UNIVERSITY OF CALIFORNIA
SANTA CRUZ

BOOST SMOOTHING AND SMRF MODELS

A project document submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

STATISTICS AND APPLIED MATHEMATICS

by

Jacob B. Colvin

September 2011

The project document of Jacob B. Colvin is approved:

Associate Professor Abel Rodriguez, Chair

Professor Raquel Prado

Table of Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Boost Smoothing and SMRF Models

by

Jacob B. Colvin

This proposal explores the use of Sequential Monte Carlo (SMC) for use in Bayesian inference for both sequential and non sequential problems. The main contribution are two new joint SMC smoothing algorithms, named Boost Smoothing and Kernel Smoothing, and a scheme for converting a Markov Random Field (MRF) into a state space model.

Part I reviews state space models with an introduction to Dynamic Linear Models (DLM) for linear Gaussian state space models and SMC for general state space models, both with optional static parameter estimation. The class of Forward-Backward SMC Smoothing algorithms is then introduced, with preexisting algorithms having a computational complexity of order $\mathcal{O}(TN^2)$. Boost Smoothing is introduced as a way of combining the results of K SMC Smoothing replications using $M = N/K$ particles and has an improved computational complexity of order $\mathcal{O}(TKN)$ for $K << N$. Kernel Smoothing is introduced as a way of using Kernel Density Estimation (KDE) to estimate the needed normalizing constant, and has a computation complexity of order $\mathcal{O}(TN \log N)$. This section is then concluded with a linear and a nonlinear univariate state space model to show the relative strengths of the different algorithms.

Part II discusses a variety of Markov Chain Monte Carlo (MCMC) implementations of MRFs with static parameters, and shows how MRFs can be transformed into both multivariate and univariate state space models by inducing an ordering on the hidden states. These state space models are called Sequential MRFs (SMRF) models. Numerical performance is evaluated on a nearest 4 neighbor Gaussian MRF on a regular lattice. For this example, it is shown that by using Boost Smoothing on the univariate SMRF model, samples from the joint posterior can be generated with order $\mathcal{O}(KNIJ^2)$ computational complexity.

Chapter 1

State Space Models

State space models are characterized by a set of unobservable hidden states, $X^T = \{X_t \in \mathbb{R}^n : t = 1, ..., T\}$, with observations $Y^T = \{Y_t \in \mathbb{R}^r : t = 1, ..., T\}$, such that Y_t depends only upon X_t. The hidden state X_t depends only on the hidden states X_{t-1} and X_{t+1} and the observation Y_t. The model is then described by the system equation $p(X_t|X_{t-1}, \theta)$, which represents a stochastic process in time, and the observa-

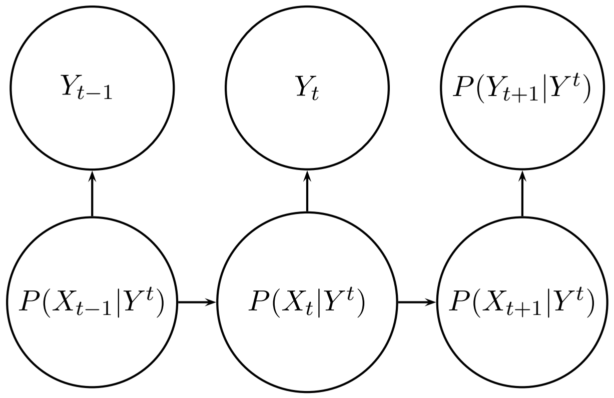


Figure 1.1: General Markov Structure Of State Space Models, Given Y^t

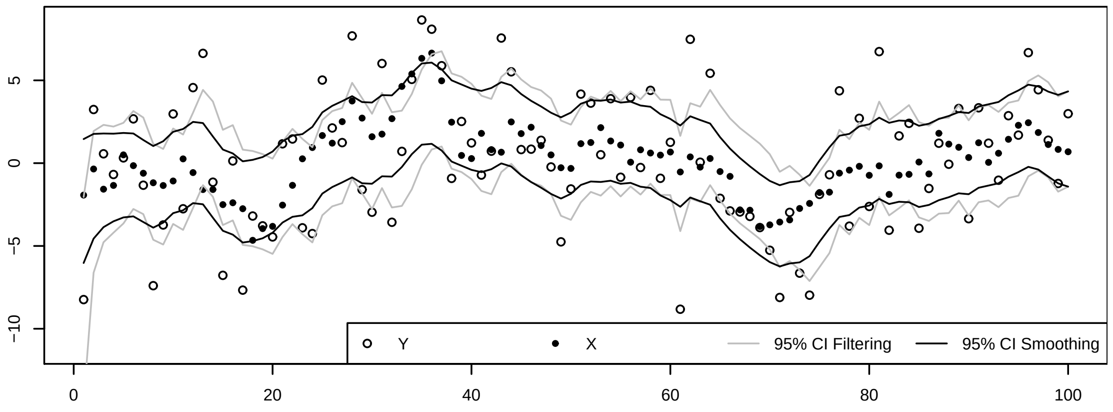


Figure 1.2: Example Of Filtering And Smoothing Distributions, 95% Credible Intervals

tion equation $p(Y_t|X_t, \theta)$ representing the error distribution of observation Y_t given the hidden state X_t, where θ is an optional vector of static parameters. This limiting of the dependence structure of the hidden states to only the immediately adjacent hidden states is known as the Markov property, and simplifies calculations greatly as illustrated in Figure 1.1. Inference on these models are usually visualized by the marginal distributions of either the filtering densities $p(X_t|Y^t)$ or the smoothing densities $p(X_t|Y^T)$ as shown in Figure 1.2. Under Bayesian inference the posterior is specified as follows.

$$p(X^T, \theta|Y^T) \propto p(X_1, \theta) \prod_{t=2}^{T} p(X_t|X_{t-1}, \theta) \prod_{t=1}^{T} p(Y_t|X_t, \theta)$$

1.1 Linear State Space Models

Multivariate Dynamic Linear Models are examples of state space models where both the observation and system equation are linear and Gaussian, which allows all distributions of interest to be analytically tractable. DLMs are described by the set

$\{F_t, G_t, V_t, W_t\}$ where t indexes time. This then defines a state space model as follows for $t \in 1:T$. [26, 28, 34]

$$Y_t | X_t \sim N(F_t' X_t, V_t)$$

$$X_t | X_{t-1} \sim N(G_t X_{t-1}, W_t)$$

Y_t	Observation Vector $(r \times 1)$	X_t	Hidden State Vector $(n \times 1)$
F_t	Observation Matrix $(n \times r)$	G_t	System Matrix $(n \times n)$
V_t	Observational Variance $(r \times r)$	W_t	System Variance $(n \times n)$

1.1.1 Filtering

Inference for the hidden states starts with a normal prior on $X_1 \sim N(a_1, R_1)$, and then proceeds sequentially, one observation at a time, creating a series of forward filtering distributions of the form $p(X_t|Y^t)$. For the hidden states, the filtering distributions are $p(X_t|Y^t) = N(m_t, C_t)$, which can be calculated recursively as shown below.

```
for $t = 1$ to $T$ do
    Prior: $p(X_t|Y^{t-1}) = N(a_t, R_t)$ where $a_t = G_t m_{t-1}, R_t = G_t C_{t-1} G_t' + W_t$
    Predictive: $p(Y_t|Y_{1:t-1}) = N(f_t, Q_t)$ where $f_t = F_t' a_t, Q_t = F_t' R_t F_t + V_t$
    Filtering: $p(X_t|Y^t) = N(m_t, C_t)$ where $m_t = a_t + A_t(Y_t - f_t)$, $C_t = R_t - A_t Q_t^{-1} A_t'$,
    and $A_t = R_t F_t Q_t^{-1}$
```

Algorithm 1: DLM Filtering Recursion

1.1.2 Smoothing

After filtering, the smoothing distributions can be calculated as shown below. These calculations with the final filtering distribution $N(m_T, C_T) = N(a_{T|T}, R_{T|T})$.

```
**Forward Filter:** $p(X_t|Y^t) = N(m_t, C_t)$ for $t = 1 : T$
**for** $t = T - 1$ to 1 **do**
   **Marginal Smoothing:** $p(X_t|Y^T) \sim N(a_{t|T}, R_{t|T})$ where
   $a_{t|T} = m_t + B_t(a_{t+1|T} - a_{t+1}), R_{t|T} = C_t - B_t(R_{t+1|T} - R_{t+1})B'_t, B_t = C_t G'_{t+1} R^{-1}_{t+1}$
   **Joint Smoothing:** $p(X^T|Y^T) = N\left(\begin{bmatrix} a_{1|T} \\ \vdots \\ a_{T|T} \end{bmatrix}, \begin{bmatrix} \Sigma_{1,1} & \dots & \Sigma_{1,T} \\ \vdots & \ddots & \vdots \\ \Sigma_{T,1} & \dots & \Sigma_{T|T} \end{bmatrix}\right)$ where
   $\Sigma_{i,j} = A_{i,j} R_{j|T}$ for $i \leq j$ and $A_{i,j} = \prod_{k=i}^{j} B_k$.
```
Algorithm 2: DLM Smoothing Recursion

1.1.3 Sampling

The joint posterior of a DLM can be factored using the filtering distributions.

$$p(X^T|\theta, Y^T) = p(X_T, \theta|Y^T) \prod_{t=1}^{T-1} p(X_{t+1}|\theta, X_t) p(X_t|\theta, Y^t)$$

This structure allows states to be efficiently sampled backwards from $p(X_t|X_{t+1}, \theta, Y^t)$, and is known as Forward-Filter Backward-Sampling (FFBS). [7, 14]

```
**Forward Filter:** $p(X_t|Y^t) = N(m_t, C_t)$ for $t = 1 : T$
**Sample:** $X_T \sim N(m_T, C_T)$
**for** $t = T - 1$ to 1 **do**
   **Sample:** $X_t|X_{t+1} \sim N(m^*_t, C^*_t)$, where
      $m^*_t = m_t + B_t(X_{t+1} - a_{t+1})$,$C^*_t = C_t - B_t R_{t+1} B'_t$, and $B_t = C_t G'_{t+1} R^{-1}_{t+1}$
```
Algorithm 3: Forward-Filter Backward-Sampling (FFBS)

1.1.4 Static Parameters

Static parameters arise in DLMs when any of the matrices $\{F_t, G_t, V_t, W_t\}$ are not time dependent constants, but functions of unknown parameters. Common examples include placing a prior on the observation or system variance indicating that

V_t and or W_t are unknown constants. In AR(p) time series models, G_t may need to be estimated as part of a larger model structure. Except for the case where V_t, W_t, and C_0 are all proportional to a common unknown parameter, models with unknown parameters will not be analytically tractable. For off-line inference, MCMC methods [30] can be used, and work by iteratively sampling from the full conditional posterior distributions, discarding the initial samples from the burn in stage until the chain has reached the stationary distribution. With an appropriately chosen conjugate prior, $p(\theta|X^T, Y^T)$ can usually be sampled directly. This is generally preferable because it will minimize the autocorrelation of the MCMC sample.

```
for $i = 1$ to $N$ do
    Sample: $X^{T(i)}|\theta^{(i-1)}, Y^T \sim p(X^T|Y^T, \theta^{(i-1)})$ via FFBS
    Sample: $\theta^{(i)}|X^{T(i)}, Y^T \sim p(\theta|X^{T(i)}, Y^T)$
```

Algorithm 4: DLM Gibbs

1.2 Nonlinear State Space Models

The advent of MCMC methods became part of the the foundation of modern Bayesian inference by allowing for efficient sampling of intractable distributions derived from high dimensional integrals. Unfortunately, the Markov Chain theory behind MCMC is unscalable for on-line time series applications. Even when computation can be performed off-line and time complexity is not an issue, finding efficient proposal distributions for the hidden states can be difficult. Because the hidden states can be multi modal and highly correlated, sampling each state individually can cause the Markov chain to mix slowly. Due to these limitations, Sequential Monte Carlo [6] has become a popular alternative to MCMC for state space models.

1.2.1 Importance Sampling

This is the classic Monte Carlo method for approximating expectations.

$$\mathbb{E}_f[h(X)] = \int_{\mathcal{X}} h(x)f(x)\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} h(x^{(i)}) \quad \text{where} \quad x^{(i)} \sim f(x)$$

Importance Sampling (IS) generalizes this approach by allowing samples to come from a different importance distribution, creating a weighted sample $\{w^{(i)}, \tilde{x}^{(i)}\}_{i=1}^{N}$ with weights $w^{(i)} = f(\tilde{x}^{(i)})/g(\tilde{x}^{(i)})$ to correct for bias and an importance sample $\tilde{x}^{(i)} \sim g(x)$ to define the support. This weighted sample is a discrete approximation of $f(x)$ such that $\dot{p}(x) = \sum_{i=1}^{N} w^{(i)}\delta_{x^{(i)}}(x)$, and is used to calculate expectations as follows. [30]

$$\mathbb{E}_f[h(x)] = \int_{\mathcal{X}} h(x)\frac{f(x)}{g(x)}g(x)\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(\tilde{x}^{(i)})}{g(\tilde{x}^{(i)})}h(\tilde{x}^{(i)}) \quad \text{where} \quad \tilde{x}^{(i)} \sim g(x)$$

The choice of the importance distribution $g(x)$ is arbitrary, but must cover the full support of $f(x)$. Figure 1.3 shows different weighted samples approximating the same distribution, but using different importance distributions. To ensure finite variance of the estimator, $g(x)$ must have heavier tails then the target distribution $f(x)$, and thus $\lim_{x\to\pm\infty} f(x)/g(x) < \infty$. For convenience, $g(x)$ is typically chosen to be an easily sampled distribution. Also note that $f(x)$ may only be known up to a proportionality constant, so weights may need to be normalized for certain calculations.

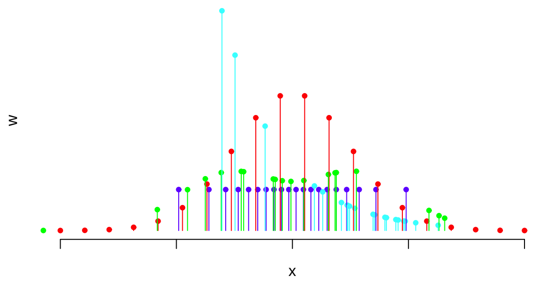


Figure 1.3: Examples Of Weighted Samples, approximating the same distribution

1.2.2 Filtering

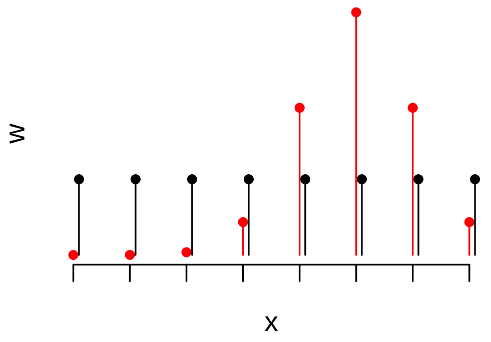
The description of SMC filtering algorithms will begin by describing a series of IS operations. Assume $\dot{q}(x) \sim \left\{w^{(i)}, x^{(i)}\right\}_{i=1}^{N}$ and $\dot{q}(x_{t:t+1}) \sim \left\{w^{(i)}, x^{(i)}_{t:t+1}\right\}_{i=1}^{N}$.

- **Reweight:** $\dot{q}(x|y) \propto q(y|x)\dot{q}(x) \sim \left\{q(y|x^{(i)})w^{(i)}, x^{(i)}\right\}_{i=1}^{N}$
- **Resample:** $[\dot{q}(x)] \sim \left\{1/N, \tilde{x}^{(i)}\right\}_{i=1}^{N}$ where $\tilde{x}^{(i)} \sim \dot{q}(x)$
- **Propagate:** $q(x_{t+1}|x_t)\dot{q}(x_t) \sim \left\{w^{(i)}, x^{(i)}_{t:t+1}\right\}_{i=1}^{N}$ where $x^{(i)}_{t+1} \sim q(x^{(i)}_{t+1}|x^{(i)}_t)$
- **Marginalize:** $\int \dot{q}(x_{t:t+1})\mathrm{d}x_t \sim \left(w^{(i)}, x^{(i)}_{t+1}\right)_{i=1}^{N}$

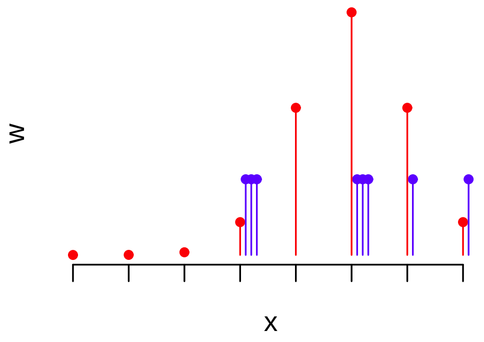
These operations can be visualized as seen in Figure 1.4. Most existing SMC algorithms are summarized by the following recursion.

$$\begin{aligned}\dot{p}(x^{t+1}|y^{t+1}) &\propto p(y_{t+1}|x_{t+1})p(x_{t+1}|x_t)\dot{p}(x^t, y^t) \\ &\propto \overbrace{\frac{p(y_{t+1}|x_{t+1})p(x_{t+1}|x_t)}{v_t(y^{t+1}, x^t)q_t(x_{t+1}|x^t, y^{t+1})}}^{\text{(4)Reweight}} \overbrace{q_t(x_{t+1}|x^t, y^{t+1})}^{\text{(3)Propagate}} \overbrace{\left[v_t(y^{t+1}, x^t)\dot{p}(x^t|y^t)\right]}^{\text{(1-2)Auxiliary-Resample}}\end{aligned}$$

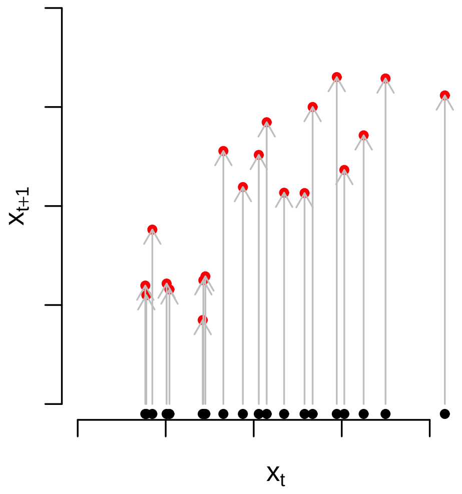
Implementation is dependent on selecting the auxiliary function, $v_t(y_{t+1}, x_t)$, to select which particles to propagate, and the particle propagation distribution, $q_t(x_{t+1}|x_t, y_{t+1})$,



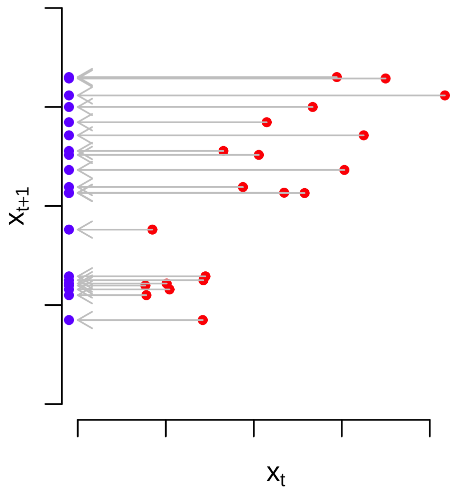
(a) Reweight: $\dot{q}(x|y) \propto q(y|x)\dot{q}(x)$, in this case the left hand side is the unequally weighted sample (red), and the right hand side is the result of applying a reweight function $q(y|x)$ to the equally weighted sample $\dot{q}(x)$ in black.



(b) Resample: the original distribution $\dot{q}(x)$ is represented in red, while the resampled version $[\dot{q}(x)]$ is the equally weighted approximation of the same distribution represented in blue.



(c) Propagate: in black is the original weighted sample $\dot{q}(x_t)$, for which the particle locations x_t are propagated using $q(x_{t+1}|x_t)$ to obtain $\dot{p}(x_{t+1}, x_t)$ in red.



(d) Marginalize: in red is the weighted sample $\dot{p}(x_{t+1}, x_t)$, which is marginalized by dropping particle locations x_t to form the weighted sample $\dot{p}(x_{t+1})$ in blue.

