methods for the stochastic volatility model. It may be possible to obtain even larger gains on different data sets, and with even better tuning. In particular, it

L_x	L_η	Iterations	Time/iter (s)	ACT			ACT × time		
				c	γ	η	c	γ	η
10	1	195000	0.11	2.6	99	160	0.29	11	18
	10	180000	0.12	2.7	95	150	0.32	11	18
	30	155000	0.14	2.6	81	130	0.36	11	18
	50	140000	0.16	2.3	91	140	0.37	15	22
30	1	155000	0.14	2.4	35	71	0.34	4.9	9.9
	10	135000	0.16	2.2	18	26	0.35	2.9	4.2
	30	110000	0.20	2.3	19	26	0.46	3.8	5.2
	50	65000	0.33	1.9	16	24	0.63	5.3	7.9
50	1	115000	0.19	1.9	34	68	0.36	6.5	13
	10	95000	0.23	1.9	11	17	0.44	2.5	3.9
	30	55000	0.38	2.2	8.9	12	0.84	3.4	4.6
	50	55000	0.39	1.9	11	14	0.74	4.3	5.5
70	1	85000	0.25	2.2	33	67	0.55	8.3	17
	10	60000	0.38	1.9	8.3	11	0.72	3.2	4.2
	30	50000	0.42	1.8	8.4	11	0.76	3.5	4.6
	50	45000	0.48	1.9	9.1	12	0.91	4.4	5.8

Table 4.2: Performance of method ENS1.

L_x	L_η	Acc. Rate for γ	Iterations	Time/iter (s)	ACT			ACT × time		
					c	γ	η	c	γ	η
10	1	0.32	110000	0.20	2.5	100	170	0.5	20	34
	10	0.32	95000	0.23	2.4	91	140	0.55	21	32
	30	0.32	80000	0.27	2.5	97	150	0.68	26	41
	50	0.32	70000	0.30	2.7	90	140	0.81	27	42
30	1	0.33	80000	0.26	2.3	34	68	0.6	8.8	18
	10	0.33	70000	0.31	2.3	18	26	0.71	5.6	8.1
	30	0.33	55000	0.39	2	18	27	0.78	7	11
	50	0.34	35000	0.61	2.4	12	19	1.5	7.3	12
50	1	0.34	60000	0.36	1.7	33	69	0.61	12	25
	10	0.35	50000	0.44	2.1	10	15	0.92	4.4	6.6
	30	0.34	30000	0.71	1.8	10	15	1.3	7.1	11
	50	0.34	25000	0.81	1.8	12	17	1.5	9.7	14
70	1	0.34	45000	0.49	2.2	29	61	1.1	14	30
	10	0.35	30000	0.72	1.6	7.3	11	1.2	5.3	7.9
	30	0.36	25000	0.86	1.6	7.3	9.3	1.4	6.3	8
	50	0.36	25000	0.96	1.8	5.9	7.8	1.7	5.7	7.5

Table 4.3: Performance of method ENS2.

Method	c	γ	η
KF	0.2300 ($\pm$ 0.0004)	4.3265 ($\pm$ 0.0053)	-3.7015 ($\pm$ 0.0054)
ENS1	0.2311 ($\pm$ 0.0006)	4.3228 ($\pm$ 0.0017)	-3.6986 ($\pm$ 0.0015)
ENS2	0.2306 ($\pm$ 0.0008)	4.3303 ($\pm$ 0.0025)	-3.7034 ($\pm$ 0.0021)

Table 4.4: Estimates of posterior means, with standard errors of posterior means shown in brackets.

is possible that the method of updating ϕ with an ensemble, or some variation of it, actually performs better than a single sequence method in some other instance.

The method of Kastner and Fruhwirth-Schnatter (2014) relies on the assumption that the state process is linear and Gaussian, which enables efficient state sequence sampling using Kalman filters. The method would not be applicable if this was not the case. However, the ensemble method could still be applied to this case as well. It would be of interest to investigate the performance of ensemble methods for stochastic volatility models with different noise structures for the latent process. It would also be interesting to compare the performance of the ensemble MCMC method with the PMCMC-based methods of Andrieu et. al (2010) and also to see whether techniques used to improve PMCMC methods can be used to improve ensemble methods and vice versa.

Multivariate versions of stochastic volatility models, for example those considered in Scharth and Kohn (2013) are another class of models for which inference is difficult, and that it would be interesting to apply the ensemble MCMC method to. We have done preliminary experiments applying ensemble methods to multivariate stochastic volatility models, with promising results. For these models, even though the latent process is linear and Gaussian, due to a non-constant covariance matrix the observation process does not have a simple and precise mixture of Gaussians approximation.

Appendix

Here, we derive the sufficient statistics for the stochastic volatility model in the two parametrizations and the likelihoods in terms of sufficient statistics. For NC, we derive

low-dimensional sufficient statistics for ϕ as follows

$$\begin{aligned} p(x|\phi) &\propto \sqrt{1-\phi^2}\exp\Big(-(1-\phi^2)x_1^2/2\Big)\exp(-\sum_{i=2}^{N}(x_i-\phi x_{i-1})^2/2\Big) \\ &\propto \sqrt{1-\phi^2}\exp\Big(-(x_1^2-\phi^2x_1^2+\sum_{i=2}^{N}x_i^2-2\phi\sum_{i=2}^{N}x_ix_{i-1}+\phi^2\sum_{i=2}^{N}x_{i-1}^2)/2\Big) \\ &\propto \sqrt{1-\phi^2}\exp\Big(-(\phi^2\sum_{i=2}^{N-1}x_i^2-2\phi\sum_{i=2}^{N}x_ix_{i-1}+\sum_{i=1}^{N}x_i^2)/2\Big) \end{aligned}$$

Letting

$$t_1=\sum_{i=1}^{N}x_i^2,\quad t_2=\sum_{i=2}^{N}x_{i-1}x_i,\quad t_3=x_1^2+x_N^2$$

we can write

$$p(x|\phi) \propto \sqrt{1-\phi^2}\exp(-(\phi^2(t_1-t_3)-2\phi t_2+t_1)/2)$$

For C, the probability $p(x|c,\phi,\sigma^2)$ is proportional to

$$\begin{aligned} &\frac{\sqrt{1-\phi^2}}{\sigma^{n/2}}\exp\Big(-(1-\phi^2)(\tilde{x}_1-c)^2/2\sigma^2\Big)\exp\Big(-\sum_{i=2}^{N}((\tilde{x}_i-(c+\phi(\tilde{x}_{i-1}-c))^2/2\sigma^2\Big) \\ \propto\ &\frac{\sqrt{1-\phi^2}}{\sigma^{n/2}}\exp\Big(-(\tilde{x}_1^2-\phi^2\tilde{x}_1^2-2\tilde{x}_1c+c^2+2c\phi^2\tilde{x}_1-c^2\phi^2+\sum_{i=2}^{N}\tilde{x}_i^2 \\ &-2\sum_{i=2}^{N}\tilde{x}_i(c+\phi(\tilde{x}_{i-1}-c))+\sum_{i=2}^{N}(c+\phi(\tilde{x}_{i-1}-c))^2)/2\sigma^2\Big) \\ \propto\ &\frac{\sqrt{1-\phi^2}}{\sigma^{n/2}}\exp\Big(-(\sum_{i=1}^{N}\tilde{x}_i^2-\phi^2\tilde{x}_1^2-2\tilde{x}_1c+c^2+2c\phi^2\tilde{x}_1-c^2\phi^2-2c\sum_{i=2}^{N}\tilde{x}_i \\ &-2\phi\sum_{i=2}^{N}\tilde{x}_i\tilde{x}_{i-1}+2c\phi\sum_{i=2}^{N}\tilde{x}_i+(N-1)c^2+2c\phi\sum_{i=2}^{N}(\tilde{x}_{i-1}-c) \\ &+\phi^2\sum_{i=2}^{N}(\tilde{x}_{i-1}-c)^2)/2\sigma^2\Big) \end{aligned}$$

