Towards Smooth Particle Filters for Likelihood Estimation with Multivariate Latent Variables

by

Anthony Lee

B.Sc., The University of British Columbia, 2006

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

August, 2008

Abstract

In parametrized continuous state-space models, one can obtain estimates of the likelihood of the data for fixed parameters via the Sequential Monte Carlo methodology. Unfortunately, even if the likelihood is continuous in the parameters, the estimates produced by practical particle filters are not, even when common random numbers are used for each filter. This is because the same resampling step which drastically reduces the variance of the estimates also introduces discontinuities in the particles that are selected across filters when the parameters change.

When the state variables are univariate, the methodology of [23] gives an estimator of the log-likelihood that is continuous in the parameters. We present a non-trivial generalization of this method using tree-based $o(N^2)$ (and as low as $O(N \log N)$) resampling schemes that induce significant correlation amongst the selected particles across filters. In turn, this reduces the variance of the difference between the likelihood evaluated for different values of the parameters and the resulting estimator is considerably smoother than naively running the filters with common random numbers.

Importantly, in practice our methods require only a change to the *resample* operation in the SMC framework without the addition of any extra parameters and can therefore be used for any application in which particle filters are already used. In addition, excepting the optional use of interpolation in the schemes, there are no regularity conditions for their use although certain conditions make them more advantageous.

In this thesis, we first introduce the relevant aspects of the SMC methodology to the task of likelihood estimation in continuous state-space models and present an overview of work related to the task of smooth likelihood estimation. Following this, we introduce theoretically correct resampling schemes that cannot be implemented and the practical tree-based resampling schemes that were developed instead. After presenting the performance of our schemes in various applications, we show that two of the schemes are asymptotically consistent with the theoretically correct but unimplementable methods introduced earlier. Finally, we conclude the thesis with a discussion.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Acknowledgements

I thank my supervisor, Arnaud Doucet, for his help, advice and wisdom during the course of this degree. I also thank Nando de Freitas, who in addition to being my second reader also introduced me to the field of machine learning when I was an undergraduate and impressed upon me the notion that statistics provides a meaningful way to build and measure machines that learn. More broadly, I would like to thank anyone and everyone who has taught me something about anything. I am grateful for the financial support I received from the Natural Sciences and Engineering Research Council of Canada, which has allowed me to pursue this work. Finally, I would like to thank my family and friends for their love and support throughout the years.

Chapter 1

Introduction

Sequential Monte Carlo (SMC) methods are effective tools for sampling from high-dimensional probability distributions. However, in many situations we are interested in parameter estimation and not the latent variables generated by such methods. While SMC methods are still useful in these situations, a major drawback is that the likelihood estimates given are far from smooth. Assume we have data $\mathbf{y}$, latent variables $\mathbf{x}$ and parameters θ along with a model that specifies the probability density $p(\mathbf{x}, \mathbf{y}|\theta)$. Furthermore, assume that the marginal likelihood $p(\mathbf{y}|\theta)$ is continuous in θ. For an N-sample Monte Carlo estimator $\hat{L}_N(\theta)$ of this marginal likelihood to be smooth we require it to be continuous in θ as well. Additionally, we want $\hat{L}_N(\theta)$ to be consistent.

In the SMC framework, we often obtain an estimate of $\log p(\mathbf{y}|\theta)$ by running a particle filter with fixed parameter θ. Unfortunately, the interaction of the particles or *resampling* step renders such estimates non-smooth in θ even if each filter is run with common random numbers. In particular, sequential importance sampling *without* resampling provides smooth but prohibitively high-variance estimates of the log-likelihood using common random numbers. The resampling step, on the other hand, can typically be viewed as the use of the inverse transform on the empirical cumulative distribution function of the particle indices. As the weights of the particles change with θ, the same random numbers will lead to the selection of a particle with a different but 'close' index. Unfortunately, particles with close indices are not generally close themselves and so this introduces large differences in the resampled particles as θ changes. Of course, it is this same resampling step that allows us to estimate the log-likelihood with reasonable variance and so discarding this crucial element of SMC methodology for the sake of smoothness is not really an option.

It may be impossible to produce a tractable estimator with both these properties in the general case. For continuous state-space models, a particle filter has already been produced that possesses these properties in the case where the state variables are univariate [23]. This thesis attempts to tackle the problem in the case where the state variables are multivariate. While the algorithms presented do not satisfy the smoothness criteria defined above, they provide significant improvements in smoothness over the generic particle filter algorithms they extend and run in $o(N^2)$ time, which is faster than other methods that also attempt to solve this problem.

The rest of this thesis is organized as follows. Chapter 2 provides a brief overview of the statistical

concepts required to understand the material that follows. Chapter 3 begins the treatment smooth likelihood estimation in earnest, with a discussion of related work as well as the introduction of theoretical algorithms we would like to approximate. In chapter 4, we present novel practical algorithms utilizing tree-based resampling. In chapter 5, we apply the new algorithms to a variety of application domains and show our results. Chapter 6 provides some analysis of the algorithms and chapter 7 concludes the thesis with a discussion.

Chapter 2

Sequential Monte Carlo Methods and Likelihood Evaluation

This chapter provides the reader with a brief overview of the statistical concepts required to understand the rest of the thesis. While the reader is expected to be familiar with the most basic concepts in probability, advanced knowledge of Monte Carlo methods is not required as the relevant details are covered here. If a more rigorous treatment of the material is desired, such treatment is available in [29] and [26].

2.1 Likelihood Estimation

A parametric model is a set of distributions that can be parametrized by a finite number of parameters. For continuous distributions, each member of such a set can be represented by a density $p(\cdot|\theta)$, where θ belongs to a finite-dimensional parameter space Θ. Given a set of parametrized densities and some fixed data $\mathbf{y}$, the likelihood function $p(\mathbf{y}|\theta)$ is a function of θ.

It is often the case that we want to evaluate $p(\mathbf{y}|\theta)$ for various values of θ. Unfortunately, in many models it is impossible to compute this quantity directly. However, in some cases it is possible to introduce a latent random variable $\mathbf{x}$ and evaluate $p(\mathbf{x}, \mathbf{y}|\theta)$. Integrating out $\mathbf{x}$ can yield an expression for $p(\mathbf{y}|\theta)$

$$p(\mathbf{y}|\theta) = \int_{\mathcal{X}} p(\mathbf{x}, \mathbf{y}|\theta) d\mathbf{x}$$

If we could compute this integral analytically we would have an exact expression for $p(\mathbf{y}|\theta)$. In general, however, this integral can only be evaluated numerically. When $\mathbf{x}$ is high-dimensional, Monte Carlo methods are often used to approximate integrals of this type.

2.2 Monte Carlo Methods

Monte Carlo integration is based on the understanding that the expectation of function of a random variable is an integral. Indeed, the expectation of the function $\phi : \mathcal{X} \to \mathbb{R}$ evaluated at random variable $X \in \mathcal{X}$ with probability density function π is

$$E_\pi[\phi(X)] = \int_{\mathcal{X}} \phi(\mathbf{x})\pi(\mathbf{x})d\mathbf{x}$$

Importantly, we can use samples $\{\mathbf{x}_i\}_{i=1}^N$from the distribution defined by π to approximate this integral with the empirical average

$$I_1 = \frac{1}{N}\sum_{i=1}^{N} \phi(\mathbf{x}_i)$$

which has variance

$$\text{var}[I_1] = \frac{1}{N}\left\{E_\pi[\phi(\mathbf{x})^2] - E_\pi[\phi(\mathbf{x})]^2\right\}$$

An extension of classical Monte Carlo integration is importance sampling, in which we sample instead from a proposal distribution γ with the requirement that $\gamma(\mathbf{x}) > 0$ whenever $\pi(\mathbf{x}) > 0$:

$$E_\pi[\phi(X)] = \int_{\mathcal{X}} \phi(\mathbf{x})\frac{\pi(\mathbf{x})}{\gamma(\mathbf{x})}\gamma(x)d\mathbf{x}$$

This yields the approximation

$$I_2 = \frac{1}{N}\sum_{i=1}^{N} \phi(\mathbf{x}_i)\frac{\pi(\mathbf{x}_i)}{\gamma(\mathbf{x}_i)}$$

The variance of this estimate is given by

$$\begin{aligned}
\text{var}[I_2] &= E_\gamma[I_2^2] - E_\gamma[I_2]^2 \\
&= \frac{1}{N}\left\{\int_{\mathcal{X}} \phi(\mathbf{x})^2\frac{\pi(\mathbf{x})^2}{\gamma(\mathbf{x})^2}\gamma(\mathbf{x})d\mathbf{x} - \left(\int_{\mathcal{X}} \phi(\mathbf{x})\frac{\pi(\mathbf{x})}{\gamma(\mathbf{x})}\gamma(\mathbf{x})d\mathbf{x}\right)^2\right\} \\
&= \frac{1}{N}\left\{E_\pi[\phi(\mathbf{x})^2\frac{\pi(\mathbf{x})}{\gamma(\mathbf{x})}] - E_\pi[\phi(\mathbf{x})]^2\right\}
\end{aligned}$$

For both classical Monte Carlo and importance sampling, the estimates are unbiased and converge almost surely via the Strong Law of Large Numbers. In practice, importance sampling can be used as a variance reduction technique or because γ is much easier to sample from. For our purposes, the

latter is most often the case, and care should be taken to ensure that $E_\pi[\phi(x)^2 \frac{\pi(x)}{\gamma(x)}] < \infty$ so that the estimate has at least finite variance.

It is important to note that often we only know π and even γ up to a normalizing constant, ie. we know $\pi^*(\mathbf{x}) \propto \pi(\mathbf{x})$ and $\gamma^*(\mathbf{x}) \propto \gamma(\mathbf{x})$ where $\int_\mathcal{X} \pi(\mathbf{x})d\mathbf{x} = \int_\mathcal{X} \gamma(\mathbf{x})d\mathbf{x} = 1$. In this case we can use the estimate

$$I_2^* = \frac{\sum_{i=1}^N \phi(\mathbf{x}_i)w^*(\mathbf{x}_i)}{\sum_{i=1}^N w^*(\mathbf{x}_i)}$$

where

$$w^*(\mathbf{x}_i) = \frac{\pi^*(\mathbf{x}_i)}{\gamma^*(\mathbf{x}_i)}$$

Alternatively, we can write this as

$$I_2^* = \sum_{i=1}^N \phi(\mathbf{x}_i)W^{(i)}$$

where

$$W^{(i)} = \frac{w^*(\mathbf{x}_i)}{\sum_{j=1}^N w^*(\mathbf{x}_j)}$$

This estimate is not unbiased but it is asymptotically consistent since the squared bias is of order $O(N^{-2})$ and the variance is of order $O(N^{-1})$.

Let us also define the empirical density

$$P_N(d\mathbf{x}) = \sum_{i=1}^N W^{(i)}\delta_{\mathbf{x}_i}(d\mathbf{x})$$

When $\mathbf{x}$ is univariate, we can define the weighted empirical distribution function

$$F_N(x) = \sum_{i=1}^N W^{(i)}\mathbb{I}(x_i \leq x)$$

where

$$\mathbb{I}(x_i \leq x) = \begin{cases} 1 & \text{if } x_i \leq x, \\ 0 & \text{if } x_i > x. \end{cases}$$

In fact, since we can set $\phi(x) = \mathbb{I}(x_i \leq a)$, we have that

$$F_N(a) = \sum_{i=1}^{N} W^{(i)} \mathbb{I}(x_i \leq a)$$

is an asymptotically consistent estimator of

$$F(a) = \int \mathbb{I}(x \leq a)\pi(x)dx$$

ie. P_N converges in distribution to π.

In the case of multivariate $\mathbf{x}$, we also have convergence in distribution since

$$P_N(\mathbf{x} \in S) = \sum_{i=1}^{N} W^{(i)} \mathbb{I}(\mathbf{x}_i \in S)$$

is an asymptotically consistent estimator of

$$\Pr_\pi[\mathbf{x} \in S] = \int \mathbb{I}(\mathbf{x} \in S)\pi(\mathbf{x})d\mathbf{x} = \int_S \pi(\mathbf{x})d\mathbf{x}$$

2.3 State-Space Models

In this thesis, we will restrict our discussion to a particular class of models that is widely applicable and commonly associated with the sequential Monte Carlo methodology. It should be stressed that sequential Monte Carlo is not itself restricted to models of this form.

Consider a time-homogeneous Markovian state-space model with hidden states $\{\mathbf{x}_t : t \in \{0, \ldots, T\}\}$, $\mathbf{x}_t \in \mathcal{X}$ and observations $\{\mathbf{y}_t : t \in \{0, \ldots, T\}\}, \mathbf{y}_t \in \mathcal{Y}$. The model is given by

$$\begin{aligned} p(\mathbf{x}_0|\theta) & \qquad \text{(initial state)} \\ p(\mathbf{x}_t|\mathbf{x}_{t-1}, \theta) \text{ for } 1 \leq t \leq T & \qquad \text{(evolution)} \\ p(\mathbf{y}_t|\mathbf{x}_t, \theta) \text{ for } 0 \leq t \leq T & \qquad \text{(observation)} \end{aligned}$$

The defining characteristic in such models is that the observation $\mathbf{y}_t$ is conditionally independent of the other observations given the latent variable $\mathbf{x}_t$. In practice, this allows for the expression of a large number of possible systems. Intuitively, this type of model is natural if $\mathbf{x}_t$ represents the state of the system at time t, the transition from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$ is captured in the evolution distribution and the observation at time t depends only on the state of the system at that time.

Likelihood Evaluation

For this class of models, likelihood evaluation is complicated by the presence of latent variables $\mathbf{x}_{0:T}$. A first attempt at computing $p(\mathbf{y}_{0:T}|\theta)$ could be to decompose this probability as

$$\begin{aligned} p(\mathbf{y}_{0:T}|\theta) &= \int p(\mathbf{x}_{0:T}, \mathbf{y}_{0:T}|\theta) d\mathbf{x}_{0:T} \\ &= \int p(\mathbf{y}_{0:T}|\mathbf{x}_{0:T}, \theta) p(\mathbf{x}_{0:T}|\theta) d\mathbf{x}_{0:T} \\ &= \int \left(\prod_{i=0}^{T} p(\mathbf{y}_i|\mathbf{x}_i, \theta) \right) p(\mathbf{x}_{0:T}|\theta) d\mathbf{x}_{0:T} \end{aligned}$$

and proceed with a Monte Carlo estimate by sampling $\mathbf{x}_{0:T} \sim p(\mathbf{x}_{0:T}|\theta) = p(\mathbf{x}_0) \prod_{i=1}^{T} p(\mathbf{x}_i|\mathbf{x}_{i-1})$. Unforunately, while theoretically valid, this estimate has prohibitively high variance whenever the data $\mathbf{y}_{0:T}$ is informative. In general, we would like to incorporate the data when proposing samples $\mathbf{x}_{0:T}$. Using importance sampling, we could conceivably reduce the variance by sampling $\mathbf{x}_{0:T} \sim q(\mathbf{x}_{0:T}|\mathbf{y}_{0:T}, \theta)$ since

$$p(\mathbf{y}_{0:T}|\theta) = \int \left(\prod_{i=0}^{T} p(\mathbf{y}_i|\mathbf{x}_i, \theta) \right) \frac{p(\mathbf{x}_{0:T}|\theta)}{q(\mathbf{x}_{0:T}|\mathbf{y}_{0:T}, \theta)} q(\mathbf{x}_{0:T}|\mathbf{y}_{0:T}, \theta) d\mathbf{x}_{0:T}$$

However, it is difficult to come up with a high-dimensional proposal density q that is easy to sample from that would sufficiently reduce the variance.

An alternative decomposition of the likelihood is attained by noticing that

$$p(\mathbf{y}_{0:T}|\theta) = p(\mathbf{y}_0|\theta) \prod_{k=1}^{T} p(\mathbf{y}_k|\mathbf{y}_{0:k-1}, \theta)$$

where, dropping θ for clarity

$$\begin{aligned} p(\mathbf{y}_k|\mathbf{y}_{0:k-1}) &= \int p(\mathbf{y}_k, \mathbf{x}_{k-1}|\mathbf{y}_{0:k-1}) d\mathbf{x}_{k-1} \\ &= \int p(\mathbf{y}_k|\mathbf{x}_{k-1}) p(\mathbf{x}_{k-1}|\mathbf{y}_{0:k-1}) d\mathbf{x}_{k-1} \\ &= \int \int p(\mathbf{y}_k, \mathbf{x}_k|\mathbf{x}_{k-1}) d\mathbf{x}_k p(\mathbf{x}_{k-1}|\mathbf{y}_{0:k-1}) d\mathbf{x}_{k-1} \\ &= \int \int p(\mathbf{y}_k|\mathbf{x}_k) p(\mathbf{x}_k|\mathbf{x}_{k-1}) d\mathbf{x}_k p(\mathbf{x}_{k-1}|\mathbf{y}_{0:k-1}) d\mathbf{x}_{k-1} \end{aligned}$$

Therefore, if we can produce samples from $p(\mathbf{x}_{k-1}|\mathbf{y}_{0:k-1})$ we can estimate each $p(\mathbf{y}_k|\mathbf{y}_{0:k-1})$ via

$$\hat{p}(\mathbf{y}_k|\mathbf{y}_{0:k-1}) = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M} p(\mathbf{y}_k|\mathbf{x}_k^{(i,j)}) \tag{2.1}$$

where we define $\mathbf{x}_{k-1}^{(i)}$ as the ith sample from $p(\mathbf{x}_{k-1}|\mathbf{y}_{0:k-1})$ and $\mathbf{x}_k^{(i,j)}$ as the jth sample from the distribution with density $p(\mathbf{x}_k|\mathbf{x}_{k-1}^{(i)})$. We will see that sequential Monte Carlo allows us to sample approximately from $p(\mathbf{x}_{k-1}|\mathbf{y}_{0:k-1})$.

Alternatively, we can use an importance distribution $q(\mathbf{x}_k|\mathbf{y}_k, \mathbf{x}_{k-1})$ within the inner integral:

$$\begin{aligned} p(\mathbf{y}_k|\mathbf{y}_{0:k-1}) &= \int\int p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{x}_{k-1})d\mathbf{x}_k p(\mathbf{x}_{k-1}|\mathbf{y}_{0:k-1})d\mathbf{x}_{k-1} \\ &= \int\int \frac{p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{x}_{k-1})}{q(\mathbf{x}_k|\mathbf{y}_k, \mathbf{x}_{k-1})} q(\mathbf{x}_k|\mathbf{y}_k, \mathbf{x}_{k-1})d\mathbf{x}_k p(\mathbf{x}_{k-1}|\mathbf{y}_{0:k-1})d\mathbf{x}_{k-1} \end{aligned}$$

yielding

$$\hat{p}(\mathbf{y}_k|\mathbf{y}_{0:k-1}) = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M} \frac{p(\mathbf{y}_k|\mathbf{x}_k^{(i,j)})p(\mathbf{x}_k^{(i,j)}|\mathbf{x}_{k-1}^{(i)})}{q(\mathbf{x}_k^{(i,j)}|\mathbf{y}_k, \mathbf{x}_{k-1}^{(i)})} \tag{2.2}$$

where again $\mathbf{x}_{k-1}^{(i)}$ is the ith sample from $p(\mathbf{x}_{k-1}|\mathbf{y}_{0:k-1})$ but $\mathbf{x}_k^{(i,j)}$ is the jth sample from the distribution with density $q(\mathbf{x}_k|\mathbf{y}_k, \mathbf{x}_{k-1}^{(i)})$.

2.4 Sequential Monte Carlo

In order to appreciate the problem of smooth likelihood estimation using particle filters, it is necessary to specify the particle filtering method so that we can highlight the difficulties that are faced. Details are omitted that are not directly relevant to smooth likelihood estimation for state-space models and so the reader is directed to [8] for a more general overview of sequential Monte Carlo methods.

Given a target distribution $p(\mathbf{x}_{0:T}|\mathbf{y}_{0:T})$, the central idea in sequential Monte Carlo algorithms is to define a sequence of intermediate target distributions $p(\mathbf{x}_0|\mathbf{y}_0), p(\mathbf{x}_{0:1}|\mathbf{y}_{0:1}), \ldots, p(\mathbf{x}_{0:T}|\mathbf{y}_{0:T})$ that move 'smoothly' towards the target distribution. Importantly, we can evaluate each intermediate distribution up to a normalizing constant since

$$p(\mathbf{x}_{0:t}|\mathbf{y}_{0:t}) = \frac{p(\mathbf{x}_{0:t}, \mathbf{y}_{0:t})}{p(\mathbf{y}_{0:t})}$$

and $p(\mathbf{x}_{0:t}, \mathbf{y}_{0:t})$ is readily computable. We utilize an importance distribution of the form

$$q(\mathbf{x}_{0:T}|\mathbf{y}_{0:T}) = q(\mathbf{x}_0|\mathbf{y}_0)\prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t)$$

2.4.1 Sequential Importance Sampling

The generic sequential importance sampling (SIS) algorithm using these distributions is given in Algorithm 1.

Algorithm 1 The generic sequential importance sampling algorithm

1. At time $t = 0$.
   - For $i = 1, \ldots, N$, sample $\mathbf{x}_0^{(i)} \sim q(\mathbf{x}_0|\mathbf{y}_0)$
   - For $i = 1, \ldots, N$, evaluate the importance weights:

$$w_0(\mathbf{x}_0^{(i)}) = \frac{p(\mathbf{x}_0^{(i)}, \mathbf{y}_0)}{q(\mathbf{x}_0^{(i)}|\mathbf{y}_0)} = \frac{p(\mathbf{y}_0|\mathbf{x}_0^{(i)})p(\mathbf{x}_0^{(i)})}{q(\mathbf{x}_0^{(i)}|\mathbf{y}_0)}$$

2. For times $t = 1, \ldots, T$.
   - For $i = 1, \ldots, N$, sample $\mathbf{x}_t^{(i)} \sim q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1}^{(i)})$ and set $\mathbf{x}_{0:t}^{(i)} \stackrel{\text{def}}{=} (\mathbf{x}_{0:t-1}^{(i)}, \mathbf{x}_t^{(i)})$
   - For $i = 1, \ldots, N$, evaluate the importance weights:

$$w_t(\mathbf{x}_{0:t}^{(i)}) = w_{t-1}(\mathbf{x}_{0:t-1}^{(i)})\frac{p(\mathbf{x}_{0:t}^{(i)}, \mathbf{y}_{0:t})}{p(\mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{0:t-1})q(\mathbf{x}_t^{(i)}|\mathbf{y}_t, \mathbf{x}_{t-1}^{(i)})} = w_{t-1}(\mathbf{x}_{0:t-1}^{(i)})\frac{p(\mathbf{y}_t|\mathbf{x}_t^{(i)})p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}^{(i)})}{q(\mathbf{x}_t^{(i)}|\mathbf{y}_t, \mathbf{x}_{t-1}^{(i)})}$$

We can prove by induction that the importance weights in this algorithm satisfy

$$w_t(\mathbf{x}_{0:t}^{(i)}) = \frac{p(\mathbf{x}_{0:t}^{(i)}, \mathbf{y}_{0:t})}{q(\mathbf{x}_{0:t}^{(i)}|\mathbf{y}_{0:t})}$$

where $\mathbf{x}_{0:t}^{(i)} \sim q(\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$. Indeed, we have

$$\begin{aligned} w_t(\mathbf{x}_{0:t}^{(i)}) &= w_{t-1}(\mathbf{x}_{0:t-1}^{(i)})\frac{p(\mathbf{x}_{0:t}^{(i)}, \mathbf{y}_{0:t})}{p(\mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{0:t-1})q(\mathbf{x}_t^{(i)}|\mathbf{y}_t, \mathbf{x}_{t-1}^{(i)})} \\ &= \frac{p(\mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{0:t-1})}{q(\mathbf{x}_{0:t-1}^{(i)}|\mathbf{y}_{0:t-1})}\frac{p(\mathbf{x}_{0:t}^{(i)}, \mathbf{y}_{0:t})}{p(\mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{0:t-1})q(\mathbf{x}_t^{(i)}|\mathbf{y}_t, \mathbf{x}_{t-1}^{(i)})} \\ &= \frac{p(\mathbf{x}_{0:t}^{(i)}, \mathbf{y}_{0:t})}{q(\mathbf{x}_{0:t}^{(i)}|\mathbf{y}_{0:t})} \end{aligned}$$

Note also that

$$w_t(\mathbf{x}_{0:t}^{(i)}) = \frac{p(\mathbf{x}_{0:t}^{(i)}, \mathbf{y}_{0:t})}{q(\mathbf{x}_{0:t}^{(i)}|\mathbf{y}_{0:t})} = p(\mathbf{y}_{0:t})\frac{p(\mathbf{x}_{0:t}^{(i)}|\mathbf{y}_{0:t})}{q(\mathbf{x}_{0:t}^{(i)}|\mathbf{y}_{0:t})} \propto \frac{p(\mathbf{x}_{0:t}^{(i)}|\mathbf{y}_{0:t})}{q(\mathbf{x}_{0:t}^{(i)}|\mathbf{y}_{0:t})}$$

Since $p(\mathbf{y}_{0:t})$ is an unknown normalizing constant[1], we are forced to normalize the importance weights as in normalized importance sampling. This gives the following empirical distribution to approximate $p(\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$:

$$\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t}) = \sum_{i=1}^{N} W_t^{(i)} \delta_{\mathbf{x}_{0:t}^{(i)}}(d\mathbf{x}_{0:t})$$

where

$$W_t^{(i)} = \frac{w_t(\mathbf{x}_{0:t}^{(i)})}{\sum_{j=1}^{N} w_t(\mathbf{x}_{0:t}^{(j)})}$$

While theoretically valid, for practical values of N this algorithm suffers from degeneracy for even small values of t due to the unavoidable increase in variance of the importance weights over time. This means that the estimate $\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$ has significant mass on only a few particles, regardless of the shape of the distribution $p(\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$, and this leads to prohibitively high variance of Monte Carlo estimates using these particles.

2.4.2 Sequential Importance Sampling/Resampling

To alleviate the problem of degeneracy, we can replace intermediate weighted empirical distributions with unweighted empirical distributions such that none of the relative weights are negligible. More formally, we replace

$$\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t}) = \sum_{i=1}^{N} W_t^{(i)} \delta_{\mathbf{x}_{0:t}^{(i)}}(d\mathbf{x}_{0:t})$$

with

$$\tilde{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t}) = \frac{1}{N}\sum_{i=1}^{N} n_t^{(i)} \delta_{\mathbf{x}_{0:t}^{(i)}}(d\mathbf{x}_{0:t})$$

where $n_t^{(i)} \in \{0, 1, \ldots, N\}$ and $\sum_{i=1}^{N} n_t^{(i)} = N$.

[1] If we could compute this, we could evaluate $p(\mathbf{y}_{0:T}) = p(\mathbf{y}_{0:T}|\theta)$ without using SMC.

A replacement, or resampling, scheme is acceptable if for any function $\phi(\cdot)$ we have

$$E\left[\left(\int \phi(\mathbf{x}_{0:t})\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t}) - \int \phi(\mathbf{x}_{0:t})\tilde{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t})\right)^2\right] \overset{N\to\infty}{\longrightarrow} 0$$

since convergence of the expectation of ϕ with respect to the empirical density $\tilde{P}_N$ to the true expectation is then guaranteed as $N \to \infty$.

A variety of common resampling techniques are discussed in [15]. The general idea of many resampling algorithms is to generate a set of N uniform random variables $u_1, \ldots, u_N$ (not necessarily identically distributed) and use them to sample N values of $\mathbf{x}_{0:t}$ from $\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$ such that

$$E[n_t^{(i)}] = NW_t^{(i)}$$

The variable $n_t^{(i)}$ corresponds to the number of times $\mathbf{x}_{0:t}^{(i)}$ is selected. It is vital to note that the resampling process is a source of flexibility within the sequential importance sampling/resampling framework. In fact, we will later see that it is this process that is responsible for the lack of smoothness in likelihood estimation and which can be modified to alleviate this problem.

The sequential importance sampling algorithm augmented with a resampling (SIR) procedure is given in Algorithm 2. Although we no longer have

$$w_t(\mathbf{x}_{0:t}^{(i)}) \propto \frac{p(\mathbf{x}_{0:t}^{(i)}|\mathbf{y}_{0:t})}{q(\mathbf{x}_{0:t}^{(i)}|\mathbf{y}_{0:t})}$$

we know that our resulting empirical distribution still converges in distribution[2] to $p(\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$ and that Monte Carlo estimates of expectations of test statistics (such as the likelihood estimators in equations 2.1 and 2.2) converge in probability to their true values. Furthermore, at each step we have a greater diversity of particles than with the original algorithm. The problem of degeneracy is not removed: at time t the marginals $p(\mathbf{x}_i|\mathbf{y}_{0:t})$ for $i << t$ are poorly represented. However, the marginal $p(\mathbf{x}_t|\mathbf{y}_{0:t})$ is well represented and this is the marginal that is used for likelihood estimation and the generation of particles from $p(\mathbf{x}_{t+1}|\mathbf{y}_{0:t})$. In practice, the resampling step is what permits the importance sampling/resampling algorithm to give reasonable estimates for practical values of N.