Figure 1.4: Visualizations Of Importance Sampling Operations

and is described in Algorithm 5. Depending on these choices, the algorithm simplifies to one of the SMC algorithms described in the next subsection.

```
for $t = 1$ to $T$ do
  Auxiliary: $\dot{\tilde{p}}(x^t|y^{t+1}) \sim v_t(y^{t+1}, x^t)\dot{p}(x^t|y^t) \to (\tilde{w}_t^{(i)}, x^{t(i)})_{i=1}^N$
  Resample: $\dot{\tilde{p}}(x^t|y^{t+1}) \sim \left[\dot{\tilde{p}}(x^t|y^{t+1})\right] \to (1/N, \tilde{x}^{t(i)})_{i=1}^N$
  Propagate: $\dot{\tilde{p}}(x^{t+1}|y^{t+1}) \sim q_t(x_{t+1}|x^t, y^{t+1})\dot{\tilde{p}}(x^t|y^{t+1}) \to (1/N, \tilde{x}^{t+1(i)})_{i=1}^N$
  Reweight: $\dot{p}(x^{t+1}|y^{t+1}) \sim \frac{p(y_{t+1}|x_{t+1})p(x_{t+1}|x_t)}{v_t(y^{t+1},x^t)q_t(x_{t+1}|x^t,y^{t+1})}\dot{\tilde{p}}(x^{t+1}|y^{t+1}) \to (w_{t+1}^{(i)}, x^{t+1(i)})_{i=1}^N$
```

Algorithm 5: SMC Filtering Recursion

1.2.2.1 Sequential Importance Sampling (SIS)

Since SIS lacks a resample step, set $v_t(y^{t+1}, x_t) \propto 1$ and remember that the reweight step will be reweighting a weighted sample, which will eventually become degenerate without a resample step.

$$\dot{p}(x^{t+1}|y^{t+1}) \propto \frac{p(y_{t+1}|x_{t+1})p(x_{t+1}|x_t)}{q_t(x_{t+1}|x_t, y^{t+1})} q_t(x_{t+1}|x^t, y^{t+1})\dot{p}(x^t|y^t)$$

1.2.2.2 Sequential Importance Resampling (SIR)

By adding a resample step, the reweight step will be reweighting an equally weighted sample. This allows the algorithm to recover from degeneracy in the filtering distributions. [16]

$$\dot{p}(x^{t+1}|y^{t+1}) \propto \frac{p(y_{t+1}|x_{t+1})p(x_{t+1}|x_t)}{q_t(x_{t+1}|x_t, y^{t+1})} q_t(x_{t+1}|x_t, y^{t+1}) \left[\dot{p}(x^t|y^t)\right]$$

1.2.2.3 Auxiliary Particle Filter (APF)

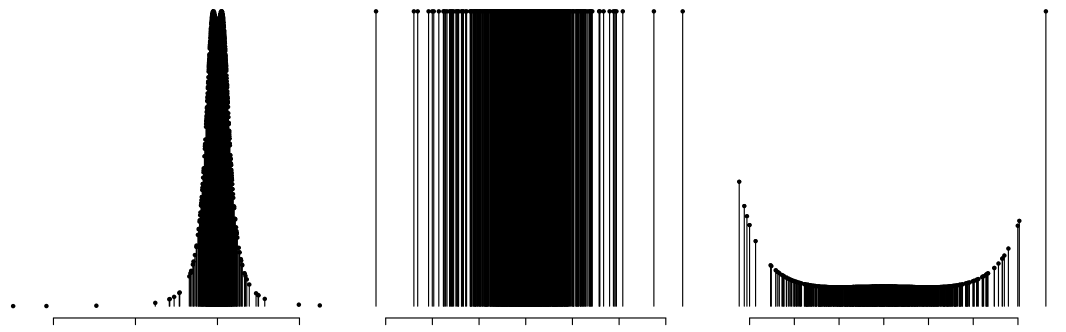
The next improvement concerned selecting the optimal particles to propagate in the resample step using the auxiliary reweight function $v_t(y^{t+1}, x^t)$, and then correcting for this bias in the following reweight step. The most natural auxiliary distribution to use would be $p(y_{t+1}|x_t) = \int p(y_{t+1}|x_{t+1})p(x_{t+1}|x_t)\mathrm{d}x_{t+1}$ so that particles are resampled according to $p(x_t|y^{t+1})$, but this integral is generally not available in analytical form. Alternatively, a point estimate $\hat{x}_{t+1}$, such as $\mathbb{E}(x_{t+1}|x_t, y_{t+1})$, can be used with the observation equation, so that $p(y_{t+1}|x_t)$ is approximated by $p(y_{t+1}|\hat{x}_{t+1})$. [11, 27]

$$\dot{p}(x^{t+1}|y^{t+1}) \propto \frac{p(y_{t+1}|x_{t+1})p(x_{t+1}|x_t)}{v_t(y^{t+1}, x^t)q_t(x_{t+1}|x^t, y^{t+1})} q_t(x_{t+1}|x^t, y^{t+1}) \left[v_t(y^{t+1}, x^t)\dot{p}(x^t|y^t)\right]$$

1.2.2.4 Particle Learning (PL)

In Particle Learning, a specific auxiliary function and propagation distribution is chosen such that the final reweight step produces equally weighted samples from $p(x_{t+1}|y^{t+1})$ by definition. This also allows the reweight step following the propagation step to be omitted. [8, 18, 24]

$$\begin{aligned} v_t(y^{t+1}, x^t) &= \int p(y_{t+1}|x_{t+1})p(x_{t+1}|x_t)\mathrm{d}x_{t+1} \\ q_t(x_{t+1}|x_t, y_{t+1}) &\propto p(y_{t+1}|x_{t+1})p(x_{t+1}|x_t) \\ \dot{p}(x^{t+1}|y^{t+1}) &\propto p(x_{t+1}|x_t, y^{t+1}) \left[p(y_{t+1}|x_t)\dot{p}(x^t|y^t)\right] \end{aligned}$$



(a) Robust: $g(x) \sim T_{df=2}$ (b) Optimal: $g(x) \sim T_{df=5}$ (c) Degenerate: $g(x) \sim N(0,1)$

Figure 1.5: Plots Of Weights Versus Locations For Different Proposals $g(x)$. In this case, the target is $f(x) \sim T_{df=5}$ and $w^{(i)} = f(x^{(i)})/g(x^{(i)})$.

1.2.2.5 Discussion On Forward Filtering

One metric for evaluating the efficiency of an importance sample is the Effective Sample Size (ESS) [23], which is calculated using the normalized weights as

$$ESS(x) = 1 \Big/ \sum_{i=1}^{N} \left(w^{(i)}\right)^2 \qquad \text{where} \qquad x \sim \left\{w^{(i)}, x^{(i)}\right\}_{i=1}^{N}$$

Values near N imply an equally weighted sample, while values near 1 imply sample degeneracy where a single particle has very large weight.

SMC performance is heavily dependent on the chosen proposal distribution. Although the system equation $p(x_{t+1}|x_t)$ is a natural choice, it is often considered unsatisfactory because it does not incorporate the current observation y_{t+1}. When analytically tractable, one natural choice is known as the optimal proposal, $p(x_{t+1}|x_t, y^{t+1}) \propto p(y_{t+1}|x_{t+1})p(x_{t+1}|x_t)$, which minimizes the variance of the weights (or maximizes the ESS), conditional on the previous particles x_t. However, not all particles are good candidates for propagating to the next step, motivating the need for the auxiliary step. Figure 1.5 shows graphically how weighted samples behave under different proposal distributions. [6]

Various forms of systematic sampling [20] can be used in the resample step to reduce MCMC error compared to using simple random sampling. For example, after normalizing weights $w^{(i)} = w^{(i)} / \sum_{j=1}^{N} w^{(j)}$, take n_i copies of particle i where $n_i = \lfloor N w^{(i)} \rfloor$ to select the first $\sum n_i$ particles deterministically, and then perform simple random sampling to select $N - \sum n_i$ particles with weights $w^{*(i)} \propto w^{(i)} - n_i/N$.

To reduce machine round off error from degenerate samples with very small weights, it is best to work on the log weight scale where weights can be multiplied and divided with addition and subtraction. Since the weights need only be specified up to a normalizing constant, it is best to center the log weights such that $\max(w^{(i)}) = 1$. This can be done by setting $w^{(i)} = \exp\left(\log w^{(i)} - \max(\log w^{(i)})\right)$.

To save space and computation, note that $\dot{p}(x^t|y^t) \sim \left\{w^{t(i)}, x^{t(i)}\right\}_{i=1}^{N}$ can be marginalized to retain only the forward filtering distributions $\dot{p}(x_t|y^t) \sim \left\{w_t^{(i)}, x_t^{(i)}\right\}_{i=1}^{N}$ if the filter smoothing approximation of $p(x^t|y^t)$ is not needed. Additionally, the auxiliary weights can be reused in the reweight step if preserved through the resample step.

1.2.3 Static Parameters

While static parameters can be modeled as non-evolving hidden states initially sampled from the prior distribution, the resulting samples quickly degenerate as the initial sample is over-dispersed and particle trajectories are removed and duplicated in the resample step. One of the earliest solutions was to add a small amount of random noise in the system equation for the static parameters [16], but the inherent loss of information associated with this artificial evolution created over-dispersion in the posteriors. This was later addressed by shrinking the resampled static parameters about the mean. [22] Later, the next big development was the introduction of sufficient statistics that allowed for the particle trajectories to be summarized as a deterministically evolving state of finite dimension, allowing the static parameters to be resampled directly at each time step. [33] Unfortunately, not all general state space models allow for such a finite-dimensional sufficient statistic. To incorporate static parameter estimation, a new replenish step is added to create the following algorithms.

1.2.3.1 Liu & West

This method is similar to discount factors for DLMs, and does not rely on the sufficient statistics structure. The idea is to replenish from a multivariate normal while maintaining the same covariance and mean by using a shrinkage factor controlled by δ. The parameters should be transformed to an appropriate scale for a normal approximation before replenishing. For a good approximation, the discount factor should be set close to 1 to minimize the use of shrinkage. Typically discount factors are $0.95 \leq \delta \leq 0.99$. [22]

$$\dot{p}(x^{t+1}, \tilde{\theta}|y^{t+1}) \propto \overbrace{\frac{p(y_{t+1}|x_{t+1}, \tilde{\theta})p(x_{t+1}|x_t, \tilde{\theta})}{v_t(x^t, \tilde{\theta}, y^{t+1})q_t(x_{t+1}|x^t, \tilde{\theta}, y^{t+1})}}^{\text{(5)Reweight}} \times$$

$$\overbrace{q_t(x_{t+1}|x^t, \tilde{\theta}, y^{t+1})}^{\text{(4)Propagate}} \int \overbrace{q_t(\tilde{\theta}|\theta)}^{\text{(3)Replenish}} \overbrace{\left[v_t(x^t, \theta, y^{t+1})\dot{p}(x^t, \theta|y^t)\right]}^{\text{(1-2)Auxiliary-Resample}} \mathrm{d}\theta$$

where $q_t(\tilde{\theta}|\theta) = N(a\theta + (1-a)\bar{\theta},\, (1-a^2)\text{Cov}(\theta))$, and $a = (3\delta - 1)/2\delta$.

1.2.3.2 Storvik

This method introduced the use of sufficient statistics, which allows for replenishing the static parameters by recursively updating $S_t = f(x_t, y_t, S_{t-1})$. If the dimension of S_t does not grow with time, then the entire particle trajectory is not required when evaluating $p(\theta|S_t)$. The replenish distribution $q_t(\theta|x^t, \theta, y^{t+1})$ is arbitrary, but a direct sample from $p(\tilde{\theta}|x^t, y^t)$ is the most common. [33]

$$\dot{p}(x^{t+1}, \tilde{\theta}|y^{t+1}) \propto \overbrace{\frac{p(y_{t+1}|x_{t+1}, \tilde{\theta})p(x_{t+1}|x_t, \tilde{\theta})p(\tilde{\theta}|x^t, y^t)}{v_t(x^t, \theta, y^{t+1})q_t(x_{t+1}|\tilde{\theta}, x^t, y^{t+1})q_t(\tilde{\theta}|x^t, y^{t+1})}}^{\text{(5)Reweight}} \times$$

$$\overbrace{q_t(x_{t+1}|\tilde{\theta}, x^t, y^{t+1})}^{\text{(4)Propagate}} \overbrace{q_t(\tilde{\theta}|x^t, y^{t+1})}^{\text{(3)Replenish}} \int \overbrace{\left[v_t(x^t, \theta, y^{t+1})\dot{p}(x^t, \theta|y^t)\right]}^{\text{(1-2)Auxiliary-Resample}} \mathrm{d}\theta$$

1.2.3.3 Particle Learning (PL)

Choose $v_t(x^t, y^{t+1})$ and $q_t(x_{t+1}|x^t, y^{t+1})$ as described earlier for PL with fixed parameters. After the propagation step at time t, replenish $\tilde{\theta} \sim p(\tilde{\theta}|x^t, y^t)$ to maintain sample diversity. Notice that the distribution of θ after the auxiliary step is the same as $\tilde{\theta}$ after the replenish step, and hence the replenish step may be performed only as need. [8, 9, 18]

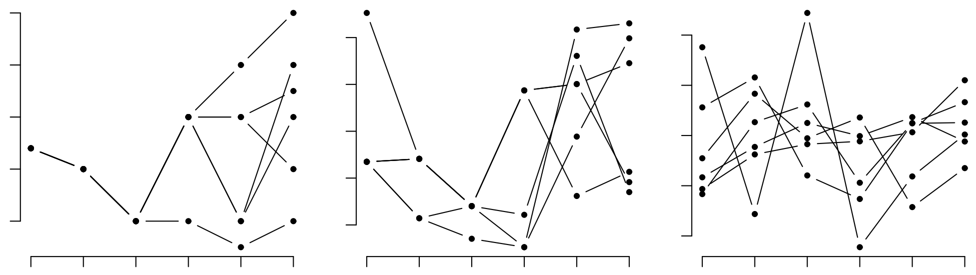
$$\dot{p}(x^{t+1}, \tilde{\theta}|y^{t+1}) \propto \overbrace{p(\tilde{\theta}|x^{t+1}, y^{t+1})}^{\text{(4)Replenish}} \overbrace{p(x_{t+1}|x^t, \theta, y^{t+1})}^{\text{(3)Propagate}} \overbrace{\left[p(y_{t+1}|x_t, \theta)\dot{p}(x^t, \theta|y^t)\right]}^{\text{(1-2)Auxiliary-Resample}}$$

1.2.3.4 Discussion On Static Parameter Estimation

While both Storvik and PL usually assume the common case that $p(\tilde{\theta}|S_t)$ can be sampled directly, PL can also be replenished from $p(\tilde{\theta}|\theta, S_t)$ in a MCMC step without burn in since θ is already an equally weighted sample from the stationary distribution after the auxiliary-resample step. The difficulty, is that PL requires $p(y_{t+1}|x_t, \theta)$ and $p(x_{t+1}|x_t, \theta, y_{t+1})$ to be analytically tractable while Storvik allows for arbitrary auxiliary and proposal distributions. For Storvik, a series of MCMC iterations with minimal burn in is often used in the replenish step when θ cannot be directly sampled.

1.2.4 Forward-Backward Smoothing

The most basic form of SMC smoothing consists of using the particle trajectories from the final filtering distribution and is known as simple smoothing [20], filter smoothing, or trajectory smoothing. However, filter smoothing progressively degenerates as the particle trajectories become less diverse after each resample step. Simple smoothing can also be used to recover the sufficient statistics used in static parameter estimation, since it can be shown that $S_t^{(i)}$ is a function of particle trajectories $x^{t(i)}$



(a) Filter Smoothing (b) GDW04 Smoothing Or Boost Smoothing (c) Kernel Smoothing

Figure 1.6: Example Trajectories of Different Smoothing Methods: (a) Simple Smoothing: particles become less diverse as t decreases, (b) GDW04 or Boost Smoothing: no new particle locations, but particle trajectories are reassigned, (c) Kernel Smoothing: new particle locations.

as indexed in the particle approximation $\dot{p}(x^t|y^t)$. Hence, if filter smoothing performs poorly, sufficient statistic methods should perform poorly as well. Figure 1.6 shows example particle trajectories of three types of SMC smoothing methods.

The forward-backward class of smoothing algorithms starts with forward filtering to find $\dot{p}(x_t|y^t)$ for $t = 1 : T$, and then recursively calculates the smoothing densities using the previously sampled $\dot{p}(x_{t+1:T}|y^T)$. By the Markov property, conditional distributions can be simplified such that $p(x^t|x_{t+1:T}, y^T) = p(x^t|x_{t+1}, y^t)$. Usually, $p(x_t|x_{t+1}, y^t)$ does not allow for direct sampling, with the notable exception of DLMs via FFBS, and so the main difficulty is calculating the normalizing constant.

$$\begin{aligned} p(x_{t:T}|y^T) = p(x_t|x_{t+1}, y^t)p(x_{t+1:T}|y^T) &= \frac{p(x_{t:t+1}|y^t)}{p(x_{t+1}|y^t)}p(x_{t+1:T}|y^T) \\ &= \frac{p(x_{t+1}|x_t)p(x_t|y^t)}{p(x_{t+1}|y^t)}p(x_{t+1:T}|y^T) \qquad (1.2.1) \end{aligned}$$

$$p(x_t|x_{t+1:T}, y^T) \propto p(x_{t+1}|x_t)p(x_t|y^t)p(x_{t+1:T}|y^T) \qquad (1.2.2)$$

The first SMC Smoothing method for joint samples (GDW04) [15] and the new Boost Smoothing algorithm use the proportionality argument in (1.2.2) to normalize the weight of each particle. Kernel Smoothing is another new alternative solution that uses kernel density estimation to approximate the normalizing constant directly using (1.2.1).

The derivation for the static parameter case was introduced in [24]. The version presented here allows for sampling from the filter smoothing distributions $p(x^t|y^t)$ to create joint samples with $p(x_{t+1:T}, \theta|y^T)$. This allows for forward-backward smoothing at all time steps $t < T$, or periodically depending upon the rate that particle trajectories degenerate in order to save on computation.

$$\begin{aligned}
p(x^T, \theta|y^T) &= p(x^t|\theta, x_{t+1}, y^t)p(x_{t+1:T}, \theta|y^T) \\
&= \frac{p(x_{t+1}|x_t, \theta)p(\theta|x^t, y^t)p(x^t|y^t)}{p(x_{t+1}, \theta|y^t)} p(x_{t+1:T}, \theta|y^T) \\
p(x^t|x_{t+1:T}, \theta, y^t) &\propto p(x_{t+1}|x_t, \theta)p(\theta|x^t, y^t)p(x^t|y^t)p(x_{t+1:T}, \theta|y^T)
\end{aligned}$$

Hence, given $x^{(i)}_{t+1:T}, \theta^{(i)} \sim p(x_{t+1:T}, \theta|y^T)$, sample $x^{t(j)} \sim p(x^t|y^t)$ with weights proportional to $w^{(j)} \propto p(\theta^{(i)}|x^{t(j)}, y^t)p(x^{(i)}_{t+1}|x^{(j)}_t, \theta^{(i)})$.

Additionally, there is an alternative class of algorithms known as two filter smoothing, which is a technique for calculating the marginal distributions $p(x_t|y^T)$. This is done by using the forward and backward filtering densities, $p(x^t|y^t)$ and $p(x_{t:T}|y_{t:T})$, in order to sample new states. This class of algorithms are generally order $\mathcal{O}(TN^2)$ complexity. [5, 13]

1.2.4.1 GDW04 Smoothing

The first SMC smoothing method was introduced in [12] and produced weighted samples of the marginal distributions $p(x_t|y^T)$. Next, the method was improved by [15] to generate joint samples from $p(x^T|y^T)$. This method was then extended for static parameter estimation in [24] for Particle Learning, and is presented here for the case where $\dot{p}(x_t,\theta|y^t)$ is not an equally weighted sample. The computational cost is $\mathcal{O}(TN)$ per sample or $\mathcal{O}(TN^2)$ for N samples. Since the forward filtering cost is typically $\mathcal{O}(TN)$ for N samples, the computational complexity is dominated by the smoothing process. A second issue is that no new particle locations, $x_t^{(i)}$, are sampled, merely the links connecting particle trajectories are reshuffled as in Figure 1.6b. To generate a sample $\tilde{x}^T$, proceed as described in Algorithm 6.

```
**Forward Filter** to generate $\dot{p}(x_t,\theta|y^t) = \left\{w_t^{(i)}, x_t^{(i)}, \theta^{(i)}\right\}_{i=1}^N$ for $t = 1:T$
**Sample** $\tilde{x}_T, \tilde{\theta} \sim \dot{p}(x_T,\theta|y^T)$
**for** $t = T-1$ to 1 **do**
  **Sample** $\tilde{x}_t$ from $x_t^{(i)}$ with probability $w_{t|T}^{(i)} \propto p(\tilde{\theta}|x^{t(i)}, y^t)p(\tilde{x}_{t+1}|x_t^{(i)}, \tilde{\theta})w_t^{(i)}$
```

Algorithm 6: GDW04 Smoothing

Theoretical results assuming fixed parameters show that the rate of convergence in mean squared error is order $\mathcal{O}(1/N)$ for the number of particles and order $\mathcal{O}(1/\sqrt{N})$ in terms of computational complexity. For details, see Theorem A.1.2 in Appendix A.1.

1.2.4.2 Boost Smoothing (New)

The main computational problem with the GDW04 algorithm is that the reduction in MC error provided by one additional particle is eventually overshadowed by the computational cost of that one additional particle.