$$\begin{aligned}
\propto \quad & \frac{\sqrt{1-\phi^2}}{\sigma^{n/2}} \exp\Big(-(\sum_{i=1}^{N} \tilde{x}_i^2 - \phi^2\tilde{x}_1^2 - 2\tilde{x}_1 c + c^2 + 2c\phi^2\tilde{x}_1 - c^2\phi^2 - 2c\sum_{i=2}^{N}\tilde{x}_i \\
& -2\phi\sum_{i=2}^{N}\tilde{x}_i\tilde{x}_{i-1} + 2c\phi\sum_{i=2}^{N}\tilde{x}_i + (N-1)c^2 + 2c\phi\sum_{i=2}^{N}\tilde{x}_{i-1} - 2(N-1)c^2\phi) \\
& +\phi^2\sum_{i=2}^{N}\tilde{x}_{i-1}^2 - 2c\phi^2\sum_{i=2}^{N}\tilde{x}_{i-1} + (N-1)c^2\phi^2)/2\sigma^2\Big) \\
\propto \quad & \frac{\sqrt{1-\phi^2}}{\sigma^{n/2}} \exp\Big(-(\sum_{i=1}^{N}\tilde{x}_i^2 + \phi^2\sum_{i=2}^{N-1}\tilde{x}_i^2 - 2\phi\sum_{i=2}^{N}\tilde{x}_{i-1}\tilde{x}_i + 2c\phi^2\sum_{i=2}^{N-1}\tilde{x}_i - 2c\sum_{i=1}^{N}\tilde{x}_i \\
& +4c\phi\sum_{i=2}^{N-1}\tilde{x}_i + 2c\phi(\tilde{x}_1 + \tilde{x}_N) + (N-1)(c(\phi-1))^2 + c^2(1-\phi^2))/2\sigma^2\Big)
\end{aligned}$$

Letting

$$\tilde{t}_1 = \sum_{i=1}^{N}\tilde{x}_i^2, \quad \tilde{t}_2 = \sum_{i=2}^{N-1}\tilde{x}_i^2 \quad \tilde{t}_3 = \sum_{i=2}^{N}\tilde{x}_{i-1}\tilde{x}_i \quad \tilde{t}_4 = \sum_{i=2}^{N-1}\tilde{x}_i \quad \tilde{t}_5 = \tilde{x}_1 + \tilde{x}_N$$

we can write

$$\begin{aligned}
p(x|c,\phi,\sigma^2) \quad = \quad & \frac{\sqrt{1-\phi^2}}{\sigma^{n/2}} \exp\Big(-(\tilde{t}_1 + \phi^2\tilde{t}_2 - 2\phi\tilde{t}_3 - 2c\phi^2\tilde{t}_4 - 2c(\tilde{t}_4 + \tilde{t}_5) \\
& +4c\phi\tilde{t}_4 + 2c\phi\tilde{t}_5 + (N-1)(c(\phi-1))^2 + c^2(1-\phi^2))/2\sigma^2\Big)
\end{aligned}$$

Chapter 5

Embedded HMMs for High Dimensions

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 applied to models with a one-dimensional state space. 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. We also consider a model with a multimodal posterior, and show how a technique we term "mirroring" can be used to efficiently move between the modes. [1]

5.1 Introduction

Consider a non-linear, non-Gaussian state space model for an observed sequence $y = (y_1, \ldots, y_n)$. 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 continu-

[1] Material in this chapter was first presented in Shestopaloff and Neal (2016).

ous or discrete. We may be interested in inferring both the realized values of the Markov process $x = (x_1, \ldots, x_n)$ and the model parameters θ. In a Bayesian approach to this problem, this can be done by drawing a sample of values for x and θ using a Markov chain that alternately samples from the conditional posterior distributions $p(x|\theta, y)$ and $p(\theta|x, y)$. In this chapter, we will only consider inference for x by sampling from $p(x|\theta, y)$, 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, Beal and Roweis, 2004). An embedded 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 x' is selected from the "ensemble" of L^n sequences passing through the set 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, a new set of pool states is constructed, so that the chain can sample all possible x_i, even when the set of possible values is infinite.

Another method is the Particle Gibbs with Backward Sampling (PGBS) method. The Particle Gibbs (PG) method was first introduced in Andrieu, Doucet and Holenstein (2010); Whiteley suggested the backward sampling modification in the discussion following this paper. Lindsten and Schon (2012) implemented backward sampling and showed that it improves the efficiency of PG. Starting with a current sequence x, PGBS first uses conditional Sequential Monte Carlo (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 takes time that is only linear in the number of particles.

Both the PGBS and embedded HMM methods can facilitate sampling of a latent state sequence, 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 $p(x|y,\theta)$, 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 chapter, 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 Chapter 3, 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 Chapter 4, 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 for dimensions as small as three. Since these schemes are global, 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 doesn't 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, such as making 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.

In this chapter, 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, 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.

5.2 Embedded HMM MCMC

We review the embedded HMM method (Neal, 2003; Neal, Beal and Roweis, 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.

Suppose our current sequence is $x = (x_1, \ldots, x_n)$. The embedded HMM sampler updates x to 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 a pool density 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

$$\kappa_i(x) R_i(x'|x) = \kappa_i(x') \tilde{R}_i(x|x') \tag{5.1}$$

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 $x_i^{[l_i+1]}, \ldots, x_n^{[L]}$.

At each $i = 1, \ldots, n$, we then compute the forward probabilities $\alpha_i(x)$, with x taking

values in $\mathcal{P}_i$. At time $i = 1$, we have

$$\alpha_1(x) = \frac{p(x)p(y_1|x)}{\kappa_1(x)} \tag{5.2}$$

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

$$\alpha_i(x) = \frac{p(y_i|x)}{\kappa_i(x)} \sum_{l=1}^{L} p(x|x_{i-1}^{[l]})\alpha_{i-1}(x_{i-1}^{[l]}) \tag{5.3}$$

Finally, we sample a new state sequence x' using a stochastic backwards pass. This is done by selecting x'_n amongst the set, $\mathcal{P}_n$, of pool states at time n, with probabilities proportional to $\alpha_n(x)$, and then going backwards, sampling x'_{i-1} from the set P_{i-1}, with probabilities proportional to $\alpha_{i-1}(x)p(x'_i|x)$. 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.