[2] This thesis is not particularly concerned with extolling the virtues of particle filters in general, and the algorithms presented are only intended to be used when SMC methods are effective. As such, more sophisticated results about the convergence of particle filters are not discussed. However, [6] provides a survey of convergence results for those who are interested.

Algorithm 2 The generic sequential importance sampling/resampling algorithm

1. At time $t = 0$.
   - For $i = 1, \ldots, N$, sample $\tilde{\mathbf{x}}_0^{(i)} \sim q(\mathbf{x}_0|\mathbf{y}_0)$
   - For $i = 1, \ldots, N$, evaluate the importance weights:
   $$w_0(\tilde{\mathbf{x}}_0^{(i)}) = \frac{p(\tilde{\mathbf{x}}_0^{(i)}, \mathbf{y}_0)}{q(\tilde{\mathbf{x}}_0^{(i)}|\mathbf{y}_0)} = \frac{p(\mathbf{y}_0|\tilde{\mathbf{x}}_0^{(i)})p(\tilde{\mathbf{x}}_0^{(i)})}{q(\tilde{\mathbf{x}}_0^{(i)}|\mathbf{y}_0)}$$
   - For $i = 1, \ldots, N$, normalize the importance weights:
   $$W_0^{(i)} = \frac{w_0(\tilde{\mathbf{x}}_0^{(i)})}{\sum_{j=1}^N w_0(\tilde{\mathbf{x}}_0^{(j)})}$$
   - Resample (with replacement) N particles $\{\mathbf{x}_0^{(i)} : i = 1, \ldots, N\}$ from $\{\tilde{\mathbf{x}}_0^{(i)} : i = 1, \ldots, N\}$ according to the importance weights $\{W_0^{(i)} : i = 1, \ldots, N\}$. Set $w_0^{(i)} = \frac{1}{N}$ for $i = 1, \ldots, N$.
2. For times $t > 0$.
   - For $i = 1, \ldots, N$, sample $\tilde{\mathbf{x}}_t^{(i)} \sim q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1}^{(i)})$ and set $\tilde{\mathbf{x}}_{0:t}^{(i)} \stackrel{\text{def}}{=} (\mathbf{x}_{0:t-1}^{(i)}, \tilde{\mathbf{x}}_t^{(i)})$
   - For $i = 1, \ldots, N$, evaluate the importance weights:
   $$w_t(\tilde{\mathbf{x}}_{0:t}^{(i)}) = w_{t-1}^{(i)} \frac{p(\mathbf{y}_t|\tilde{\mathbf{x}}_t^{(i)})p(\tilde{\mathbf{x}}_t^{(i)}|\mathbf{x}_{t-1}^{(i)})}{q(\tilde{\mathbf{x}}_t^{(i)}|\mathbf{y}_t, \mathbf{x}_{t-1}^{(i)})}$$
   - For $i = 1, \ldots, N$, normalize the importance weights:
   $$W_t^{(i)} = \frac{w_t(\tilde{\mathbf{x}}_{0:t}^{(i)})}{\sum_{j=1}^N w_t(\tilde{\mathbf{x}}_{0:t}^{(j)})}$$
   - Resample (with replacement) N particles $\{\mathbf{x}_{0:t}^{(i)} : i = 1, \ldots, N\}$ from $\{\tilde{\mathbf{x}}_{0:t}^{(i)} : i = 1, \ldots, N\}$ according to the importance weights $\{W_t^{(i)} : i = 1, \ldots, N\}$. Set $w_t^{(i)} = \frac{1}{N}$ for $i = 1, \ldots, N$.

Multinomial Resampling

In resampling, we wish to replace the weighted empirical distribution $\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$ with an unweighted empirical distribution $\tilde{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$. One of the most common resampling schemes employed by practitioners is that of multinomial resampling. In this scheme, we construct the empirical cumulative distribution function

$$\hat{F}_{t,N}(j) = \sum_{i=1}^{j} W_t^{(i)}$$

It is important to note that this is a cumulative distribution function where the random variable of interest is not $X_{0:t}$ but instead the *index* of the particle we wish to choose. We can sample an index according to its weight by defining the inverse of this function as

$$\hat{F}_{t,N}^{-1}(u) = \min\{j : \hat{F}_{t,N}(j) \geq u\},$$

generating a uniform random number on $[0, 1]$ and evaluating the inverse at this value. If we repeat this process N times and increment $n_t^{(i)}$ (with each $n_t^{(i)}$ starting at 0) each time the index i is sampled we end up with a valid $\tilde{P}_N(d\mathbf{x})$. If we denote each uniform random number as u_i we can define

$$k_t^{(i)} = \hat{F}_{t,N}^{-1}(u_i)$$

and call this the selection index that is picked during resampling. Indeed, with the selection index made explicit we have

$$\mathbf{x}_{0:t}^{(i)} = \tilde{\mathbf{x}}_{0:t}^{(k_t^{(i)})}$$

The most relevant issue for the topic at hand is that particles whose indices are close are not necessarily close themselves.

The computational complexity of resampling naively is in $O(N^2)$. However, one can reduce this complexity by generating pre-ordered uniform random numbers as in [1] and traversing an array of cumulative weights without backtracking to give a time complexity of $O(N)$.

An Alternative View of Resampling

Instead of viewing resampling as the replacement of the weighted empirical distribution $\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$ with an unweighted empirical distribution $\tilde{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$, we can view it as the approximation of the true density $p(\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$ with $\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$.

In particular, in the context of likelihood evaluation in time-homogeneous Markovian state-space models we are only interested in the marginal $p(\mathbf{x}_t|\mathbf{y}_{0:t})$ which is approximated by

$$\hat{P}_N(d\mathbf{x}_t|\mathbf{y}_{0:t}) = \sum_{i=1}^{N} W_t^{(i)} \delta_{\mathbf{x}_t^{(i)}}(d\mathbf{x}_t)$$

The predictive distribution $p(\mathbf{x}_{t+1}|\mathbf{y}_{0:t})$ is thus approximated by the density

$$\hat{p}_N(\mathbf{x}_{t+1}|\mathbf{y}_{0:t}) = \sum_{i=1}^{N} W_t^{(i)} p(\mathbf{x}_{t+1}|\mathbf{x}_t^{(i)})$$

We can think of this density as the marginal of a joint distribution

$$\hat{p}_N(k, \mathbf{x}_{t+1}|\mathbf{y}_{0:t}) = W_t^{(k)} p(\mathbf{x}_{t+1}|\mathbf{x}_t^{(k)})$$

The proposal distribution used in sequential importance sampling can then be defined as

$$q_N(\mathbf{x}_{t+1}|\mathbf{x}_t^{(1:N)}, W_t^{(1:N)}, \mathbf{y}_{t+1}) = \sum_{i=1}^{N} W_t^{(i)} q(\mathbf{x}_{t+1}|\mathbf{y}_{t+1}, \mathbf{x}_t^{(i)})$$

This density is the marginal of a joint distribution

$$q_N(k, \mathbf{x}_{t+1}|\mathbf{x}_t^{(1:N)}, W_t^{(1:N)}, \mathbf{y}_{t+1}) = W_t^{(k)} q(\mathbf{x}_{t+1}|\mathbf{y}_{t+1}, \mathbf{x}_t^{(k)})$$

The resampling step then corresponds to (at each stage i) sampling the auxiliary variable k, which is the index into both the mixture representation of $\hat{p}_N(\mathbf{x}_{t+1}|\mathbf{y}_{0:t})$ and $q_N(\mathbf{x}_{t+1}|\mathbf{x}_t^{(1:N)}, W_t^{(1:N)}, \mathbf{y}_{t+1})$. This is followed by sampling $\mathbf{x}_{t+1}^{(i)} \sim q_N(\mathbf{x}_{t+1}|k, \mathbf{x}_t^{(1:N)}, W_t^{(1:N)}, \mathbf{y}_{t+1}) = q(\mathbf{x}_{t+1}|\mathbf{x}_t^{(k)}, \mathbf{y}_{t+1})$ and then evaluating the weight

$$w_{t+1}(\mathbf{x}_{t+1}^{(i)}) = \frac{p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1}^{(i)}) \hat{p}_N(k, \mathbf{x}_{t+1}^{(i)}|\mathbf{y}_{0:t})}{q_N(k, \mathbf{x}_{t+1}^{(i)}|\mathbf{x}_t^{(1:N)}, W_t^{(1:N)}, \mathbf{y}_{t+1})} = \frac{p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1}^{(i)}) p(\mathbf{x}_{t+1}^{(i)}|\mathbf{x}_t^{(k)})}{q(\mathbf{x}_{t+1}^{(i)}|\mathbf{y}_{t+1}, \mathbf{x}_t^{(k)})}$$

It may seem strange to compute the weights using the joint distributions involving the auxiliary variable k since we are essentially sampling $\mathbf{x}_{t+1}^{(i)} \sim q_N(\mathbf{x}_{t+1}|\mathbf{y}_{0:t+1})$. However, evaluating the importance weights using the marginals via

$$w_{t+1}(\mathbf{x}_{t+1}^{(i)}) = \frac{p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1}^{(i)}) \hat{p}_N(\mathbf{x}_{t+1}^{(i)}|\mathbf{y}_{0:t})}{q_N(\mathbf{x}_{t+1}^{(i)}|\mathbf{x}_t^{(1:N)}, W_t^{(1:N)}, \mathbf{y}_{t+1})} = \frac{p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1}^{(i)}) \sum_{j=1}^{N} W_t^{(j)} p(\mathbf{x}_{t+1}^{(i)}|\mathbf{x}_t^{(j)})}{\sum_{j=1}^{N} W_t^{(j)} q(\mathbf{x}_{t+1}^{(i)}|\mathbf{y}_{t+1}, \mathbf{x}_t^{(j)})}$$

as in the marginal particle filter of [17] would take $O(N)$ time per particle[3], giving an $O(N^2)$ time complexity for weighting N particles. It is easily shown that the expectation of any test function ϕ is unchanged by weighting the particles using the joint distributions as opposed to the marginals.

[3] It is possible in some situations to speed up this computation using N-body methods, but usually this requires approximation and sometimes it provides no gains in efficiency at all for a given tolerance.

Letting $\psi = (\mathbf{x}_t^{(1:N)}, W_t^{(1:N)}, \mathbf{y}_{t+1})$, we have:

$$\begin{aligned}
&\int\int \phi(\mathbf{x}_{t+1}) \frac{p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1})\hat{p}_N(k, \mathbf{x}_{t+1}|\mathbf{y}_{0:t})}{q_N(k, \mathbf{x}_{t+1}|\psi)} q_N(k, \mathbf{x}_{t+1}|\psi) d\mathbf{x}_{t+1} dk \\
&= \int\int \phi(\mathbf{x}_{t+1}) \frac{p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1})\hat{p}_N(\mathbf{x}_{t+1}|\mathbf{y}_{0:t})\hat{p}_N(k|\mathbf{x}_{t+1}, \mathbf{y}_{0:t})}{q_N(\mathbf{x}_{t+1}|\psi) q_N(k|\mathbf{x}_{t+1}, \psi)} q_N(\mathbf{x}_{t+1}|\psi) q_N(k|\mathbf{x}_{t+1}, \psi) d\mathbf{x}_{t+1} dk \\
&= \int \phi(\mathbf{x}_{t+1}) \frac{p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1})\hat{p}_N(\mathbf{x}_{t+1}|\mathbf{y}_{0:t})}{q_N(\mathbf{x}_{t+1}|\psi)} q_N(\mathbf{x}_{t+1}|\psi) \int \frac{\hat{p}_N(k|\mathbf{x}_{t+1}, \mathbf{y}_{0:t})}{q_N(k|\mathbf{x}_{t+1}, \psi)} q_N(k|\mathbf{x}_{t+1}, \psi) dk d\mathbf{x}_{t+1} \\
&= \int \phi(\mathbf{x}_{t+1}) \frac{p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1})\hat{p}_N(\mathbf{x}_{t+1}|\mathbf{y}_{0:t})}{q_N(\mathbf{x}_{t+1}|\psi)} q_N(\mathbf{x}_{t+1}|\psi) d\mathbf{x}_{t+1}
\end{aligned}$$

since $\int \hat{p}_N(k|\mathbf{x}_{t+1}, \mathbf{y}_{0:t}) dk = 1$.

2.4.3 Likelihood Evaluation

It should be clear that the estimators for the likelihood given in Equations 2.1 and 2.2 are particularly suitable when particle filters are used to generate the latent variables $\mathbf{x}_{0:T}$. This is because it is the marginals $p(\mathbf{x}_k|\mathbf{y}_{0:k})$, for $k = 0, \ldots, T$ that are used at each step k and these marginals do not suffer from degeneracy in the sequential importance sampling/resampling algorithm.

A computational issue associated with the evaluation of the likelihood as proposed, however, is that the repeated multiplication of terms is prone to numerical inaccuracy. A way to circumvent this problem is to evaluate instead the log-likelihood $\log p(\mathbf{y}_{0:t})$. A biased estimate of this quantity is

$$\hat{l}_1(\mathbf{y}_{0:t}) = \log \hat{p}(\mathbf{y}_0) + \sum_{i=1}^{t} \log \hat{p}(\mathbf{y}_i|\mathbf{y}_{0:i-1})$$

An approximate bias correction by Taylor expansion, as done in [23] gives the improved estimate:

$$\hat{l}_2(\mathbf{y}_{0:t}) = \tilde{l}(\mathbf{y}_0) + \sum_{i=1}^{t} \tilde{l}(\mathbf{y}_i|\mathbf{y}_{0:i-1})$$

where

$$\tilde{l}(\mathbf{y}_0) = \log \hat{p}(\mathbf{y}_0) + \frac{1}{2} \frac{\hat{\sigma}_0^2}{NM\hat{p}(\mathbf{y}_0)^2}$$

and

$$\tilde{l}(\mathbf{y}_k|\mathbf{y}_{0:k-1}) = \log \hat{p}(\mathbf{y}_k|\mathbf{y}_{0:k-1}) + \frac{1}{2} \frac{\hat{\sigma}_k^2}{NM\hat{p}(\mathbf{y}_k|\mathbf{y}_{0:k-1})^2}$$

and $\hat{\sigma}_0^2$ is the sample variance of $\hat{p}(\mathbf{y}_0)$ and $\hat{\sigma}_k^2$ is the sample variance of $\hat{p}(\mathbf{y}_k|\mathbf{y}_{0:k-1})$.

Rationale

Let X be an unbiased estimate of the quantity μ. Then $\bar{X}$, the sample mean of X with R samples, is also an unbiased estimate of μ. Furthermore, if the variance of X is σ^2 then the variance of $\bar{X}$ is $\frac{\sigma^2}{R}$. The second order Taylor expansion of $\log \bar{X}$ at μ is

$$\log \bar{X} \approx \log \mu + \frac{1}{\mu}(\bar{X} - \mu) - \frac{1}{\mu^2}\frac{(\bar{X} - \mu)^2}{2}$$

so we have

$$E[\log \bar{X}] \approx \log \mu - \frac{1}{\mu^2}\frac{E[(\bar{X} - \mu)^2]}{2}$$

and therefore $\log \bar{X} + \frac{\hat{\sigma}^2}{2R\bar{X}^2}$ is an estimate which is good for large R.

Chapter 3

Smooth Likelihood Estimation

3.1 Desired Estimator Properties

We are now ready to consider the smooth likelihood problem. First, we should define formally the three properties that we would like a smooth estimator to have. We would like the estimator $\hat{L}_N(\theta)$ of $p(\mathbf{y}|\theta)$ to be consistent. This means

$$\hat{L}_N(\theta) \xrightarrow{\text{P}} p(\mathbf{y}|\theta)$$

as $N \to \infty$. In addition, we would like the estimator to vary smoothly with θ if $p(\mathbf{y}|\theta)$ does, ie. if

$$\forall\theta, \forall\epsilon > 0, \exists\delta > 0 \text{ s.t. } \forall\theta' \in \mathbb{B}_\delta(\theta), p(\mathbf{y}|\theta') \in \mathbb{B}_\epsilon(p(\mathbf{y}|\theta))$$

then

$$\forall\theta, \forall\epsilon > 0, \exists\delta > 0 \text{ s.t. } \forall\theta' \in \mathbb{B}_\delta(\theta), \hat{L}_N(\theta') \in \mathbb{B}_\epsilon(\hat{L}_N(\theta))$$

where $\mathbb{B}_r(\theta)$ denotes the ball of radius r around θ under some appropriate metric (eg. the metric induced by the L2-norm on $\mathbb{R}^d$). Finally, we want $\hat{L}_N(\theta)$ to have a computational complexity in $o(TN^2)$ and which permits practical values of N to be used for accurate estimation.

It might seem like the smoothness criterion is a strange one. However, there are two reasons to believe that a smooth likelihood estimator is advantageous. First, it captures the continuous nature of the true likelihood $p(\mathbf{y}|\theta)$ and can better facilitate likelihood maximization [23]. In particular, it leads to reduced variance estimates of the difference in likelihood for some θ_1 and θ_2:

$$\hat{L}_N(\theta_2) - \hat{L}_N(\theta_1) \approx p(\mathbf{y}|\theta_2) - p(\mathbf{y}|\theta_1)$$

Indeed, assume we have two consistent estimators $\tilde{L}_N(\theta)$ and $\hat{L}_N(\theta)$ both with variance given by

$\frac{A(\theta)}{N}$ but

$$\text{cov}(\tilde{L}_N(\theta_1), \tilde{L}_N(\theta_2)) = 0$$

and $\hat{L}_N(\theta)$ is smooth as defined above. Necessarily, we have

$$\text{cov}(\hat{L}_N(\theta_1), \hat{L}_N(\theta_2)) > 0$$

to satisfy smoothness when N is finite and $A(\theta_1)$ or $A(\theta_2)$ are nonzero. Then we have

$$\begin{aligned}\text{var}(\hat{L}_N(\theta_2) - \hat{L}_N(\theta_1)) &= \text{var}(\hat{L}_N(\theta_1)) + \text{var}(\hat{L}_N(\theta_2)) - 2\text{cov}(\hat{L}_N(\theta_1), \hat{L}_N(\theta_2)) \\ &< \text{var}(\hat{L}_N(\theta_1)) + \text{var}(\hat{L}_N(\theta_2)) \\ &= \text{var}(\tilde{L}_N(\theta_2) - \tilde{L}_N(\theta_1))\end{aligned}$$

Note that this reduction in variance does not come at the cost of consistency since the difference between two consistent estimators gives a consistent estimate of the difference between the two values they estimate. This is because the expectation of sums of random variables is equal to the sum of the expectations of those random variables, even if they are dependent.

A caution at this point is that smooth estimators are not useful in all endeavours. In particular, if in a Bayesian framework one wished to estimate $p(\mathbf{y}) = \int_\Theta p(\mathbf{y}|\theta)p(\theta)d\theta$ by sampling $\{\theta_i\}_{i=1}^R$ from a prior $p(\theta)$ and computing

$$\hat{p}(\mathbf{y}) = \sum_{i=1}^{R} \hat{L}_N(\theta_i)$$

then this estimate would have higher variance by the same reasoning since

$$\text{var}(\hat{L}_N(\theta_1) + \hat{L}_N(\theta_2)) = \text{var}(\hat{L}_N(\theta_1)) + \text{var}(\hat{L}_N(\theta_2)) + 2\text{cov}(\hat{L}_N(\theta_1), \hat{L}_N(\theta_2))$$

For such estimates, lower variance would be attained by using negatively correlated (and hence non-smooth) estimates of the likelihood.

3.2 Common Random Numbers

One way in which we can try to attain a smooth estimator of the likelihood is by using common random numbers. For example, imagine we want to compute

$$p(\mathbf{y}|\theta) = \int p(\mathbf{y}, \mathbf{x}|\theta)d\mathbf{x} = \int p(\mathbf{y}|\mathbf{x}, \theta)\frac{p(\mathbf{x}|\theta)}{q(\mathbf{x})}q(\mathbf{x})d\mathbf{x}$$

via the Monte Carlo estimate

$$\hat{I}(\theta) = \frac{1}{N}\sum_{i=1}^{N} p(\mathbf{y}|\mathbf{x}_i, \theta)\frac{p(\mathbf{x}_i|\theta)}{q(\mathbf{x}_i)}$$

where $\mathbf{x}_i \sim q(\mathbf{x})$, for $i = 1, \ldots, N$. Using *common* random numbers $\mathbf{x}_i$ and allowing θ to vary, we have that $\hat{I}(\theta)$ is a continuous function in θ as long as $p(\mathbf{x}|\theta)$ and $p(\mathbf{y}|\mathbf{x}, \theta)$ are continuous in θ, since the product and quotient of continuous functions is continuous[4].

In practice, however, we often use proposal distributions of the form $q(\mathbf{x}|\theta)$. In this scenario it is not possible to use samples $\mathbf{x}_i$ that are identical. However, the utility of common random numbers can be retained by generating $\mathbf{x}_i$'s that vary continuously with θ. In particular, assume we can sample a random variable $\mathbf{z}$ from $g(\mathbf{z})$ and we have a continuous (in θ) deterministic function $f(\mathbf{z}; \theta)$ that transforms $\mathbf{z}$, given θ, to a sample from a distribution $q(\mathbf{x}|\theta)$ that we can evaluate pointwise. This means we require $\mathbf{z} \sim g(\cdot) \Rightarrow f(\mathbf{z}; \theta) \sim q(\cdot|\theta)$. Therefore, if we use common random numbers $\mathbf{z}_i$ with θ varying, we still have that $\hat{I}(\theta)$ is continuous in θ since the composition of continuous functions is continuous.

For example, imagine we want to sample random vectors from a d-dimensional multivariate Gaussian distribution $q(\mathbf{x}|\theta)$ with $\theta = (\boldsymbol{\mu}, \Sigma)$ where $\boldsymbol{\mu}$ is the mean and Σ is the covariance matrix of the distribution. Furthermore, define A such that it satisfies $AA^T = \Sigma$. We can sample a standard multivariate Gaussian random vector $\mathbf{z} \in \mathbb{R}^d$ by sampling each component from the standard univariate Gaussian distribution $N(0, 1)$. Then we can set $\mathbf{x} = f(\mathbf{z}; \theta) = \boldsymbol{\mu} + A\mathbf{z}$ and we have that $\mathbf{x}$ is a sample from $q(\mathbf{x}|\theta)$. Furthermore, since $f(\mathbf{z}; \theta)$ is continuous in θ, this method can be used to produce 'common' random vectors from $q(\mathbf{x}|\theta)$ by using common random numbers from $N(0, 1)$.

Indeed, if we compute

$$p(\mathbf{y}|\theta) = \int p(\mathbf{y}, \mathbf{x}|\theta)d\mathbf{x} = \int p(\mathbf{y}|\mathbf{x}, \theta)\frac{p(\mathbf{x}|\theta)}{q(\mathbf{x}|\theta)}q(\mathbf{x}|\theta)d\mathbf{x}$$

via the Monte Carlo estimate

$$\hat{I}(\theta) = \frac{1}{N}\sum_{i=1}^{N} p(\mathbf{y}|\mathbf{x}_i, \theta)\frac{p(\mathbf{x}_i|\theta)}{q(\mathbf{x}_i|\theta)}$$

where $\mathbf{x}_i \sim q(\mathbf{x}|\theta)$ for $i = 1, \ldots, N$, we still have that $\hat{I}(\theta)$ is a continuous function in θ since the composition of continuous functions is continuous. In addition, for this case we require continuity in $\mathbf{x}$ as well as θ for $p(\mathbf{x}|\theta)$ and $p(\mathbf{y}|\mathbf{x}, \theta)$, continuity in both $\mathbf{x}$ and θ for $q(\mathbf{x}|\theta)$ and the existence of a function $f(\mathbf{z}|\theta)$ as detailed above.

[4] An additional requirement is that $q(\mathbf{x}) \neq 0$ but this is guaranteed by sampling from q and we require $q(\mathbf{x}) > 0$ whenever $p(\mathbf{x}|\theta) > 0$ for importance sampling to be valid anyway

On a practical note, when we say that we can use common random numbers or refer to their use as a 'technique', we are referring to the ability to run algorithms with varying values of θ but using the same random numbers in all runs. On a computer, this can often be achieved by resetting the random seed of the random number generator to a fixed number at the beginning of each run. Naturally, this only applies when the number of random numbers that are sampled does not vary across runs and each one is used for the same purpose in each run. This implies, for example, that we cannot use rejection sampling in such an algorithm when the acceptance probability depends on θ.

CRN and Particle Filters

Now that common random numbers have been introduced and motivated, it is worth mentioning that this technique alone cannot produce a smooth likelihood estimator within the particle filter framework. First, it is worth noting that using CRN within the sequential importance sampling framework presented in Algorithm 1 and computing an estimate of the log-likelihood using the expression given in Section 2.4.3 will indeed produce a smooth estimate. However, this method is hopelessly slow as any practical[5] value of N gives very high variance estimates of the likelihood. Unfortunately, the sequential importance sampling/resampling framework presented in Algorithm 2 that provides lower variance estimation simultaneously makes the likelihood estimator discontinuous, as remarked in [23]. In fact, smoothness of SIS is a result of the smoothness of the sampled latent variables $\{\mathbf{x}_{0:k}^{(i)}\}_{i=1}^{N}$ at every time-step k. The introduction of the resampling step in SIR introduces discontinuities in the latent variables $\{\mathbf{x}_{0:k}^{(i)}\}_{i=1}^{N}$. Indeed, from the multinomial sampling algorithm in Section 2.4.2, it is apparent that small changes in θ can cause drastically different values of $\mathbf{x}_{0:k}$ to be chosen. For example, if particle index j is chosen for a given θ but particle index $j+1$ is chosen for θ' that is close to θ, there is no guarantee that $\mathbf{x}_{0:k}^{(j)}$ will be at all close to $\mathbf{x}_{0:k}^{(j+1)}$. Additionally, even if we could construct a resampling procedure that made such a guarantee the resulting particles would not be continuous in θ without additional interpolation.

In summary, it is a non-trivial task to produce an estimator that satisfies the properties of consistency, smoothness and $o(N^2)$ time complexity for practical values of N. However, it is trivial to produce an estimator that has any two of these properties. If we are willing to sacrifice smoothness, the standard estimate using sequential importance sampling/resampling suffices. On the other hand, if we are willing to sacrifice reasonable time complexity, the standard estimate using sequential importance sampling (*without* resampling) suffices. It is hardly worth considering constructing an estimator that is not consistent, and it is trivial to construct one that is both fast and smooth.

[5] Although this estimator runs in $O(N)$ time, the variance increases exponentially with T since SIS is just importance sampling with a special proposal distribution. As such, for comparable variance to SIR, impractical values of N need to be chosen to compensate for the higher variance.

3.3 Related Work

The task of smooth likelihood estimation using sequential Monte Carlo methods is a relatively new endeavour, mostly because particle filters have only recently become popular. However, there has already been substantial work in this area and one method in particular is the main inspiration for the new ideas presented here. A brief survey of related advances in the field follows.

3.3.1 Smooth Likelihood Estimation with One-Dimensional State Variables

In [23], a resampling scheme is proposed that, coupled with common random numbers, gives us all three estimator properties we desire in the case where the latent variables for state are one-dimensional. The idea is quite simple and follows naturally from the problem statement outlined in Section 2.4.2: that particles whose indices are close are not necessarily close themselves. Indeed, it is straightforward to guarantee that particles whose indices are close are themselves close simply by sorting the particles and reassigning their indices.