One solution is to run the filtering algorithm on N particles and the GDW04 smoothing algorithm M times with K particles, combining results for a total of $N = KM$ samples. Now when GDW04 is applied in the backward pass, the cost is only $\mathcal{O}(TK)$ per sample instead of $\mathcal{O}(TN)$. Since the computational complexity of SMC filtering algorithms is generally $O(TN)$, there is no reason to divide the forward pass into groups. In Machine Learning terminology this strategy is called Boosting, where each group of size K represents a weak learner (compared to a group of size N) which can be combined to make a strong learner. In this case, the motivation for using the weak learners is computational, because now the computational complexity is order $\mathcal{O}(TMK^2) = \mathcal{O}(TKN)$ instead of $\mathcal{O}(TN^2)$. This process is described in Algorithm 7.

```
**Forward Filter** to generate $\dot{p}(x_t, \theta | y^t) = \left\{ w_t^{(i)}, x_t^{(i)}, \theta^{(i)} \right\}_{i=1}^{N}$ for $t = 1 : T$
**Sample** without replacement, $M$ groups of $K$ particles
**Backward Smooth** by performing GDW04 on each of the $M$ groups of $K$ particles
```

Algorithm 7: Boost Smoothing

Theoretical results suggest that Boost Smoothing is making a trade off between the bias and the variance of the estimate. As seen in Appendix A.1 and assuming fixed parameters, the rate of convergence in mean squared error is order $\mathcal{O}(1/KM + \mu^2)$ where μ is the bias of GDW04 with K particles. If the rate of this bias decreases on the order of $\mathcal{O}(1/N)$, and setting $M = K$, then Boost Smoothing has the same order of convergence in mean squared error, but with a computational complexity of $\mathcal{O}(TN^{3/2})$. In practice, for well behaving state space models and sufficiently large K, the estimate is essentially unbiased relative to the variance, and the results of Section 1.3 and 1.4 suggest choosing $K << M$ given sufficient signal in the observations. In this setting, for fixed K large enough that $E(h(x^T|y^T)) < \epsilon$, the computational complexity of Boost Smoothing becomes linear in the number of particles. The choice of K needs to be tuned for each individual problem and should increase as N increases to even out the sources of MC error, with an ideal choice minimizing the MC error for a given amount of computation.

1.2.4.3 Kernel Smoothing (New)

An alternative smoothing method arises from using kernel density estimation to calculate the normalizing constant in (1.2.1), avoiding the proportionality argument from (1.2.2). Furthermore, this allows new particle locations to be sampled in the backward smoothing step. After the obligatory forward filtering pass, the next step is to calculate $\dot{p}(x_{t:t+1}|y^T)$ for $t = T-1 : 1$ where x_t are new particle locations and x_{t+1} are the one step filter smoothing estimates. These will be used later to bind the different bivariate samples $\dot{p}(x_{t:t+1}|y^T)$, to create joint samples $\dot{p}(x^T|y^T)$.

Recall that $\dot{p}(x) = \frac{1}{n}\sum_{i=1}^{N} w^{(i)}\delta_{x^{(i)}}(x)$ is the weighted sample representation of a density function. Then the kernel density estimate is

$$\hat{p}(x) = \frac{1}{N}\sum_{i=1}^{N} w^{(i)} K_h(x^{(i)} - x)$$

where $K_h(u)$ is a kernel with bandwidth h and $\mathrm{E}(K_h(u)) = 0$. One common example is the Gaussian kernel, defined as $K_h(u) = N(u|0, h^2)$. The kernel provides for density estimation between particle locations by smoothing over the gaps in the weighted sample. In general, the only requirement for consistency of the density estimate is that $\lim_{N\to\infty} h = 0$ while $\lim_{N\to\infty} Nh = \infty$. In histogram terminology, this merely means that the number of samples per bin must increase despite the bin width decreasing. A bandwidth that is too small produces jagged density estimates, while large bandwidths over smooth the density estimate and leads to over-dispersion. For known bandwidth and certain kernel families, efficient calculations can be performed using discrete Fourier transformations. This creates a grid of K values that can be interpolated to find density estimates at arbitrary locations in $\mathcal{O}(NK \log K)$ time. Efficient implementations of kernel density estimation are available in many statistics packages.

Automatic bandwidth selection schemes range from computationally complex cross validation techniques to fast plug-in estimates usually based upon a scaled standard deviation. Plug-in estimators perform well when the estimated density is approximately normal, while cross-validation or user defined bandwidths are best for distributions when the standard deviation is a poor estimate of variability. An example of this is the multi-modal distributions arising from the nonlinear state space model of Section

1.4.

This new smoothing algorithm is motivated by the following recursion, similar to running SIR in the reverse direction, where the needed filtering and prediction densities for the reweight step are calculated using kernel density estimation.

$$\dot{p}(x_{t:t+1}|y^T) = \overbrace{\frac{p(x_{t+1}|x_t)\hat{p}(x_t|y^t)}{\hat{p}(x_{t+1}|y^t)q(x_t|x_{t+1}, y^t)}}^{\text{reweight}} \overbrace{q(x_t|x_{t+1}, y^t)}^{\text{propagate}} \int \dot{p}(x_{t+1:t+2}|y^T)\mathrm{d}x_{t+2}$$

For best performance, the predictive density can be rearranged such that the system equation becomes the kernel in KDE.

$$p(x_{t+1}|y^t) = \int p(x_{t+1}|x_t)p(x_t|y^t)\mathrm{d}x_t \simeq \int K_h\left(x_t - \mathrm{E}(x_{t+1}|x_t)\right)\dot{p}(x_t|y^t)\mathrm{d}x_t$$

For the case when the system equation is Gaussian, a Gaussian kernel with bandwidth equal to the standard deviation of the system equation can be used. This improves the density estimate by removing the limiting argument that $h \to \infty$, which is particularly important when estimating tail densities. In the general case, $p(x_{t+1}|y^T)$ can be approximated by propagating $x_{t+1}^{(i)} \sim p(x_{t+1}|x_t^{(i)})$ and performing KDE on $\dot{p}(x_{t+1}|y^t) = \left\{w_t^{(i)}, x_{t+1}^{(i)}\right\}_{i=1}^N$.

This method also depends upon a reasonable choice for the backward propagate distribution $q(x_t|x_{t+1}, y^t)$, which is chosen based on similar concerns as the forward filtering propagate distribution. The naturally choice would be $q(x_t|x_{t+1}, y^t) \propto p(x_{t+1}|x_t)p(x_t|y^t)$ whereupon the reweight step would not be necessary. This choice is rarely feasible except for a Dynamic Linear Model (DLM), where this algorithm reduces to Forward Filter-Backward Sampling (FFBS).

As stated, this algorithm produces samples for the marginal distributions $p(x_t|y^T)$, and can be used to find filter smoothing estimates of $p(x_{t:t+k}|y^T)$, but this quickly degenerates for large k. However, samples of the form $p(x_{t:t+1}|y^T)$ can be modified to generate joint samples $p(x^T|y^T)$ in $\mathcal{O}(TN\log N)$ time by using the one step filter smoothing distribution to capture the dependency between states. By sorting the filter smoothing estimate $\check{x}_{t+1}$ from $\dot{p}(x_{t:t+1}|y^T)$ and $\tilde{x}_{t+1}$ from $\dot{p}(x_{t+1:t+2}|y^T)$, $\dot{p}(x_{t:t+1}|y^T)$ can be approximated by $\left\{\check{x}_t^{(i)}, \tilde{x}_{t+1:t+2}^{(i)}\right\}_{i=1}^{N}$. This is because the distribution of the order statistics allows for the simple smoothing estimate $\check{x}_{t+1}$ to be replenished by a second sample from the same marginal distribution, since the distribution of central order statistics converge to fixed values and hence central order variates are exchangeable in the limit. Therefore, joint smoothing is done by using a replenishing series of one step filter smoothing operations. The $\mathcal{O}(N\log N)$ time complexity arises from the need to sort particle locations. See Algorithm 8 for implementation details.

The reason for using three sorting operations per iteration instead of two is to avoid resampling all of $\tilde{x}_{t+1:T}$ particles after each iteration. Hence, $\tilde{x}_t$ can immediately be stored on disk to free up main memory. Note that $u(1:N)$ represents the indices for sorting $t(1:N)$, much like how $t(1:N)$ represents the indices for sorting $x_{t+1}^{(1:N)}$.

Unfortunately, due to the poor performance of multivariate KDE, the algorithm here is specific for state space models that are univariate in both the observations and hidden states, and without the inclusion of static parameter estimation. However,

```
**Forward Filter** to generate $\dot{p}(x_t, \theta | y^t) = \left\{ w_t^{(i)}, x_t^{(i)}, \theta^{(i)} \right\}_{i=1}^N$ for $t = 1 : T$
**Sample** $\tilde{x}_T^{(i)} \sim \dot{p}(x_T | y^T)$ for $i = 1 : N$
**for** $t = T - 1$ to $1$ **do**
  **Propagate** $x_t^{(i)} \sim q\left( x_t \middle| \tilde{x}_{t+1}^{(i)}, y^t \right)$
  **Weights** $w_t^{(i)} \propto p\left( \tilde{x}_{t+1}^{(i)} \middle| x_t^{(i)} \right) \hat{p}\left( x_t^{(i)} \middle| y^t \right) \Big/ \hat{p}\left( \tilde{x}_{t+1}^{(i)} \middle| y^t \right) q\left( x_t^{(i)} \middle| \tilde{x}_{t+1}^{(i)}, . \right)$
  **Resample** $\dot{p}(x_{t:t+1} | y^T) = \left\{ w_t^{(i)}, x_t^{(i)}, \tilde{x}_{t+1}^{(i)} \right\}_{i=1}^N$ to create
      $\dot{p}(x_{t:t+1} | y^T) = \{1, \check{x}_t^{(i)}, \check{x}_{t+1}^{(i)}\}_{i=1}^N$
  **Create index** $s(i)$ for $i = 1 : N$ such that $\tilde{x}_{t+1}^{s(i)} < \tilde{x}_{t+1:T}^{s(i+1)}$
  **Create index** $t(i)$ for $i = 1 : N$ such that $\check{x}_{t+1}^{t(i)} < \check{x}_{t+1:T}^{t(i+1)}$
  **Create index** $u(i)$ for $i = 1 : N$ such that $t(u(i)) < t(u(i+1))$
  **Then** $\dot{p}(x_{t:T} | y^T) = \{1, \tilde{x}_t^{(i)} = \check{x}_t^{s(u(i))}, \tilde{x}_{t+1:T}^{(i)}\}_{i=1}^N$
```

Algorithm 8: Kernel Smoothing

if multivariate KDE is acceptable, this algorithm easily generalizes to higher dimensions and static parameters.

1.3 Example: AR(1), Fixed And Static Parameters

This section discusses a standard AR(1) model with normal errors with either fixed parameters (Model 1) or static parameters (Model 2). For each model, the data set consist of a common hidden state x^T of length 100 generated from the system equation, and three sets of observations y^T for low, medium, and high Signal to Noise Ratios (SNR). To compare the numerical performance of each algorithm, Root Mean Squared Error (RMSE) calculations are used with respect to 95% Credible Intervals (CI) over 100 repeated runs of each algorithm with $2^{13} = 8192$ particles. The model is described

Model 1	**Model 2**
$\phi = 0.9$	$p(\phi) = N(0, \phi_0 = 1)$
$p(x_1 \mid \phi, \sigma^2) = N\left(0, x_V = \frac{\sigma^2}{1-\phi^2}\right)$	$p(x_1) \propto 1$
$\sigma^2 = 1$	$p(\sigma^2) = IG(\sigma_a^2 = 10, \sigma_b^2 = 10)$
$\tau^2 = \{0.1^2, 1, 10^2\}$	$p(\tau^2) = IG(\tau_a^2 = 10, \tau_b^2 = \{0.1, 10, 100\})$

Table 1.1: Linear Models 1 & 2: Prior Specifications

Algorithm	**Auxiliary**	**Propagate**
Bootstrap		$p(x_{t+1} \mid x_t)$
SIR		$p(x_{t+1} \mid x_t, y_{t+1})$
APF	$N\left(y_{t+1} \mid \phi x_t, \tau^2\right)$	$p(x_{t+1} \mid x_t, y_{t+1})$
PL	$p(y_{t+1} \mid x_t)$	$p(x_{t+1} \mid x_t, y_{t+1})$
Kernel		$p(x_t \mid x_{t+1}, y^t)$
Liu & West	$p(y_{t+1} \mid x_t, \theta)$	$p(x_{t+1} \mid x_t, \tilde{\theta}, y^{t+1}) q_t(\tilde{\theta} \mid \theta, \delta = 0.99)$
Storvik	$p(y_{t+1} \mid x_t, \theta)$	$p(x_{t+1} \mid x_t, \tilde{\theta}, y^{t+1}) p(\tilde{\theta} \mid x^t, y^t)$
PL	$p(y_{t+1} \mid x_t, \theta)$	$p(\tilde{\theta} \mid x^{t+1}, y^{t+1}) p(x_{t+1} \mid x_t, \theta, y^{t+1})$

Table 1.2: Linear Models 1 & 2: Algorithm Specifications ($\theta = \{\phi, \sigma^2, \tau^2\}$)

with prior specifications found in Table 1.1 and implementation choices in Table 1.2.

$$y_t \mid x_t, \tau^2 \sim N\left(x_t, \tau^2\right)$$

$$x_t \mid x_{t-1}, \phi, \sigma^2 \sim N\left(\phi x_{t-1}, \sigma^2\right)$$

From this state space model, a number of conditional and derived distributions are used for the different MCMC and SMC implementations and are presented next.

$$
\begin{aligned}
x_1|x_2, y_1, \phi, \sigma^2, \tau^2 &\sim N\left(\frac{\phi x_2 \sigma^{-2} + y_1 \tau^{-2}}{\tau^{-2} + \phi^2 \sigma^{-2}}, \frac{1}{\tau^{-2} + \phi^2 \sigma^{-2}}\right) \\
x_t|x_{t-1}, x_{t+1}, y_t, \phi, \sigma^2, \tau^2 &\sim N\left(\frac{\phi x_{t-1} \sigma^{-2} + \phi x_{t+1} \sigma^{-2} + y_t \tau^{-2}}{\sigma^{-2} + \phi^2 \sigma^{-2} + \tau^{-2}}, \frac{1}{\sigma^{-2} + \phi^2 \sigma^{-2} + \tau^{-2}}\right) \\
x_T|x_{T-1}, y_T, \phi, \sigma^2, \tau^2 &\sim N\left(\frac{\phi x_{T-1} \sigma^{-2} + y_T \tau^{-2}}{\sigma^{-2} + \tau^{-2}}, \frac{1}{\sigma^{-2} + \tau^{-2}}\right) \\
x_1|y_1, \tau^2 &\sim N(y_1, \tau^2) \\
x_{t+1}|x_t, y_{t+1}, \phi, \sigma^2, \tau^2 &\sim N\left(\frac{\phi x_t \sigma^{-2} + y_{t+1} \tau^{-2}}{\sigma^{-2} + \tau^{-2}}, \frac{1}{\sigma^{-2} + \tau^{-2}}\right) \\
y_{t+1}|x_t, \phi, \sigma^2, \tau^2 &\sim N\left(\phi x_t, \tau^2 + \sigma^2\right) \\
x_t|x_{t+1}, y^t, \phi, \sigma^2, \tau^2 &\sim N\left(\frac{\mathrm{E}(x_t|y_t)/\mathrm{V}(x_t|y^t) + \phi/\sigma^2 x_{t+1}}{1/\mathrm{V}(x_t|y^t) + \phi^2/\sigma^2}, \frac{1}{1/\mathrm{V}(x_t|y^t) + \phi^2/\sigma^2}\right) \\
\phi|\sigma^2, x^t, y^t &\sim N\left(\frac{\sum_{i=2}^t x_i x_{i-1}}{\phi_0^{-1} + \sum_{i=1}^{t-1} x_i^2}, \frac{\sigma^2}{\phi_0^{-1} + \sum_{i=1}^{t-1} x_i^2}\right) \\
\sigma^2|x^t, y^t &\sim IG\left(\sigma_a^2 + \frac{t-1}{2}, \sigma_b^2 + \frac{1}{2}\left(\sum_{i=2}^t x_i^2 - \frac{\left(\sum_{i=2}^t x_i x_{i-1}\right)^2}{\phi_0^{-1} + \sum_{i=1}^{t-1} x_i^2}\right)\right) \\
\tau^2|x^t, y^t &\sim IG\left(\tau_a^2 + \frac{t}{2}, \tau_b^2 + \frac{1}{2}\sum_{i=1}^t (y_i - x_i)^2\right) \\
S_t = f(x_t, x_{t-1}, y_t, S_{t-1}) &= \left(t, \sum_{i=1}^t (y_i - x_i)^2, \sum_{i=1}^{t-1} x_i^2, \sum_{i=2}^t x_{i-1} x_i, \sum_{i=2}^t x_i^2\right)
\end{aligned}
$$

For the RMSE calculations, equations (1.3.1) and (1.3.2) are used for the analytic case where the quantiles ξ_p are known (Model 1) and equations (1.3.3) and (1.3.4) are used when the quantiles $\bar{x}_p$ are estimated from the data (Model 2). In these equations, t refers to a parameter at a particular time step and i refers to the replication. When presenting the results, boxplots will use equations (1.3.1) and (1.3.3), and equations (1.3.2) and (1.3.4) are used to show the RMSE as a function of time. The

RMSE boxplots are ideal for comparing algorithm performance in general, while plotting RMSE as a function of time is ideal for showing the expected variation in estimation performance. The RMSE for the static parameters are defined analogously.

$$RMSE(x^T) = \sqrt{\frac{1}{2NT}\sum_{t=1}^{T}\sum_{i=1}^{N}\left(x_{2.5\%}^{(i,t)} - \xi_{2.5\%}^{(t)}\right)^2 + \left(x_{97.5\%}^{(i,t)} - \xi_{97.5\%}^{(t)}\right)^2} \tag{1.3.1}$$

$$RMSE(x_t) = \sqrt{\frac{1}{2N}\sum_{i=1}^{N}\left(x_{2.5\%}^{(i,t)} - \xi_{2.5\%}^{(t)}\right)^2 + \left(x_{97.5\%}^{(i,t)} - \xi_{97.5\%}^{(t)}\right)^2} \tag{1.3.2}$$

$$RMSE(x^T) = \sqrt{\frac{1}{2NT}\sum_{t=1}^{T}\sum_{i=1}^{N}\left(x_{2.5\%}^{(i,t)} - \bar{x}_{2.5\%}^{(t)}\right)^2 + \left(x_{97.5\%}^{(i,t)} - \bar{x}_{97.5\%}^{(t)}\right)^2} \tag{1.3.3}$$

$$RMSE(x_t) = \sqrt{\frac{1}{2N}\sum_{i=1}^{N}\left(x_{2.5\%}^{(i,t)} - \bar{x}_{2.5\%}^{(t)}\right)^2 + \left(x_{97.5\%}^{(i,t)} - \bar{x}_{97.5\%}^{(t)}\right)^2} \tag{1.3.4}$$

1.3.1 Linear Model 1: Fixed Parameters

This simulation study quantifies the RMSE for both filtering and smoothing 95% credible intervals with a data set generated from the state space model under three different Signal to Noise Ratios (SNR) for the fixed parameter case. The marginal posteriors for the filtering and smoothing distributions can be found in Figures 1.7 and 1.10, the ESS values are described in Figures 1.12 and 1.8, and the RMSE results are shown in Figures 1.12 and 1.9.

For the filtering comparisons, the Bootstrap filter proved to be surprisingly resilient despite being the simplest of the filtering algorithms. In the low SNR case, the ESS values of the Bootstrap and SIR filter occasionally drop to about 80% while the ESS values of APF and PL continually stay near 100%, however there is no distinguishable difference in RMSE between the algorithms. Performance begins to be distinguishable in the medium SNR case, with PL having the most consistent RMSE performance. ESS values of SIR and APF are essentially the same with nearly the same RMSE performance over time. The Bootstrap filter has the best RMSE performance over most individual states despite its poor ESS performance. The catch is that when the ESS value drops there is usually a noticeable spike in the RMSE which makes its overall performance more erratic. PL offers the most reliable performance by generating i.i.d. samples, however these are not ideal for calculating quantiles where proposals with heavier tails can offer better precision. The situation changes again with a high SNR, with APF having the worse RMSE values. This is due to the auxiliary function using only a point estimate for the evolution for each particle, creating a degenerate weighted samples that underestimate the variance of the previous state.

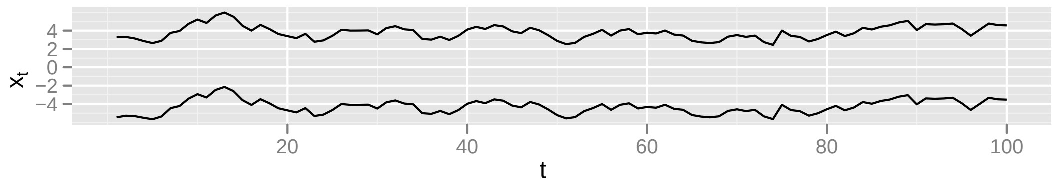
The smoothing algorithms where all implemented with a PL filter, with results from FFBS provided to give an estimate of the expected RMSE for i.i.d. samples. Results show that Kernel Smoothing is extremely fast and works particularly well in the low SNR situation because it is able to sample new particle locations in the smoothing step. While the same optimal backward propagation distribution is used from FFBS, Kernel Smoothing is still using kernel density estimation to estimate weights instead of assuming equal weights. Boost Smoothing works nearly as well as GDW04 smoothing for sufficiently large K, but with a substantially reduced computation cost. This confirms the results of Theorem A.1.3, suggesting that Boost Smoothing is approximately equal in performance to GDW04 when the value of K is large enough that Boost Smoothing is unbiased. Moreover, this minimum threshold for K seems to increase with lower SNR models.

Observed computation time can be seen in Table 1.3. The code was developed in R [29], and while attempts where made to optimize performance, there is a certain divergence from what theory would suggest. Namely, that for fixed N, the computation time for Boost Smoothing should decrease as K decreases. Results do not show this for small K, suggesting that the overhead of using an interpretive programming language becomes more significant as K decreases. Additionally, the code was run under a shared departmental cluster running the ROCKS operating system, so results may vary.