Alternatively, given a set of pool states, embedded HMM updates can be done by first computing the backward probabilities. We will see later on that the backward probability formulation of the embedded HMM method allows us to introduce a variation of our proposed pool state selection scheme. Setting $\beta_n(x) = 1$ for all $x \in \mathcal{P}_n$, we compute for $i < n$

$$\beta_i(x) = \frac{1}{\kappa_i(x)} \sum_{l=1}^{L} p(y_{i+1}|x_{i+1}^{[l]})p(x_{i+1}^{[l]}|x)\beta_{i+1}(x_{i+1}^{[l]}) \tag{5.4}$$

A new state sequence is then sampled using a stochastic forward pass, setting x'_1 to one of the x in the pool $\mathcal{P}_1$ with probabilities proportional to $\beta_1(x)p(x)p(y_1|x)$ and then choosing subsequent states x'_i from the pools $\mathcal{P}_i$ with probabilities proportional to $\beta_i(x)p(x|x'_{i-1})p(y_i|x)$.

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 (5.3) or (5.4). Hence each iteration of the embedded HMM sampler takes time proportional to $L^2 n$.

5.3 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, Doucet and Holenstein (2010) and Lindsten and Schon (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 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:

$$w_1^{[l]} = \frac{p(x_1^{[l]})p(y_1|x_1^{[l]})}{q_1(x_1^{[l]}|y_1)} \tag{5.5}$$

$$W_1^{[l]} = \frac{w_1^{[l]}}{\sum_{m=1}^{L} w_1^{[m]}} \tag{5.6}$$

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, $x_i^{[l]}$, at time i, for $l = 2, \ldots, L$, from $q_i(x|y_i, x_{i-1}^{[A_{i-1}^{[l]}]})$ and compute and normalize the weights at time i

$$w_i^{[l]} = \frac{p(x_i^{[l]}|x_{i-1}^{[A_{i-1}^{[l]}]})p(y_i|x_i^{[l]})}{q_i(x_i^{[l]}|y_i, x_{i-1}^{[A_{i-1}^{[l]}]})} \tag{5.7}$$

$$W_i^{[l]} = \frac{w_i^{[l]}}{\sum_{m=1}^{L} w_i^{[m]}} \tag{5.8}$$

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

$$\frac{w_i^{[l]}p(x'_{i+1}|x_i^{[l]})}{\sum_{m=1}^{L} w_i^{[m]}p(x'_{i+1}|x_i^{[m]})} \tag{5.9}$$

A common choice for q is the model's transition density, which is what is compared to in this chapter.

Note that each iteration of the PGBS sampler takes time proportional to Ln, 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.

5.4 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 $i+1$.

5.4.1 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

$$\kappa_1^f(x) \propto p(x)p(y_1|x) \tag{5.10}$$

As a result of equation (5.2), $\alpha_1(x)$ is constant. At time $i > 1$, we set the pool state distribution to

$$\kappa_i^f(x|\mathcal{P}_{i-1}) \propto p(y_i|x)\sum_{\ell=1}^{L} p(x|x_{i-1}^{[\ell]}) \tag{5.11}$$

which makes $\alpha_i(x)$ constant for $i > 1$ as well (see equation (5.3)).

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

$$\kappa_i^b(x|\mathcal{P}_{i+1}) \propto \sum_{\ell=1}^{L} p(y_{i+1}|x_{i+1}^{[\ell]})p(x_{i+1}^{[\ell]}|x) \tag{5.12}$$

so that $\beta_i(x)$ is constant for all $i = 1,\ldots,n$ (see equation 5.4).

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.

5.4.2 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 $L^2 n$, 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 $\ell = 1, \ldots, L$ that indexes a pool state at the previous time. Specifically, κ_i^f can be viewed as a marginal of the density

$$\lambda_i(x, \ell) \propto p(y_i|x)p(x|x_{i-1}^{[\ell]}) \tag{5.13}$$

while κ_i^b is a marginal of the density

$$\gamma_i(x, \ell) \propto p(y_{i+1}|x_{i+1}^{[\ell]})p(x_{i+1}^{[\ell]}|x) \tag{5.14}$$

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 $i+1$. This technique is reminiscent of the auxiliary particle filter of Pitt and Shephard (1999). We then use Markov transitions, R_i, to sample a set of values of x and ℓ, with probabilities proportional to λ_i for the forward scheme, or probabilities proportional to γ_i for the backward scheme.

The chain is started at x set to the current x_i, and the initial value of ℓ is chosen randomly with probabilities proportional to $p(x_i|x_{i-1}^{[\ell]})$ for the forward scheme or $p(y_{i+1}|x_{i+1}^{[\ell]})p(x_{i+1}^{[\ell]}|x_i)$ for the backward scheme. This stochastic initialization of ℓ 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 ℓ from λ_i or γ_i can be done by updating each of x and ℓ separately, alternately sampling values of x conditional on ℓ, and values of ℓ conditional on x, or by updating x and ℓ jointly, or by a combination of these.

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

To jointly update x and ℓ, we propose two novel updates, a "shift" update and a "flip" update. Since these are also Metropolis updates, using them together with Metropolis or autoregressive updates, for each of x and ℓ separately, allows embedded HMM updates to be performed in time proportional to nL.

5.4.3 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.

This autoregressive update proceeds as follows. Let L be the lower triangular Cholesky decomposition of Σ, so $\Sigma = LL^T$, and n be a vector of i.i.d. normal random variables with zero mean and identity covariance. Let $\epsilon \in [-1, 1]$ be a tuning parameter that

determines the scale of the proposal. Starting at w, we propose

$$w' = \mu + \sqrt{1-\epsilon^2}(w-\mu) + \epsilon L n \tag{5.15}$$

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

$$\min\left(1, \frac{p(y|w')}{p(y|w)}\right) \tag{5.16}$$

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.

5.4.4 Shift updates

We can simultaneously update ℓ and x at time $i > 1$ by proposing to update (x, ℓ) to (x', ℓ') where ℓ' is proposed in any valid way while x' is chosen in a way such that x' and $x_{i-1}^{[\ell']}$ are linked in the same way as x and $x_{i-1}^{[\ell]}$. The 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 we use later, 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 ℓ', we propose $x_i' = x_i + \Phi(x_{i-1}^{[\ell']} - x_{i-1}^{[\ell]})$. This update is accepted with probability

$$\min\left(1, \frac{p(y_i|x_i')}{p(y_i|x_i)}\right) \tag{5.17}$$

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

$$\begin{aligned} x_i' - \Phi x_{i-1}^{[\ell']} &= x_i + \Phi(x_{i-1}^{[\ell']} - x_{i-1}^{[\ell]}) - \Phi x_{i-1}^{[\ell']} && (5.18) \\ &= x_i - \Phi x_{i-1}^{[\ell]} && (5.19) \end{aligned}$$

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 and ℓ may lead to a relatively high acceptance rate on updates of x, in order to ensure that the acceptance rate for the shift updates isn't low.