More formally, assume that at stage t we have a weighted empirical distribution $\widehat{P}_N(dx_{0:t}|\mathbf{y}_{0:t}) = \sum_{i=1}^N W_t^{(i)} \delta_{x_{0:t}^{(i)}}(dx_{0:t})$. Since the models we are working with are first-order Markov in x, for the purposes of filtering and likelihood estimation we are only interested in the tth component of $x_{0:t}$ when resampling. As such, we can reorder the particles such that $\forall i, j \in \{1, \ldots, N\}, i < j \Rightarrow x_t^{(i)} < x_t^{(j)}$. An important result is that instead of using the empirical cumulative distribution function for indices

$$\hat{F}_{t,N}(j) = \sum_{i=1}^{j} W_t^{(i)}$$

introduced in Section 2.4.2 we can instead use the empirical distribution function for particles

$$\hat{F}_{t,N}(x) = \sum_{x_t^{(i)} \leq x} W_t^{(i)}$$

The corresponding inverse of this empirical cdf is given by

$$\hat{F}_{t,N}^{-1}(u) = \min\{x_t^{(i)} : \hat{F}_N(x_t^{(i)}) \geq u\}$$

The cost of sorting each set of N particles by their last component is in $O(N \log N)$. Since this is done at each time-step of the particle filter and the cost of resampling at each step is in $O(N)$ the time complexity of the algorithm is in $O(TN \log N)$. Of course, this method does not give continuous resampled particles without the addition of an interpolation scheme to transform the

sorted discrete cdf into a continuous cdf. In [23], a linear interpolation scheme is proposed to attain continuity. While interpolation introduces a bias, this bias becomes negligible as N increases.

As impressive as this new particle filtering framework is, unfortunately it cannot be trivially extended to the multivariate case since only when the latent state variables are one-dimensional can we sort them in this manner and guarantee closeness.

3.3.2 Smooth Likelihood Estimation with Multi-Dimensional State Variables in $O(N^2)$ time

While we often think of the empirical distribution that approximates $p(\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$ in our discussion:

$$\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t}) = \sum_{i=1}^{N} W_t^{(i)} \delta_{\mathbf{x}_{0:t}^{(i)}}(d\mathbf{x}_{0:t})$$

which is a mixture of point masses. However, we can instead think of the empirical density that approximates $p(\mathbf{x}_{t+1}|\mathbf{y}_{0:t})$:

$$\hat{p}_N(\mathbf{x}_{t+1}|\mathbf{y}_{0:t}) = \sum_{i=1}^{N} W_t^{(i)} p(\mathbf{x}_{t+1}|\mathbf{x}_t^{(i)})$$

which is a mixture of densities (see Section 2.4.2). Indeed, the resampling process followed by the transition step provides samples from this distribution. Furthermore, this density is continuous when $p(\mathbf{x}_{t+1}|\mathbf{x}_t)$ is continuous since it is a weighted sum of continuous densities. If we could produce smoothly varying (with $\{W^{(i)}\}_{i=1}^N$ and $\{\mathbf{x}_t^{(i)}\}_{i=1}^N$) samples from this mixture distribution we could use those samples to make a smooth likelihood estimator. Unfortunately, even if it is easy to sample smoothly from $p(\mathbf{x}_{t+1}|\mathbf{x}_t)$, it is not generally feasible to sample smoothly from a mixture of such distributions when the weights vary with θ.

A solution to this problem, proposed in [7], is to approximate $\hat{p}_N(\mathbf{x}_{t+1}|\mathbf{y}_{0:t})$ with another distribution $g(\mathbf{x}_{t+1}; \alpha_{t+1})$, which is easy to produce smoothly varying (in α_{t+1}) samples from. The density $g(\mathbf{x}_{t+1}; \alpha_{t+1})$ is a member from the parametric class of densities $\{g(\mathbf{x}_{t+1}|\alpha_{t+1}) : \alpha_{t+1} \in \mathcal{A}\}$. In particular, the parameter α_{t+1} is obtained in a way that minimizes some distributional distance between $g(\mathbf{x}_{t+1}; \alpha_{t+1})$ and $\hat{p}_N(\mathbf{x}_{t+1}|\mathbf{y}_{0:t})$. For this to be useful, it must be true that the parameter α_{t+1} obtained by this minimization varies smoothly with θ.

Using $g(\mathbf{x}_{t+1}; \alpha_{t+1})$ as a proposal distribution, we can approximate $p(\mathbf{y}_{t+1}|\mathbf{y}_{0:t})$ as follows

$$\hat{p}(\mathbf{y}_{t+1}|\mathbf{y}_{0:t}) = \frac{1}{N} \sum_{i=1}^{N} p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1}^{(i)}) \frac{\hat{p}_N(\mathbf{x}_{t+1}^{(i)}|\mathbf{y}_{0:t})}{g(\mathbf{x}_{t+1}^{(i)}; \alpha_{t+1})}$$

The drawback is that $\hat{p}_N(\mathbf{x}_{t+1}^{(j)}|\mathbf{y}_{0:t}) = \sum_{i=1}^{N} W_t^{(i)} p(\mathbf{x}_{t+1}^{(j)}|\mathbf{x}_t^{(i)})$ takes $O(N)$ time to compute and so $\hat{p}(\mathbf{y}_{t+1}|\mathbf{y}_{0:t})$ takes $O(N^2)$ time to compute[6]. In addition, it can be difficult to define parametric families of densities that can both be smoothly sampled from and can represent well the target distribution $p(\mathbf{x}_{t+1}|\mathbf{y}_{0:t})$ when this target has certain properties, eg. multiple modes.

In summary, this method satisfies smoothness and consistency but requires time in $O(TN^2)$. In general, while proposal distributions that can be smoothly sampled from do provide smooth estimation, the evaluation of each importance weight will always take $O(N)$ time unless some kind of cancellation occurs.

3.3.3 Smooth Likelihood Estimation Using Fixed Selection Indices

An approach introduced in [5] is to run a particle filter for a given $\theta = \theta^*$ and to store the selection indices $\{k_t^{(i)} : i \in \{1,\dots,N\}, t \in \{0,\dots,T\}\}$ and weights $\{W_{t,\theta^*}^{(k_t^{(i)})} \stackrel{\text{def}}{=} W_t^{(k_t^{(i)})} : i \in \{1,\dots,N\}, t \in \{0,\dots,T\}\}$ (see Section 2.4.2). Then, for θ close to θ^*, the 'filter' is run such that the same indices are selected. In practice, this means that the weights of resampled particles $w_t^{(i)}$ are no longer equal. Indeed, if θ causes the weight of a selected particle to increase relative to the weight under θ^* then the weight will increase and if the opposite is true, the weight will decrease. In Algorithm 2 this can be seen as replacing the resampling step with:

- For $i = 1,\dots,N$:
  - Set $\mathbf{x}_{0:t}^{(i)} = \tilde{\mathbf{x}}_{0:t}^{(k_t^{(i)})}$
  - Set $w_t^{(i)} = \dfrac{W_t^{(k_t^{(i)})}}{W_{t,\theta^*}^{(k_t^{(i)})}}$
- Renormalize the weights $w_t^{(i)}$ so $\sum_{i=1}^{N} w_t^{(i)} = 1$.

This method does provide smooth estimates of the likelihood when coupled with common random numbers. However, this method is restricted to a small neighbourhood around θ as many of the weights will become negligible when the discrepancy in the weights increases. Of particular concern is the fact that without conventional resampling (based on the actual weights of the particles in question) the effective sample size of the particle 'swarm' decreases over time, which can reduce significantly the variance of the estimates. In this scenario, the high positive correlation in log-likelihood estimates can be offset by the increase in the variance of estimates themselves (see Section 3.1) when considering the variance of the estimated difference in the log-likelihood. In addition, the numerical stability of this method on a computer is not guaranteed due to the repeated use of

[6] As with the marginal particle filter, it may be possible to speed up this computation in some circumstances. In general, however, the complexity is $O(N^2)$

products in the algorithm. Of course, this method is appealing for use in a small neighbourhood of θ^* if T is not too large. In particular, it is used in [5] to estimate the partial derivatives of $p(\mathbf{y}_{0:t}|\theta)$ with respect to θ.

3.4 Theoretical Solutions

This chapter is concluded with the introduction of two theoretical solutions to the problem of smooth resampling. These can both be seen as extensions of [23] to the case where the state variables are multivariate. The main idea in that work is that we can, through sorting the particles, construct an empirical cdf for x_t

$$\hat{F}_{t,N}(z) = \sum_{x_t^{(i)} \leq z} w_t^{(i)}$$

that approximates the cdf

$$F_t(z) = \int_{-\infty}^{z} p(x_t|y_{0:t})dx_t$$

When $\mathbf{x_t}$ is multi-dimensional, however, this no longer applies. After introducing some regularity conditions and some modifications to notation, we will present two solutions that inspire new approaches.

3.4.1 Regularity Conditions and Notation

To deal with the case when $\mathbf{x_t} \in \mathbb{R}^d$ is multi-dimensional, we first simplify our notation. Instead of referring to the density $p(\mathbf{x}_t|\mathbf{y}_{0:t})$ we will drop explicit references to $\mathbf{y}_{0:t}$ and the time index t. Thus from this point on the density $p(\mathbf{x}) = p(x_{1:d})$ refers implicitly to $p(\mathbf{x}_t|\mathbf{y}_{0:t})$. Using this notation, we will refer to marginal distributions using

$$p_j(x_j) = \int\int p(x_{1:j-1}, x_j, x_{j+1:d})dx_{1:j-1}dx_{j+1:d}$$

and

$$p_{1:j}(x_{1:j}) = \int p(x_{1:j}, x_{j+1:d})dx_{j+1:d}$$

We will also refer to conditional distributions in a limited context with

$$p_j(x_j|x_{1:j-1}) = \int p(x_j, x_{j+1:d}|x_{1:j-1})dx_{j+1:d}$$

Finally, we can express the cdf of x_1 as

$$F_1(z) = \int_{-\infty}^{z} p_1(x_1)dx_1$$

and the conditional cdf of x_j given $x_{1:j-1}$ as

$$F_j(z|x_{1:j-1}) = \int_{-\infty}^{z} p_j(x_j|x_{1:j-1})dx_j$$

In addition to the change in notation, we restrict our attention to situations in which the set $\{\mathbf{x} : p(\mathbf{x}) > 0\}$ is convex[7]. The reason for this condition is that when it is satisfied, the cdf $F_1(z)$ and the conditional cdfs $F_j(z|x_{1:j-1})$ are strictly monotonic for all z such that these functions are in $(0, 1)$. As such, the quantile function, or inverse cdf, F_1^{-1} is given by

$$F_1^{-1}(u) = z : F_1(z) = u$$

Similarly, the conditional quantile function F_j^{-1} is given by

$$F_j^{-1}(u|x_{1:j-1}) = z : F_j(z|x_{1:j-1}) = u$$

Note that when one of these cdfs is not strictly monotonic, there exist values of $u \in (0, 1)$ and $x_{1:j-1}$ such that z above is not determined uniquely by the constraint $F_j(z|x_{1:j-1}) = u$. Furthermore, the cdf of any marginal distribution over any convex subset of $\{\mathbf{x} : p(\mathbf{x}) > 0\}$ is strictly monotonic.

3.4.2 A Generalized CDF

We can define a bijective generalized cdf $F : \mathbb{R}^d \to \mathbb{R}^d$, where the state variables are d-dimensional. First let us define

$$F_j(z; z_{1:j-1}) = F_j(z|z_{1:j-1}) = \int_{-\infty}^{z} p_j(x_j|x_{1:j-1} = z_{1:j-1})dx_j$$

The inverse of this function is

$$\begin{aligned} F_j^{-1}(u; u_{1:j-1}) &= x_j : F_j(x_j; F_1^{-1}(u_1), F_2^{-1}(u_2; u_1), \ldots, F_{j-1}^{-1}(u_{j-1}; u_{1:j-2})) = u \\ &= x_j : F_j(x_j; z_{1:j-1}) = u \\ &= F_j^{-1}(u|z_{1:j-1}) \end{aligned}$$

[7] It is possible that this condition can be relaxed but this is sufficient for all the marginal distributions of interest to be non-zero only on a convex subset (interval) of $\mathbb{R}$.

where $z_1 = F_1^{-1}(u_1)$, $z_2 = F_2^{-1}(u_2; u_1) = F_2^{-1}(u_2|z_1)$ and in general $z_i = F_i^{-1}(u_i; u_{1:i-1}) = F_i^{-1}(u_i|z_{1:i-1})$.

With these definitions, we can define a bijection F as

$$F(z_1, z_2, \ldots, z_d) = (F_1(z_1), F_2(z_2; z_1), \ldots, F_d(z_d; z_{1:d-1}))$$

The inverse of F is given by

$$F^{-1}(u_1, u_2, \ldots, u_d) = \left(F_1^{-1}(u_1), F_2^{-1}(u_2; u_1), \ldots, F_d^{-1}(u_d; u_{1:d-1})\right)$$

While this formulation might appear daunting, the intuition is simple when the function $F(z_{1:d}) = u_{1:d}$ is considered sequentially. We can view u_1 as the probability that $x_1 \leq z_1$, u_2 as the probability that $x_2 \leq z_2$ given that $x_1 = z_1$ and so on until u_d is the probability that $x_d \leq z_d$ given that $x_{1:d-1} = z_{1:d-1}$. For the evaluation of the inverse F^{-1}, an algorithmic approach is presented in Algorithm 3. It should also be clear that if $\mathbf{u}$ is drawn uniformly from the unit hypercube of dimension d, then the output of the algorithm is a corresponding sample from the distribution $p(\mathbf{x})$. This is because $z_1 \sim p(x_1)$, $z_2 \sim p(x_2|x_1 = z_1), \ldots, z_d \sim p(x_d|x_{1:d-1} = z_{1:d-1})$.

Algorithm 3 An algorithm to evaluate the inverse of the generalized cdf $F^{-1}(u_1, \ldots, u_d)$

1. Set $z_1 = F_1^{-1}(u_1) = z : F_1(z) = u_1$
2. For $j = 2, \ldots, d$
   - Set $z_j = F_j^{-1}(u_j; u_{1:j-1}) = z : F_j(u_j|z_{1:j-1}) = u_j$
3. Output $(z_1, \ldots, z_d)$.

3.4.3 Median-Cuts

While the generalized CDF proposed above is *in theory* a clean extension of the approach in [23], we will see later that there are practical issues involved in trying to approximate conditional quantiles (F_j^{-1} for $j > 1$) directly using an empirical distribution. In addition, many of the properties of the generalized CDF are unnecessary for smooth resampling. In particular, we are looking for a function f such that if a random number u is sampled uniformly from $[0, 1]$, $f(u)$ returns a point sampled according to the distribution with density $p(\mathbf{x})$. We require this function to be continuous in θ (recall that this density does depend on θ) but not u and it need not have an inverse. Nevertheless, the mapping function f_n described in Algorithm 4 is largely inspired by the inverse transform technique. Furthermore, we can think of $f(u)$ as equal to unique element of $\lim_{n\to\infty} f_n(u)$ as the

type of function described above since $\lim_{n\to\infty} f_n(u)$ almost surely[8] returns a set containing a single point in $\mathbb{R}^d$.

Algorithm 4 The mapping function $f_n(u)$

1.
   - Abusing notation, set $a_1^{(1)} = a_2^{(1)} = \cdots = a_d^{(1)} = -\infty$ and $b_1^{(1)} = b_2^{(1)} = \cdots = b_d^{(1)} = \infty$
   - Let $\mathcal{X}^{(1)} \stackrel{\text{def}}{=} \prod_{i=1}^d [a_i^{(1)}, b_i^{(1)}]$
   - Let $\mathcal{X}_{-1}^{(1)} \stackrel{\text{def}}{=} \prod_{i=2}^d [a_i^{(1)}, b_i^{(1)}]$
   - Compute t_1 satisfying $\int_{a_1^{(1)}}^{t_1} \int_{\mathcal{X}_{-1}^{(1)}} p(\mathbf{x}) dx_{2:d} dx_1 = 0.5$
   - For $i = 1, \ldots, d$
     - Set $a_i^{(2)} = a_i^{(1)}$ and $b_i^{(2)} = b_i^{(1)}$
   - If $u < 0.5$, set $b_1^{(2)} = t_1$ and $u = 2u$, else set $a_1^{(2)} = t_1$ and $u = 2(u - 0.5)$
2. For $j = 2, \ldots, n$
   - Let $k = j \mod d$. If $k = 0$, let $k = d$.
   - Let $\mathcal{X}^{(j)} \stackrel{\text{def}}{=} \prod_{i=1}^d [a_i^{(j)}, b_i^{(j)}]$
   - Let $\mathcal{X}_{-k}^{(j)} \stackrel{\text{def}}{=} \prod_{i=1}^{k-1} [a_i^{(j)}, b_i^{(j)}] \prod_{i=k+1}^d [a_i^{(j)}, b_i^{(j)}]$
   - Compute t_j satisfying $2^{j-1} \int_{a_k^{(j)}}^{t_j} \int_{\mathcal{X}_{-k}^{(j)}} p(\mathbf{x}) dx_{1:k-1} dx_{k+1:d} dx_k = 0.5$
   - For $i = 1, \ldots, d$
     - Set $a_i^{(j+1)} = a_i^{(j)}$ and $b_i^{(j+1)} = b_i^{(j)}$
   - If $u < 0.5$, set $b_k^{(j+1)} = t_j$ and $u = 2u$, else set $a_k^{(j+1)} = t_j$ and $u = 2(u - 0.5)$
3. Return $\mathcal{X}^{(n+1)} = \prod_{i=1}^d [a_i^{(j+1)}, b_i^{(j+1)}]$.

Algorithm 5 Algorithm to convert a number u in [0,1] to a string of bits of length n

1. For $j = 1, 2, \ldots, n$

   If $u < 0.5$

   - Set $s_j = 0$ and $u = 2u$.

   Else

   - Set $s_j = 1$ and $u = 2(u - 0.5)$.
3. Return $\mathbf{s}$.

Correctness

The function f_n is a mapping from $[0, 1]$ to a region in $\mathbb{R}^d$. In the algorithm we essentially treat the number u as an n-bit string $\mathbf{s}$ of 0's and 1's, where each s_j is independent and identically distributed

[8] By this we mean for almost all $u \in [0, 1]$.

from the Bernoulli distribution with success probability 0.5 (without conditioning on u). Algorithm 5 shows how to produce such a string of arbitrary length n. It is sometimes easier to think of the function f_n taking an argument $s_{1:n}$ when n is finite (instead of conditioning on $u < 0.5$ at step j, one can condition directly on $s_j = 0$).

For any n and $\mathbf{s} = s_{1:n} \in \{0,1\}^n$, the region $\mathcal{X}^{(n+1)}$ returned satisfies

$$\int_{\mathcal{X}^{(n+1)}} p(\mathbf{x}) d\mathbf{x} = \frac{1}{2^n}$$

by construction. This corresponds to the probability of the n-bit binary string $\mathbf{s}$ used to select the region. Furthermore, we can see that

$$\bigcup_{\mathbf{s}\in\{0,1\}^n} f_n(\mathbf{s}) = \mathbb{R}^d$$

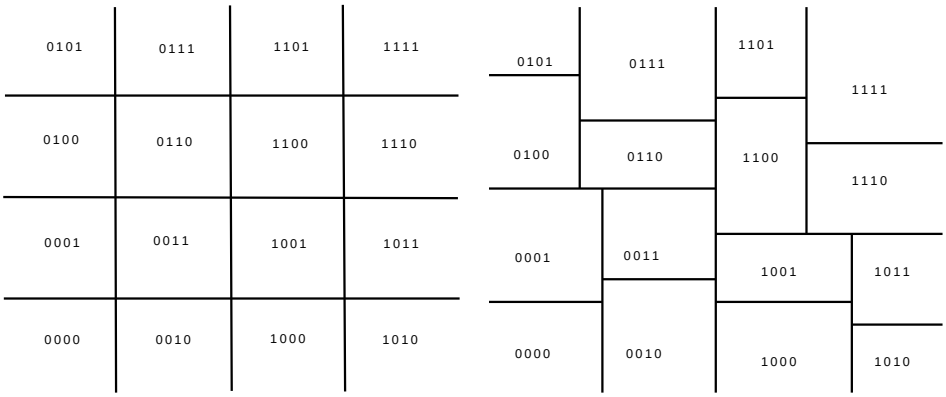
In other words, each equally probable uniform random n-bit $\mathbf{s} \in \{0,1\}^n$ maps to an equally probable (under p) region $f_n(\mathbf{s}) \in \mathbb{R}^d$. In addition, the intersection of any two regions $f_n(\mathbf{s}_1)$ and $f_n(\mathbf{s}_2)$ for $\mathbf{s}_1 \neq \mathbf{s}_2$ is a subset of $\mathbb{R}^d$ with dimension less than d and hence it has null Lebesgue measure in $\mathbb{R}^d$. As such, there are no overlapping regions of non-zero probability for distinct $\mathbf{s}$ and we have

$$\sum_{\mathbf{s}\in\{0,1\}^n} \int_{f_n(\mathbf{s})} p(\mathbf{x}) d\mathbf{x} = 1$$

In sum, each of the regions that can be returned for a given n have equal probability mass $\frac{1}{2^n}$ and cover the entire space $\mathbb{R}^d$ while the intersection of the interior of any two regions is empty. Given a uniform random variable u, this variable is treated as an n-bit binary string $\mathbf{s}$ drawn from the uniform distribution over n-bit strings (each possible string has probability $\frac{1}{2^n}$) and used to select one of the regions.

It should be emphasized that if we think of the 2^n regions that f_n can return as a partitioning[9] of $\mathbb{R}^d$, the relative positions of the regions are fixed and so the relative position of the region that is picked by u (via $\mathbf{s}$) is not dependent on p (although the coordinates of the region are). Take, for example, $d = 2$ and $s_{1:4} = 1001$. This maps $s_{1:4}$ to the region $[t_1, t_3] \times [t_4, t_2]$ with $a_1^{(2)} = t_1$, $b_2^{(3)} = t_2$, $b_1^{(4)} = t_3$ and $a_2^{(5)} = t_4$. Clearly the values of t_i depend on p, but the position of the partition is fixed with respect to the other partitions. This is probably best understood by looking at Figure 3.1, which shows how the function f_n partitions $\mathbb{R}^d$ for two different densities $p(\mathbf{x})$. In the context where $p(\mathbf{x})$ is continuous in θ, it is thus the coordinates of the orthotope $\mathcal{X}^{(n+1)}$ that change as θ varies with u fixed. In fact, for $p(\mathbf{x})$ continuous in θ and $\mathbf{x}$, each t_j is continuous in θ and so the coordinates of the orthotope are continuous in θ.

[9] As noted above, the intersection of two regions is not always null, but the intersection of their interiors are.



(a) uniform (b) non-uniform

Figure 3.1: Median-cut partitioning of the space for $p(\mathbf{x})$ uniform and $p(\mathbf{x})$ non-uniform

Characterizing the Output

Note that for fixed u, the sequences $\{a_i^{(n)}\}$ for $i \in \{1, \ldots, d\}$ are weakly monotone increasing. Similarly, the sequences $\{b_i^{(n)}\}$ for $i \in \{1, \ldots, d\}$ are weakly monotone decreasing. Furthermore, for any i, j, k, $a_i^{(k)} < b_i^{(j)}$ and $b_i^{(k)} > a_i^{(j)}$. It can be shown that the set $\{u : \exists i \text{ s.t. } \forall j, b_i^{(j)} = \infty \text{ or } a_i^{(j)} = -\infty\}$ has measure zero since this corresponds to the infinite binary string representation of u having infinitely repeated 1's or 0's at fixed intervals. Thus $\{a_i^{(n)}\}$ and $\{b_i^{(n)}\}$ are almost always bounded above and below respectively. Since they are weakly monotone this implies that they converge and their limits are given by

$$a_i \stackrel{\text{def}}{=} \lim_{n \to \infty} a_i^{(n)} = \sup_n \{a_i^{(n)}\}$$
$$b_i \stackrel{\text{def}}{=} \lim_{n \to \infty} b_i^{(n)} = \inf_n \{b_i^{(n)}\}$$

By the nested interval theorem, we know that

$$\mathcal{X}^* \stackrel{\text{def}}{=} \lim_{n \to \infty} \mathcal{X}^{(n+1)} = \prod_{i=1}^{d} [a_i, b_i]$$

is a non-empty subset of $\mathbb{R}^d$. In fact, we know that for at least one i, $a_i = b_i$. To show this, assume that $a_i < b_i$ for all i. Then

$$\int_{\mathcal{X}^*} p(\mathbf{x}) d\mathbf{x} = c$$

where c is a constant. However, we know $\int_{\mathcal{X}^{(j)}} p(\mathbf{x}) d\mathbf{x} = \frac{1}{2^{j-1}}$ so there exists a j such that $\int_{\mathcal{X}^j} p(\mathbf{x}) d\mathbf{x} < \int_{\mathcal{X}^*} p(\mathbf{x}) d\mathbf{x}$ which is a contradiction since $\mathcal{X}^* \subseteq \mathcal{X}^{(j)}$. The following shows that

for every i, $a_i = b_i$.

Claim: For any $i \in \{1, \ldots, d\}$, $a_i = b_i$ for almost all $u \in [0, 1]$.

Proof: Consider the sequence of natural numbers defined by $m_k = ki$ for $k \in \mathbb{N}$. Then consider the subsequences $\{a_i^{(m_j)}\}$, $\{b_i^{(m_j)}\}$ and $\{t_{m_j}\}$. Note that these subsequences satisfy

$$t_{m_j} \in (a_i^{(m_j)}, b_i^{(m_j)})$$

$$b_i^{(m_{j+1})} = \begin{cases} t_{m_j} & \text{if } s_{m_j} = 0, \\ b_i^{(m_j)} & \text{if } s_{m_j} = 1 \end{cases}$$

$$a_i^{(m_{j+1})} = \begin{cases} a_i^{(m_j)} & \text{if } s_{m_j} = 0, \\ t_{m_j} & \text{if } s_{m_j} = 1 \end{cases}$$

Now assume that there exists an $h > 0$ such that $b_i - a_i \geq h$. This implies that $b_i^{(m_j)} - a_i^{(m_j)} \geq h$ for all j. We already know that $a_i^{(m_j)}$ and $b_i^{(m_j)}$ are finite almost surely as $j \to \infty$. Let k be the first integer such that $a_i^{(m_k)}$ and $b_i^{(m_k)}$ are both finite and let $d_j = b_i^{(m_j)} - a_i^{(m_j)}$. Then for any $j \geq k$ we require $d_j \geq h$.

For any j, $t_{m_j} \in (a_i^{(m_j)}, b_i^{(m_j)})$. If $b_i^{(m_j)} - t_{m_j} \geq h$ and $t_{m_j} - a_i^{(m_j)} \geq h$, then regardless of the value of s_{m_j} we will have $d_{j+1} \leq d_j - h$. Since d_k is finite, there are only finitely many times this can occur for $j > k$. If, on the other hand, $b_i^{(m_j)} - t_{m_j} \leq h$ or $t_{m_j} - a_i^{(m_j)} \leq h$, we require s_{m_j} to be equal to 1 or 0 respectively, in order for $d_{j+1} \geq h$. Since the previous case can only occur finitely many times, this case must occur infinitely many times as there are infinitely many $j > k$. Note that the values of t_{m_j} depend on $p(\mathbf{x})$ and $s_{1:m_j-1}$ but not s_{m_j}. In fact, s_{m_j} can be thought of as drawn from the Bernoulli distribution with success probability 0.5. As such, we are requiring that an infinite number of independent random bits of $\mathbf{s}$ coincide with infinitely many values of t_{m_j}, which has probability zero. Hence we have that $h = 0$ almost surely and so $a_i = b_i$ almost surely $\square$.

Of course, this means that for almost all u,

$$\mathcal{X}^* \stackrel{\text{def}}{=} \lim_{n \to \infty} \mathcal{X}^{(n+1)} = \prod_{i=1}^{d} [a_i, b_i] = \{\mathbf{c}\}$$

where $\mathbf{c} \in \mathbb{R}^d$ and $c_i = \sup_n \{a_i^{(n)}\} = \inf_n \{b_i^{(n)}\}$ for $i \in \{1, \ldots, d\}$. Thus we can define (for almost all u) $f(u)$ as the single member of $\lim_{n \to \infty} f_n(u)$.

3.4.4 Practical Notes

While the theoretical resampling algorithms described above would yield smoothly varying particles, they are impossible to implement in practice because they require the repeated evaluation of integrals that have no analytical solution in general. As such, in the next chapter we focus on solutions that are implementable on a computer.

Chapter 4

Fast and Approximately Smooth Likelihood Estimation

In this chapter, we present a tree-based representation of the particles for resampling. This representation is itself flexible and can be used within a wide variety of resampling schemes. However, we restrict our discussion to three different realizations.

4.1 A Weighted Tree Structure for $\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$

A valid resampling scheme for $\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t}) = \sum_{i=1}^{N} W_t^{(i)} \delta_{\mathbf{x}_{0:t}^{(i)}}(d\mathbf{x}_{0:t})$ can be constructed by partitioning the particles $\mathbf{x}_{0:t}^{(i)}$ into k_1 subsets where each subset is given a weight equal to the normalized sum of the weights of the particles within it. Each of the k_1 sets is then further partitioned into some number of subsets where weights are again assigned so that the weight of a subset is given by the normalized sum of the weights of the particles within it. This process continues until each particle is in only one partition at the lowest depth of the tree.