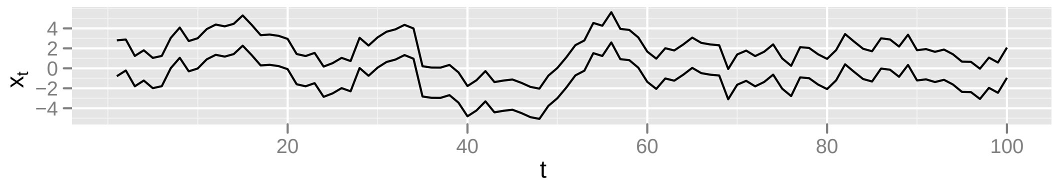
1.3.2 Linear Model 2: Static Parameters

This example is for comparing the numerical precision of the different algorithms when incorporating static parameter estimation. Since these models are not analytic, the results of a Gibbs sampler using FFBS is used to calculate the $\bar{x}_p$ values in the RMSE calculations. Because it is impractical to repeat this MCMC for each of the filtering distributions, the $\bar{x}_p^{(i,t)}$ values in equation (1.3.3) are calculated for each algorithms separately. Since the Liu & West algorithm utilizes a discount factor($\delta = .99$), the results will not be consistent with Storvik or PL unless $\delta \to 1$. PL is used for filtering when comparing the RMSE results for smoothing under different values of K.

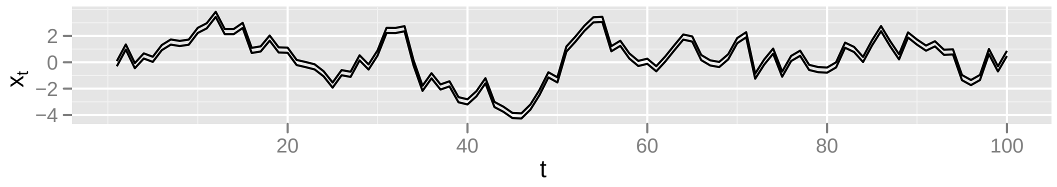
Of the filtering algorithms, only Storvik does not maintain high ESS values throughout, and this is particularly a problem at the start of the time series when the static parameter posteriors are rapidly changing. The drop in ESS is caused by



(a) Low SNR: Filtering 95% CI x

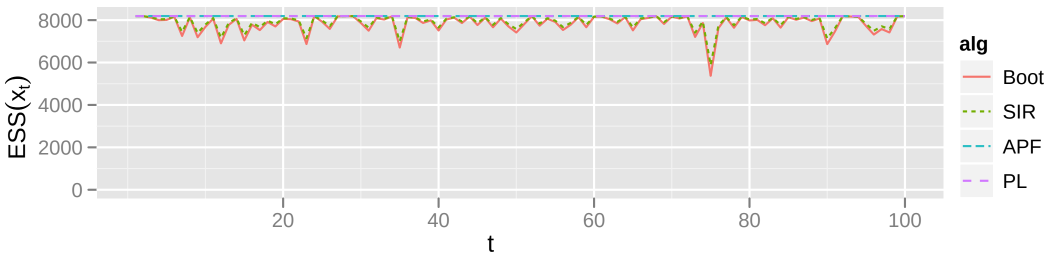


(b) Medium SNR: Filtering 95% CI x

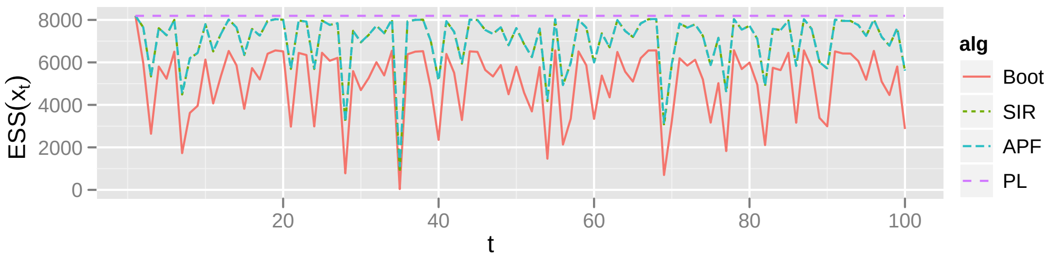


(c) High SNR: Filtering 95% CI x

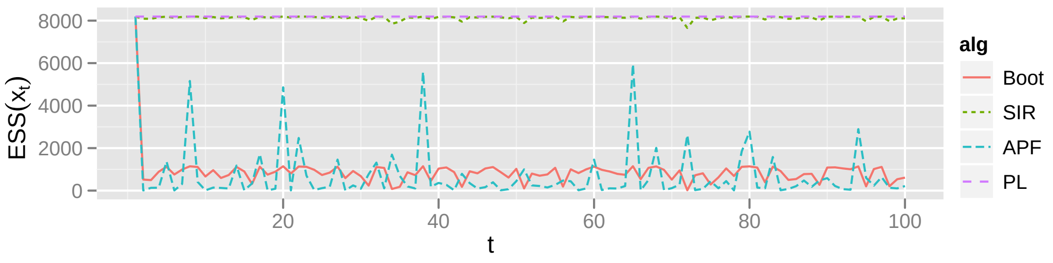
Figure 1.7: Linear Model 1: Filtering Posteriors. Note that in the low SNR case, very little inference can be performed on the hidden states.



(a) Low SNR: Filtering ESS(x_t)

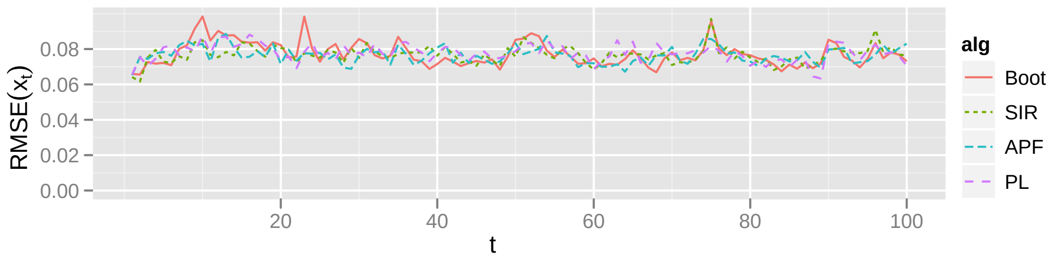


(b) Medium SNR: Filtering ESS(x_t)

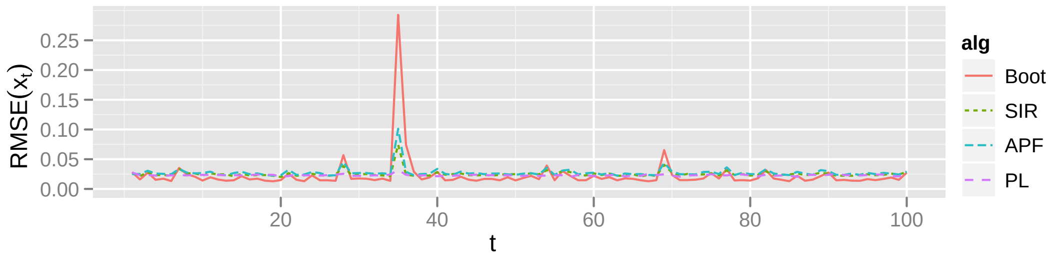


(c) High SNR: Filtering ESS(x_t)

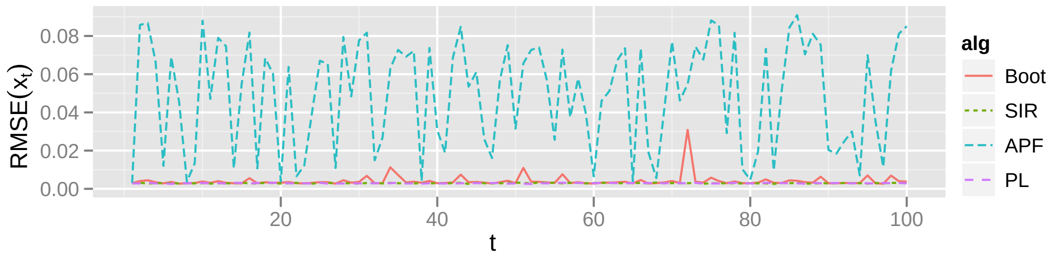
Figure 1.8: Linear Model 1: Filtering ESS(x_t). Results show that in the low SNR case (a), all algorithms show adequate ESS performance with APF and PL consistently at 100% and so showing the usefulness of the auxiliary function. In the medium SNR case (b), the point estimate for the auxiliary function used in the APF is insufficient to match the performance of PL. Both APF and SIR use the optimal proposal distribution, which explain the improvement over the Bootstrap filter using the system equation. In the high SNR case (c), what is most prominently demonstrated is the importance of using the optimal proposal distribution, and the adverse effect of using a poor point estimate for the auxiliary function.



(a) Low SNR: Filtering RMSE(x_t)

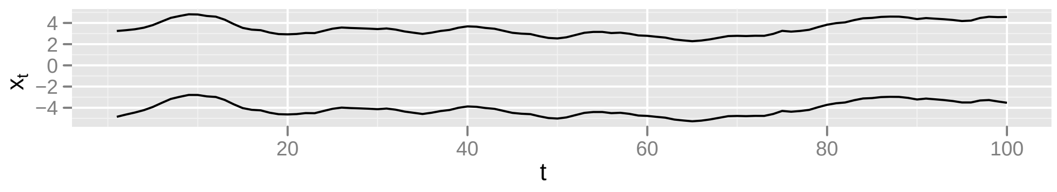


(b) Medium SNR: Filtering RMSE(x_t)

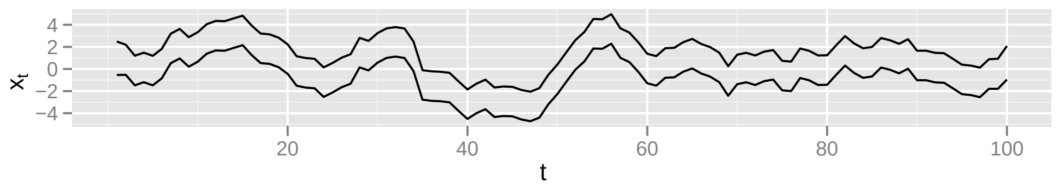


(c) High SNR: Filtering RMSE(x_t)

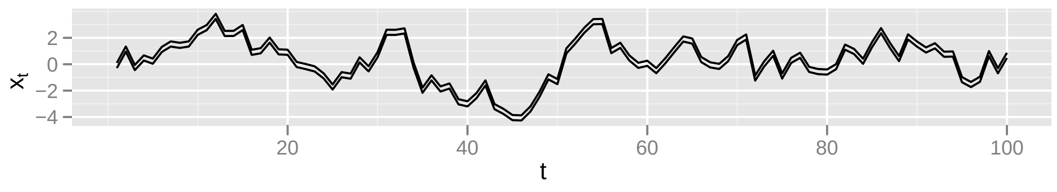
Figure 1.9: Linear Model 1: Filtering RMSE(x_t). Figure (a) shows that in a low SNR, all filters perform about the same. Figure (b) shows the Bootstrap filter suffering a large spike in RMSE that would have been reduced had the filter been using a better propagation distribution as in SIR. Figure (c) shows how using a point estimate for the system equation in the auxiliary function can produce very erratic estimates in the APF.



(a) Low SNR: Smoothing 95% CI x

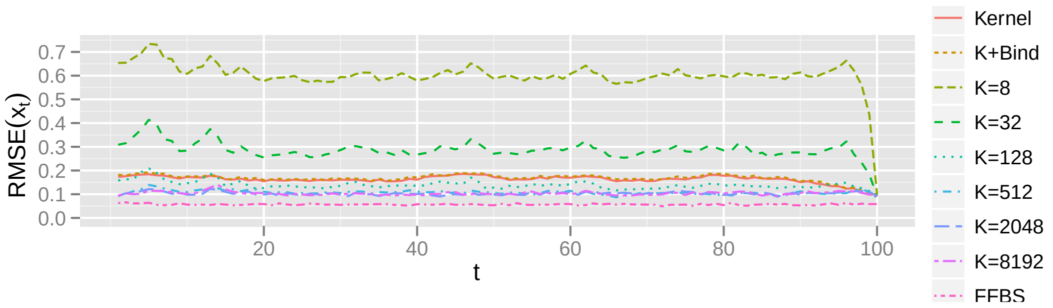


(b) Medium SNR: Smoothing 95% CI x

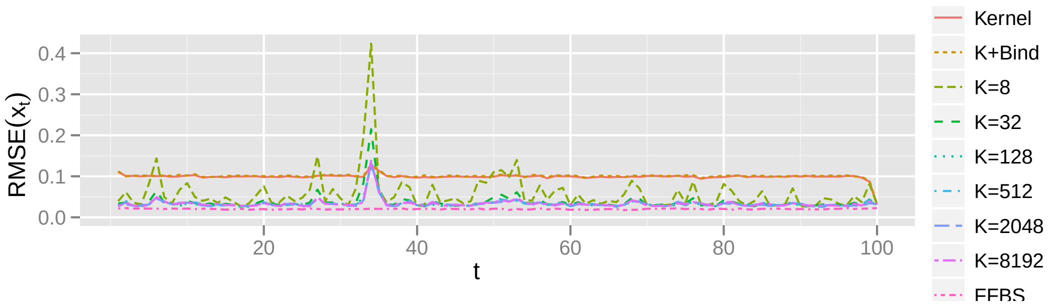


(c) High SNR: Smoothing 95% CI x

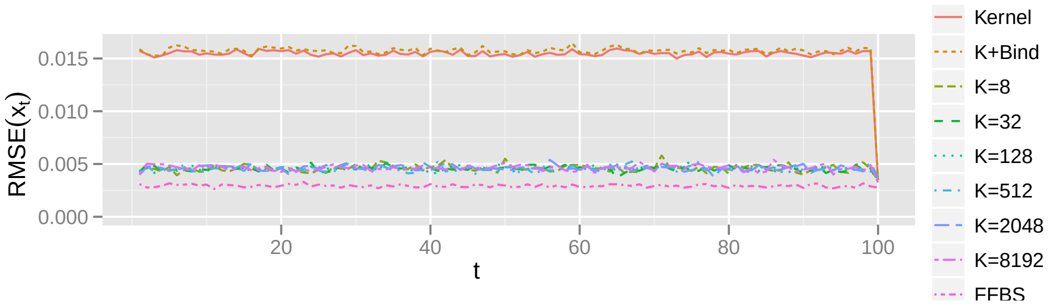
Figure 1.10: Linear Model 1: Smoothing Posteriors. Again, even with smoothing, very little inference can be done in the low SNR case.



(a) Low SNR: Smoothing RMSE(x_t)



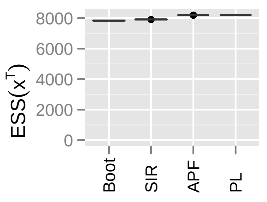
(b) Medium SNR: Smoothing RMSE(x_t)



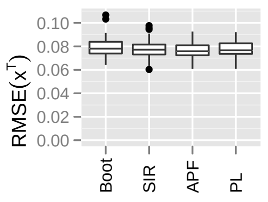
(c) High SNR: Smoothing RMSE(x_t)

Figure 1.11: Linear Model 1: Smoothing RMSE(x_t). Kernel Smoothing (NlogN) shows very consistent rate of RMSE, which is due to bias induced by the kernel bandwidth, and is exceptionally fast. Boost Smoothing performs as well as GDW04 ($K = N$) for sufficiently large K, showing that there is not always a trade off between speed and precision.

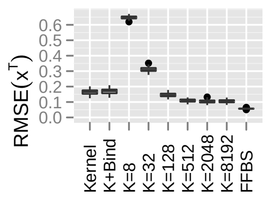
Low SNR



(a) Filtering ESS(x^T)

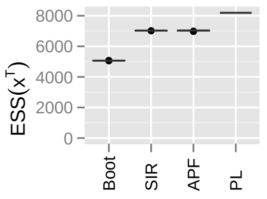


(b) Filtering RMSE(x^T)

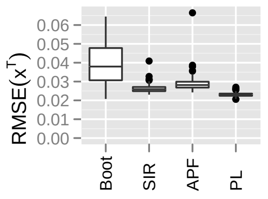


(c) Smoothing RMSE(x^T)

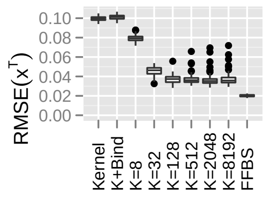
Medium SNR



(d) Filtering ESS(x^T)

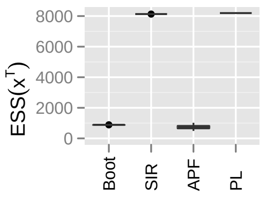


(e) Filtering RMSE(x^T)

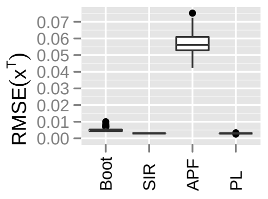


(f) Smoothing RMSE(x^T)

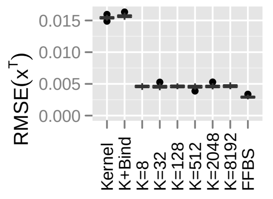
High SNR



(g) Filtering ESS(x^T)



(h) Filtering RMSE(x^T)



(i) Smoothing RMSE(x^T)

Figure 1.12: Linear Model 1: Boxplots For ESS, Filtering, And Smoothing. Note that the Bootstrap filter performs nearly as well as the other filtering algorithms (b), (e), (h), and for Boost Smoothing how the RMSE performance stops improving for sufficiently large K, (c), (f), (i).

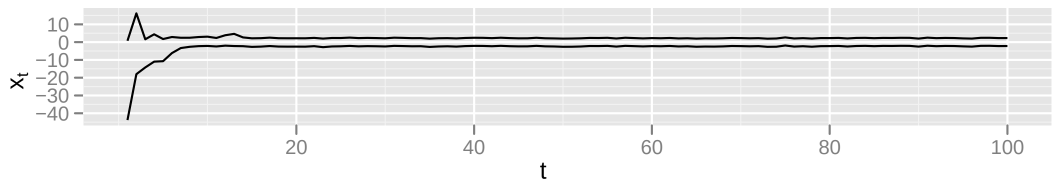
Algorithm		**Run Time (Seconds)**
Bootstrap Filter	(Boot)	1.3
Sequential Importance Resampling	(SIR)	2.4
Auxiliary Particle Filter	(APF)	3.0
Particle Learning	(PL)	1.5
Kernel Smoothing	(Kernel)	4.9
+ Marginal Binding	(K+Bind)	6.6
Boost Smoothing	K=8	167.1
Boost Smoothing	K=32	137.7
Boost Smoothing	K=128	153.2
Boost Smoothing	K=512	270.4
Boost Smoothing	K=2048	777.2
GDW04	N=8192	2949.8

Table 1.3: Linear Model 1: Average Run Time. The filtering algorithms are exceptionally fast compared to the smoothing algorithms. Of the Smoothing algorithms, Kernel Smoothing is significantly faster then the other smoothing methods. As expected, the run time of Boost Smoothing is heavily dependent on K, and is significantly faster then GDW04. The reason Boost Smoothing with $K = 8$ is slower then with $K = 32$ or 128 is due to the importance of vectorizing operations when using an interpretive language like R.

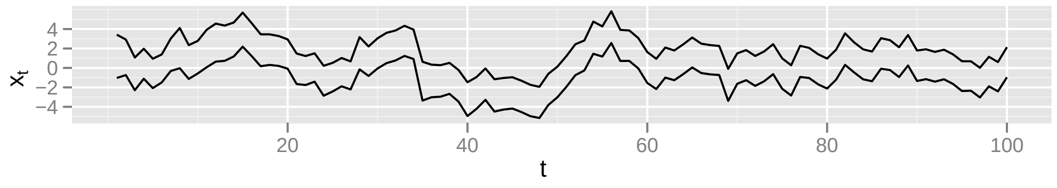
replenishing the static parameters before the propagation step without the ability to sample directly from $p(\theta|x^t, y^{t+1})$. The filtering RMSE plots of the states show periods of increased MC error in the low and medium SNR cases and relatively flat and consistent performance for the high SNR case with PL offering the best implementation. The upward slope of the RMSE in the high SNR case can be attributable to the progressive degradation of the sufficient statistics for the static parameters. This would noticeable in the other models for either a much larger time series or fewer particles. The advantages of Particle Learning become more apparent when looking at the RMSE plots for the static parameters, especially in the low SNR model.

Smoothing RMSE plots show that SMC can have difficulty estimating the earlier states in the series, particularly in the low SNR example. This numerical instability is not present in the medium and high SNR examples, and seems to be related to the much wider credible intervals for the hidden states at the start of the low SNR model. It is curious that the smallest value of K yields the best smoothing performance in some cases, a trend not observed in the fixed parameter setting.

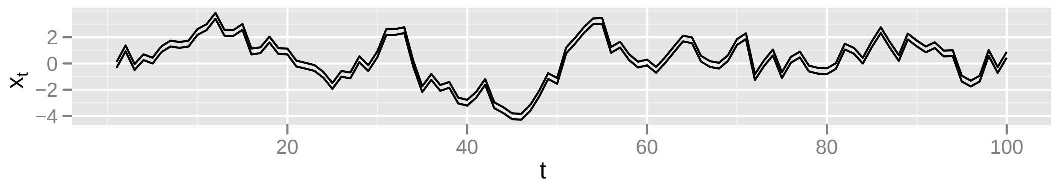
The timing results in Table 1.4 are given to verify effect of algorithms with different computation complexity. While it appears that $K = 128$ is the fastest choice, this is really a side effect of programming overhead from too fine a parallelization for these smaller values of K, and better programming techniques would fix this issue. These results show that for $K = 512$, Boost Smoothing is over 10 times faster than GDW04 Smoothing, and in the medium and high SNR cases, even smaller choices of K would be appropriate yielding even faster performance.