5.4.5 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 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 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, ℓ) to $(-x, \ell')$ such that $x_{i-1}^{[\ell']} = -x_{i-1}^{[\ell]}$. 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 ℓ to ℓ' 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 ℓ to ℓ' can be seen as proposing to flip the lower order bit in a binary representation of ℓ'. For example, a proposal to move from $\ell = 3$ to $\ell = 2$ can be seen as proposing to change ℓ 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 $|x_i|$.

5.4.6 Relation to PGBS

The forward pool state selection scheme can be used to construct a sampler with properties similar to PGBS. This is done by using independence Metropolis to sample values of x and ℓ from λ_i.

At time 1, we propose our pool states from $p(x)$. At times $i > 2$, we propose ℓ' by selecting it uniformly at random from $\{1, \ldots, L\}$ and we propose x' by sampling from $p(x|x_{i-1}^{[\ell']})$. The proposals at all times i are accepted with probability

$$\min\left(1, \frac{p(y_i|x_i')}{p(y_i|x_i)}\right) \tag{5.20}$$

This sampler has computational cost proportional to Ln per iteration, like PGBS. It is analogous to a PGBS sampler with importance densities

$$q_1(x|y_1) = p(x) \tag{5.21}$$

and

$$q_i(x|x_{i-1}, y_i) = p(x|x_{i-1}), \quad i > 2 \tag{5.22}$$

with the key difference between these two samplers being that PGBS uses importance weights $p(y_i|x_i)$ on each particle, instead of an independence Metropolis accept-reject step.

5.5 Proof of correctness

We modify the original proof of Neal (2003), which assumes that the sets of pool states $\mathcal{P}_1, \ldots, \mathcal{P}_n$ are selected independently at each time, to show the validity of our new sequential pool state selection scheme. Another change in the proof is to account for generating the pool states by sampling them from λ_i or γ_i instead of κ_i^f or κ_i^b.

This proof shows that the probability of starting at x and moving to x' with given sets of pool states $\mathcal{P}_i$ (consisting of values of x 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 x' and moving to 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 detailed balance with respect to $p(x|y)$, and hence leaves $p(x|y)$ invariant.

Suppose we use the sequential forward scheme. The probability of starting at x and moving to x' decomposes into the product of the probability of starting at x, which is $p(x|y)$, the probability of choosing a set of pool state indices l_i, which is $\frac{1}{L^n}$, the probability of selecting the initial values of ℓ_i for the stochastic initialization step, the probability of selecting the sets of pool states $\mathcal{P}_i$, $P(\mathcal{P}_1, \ldots, \mathcal{P}_n)$, and finally the probability of choosing x'.

The probability of selecting given initial values for the links to previous states $\ell_2, \ldots, \ell_n$ is

$$\prod_{i=2}^{n} \frac{p(x_i|x_{i-1}^{[\ell_i]})}{\sum_{m=1}^{L} p(x_i|x_{i-1}^{[m]})} \tag{5.23}$$

The probability of choosing a given set of pool states is

$$P(\mathcal{P}_1, \ldots, \mathcal{P}_n) = P(\mathcal{P}_1) \prod_{i=2}^{n} P(\mathcal{P}_i|\mathcal{P}_{i-1}) \tag{5.24}$$

At time 1, we use a Markov chain with invariant density κ_1 to select pool states in $\mathcal{P}_1$. Therefore

$$\begin{aligned}
P(\mathcal{P}_1) &= \prod_{j=l_1+1}^{L} R_1(x_1^{[j]}|x_1^{[j-1]}) \prod_{j=l_1-1}^{1} \tilde{R}_1(x_1^{[j]}|x_1^{[j+1]}) \\
&= \prod_{j=l_1+1}^{L} R_1(x_1^{[j]}|x_1^{[j-1]}) \prod_{j=l_1-1}^{1} R_1(x_1^{[j+1]}|x_1^{[j]}) \frac{\kappa_1(x_1^{[j]})}{\kappa_1(x_1^{[j+1]})} \\
&= \prod_{j=l_1}^{L-1} R_1(x_1^{[j+1]}|x_1^{[j]}) \prod_{j=l_1-1}^{1} R_1(x_1^{[j+1]}|x_1^{[j]}) \frac{\kappa_1(x_1^{[j]})}{\kappa_1(x_1^{[j+1]})} \\
&= \frac{\kappa_1(x_1^{[1]})}{\kappa_1(x_1^{[l_1]})} \prod_{j=1}^{L-1} R_1(x_1^{[j+1]}|x_1^{[j]})
\end{aligned} \tag{5.25}$$

For times $i > 1$ we use a Markov chain with invariant density λ_i to sample a set of pool

states, given $\mathcal{P}_{i-1}$. The chain is started at $x_i^{[l_i]} = x_i$ and $\ell_i^{[l_i]} = \ell_i$. Therefore

$$\begin{aligned}
P(\mathcal{P}_i|\mathcal{P}_{i-1}) &= \prod_{j=l_1+1}^{L} R_i(x_i^{[j]}, \ell_i^{[j]}|x_i^{[j-1]}, \ell_i^{[j-1]}) \prod_{j=l_i-1}^{1} \tilde{R}_i(x_i^{[j]}, \ell_i^{[j]}|x_i^{[j+1]}, \ell_i^{[j+1]}) \\
&= \prod_{j=l_i+1}^{L} R_i(x_i^{[j]}, \ell_i^{[j]}|x_i^{[j-1]}, \ell_i^{[j-1]}) \prod_{j=l_i-1}^{1} R_i(x_1^{[j+1]}, \ell_i^{[j+1]}|x_i^{[j]}, \ell_i^{[j]}) \frac{\lambda_i(x_i^{[j]}, \ell_i^{[j]})}{\lambda_i(x_1^{[j+1]}, \ell_i^{[j+1]})} \\
&= \prod_{j=l_i}^{L-1} R_i(x_i^{[j+1]}, \ell_i^{[j+1]}|x_i^{[j]}, \ell_i^{[j]}) \prod_{j=l_i-1}^{1} R_i(x_i^{[j+1]}, \ell_i^{[j+1]}|x_i^{[j]}, \ell_i^{[j]}) \frac{\lambda_i(x_i^{[j]}, \ell_i^{[j]})}{\lambda_i(x_i^{[j+1]}, \ell_i^{[j+1]})} \\
&= \frac{\lambda_i(x_i^{[1]}, \ell_i^{[1]})}{\lambda_i(x_i^{[l_i]}, \ell_i^{[l_i]})} \prod_{j=1}^{L-1} R_i(x_i^{[j+1]}, \ell_i^{[j+1]}|x_i^{[j]}, \ell_i^{[j]}) \qquad (5.26)
\end{aligned}$$

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

$$\prod_{i=2}^{n} \frac{p(x'_i|x_{i-1}^{[l'_{i-1}]})}{\sum_{m=1}^{L} p(x'_i|x_{i-1}^{[m]})} \qquad (5.27)$$