Let the depth of the tree be d and assume we have a d-dimensional uniform random number $(u_1, \ldots, u_d)$. Then we can use u_1 to select the subset on the first level according to the weights of each partition, u_2 to select a subset of that subset according to the weights of each of these subsets, and so on until we use some u_j to select a single particle[10].

Let the kth level subset containing the particle with index i be denoted $S_k(i)$ and the sum of the weights in $S_k(i)$ be denoted $W(S_k(i))$. The probability of selecting any individual particle index i

[10] $u_{j+1}, \ldots, u_d$ might not be needed since in this general construction the leaves of the tree do not necessarily all first appear on the dth level.

is given by:

$$\begin{aligned}\Pr[\text{select index } i] &= \frac{W(S_1(i))}{\text{sum of all weights}} \cdot \frac{W(S_2(i))}{W(S_1(i))} \cdot \frac{W(S_3(i))}{W(S_2(i))} \cdots \frac{W(S_d(i))}{W(S_{d-1}(i))} \\ &= \frac{W(S_d(i))}{\text{sum of all weights}} \\ &= \frac{\text{weight of particle } i}{\text{sum of all weights}}\end{aligned}$$

ie. the probability of selecting a particle from such a tree is proportional to its weight.

It is clear that this general description allows for a wide variety of possible trees to be constructed. Our main interest is in constructing a tree such that for fixed $\mathbf{u}$, small perturbations in the weights of the particles give rise to small perturbations in the selected particle.

4.2 Splitting via the Unweighted Median (Weighted Binary Trees)

Let us assume we have $N = 2^k$ for some $k \in \mathbb{N}$. Then one possible construction of a tree is to have each parent have two child nodes each containing half the number of particles of their parent, ie. a perfect binary tree. The depth of this tree is $k+1 = \log_2 N + 1$. To partition each block of particles at each level, we propose to sort the particles along one dimension, cycling through each dimension as we traverse the levels. For example, we can sort the N particles according to the first dimension of $\mathbf{x}_t$, then sort each subset of $N/2$ particles on the second level of the tree by the second dimension of $\mathbf{x}_t$, etc., and we return to the first dimension after sorting by the dth dimension of $\mathbf{x}_t$ if $\mathbf{x}_t$ is d-dimensional.

The cost to construct this tree is given by:

$$\begin{aligned}T(N) &= N\log(N) + 2\frac{N}{2}\log(\frac{N}{2}) + 4\frac{N}{4}\log(\frac{N}{4}) + \cdots + \frac{N}{2}\frac{N}{\frac{N}{2}}\log(\frac{N}{\frac{N}{2}}) \\ &= Nk + N(k-1) + N(k-2) + \cdots + N(2) + N \\ &= N(k + (k-1) + (k-2) + \cdots + 2 + 1) \\ &= N\frac{k(k+1)}{2} \\ \Rightarrow T(N) &\in O(N(\log N)^2)\end{aligned}$$

The cost to select a particle from this tree is in $O(\log N)$ so the cost to evaluate it N times is in $O(N \log N)$. This means the total cost of both constructing the tree and resampling N particles is

in $O(N(\log N)^2)$.

Algorithm 6 Constructing the weighted binary tree: constructq($particles$, $level$)

1. Set dimension r = level mod d. If $r = 0$, set $r = d$.
2. Set particle m = the median of $particles$ in the rth dimension.
3. Create node $node$, empty sets $left$ and $right$, and set $w_{left} = w_{right} = 0$.
4. If $level = k + 1$, set $node.particle = p$ and return $node$.
5. For each particle p in $particles$
   - If the rth coordinate of p is less than or equal to the rth coordinate of m,
     - Add p to the set $left$.
     - Set $w_{left} = w_{left}$+weight(p).
   - If the rth coordinate of p is greater than the rth coordinate of m
     - Add p to the set $right$.
     - Set $w_{right} = w_{right}$+weight(p).
6. - Set $node.w_{left} = w_{left}/(w_{left} + w_{right})$
   - Set $node.w_{right} = w_{right}/(w_{left} + w_{right})$
7. - Set $node.left$ = constructq($left$, $level + 1$)
   - Set $node.right$ = constructq($right$, $level + 1$)
8. Return $node$.

Algorithm 7 Selecting a particle from the weighted binary tree: selectq($node$, $level$, $u_1, \ldots, u_d$)

1. If $level = k + 1$, return $node.particle$.
2. Set dimension $j = level$ mod d. If $j = 0$, set $j = d$.
3. If $u_j < node.w_{left}$
   - Set $u_j = u_j/node.w_{left}$.
   - Return selectq($node.left$, $level + 1$, $u_1, \ldots, u_d$)

   Else
   - Set $u_j = (u_j - node.w_{left})/(1 - node.w_{left})$.
   - Return selectq($node.right$, $level + 1$, $u_1, \ldots, u_d$)

While the unweighted median along a particular dimension is used to split the particles into two groups, there are obviously unequal weights associated with each group except in the case where the weights of all the particles are equal. As such, splitting along the unweighted median gives rise to a weighted binary tree. While this tree might resemble the theoretical median-cuts approach in Section 3.4.3 there are important differences. First, the particles are approximately split along the median of the proposal distribution q and not p. Second, the values of the weights of each node are dependent on θ since the weights $W_t^{(i)}$ depend on θ. This means that while using the same $\mathbf{u}$ to traverse two similarly weighted trees of even continuously varying particles will often lead to

selecting the same intermediate nodes, there will be times when the change in weights causes a different node to be picked (this is also what makes the selection scheme correct). Obviously, the specific level of the tree that this occurs in impacts greatly how far apart the corresponding particles will be.

4.2.1 From $O(N(\log N)^2)$ to $O(N \log N)$ time

To construct each level of the tree, it is not necessary to sort the particles completely. In particular, we only need to find the median particle to split the particles into two blocks of equal numbers of particles and sum their weights. There exists a deterministic algorithm to find the median in linear time ("Median of Medians") [2], although in practice its performance is slower than some randomized algorithms (eg. Hoare's selection algorithm [14]) which run in expected linear time [18].

Upon discovering the median particle, the rest of the work at each level of the tree also takes linear time so we can construct the tree in $O(N \log N)$ time. This means the overall resampling step takes $O(N \log N)$ time.

4.2.2 From $\mathbf{u} \in \mathbb{R}^{\mathbf{k}}$ to $\mathbf{u} \in \mathbb{R}^{\mathbf{d}}$

It should be clear that the use of k uniform random numbers to select a single particle is unnecessary since we traverse the tree at level j by checking only if u_j is less than the sum of weights in a level j subpartition. Let the level j subset reached by using $u_1, \ldots, u_{j-1}$ be denoted $B_j(u_{1:j-1})$. Let

$$w_{j,0} = \frac{W(B_{j+1}(u_{1:j-1}, 0))}{W(B_j(u_{1:j-1}))}$$

and

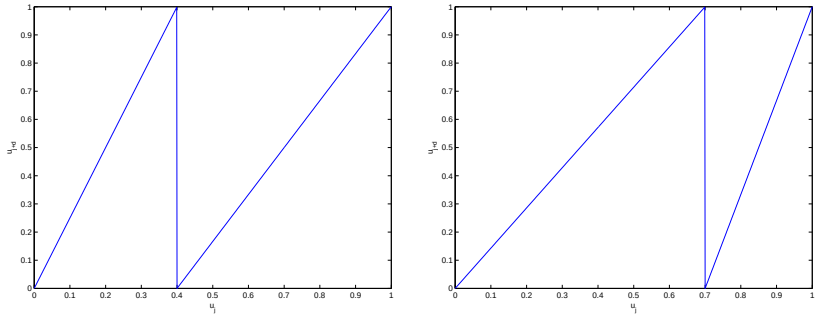
$$w_{j,1} = \frac{W(B_{j+1}(u_{1:j-1}, 1))}{W(B_j(u_{1:j-1}))}$$

ie. $w_{j,i} \propto W(B_{j+1}(u_{1:j-1}, i))$ and $\sum_{i \in \{0,1\}} w_{j,i} = 1$.

Then conditioned *only* on $u_j < w_{j,0}$, the value $\frac{u_j}{w_{j,0}}$ is uniformly distributed on $[0, 1)$. Similarly, conditioned *only* on $u_j \geq w_{j,0}$ the value $\frac{u_j - w_{j,0}}{1 - w_{j,0}}$ is uniformly distributed on $[0, 1]$. Thus we can have $u_1, \ldots, u_d$ given and then at every level j we set $u_{j+d} = \frac{u_j}{w_{j,0}}$ or $u_{j+d} = \frac{u_j - w_{j,0}}{1 - w_{j,0}}$ if $u_j < w_{j,0}$ or $u_j \geq w_{j,0}$ respectively.

This is not just a trick to reduce the number of uniform random variables we generate, especially since this generation is not costly. Instead, it provides a mechanism by which we ensure that when we are 'near' the boundary $w_{j,0}$, the next time we choose a node after splitting along the same

dimension we use a consistent value u_{j+d}. Intuitively, imagine $u_j = w_{j,0} - \epsilon$. Then $u_{j+d} = 1 - \frac{\epsilon}{w_{j,0}}$, so the next time we choose a node after splitting along the same dimension we are highly likely to pick the second one. This is consistent with being very close to having picked the second partition at level j. Of course, this only mitigates the problem by trying to reduce the distance between selected particles when the weights change - the distance between two particles can only be strictly bounded by the size of their last common ancestor.



(a) $w_{j,0} = 0.4$ (b) $w_{j,0} = 0.7$

Figure 4.1: Plots of u_{j+d} against u_j for $w_{j,0} = 0.4$ and $w_{j,0} = 0.7$

At first glance, it might appear that the manipulation of each u_j to produce u_{j+d} introduces a bias in the selection of the particles. This is not so. The fact that $u_j = w_{j,0} - \epsilon \Rightarrow u_{j+d} = 1 - \frac{\epsilon}{w_{j,0}}$ just shows that if we know the actual value of u_j then u_{j+d} is not random at all. However, in the algorithm the only use of u_j is to check $u_j < w_{j,0}$ and compute u_{j+d}. We never explicitly check if u_j is by some measure close to $w_{j,0}$. Thus the distribution of u_{j+d} is always uniform on either $[0, 1)$ or $[0, 1]$ and the probability of selecting the left node at stage $j + d$ is equal to $w_{j+d,0}$ for $u_{1:d}$ drawn from the uniform distribution on $[0, 1]^d$. Figure 4.1 shows the impact of the weight and the value of u_j on the value of u_{j+d}. Inspection of these graphs should reassure the reader that $\Pr[u_{j+d} < w_{j+d,0} | u_j < w_{j,0}] = w_{j+d,0}$.

4.2.3 Interpolation

The construction of the tree could provide even smoother estimates of the likelihood if we are willing to introduce some bias by interpolating between particles at the last (few) layer(s) of the tree. One strategy is to interpolate between the last 2^d particles, which for N large should be quite close.

Since each node of the tree has two weighted children, we need to interpolate between two weighted points p_1 and p_2 with weights w and $1 - w$. We first define the interpolated point:

$$p^\star = c(u, w)p_1 + (1 - c(u, w))p_2$$

We would like $c(u,w) = 1$ for $u = 0$, $c(u,w) = 0$ for $u = 1$, c continuous and monotonic in u and w, and $\mathrm{E}_u[c(u,w)] = w$. We can satisfy these constraints with

$$c(u,w) = \begin{cases} (1-u)^{\frac{1-w}{w}} & \text{if } w < \frac{1}{2}, \\ 1 - u^{\frac{w}{1-w}} & \text{if } w \geq \frac{1}{2}. \end{cases}$$

We also have $c(u,w) + c(1-u, 1-w) = 1$ which means that the choice of which point is labelled p_1 and which point is labelled p_2 does not matter.

To interpolate 2^d points we can interpolate the 2^{d-1} pairs of points on the bottom level to produce 2^{d-2} points, then interpolate the 2^{d-3} pairs of these points and so on, using the weights of each node.

4.2.4 Remarks

A cause for concern with this method is that it arbitrarily large discontinuities can be induced even for large N. Recall that in the median-cuts algorithm we have

$$\bigcup_{\mathbf{s} \in \{0,1\}^n} f_n(\mathbf{s}) = \mathbb{R}^d$$

For the weighted binary tree, we can define a string $\beta_{1:k}$ via

$$\beta_j = \begin{cases} 0 & \text{if } u_j < w_{j,0}, \\ 1 & \text{if } u_j \geq w_{j,0}. \end{cases}$$

Note that $\beta_{1:k}$ depends not only on $u_{1:d}$ but also the values of the weights of the nodes that are traversed. As such, fixed $u_{1:d}$ does not correspond to the same $\beta_{1:k}$ when the weights change. Let us define the set of particles in the node at level $j+1$ reached by a given $\beta_{1:j}$ as $b_j(\beta_{1:j})$.

Nevertheless, we have the analogous result that

$$\bigcup_{\boldsymbol{\beta} \in \{0,1\}^j} b_j(\boldsymbol{\beta}) = \bigcup_{i=1}^{N} \{\mathbf{x}_t^{(i)}\}$$

Furthermore, we have for any $\beta_{1:i} \in \{0,1\}^i$

$$\bigcup_{\beta_{i+1:j} \in \{0,1\}^{j-i}} b_j(\beta_{1:j}) = b_i(\beta_{1:i})$$

With fixed $\mathbf{u}$ across different runs of the filter, we can expect many $\beta_{1:k}$ to share the prefix $\beta_{1:i}$ for

some i. These particles maximum distance from each other is obviously bounded by the size of the region $b_i(\beta_{1:i})$ and shrinks as i increases. However, there exist $\mathbf{u}$ for any change in θ such that β_1 will be different across different runs. As such, arbitrarily large distances between selected particles across different runs can occur. The use of only d-dimensional uniform random vectors is used to temper the effects of this but it cannot be escaped. The problem is exacerbated when the density of $\mathbf{x}_t$ is very different in different regions[11] and it is made more frequent when the weights change drastically for even small changes in θ. The former is a property solely of the target density but the latter depends on the proposal density used to sample the particles.

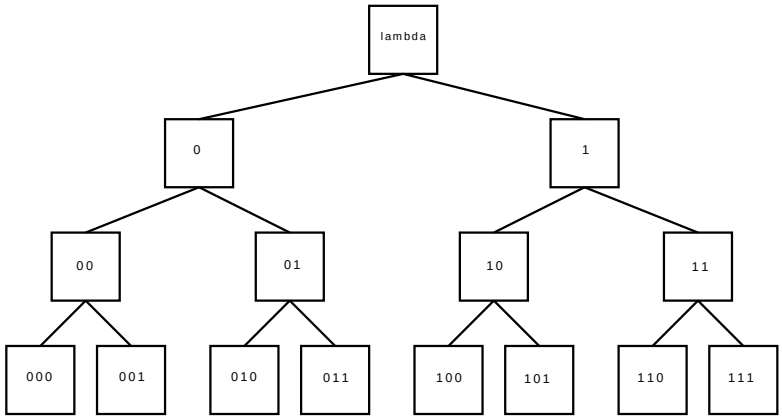


Figure 4.2: The binary tree representation of the sets as a function of $\boldsymbol{\beta}$

There are many ways of visualizing the sets of particles as a function of $\boldsymbol{\beta}$. One such way is to view them as a partitioning of the particles similar to that presented in Figure 3.1 with the understanding that the location of the hyperplanes (in arbitrary dimension d) that separate the space is determined by the proposal distribution and not the target distribution. Alternatively, one can think of the sets as nodes in a binary tree where each set at level i is given by the binary string $\beta_0 + \beta_{1:i}$ with $\beta_0 = \lambda$[12] A visualization of this binary tree is given in Figure 4.2.

4.3 Splitting via the Weighted Median (Unweighted Binary Trees)

An alternative to the algorithm presented in Section 4.2 is to split the particles such that the weight of each subset is always equal. The weighted median along a particular dimension is thus used to split the particles into two equally weighted sets of particles. Alternatively, one can see the binary

[11] If this is true, the use of $\mathbf{u} \in \mathbb{R}^d$ makes little difference since trying to be close to the region the next time we split along the same dimension does not affect the large differences to the regions selected in between.

[12] λ is the empty string.

tree that is constructed as being unweighted. Of course, the number of particles in each node is not equal. In addition, one particle must be split into two particles every time two child nodes are created so that the weight of each node is equal. However, the general construction of the tree is identical.

Algorithm 8 Constructing the unweighted binary tree: constructp(*particles*, *level*)

1. Create node *node*, empty sets $left$ and *right*, and set $w_{left} = w_{right} = 0$.
2. If *particles* contains only one particle, set $node.particle = p$ and return *node*.
3. Set dimension $r = level \bmod d$. If $r = 0$, set $r = d$.
4. Set particle $m =$ the *weighted* median of *particles* in the rth dimension.
5. For each particle p in *particles*
   - If the rth coordinate of p is less than the rth coordinate of m,
     - Add p to the set $left$.
     - Set $w_{left} = w_{left}$+weight(p).
   - If the rth coordinate of p is greater than the rth coordinate of m
     - Add p to the set *right*.
     - Set $w_{right} = w_{right}$+weight(p).
6. Create two particles m_1 and m_2 identical to m.
7. - Set weight(m_1) = (w_{right} + weight(m) - w_{left}) / 2
   - Set weight(m_2) = weight(m) - weight(m_1)
   - Add m_1 to the set $left$.
   - Add m_2 to the set *right*.
8. - Set $node.left$ = constructp($left$, $level + 1$)
   - Set $node.right$ = constructp(*right*, $level + 1$)
9. Return *node*.

Algorithm 9 Selecting a particle from the unweighted binary tree: selectp(*node*, *level*, u)

1. If *node* contains only one particle, return $node.particle$.
2. If $u \leq node.w_{left}$
   - Set $u = 2u$.
   - Return selectp($node.left$, $level + 1$, u)

   Else
   - Set $u = 2(u - 0.5)$.
   - Return selectp($node.right$, $level + 1$, u)

A key feature of this tree is that the same relative region is selected each time - it is the particles that make up the region that vary when the weights change. This is due to the unweighted nature of the nodes in the binary tree that is constructed. A drawback is that the imbalance in the number

of particles in sibling nodes and the introduction of an extra particle at every branch makes time complexity analysis more complicated.

4.3.1 Running Time

To analyze the running time, we first remark that the weighted median can also be computed in $O(N)$ time. Given points $x_1, \ldots, x_N$ with associated (normalized) weights $w_1, \ldots, w_N$, the weighted median is the point x_i such that

$$\sum_{x_j < x_i} w_j < 0.5 \text{ and } \sum_{x_j \leq x_i} w_j \geq 0.5$$

Note that we can find the unweighted median of $x_1, \ldots, x_N$, split the particles into two groups of $\frac{N}{2}$ particles using this median and compute the sum of the weights in each group in $O(N)$ time. We can pick which group contains the weighted median by looking at the weight of both groups and then repeating this process on that group. This type of recursion is repeated until the weighted median is found. The running time is thus given by the solution to the recursion $T(N) = O(N) + T(N/2)$, which is satisfied by $T(N) \in O(N)$.

Of course, in the case where we split by the unweighted median, the running time is in $O(N \log N)$ because the depth of the tree is in $O(\log N)$. However, in the worst case the tree constructed when splitting by the weighted median has depth N. Nevertheless, in most situations we expect the tree to have depth in $O(\log N)$ since the worst case is only achieved in pathological cases. In addition, the number of particles increases with the depth due to the replication of the median particle at each split. The running time is thus in

$$O(N) + O(N+1) + O(N+2) + O(N+4) + O(N+8) + \cdots + O(N + 2^{\text{depth}})$$

where we have $O(\log N)$ terms and $2^{\text{depth}} \in O(N)$ if the depth is in $O(\log N)$. Thus the overall running time for tree construction is in $O(N \log N)$. If we again assume that the depth of the tree is in $O(\log N)$, the selection of N particles also takes $O(N \log N)$ time.

4.3.2 Interpolation

Due to the imbalanced numbers of particles in the branches of the binary tree, a sophisticated interpolation scheme is a little more difficult to implement. Therefore, for this structure we only interpolate when we reach a node with exactly two particles and use the same method as in Section 4.2.3. In the case where the node reached has only particle (which is always due to an uneven split) we do not interpolate at all.

4.3.3 Remarks

This method is more involved implementation-wise compared to that of splitting by the unweighted median. However, it has the same computational complexity in non-pathological cases and it has asymptotic properties that we will show in Chapter 6 but are outlined below.

This approach can be seen as an approximation to the median-cuts approach introduced in Section 3.4.3. It is difficult to directly compare the two approaches since the median-cuts algorithm partitions the space for a given $u \in \mathbb{R}$. While the calculation of the splitting hyperplanes does not depend on u, this algorithm does not partition the space of $\mathbb{R}^d$ that it is not interested in - it calculates only n conditional medians $t_{1:n}$ despite selecting a region of probability $\frac{1}{2^n}$. In contrast, for reasons of computational complexity the unweighted binary tree is fully constructed so that each selection takes only $O(\log N)$ time. Indeed, this involves computation of $O(N)$ conditional medians of the empirical distribution. However, if we consider the selection of a single particle using the same u with the unweighted binary tree, we can denote by $\hat{t}_{1:k}$ the k conditional medians that split each node that is visited. Then, we can show that for any m, as $N \to \infty$, $\sqrt{N}(\widehat{t_{1:m}} - t_{1:m}) \xrightarrow{\text{D}} N(\mathbf{0}, \Sigma)$ for some Σ independent of N.

The major difference between this tree and the weighted binary tree is that there is no dependency on the weights of the particles for the relative region that is chosen - it is the constituent particles that change. In terms of the binary string representation of u, we can visualize the partitioning of the particles again using Figures 3.1 and 4.2 but since the string is independent of the weights of the particles the same relative region is picked. Of course, the effect of the changing membership of particles in each node in the unweighted binary tree should not be underestimated for practical values of N as this can lead to large discontinuities in the selected particles.

4.4 An Unweighted k-ary Tree

The last tree construction algorithm we consider here is similar to the weighted-median-splitting algorithm in that it produces an unweighted tree. However, unlike that approach each parent node in this tree has k children. Furthermore, we let $N = k^d$, where d is the dimension of the latent state variables. Therefore, as we increase N the number of children per node increases. In fact, sampling from this tree provides an approximation to the theoretical approach outlined in Algorithm 3.

Algorithm 10 Constructing the unweighted k-ary tree: constructk(*particles*, *level*)

1. Create node *node*.
2. If $level = d$,
   - Sort *particles* by dimension d.
   - Set $node.particles = particles$ and return *node*.
   - Normalize the particle weights.
3. Sort by dimension *level* and then partition *particles* into k sets $S_1, \ldots, S_k$ each with equal normalized weight of $1/k$ (replicating particles as necessary).
4. For $i = 1, \ldots, k$
   - Set $node.child(i) =$ constructk(S_i, $level + 1$).
5. Return *node*.

Algorithm 11 Selecting a particle from the unweighted k-ary tree: selectk(*node*, *level*, $u_1, \ldots, u_d$)

1. If $level = d$,
   - Find index i of $node.particles = p_1, \ldots$ such that $\sum_{j=1}^{i-1} \text{weight}(p_j) < u_d$ and $\sum_{j=1}^{i} \text{weight}(p_j) \geq u_d$
   - Return particle p_i
2. Return selectk($\lceil k u_{level} \rceil$, $level + 1$, $u_1, \ldots, u_d$)

4.4.1 Running Time

The computational time to construct the tree is clearly in $O(N \log N)$ since the computational effort at each level of construction is in $O(N \log N)$ and there are only d (which is independent of N) levels. However, the time to select is in expected $O(k)$ time since there are k particles on average in each leaf node (each of which are equally likely to be chosen). As such, the cost to select N particles is in $O(Nk) = O(N^{\frac{d+1}{d}})$. Of course, this is still in $o(N^2)$ but is slower than $O(N \log N)$.

4.4.2 Interpolation

The interpolation scheme used in this approach is slightly different to that of the binary trees. This is because we now have a variable number of particles in the leaf nodes of the tree. However, let us assume there are r particles in the leaf node reached. We can split these r particles into $r + 1$ blocks such that the first block contains only the first particle p_1, the $(r + 1)$th block contains the last particle p_r and every other block i contains two particles p_{i-1} and p_i. Furthermore, the weight associated with each block is the sum of the weights of the constituent particles divided by 2. As such, we proceed as normal to select a block and then within each block we use the remaining weight

to interpolate between the two remaining particles (in the case where the block selected is either the first or last, no interpolation is done).

4.4.3 Remarks

The unweighted k-ary tree approach we have just seen has the defining characteristic that the number of children increases with N. This allows for each coordinate of $\mathbf{u}$ to specify a shrinking region of the space as N increases, which allows for the sampled particles to more closely approximate the particles that would be returned by the theoretical Algorithm 3. In fact, we will show in Chapter 6 that if $F^{-1}(\mathbf{u}) = \boldsymbol{\mu}$ and the particle returned by the unweighted k-ary tree using $\mathbf{u}$ is denoted $\hat{\mathbf{x}}$, then as $N \to \infty$, $\hat{\mathbf{x}} \xrightarrow{\text{P}} \boldsymbol{\mu}$.

A practical disadvantage is that there is an imbalance in the accuracy of each dimension of the particle selected since less particles are used for estimation in successive levels of the tree. At the same time, for practical values of N it is very difficult to make the regions very small. For example, if $d = 3$ then using $N = 10^3 = 1000$ only allows us to partition the first dimension of the particles into 10 intervals. In addition, while we use all 1000 particles to approximate the interval values in the first dimension, only 100 particles are used to approximate the interval values in the second dimension and only 10 particles are available to represent the third dimension. In addition, since the particles are not partitioned very finely for practical values of N, interpolation can also introduce considerable bias to the procedure.

4.5 Overall Remarks

We have seen three different types of trees that can be constructed and sampled from in $o(N^2)$ time. Of course, there are many other possible variants since any properly weighted/constructed tree can be used to resample particles according to $\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$. Indeed, it is possible that a different type of tree could provide much better results than those presented above while still retaining $o(N^2)$ time complexity for construction and selection. Such a tree could utilize, for example, different and possibly variable numbers of child nodes, more aggressive replication of particles or a different method for partitioning the particles. Nevertheless, the three trees we have seen each showcase a different tree-based approach with unique characteristics.

The weighted binary tree approach is the quickest of the methods presented. While it is not asymptotically faster than the unweighted binary tree in non-pathological cases[13], the latter's reliance on the more computationally intensive weighted median in addition to the imbalanced nature of the

[13]The weighted binary tree approach's worst-case performance is asymptotically faster than that of the unweighted binary tree.

constructed tree means it is slower in practice. As such, the fact that the weighted binary tree does not give smooth samples in the asymptotic case does not necessarily make it worse, especially since the asymptotic case is not necessarily interesting in practice[14]. In addition, while the k-ary tree spends unequal numbers of particles and computation time on the different dimensions of the sampled particles, both binary tree approaches spend roughly equal time in each dimension since they cycle through the dimensions used to partition the particles.

The unweighted binary tree and k-ary trees are of interest mainly because of their asymptotic properties. In fact, empirical results suggest that the weighted binary tree outperforms the other two trees with respect to smoothness with practical values of N. As a final note, it should be obvious that all of the proposed trees can be seen as a generalization of the sorting idea in [23]. This is because in the case where $d = 1$, each of these trees are reduced to sorting along one dimension.

[14] As $N \to \infty$, we can using standard multinomial resampling since the likelihood estimate will converge towards the true likelihood which is a smooth function of θ. Alternatively, we can use CRN sequential importance sampling without resampling to guarantee continuity.

Chapter 5

Applications

To investigate the effectiveness of the the various tree-based resampling schemes in practice, we compare filters using the weighted binary tree, unweighted binary tree, weighted k-ary tree with the CRN[15] and plain vanilla particle filters in different applications. In all instances, the number of particles used for the vanilla filter is selected such that the time taken by the vanilla filter is slightly more than that taken by the smooth filter. Unless explictly stated, we use the importance distribution $q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1}^{(i)}) = p(\mathbf{x}_t|\mathbf{x}_{t-1}^{(i)})$.

5.1 Gaussian State-Space Model

We consider the model:

$$\begin{aligned}
\mathbf{x}_t &\sim N(A\mathbf{x}_{t-1}, \Sigma_1), t = 1, \ldots, T \\
\mathbf{y}_t &\sim N(\mathbf{x}_t, \Sigma_2), t = 1, \ldots, T \\
\mathbf{x}_0 &= \mathbf{0}
\end{aligned}$$

where $\mathbf{y}_t$ is observed for $t \in \{1, \ldots, T\}$.