(a) Low SNR: Filtering 95% CI x

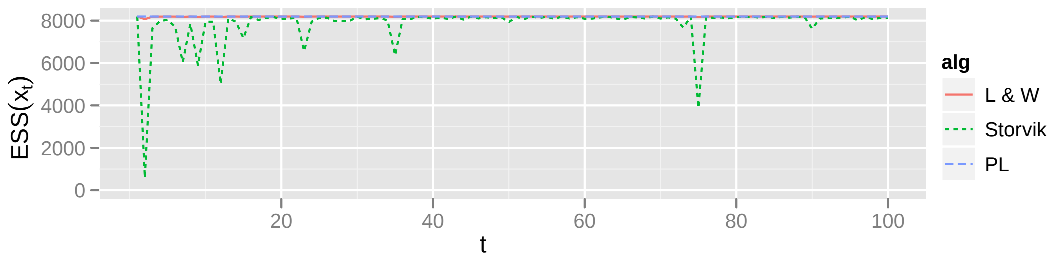


(b) Medium SNR: Filtering 95% CI x

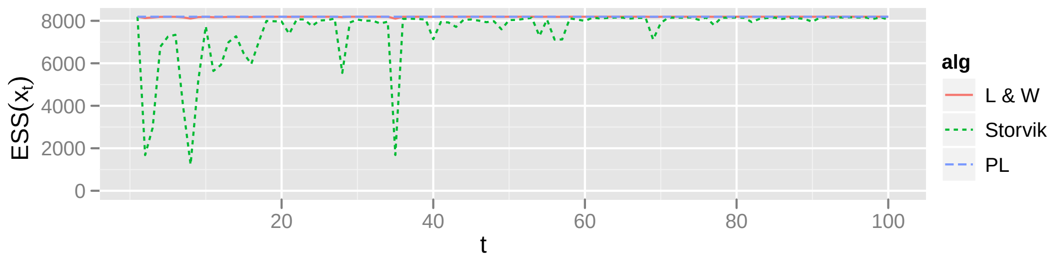


(c) High SNR: Filtering 95% CI x

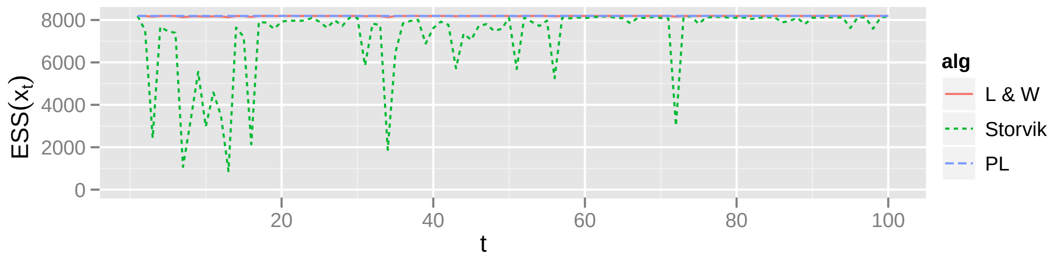
Figure 1.13: Linear Model 2: Filtering Posteriors. While the width of the interval bands are relatively consistent throughout the series for the Medium and High SNR cases, the low SNR case has an extremely wide interval in the beginning of the series. The presence of these wide intervals then become the dominate source of RMSE in the later RMSE boxplots.



(a) Low SNR: Filtering ESS(x_t)

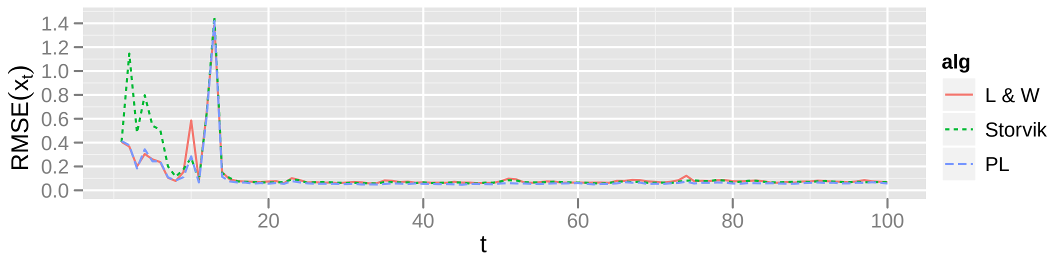


(b) Medium SNR: Filtering ESS(x_t)

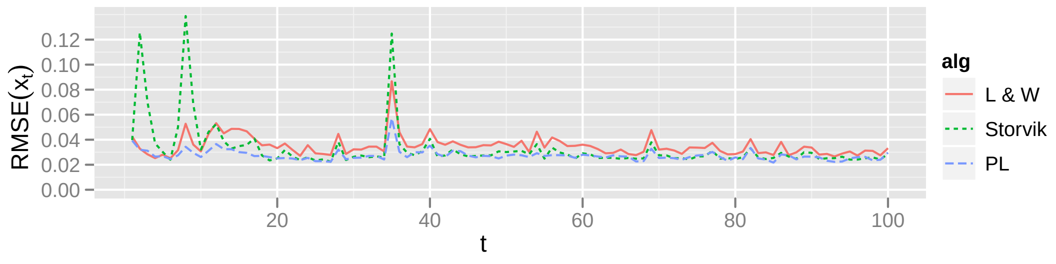


(c) High SNR: Filtering ESS(x_t)

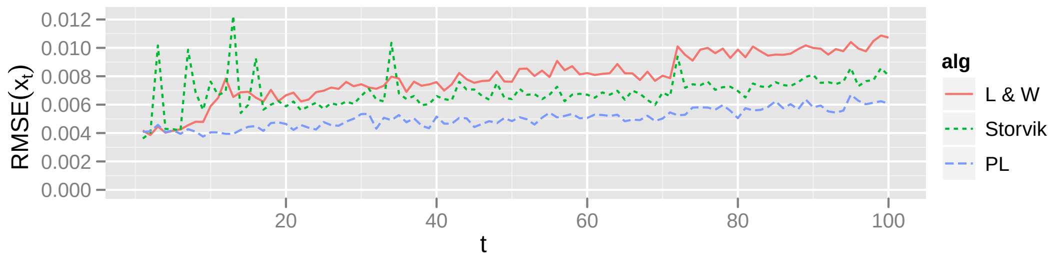
Figure 1.14: Linear Model 2: Filtering ESS(x_t). Here, only Storvik has a difficult time maintaining high effective sample size.



(a) Low SNR: Filtering RMSE(x_t)

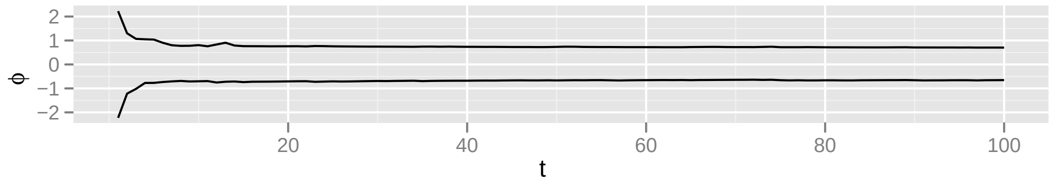


(b) Medium SNR: Filtering RMSE(x_t)

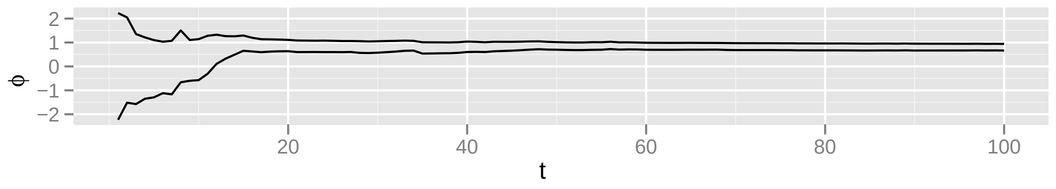


(c) High SNR: Filtering RMSE(x_t)

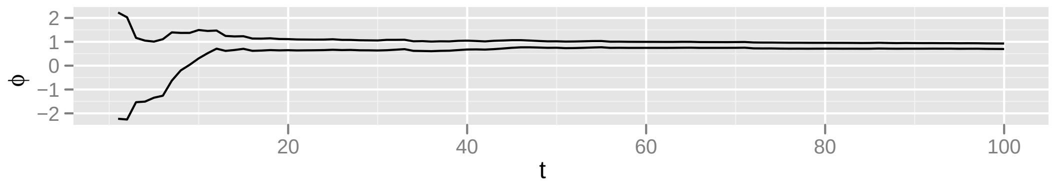
Figure 1.15: Linear Model 2: Filtering RMSE(x_t). In general, RMSE is large at the beginning of the series when the interval bands are widest. However, performance is not always the same from one time step to the next. The High SNR case shows the effects of the slight degradation of the sufficient statistics.



(a) Low SNR: Filtering 95% CI ϕ

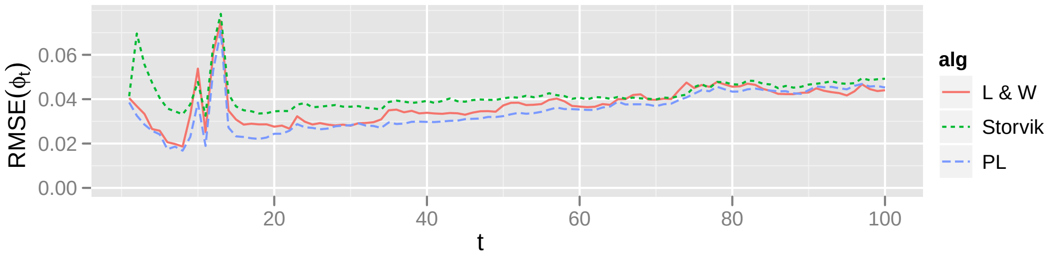


(b) Medium SNR: Filtering 95% CI ϕ

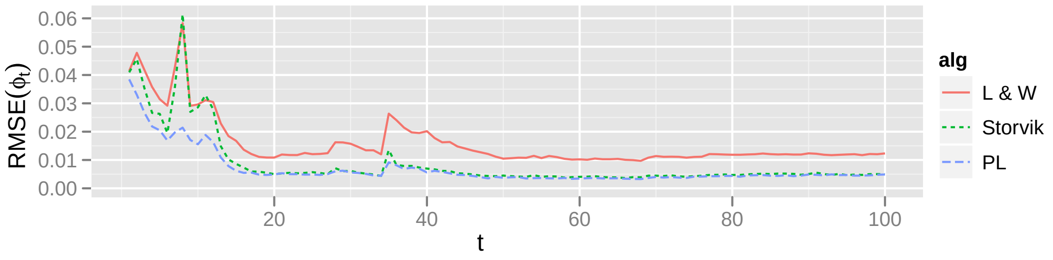


(c) High SNR: Filtering 95% CI ϕ

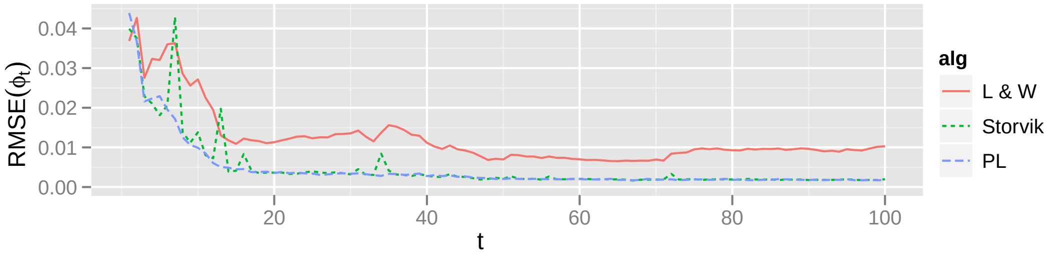
Figure 1.16: Linear Model 2: ϕ Posteriors. For values of $\phi \in (-1, 1)$, the time series is stationary and for the Low SNR case, this is all the filtering posteriors for ϕ can infer. In the Medium and High SNR cases, the model is quickly able to accurately estimate the simulated value of ϕ.



(a) Low SNR: RMSE(ϕ_t)

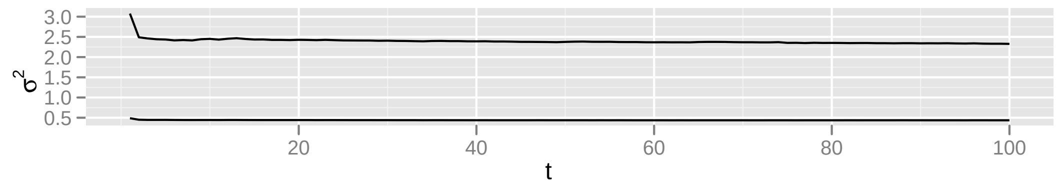


(b) Medium SNR: RMSE(ϕ_t)

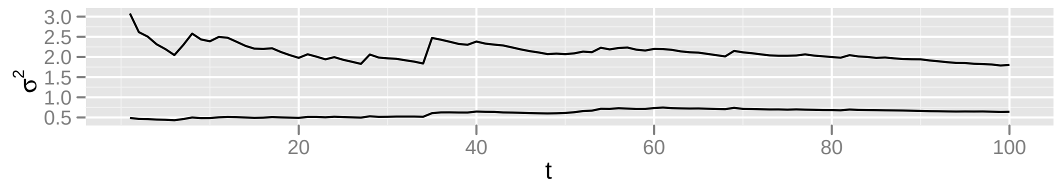


(c) High SNR: RMSE(ϕ_t)

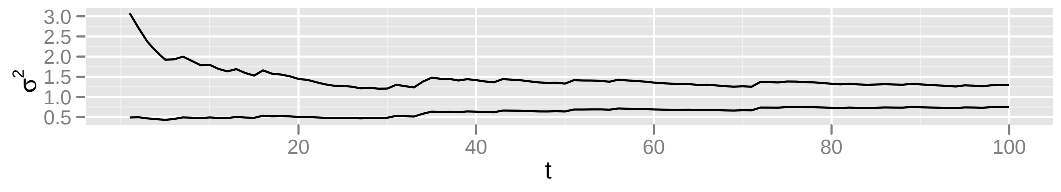
Figure 1.17: Linear Model 2: ϕ RMSE. As seen in Figure 1.16, the rate of parameter learning drops noticeably after $t = 20$ and so the RMSE tends to level out, with Storvik matching the performance of PL. In the Low SNR case, all that was learned is that this is a stationary model such that $\phi \in (-1, 1)$, and so the different algorithms perform similarly.



(a) Low SNR: Filtering 95% CI σ^2

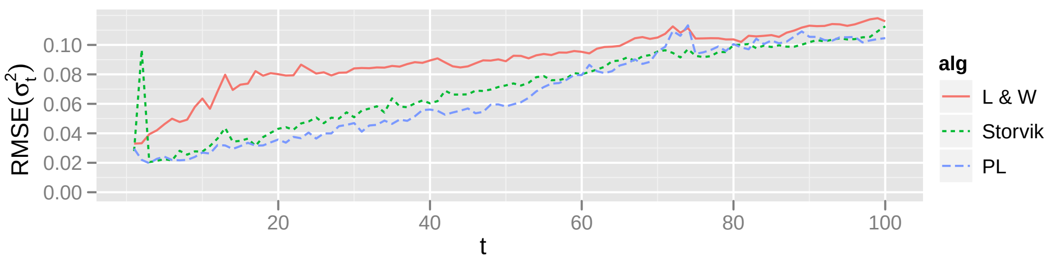


(b) Medium SNR: Filtering 95% CI σ^2

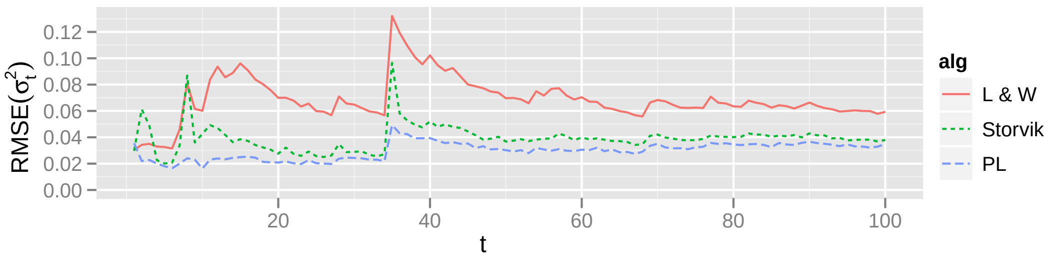


(c) High SNR: Filtering 95% CI σ^2

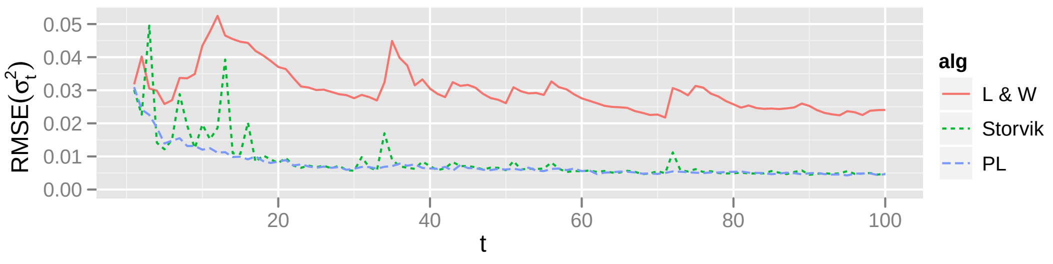
Figure 1.18: Linear Model 2: σ^2 Posteriors. Here, all models are slowly estimating σ^2 at a rate associated with how informative the observations are. Since ϕ in the Low SNR case is so difficult to estimate, precise identifying σ^2 is a difficult task.



(a) Low SNR: RMSE(σ_t^2)

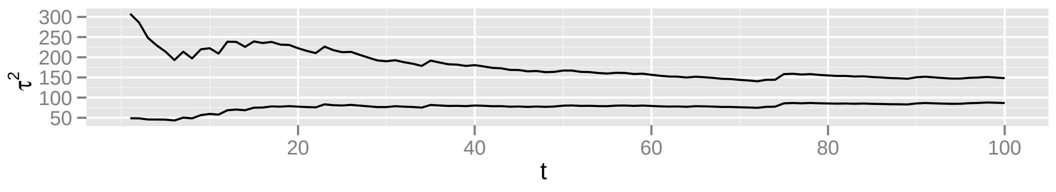


(b) Medium SNR: RMSE(σ_t^2)

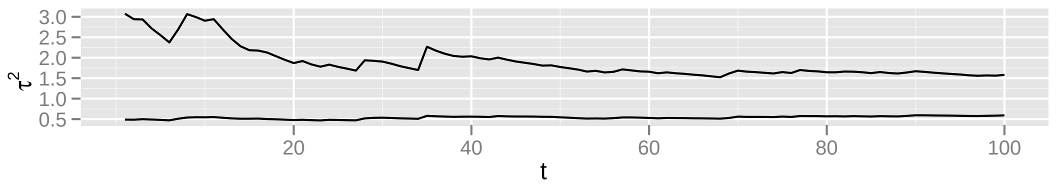


(c) High SNR: RMSE(σ_t^2)

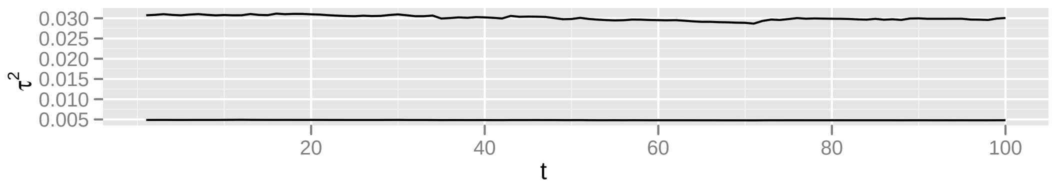
Figure 1.19: Linear Model 2: σ^2 RMSE. Once the static parameter posteriors stop changing dramatically after each observation, Storvik and PL perform nearly the same. The Liu & West implementation has different performance characteristics because it does not rely on a sufficient statistics structure. The slight upward trend in RMSE values for the low SNR case can be attributed to the degradation of the sufficient statistics or issues related to the choice of discount factor.