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

$$\begin{aligned}
& p(x|y) \times \frac{1}{L^n} \times \prod_{i=2}^{n} \frac{p(x_i|x_{i-1}^{[\ell_i]})}{\sum_{m=1}^{L} p(x_i|x_{i-1}^{[m]})} \times \frac{\kappa_1(x_1^{[1]})}{\kappa_1(x_1^{[l_1]})} \prod_{j=1}^{L-1} R_1(x_1^{[j+1]}|x_1^{[j]}) \qquad (5.28) \\
& \times \prod_{i=2}^{n} \left[\frac{\lambda_i(x_i^{[1]}, \ell_i^{[1]})}{\lambda_i(x_i^{[l_i]}, \ell_i^{[l_i]})} \prod_{j=1}^{L-1} R_i(x_i^{[j+1]}, \ell_i^{[j+1]}|x_i^{[j]}, \ell_i^{[j]}) \right] \times \frac{1}{L} \times \prod_{i=2}^{n} \frac{p(x'_i|x_{i-1}^{[l'_{i-1}]})}{\sum_{m=1}^{L} p(x'_i|x_{i-1}^{[m]})} \\
&= \kappa_1(x_1^{[1]}) \prod_{j=1}^{L-1} R_1(x_1^{[j+1]}|x_1^{[j]}) \times \prod_{i=2}^{n} \left[\lambda_i(x_i^{[1]}, \ell_i^{[1]}) \prod_{j=1}^{L-1} R_i(x_i^{[j+1]}, \ell_i^{[j+1]}|x_i^{[j]}, \ell_i^{[j]}) \right] \\
& \times \frac{1}{L^{n+1}} \times \frac{p(x|y)}{\kappa_1(x_1) \prod_{i=2}^{n} \lambda_i(x_i, \ell_i)} \times \prod_{i=2}^{n} \frac{p(x_i|x_{i-1}^{[\ell_i]})}{\sum_{m=1}^{L} p(x_i|x_{i-1}^{[m]})} \prod_{i=2}^{n} \frac{p(x'_i|x_{i-1}^{[l'_{i-1}]})}{\sum_{m=1}^{L} p(x'_i|x_{i-1}^{[m]})}
\end{aligned}$$

Here, we have $x_{i-1}^{[l'_{i-1}]} = x'_{i-1}$. Also $\kappa_1(x_1) = p(x_1)p(y_1|x_1)/\sum_{x_1 \in \mathcal{P}_1} p(x_1)p(y_1|x_1)$ and

$$\prod_{i=2}^{n} \lambda_i(x_i, \ell_i) = \prod_{i=2}^{n} \frac{p(y_i|x_i)p(x_i|x_{i-1}^{[\ell_i]})}{\sum_{x_i \in \mathcal{P}_i} \sum_{m=1}^{L} p(y_i|x_i)p(x_i|x_{i-1}^{[m]})} \qquad (5.29)$$

and

$$p(x|y) = \frac{p(x_1)\prod_{i=2}^{n} p(x_i|x_{i-1})\prod_{i=1}^{n} p(y_i|x_i)}{p(y)} \tag{5.30}$$

Therefore (5.28) can be simplified to

$$\begin{aligned}
&\frac{1}{p(y)}\kappa_1(x_1^{[1]})\prod_{j=1}^{L-1} R_1(x_1^{[j+1]}|x_1^{[j]}) \times \prod_{i=2}^{n}\left[\lambda_i(x_i^{[1]},\ell_i^{[1]})\prod_{j=1}^{L-1} R_i(x_i^{[j+1]},\ell_i^{[j+1]}|x_i^{[j]},\ell_i^{[j]})\right] \times \frac{1}{L^{n+1}} \\
&\times \prod_{i=2}^{n} p(x_i|x_{i-1}) \times \prod_{i=2}^{n} p(x_i'|x_{i-1}') \times \prod_{i=2}^{n} \frac{1}{\sum_{m=1}^{L} p(x_i|x_{i-1}^{[m]})} \times \prod_{i=2}^{n} \frac{1}{\sum_{m=1}^{L} p(x_i'|x_{i-1}^{[m]})} \\
&\times \sum_{x_1\in\mathcal{P}_1} p(x_1)p(y_1|x_1)\prod_{i=2}^{n}\sum_{x_i\in\mathcal{P}_i}\sum_{m=1}^{L} p(y_i|x_i)p(x_i|x_{i-1}^{[m]})
\end{aligned} \tag{5.31}$$

The last factor in the product only depends on the selected set of pool states. By exchanging x and x' we see that the probability of starting at x' and then going to 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.

5.6 Experiments

5.6.1 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

$$X_1 \sim N(0, \Sigma_{\text{init}}) \tag{5.32}$$

$$X_i|x_{i-1} \sim N(\Phi x_{i-1}, \Sigma), \quad i = 2, \ldots, n \tag{5.33}$$

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

$$\Phi = \begin{pmatrix} \phi_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \phi_P \end{pmatrix} \tag{5.34}$$

$$\Sigma = \begin{pmatrix} 1 & \ldots & \rho \\ \vdots & \ddots & \vdots \\ \rho & \ldots & 1 \end{pmatrix} \tag{5.35}$$

$$\Sigma_{\text{init}} = \begin{pmatrix} \frac{1}{1-\phi_1^2} & \cdots & \frac{\rho}{\sqrt{1-\phi_1^2}\sqrt{1-\phi_P^2}} \\ \vdots & \ddots & \vdots \\ \frac{\rho}{\sqrt{1-\phi_P^2}\sqrt{1-\phi_1^2}} & \cdots & \frac{1}{1-\phi_P^2} \end{pmatrix} \tag{5.36}$$

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

For model 1, the observations are given by

$$Y_{i,j}|x_{i,j} \sim \text{Poisson}(\exp(c_j + \sigma_j x_{i,j})), \quad i = 1, \ldots, n, \quad j = 1, \ldots, P \tag{5.37}$$

For model 2, the observations are given by

$$Y_{i,j}|x_{i,j} \sim \text{Poisson}(\sigma_j |x_{i,j}|), \quad i = 1, \ldots, n, \quad j = 1, \ldots, P \tag{5.38}$$

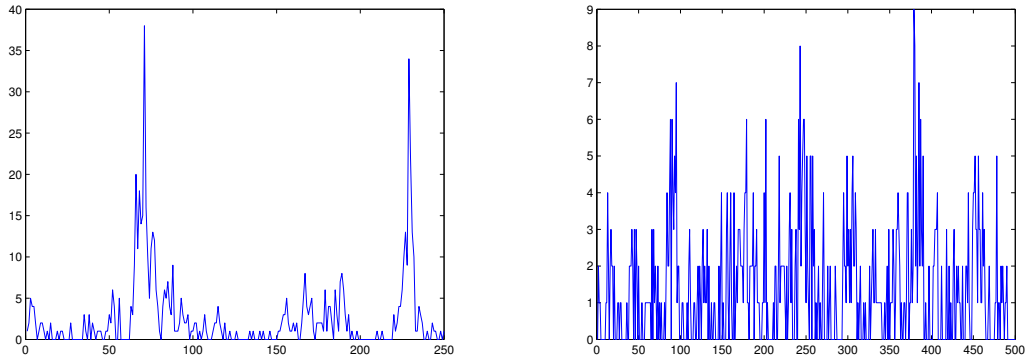
For model 1, we use a 10-dimensional latent state and a sequence length of $n = 250$, setting parameter values to $\rho = 0.7$, and $c_j = -0.4$, $\phi_j = 0.9$, $\sigma_j = 0.6$ for $j = 1, \ldots, P$, with $P = 10$.