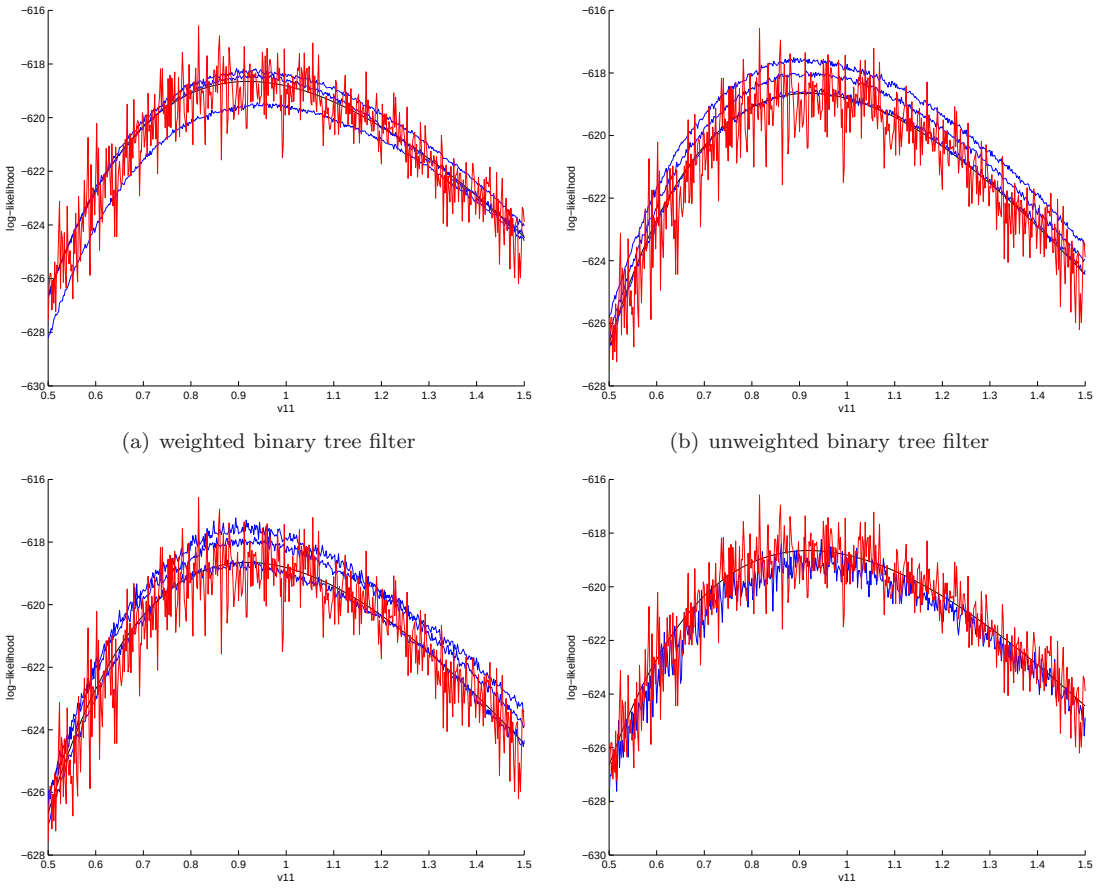
Two-Dimensional Case

To generate the observations we use

$$\Sigma_1 = \begin{pmatrix} v_{11} & \rho\sqrt{v_{11}v_{12}} \\ \rho\sqrt{v_{11}v_{12}} & v_{12} \end{pmatrix},$$

$$\Sigma_2 = \begin{pmatrix} v_{21} & 0 \\ 0 & v_{22} \end{pmatrix}$$

[15] The CRN vanilla filter is just a filter that uses the same random numbers each time.



(a) weighted binary tree filter

(b) unweighted binary tree filter

(c) unweighted k-ary tree filter

(d) CRN vanilla filter

Figure 5.1: Plot of the estimated log-likelihood for the 2D Gaussian state-space model as a function of v_{11} using the modified filters (blue) and the vanilla particle filter (red). The true log-likelihood given by the Kalman filter is shown in black. For the the CRN vanilla filter, we show estimates for only one set of common random numbers.

and

$$A = \begin{pmatrix} \phi & 0 \\ 0 & \phi \end{pmatrix}$$

with $v_{11} = v_{12} = 1$, $\rho = 0.8$, $\phi = 0.5$, $v_{21} = v_{22} = 0.5$ and $T = 200$.

We ran the Kalman filter, the tree-based particle filters with 1024 particles and the vanilla particle filters with 1536 particles, for 500 values of $v11$ from 0.5 to 1.5 and the other parameters set to the values used to generate the data. Figures 5.1 - 5.2 show the results. Each run of each particle filter took about 2 seconds on our machine. Since we have access to the true likelihood via the Kalman filter for this model, we also plot the statistical errors of the log-likelihood in Figure 5.3.

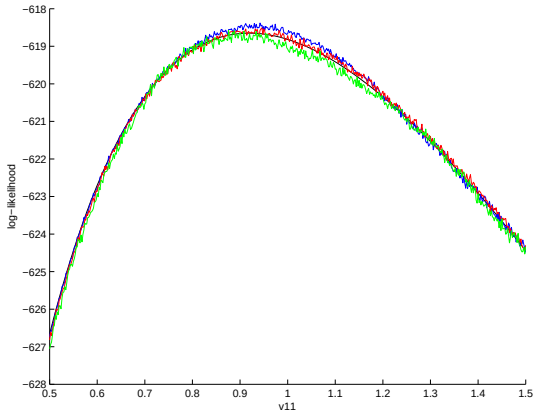
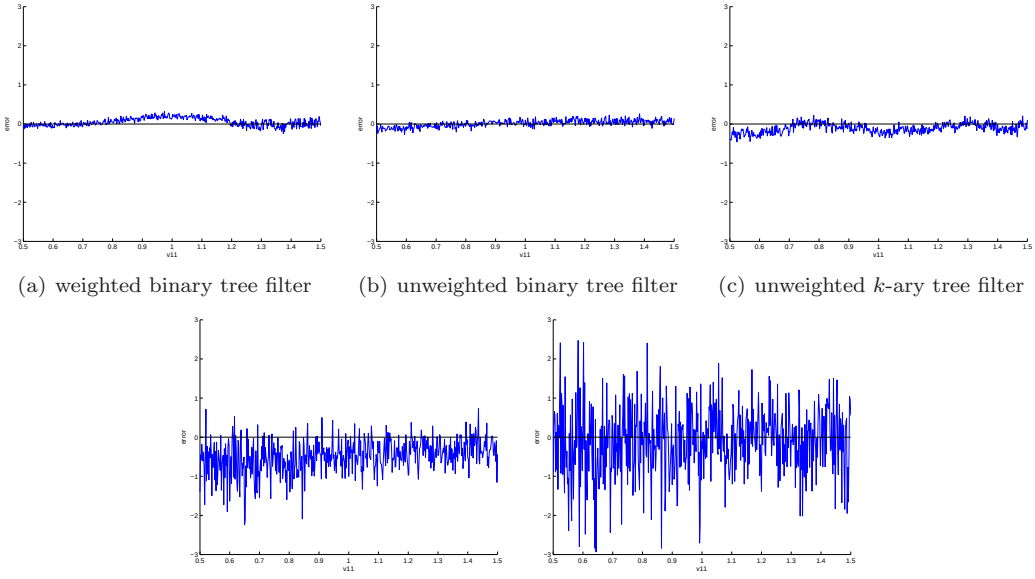


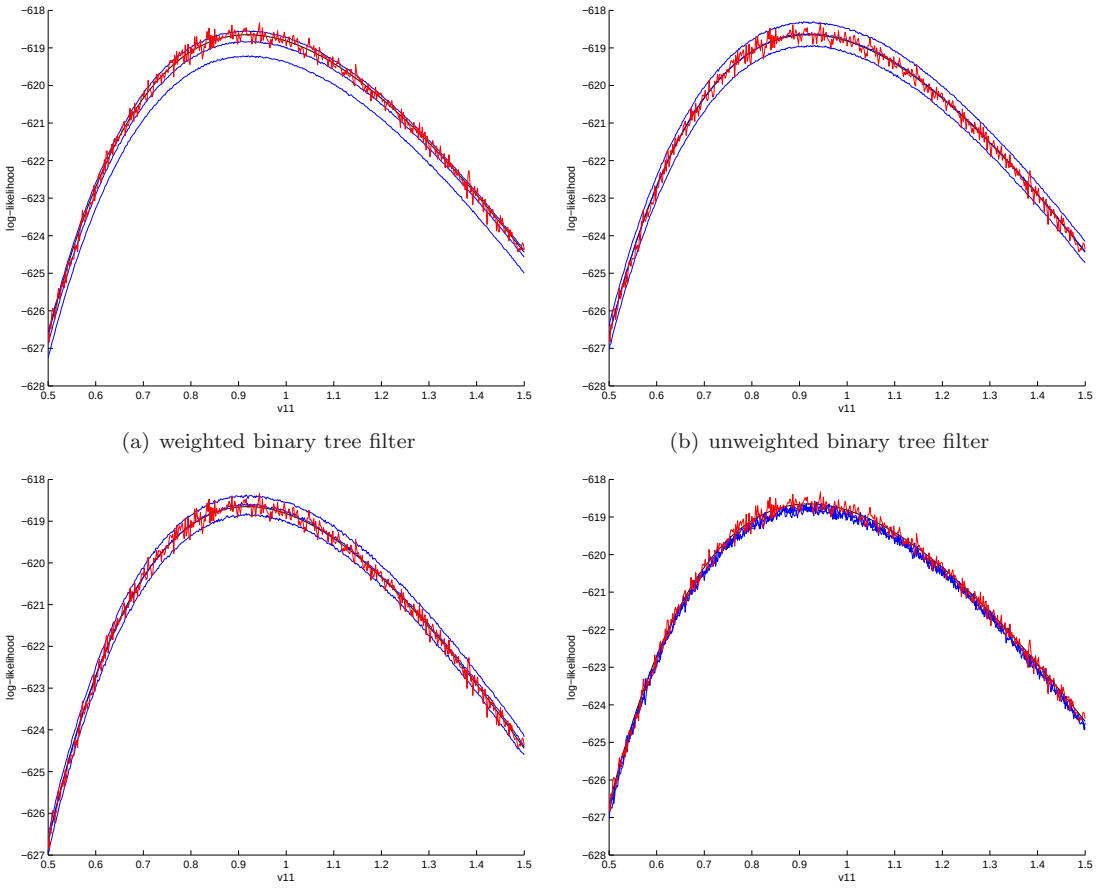
Figure 5.2: The plots of the estimated log-likelihood for the 2D Gaussian state-space model as a function of v_{11} using the weighted binary tree particle filter (blue), the unweighted binary tree particle filter (red) and the unweighted k-ary tree particle filter (green) together with the true log-likelihood given by the Kalman filter (black).



(a) weighted binary tree filter (b) unweighted binary tree filter (c) unweighted k-ary tree filter

(d) CRN vanilla filter (e) vanilla filter

Figure 5.3: Plot of the estimated log-likelihood errors for the 2D Gaussian state-space model as a function of v_{11} using the modified filters



(a) weighted binary tree filter

(b) unweighted binary tree filter

(c) unweighted k-ary tree filter

(d) CRN vanilla filter

Figure 5.4: Plot of the estimated log-likelihood for the 2D Gaussian state-space model as a function of v_{11} using the modified filters (blue) and the vanilla particle filter (red). The true log-likelihood given by the Kalman filter is shown in black. For the particle filters, the locally optimal proposal distribution is used. We show estimates for only one set of common random numbers using the CRN vanilla filter.

We also ran the filters using the 'locally optimal' proposal distribution

$$q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1}^{(i)}) \propto p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{x}_{t-1}^{(i)})$$

with the same values of the parameters. Figures 5.4 - 5.5 show the results. The statistical errors of the log-likelihood are plotted in Figure 5.6.

The estimates of the log-likelihood given by the various filters are slightly biased due to the use of logarithms. In addition, there is a small amount of bias from the interpolation schemes in the tree-based filters, which is reduced quickly as N increases. Table 5.1 shows how as N increases the estimates of the log-likelihood indicate very little bias and also that the sample variance decreases

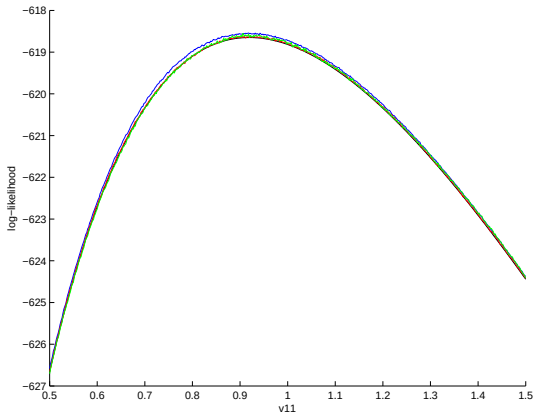
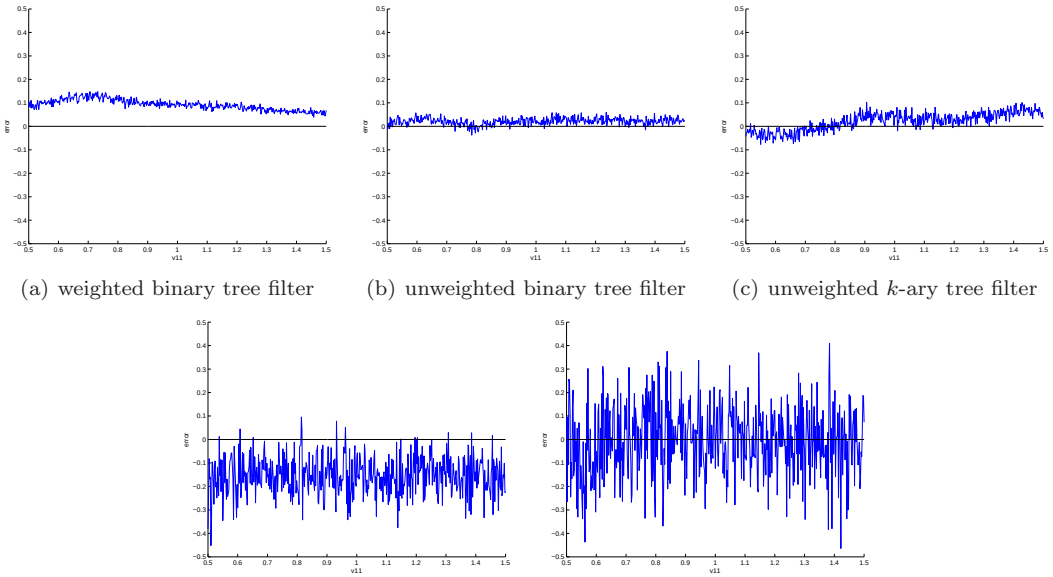


Figure 5.5: The plots of the estimated log-likelihood for the 2D Gaussian state-space model as a function of v_{11} using the weighted binary tree particle filter (blue), the unweighted binary tree particle filter (red) and the unweighted k-ary tree particle filter (green) together with the true log-likelihood given by the Kalman filter (black). For the particle filters, the locally optimal proposal distribution is used.



(a) weighted binary tree filter (b) unweighted binary tree filter (c) unweighted k-ary tree filter

(d) CRN vanilla filter (e) vanilla filter

Figure 5.6: Plot of the estimated log-likelihood errors for the 2D Gaussian state-space model as a function of v_{11} using the various filters with the locally optimal proposal distribution.

as $\frac{1}{N}$.

Table 5.1: Mean and standard deviation of the log-likelihood as N increases, with θ fixed. For each value of N, 100 filters were run with different random seeds to collect the data. The true value of the log-likelihood given by the Kalman filter is -618.8168.

N		vanilla	weighted binary	unweighted binary	k-ary
1024	$\hat{l}$	-619.02	-619.11	-618.67	-618.67
	$\hat{\sigma}$	1.01	0.97	1.05	1.06
2048	$\hat{l}$	-618.83	-618.82	-618.85	-618.86
	$\hat{\sigma}$	0.76	0.70	0.75	0.76
4096	$\hat{l}$	-618.85	-618.78	-618.82	-618.82
	$\hat{\sigma}$	0.51	0.50	0.54	0.53
8192	$\hat{l}$	-618.80	-618.81	-618.85	-618.84
	$\hat{\sigma}$	0.34	0.38	0.34	0.34
16384	$\hat{l}$	-618.81	-618.81	-618.79	-618.77
	$\hat{\sigma}$	0.27	0.24	0.26	0.26

Three-Dimensional Case

To generate the observations we use

$$\Sigma_1 = \begin{pmatrix} v_{11} & \rho_{12}\sqrt{v_{11}v_{12}} & \rho_{13}\sqrt{v_{11}v_{13}} \\ \rho_{12}\sqrt{v_{11}v_{12}} & v_{12} & \rho_{23}\sqrt{v_{12}v_{13}} \\ \rho_{13}\sqrt{v_{11}v_{13}} & \rho_{23}\sqrt{v_{12}v_{13}} & v_{13} \end{pmatrix},$$

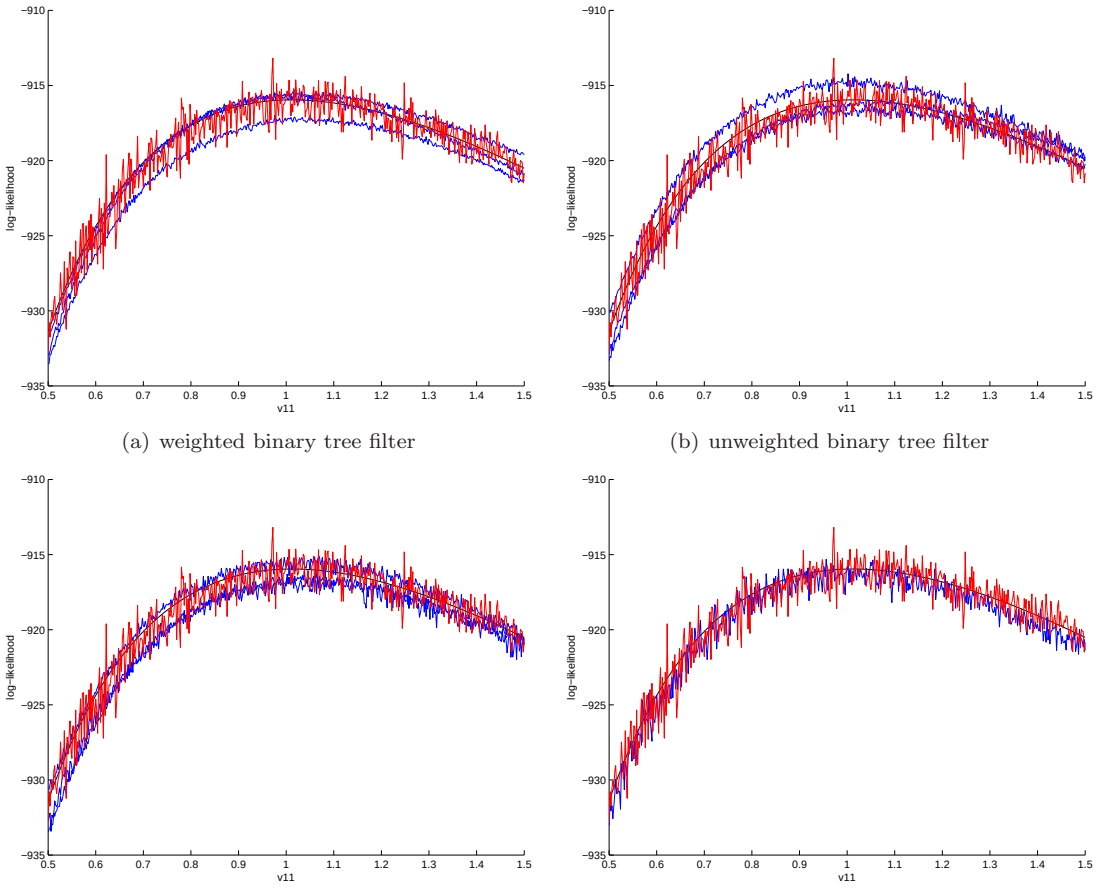
$$\Sigma_2 = \begin{pmatrix} v_{21} & 0 & 0 \\ 0 & v_{22} & 0 \\ 0 & 0 & v_{23} \end{pmatrix}$$

and

$$A = \begin{pmatrix} \phi & 0 & 0 \\ 0 & \phi & 0 \\ 0 & 0 & \phi \end{pmatrix}$$

with $v_{11} = v_{12} = v_{13} = 1$, $\rho_{12} = 0.8$, $\rho_{13} = 0.4$, $\rho_{23} = 0.4$, $\phi = 0.5$, $v_{21} = v_{22} = v_{23} = 0.5$ and $T = 200$.

We ran the Kalman filter, the tree-based particle filters with 2048 particles and the vanilla particle filters with 5120 particles, for 500 values of $v11$ from 0.5 to 1.5 and the other parameters set to the values used to generate the data. Figures 5.7 - 5.8 show the results. Each run of each particle filter



(a) weighted binary tree filter

(b) unweighted binary tree filter

(c) unweighted k-ary tree filter

(d) CRN vanilla filter

Figure 5.7: Plot of the estimated log-likelihood for the 3D Gaussian state-space model as a function of v_{11} using the modified filters (blue) and the vanilla particle filter (red). The true log-likelihood given by the Kalman filter is shown in black. For the CRN vanilla filter we show estimates for only one set of common random numbers.

took about 6.5 seconds on our machine. Since we have access to the true likelihood via the Kalman filter for this model, we also plot the statistical errors of the log-likelihood in Figure 5.9.

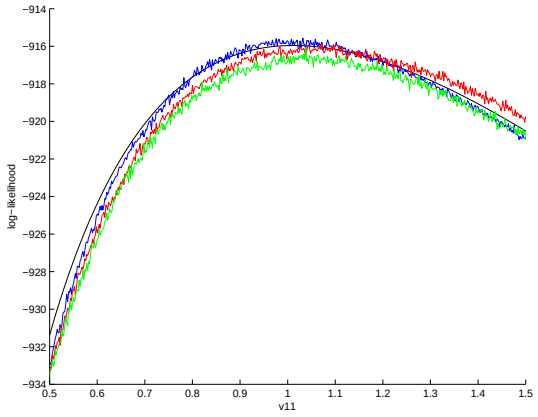
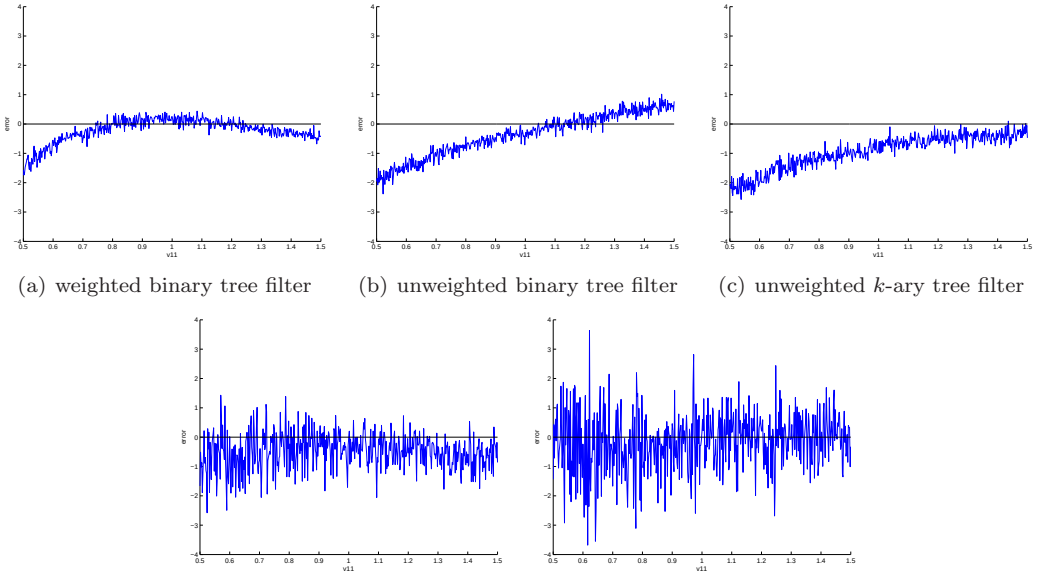


Figure 5.8: The plots of the estimated log-likelihood for the 3D Gaussian state-space model as a function of v_{11} using the weighted binary tree particle filter (blue), the unweighted binary tree particle filter (red) and the unweighted k-ary tree particle filter (green) together with the true log-likelihood given by the Kalman filter (black).



(a) weighted binary tree filter (b) unweighted binary tree filter (c) unweighted k-ary tree filter

(d) CRN vanilla filter (e) vanilla filter

Figure 5.9: Plot of the estimated log-likelihood for the 3D Gaussian state-space model as a function of v_{11} using the modified filters.

5.2 Factor Stochastic Volatility

We consider the factor stochastic volatility model most similar to that proposed in [20]:

$$\begin{aligned}\mathbf{y}_t &\sim N(\mathbf{B}\mathbf{f}_t, \mathbf{\Psi})\\ \mathbf{f}_t &\sim N(\mathbf{0}, \mathbf{H}_t)\\ \boldsymbol{\alpha}_t &\sim N(\mathbf{\Phi}\boldsymbol{\alpha}_{t-1}, \mathbf{U})\end{aligned}$$

where

$$\begin{aligned}\mathbf{\Psi} &\stackrel{\text{def}}{=} \text{diag}(\psi_1, \ldots, \psi_M)\\ \mathbf{H}_t &\stackrel{\text{def}}{=} \text{diag}(\exp(\boldsymbol{\alpha}_t))\\ \mathbf{\Phi} &\stackrel{\text{def}}{=} \text{diag}(\phi_1, \ldots, \phi_K)\end{aligned}$$

Here, $\mathbf{f}_t$ is K-dimensional, $\mathbf{y}_t$ is M-dimensional and $\mathbf{B}$ is an $M \times K$ factor loading matrix. The latent variable at each time step t is the K-dimensional vector $\boldsymbol{\alpha}_t$. Importantly, $\mathbf{U}$ is allowed to have non-zero off-diagonal entries to allow for correlation in the changes in log factor variances with time. The likelihood of the data, $\mathbf{y}_t$, given $\boldsymbol{\alpha}_t$ is Gaussian with:

$$\mathbf{y}_t|\boldsymbol{\alpha}_t \sim N(\mathbf{0}, \mathbf{B}\mathbf{H}\mathbf{B}^T + \mathbf{\Psi})$$

For identifiablity, the factor loading matrix has the form:

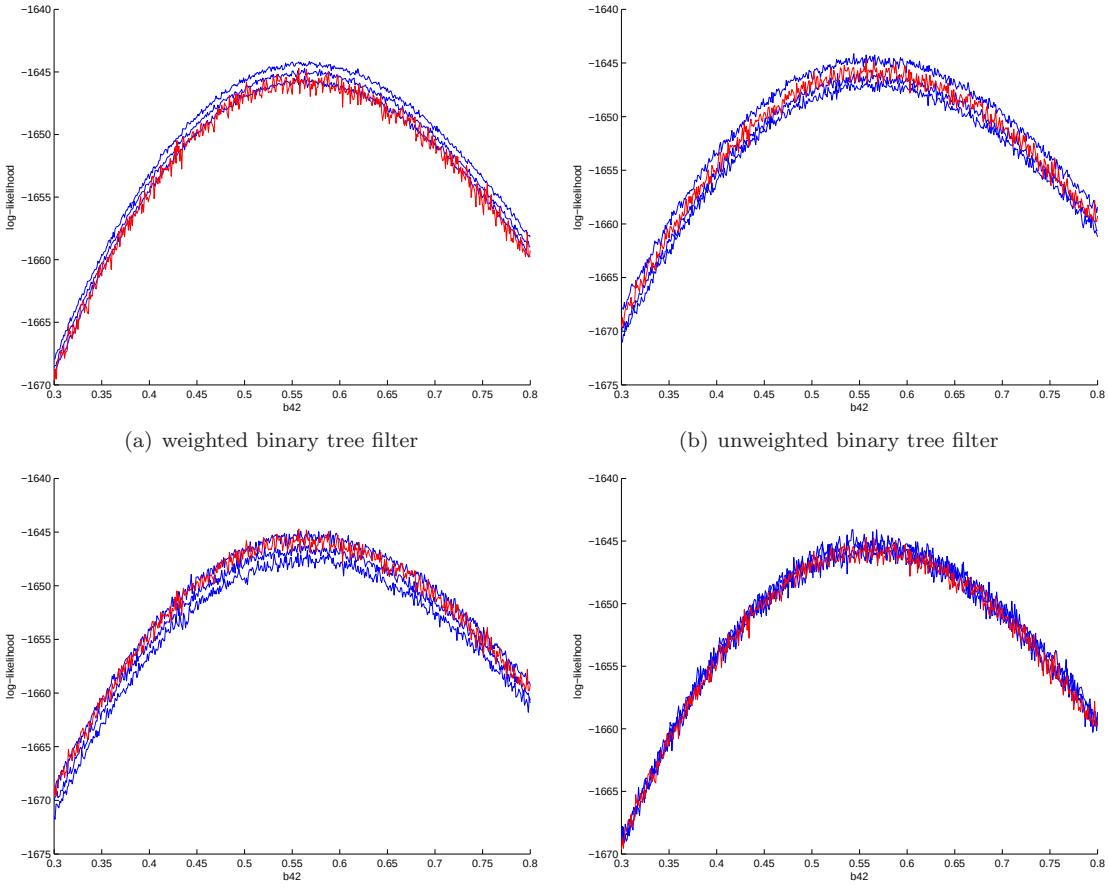
$$B = \begin{pmatrix} 1 & 0 & 0 \\ b_{21} & 1 & 0 \\ b_{31} & b_{32} & 1 \\ b_{41} & b_{42} & b_{43} \\ b_{51} & b_{52} & b_{53} \end{pmatrix}$$

We generate data using $\boldsymbol{\alpha}_0 = \mathbf{0}, \psi_i = 0.5, i \in \{1, \ldots, M\}$, $\phi_i = 0.9, i \in \{1, \ldots, K\}$

$$U = \begin{pmatrix} 0.5 & 0.2 & 0.1 \\ 0.2 & 0.5 & 0.2 \\ 0.1 & 0.2 & 0.5 \end{pmatrix} \text{ and } B = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.5 & 0.5 & 1 \\ 0.2 & 0.6 & 0.3 \\ 0.8 & 0.7 & 0.5 \end{pmatrix}$$

We do not consider time-varying covariances as per [24] here because this would increase the dimension of the latent variable space to $M + K$ as opposed to the current dimension of K. While the same smooth filtering algorithm would be applicable with higher-dimensional latent variables, the number of particles required to approach smoothness increases with dimension.