(a) Low SNR: Filtering 95% CI τ^2

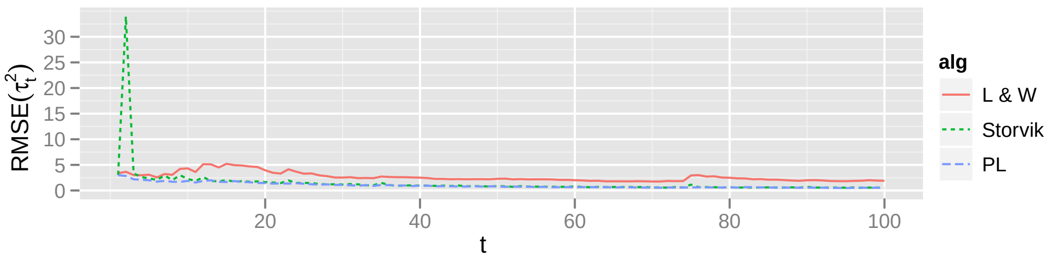


(b) Medium SNR: Filtering 95% CI τ^2

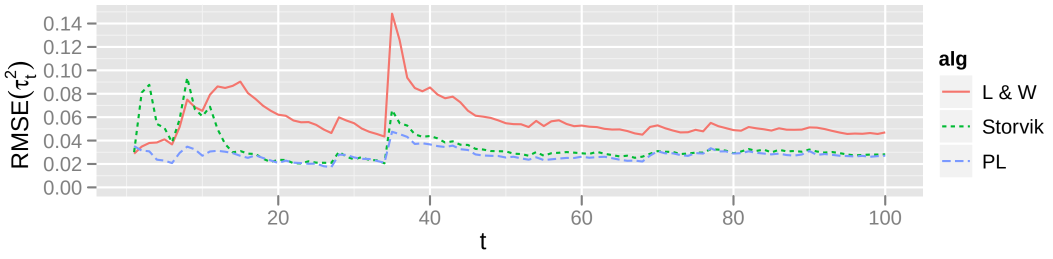


(c) High SNR: Filtering 95% CI τ^2

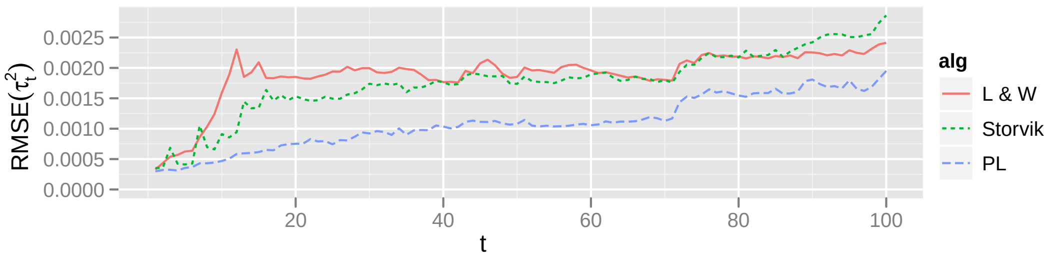
Figure 1.20: Linear Model 2: τ^2 Posteriors. Here is the only static parameter for the Low SNR case that shows noticeable narrowing of the interval bands, suggesting that the other static parameters are not going to show much improvement until a more precise estimate for observation error is obtained. This is in contrast to the High SNR case, where the interval bands are already so narrow that the other static parameters need to be better estimated before any significant improvements in estimating the observation error are possible.



(a) Low SNR: RMSE(τ_t^2)

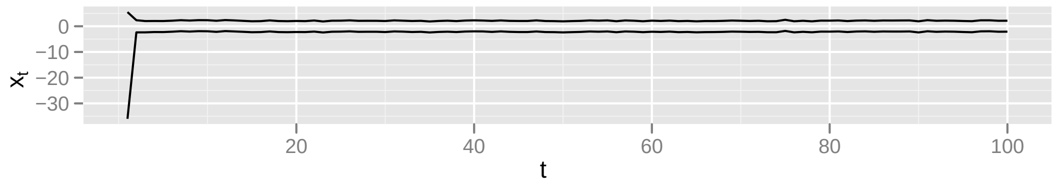


(b) Medium SNR: RMSE(τ_t^2)

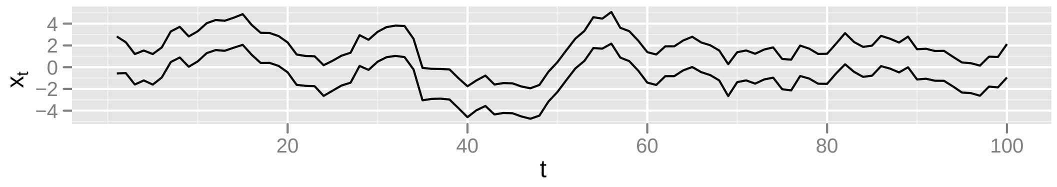


(c) High SNR: RMSE(τ_t^2)

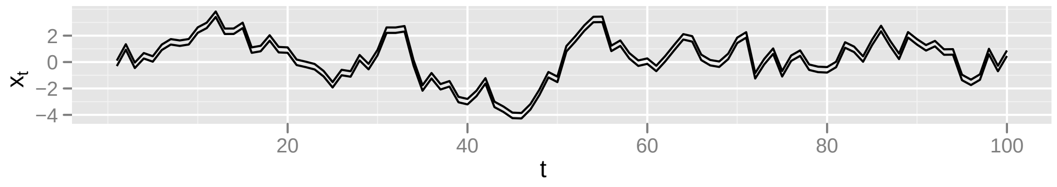
Figure 1.21: Linear Model 2: τ^2 RMSE. The Storvik implementation suffers from a very large spike in RMSE at the start of the series in the Low SNR case. In the Medium SNR case, there is a sharp increase in RMSE near step 35 for all algorithms, with the Liu & West implementation suffering the most. Again, in the High SNR case, the slight upward trend in RMSE values can be attributed to the degradation of the sufficient statistics or issues related to the choice of discount factor. In all cases, Particle Learning provides the best performance.



(a) Low SNR: Smoothing 95% CI x

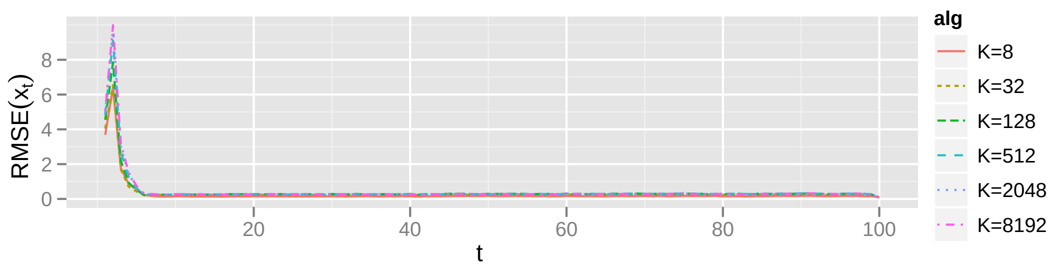


(b) Medium SNR: Smoothing 95% CI x

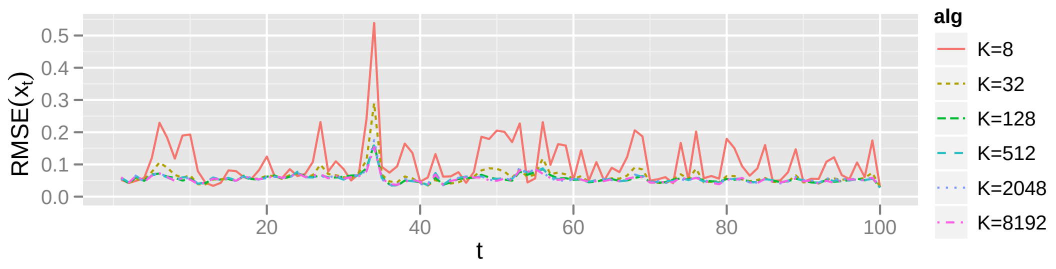


(c) High SNR: Smoothing 95% CI x

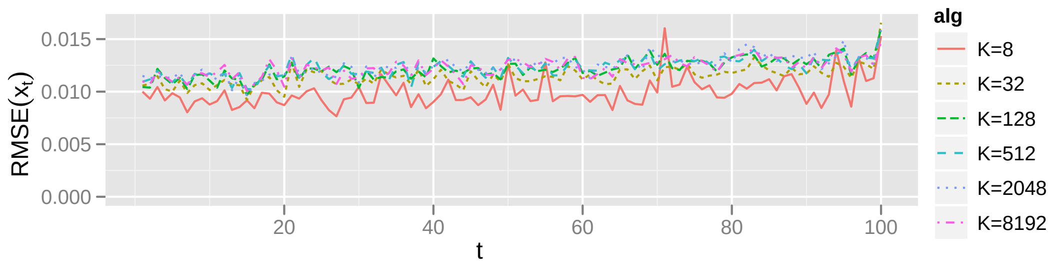
Figure 1.22: Linear Model 2: Smoothing Posteriors. As was seen for the filtering intervals, the widths of the interval bands are relatively consistent throughout, except for the beginning of the Low SNR case, and has important implications for calculating RMSE for the smoothed hidden states.



(a) Low SNR: Smoothing RMSE(x_t)

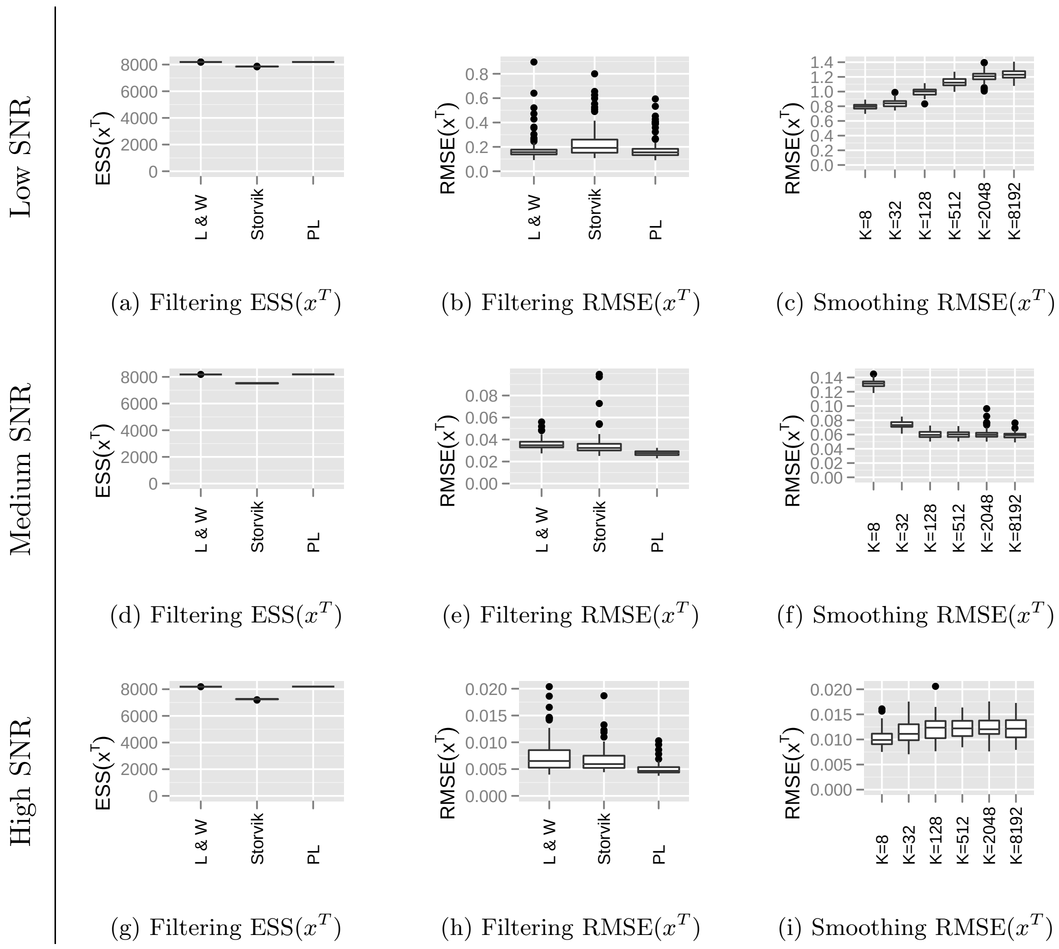


(b) Medium SNR: Smoothing RMSE(x_t)



(c) High SNR: Smoothing RMSE(x_t)

Figure 1.23: Linear Model 2: Smoothing RMSE(x_t). Again, in the Low SNR case it is hard to differentiate the different algorithms because of the extremely high RMSE at the beginning of the series. For the High SNR case, the slight growth in RMSE attributable to the degradation of the static parameters as was seen in the filtering plots is again evident here in the smoothing plots. With respect to the value of K for Boost Smoothing, values as low as $K = 128$ yield good results in all cases. Even so, $K = 8$ does a remarkable job in the High SNR case for nearly all time steps.



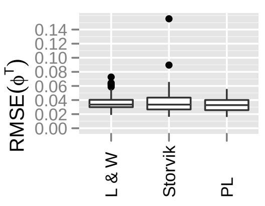
(a) Filtering ESS(x^T) (b) Filtering RMSE(x^T) (c) Smoothing RMSE(x^T)

(d) Filtering ESS(x^T) (e) Filtering RMSE(x^T) (f) Smoothing RMSE(x^T)

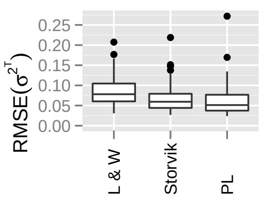
(g) Filtering ESS(x^T) (h) Filtering RMSE(x^T) (i) Smoothing RMSE(x^T)

Figure 1.24: Linear Model 2: Boxplots For ESS, Filtering, And Smoothing. Average ESS values are not significantly different across different algorithms. Filtering RMSE performance is also largely equal across the different static parameter estimation implementations, suggesting that the choice of auxiliary function and forward propagation function are more important. Smoothing RMSE results are counter-intuitive in that smaller values of K give better performance in the Low and High SNR cases.

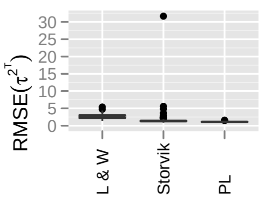
Low SNR



(a) RMSE(ϕ^T)

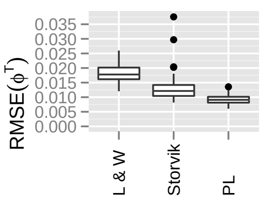


(b) RMSE(σ^{2^T})

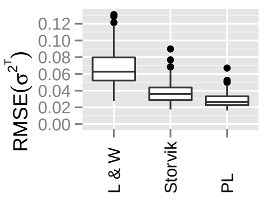


(c) RMSE(τ^{2^T})

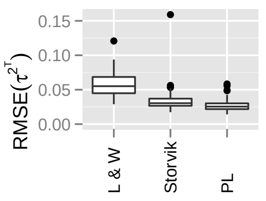
Medium SNR



(d) RMSE(ϕ^T)

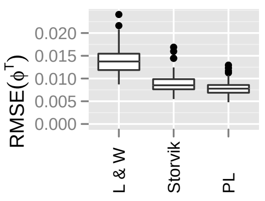


(e) RMSE(σ^{2^T})

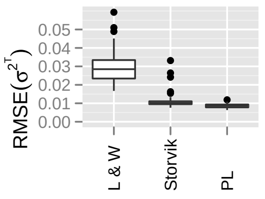


(f) RMSE(τ^{2^T})

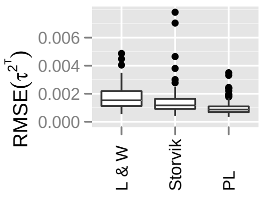
High SNR



(g) RMSE(ϕ^T)



(h) RMSE(σ^{2^T})



(i) RMSE(τ^{2^T})

Figure 1.25: Linear Model 2: Boxplots For Static Parameters. In the Low SNR case, there is not enough parameter learning to easily distinguish the performance of the different algorithms. In the Medium and High SNR cases, the sufficient statistics implementations outperform Liu & West. Since PL gives the best results, this is the forward filter implementation used when evaluating Boost Smoothing with different values of K.

Algorithm		**Run Time (Seconds)**
Liu & West	(L & W)	7.3
Storvik		6.9
Particle Learning	(PL)	4.5
Boost Smoothing	K=8	554.7
Boost Smoothing	K=32	268.8
Boost Smoothing	K=128	229.1
Boost Smoothing	K=512	370.7
Boost Smoothing	K=2048	1034.8
GDW04	N=8192	3950.5

Table 1.4: Linear Model 2: Average Run Time. The forward filtering algorithms are $O(TN)$ and hence are exceptionally fast compared to the backward smoothing step. It can also be seen that using $K << N$ can produce a significant performance increase, allowing for particle smoothing applications with many more particles. The reason that $K = 8$ or 32 produced slower run times then $K = 128$ concerns the overhead of using a scripting language like R for non vectorized code, where the function overhead of processing more groups M prevents the computational savings of using a smaller K to be realized.

1.4 Example: Nonlinear

This is the famous nonlinear example used to introduce the Bootstrap and SIR filters in 1993 [16]. The first difficulty is that the observations are squared, removing any directly observable information about the sign of the hidden state. Instead, the sign is inferred by the periodic influence of the trigonometric component of the system equation. If this component is removed, the posteriors become symmetric about the origin. As it is, the posterior typically has up to 3 modes which are frequently disjoint for all practical purposes. The state space model is presented below.

$$\begin{aligned} p(x_{t+1}|x_t) &= N\left(x_{t+1}\middle|\frac{x_t}{2} + \frac{25x_t}{1+x_t^2} + 8\cos(1.2(t+1)), \sigma^2\right) \\ p(y_t|x_t) &= N\left(y_t\middle|\frac{x_t^2}{20}, \tau^2\right) \end{aligned}$$

For MCMC algorithms, this makes finding satisfactory mixing proposal distributions extremely difficult. The Gibbs sampler will likely not mix properly between the different modes of the posterior because of the low probability in transitioning a sequence of states across signs when sampling one state at a time, even if the model admits conditional Gibbs steps for $p(x_t|x_{t-1}, x_{t+1}, \theta, y^T)$. Joint Gibbs samplers can be used, but without an obvious direct sampling scheme like FFBS, a Metropolis-Hastings step would need to be used which again is not likely to mix between modes very well. The difficulty is great enough that Particle Markov Chain Monte Carlo (PMCMC) [1] has been proposed as a way of using a SMC proposal to generate direct samples of $p(x^T|\theta, y^T)$

at each iteration to facilitate a better mixing MCMC implementation. However, running a new SMC at each MCMC iteration adds a substantial computation cost to the algorithm.

For an APF implementation, the first stage weights use a numerical estimate of $p(y_{t+1}|x_t)$, specified by using $\tilde{x}_{t+1}^{(k,i)}$ for $k = 1 : N_K$, generated from the inverse CDF of $p(x_{t+1}|x_t^{(i)})$ over a grid of probability values. These samples are used to approximate the optimal first stage weight distribution, $p(y_{t+1}|x_t) = \int p(y_{t+1}|x_{t+1})p(x_{t+1}|x_t)\mathrm{d}x_{t+1}$, with N_K chosen large enough that the approximation error is negligible with respect to the performance of the algorithm. While this is not a computationally elegant solution, the approximation will show what possible advantages APF provides for nonlinear models. The system equation is used for the forward propagation distribution in both algorithms.

$$v_t(y_{t+1}, x_t^{(i)}) = \frac{1}{N_k}\sum_{k=1}^{N_k} p\left(y_{t+1}\middle|\tilde{x}_{t+1}^{(k,i)}\right) \quad \text{where} \quad \tilde{x}_{t+1}^{(k,i)} = P^{-1}_{x_{t+1}|x_t^{(i)}}(k/N_k - 1/2N_k)$$

For comparison, a data set of length $T = 100$ was generated and 95% credible intervals where constructed to generate RMSE plots by equations (1.3.3) and (1.3.4) using the results of 100 replications for each algorithm. As before, posteriors and RMSE performance of the Bootstrap filter, APF, and Boost Smoothing (using SIR) with fixed parameters are presented in Figures 1.26, 1.27, and 1.28, with computation times in Table 1.5. For the auxiliary function, $N_K = 250$ was chosen.

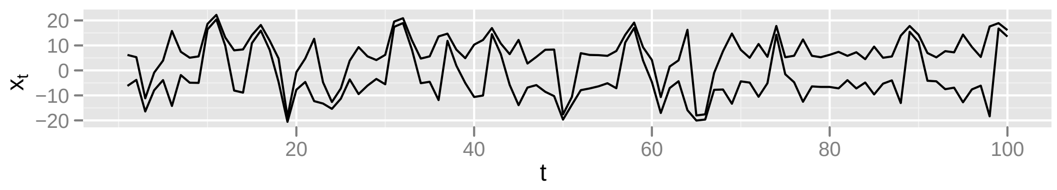
The changing width of the credible intervals in the filtering and smoothing posteriors is primarily a reflection of the models ability to predict the sign of the hidden

Algorithm		**Run Time (Seconds)**
Bootstrap Filter	(Boot)	1.8
Auxiliary Particle Filter	(APF)	174.6
Boost Smoothing	K=8	161.5
Boost Smoothing	K=32	139.4
Boost Smoothing	K=128	162.3
Boost Smoothing	K=512	297.5
Boost Smoothing	K=2048	878.3
GDW04	N=8192	3370.4

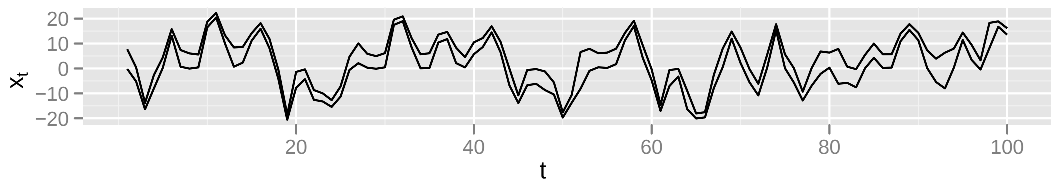
Table 1.5: Nonlinear Model: Average Run Time. Note the substantial improvement in the run time of Boost Smoothing over GDW04 afforded by choosing $K << N$.

state. Only when the intervals narrow and are not centered about the origin does the model has significant information about the sign of the hidden state. Using future observations also allows the smoothing distribution to more accurately predict the hidden state, producing much narrower credible intervals.