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 $P = 15$.

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 5.1. Note that we are testing only sampling of the latent variables, with the parameters set to their true values.

5.6.2 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 $x_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_i)$ 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 5.4.3).



(a) Model 1 (b) Model 2

Figure 5.1: Observations from Model 1 and Model 2 along dimension $j = 1$.

The conditional densities of the X_i are

$$\begin{aligned}
p(x_1|x_{-1}, y) &\propto p(x_1|x_2)p(y_1|x_1) \propto p(x_1)p(x_2|x_1)p(y_1|x_1) \\
p(x_i|x_{-i}, y) &\propto p(x_i|x_{i-1}, x_{i+1})p(y_i|x_i) \propto p(x_i|x_{i-1})p(x_{i+1}|x_i)p(y_i|x_i), \quad 2 \leq i \leq n-1 \\
p(x_n|x_{-n}, y) &\propto p(x_n|x_{n-1})p(y_n|x_n)
\end{aligned}$$

The densities $p(x_1|x_2), p(x_i|x_{i-1}, x_{i+1})$, and $p(x_n|x_{n-1})$ are all Gaussian. The means and covariances for these densities can be derived by viewing $p(x_1)$ or $p(x_i|x_{i-1})$ as a Gaussian prior for x_i and $p(x_{i+1}|x_i)$ as a Gaussian likelihood for x_i. In particular, we have

$$\begin{aligned}
X_1|x_2 &\sim N(\mu_1, \Sigma_1) \\
X_i|x_i, x_{i+1} &\sim N(\mu_i, \Sigma_i) \\
X_n|x_{n-1} &\sim N(\mu_n, \Sigma_n)
\end{aligned}$$

where

$$\begin{aligned}
\mu_1 &= [(\Phi^{-1}\Sigma\Phi^{-1})^{-1} + \Sigma_{\text{init}}^{-1}]^{-1}[(\Phi^{-1}\Sigma\Phi^{-1})^{-1}\Phi^{-1}x_2] \\
&= [(\Phi^{-1}\Sigma\Phi^{-1})^{-1} + \Sigma_{\text{init}}^{-1}]^{-1}[\Sigma^{-1}(\Phi x_2)] \\
&= [\Phi^2 + \Sigma_{\text{init}}^{-1}\Sigma]^{-1}\Phi x_2 \\
\Sigma_1 &= [(\Phi^{-1}\Sigma\Phi^{-1})^{-1} + \Sigma_{\text{init}}^{-1}]^{-1} \\
&= [\Phi(\Sigma^{-1}\Phi) + \Sigma_{\text{init}}^{-1}]^{-1}
\end{aligned}$$

$$\begin{aligned}
\mu_i &= [(\Phi^{-1}\Sigma\Phi^{-1})^{-1} + \Sigma^{-1}]^{-1}[\Sigma^{-1}\Phi x_{i-1} + (\Phi^{-1}\Sigma\Phi^{-1})^{-1}\Phi^{-1}x_{i+1}] \\
&= (\Phi^{-1}\Sigma\Phi^{-1})^{-1} + \Sigma^{-1}]^{-1}[\Sigma^{-1}(\Phi(x_{i-1} + x_{i+1}))] \\
&= [\Phi^2 + I]^{-1}\Phi(x_{i-1} + x_{i+1}) \\
\Sigma_i &= [(\Phi^{-1}\Sigma\Phi^{-1})^{-1} + \Sigma^{-1}]^{-1} \\
&= [\Phi(\Sigma^{-1}\Phi) + \Sigma^{-1}]^{-1}
\end{aligned}$$

$$\begin{aligned}
\mu_n &= \Phi x_{n-1} \\
\Sigma_n &= \Sigma
\end{aligned}$$

To speed up the Metropolis updates, we precompute and store the matrices $[\Phi^2 + \Sigma_{\text{init}}^{-1}\Sigma]^{-1}\Phi$, $[\Phi^2 + I]^{-1}\Phi$ as well as the Cholesky decompositions of the posterior covariances.

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, in the autoregressive update when performing an iteration of the Metropolis sampler.

5.6.3 Particle Gibbs with Backward Sampling with Metropolis

We implement the PGBS method as described in Section 5.3, 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 5.6.2. 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.

5.6.4 Tuning the Baseline Samplers

For model 1, we compared the embedded HMM sampler to the simple single-state Metropolis sampler, to the PGBS sampler, and to the combination of PGBS with Metropolis. For model 2, we compared the embedded HMM sampler to the PGBS with Metropolis sampler. For both models and all samplers, we ran the sampler five times using five different random number generator seeds. We implemented the samplers in MATLAB on a Linux system with a 2.60 GHz Intel i7-3720QM CPU.

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 1000000 iterations, and prior to analysis, the resulting sample was thinned by a factor of 10, to 100000. 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 100000 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 70000 iterations. Each iteration produced two samples for a total of 140000 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 30000 iterations of the sampler. Each iteration was used to produce four samples, for a total of 120000 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 120000 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 80000, 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 1000 samples, and consisted of a PGBS update with the forward sequence, fifty Metropolis updates (of which we only keep the value after the last one), a PGBS update using the reversed sequence, and another fifty Metropolis updates (again only keeping the last value). It took about 26 seconds to draw each sample.

5.6.5 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. As for 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 $\lambda_i(x, l)$ by combining an autoregressive and shift update. The autoregressive update proposes to only change x, keeping the current l fixed. The shift update samples both x and l, with a new l 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.

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.

As for the single-state Metropolis updates, it is beneficial to use a different proposal scaling, ϵ, when generating each pool state at each 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.

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 9000 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 18000 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 9000 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 18000 samples took about 1.4 seconds to draw.

5.6.6 Comparisons

As a way of comparing the performance of the two methods, we use an estimate of autocorrelation time[2] 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 the equivalent of one independent sample. The autocorrelation time is defined as $\tau =$

[2]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 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.

$1 + 2\sum_{i=1}^{\infty} \rho_k$, where ρ_k is the autocorrelation at lag k. It is commonly estimated as

$$\hat{\tau} = 1 + 2\sum_{i=1}^{K} \hat{\rho}_k \tag{5.39}$$

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

$$\hat{\rho}_k = \frac{\hat{\gamma}_k}{\hat{\gamma}_0} \tag{5.40}$$

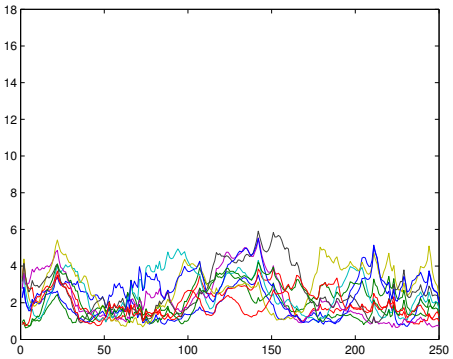
where $\hat{\gamma}_k$ is an estimate of the lag-k autocovariance

$$\hat{\gamma}_k = \frac{1}{n}\sum_{l=1}^{n-k}(x_l - \bar{x})(x_{k+l} - \bar{x}) \tag{5.41}$$

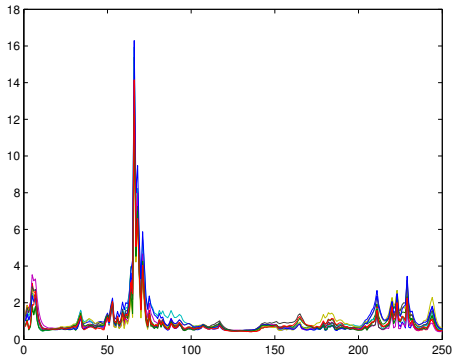
When estimating autocorrelation time, we remove the first 10% of the sample as burn-in. Then, to estimate $\hat{\gamma}_k$, we first estimate autocovariances for each of the five runs, taking $\bar{x}$ to be the overall mean over the five runs. We then average these five autocovariance estimates to produce $\hat{\gamma}_k$. 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 5.2. 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 the use of the diffuse transition distribution the particles are drawn from inefficient and 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 depends a lot on how well vectorization can be exploited. Implementing the samplers in a different language might change the relative comparisons.