We ran the tree-based particle filters with 1024 particles and the vanilla particle filters with 1536 particles, for 500 values of b_{42} from 0.3 to 0.8 and the other parameters set to the values used to generate the data. Figures 5.10 - 5.11 show the results. Each run of each particle filter took about 6 seconds on our machine.



(a) weighted binary tree filter

(b) unweighted binary tree filter

(c) unweighted k-ary tree filter

(d) CRN vanilla filter

Figure 5.10: Plot of the estimated log-likelihood for the FSV model as a function of b_{42} using the modified filters (blue) and the vanilla particle filter (red). For the CRN vanilla filter we show estimates for only one set of common random numbers.

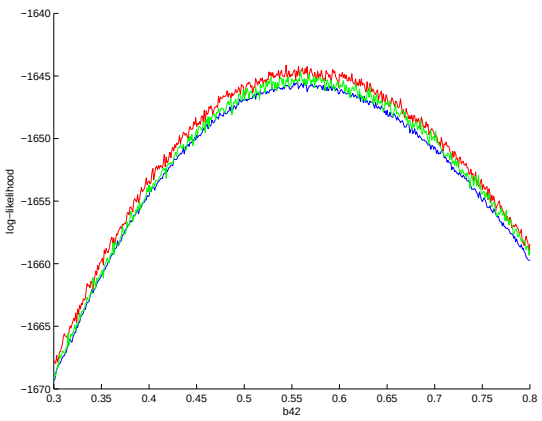


Figure 5.11: The plots of the estimated log-likelihood for the FSV model as a function of b_{42} using the weighted binary tree particle filter (blue), the unweighted binary tree particle filter (red) and the unweighted k-ary tree particle filter (green).

5.3 Stochastic Kinetic Model

Stochastic kinetic models are important in systems biology [4]. One of the simpler models in this class is the Lotka-Volterra model, which describes the evolution of prey ($Z_t^{(1)}$) and predator ($Z_t^{(2)}$) in time. In particular, in a small time interval $(t, t+dt]$ the process evolves according to

$$\begin{aligned}
\Pr(Z_{t+dt}^{(1)} = z_t^{(1)} + 1, Z_{t+dt}^{(2)} = z_t^{(2)} | z_t^{(1)}, z_t^{(2)}) &= \alpha z_t^{(1)} + o(dt), \\
\Pr(Z_{t+dt}^{(1)} = z_t^{(1)} - 1, Z_{t+dt}^{(2)} = z_t^{(2)} + 1 | z_t^{(1)}, z_t^{(2)}) &= \beta z_t^{(1)} z_t^{(2)} + o(dt), \\
\Pr(Z_{t+dt}^{(1)} = z_t^{(1)}, Z_{t+dt}^{(2)} = z_t^{(2)} - 1 | z_t^{(1)}, z_t^{(2)}) &= \gamma z_t^{(2)} + o(dt)
\end{aligned}$$

where each equation corresponds to prey reproduction, predator reproduction and predator death, respectively.

This model, and stochastic kinetic models in general, are straightforward to simulate using the Gillespie algorithm with time complexity linear in the number of reactions. However, it is often the case that we wish to estimate the posterior distribution of $\theta \stackrel{\text{def}}{=} (\alpha, \beta, \gamma)$ given discrete (ie. incomplete) observations of the reactants $\{\mathbf{Z}_t | t \in \mathcal{I} \subset \mathbb{R}\}$. The main obstacle to doing this in a reasonable amount of time is that it is difficult to generate low variance estimates of $p(\mathbf{z}_j | \mathbf{z}_i, \theta)$ in acceptable time. For example, the Monte Carlo estimate

$$\hat{p}(\mathbf{z}_j | \mathbf{z}_i) = \frac{1}{N} \sum_{k=1}^{N} \delta_{\mathbf{z}_j}(\mathbf{x}^{(k)})$$

where $\mathbf{x}^{(k)} \sim p(\cdot | \mathbf{z}_i)$ has unacceptable variance in general when many reactions occur in the time

$j-i$. Even with recent methods proposed in the 'likelihood-free' inference literature [21, 3, 28], such estimates remain unacceptable because the simulation process is naturally diffuse.

It is, however, possible to sample from this posterior effectively using Markov Chain Monte Carlo methods by including the intermediate (unobserved) population levels as auxiliary variables in an extended state-space [4] but this approach can lack computational efficiency for large models. A promising alternative to dealing with the stochastic kinetic model is to approximate the true evolution process with a nonlinear diffusion process observed with noise for the purposes of parameter estimation [12]. This diffusion approximation has been used to speed up MCMC methods for approximate inference of the posterior of θ since the number of intermediate population numbers is far lower. In particular, the number of intermediate values is $\frac{T}{\epsilon} - 1$ where $[0, T]$ is the time interval of interest and ϵ is the time discretization step, which is advantageous computationally when the number of expected reactions in time T far exceeds $\frac{T}{\epsilon}$. Another recent advance in inference with nonlinear diffusion models involves utilising a modified diffusion bridge [13] in the spirit of [9].

The approach proposed here is to use sequential Monte Carlo to estimate $p(\{\mathbf{Z}_t | t \in \mathcal{I}\}|\theta)$ smoothly for given values of θ using the diffusion approximation at each step of a particle filter for the discretized diffusion process. In addition, we use observations of only $Z_t^{(1)}$, corresponding to the what is called the partially observed case in the literature. This is because SMC seems particularly well-suited to inference in this context since each trajectory includes an estimate of $X^{(2)} \approx Z^{(2)}$ for each time.

The diffusion process follows the model:

$$\begin{aligned}\mathbf{X}_t &\sim N(\mathbf{X}_{t-\epsilon} + \epsilon\boldsymbol{\mu}, \epsilon\Sigma_1), t = \epsilon, 2\epsilon, \ldots, T-\epsilon, T \\ \mathbf{Z}_t &\sim N(\mathbf{X}_t, \Sigma_2), t = \Delta, 2\Delta, \ldots, T-\Delta, T\end{aligned}$$

where μ and Σ_1 are the drift and instantaneous variance-covariance respectively. For the Lotka-Volterra model we use drift and instantaneous variance-covariance given in [10]:

$$\boldsymbol{\mu} = \begin{pmatrix} \alpha X^{(1)} - \beta X^{(1)} X^{(2)} \\ \beta X^{(1)} X^{(2)} - \gamma X^{(2)} \end{pmatrix} \text{ and } \Sigma_1 = \begin{pmatrix} \alpha X^{(1)} + \beta X^{(1)} X^{(2)} & -\beta X^{(1)} X^{(2)} \\ -\beta X^{(1)} X^{(2)} & \beta X^{(1)} X^{(2)} + \gamma X^{(2)} \end{pmatrix}$$

Σ_2, which encodes the noisiness of the observations, typically has only non-zero diagonal entries.

To generate the data we use the Gillespie algorithm with $\mathbf{X}_0 = (100, 100), \alpha = 0.5$, $\beta = 0.0025$, $\gamma = 0.3$ and $T = 20$. Observations are taken with $\Delta = 0.1$. The particle filter uses the diffusion approximation with $\epsilon = 0.01$ and $\Sigma_2 = \mathrm{I}$. The total number of time steps in the filter is 200 but each sample from the proposal involves sampling from 10 diffusion distributions.

We ran the tree-based particle filters with 1024 particles and the vanilla particle filters with 1536 particles, for 500 values of γ from 0.2 to 0.4 and the other parameters set to the values used to

generate the data. Figures 5.13 - 5.14 show the results. Each run of each particle filter took about 11 seconds on our machine. In Figure 5.12 we show the mean population estimates at each time from the particle filter with the parameters used to generate the data. Since the predator levels are not observed, they have more variance with respect to the true levels.

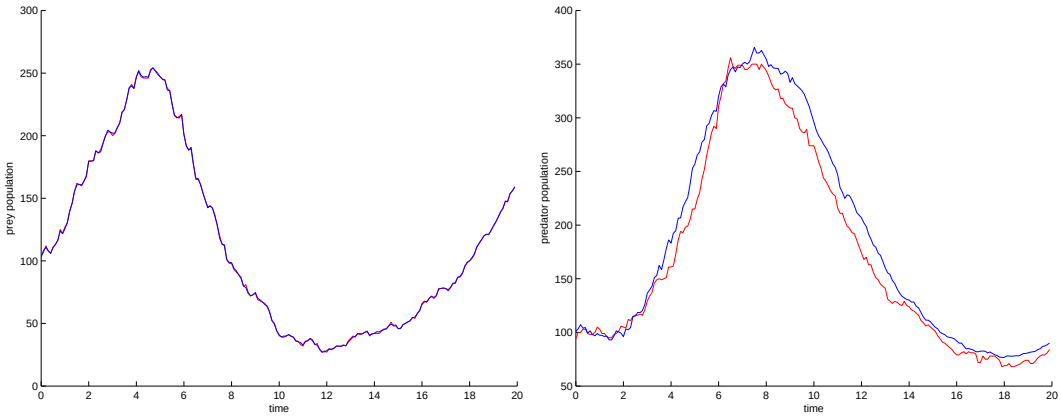
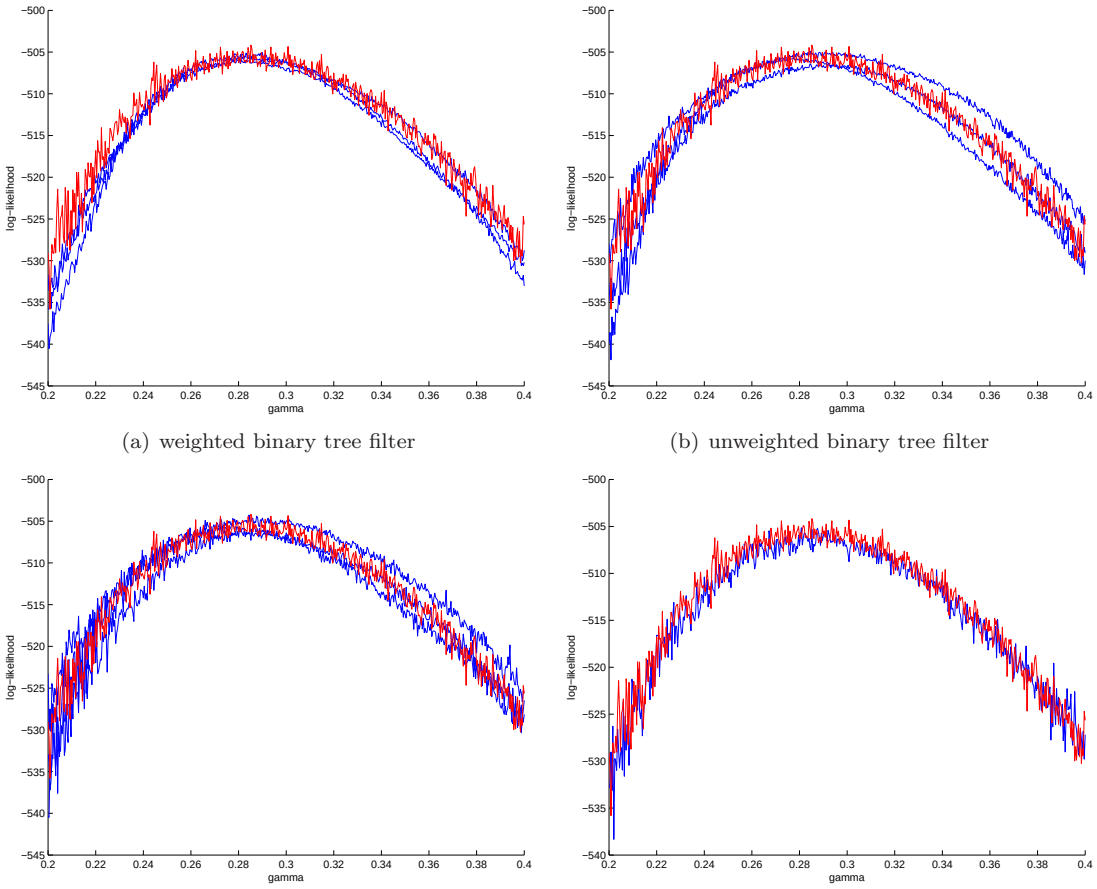


Figure 5.12: Left: The plots of the true prey population level (red) and estimated expected prey population level (blue) at each time. Right: The plots of the true predator population level (red) and estimated expected predator population level (blue) at each time.

5.4 Dynamic Stochastic General Equilibrium Model

Dynamic stochastic general equilibrium (DGSE) models are used to explain macroeconomic phenomena like growth and business cycles. These models specify the preferences of agents, the actions they can take and the (stochastic) dynamics of the environment. Although they are difficult to solve in practice, these models allow us to compute distributions over various quantities of interest like output, consumption and employment as well as predicting the effect of changing structural parameters of the model. In addition, given data from such a system, it is possible to compute the likelihood of the data for various values of the structural parameters of the model. The defining characteristic of such models is that all agents seek to maximize their expected utility whenever choosing an action and it is this rationality condition that makes such models so difficult to solve in practice.

Following [7], we restrict our attention to a simple case of the DSGE model: a single sector real business cycle (RBC) model. This type of model explains short-run fluctuations in economic indicators via stochastic shocks to the system [19]. Let q_t, k_t, c_t, i_t and a_t represent output, capital, consumption, investment and total factor productivity respectively. To simplify the model, it is assumed that labour is fixed at 1. The model assumes we observe q_t and i_t noisily for a representative



(a) weighted binary tree filter

(b) unweighted binary tree filter

(c) unweighted k-ary tree filter

(d) CRN vanilla filter

Figure 5.13: Plot of the estimated log-likelihood for the partially observed Lotka-Volterra model as a function of γ using the modified filters[17] (blue) and the vanilla particle filter (red). For the CRN vanilla filter we show estimates for only one set of common random numbers.

agent who at every time t maximizes expected lifetime utility:

$$U_t = E\left[\sum_{s=t}^{\infty} \beta^{s-t} \log(c_t) \middle| q_{0:t}, k_{0:t}\right]$$

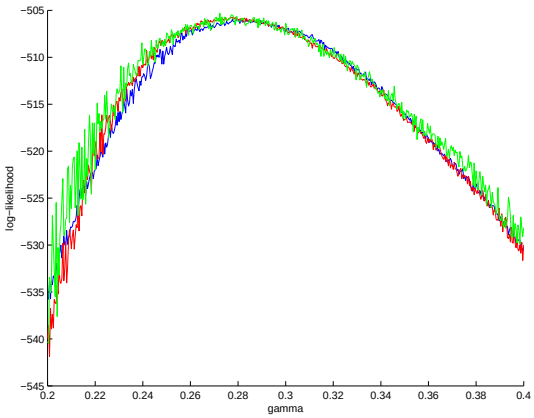


Figure 5.14: The plots of the estimated log-likelihood for the partially observed stochastic kinetic model as a function of γ using the weighted binary tree particle filter (blue), the unweighted binary tree particle filter (red) and the unweighted k-ary tree particle filter (green).

subject to the dynamics

$$\begin{aligned} q_t &= a_t^{1-\alpha} k_t^{\alpha} \\ c_t &= q_t - i_t \\ k_{t+1} &= i_t + (1-\delta)k_t \\ \log(a_{t+1}) &= \rho \log(a_t) + v_{t+1} \\ v_t &\sim N(0, \sigma^2) \end{aligned}$$

and given the parameters α, β, δ, ρ and σ. α corresponds to capital's share of output, γ is the discount factor, δ is capital depreciation and (ρ, σ) determines the type of shocks in the system. The observations are given by

$$\mathbf{y}_t \sim N\left(\begin{bmatrix} q_t \\ i_t \end{bmatrix}, \begin{bmatrix} \sigma_q^2 & 0 \\ 0 & \sigma_i^2 \end{bmatrix} \right)$$

where σ_q and σ_i are also structural parameters. We further assume $\delta = 0$, ie. full depreciation. This simplifies the model so that the consumption policy $c_t(q_t, a_t)$ that maximizes expected utility can be found analytically. In this case, the solution is given by $c_t(q_t, a_t) = (1 - \alpha\beta)q_t$ (see, for example [27]) which implies $i_t = \alpha\beta q_t$. As such, we have the following identities

$$\begin{aligned} k_{t+1} &= \alpha\beta q_t \\ q_{t+1} &= a_{t+1}^{1-\alpha} (\alpha\beta q_t)^{\alpha} \end{aligned}$$

This suggests use of $s_t = (z_t, q_t)$ as the state variable in the state-space model, where $z_t = log(a_t)$.

The dynamics are fully captured by

$$z_{t+1} = \rho z_t + v_{t+1}$$
$$q_{t+1} = (e^{z_t+1})^{1-\alpha}(\alpha\beta q_t)^{\alpha}$$

with the distribution over (z_0, q_0) given. The observation equations are then

$$\mathbf{y}_t \sim N\left(\begin{bmatrix} 1 \\ \alpha\beta \end{bmatrix} q_t, \begin{bmatrix} \sigma_q^2 & 0 \\ 0 & \sigma_i^2 \end{bmatrix}\right)$$

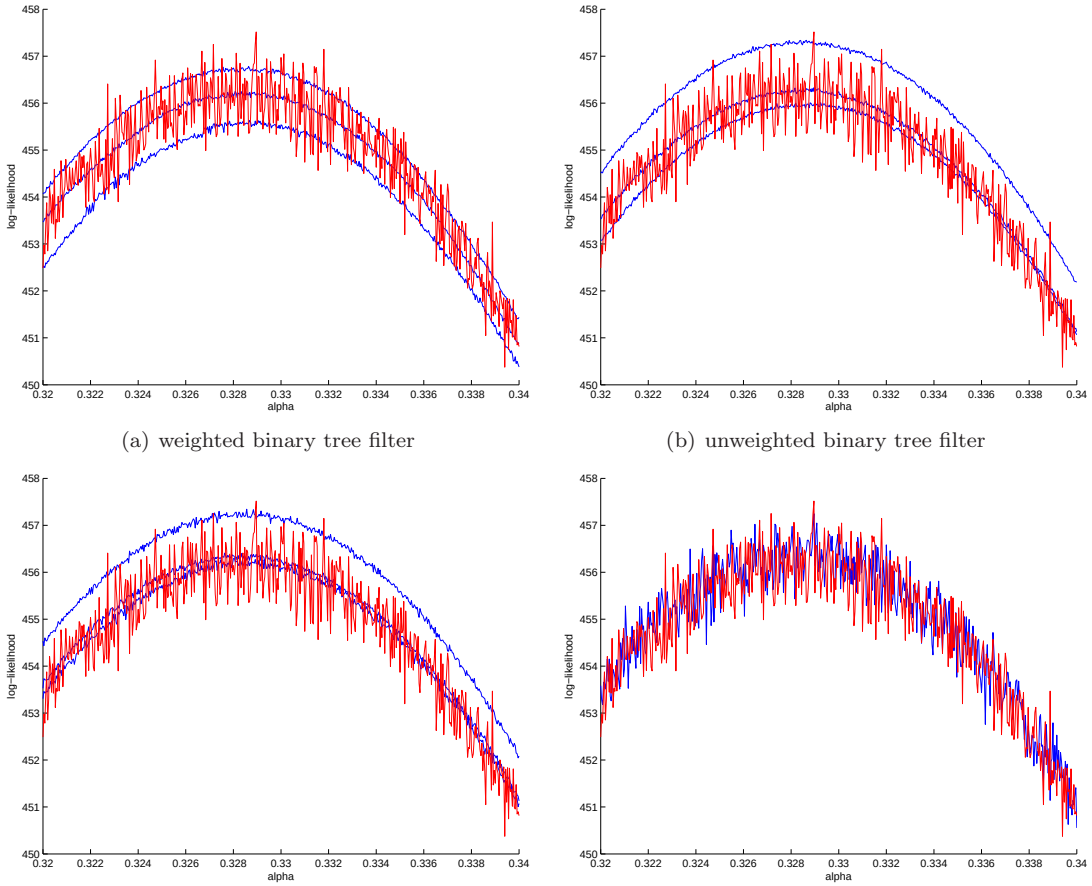
We simulated observations using parameters $(\alpha, \beta, \rho, \sigma, \sigma_q, \sigma_i) = (0.33, 0.96, 0.8, 0.05, 0.014, 0.02)$. We also set $(z_0, q_0) = (0, 1)$. We ran the tree-based particle filters with 1024 particles and the vanilla particle filters with 1536 particles, for 500 values of α between 0.32 and 0.34 and the other parameters set to the values used to generate the data. Figures 5.15 - 5.16 show the results. Each run of each particle filter took about 1.5 seconds on our machine. To show how large the improvement in smoothness is we also ran the vanilla particle filter with 20000 particles to compare with the results from the weighted binary tree filter with only 1024 particles, despite the large difference in computation time. The results are shown in Figure 5.17. For the graphs associated with this simple DSGE model, we plot estimates of the log-likelihood for three different sets of common random numbers to emphasise that the variance of any individual log-likelihood estimate is not improved by our resampling techniques. It is the variance in the difference between log-likelihood estimates for different values of θ that is reduced.

5.5 Remarks

The applications we have seen all show a marked improvement in the smoothness of log-likelihood estimates for the tree-based particle filters over the CRN vanilla and vanilla particle filters. The most impressive algorithm is the weighted binary tree filter, despite the fact that it does not have interesting asymptotic properties like the unweighted tree filters. This highlights the fact that asymptotic properties (and their lack) can be misleading when assessing the efficacy of finite-sample methods.

One characteristic of the tree-based resampling schemes is that while they are not much slower than the vanilla filter, they always require more time to run. As such, for a fixed computational budget the number of particles they can use is less. Naturally, this means that their estimates of the log-likelihood have higher variance than their non-smooth counterparts given fixed time.

It is worth mentioning that in the process of applying the tree-based resampling approaches to these



(a) weighted binary tree filter

(b) unweighted binary tree filter

(c) unweighted k-ary tree filter

(d) CRN vanilla filter

Figure 5.15: Plot of the estimated log-likelihood for the DSGE model as a function of α using the modified filters (blue) and the vanilla particle filter (red). For the vanilla filter we show estimates for only one set of common random numbers.

Table 5.2: Running time (in seconds) for the vanilla, tree-based and $O(N^2)$ vanilla filter for various values of N.

N	vanilla	tree-based	$O(N^2)$ vanilla
64	0.07	0.14	2.5
128	0.125	0.2	10
256	0.26	0.5	40
512	0.5	1	150
1024	1	2	600
2048	2.3	5	2500
4096	4.6	10	9000

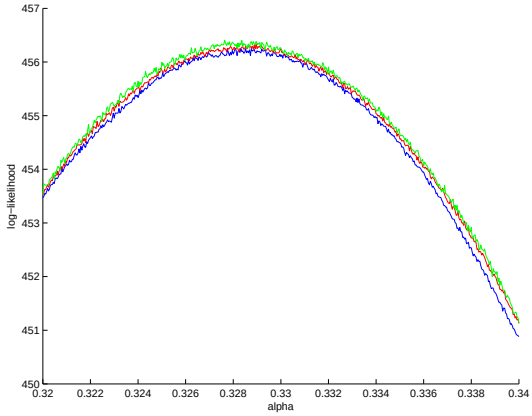


Figure 5.16: The plots of the estimated log-likelihood for the DSGE model as a function of α using the weighted binary tree particle filter (blue), the unweighted binary tree particle filter (red) and the unweighted k-ary tree particle filter (green).

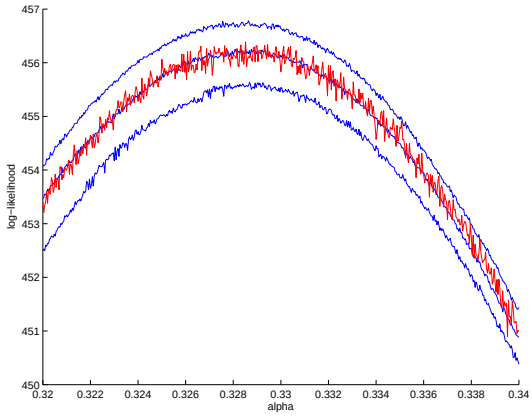


Figure 5.17: The plots of the estimated log-likelihood for the DSGE model as a function of α using the weighted binary tree particle filter with 1024 particles (blue) and the vanilla particle filter with 20000 particles (red).

application domains, we considered the effect of adding only the $O(N^2)$ computation

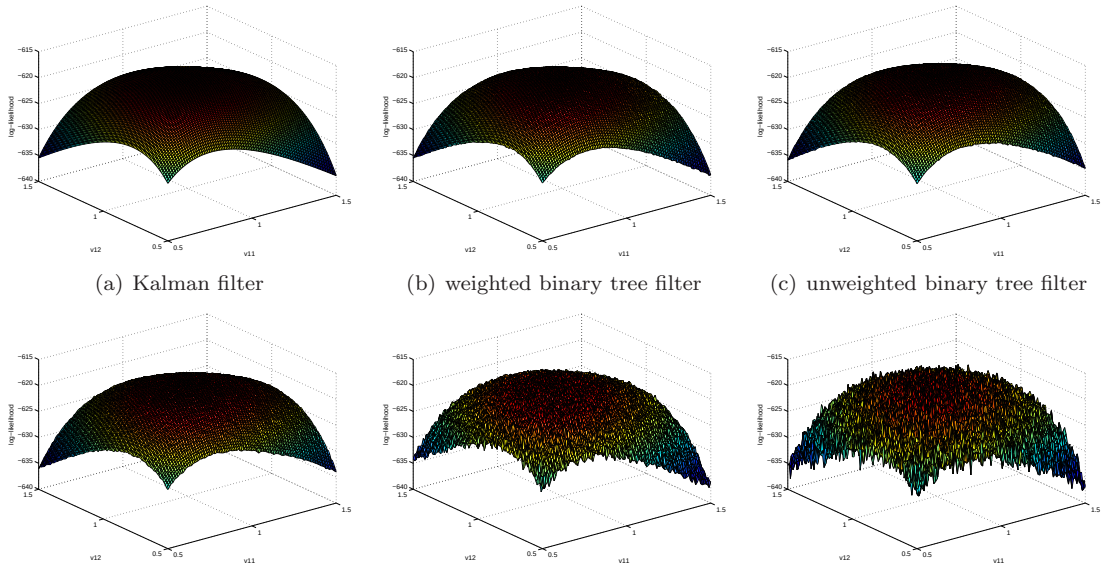
$$\hat{p}_N(\mathbf{x}_{t+1}^{(j)}|\mathbf{y}_{0:t}) = \sum_{i=1}^{N} W_t^{(i)} p(\mathbf{x}_{t+1}^{(j)}|\mathbf{x}_t^{(i)})$$

to the vanilla filter for the 2D Gaussian state-space model. This shows insight into the efficiency ramifications of the approach discussed in Section 3.3.2. Table 5.2 shows a comparison of the running times for the vanilla filter, our tree-based methods[18] and the $O(N^2)$ vanilla filter for some small

[18]While the tree-based methods do exhibit different time complexities, for small values of N these differences are almost negligible.

values of N. Indeed for $N > 128$ the running time quickly blows up to levels that would generally be unmanageable in real-world applications. At the same time, $N \leq 128$ severely limits the types of distributions that can be represented accurately. Nevertheless, the applications in [7] usually use $N = 100$ particles to represent $p(\mathbf{x}_{0:t}|\mathbf{y}_{0:t})$ for the purpose of filtering and this leads to reasonable results for the applications they consider. For estimation of the log-likelihood more particles are used since they need not be resampled. In constrast, for our tree-based methods we found that using more particles only for estimation of the log-likelihood was not as helpful as increasing the number of particles for both filtering and estimation (keeping the computation time constant).