The ESS plots show only slightly improved results for APF, but with larger RMSE spikes then the Bootstrap filter. Performance is also highly erratic for both algorithms, having highly oscillating ESS values ranging from close to 0% to near 100%. Here, the use of APF actually decreases RMSE performance despite using a near ideal auxiliary function and a 100x increase in computation time. For Boost Smoothing, the minimum reasonable value of K seems to be about 128, which affords an observed 20x performance increase over GDW04 smoothing. Admittedly these posteriors are often multi-modal, and symmetric 95% credible intervals may be ideal, but it can be seen that the smoothing distribution can often resolve the sign of the hidden state.

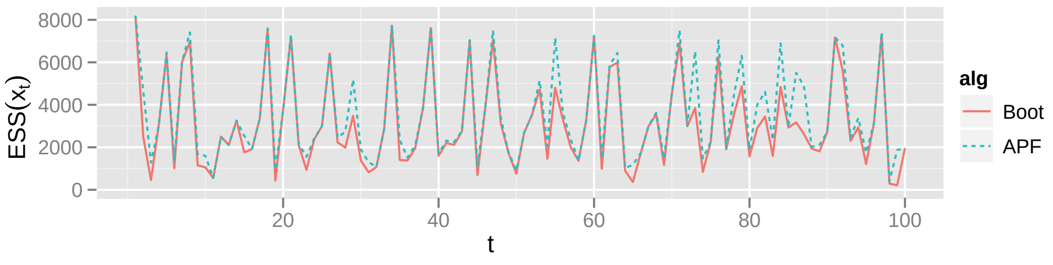


(a) Filtering 95% CI x

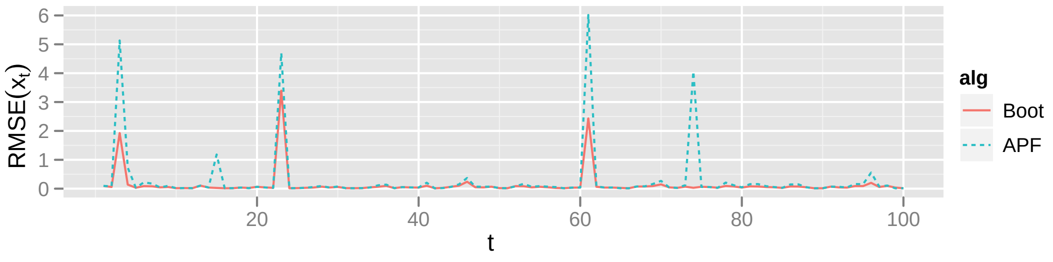


(b) Smoothing 95% CI x

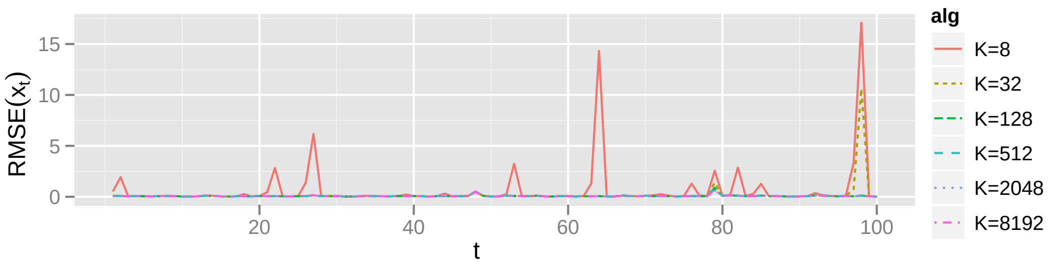
Figure 1.26: Nonlinear Model: Filtering And Smoothing Posteriors. In this case, sometimes the symmetric 95% credible intervals narrow significantly for the smoothing distributions. This is usually because the model is able to predict the sign of the hidden state. For those posterior intervals that appear relatively wide and approximately centered about the origin, the model is not able to infer the sign of the hidden state.



(a) Filtering ESS(x_t)

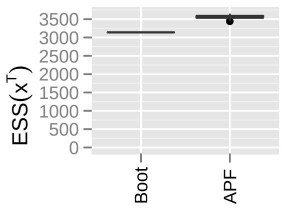


(b) Filtering RMSE(x_t)

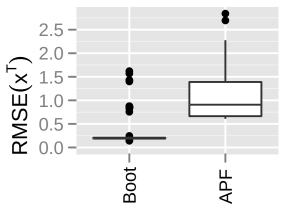


(c) Smoothing RMSE(x_t)

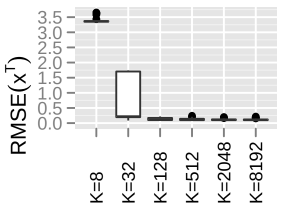
Figure 1.27: Nonlinear Model: Filtering And Smoothing, ESS(x_t) And RMSE(x_t). While the two filtering algorithms perform near identically for most time steps, spikes in RMSE occur when the new observation is particularly informative. The ESS plot show the limited value of the auxiliary function when the optimal proposal distribution is not available. The smoothing RMSE plot suggests that while usually $K = 8$ or 32 is acceptable, larger values of K will be necessary in time steps where the filtering distribution makes for a poor importance distribution for the smoothing distribution.



(a) Filtering ESS(x^T)



(b) Filtering RMSE(x^T)



(c) Smoothing RMSE(x^T)

Figure 1.28: Nonlinear Model: Boxplots For ESS, Filtering, And Smoothing. The improvement in ESS values of the APF is negligible compared to the Bootstrap filter. When considering the filtering RMSE, the more widely dispersed particles of the Bootstrap filter give better performance when estimating the 95% credible intervals. For Boost Smoothing, it appears that a value of K between 128 and 512 gives acceptable amounts of MC error at substantial computational savings.

Chapter 2

Markov Random Fields

Markov Random Fields (MRF) are models for describing spatial structures using arbitrary conditional distributions based on a locally defined neighborhood structure. The first commonly used MRFs used Gaussian distributions for their analytic tractability and where known as Gaussian Markov Random Fields (GMRF). These where then used to model spatial lattice data and became known as Conditional Auto Regressive (CAR) models because of their similarity to autoregressive time series models, where the mean of the conditional distribution is a linear combination of its neighbors. [3] A graph of this dependence structure for a regular lattice model can be found in Figure 2.1, where each node corresponds to a random variable conditional on its neighbors, denoted by the links that connects the different nodes. In practice, MRFs are often used as part of a hierarchical model for the spatial random effects of observations. One example is modeling noisy images by describing the distribution of a hidden pixel conditioned on its neighbors in order to define a global distribution on images.

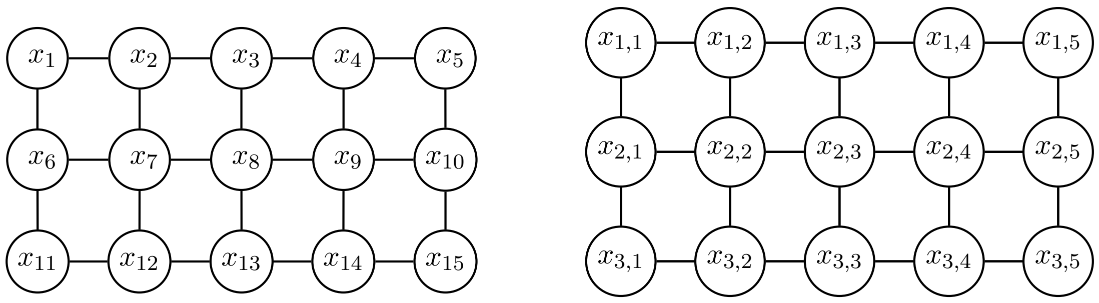


Figure 2.1: Nearest 4 Neighborhood Lattice Model Indexing, x_t and x_{ij}

Irregular lattice models can arise in spatial data when looking at states, counties, or other geographic regions. More details can be found in [2, 4, 19].

While a wide variety of MRFs with irregular neighborhood structures can be defined, this section will concern a simple nearest 4 neighbor GMRF on a regular $(I \times J)$ lattice. This will provide for a numeric comparison of a large variety of MCMC and SMC methods, but this does not imply that these methods are restricted to this simple model. The primary exception is that the Joint Gibbs sampler is typically not available for non GMRFs, although it will work for arbitrary neighborhood structures. An alternative is to use SMC for the proposal distribution to sample jointly from $p(x^T|\theta, y^T)$, and is known as Particle Markov Chain Monte Carlo (PMCMC). [1] The model is thus defined, where ∂ij is the set of 2-4 immediate adjacent neighbors at location ij, $\bar{x}_{\partial ij}$ is the mean of these neighbors, $N_{\partial ij}$ is the number of neighbors at location ij, and $ij \sim i'j'$ denotes a neighbor relation.

$$y_{ij}|x_{ij},\tau^2 \sim N(x_{ij},\tau^2)$$
$$x_{ij}|x_{\partial ij},\sigma^2 \sim N\left(\bar{x}_{\partial ij},\sigma^2/N_{\partial ij}\right)$$

This defines an improper multivariate prior on $p(X|\sigma^2)$ known as an Intrinsic Gaussian Markov Random Field (IGMRF), with joint densities defined as follows,

$$\begin{aligned} Y|X,\tau^2 &\sim N(X,\tau^2\mathbb{I}_{IJ}) \\ p(X|\sigma^2) &= (2\pi)^{\frac{-(IJ-1)}{2}}\left|\frac{\mathbb{P}_{MRF}}{\sigma^2}\right|^{*1/2}\exp\left(-\frac{1}{2}X'\frac{\mathbb{P}_{MRF}}{\sigma^2}X\right) \\ &= (2\pi\sigma^2)^{\frac{-(IJ-1)}{2}}\exp\left(-\frac{1}{2\sigma^2}\sum_{ij\sim i'j'}(x_{ij}-x_{i'j'})^2\right) \end{aligned}$$

where $|A|^*$ is the product of non zero eigenvalues of A, known as the generalized determinate, and $\mathbb{P}_{MRF} = \mathbb{T}_I \otimes \mathbb{I}_J + \mathbb{I}_I \otimes \mathbb{T}_J$, where

$$\mathbb{T}_n = \mathbb{H}_n'\mathbb{H}_n = \begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ -1 & 2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & -1 \\ 0 & \dots & 0 & -1 & 1 \end{bmatrix}_{n\times n} \qquad \mathbb{H}_n = \begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & \ddots & \ddots & \vdots \\ \dots & \ddots & \ddots & -1 & 0 \\ 0 & \dots & 0 & 1 & -1 \end{bmatrix}_{(n-1)\times n} \tag{2.0.1}$$

with priors $p(\sigma^2) \sim IG(\sigma_a^2,\sigma_b^2)$ and $p(\tau^2) \sim IG(\tau_a^2,\tau_b^2)$ yielding the following conditionals.

$$X|\sigma^2, \tau^2, Y \sim N((\mathbb{P}_{MRF}/\sigma^2 + \mathbb{I}_{IJ}/\tau^2)^{-1}Y/\tau^2, (\mathbb{P}_{MRF}/\sigma^2 + \mathbb{I}_{IJ}/\tau^2)^{-1}) \quad (2.0.2)$$

$$x_{ij}|x_{\partial ij}, \sigma^2, \tau^2, y_{ij} \sim N\left(\frac{y_{ij}/\tau^2 + \bar{x}_{\partial ij}N_{\partial ij}/\sigma^2}{1/\tau^2 + N_{\partial ij}/\sigma^2}, \frac{1}{1/\tau^2 + N_{\partial ij}/\sigma^2}\right) \quad (2.0.3)$$

$$\sigma^2|X, \sigma^2, \tau^2 \sim IG\left(\sigma_a^2 + (IJ-1)/2, \sigma_b^2 + X'\mathbb{P}_{MRF}X/2\right) \quad (2.0.4)$$

$$\tau^2|X, \sigma^2, \tau^2 \sim IG\left(\tau_a^2 + IJ/2, \tau_b^2 + (Y-X)'(Y-X)/2\right) \quad (2.0.5)$$

Note that the precision matrix, $\mathbb{P}_{MRF}$, is a singular matrix with one zero eigenvalue, which implies an improper prior. Adding a constraint that centers the prior, such as $\sum x_{ij} = 0$, will define a proper distribution. Care must be taken when using constraints, especially in implementations where the x_{ij}s are updated individually, to ensure that the constraint holds. However, in this model no constraint or proper prior is need on the $\sum x_{ij}$ since $\mathbb{P}_{MRF}/\sigma^2 + \mathbb{I}_{IJ}/\tau^2$ is of full rank.

The rest of this section discusses different strategies for performing Bayesian inference on this model. First, existing MCMC algorithms are reviewed, followed by an introduction to SMC algorithms for MRFs that have been converted to Sequential MRF models by inducing an ordering on the nodes. Of utmost importance is exploitation of the sparsity pattern of the precision matrix. The different methods are evaluated using RMSE calculations of 95% credible intervals for the states and static parameters under 3 different signal to noise ratios. An implementation comparison is given in Table 2.1.

Algorithm	**Time Complexity**	**Caveats / Notes**
Conditional Gibbs	$\mathcal{O}(NIJ)$	mixing is a concern
Joint Gibbs	$\mathcal{O}(NIJ^2 + NIJ^3)$	$\mathcal{O}(NIJ^2)$ for fixed σ^2, τ^2, but only works for GMRF
FFBS Gibbs	$\mathcal{O}(NIJ^3)$	more complex code then sparse matrix operations
Multivariate SMRF	$\mathcal{O}(NIJ^3)$	multivariate sampling can be difficult if not direct
M SMRF Boost Smoothing	$\mathcal{O}(KNIJ)$	multivariate proposals can be difficult if optimal proposal not available
Univariate SMRF	$\mathcal{O}(NIJ)$	poor estimation of vertical neighbor
U SMRF Boost Smoothing	$\mathcal{O}(KNIJ^2)$	large constant factor

Table 2.1: MRF: Method Comparison. Comparison of different MRF estimation strategies of a nearest 4 neighbor GMRF on an $I \times J$ regular lattice where N = number particles or iterations.

2.1 MCMC Methods For MRFs

This section explains existing algorithms, and introduces the motivating algorithm behind Sequential MRF implementations. The differences lie in how the posterior distribution associated with the MRF is sampled, the conditional distributions for the static parameters being otherwise the same. The MCMC samplers will all take the form seen in Algorithm 9, and for more details see [30].

```
for $n = 1$ to $N$ do
  Sample: $X|\sigma^2, \tau^2, Y$ using any of the methods described in Sections 2.1.1 - 2.1.3.
  Sample: $\sigma^2|X, \tau^2, Y$ using equation (2.0.4)
  Sample: $\tau^2|X, \sigma^2, Y$ using equation (2.0.5)
```

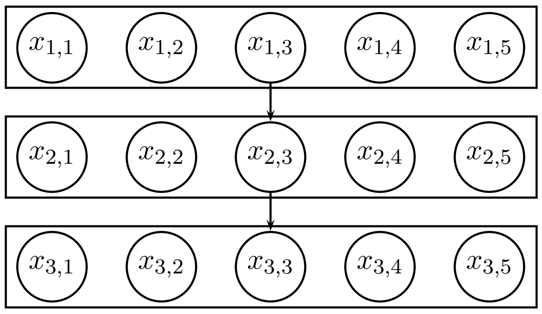
Algorithm 9: Generic Gibbs

2.1.1 Conditional Gibbs Sampling

A common implementation strategy utilizes the conditional distribution of $p(x_{ij}|x_{\partial ij}, y_{ij})$ to specify a Gibbs sampler. The primary concern with this approach is poor mixing because conditioning on the neighbors prevents x_{ij} from making very large jumps at each iteration. This issue becomes particularly acute in low signal to noise ratio situations.

```
for $n = 1$ to $N$ do
  for $i = 1$ to $I$ do
    for $j = 1$ to $J$ do
      Sample: $x_{ij}|x_{\partial ij}, \sigma^2, \tau^2$ using equation (2.0.3)
  Sample: $\sigma^2|X, \tau^2, Y$ using equation (2.0.4)
  Sample: $\tau^2|X, \sigma^2, Y$ using equation (2.0.5)
```

Algorithm 10: Conditional Gibbs


$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$
(a) Multivariate SMRF Model

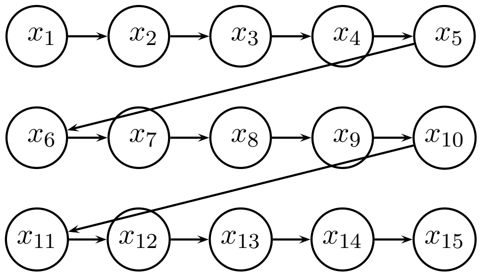

x_1 x_2 x_3 x_4 x_5
x_6 x_7 x_8 x_9 x_{10}
x_{11} x_{12} x_{13} x_{14} x_{15}
(b) Univariate SMRF Model

Figure 2.2: State Space Model Indexing Scheme For Sequential MRF

2.1.2 Joint Gibbs Sampling

Multivariate normal distributions can be sampled directly by first performing Cholesky decomposition on the covariance matrix such that $\Sigma = LL'$, and then generating a sample by $X = \mu + Lz$ where $z_i \sim N(0,1)$. However, by performing decomposition on the (sparse) precision matrix, and then back-solving the resulting triangular matrix, only the inverse of a triangular matrix is needed. In this case, the method is modified such that $\sigma^{-2}\mathbb{P}_{MRF} = \Sigma^{-1} = LL'$ and $L(X - \mu) = z$. This is a sparse matrix explanation of the ideas originally found in [31, 32].

```
for $n = 1$ to $N$ do
    Sample: $X|\sigma^2, \tau^2$ directly using the mean and precision matrix from equation (2.0.2)
    Sample: $\sigma^2|X, \tau^2, Y$ using equation (2.0.4)
    Sample: $\tau^2|X, \sigma^2, Y$ using equation (2.0.5)
```

Algorithm 11: Joint Gibbs

2.1.3 Multivariate DLM

It was noted in [21] that MRFs could be transformed into a special kind of DLM with pseudo observations by treating each row of the MRF as the hidden state of a multivariate time series model with rows indexed by time. The between row correlation is accounted for with the system equation, with the pseudo observations accounting for the within row correlation. A state space model for the method is presented below, with $X_i = [x_{i1}, x_{i2}, \ \dots\ , x_{iJ}]'$, pseudo observations $\mathbb{Z}_i$ is a vector of zeros of length $J-1$, and $\mathbb{H}_J$ is the $(J-1)\times J$ difference matrix previously described in equation 2.0.1.

$$\left.\begin{bmatrix} Y_i \\ \\ \mathbb{Z}_i \end{bmatrix}\right| X_i, \sigma^2, \tau^2 \sim N_{2J-1}\left(\begin{bmatrix} \mathbb{I}_J \\ \\ \mathbb{H}_J \end{bmatrix} X_i, \begin{bmatrix} \tau^2\mathbb{I}_J & 0 \\ \\ 0 & \sigma^2\mathbb{I}_{J-1} \end{bmatrix} \right) \qquad \text{(Observation/Pseudo)}$$

$$X_i | X_{i-1}, \sigma^2 \sim N_J(X_{i-1}, \sigma^2\mathbb{I}_J) \qquad \text{(System)}$$

The joint posterior for $p(X^I|\mathbb{Z}^I, Y^I)$ with $p(X_1) \propto 1$ is multivariate normal with precision matrix shown below.

$$\mathbb{P}_{MRF}/\sigma^2 + \mathbb{I}_{IJ}/\tau^2 = \overbrace{\mathbb{T}_I \otimes \mathbb{I}_J/\sigma^2}^{System} + \overbrace{\mathbb{I}_I \otimes \mathbb{T}_J/\sigma^2}^{Pseudo} + \overbrace{\mathbb{I}_I \otimes \mathbb{I}_J/\tau^2}^{Observation}$$

The states can then be sampled directly using FFBS.

```
for $n = 1$ to $N$ do
    Sample: $X|\sigma^2, \tau^2$ with $p(X_1) \propto 1$ and observations $Y_t^* = \begin{bmatrix} Y_t' & \mathbb{Z}_t' \end{bmatrix}'$ using FFBS,
        $\left\{ F_t = \begin{bmatrix} \mathbb{I}_J \\ \mathbb{H}_J \end{bmatrix}, G_t = \mathbb{I}_J, V_t = \begin{bmatrix} \tau^2\mathbb{I}_J & 0 \\ 0 & \sigma^2\mathbb{I}_{J-1} \end{bmatrix}, W_t = \sigma^2\mathbb{I}_J \right\}$
    initiating the forward filtering with
    $p(X_1|Y, \mathbb{Z}, \sigma^2, \tau^2) = N\left(m_1 = (\sigma^{-2}\mathbb{T}_J + \tau^{-2}\mathbb{I}_J)^{-1}X_1/\tau^2, C_1 = (\sigma^{-2}\mathbb{T}_J + \tau^{-2}\mathbb{I}_J)^{-1}\right)$
    Sample: $\sigma^2|X, \tau^2, Y$ using equation (2.0.4)
    Sample: $\tau^2|X, \sigma^2, Y$ using equation (2.0.5)
```

Algorithm 12: DLM Gibbs

2.2 SMC Methods For Sequential MRFs (SMRF)

These methods rely on a sequential interpretation of MRFs by defining a state space model such that the joint distribution is equivalent to the posterior distribution of the MRF. This class of state space models will be called Sequential Markov Random Fields (SMRF).