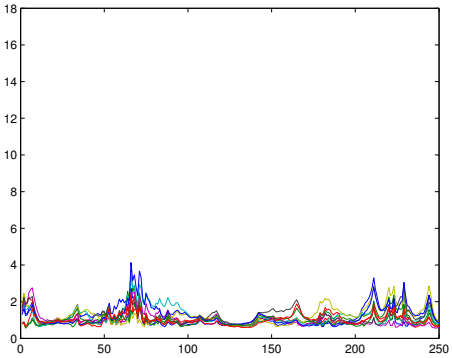
We now look at how the samplers perform on the more challenging Model 2. We first did a preliminary check of whether the samplers do indeed explore the different modes of the distribution by looking at variables far apart in the sequence, where we



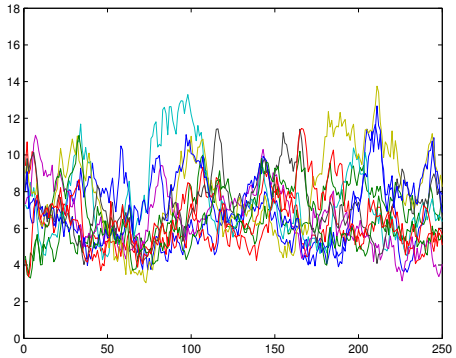
(a) Metropolis (0.17 seconds/sample)



(b) PGBS (0.12 seconds/sample)



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

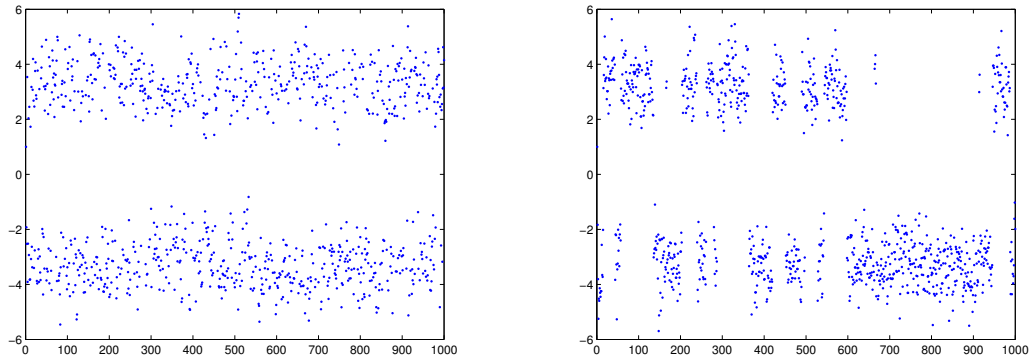


(d) Embedded HMM (0.81 seconds/sample)

Figure 5.2: Estimated autocorrelation times for each latent variable for Model 1, adjusted for computation time

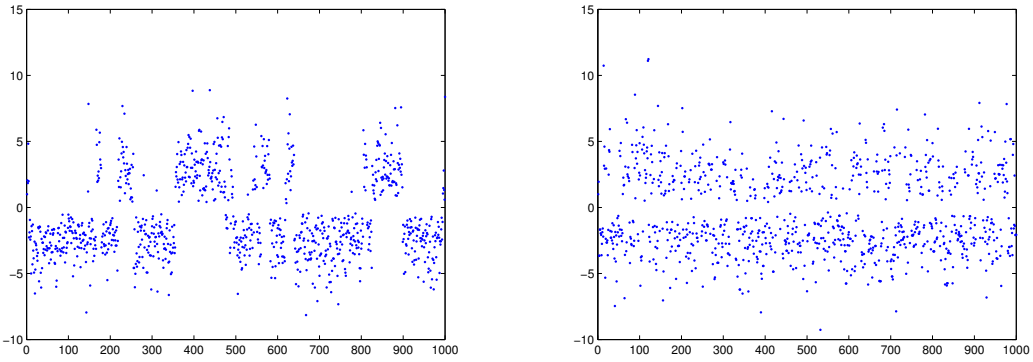
expect to see four modes (with all possible combinations of signs). This is indeed the case for both the PGBS with Metropolis and Embedded HMM samplers. Next, we look at how efficiently the latent variables are sampled. Of particular interest are the latent variables with well-separated modes, since 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 5.3 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 5.4. In this case, we can see that the PGBS



(a) Embedded HMM (b) Particle Gibbs with Metropolis

Figure 5.3: Comparison of Samplers for Model 2, $x_{1,300}$



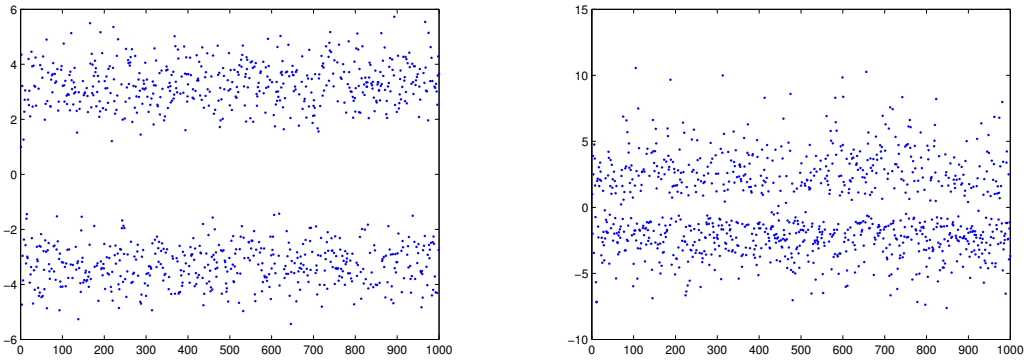
(a) Embedded HMM (b) Particle Gibbs with Metropolis

Figure 5.4: Comparison of Samplers for Model 2, $x_{3,208}x_{4,208}$

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, 80000, versus 80 for the embedded HMM method, works to the advantage 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 5.5. 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 10000 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



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

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

Figure 5.5: Combination of embedded HMM and PGBS with Metropolis samplers

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.

5.7 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. 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. 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 Chapter 3. 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 Chapter 4.