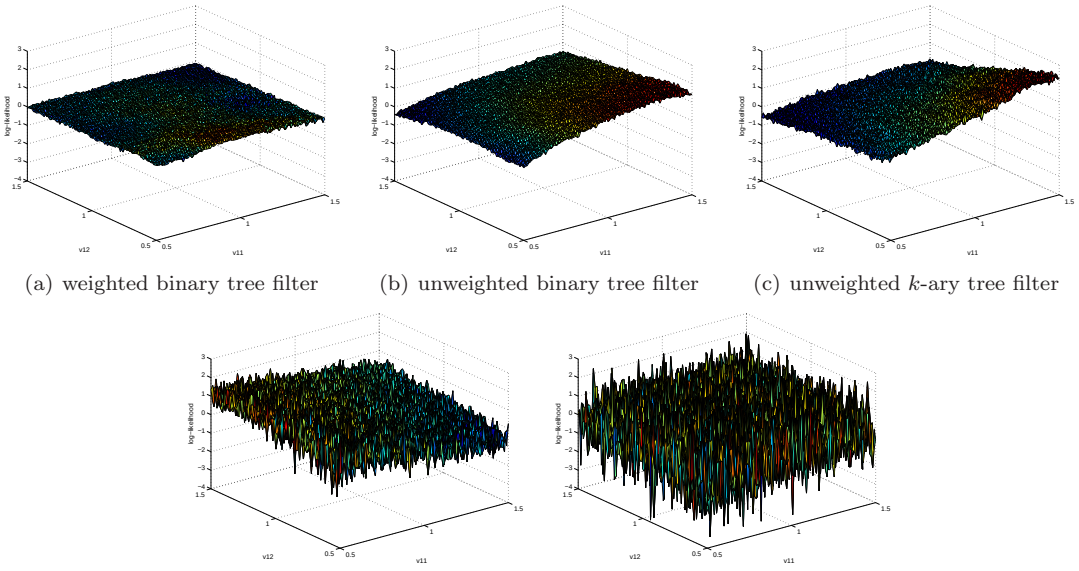
Lastly, we have shown log-likelihood plots with only one parameter changing since this allows the differences in smoothness to be seen very easily. For the two-dimensional Gaussian state-space model, Figure 5.18 shows surface maps for the log-likelihood where $v_1 1$ and $v_1 2$ each range from 0.5 to 1.5 with 10000 different data points in total. Figure 5.19 shows the statistical errors.



(a) Kalman filter (b) weighted binary tree filter (c) unweighted binary tree filter

(d) unweighted k-ary tree filter (e) CRN vanilla filter (f) vanilla filter

Figure 5.18: Surface maps of the estimated log-likelihood for the 2D Gaussian state-space model as a function of v_{11} and v_{12} for the various filters.



(a) weighted binary tree filter (b) unweighted binary tree filter (c) unweighted k-ary tree filter

(d) CRN vanilla filter (e) vanilla filter

Figure 5.19: Surface map of the estimated log-likelihood errors for the 2D Gaussian state-space model as a function of v_{11} and v_{12} using the various filters

Chapter 6

Analysis

In this chapter we show the asymptotic properties of the unweighted binary and k-ary tree algorithms. To begin, we first present some results that are used in our analysis.

6.1 Preliminaries

6.1.1 Quantile Estimation

In all of our methods, we have used empirical estimates of quantiles to partition the particles. Therefore, in order to understand the asymptotic properties of our methods it is necessary to show the asymptotic properties of these quantile estimates. In particular, the analysis of quantile estimates is complicated by the fact that the quantile is not a linear statistical functional. However, it has been shown that such estimates do satisfy asymptotic normality results under very weak regularity conditions.

Classical Monte Carlo Estimates

Assume we want to compute $\alpha = F^{-1}(p)$ where F is a c.d.f. and $p \in [0,1]$. Thus α is the pth quantile of F. Let $f(x) = F'(x)$ be the p.d.f. of the distribution of interest. The classical Monte Carlo estimate of α with N i.i.d. samples $\{x_i\}_{i=1}^N$ from f is given by

$$\alpha_N = F_N^{-1}(p) \stackrel{\text{def}}{=} \inf\{x : F_N(x) \geq p\}$$

where F_N is the empirical distribution function

$$F_N(x) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(x_i < x)$$

It is well-known that $\sqrt{N}(\alpha_N - \alpha) \xrightarrow{\text{D}} N(0, \sigma^2)$ as $N \to \infty$, where

$$\sigma^2 = \frac{p(1-p)}{f(\alpha)^2}$$

as long as f exists and is continuous at α.

Importance Sampling Estimates

Of course, we are not typically in the situation where we can sample from f. It is shown in [16] that we can generalize this result when using an importance sampling estimate of the quantile. In particular, we now sample $\{x_i\}_{i=1}^N$ from a distribution with density given by q. The importance sampling estimate of α with N i.i.d. samples $\{x_i\}_{i=1}^N$ from q is again given by

$$\alpha_N = F_N^{-1}(p) \stackrel{\text{def}}{=} \inf\{x : F_N(x) \geq p\}$$

but F_N is now the weighted empirical distribution function

$$F_N(x) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(x_i < x) \frac{f(x_i)}{q(x_i)}$$

With this estimate we still have $\sqrt{N}(\alpha_N - \alpha) \xrightarrow{\text{D}} N(0, \sigma^2)$ as $N \to \infty$, but now

$$\sigma^2 = \frac{1}{f(\alpha)^2} \left\{ \int_{-\infty}^{\alpha} \frac{f(x)^2}{q(x)} dx - p^2 \right\}$$

Normalized Importance Sampling Estimates

Of course, the above variance is applicable only for estimate which uses normalized (true) densities for f and q. In SMC methods we generally only have access to unnormalized forms of these densities. In [11] it is shown that asymptotic normality still holds when the estimate uses normalized importance sampling, ie.

$$F_N(x) = \frac{1}{\sum_{i=1}^N \frac{f(x_i)}{q(x_i)}} \sum_{i=1}^{N} \mathbb{I}(x_i < x) \frac{f(x_i)}{q(x_i)}$$

in which case the estimate is unchanged by rescaling f and q by any amount. In this case, $\sqrt{N}(\alpha_N - \alpha) \xrightarrow{\text{D}} N(0, \sigma^2)$ as $N \to \infty$, with

$$\sigma^2 = \frac{1}{f(\alpha)^2} \left\{ \int_{-\infty}^{\infty} \frac{f(x)^2}{q(x)} \left[\mathbb{I}(x \leq \alpha) - p\right]^2 dx \right\} \tag{6.1}$$

6.1.2 The Mean Value Theorem for Integration

The first mean value theorem for integration can be stated as follows:

Given continuous $F : [a, b] \to \mathbb{R}$ and integrable, positive $p : [a, b] \to \mathbb{R}$ there exists a number $c \in [a, b]$ such that

$$\int_a^b F(t)p(t)dt = F(c)\int_a^b p(t)dt$$

In fact, a more general version of this theorem is true for functions on multiple variables, which we will use in our analysis. Given continuous $F : \mathcal{X} \to \mathbb{R}$ where $\mathcal{X}$ is a connected space and integrable, positive $p : \mathcal{X} \to \mathbb{R}$ there exists a vector $\mathbf{c} \in \mathcal{X}$ such that

$$\int_{\mathcal{X}} F(\mathbf{t})p(\mathbf{t})d\mathbf{t} = F(\mathbf{c})\int_{\mathcal{X}} p(\mathbf{t})d\mathbf{t}$$

It should be noted that orthotopes are convex and thus connected.

When the function $p(\mathbf{t})$ is a probability density function which is nonzero only when $\mathbf{t} \in \mathcal{X}$ then $\int_{\mathcal{X}} p(\mathbf{t})d\mathbf{t} = 1$ and so $\int_{\mathcal{X}} F(\mathbf{t})p(\mathbf{t})d\mathbf{t} = F(\mathbf{c})$ for some $\mathbf{c} \in \mathcal{X}$.

6.1.3 Taylor's Theorem

The analysis presented relies heavily on Taylor's theorem, which we state without proof. Let f be a real-valued function on $[a, b]$, n a positive integer, $f^{(n-1)}$ be continuous on $[a, b]$, $f^{(k)}(t)$ exists for all $t \in [a, b]$ and $k \in \{1, \ldots, n\}$. Then for any x and a in $[a, b]$, there exists a c between x and y such that

$$f(x) = \sum_{k=1}^{n-1} \frac{f^{(k)}(y)}{k!}(x - y)^k + \frac{f^{(n)}(c)}{n!}(x - y)^n$$

This is of particular interest when $|f^{(n)}(t)|$ is bounded for all $t \in [a, b]$ as then we can say

$$r(x, y) = \frac{f^{(n)}(c)}{n!}(x - y)^n$$

satisfies

$$\frac{r(x, y)}{||x - y||^{n-1}} \to 0 \text{ as } (x - y) \to 0$$

In our analysis we will use exclusively the case where $n = 2$, which is also called the first-order case.

This gives

$$f(x) = f(y) + f'(y)(x - y) + r(x, y)$$

The multivariate case is also needed here. For $n = 2$ with similar regularity conditions this is

$$f(\mathbf{x}) = f(\mathbf{y}) + \nabla f(\mathbf{y})^T(\mathbf{x} - \mathbf{y}) + r(\mathbf{x}, \mathbf{y})$$

where

$$r(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\frac{\partial^2 f}{\partial x_i \partial x_j}(c_{i,j})(x_i - y_i)(x_j - y_j)$$

with each $c_{i,j}$ falling on the line between $\mathbf{x}$ and $\mathbf{y}$. In particular, if all the second partial derivatives of f in a neighbourhood of $\mathbf{y}$ are bounded it can be shown that $\frac{r(\mathbf{x},\mathbf{y})}{||\mathbf{x}-\mathbf{y}||} \to 0$ as $(\mathbf{x} - \mathbf{y}) \to \mathbf{0}$ so the remainder term is insignificant when $(\mathbf{x} - \mathbf{y}) \to \mathbf{0}$.

6.1.4 Quantile Derivatives

We will refer numerous times to the inverse cumulative distribution function of a marginal distribution in our analysis. Indeed, the function $F_j^{-1}(u|z_{1:j-1})$ is such a function.

Recall that

$$F_j(z|z_{1:j-1}) = \int_{\mathbb{R}^{d-j}}\int_{-\infty}^{z} p(x_j, x_{j+1:d}|x_{1:j-1} = z_{1:j-1})dx_j dx_{j+1:d}$$

and

$$p_j(z|z_{1:j-1}) = \int_{\mathbb{R}^{d-j}} p(x_j = z, x_{j+1:d}|x_{1:j-1} = z_{1:j-1})dx_{j+1:d}$$

so that

$$F_j(z|z_{1:j-1}) = \int_{-\infty}^{z} p_j(x_j|x_{1:j-1} = z_{1:j-1})dx_j$$

We are also limiting our discussion to situations in which $\{\mathbf{x} : p(\mathbf{x}) > 0\}$ is convex. This implies that the inverse cdf $F_j^{-1}(u|z_{1:j-1})$ is unique for $u \in (0, 1)$.

$$F_j^{-1}(u|z_{1:j-1}) = z \text{ s.t. } F_j(z|z_{1:j-1}) = u$$

It is a strictly monotonic function (in u) by its definition and is thus differentiable (in u) almost everywhere with partial derivative

$$\frac{\partial}{\partial u} F_j^{-1}(u|z_{1:j-1}) = \frac{1}{p_j(z|x_{1:j-1} = z_{1:j-1})}$$

where $z = F_j^{-1}(u|z_{1:j-1})$. Furthermore, for fixed u this inverse function is defined implicitly via the equation

$$F_j(z|z_{1:j-1}) - u = 0$$

If we denote this function as $g(x_{1:j-1})$, it satisfies

$$F_j(g(z_{1:j-1})|z_{1:j-1}) - u = 0$$

By the implicit function theorem[19], the partial derivatives of g are given by

$$\begin{aligned}
\frac{\partial g}{\partial x_i}(z_{1:j-1}) &= -\frac{\partial F(z|z_{1:j-1})/\partial x_i}{\partial F(z|z_{1:j-1})/\partial x_j} \\
&= -\frac{\frac{\partial}{\partial x_i}\int_{-\infty}^{z} p_j(x_j|x_{1:j-1} = z_{1:j-1})dx_j}{p_j(z|x_{1:j-1} = z_{1:j-1})} \\
&= -\frac{\int_{-\infty}^{z} \frac{\partial}{\partial x_i} p_j(x_j|x_{1:j-1} = z_{1:j-1})dx_j}{p_j(z|x_{1:j-1} = z_{1:j-1})}
\end{aligned}$$

as long as $p_j(x_j|x_{1:j-1})$ is differentiable for all x_j when $x_{1:j-1} = z_{1:j-1}$ and the partial derivatives $\frac{\partial}{\partial x_i} p_j(x_j|z_{1:j-1})$ are integrable. One can similarly express $\frac{\partial^2 g}{\partial x_i \partial x_k}(z_{1:j-1})$ if p is twice differentiable for all x_j when $x_{1:j-1} = z_{1:j-1}$ and these derivatives are integrable as well.

6.2 An approximation of the generalized cdf

Before moving to an analysis of the unweighted k-ary and binary tree-based algorithms, we first analyse a simpler algorithm that is a more direct (though intractable) approximation to Algorithm 3. This approximation algorithm is described in Algorithm 12. For our analysis, let us define $\boldsymbol{\mu} = F^{-1}(\mathbf{u})$, ie. $\boldsymbol{\mu}$ is the true value of the inverse of the generalized cdf at $\mathbf{u} = u_{1:d}$.

For our proof we will introduce a new set of functions of the form $G_j : \mathbb{R}^j \to \mathbb{R}$ to simplify notation.

[19] Note that $dF = 0 = \sum_{i=0}^{j} \partial_{x_i} F dx_i$ and solve for $\frac{dx_j}{dx_i}$ with all other x_k constant. See [22] for more details.

Algorithm 12 An algorithm to approximate the inverse of the generalized cdf $F^{-1}(u_1, \ldots, u_d)$

1.
   - Sample N_1 points $\{\mathbf{x}^{(i)})\}_{i=1}^{N}$ from proposal distribution $q(\mathbf{x})$.
   - For $i = 1, \ldots, N$, evaluate the importance weights:

$$w_1(\mathbf{x}^{(i)}) = \frac{p(\mathbf{x}^{(i)})}{q(\mathbf{x}^{(i)})}$$

   - Normalize the importance weights

$$W_1^{(i)} = \frac{w_1(\mathbf{x}^{(i)})}{\sum_{k=1}^{N_1} w_1(\mathbf{x}^{(k)})}$$

   - Define $\widehat{x_1} \stackrel{\text{def}}{=} \inf\{z : \hat{F}_1(z) \geq u_1\}$ where

$$\hat{F}_1(z) = \sum_{i=1}^{N_1} W_1^{(i)} I(x_1^{(i)} \leq z)$$

2. For $j = 2, \ldots, d$.
   - Sample N_j points $\{x_{j:d}^{(i)}\}_{i=1}^{N_j}$ from proposal distribution $q(x_{j:d}|\widehat{x_{1:j-1}})$ and set $\mathbf{x}^{(i)} = (\widehat{x_{1:j-1}}, x_{j:d}^{(i)})$.
   - For $i = 1, \ldots, N_j$, evaluate the importance weights:

$$w_j(\mathbf{x}^{(i)}) = \frac{p(\mathbf{x}^{(i)})}{q(\mathbf{x}^{(i)})}$$

   - Normalize the importance weights

$$W_j^{(i)} = \frac{w_j(\mathbf{x}^{(i)})}{\sum_{k=1}^{N_j} w_j(\mathbf{x}^{(k)})}$$

   - Define $\widehat{x_j} \stackrel{\text{def}}{=} \inf\{z : \hat{F}_j(z|\widehat{x_{1:j-1}}) \geq u_j\}$ where

$$\hat{F}_j(z; \widehat{x_{1:j-1}}) = \sum_{i=1}^{N_j} W_j^{(i)} I(x_j^{(i)} \leq z)$$

   and set $\widehat{x_{1:j}} = (\widehat{x_{1:j-1}}, \widehat{x_j})$

3. Output $\widehat{x_{1:d}}$.

In particular, we define $G_j(u_j; x_{1:j-1})$ as follows:

$$\begin{aligned} G_j(u_j; x_{1:j-1}) &= \inf\{x_j : F_j(x_j|x_{1:j-1}) \geq u_j\} \\ &= x_j : F_j(x_j|x_{1:j-1}) = u_j \\ &= F_j^{-1}(u_j|x_{1:j-1}) \end{aligned}$$

Claim: Let $N_j = c_j N$ for $j \in \{1, \ldots, d\}$ and $\mathbf{u} \in [0,1]^d$ be given. Define $\boldsymbol{\mu} = F^{-1}(\mathbf{u})$. Assume $\frac{\partial^2 G_j(u; x_{1:j-1})}{\partial x_i \partial x_k}$ is bounded for all $i, k < j$ in a neighbourhood of $\mu_{1:j-1}$ and all $j \in \{1, \ldots, d\}$. Also, assume the set $\{\mathbf{x} : p(\mathbf{x}) > 0\}$ is convex. Then as $N \to \infty$, $\sqrt{N}(\hat{\mathbf{x}} - \boldsymbol{\mu}) \xrightarrow{\text{D}} N(\mathbf{0}, \Sigma)$ for some Σ independent of N.

Proof: We know $\widehat{x_1}$ is asymptotically distributed as $N(\mu_1, \frac{\sigma_1^2}{N_1})$ where σ_1 is of the form given in Equation 6.1. Let $v_1 = \widehat{x_1} - \mu_1 \xrightarrow{\text{D}} N(0, \frac{\sigma_1^2}{N_1})$. We also know $\widehat{x_2}$ is distributed as $N(\widehat{\mu_2}, \frac{\sigma_2^2}{N_2})$, where $\widehat{\mu_2} \stackrel{\text{def}}{=} G_2(u_2; \widehat{x_1})$ and σ_2 is of the form given in Equation 6.1.

A Taylor expansion of G_2 around μ_1 gives

$$\begin{aligned} \widehat{\mu_2} = G_2(u_2; \widehat{x_1}) &= G_2(u_2; \mu_1) + \frac{\partial G_j(u_2; \mu_1)}{\partial x_1}(\widehat{x_1} - \mu_1) + r_2(\widehat{x_1}, \mu_1) \\ &\approx \mu_2 + \frac{\partial G_j(u_2; \mu_1)}{\partial x_1} v_1 \end{aligned}$$

If we let $\widehat{x_2} - \widehat{\mu_2} = v_2 \xrightarrow{\text{D}} N(0, \frac{\sigma_2^2}{N_2})$, we have asymptotically

$$\widehat{x_2} = \mu_2 + \frac{\partial G_2(u_2; \mu_1)}{\partial x_1} v_1 + v_2$$

so

$$\begin{bmatrix} \widehat{x_1} \\ \widehat{x_2} \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ \frac{\partial G_2(u_2; \mu_1)}{\partial x_1} & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

For $j = 3$ we have $\widehat{x_3}$ is distributed as $N(\widehat{\mu_3}, \frac{\sigma_3^2}{N_3})$, where $\widehat{\mu_3} \stackrel{\text{def}}{=} G_3(u_3; \widehat{x_{1:2}})$ and σ_3 is of the form given in Equation 6.1.

A Taylor expansion of G_3 around $\mu_{1:2}$ gives

$$\begin{aligned}\widehat{\mu_3} = G_3(u_3; \widehat{x_{1:2}}) &= G_3(u_3; \mu_{1:2}) + \frac{\partial G_3(u_3;\mu_{1:2})}{\partial x_1}(\widehat{x_1} - \mu_1) + \frac{\partial G_3(u_3;\mu_{1:2})}{\partial x_2}(\widehat{x_2} - \mu_2) + r_3(\widehat{x_{1:2}}, \mu_{1:2}) \\ &\approx \mu_3 + \frac{\partial G_3(u_3;\mu_{1:2})}{\partial x_1} v_1 + \frac{\partial G_3(u_3;\mu_{1:2})}{\partial x_2}\left[\frac{\partial G_2(u_2;\mu_1)}{\partial x_1} v_1 + v_2\right]\end{aligned}$$

If we let $\widehat{x_3} - \widehat{\mu_3} = v_3 \xrightarrow{\text{D}} N(0, \frac{\sigma_3^2}{N_3})$, we have asymptotically

$$\widehat{x_3} = \mu_3 + \left[\frac{\partial G_3(u_3;\mu_{1:2})}{\partial x_1} + \frac{\partial G_3(u_3;\mu_{1:2})}{\partial x_2}\frac{\partial G_2(u_2;\mu_1)}{\partial x_1}\right] v_1 + \frac{\partial G_3(u_3;\mu_{1:2})}{\partial x_2} v_2 + v_3$$

so

$$\begin{bmatrix} \widehat{x_1} \\ \widehat{x_2} \\ \widehat{x_3} \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ \frac{\partial G_2(u_2;\mu_1)}{\partial x_1} & 1 & 0 \\ \frac{\partial G_3(u_3;\mu_{1:2})}{\partial x_1} + \frac{\partial G_3(u_3;\mu_{1:2})}{\partial x_2}\frac{\partial G_2(u_2;\mu_1)}{\partial x_1} & \frac{\partial G_3(u_3;\mu_2)}{\partial x_2} & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

In general, we have $\widehat{x_j}$ is distributed as $N(\widehat{\mu_j}, \frac{\sigma_j^2}{N_j})$, where $\widehat{\mu_j} \stackrel{\text{def}}{=} G_j(u_j; \widehat{x_{1:j-1}})$ and σ_j is of the form given in Equation 6.1.

A Taylor expansion of G_j around $\mu_{1:j-1}$ gives

$$\begin{aligned}\widehat{\mu_j} = G_j(u_j; \widehat{x_{1:j-1}}) &= G_j(u_j; \mu_{1:j-1}) + \sum_{i=1}^{j-1} \frac{\partial G_j(u_j;\mu_{1:j-1})}{\partial x_i}(\widehat{x_i} - \mu_i) + r_j(\widehat{x_{1:j-1}}, \mu_{1:j-1}) \\ &\approx \mu_j + \sum_{i=1}^{j-1} \frac{\partial G_j(u_j;\mu_{1:j-1})}{\partial x_i}(\widehat{x_i} - \mu_i)\end{aligned}$$

Note that the terms $\widehat{x_i} - \mu_i$ get progressively more complex in terms of $v_{1:i}$ as i increases. For example, $\widehat{x_4} - \mu_4$ has the form

$$\begin{aligned}\widehat{x_4} - \mu_4 = &\left[\frac{\partial G_4}{\partial x_1} + \frac{\partial G_4}{\partial x_2}\frac{\partial G_2}{\partial x_1} + \frac{\partial G_4}{\partial x_3}\frac{\partial G_3}{\partial x_1} + \frac{\partial G_4}{\partial x_3}\frac{\partial G_3}{\partial x_2}\frac{\partial G_2}{\partial x_1}\right] v_1 \\ &+ \left[\frac{\partial G_4}{\partial x_2} + \frac{\partial G_4}{\partial x_3}\frac{\partial G_3}{\partial x_2}\right] v_2 + \frac{\partial G_4}{\partial x_3} v_3 + v_4\end{aligned}$$

Nevertheless, each $\widehat{x_i} - \mu_i$ can be expressed as a linear combination of v_j's (with $j \leq i$) since every partial derivative of G_k is evaluated at $(u_k; \mu_{1:k-1})$. If we let $\lambda_{i,j}$ denote the coefficient of v_j in the expression $\widehat{x_i} - \mu_i = \sum_{j=1}^{i} \lambda_{i,j} v_j$ then we can write

$$\widehat{\mathbf{x}} = \boldsymbol{\mu} + \Lambda \mathbf{v}$$

with the matrix Λ defined by $\Lambda(i,j) = \lambda_{i,j}$.

Note that each $v_j \sim N(0, \frac{\sigma_j^2}{N_j}) = N(0, \frac{\sigma_j^2}{c_j N})$ As such, we can write

$$\widehat{\mathbf{x}} = \boldsymbol{\mu} + \frac{1}{\sqrt{N}} A\mathbf{z}$$

where the matrix A is defined by $A(i,j) = \frac{\sqrt{c_j}}{\sigma_j}\lambda_{i,j}$ and $\mathbf{z} \sim N(\mathbf{0}, I)$. In other words, as $N \to \infty$, $\sqrt{N}(\widehat{\mathbf{x}} - \boldsymbol{\mu}) \xrightarrow{\text{D}} N(\mathbf{0}, AA^T)$ where $\Sigma \stackrel{\text{def}}{=} AA^T$ is independent of N $\square$.

6.3 The Unweighted k-ary Tree

Having proven asymptotic normality of the sampled points using direct approximation to Algorithm 3, we turn to the k-ary tree sampling scheme, detailed in Section 4.4. This algorithm draws its efficiency gains from the use of partitions as opposed to sampling many vectors with identical components. The drawback is that it is more complex to analyze.

For our proof we will introduce a new set of functions of the form $H_j : \mathbb{R}^{2j-1} \to \mathbb{R}$ to simplify notation. In particular, we define $H_j(u; x_{1:j-1})$ as follows:

$$\begin{aligned} H_j(u; \underline{x_{1:j-1}}, \overline{x_{1:j-1}}) &= \inf\{x_j : \int_{-\infty}^{x_j} p_j\left(t \middle| x_{1:j-1} \in \prod_{i=1}^{j-1}[\underline{x_i}, \overline{x_i}]\right) dt \geq u\} \\ &= x_j : F_j(x_j | x_{1:j-1} \in \prod_{i=1}^{j-1}[\underline{x_i}, \overline{x_i}]) = u \\ &= F_j^{-1}(u | x_{1:j-1} \in \prod_{i=1}^{j-1}[\underline{x_i}, \overline{x_i}]) \end{aligned}$$

where

$$p_j\left(x_j \middle| x_{1:j-1} \in \prod_{i=1}^{j-1}[\underline{x_i}, \overline{x_i}]\right) = \frac{\int_{\prod_{i=1}^{j-1}[\underline{x_i}, \overline{x_i}]} p(x_{1:j}) dx_{1:j-1}}{\int_{\prod_{i=1}^{j-1}[\underline{x_i}, \overline{x_i}]} p(x_{1:j-1}) dx_{1:j-1}}$$

Note that we can use the mean value theorem to show that $H_j(u; \underline{x_{1:j-1}}, \overline{x_{1:j-1}}) = F_j^{-1}(u | c_{1:j-1})$

for some $c_{1:j-1} \in I = \prod_{i=1}^{j-1}[\underline{x_i}, \overline{x_i}]$:

$$\begin{aligned}
&\int_{-\infty}^{x_j} p_j\left(t|x_{1:j-1} \in I\right) dt = u \\
\Rightarrow &\frac{\int_{-\infty}^{x_j}\int_I p(x_{1:j-1},t)dx_{1:j-1}dt}{\int_I p(x_{1:j-1})dx_{1:j-1}} = u \\
\Rightarrow &\frac{\int_{-\infty}^{x_j}\int_I p(t|x_{1:j-1})p(x_{1:j-1})dx_{1:j-1}dt}{\int_I p(x_{1:j-1})dx_{1:j-1}} = u \\
\Rightarrow &\frac{\int_{-\infty}^{x_j} p(t|c_{1:j-1})dt\int_I p(x_{1:j-1})dx_{1:j-1}}{\int_I p(x_{1:j-1})dx_{1:j-1}} = u \\
\Rightarrow &\int_{-\infty}^{x_j} p(t|c_{1:j-1})dt = u \\
\Rightarrow &H_j(u;\underline{x_{1:j-1}},\overline{x_{1:j-1}}) = F_j^{-1}(u|c_{1:j-1})
\end{aligned}$$

Recall that $G_j(u;c_{1:j-1}) = F_j^{-1}(u|c_{1:j-1})$ so we have $H_j(u;\underline{x_{1:j-1}},\overline{x_{1:j-1}}) = G_j(u;c_{1:j-1})$.

Claim: Let $N = k^d$ and let $\mathbf{u} \in [0,1]^d$ be given. Define $\boldsymbol{\mu} = F^{-1}(\mathbf{u})$. Assume $\frac{\partial^2 G_j(u;x_{1:j-1})}{\partial x_i \partial x_k}$ is bounded for all $i,k < j$ in a neighbourhood of $\mu_{1:j-1}$ and all $j \in \{1,\ldots,d\}$. Also, assume the set $\{\mathbf{x} : p(\mathbf{x}) > 0\}$ is convex. Then as $k \to \infty$, $\hat{\mathbf{x}} \xrightarrow{\text{P}} \boldsymbol{\mu}$.

Proof: Let $\underline{u_j} = \frac{\lceil ku_j\rceil - 1}{k}$ and $\overline{u_j} = \frac{\lceil ku_j\rceil}{k}$. We denote the number of particles at each stage of the algorithm as $N_1,\ldots,N_d$. Note first that $N_1,\ldots,N_d$ are all related. We will prove later that each of these values increases as $k \to \infty$.

In Algorithm 11, the first subset that is chosen is the $\lceil ku_1\rceil$th subset on the second level. Let us define $\underline{x_1}$ and $\overline{x_1}$ as the left and right endpoints of the selected subset. Since $\underline{u_1}$ converges (this is not a random variable) to u_1, sample quantiles are asymptotically unbiased and their asymptotic variance decreases linearly in N, we know that $\underline{x_1} \xrightarrow{\text{P}} \mu_1$ (one can pick N large enough such that the mean of the estimate is close enough to μ_1 and the variance is small enough for a Chebyshev inequality to show convergence). Similarly, $\overline{x_1} \xrightarrow{\text{P}} \mu_1$. It doesn't matter that $\underline{x_1}$ and $\overline{x_1}$ are dependent since two variables that converge in probability marginally must also converge in probability jointly.

Now define $c_1^{(2)}$ such that is satisfies $G_2(u_2;c_1^{(2)}) = H_2(u_2;\underline{x_1},\overline{x_1})$. Define $\widehat{\mu_2} = G_2(u_2;c_1^{(2)})$. By the same reasoning as above, the left and right endpoints of the second partition $\underline{x_2}$ and $\overline{x_2}$ satisfy $\underline{x_2} \xrightarrow{\text{P}} \widehat{\mu_2}$ and $\overline{x_2} \xrightarrow{\text{P}} \widehat{\mu_2}$ (as long as $N_2 \to \infty$ as $k \to \infty$) where

$$\widehat{\mu_2} = \mu_2 + \frac{\partial G_2(u_2;\mu_1)}{\partial x_1}(c_1^{(2)} - \mu_1) + r_2(c_1^{(2)},\mu_1)$$

Since $c_1^{(2)} \in [\underline{x_1},\overline{x_1}]$ with $\underline{x_1} \xrightarrow{\text{P}} \mu_1$ and $\overline{x_1} \xrightarrow{\text{P}} \mu_1$, $c_1^{(2)} \xrightarrow{\text{P}} \mu_1$. Therefore $\underline{x_2} \xrightarrow{\text{P}} \mu_2$ and $\overline{x_2} \xrightarrow{\text{P}} \mu_2$.

In general we have $c^{(j)}_{1:j-1} \in \prod_{i=1}^{j-1}[\underline{x_i}, \overline{x_i}]$ satisfying $G_j(u_j; c^{(j)}_{1:j-1}) = H_j(u_j; \underline{x_{1:j-1}}, \overline{x_{1:j-1}})$ and

$$\widehat{\mu_j} \approx \mu_j + \sum_{i=1}^{j-1} \frac{\partial G_j(u_j; \mu_{1:j-1})}{\partial x_i}(c_i^{(j)} - \mu_i)$$

where $c^{(j)}_{1:j-1} \xrightarrow{\text{P}} \mu_{1:j-1}$ so $\underline{x_j} \xrightarrow{\text{P}} \mu_j$ and $\overline{x_j} \xrightarrow{\text{P}} \mu_j$ as long as $N_j \to \infty$ as $k \to \infty$. Thus we can be assured that $\widehat{x_{1:d-1}} \xrightarrow{\text{P}} \mu_{1:d-1}$ for any $\widehat{x_{1:d-1}} \in \prod_{i=1}^{d-1}[\underline{x_i}, \overline{x_i}]$.

We do not compute $\underline{x_d}$ and $\overline{x_d}$. Instead, we set $\widehat{x_d}$ to be the sample conditional u_d-quantile of the remaining particles in the dth dimension. We have some $c^{(d)}_{1:d-1} \in \prod_{i=1}^{d-1}[\underline{x_i}, \overline{x_i}]$ satisfying $G_d(u_d; c^{(d)}_{1:d-1}) = H_d(u_d; \underline{x_{1:d-1}}, \overline{x_{1:d-1}})$, with $c^{(j)}_{1:j-1} \xrightarrow{\text{P}} \mu_{1:d-1}$. Let $\widehat{\mu_d} = G_d(u_d; c^{(d)}_{1:d-1})$. Then

$$\sqrt{N_d}(\widehat{x_d} - \widehat{\mu_d}) \xrightarrow{\text{D}} N(0, \sigma_d^2)$$

with

$$\widehat{\mu_d} \approx \mu_d + \sum_{i=1}^{d-1} \frac{\partial G_d(u_d; \mu_{1:j-1})}{\partial x_i}(c_i^{(d)} - \mu_i)$$

and σ_d is of the form given in Equation 6.1. Therefore $\widehat{\mu_d} \xrightarrow{\text{P}} \mu_d$ and so $\widehat{x_d} \xrightarrow{\text{P}} \mu_d$.

All that remains to show is that as $k \to \infty$, $N_j \to \infty$ for $j \in \{1, \ldots, d\}$. Let us denote N_j as a function of k by $N_j(k)$. Recall that $q(\mathbf{x})$ is the proposal distribution used to sample the particles $\{\mathbf{x}^{(i)}\}$.

Note first that $N(k) = k^d$ and $N_1(k) = N(k)$. $N_2(k) \to N_1(k) \int_{F_1^{-1}(\underline{u_1})}^{F_1^{-1}(\overline{u_1})} q(x_1)dx_1$. We have that

$$F_1^{-1}(\underline{u_1}) = F_1^{-1}(u_1) + \frac{\partial F_1^{-1}(u_1)}{\partial u}(\underline{u_1} - u_1) + r_1(\underline{u_1}, u_1)$$

and

$$F_1^{-1}(\overline{u_1}) = F_1^{-1}(u_1) + \frac{\partial F_1^{-1}(u_1)}{\partial u}(\overline{u_1} - u_1) + r_1(\overline{u_1}, u_1)$$

which implies

$$\begin{aligned} F_1^{-1}(\overline{u_1}) - F_1^{-1}(\underline{u_1}) &\approx \frac{\partial F_1^{-1}(u_1)}{\partial u}(\overline{u_1} - \underline{u_1}) \\ &= \frac{1}{k}\frac{\partial F_1^{-1}(u_1)}{\partial u} = \frac{1}{kp(\mu_1)} \end{aligned}$$

Therefore, as $k \to \infty$ we have $\int_{F_1^{-1}(\underline{u_1})}^{F_1^{-1}(\overline{u_1})} q(x_1)dx_1 = \frac{1}{k}\frac{\partial F_1^{-1}(u_1)}{\partial u}q(\mu_1)$ and so

$$N_2(k) = N_1(k)\frac{q(\mu_1)}{kp(\mu_1)}$$

In general, we will have $N_j(k) \to N_{j-1}(k)\int_{F_j^{-1}(\underline{u_j}|\mu_{1:j-1})}^{F_j^{-1}(\overline{u_1}|\mu_{1:j-1})} q(x_j|x_{1:j-1} = \mu_{1:j-1})dx_j$, which yields

$$N_j(k) = N_{j-1}(k)\frac{1}{k}\frac{\partial F_j^{-1}(u_j|\mu_{1:j-1})}{\partial u}q(\mu_j|\mu_{1:j-1}) = N_{j-1}(k)\frac{q(\mu_j|\mu_{1:j-1})}{kp(\mu_j|\mu_{1:j-1})}$$

or

$$N_j(k) = N_1(k)\prod_{i=1}^{j-1}\frac{q(\mu_i|\mu_{1:i-1})}{kp(\mu_i|\mu_{1:i-1})} = \frac{N_1(k)}{k^{j-1}}\prod_{i=1}^{j-1}\frac{q(\mu_i|\mu_{1:i-1})}{p(\mu_i|\mu_{1:i-1})}$$

Since $N_1(k) = N(k) = k^d$

$$\begin{aligned} N_j(k) &= k^{d-j+1}\prod_{i=1}^{j-1}\frac{q(\mu_i|\mu_{1:i-1})}{p(\mu_i|\mu_{1:i-1})} \\ &\propto k^{d-j+1} \end{aligned}$$

ie. $N_j \to \infty$ as $k \to \infty$ for all $j \in \{1, \ldots, d\}\square$.

6.4 The Unweighted Binary Tree

The resampling scheme using an unweighted binary tree described in Algorithm 8 is a fairly direct approximation of Algorithm 4. To analyze it, we introduce a set of functions $Q_j : \mathbb{R}^j \to \mathbb{R}$. For any j, let $k = j \mod d$ (and if this makes $k = 0$, let $k = d$). Then

$$Q_j(z_{1:j-1}, u) = z_k : 2^{j-1}\int_{a_1^{(j)}}^{b_1^{(j)}} \cdots \int_{a_k^{(j)}-1}^{b_k^{(j)}-1}\int_{a_k^{(j)}}^{z_k}\int_{a_k^{(j)}+1}^{b_k^{(j)}+1} \cdots \int_{a_d^{(j)}}^{b_d^{(j)}} p(\mathbf{x})d\mathbf{x} = 0.5$$

where the value of u determines the value of each $a_i^{(j)}$ and $b_i^{(j)}$ given $z_{1:j-1}$. In particular, as a representation of an infinite number of bits, u determines for each (i, j) whether $a_i^{(j)}$ is equal to $-\infty$ or z_{i+rd} for some $r \in \mathbb{N}$. Similarly, u determines if $b_i^{(j)}$ is equal to ∞ or z_{i+rd} for some $r \in \mathbb{N}$.

In order to analyze the selection scheme, we pretend that the medians $\widehat{t_1}, \ldots, \widehat{t_n}$ are computed only as needed by Algorithm 9. Since the tree is unweighted, it is not the case that $N_j = 2^{k-j+1}$. Let

$I = \prod_{i=1}^{d}[a_i^{(j)}, b_i^{(j)}]$ be the region determined by $(t_{1:j-1}, u)$. Then

$$E[N_j(k)] = N(k)\int_I q(\mathbf{x})d\mathbf{x}$$

and furthermore $N_j(k)$ converges in probability to this value. As such, N_j converges in probability to a linear function of N.

Claim: Let $u \in [0,1]$ and $m \in \mathbb{N}$ be given and let $t_{1:m} = f_m(u)$. Assume $\frac{\partial^2 Q_j(z_{1:j-1},u)}{\partial z_i \partial z_k}$ is bounded for all $i, k < j$ in a neighbourhood of $t_{1:j-1}$ and all $j \in \{1,\ldots,m\}$. Also, assume the set $\{\mathbf{x} : p(\mathbf{x}) > 0\}$ is convex. Then as $N \to \infty$, $\sqrt{N}(\widehat{t_{1:m}} - t_{1:m}) \xrightarrow{\text{D}} N(\mathbf{0}, \Sigma)$ for some Σ independent of N.

Proof: We have that

$$\widehat{t_1} \xrightarrow{\text{D}} N(t_1, \frac{\sigma_1^2}{N_1})$$

where σ_1 is of the form given in Equation 6.1. Let $v_1 = \widehat{t_1} - t_1 \xrightarrow{\text{D}} N(0, \frac{\sigma_1^2}{N_1})$. We have also that $\widehat{t_2} \xrightarrow{\text{D}} N(\overline{t_2}, \frac{\sigma_2^2}{N_2})$, where $\overline{t_2} \stackrel{\text{def}}{=} Q_2(\widehat{t_1}, u)$ and σ_2 is of the form given in Equation 6.1.

A Taylor expansion of Q_2 around t_1 is given by

$$\begin{aligned}\overline{t_2} = Q_2(\widehat{t_1}, u) &= Q_2(t_1, u) + \frac{\partial Q_2(t_1,u)}{\partial z_1}(\widehat{t_1} - t_1) + r_2(\widehat{t_1}, t_1, u) \\ &\approx t_2 + \frac{\partial Q_2(t_1,u)}{\partial z_1} v_1\end{aligned}$$

If we let $\widehat{t_2} - \overline{t_2} = v_2 \xrightarrow{\text{D}} N(0, \frac{\sigma_2^2}{N_2})$, we have asymptotically

$$\widehat{t_2} = t_2 + \frac{\partial Q_2(t_1,u)}{\partial z_1} v_1 + v_2$$

In general, we have $\widehat{t_j}$ is distributed as $N(\overline{t_j}, \frac{\sigma_j^2}{N_j})$, where $\overline{t_j} \stackrel{\text{def}}{=} Q_j(t_{1:j-1}, u)$ and σ_j is of the form given in Equation 6.1.

A Taylor expansion of Q_j around $t_{1:j-1}$ gives

$$\begin{aligned}Q_j(\widehat{t_{1:j-1}}, u) &= Q_j(t_{1:j-1}, u) + \sum_{i=1}^{j-1}\left[\frac{\partial Q_j(t_{1:j-1},u)}{\partial z_i}\right](\widehat{t_i} - t_i) + r_j(\widehat{t_{1:j-1}}, t_{1:j-1}, u) \\ &\approx t_j + \sum_{i=1}^{j-1}\left[\frac{\partial Q_j(t_{1:j-1},u)}{\partial z_i}\right](\widehat{t_i} - t_i)\end{aligned}$$

It should be obvious that at most $2d$ of the partial derivatives are nonzero since Q_j does not depend

on every z_i.

Similar to the analysis for the approximation to the generalized cdf in Section 6.2, the terms $\widehat{t_i} - t_i$ get progressively more complex as i increases. Again, however, each $\widehat{t_i} - t_i$ can be expressed as a linear combination of v_j's (with $j \leq i$). If we let $\lambda_{i,j}$ denote the coefficient of v_j in the expression $\widehat{t_i} - t_i = \sum_{j=1}^{i} \lambda_{i,j} v_j$ then we can write

$$\widehat{t_{1:m}} = t_{1:m} + \Lambda_m v_{1:m}$$

with the $m \times m$ matrix Λ_m defined by $\Lambda_m(i,j) = \lambda_{i,j}$.

Let $I_j = \prod_{i=1}^{d} [a_i^{(j)}, b_i^{(j)}]$ be the region determined by $(t_{1:j-1}, u)$ for $j = 1, \ldots, m$. Then set $c_j = \int_{I_j} q(\mathbf{x}) d\mathbf{x}$ so that $N_j \xrightarrow{\text{P}} c_j N$. Note that each $v_j \sim N(0, \frac{\sigma_j^2}{N_j}) = N(0, \frac{\sigma_j^2}{c_j N})$ As such, we can write

$$\widehat{t_{1:m}} = t_{1:m} + \frac{1}{\sqrt{N}} A \rho_{1:m}$$

where the matrix A is defined by $A(i,j) = \frac{\sqrt{c_j}}{\sigma_j} \lambda_{i,j}$ and $\rho_{1:m} \sim N(\mathbf{0}, I)$. In other words, as $N \to \infty$, $\sqrt{N}(\widehat{t_{1:m}} - t_{1:m}) \xrightarrow{\text{D}} N(\mathbf{0}, AA^T)$ where $\Sigma \stackrel{\text{def}}{=} AA^T$ is independent of N $\square$.

Let us define $\boldsymbol{\mu}$ to be (almost surely) the unique point in $\lim_{n\to\infty} f_n(u)$, where f is specified in Algorithm 4. Since the above result shows that for any m, $\sqrt{N}(\widehat{t_{1:m}} - t_{1:m}) \xrightarrow{\text{D}} N(\mathbf{0}, \Sigma)$ for some Σ independent of N, it follows that $\widehat{\mathbf{x}} \xrightarrow{\text{P}} \boldsymbol{\mu}$.

6.5 Changing Parameters

The above analyses show that the resampling mechanisms provide consistent sampled particles given the same inputs, at least marginally. As $N \to \infty$, it is expected that the correlation between sampled particles due to the use of the same N particles each time a particle is selected is negligible. To show that each sampled particle is close to its corresponding particle when the parameters change is straightforward.

If $\widehat{\mathbf{x}} \xrightarrow{\text{P}} \boldsymbol{\mu}$ using target distribution $p(\cdot|\theta)$ and $\widehat{\mathbf{x}}' \xrightarrow{\text{P}} \boldsymbol{\mu}'$ using target distribution $p(\cdot|\theta')$, then

$$\begin{aligned} \widehat{\mathbf{x}}' &= \widehat{\mathbf{x}}' - \boldsymbol{\mu}' + \boldsymbol{\mu}' - \boldsymbol{\mu} + \boldsymbol{\mu} - \widehat{\mathbf{x}} + \widehat{\mathbf{x}} \\ \Rightarrow \widehat{\mathbf{x}}' &\xrightarrow{\text{P}} \widehat{\mathbf{x}} + \boldsymbol{\mu}' - \boldsymbol{\mu} \end{aligned}$$

If we have asymptotic normality, ie. $\sqrt{N}(\boldsymbol{\mu} - \widehat{\mathbf{x}}) \xrightarrow{\mathrm{D}} N(\mathbf{0}, \Sigma_1)$ and $\sqrt{N}(\widehat{\mathbf{x}}' - \boldsymbol{\mu}') \xrightarrow{\mathrm{D}} N(\mathbf{0}, \Sigma_2)$ we have

$$\sqrt{N}(\widehat{\mathbf{x}}' - \widehat{\mathbf{x}}) \xrightarrow{\mathrm{D}} N(\boldsymbol{\mu}' - \boldsymbol{\mu}, \Sigma_1 + \Sigma_2)$$

In the event that $\boldsymbol{\mu}$ is continuous in θ, this implies that $\widehat{\mathbf{x}}'$ will be close to $\widehat{\mathbf{x}}$ if θ' is close to θ.

6.6 Remarks

In the above analysis, we have shown asymptotic convergence for selected particles from a distribution $p(\mathbf{x}) = p(\mathbf{x}_t|\mathbf{y}_{0:t})$ whereas we are interested in convergence for selected particles from a distribution $\hat{P}_N(d\mathbf{x}_t|\mathbf{y}_{0:t}) = \sum_{i=1}^N w_t^{(i)} \delta_{\mathbf{x}_t^{(i)}}(d\mathbf{x}_t)$. However, it is known that $\widehat{P_N}(\mathbf{x}_t|\mathbf{y}_{0:t})$ converges in distribution to $p(\mathbf{x}_t|\mathbf{y}_{0:t})$ as $N \to \infty$ so this is not an issue.

It is worth noting that there are no regularity conditions required to guarantee that any of the tree-based resampling algorithms select a particle (without interpolation) $\mathbf{x}^{(i)}$ according to its weight $W^{(i)}$, as shown in Chapter 4. The conditions used above are only needed to show that the selected particle using common random numbers converges asymptotically to a particular particle.

Chapter 7

Discussion

This thesis has presented the current state of the art for smooth likelihood estimation using particle filters, discussing recent approaches in the literature as well as proposing a novel tree-based approach. In particular, we have seen that in continuous state-space models, it is possible to evaluate the log-likelihood of data given fixed parameters using particle filters. In addition, we can reduce the variance of differences in this log-likelihood by inducing correlation among the particles selected by each filter through the use of common random numbers.

The main contribution of this thesis is the introduction of tree-based resampling schemes to induce correlation amongst selected particles across filters when the state variables are multi-dimensional. This is accomplished with no additional constraints on the general filtering framework - any proposal distribution can be used and there are no additional parameters. In fact, the use of a tree-based resampling scheme involves only a change to the *resample* operation in the SMC framework. In addition, the running time of a filter utilising such a resampling scheme is in $o(TN^2)$, where N is the number of particles and T is the number of discrete time steps in the filter. The most empirically compelling scheme, that involving the weighted binary tree, leads to a filter that runs in expected $O(TN \log N)$ time. This is in contrast to other approaches which run in $O(TN^2)$ time and require a family of approximating distributions (see Section 3.3.2) or approaches which generally require small T and close θ (see Section 3.3.3). Furthermore, the tree-based resampling scheme can be seen as a generalization of the 'solved' case (see [23]) where the state variables are univariate.

In the analysis of the unweighted binary and k-ary trees that are proposed, we have shown that as $N \to \infty$, the particle sampled using a common random number converges in probability to a point that would be returned by a theoretical procedure involving the computation of numerous intractable integrals. While this assures us of a high degree of correlation for very large N, it should not be confused with a result for practical values of N. In our applications, it is found that the weighted binary tree resampling scheme (for which no asymptotic results are shown) gives the smoothest log-likelihood function. In addition, the variance of a particle filter estimate of the log-likelihood is proportional to $\frac{1}{N}$. Since we are concerned with situations where the true log-likelihood is continuous in the parameters, runs of uncorrelated particle filters as $N \to \infty$ will themselves become nearly smooth. Furthermore, uncorrelated filters run in $O(N)$ time and will

always provide lower variance estimates of the log-likelihood given a fixed computational budget. Alternatively, even the SIS algorithm which is naturally smooth using common random numbers will provide accurate estimation of the log-likelihood as $N \to \infty$ since the variance, despite being exponential in the fixed time index T, decreases as $\frac{1}{N}$. This obviously does not mean that our methods are not useful - they just highlight the potential risk of using asymptotic properties to justify the use of a particular resampling scheme.

In practical applications, the use of the tree-based resampling schemes show a distinct improvement over the vanilla and CRN vanilla filters with respect to smoothness of plots of the log-likelihood for changing parameters. In order to benefit from this smoothness, one could employ curve fitting or surface fitting techniques to obtain approximate maximum likelihood estimates of the parameters. In an online setting, one could assign one filter to each of M parameter values and run each filter for one step upon receipt of new data point. This would take time in $O(MN \log N)$ and could be easily parallelized to $O(N \log N)$ on M computing machines. If the surface fitting procedure is not computationally intensive given the M estimates of the log-likelihood, a smooth approximating surface could additionally be computed at each time step. In contrast, methods like those in [5] are not able to produce an accurate and almost smooth surface for large ranges of parameters but are better suited to maximum-likelihood parameter estimation via stochastic gradient ascent for fixed T.

Future Research

The tree-based resampling schemes we have presented are not necessarily the most effective. It is possible that different trees could have better properties. In addition, the methods we have proposed select indices by processing the particles without any kind of modification. In some cases, dimension reduction or some transformation of the particles could be used to build the tree and select indices such that the smoothness of the likelihood estimator is increased.

There is clearly work to be done in analyzing the correlation between particles in tree-based resampling schemes for finite N. If we could estimate the amount of correlation given a particular tree structure or set of structures, we would have better grounds for selecting the type of tree used for resampling.

Finally, the most appropriate way to utilize nearly smooth particle filters is unknown at present. They seem well-suited to the task of likelihood maximization but not necessarily via stochastic gradient ascent, for which the methods in [25] and [5] are readily applicable. Curve or surface fitting as a precursor to maximization as mentioned above is only one possible option.

Bibliography

[1] Jon Louis Bentley and James B. Saxe. Generating sorted lists of random numbers. *ACM Trans. Math. Softw.*, 6(3):359–364, 1980.

[2] M. Blum, R.W. Floyd, V. Pratt, R. Rivest, and R. Tarjan. Time bounds for selection. *J. Comput. System Sci. 7*, pages 448–461, 1973.

[3] P. Bortot, S. G. Coles, and S. A. Sisson. Inference for stereological extremes. *Journal of the American Statistical Association*, 102:84–92, 2007.

[4] R. J. Boys, D. J. Wilkinson, and T. B. L. Kirkwood. Bayesian inference for a discretely observed stochastic kinetic model. *Statistics and Computing*, 2007.

[5] Pierre-Arnaud Coquelin, Romain Deguest, and Remi Munos. Perturbation analysis for parameter estimation in continuous space HMMs. *Submitted to IEEE Transactions on Signal Processing*, 2008.

[6] D. Crisan and A. Doucet. A survey of convergence results on particle filtering methods for practitioners. *IEEE Transactions on Signal Processing*, 50(3):736–746, 2002.

[7] David N. Dejong, Hariharan Dharmarajan, Roman Liesenfeld, and Jean-Francois Richard. An efficient approach to analyzing state-space representations. *SSRN eLibrary*, 2007.

[8] Arnaud Doucet. On sequential simulation-based methods for Bayesian filtering. Technical Report CUED/F-INFENG/TR. 310, Cambridge University, Department of Engineering, 1998.

[9] Garland B Durham and A Ronald Gallant. Numerical techniques for maximum likelihood estimation of continuous-time diffusion processes. *Journal of Business & Economic Statistics*, 20(3):297–316, 2002.

[10] Paul Fearnhead. Computational methods for complex stochastic systems: a review of some alternatives to MCMC. *Statistics and Computing*, 2007.

[11] P. Glynn. Importance sampling for Monte Carlo estimation of quantiles. Technical report, Stanford University, Department of Operations Research, 1996.

[12] A. Golightly and D. J. Wilkinson. Bayesian inference for stochastic kinetic models using a diffusion approximation. *Biometrics*, 61:781–788, 2005.

[13] A. Golightly and D. J. Wilkinson. Bayesian inference for nonlinear multivariate diffusion models observed with error. *Comput. Stat. Data Anal.*, 52(3):1674–1693, 2008.

[14] C. A. R. Hoare. Algorithm 65: find. *Commun. ACM*, 4(7):321–322, 1961.

[15] Jeroen D. Hol, Thomas B. Schn, and Fredrik Gustafsson. On resampling algorithms for particle filters. In *Nonlinear Statistical Signal Processing Workshop*, Cambridge, United Kingdom, 2006.

[16] M. Vernon Johns. Importance sampling for bootstrap confidence intervals. *Journal of the American Statistical Association*, 83(403):709–714, 1988.

[17] Mike Klaas, Nando de Freitas, and Arnaud Doucet. Toward practical N^2 Monte Carlo: the marginal particle filter. In *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 308–31, Arlington, Virginia, 2005. AUAI Press.

[18] Donald E. Knuth. Mathematical analysis of algorithms. Technical Report STAN-CS-71-206, Stanford University, Department of Computer Science, 1971.

[19] Finn E. Kydland and Edward C. Prescott. Time to build and aggregate fluctuations. *Econometrica*, 50(6):1345–1370, 1982.

[20] J. Liu and M. West. Combined parameter and state estimation in simulation-based filtering. In Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors, *Sequential Monte Carlo Methods in Practice. New York*. Springer-Verlag, New York, 2000.

[21] Paul Marjoram, John Molitor, Vincent Plagnol, and Simon Tavar. Markov chain Monte Carlo without likelihoods. *Proc. Natl. Acad. Sci. U. S. A.*, 100:15324–15328, 2003.

[22] Jerrold E. Marsden and Anthony J. Tromba. *Vector Calculus*. W. H. Freeman, 4th edition, 1996.

[23] Michael K. Pitt. Smooth particle filters for likelihood evaluation and maximisation. The Warwick Economics Research Paper Series (TWERPS) 651, University of Warwick, Department of Economics, 2002. Available at http://ideas.repec.org/p/wrk/warwec/651.html.

[24] Michael K. Pitt and Neil Shephard. Time-varying covariances: A factor stochastic volatility approach. In J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics 6*. Oxford University Press, 1999.

[25] George Poyiadjis, Arnaud Doucet, and Sumeetpal S. Singh. Particle methods for optimal filter derivative: Application to parameter estimation. In *IEEE ICASSP*, pages 925–928, 2005.

[26] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Second Edition)*. Springer-Verlag, New York, 2004.

[27] David Romer. *Advanced Macroeconomics*. McGraw-Hill/Irwin, May 2001.

[28] S. A. Sisson, Y. Fan, and Mark M. Tanaka. Sequential Monte Carlo without likelihoods. *Proc. Natl. Acad. Sci. U. S. A.*, 104:1760–1765, 2007.

[29] Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer-Verlag, New York, 2004.