One might also wonder whether it is possible to use the entire current state of x in constructing the pool state density at a given time. It is not obvious how (or if it is possible) to overcome this limitation. 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.

At each time $i < n$, the pool state generation procedure does not depend on the 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, the removal of this constraint 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.

Chapter 6

Conclusion

This thesis has presented several MCMC methods that can be used to perform efficient Bayesian inference in non-linear, non-Gaussian state space models.

The main contribution of the thesis was to show how ensemble MCMC methods can be used to make sampling more efficient in non-linear, non-Gaussian state space models, where properties of the posterior make sampling difficult. We have considered cases where there is strong serial dependence amongst the latent variables, the parameter values are strongly dependent on the current value of the sequence, the state space is high-dimensional, and different observations are more or less informative about the value of the latent state.

While we gave several proposals, the most challenging question for ensemble methods remains finding a general method for selecting pool states that works in an even wider range of scenarios. We have shown that a variety of methods work well when the state space is one-dimensional. Chapter 5 is an attempt at finding a generic method which works well for higher dimensional state spaces by combining ideas from embedded HMMs and Particle Gibbs with Backwards Sampling methods. However, this method is still hard to scale to very high dimensional state spaces, the primary reason being that the proposed pool states at each time are not constrained by all of the data, which becomes more crucial in high dimensions. Using less data in generating a proposal removes a constraint on how the pool states can be placed, potentially allowing for larger moves, but at the same time leading to the possibility that the resulting ensemble has many sequences with low posterior density.

The other question which we didn't explore in much detail is constructing ensembles over parameter values. The basic technique we used is caching, where part of the computation is saved and then used later to efficiently compute values of the posterior at different parameter values. It would be of interest to explore other ways in which param-

eter ensembles can be constructed. One idea worth exploring is constructing ensembles that depend on parameter values, such as in the stochastic volatility example. If we use a scheme where we alternate between sampling sequences and parameters, then using the current values of the parameters can be helpful in generating a set of sequences with high posterior density, especially in cases where there is potential for sequences to be too widely dispersed. We have only looked at ensembles which depend on two parameter values, a current and proposed one, whereas ensembles depending on many parameter values at once can be considered instead.

We have also looked at how models doable with ABC can be done with MCMC. It would be interesting to see how, in general, this can be done for other models, where it is possible to come up with a latent variable representation of the entire data generating process and use an MCMC method to sample from the posterior over all of the latent variables. Since the data generation process can involve many highly-dependent variables, having efficient MCMC methods for inference in such models would be necessary.

Other ways of using ensembles are also worth exploring. In this thesis, we used two types of ensembles, one made possible by efficient forward-backward computations and the other by a caching technique. Perhaps there are other models where ensembles can be constructed and used to improve sampling efficiency.

Chapter 7

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.

Blum, M.G.B. and Francois, O. (2010) "Non-linear regression models for Approximate Bayesian Computation", *Statistics and Computing*, vol. 20, pp. 63-73.

Bonassi, F.V. (2013) "Approximate Bayesian Computation for Complex Dynamic Systems". Ph.D. Thesis, Department of Statistical Science, Duke University.

Christen, J. and Fox, C. (2005) "Markov Chain Monte Carlo methods using an approximation", *Journal of Computational and Graphical Statistics*, vol. 14, pp. 795-810.

Fearnhead, P., Prangle, D. (2012) "Constructing summary statistics for approximate Bayesian computation: semi-automatic approximate Bayesian computation", *Journal of the Royal Statistical Society B*, vol. 74, pp. 1-28.

Geyer, C. J. (2003) "The Metropolis-Hastings-Green Algorithm", http://www.stat.umn.edu/geyer/f05/8931/bmhg.pdf

Kastner, G. and Fruhwirth-Schnatter, S. (2014) "Ancillarity-sufficiency interweaving strategy (ASIS) for boosting MCMC estimation of stochastic volatility models", *Computational Statistics & Data Analysis*, vol. 76, pp. 408-423.

Kim, S., Shephard, N. and Chib, S. (1998) "Stochastic volatility: likelihood inference and comparison with ARCH models", *Review of Economic Studies*. vol. 65, pp. 361-393.

Leman, S.C., Chen, Y., and Lavine, M. (2009) "The multiset sampler", *Journal of the American Statistical Association*, vol. 104, pp. 1029-1041.

Lindsten, F.; Schon, 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.

Lindsten, F. and Schon, T. B. (2013) "Backward simulation methods for Monte Carlo statistical inference", *Foundations and Trends in Machine Learning*. vol. 6(1), pp. 1-143.

Liu, J.S., Liang, F., and Wong, W.H. (2000) "The multiple-try method and local optimization in Metropolis sampling", *Journal of the American Statistical Association*, vol. 95, pp. 121-134.

Liu, J.S. and Sabatti, C. (2000) "Generalized Gibbs sampler and multigrid Monte Carlo for Bayesian computation", *Biometrika*, vol. 87, pp. 353-369.

Liu, J.S. and Wu, Y.N. (1999) "Parameter expansion for data augmentation", *Journal of the American Statistical Association* vol. 94, pp. 1264-1274.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., and Teller, E. (1953) "Equation of State Calculations by Fast Computing Machines". *Journal of Chemical Physics*, vol. 21, pp. 1087-1092.

Neal, R. M. (1993) "Probabilistic Inference Using Markov Chain Monte Carlo Methods", Technical Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto.

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.

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.

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.

Neal, R. M. (2006) "Constructing Efficient MCMC Methods Using Temporary Mapping and Caching", Talk at Columbia University.

Neal, R. M. (2010) "MCMC Using Ensembles of States for Problems with Fast and Slow Variables such as Gaussian Process Regression", Technical Report No. 1011, Department of Statistics, University of Toronto, http://arxiv.org/abs/1101.0387.

Omori, Y., Chib, S., Shephard, N. and Nakajima, J. (2007) "Stochastic volatility model with leverage: fast and efficient likelihood inference", *Journal of Econometrics*, vol. 140-2, pp. 425-449.

Petris, G., Petrone, S. and Campagnoli, P. (2009) *Dynamic Linear Models with R*, Springer: New York.

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.

Scharth, M. and Kohn, R. (2013) "Particle Efficient Importance Sampling", arXiv preprint 1309.6745v1.

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

Shestopaloff, A. Y. and Neal, R. M. (2013b) "On Bayesian inference for the M/G/1 queue with efficient MCMC sampling", Technical Report, http://arxiv.org/abs/1401.5548.

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.

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, http://arxiv.org/abs/1602.06030.

Steven L. Scott (2002) "Bayesian Methods for Hidden Markov Models: Recursive Computing in the 21st Century", *Journal of the American Statistical Association*. vol. 97, no. 457, pp. 337-351.

Wood, S. (2010) "Statistical inference for noisy nonlinear ecological dynamic systems", *Nature*. vol. 466, pp. 1102-1104.

Yu, Y. and Meng, X. (2011) "To Center or Not to Center, That is Not the Question: An Ancillarity-Sufficiency Interweaving Strategy (ASIS) for Boosting MCMC Efficiency", *Journal of Computational and Graphical Statistics*, vol. 20 (2011), pp. 531-570.