2.2.1 Multivariate SMRF

The multivariate SMRF model uses the same state space model as the Multivariate DLM from Section 2.1.3. A Particle Learning implementation is described next.

$$\begin{bmatrix} Y_{t+1} \\ \\ \mathbb{Z}_{t+1} \end{bmatrix} \Bigg| X_t, \sigma^2, \tau^2 \sim N\left(\begin{bmatrix} \mathbb{I}_J \\ \\ \mathbb{H}_J \end{bmatrix} X_t, \sigma^2 \begin{bmatrix} \mathbb{I}_J & \mathbb{H}'_J \\ \\ \mathbb{H}_J & \mathbb{T}_{J-1} \end{bmatrix} + \begin{bmatrix} \tau^2 \mathbb{I}_J & 0 \\ \\ 0 & \sigma^2 \mathbb{I}_{J-1} \end{bmatrix} \right)$$

$$X_{t+1} \Bigg| X_t, \begin{bmatrix} Y_{t+1} \\ \\ \mathbb{Z}_{t+1} \end{bmatrix}, \sigma^2, \tau^2 \sim N\left(\left(\frac{Y'_{t+1}}{\tau^2} + \frac{X'_t}{\sigma^2} \right) \left(\frac{\mathbb{I}}{\tau^2} + \frac{\mathbb{T}_{J-1} + \mathbb{I}_{J-1}}{\sigma^2} \right)^{-1}, \left(\frac{\mathbb{I}_J}{\tau^2} + \frac{\mathbb{T}_{J-1} + \mathbb{I}_{J-1}}{\sigma^2} \right)^{-1} \right)$$

$$\sigma^2 | X^{t+1} \sim IG\left(\sigma^2_{a|t+1} = \sigma^2_{a|t} + \frac{J}{2}, \sigma^2_{b|t+1} = \sigma^2_{b+1} + \begin{bmatrix} X_t \\ \\ X_{t+1} \end{bmatrix}' \begin{bmatrix} \mathbb{I}_J & -\mathbb{I}_J \\ \\ -\mathbb{I}_J & \mathbb{T}_J + \mathbb{I} \end{bmatrix} \begin{bmatrix} X_t \\ \\ X_{t+1} \end{bmatrix} \right)$$

$$\tau^2 | X^{t+1}, Y^{t+1} \sim IG\left(\tau^2_{a|t+1} = \tau^2_{a|t} + \frac{J}{2}, \tau^2_{b|t+1} = \tau^2_{b|t} + \frac{(Y_{t+1} - X_{t+1})'(Y_{t+1} - X_{t+1})}{2} \right)$$

$$S_t = \{\tau^2_{a|t}, \tau^2_{b|t}, \sigma^2_{a|t}, \sigma^2_{b|t}\}$$

In this case the static parameters have been derived sequentially and the prior is specified by S_0. The prior $p(X_1|\sigma^2)p(\sigma^2)$ is improper since $p(\sum X_{1j}) \propto 1$, but becomes proper after conditioning on the first row. This is the joint distribution of an univariate AR(1) model similar to the one in Section 1.3 but with fixed $\phi = 1$. A Particle Learning implementation yielding $p(\sigma^2, \tau^2 | y_{1,1:J})$ is used with FFBS to generate $p(x_{1,1:J}|\sigma^2, \tau^2, y_{1,1:J})$ to provide equally weighted samples for the first row, $p(X_1, \sigma^2, \tau^2 | Y_1)$.

While the filter smoothing approximation of $p(X^I, \sigma^2, \tau^2 | Y^I)$ is the most convenient, this method degenerates too rapidly to be used on larger MRFs. By considering this model as a multivariate state space model, Boost Smoothing can be be implemented with the following weights, using the system equation without the pseudo observations.

$$\begin{aligned} w^{(k)} &\propto p(\sigma^{2(l)}, \tau^{2(l)} | X^{i(k)}, Y^i) p(X_{i+1}^{(l)} | X_i^{(k)}, \sigma^{2(l)}, \tau^{2(l)}) \\ &\propto p(\sigma^{2(l)} | S_i^{(k)}) p(\tau^{2(l)} | S_t^{(k)}) \prod_{j=1}^{J} p(x_{i+1,j} | x_{i,j}, \sigma^2) \end{aligned}$$

where (k) indexes the previously sampled particles from $p(x_{i+1:I}, \sigma^2, \tau^2 | y^I)$, (l) index particles in the filtering distribution $p(x^i | y^i)$, i indexes either the rows of the MRF or the time component of the state space model, and j indexes either the columns of the MRF or indexes dimensions in a multivariate state space model.

The downside to this approach is that multivariate samples from the filtering distribution may not carry high weight under the smoothing distribution, necessitating a larger value of K for higher dimensions. The advantage is that the computational complexity is linear with the number of nodes in the MRF, $\mathcal{O}(KNIJ)$, making this algorithm computationally cheap compared to the multivariate SMRF forward filtering calculations.

2.2.2 Univariate SMRF

The intuition behind this scheme is to build the MRF node by node instead of row by row, and thus the MRF can be interpreted as a univariate time series model with no need to invert matrices. The main computational burden with this approach in the forward filtering pass is maintaining the J^{th} lag smoothing distribution for calculating the contribution from the vertical neighbor. The system equation is derived below by factorizing the prior $p(x^T|\sigma^2)$ as a product of improper bivariate densities representing each link in the conditional dependence graph.

In general, the prior is defined as

$$p(x^T, \sigma^2) = p(\sigma^2) \prod_{t \sim t'} f(x_t, x_{t'}|\sigma^2) = p(\sigma^2) \prod_{t=2}^{T} f(x_t, x_{\partial t<t}|\sigma^2)$$

with $x_{\partial t<t}$ denoting the neighbors of x_t whose index is less than t.

For the neighborhood structure considered here, let

$$f(x_t, x_{\partial t<t}|\sigma^2) = (2\pi\sigma^2)^{-1/2} f(x_t, x_{t-1}|\sigma^2) f(x_t, x_{t-J}|\sigma^2)$$

where

$$f(x_t, x_{t'}|\sigma^2) = \begin{cases} \exp\left(-(x_t - x_{t'})^2/2\sigma^2\right) & \text{if } t \sim t' \\ 1 & \text{otherwise} \end{cases}$$

The posterior is factored into the product of per node and per neighbor components,

$$p(x^T|\sigma^2) = (2\pi\sigma^2)^{\frac{-(IJ-1)}{2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{t\sim t'}(x_t - x_{t'})^2\right) = \prod_{t=2}^{T}(2\pi\sigma^2)^{-1/2} \prod_{t\sim t'} f(x_t, x_{t'}|\sigma^2)$$

While this does properly factor the MRF prior, it does not mean that the system equation $f(x_t, x_{\partial t<t}|\sigma^2)$ is proper and should not be viewed as $f(x_t|x_{\partial t<t}, \sigma^2)$. The more common state space model form is $p(x^T|\sigma^2) = \prod_{t=1}^{T} p(x_t|x^{t-1}, \sigma^2)$, but this

looses the sparsity advantages of the conditional independence structure. A Particle Learning implementation is described below.

$$
\begin{aligned}
&v_t(y_t, x_{\partial t<t}, \sigma^2, \tau^2, y_t) \\
&= \begin{cases}
\sqrt{2\pi(2\tau^2+\sigma^2)} \\
\quad \times \exp\left(-\frac{1}{2}\left(\frac{x_{t-1}^2}{\sigma^2} + \frac{x_{t-J}^2}{\sigma^2} + \frac{y_t^2}{\tau^2} - \frac{\left(\frac{x_{t-1}}{\sigma^2}+\frac{x_{t-J}}{\sigma^2}+\frac{y_t}{\tau^2}\right)^2}{2/\sigma^2+1/\tau^2}\right)\right) & \text{if } t\sim t-1 \ \& \\
& t\sim t-J \\
N(y_t|x_{t-1}, \sigma^2+\tau^2) & \text{if } t\sim t-1 \\
N(y_t|x_{t-J}, \sigma^2+\tau^2) & \text{if } t\sim t-J
\end{cases}
\end{aligned}
$$

$$
q_t(x_t|x_{\partial t<t}, \sigma^2, \tau^2, y_t) = \begin{cases}
N\left(x_t\middle|\frac{(x_{t-1}+x_{t-J})\sigma^{-2}+y_t\tau^{-2}}{2\sigma^{-2}+\tau^{-2}}, \frac{1}{2\sigma^{-2}+\tau^{-2}}\right) & \text{if } t\sim t-1 \ \& \\
& t\sim t-J \\
N\left(x_t\middle|\frac{x_{t-1}/\sigma^2+y_t/\tau^2}{\sigma^{-2}+\tau^{-2}}, \frac{1}{\sigma^{-2}+\tau^{-2}}\right) & \text{if } t\sim t-1 \\
N\left(x_t\middle|\frac{x_{t-J}/\sigma^2+y_t/\tau^2}{\sigma^{-2}+\tau^{-2}}, \frac{1}{\sigma^{-2}+\tau^{-2}}\right) & \text{if } t\sim t-J
\end{cases}
$$

$$
\begin{aligned}
\tau^2|x^t, \sigma^2, \tau^2 &\sim IG\left(\tau_a^2 + \frac{t}{2}, \tau_b^2 + \frac{1}{2}\sum_{k=1}^{t}(y_k - x_k)^2\right) \\
\sigma^2|x^t, \sigma^2, \tau^2 &\sim IG\left(\sigma_a^2 + \frac{t}{2}, \sigma_b^2 + \frac{1}{2}\sum_{k\sim k'}(x_k - x_{k'})^2\right)
\end{aligned}
$$

This model can be extended for a general neighborhood structure by modifying the system equation to incorporate all previous neighbors. The ordering of the nodes is important to reduce the bandwidth of the MRF conditional dependence adjacency matrix, which will limit the maximum lag of the smoothing distribution needed to calculate the dependence from the vertical neighbor.

Smoothing is more complicated for this SMRF model due to the conditional dependence structure arising from transforming a two dimensional structure to a one dimensional structure. The main point to understand is that since the forward filtering step essentially grows the MRF one node at a time, the underlying structure of the MRF prior changes at each time step. This is different then when considering the full MRF prior and only adding data y_t one at a time in the filtering step. Hence, the smoothing weights will depend on every neighborhood link that depends on x_t yet is not represented in $p(x^{t-1}|\theta) = p(\sigma^2)\prod_{k=2}^{t-1} f(x_k, x_{\partial k<k}|\sigma^2)$.

The derivation for the general case requires some new notation. Let ∂_t^+ be the set of indices in $t+1:T$ that have a link to a node in x^t such that $\partial_t^+ = \{\partial_t \cap t+1:T\}$. Then, for any backward smoothing step, $t < T$, the resampling weights are derived as follows.

$$\begin{aligned}
p(x^T,\theta|y^T) &= p(x^t|x_{t+1:T},\theta,y^T)p(x_{t+1:T},\theta|y^T) \\
&= p(x^t|x_{\partial_t^+},\theta,y^t)p(x_{t+1:T},\theta,y^T) \\
&= \frac{p(x^t,x_{\partial_t^+},\theta|y^t)}{p(x_{\partial_t^+},\theta|y^t)}p(x_{t+1:T},\theta,y^T) \\
&= \frac{p(x_{\partial_t^+}|x^t,\theta)p(\theta|x^t,y^t)p(x^t|y^t)}{p(x_{\partial_t^+},\theta|y^t)}p(x_{t+1:T},\theta,y^T) \\
p(x^t|x_{t+1:T},\theta,y^t) &\propto \overbrace{p(x_{\partial_t^+}|x^t,\theta)p(\theta|x^t,y^t)}^{\text{reweight}}\overbrace{p(x^t|y^t)}^{\text{ff samples}}\overbrace{p(x_{t+1:T},\theta,y^T)}^{\text{previously sampled}}
\end{aligned}$$

For the nearest 4 neighborhood structure, $\theta = \{\sigma^2, \tau^2\}$ and $p(\theta|x^t,y^t)$ can be calculated using sufficient statistics $p(\sigma^2,\tau^2|S_t)$. The weights are then as follows.

$$w^{(i)} \propto p(x^{(k)}_{\partial_t^+}|x^{t(i)},\sigma^{2(k)})p(\sigma^{2(k)},\tau^{2(k)}|x^{t(i)},y^t)$$

where (i) indexes $\dot{p}(x^t|y^t)$, (k) indexes $\dot{p}(x_{t+1:T},\sigma^2,\tau^2|y^T)$ and

$$p(x_{\partial_t^+}|x^t,\sigma^2) \propto f(x_t,x_{t+1}|\sigma^2)\prod_{k=t-J+1}^{t} f(x_{k+J},x_k|\sigma^2)$$

While the computational complexity of the forward filtering univariate SMRF model is only $\mathcal{O}(NIJ)$, the backward Boost Smoothing would be $\mathcal{O}(KNIJD)$ where D is the average number of neighborhood links in the smoothing weights per step. In the case of a nearest 4 neighbor GMRF, $D = J+1$ and so the computational complexity is $\mathcal{O}(KNIJ^2)$. This scales better with regard to the number of nodes then the algorithms previously discussed except for the conditional Gibbs implementation.

Furthermore, note that if a one row smoothing step is done in the forward filter pass after finishing each row, the forward filtering distributions will have the same structure as the multivariate SMRF model distributions. Then the Boost Smoothing for the multivariate SMRF model can be applied. This sort of scheme is similar to that of [25], except that here SMC is used instead of relying on the analytical tractability of multivariate normal distributions.

2.3 MRF Results

These MRF results are for 6×6 and 10×10 GMRFs with unknown observation error τ^2 and spatial smoothing parameter σ^2 with a nearest 4 neighborhood regular lattice structure. The hidden states for each sized MRF where generated randomly using $\sigma^2 = 1$ and $\sum x_{ij} = 0$, with new observations generated for each SNR. Two sets of priors where used for the static parameters and are listed in Table 2.2.

Empirical comparisons are made using the Root Mean Squared Error (RMSE) over 30 replications for 95% credible intervals of the available marginal distributions. The RMSE calculations are shown below, with i indexing replications, t indexing nodes, and $\{\bar{x}^{(t)}, \bar{\sigma}^2, \bar{\tau}^2\}$ are calculated using the averaged results of the Joint Gibbs implementation.

$$RMSE(x^T) = \sqrt{\frac{1}{2TN}\sum_{i=1}^{N}\sum_{t=1}^{T}\left(x_{2.5\%}^{(i,t)} - \bar{x}_{2.5\%}^{(t)}\right)^2 + \left(x_{97.5\%}^{(i,t)} - \bar{x}_{97.5\%}^{(t)}\right)^2} \tag{2.3.1}$$

$$RMSE(x_t) = \sqrt{\frac{1}{2N}\sum_{i=1}^{N}\left(x_{2.5\%}^{(i,t)} - \bar{x}_{2.5\%}^{(t)}\right)^2 + \left(x_{97.5\%}^{(i,t)} - \bar{x}_{97.5\%}^{(t)}\right)^2} \tag{2.3.2}$$

$$RMSE(\sigma^2) = \sqrt{\frac{1}{2N}\sum_{i=1}^{N}\left(\sigma_{2.5\%}^{2(i)} - \bar{\sigma}_{2.5\%}^{2}\right)^2 + \left(\sigma_{97.5\%}^{2(i)} - \bar{\sigma}_{97.5\%}^{2}\right)^2} \tag{2.3.3}$$

$$RMSE(\tau^2) = \sqrt{\frac{1}{2N}\sum_{i=1}^{N}\left(\tau_{2.5\%}^{2(i)} - \bar{\tau}_{2.5\%}^{2}\right)^2 + \left(\tau_{97.5\%}^{2(i)} - \bar{\tau}_{97.5\%}^{2}\right)^2} \tag{2.3.4}$$

The algorithms demonstrated include the Univariate SMRF model using both 50,000 and 1,000,000 particles, the Multivariate SMRF model using 50,000 particles, and the Joint and Conditional Gibbs samplers with 50,000 iterations. Figures 2.5, 2.11 and 2.19 show the joint posterior for the static parameters. The observations for the 6×6 and 10×10 MRFs are listed in Figures 2.3 and 2.17 respectively. These where generated from a common hidden state as seen in Figures 2.4 and 2.18. Boxplots of the averaged state and the static parameter RMSE can be seen as well. For the hidden states, the results are further split up across nodes to see the performance of the SMRF models with respect to time. Figures 2.10, 2.16, and 2.24 show the autocorrelation plots of σ^2 for the Joint and Conditional Gibbs samplers. The autocorrelation for τ^2 was insignificant in comparison and is thus omitted.

The two things that are important when interpreting these results is the trade off between speed and accuracy. With speed comes the opportunity for an increase in either particles or iterations, and thus increasing accuracy. With accuracy, fewer particles or iterations may be required for a given level of precision, and thus an increase in speed. These trade offs are also non linear. For example, the univariate SMRF model is so fast that the main limiting factor is the available RAM, and not the CPU speed. When using an equal number of particles or iterations, the most precise implementation is typically the Joint Gibbs sampler. The trade off between speed and accuracy also changes with the size of the problem. This is where the theoretical results of computational complexity are most important, because extrapolating trends from 6×6 and 10×10 MRFs may not be relevant for very large MRFs.

It is worth noting that the observed computation times from Table 2.3 are not especially reliable. The algorithms where implemented in R [29] with parallel programming and are not meant to be examples of what optimized code can do. Additionally, it worth remembering that for specific problems, the constant factor that is typically omitted in computational complexity can be very important. For example, the Joint Gibbs sampler is order $\mathcal{O}(NIJ^3)$, but this is with respect to the number of basic floating point operations needed to perform Cholesky decomposition. Univariate SMRF smoothing is order $\mathcal{O}(NIJ^2)$, but this is with respect to the number of neighborhood density terms needed to calculate the weights of the importance distribution, an operation usually requiring the use of logarithms.

Also note, that the SMRF models here are presented in the most unfavorable light. In more complicated non Gaussian MRFs, Joint and Conditional Gibbs sampling may not not be feasible, requiring instead a Metropolis-Hastings step. This would likely increase the autocorrelation of the Markov chain, making MCMC algorithms mix more slowly then is presented here. Another problem with MCMC algorithms is their difficulty with practically disjoint posterior modes, where mixing between modes is practically impossible, allowing the algorithm to miss important parts of the distribution. SMC based algorithms operate differently, processing the data sequential in contrast to the batch processing of MCMC. The Gordon example from Part I is an example where MCMC performs particularly poorly. By missing entire modes of the distribution, RMSE calculations can become totally inaccurate and misleading. Additionally, if Univariate SMRF models can perform comparably to Joint Gibbs for a MRF, then it should compete just as favorably with PMCMC, but with a substantial computational advantage.

It is also important to anticipate how the mixing of MCMC algorithms will scale with respect to larger data sets and more informative prior information. Remember, that in this setting there are $IJ + 2$ parameters for IJ data points. Results suggest that informative priors on the static parameters (relative to amount of data) increases the amount of independence between the hidden states and static parameters, which greatly improves mixing. Excessively vague priors become too dependent on the hidden states, and thus the Markov chain will take very small steps when sampling the hidden states and static parameters separately.

The two Boost Smoothing implementations give light on the difficulty of multivariate importance sampling. Univariate SMRF smoothing shows the effectiveness of Boost Smoothing in bending the RMSE curve for the hidden states from curving upward

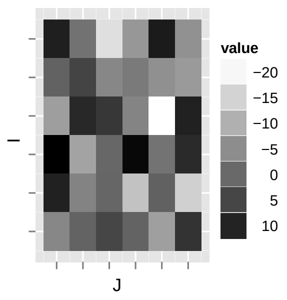
	SNR	**6 × 6**	**6 × 6**	**10×10**
$p(\sigma^2)$	All ($\sigma^2 = 1$)	$IG(10, 10)$	$IG(2, 2)$	$IG(10, 10)$
$p(\tau^2)$	Low ($\tau^2 = 100$)	$IG(10, 1000)$	$IG(2, 200)$	$IG(10, 1000)$
	Medium ($\tau^2 = 1$)	$IG(10, 10)$	$IG(2, 2)$	$IG(10, 10)$
	High ($\tau^2 = 0.01$)	$IG(10, 0.1)$	$IG(10, 0.2)$	$IG(10, 0.1)$

Table 2.2: MRF: Prior Specifications

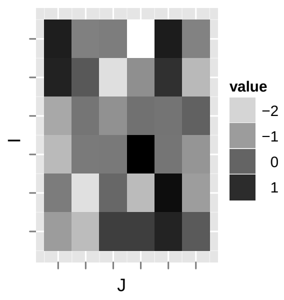
Algorithm		**6 × 6**	**10 × 10**	**Samples**	**Multi-threaded**
Univariate SMRF	(U1)	5s	0.7m	50,000	No
U SMRF K=250	(UB)	500s	45m	50,000	Yes
Univariate SMRF	(U2)	165s	16m	1,000,000	No
Multivariate SMRF	(M)	490s	19m	50,000	Yes
M SMRF K=1000	(MB)	1550s	29m	50,000	Yes
Conditional Gibbs	(CG)	455s	22m	50,000	Yes
Joint Gibbs	(JG)	310s	6.5m	50,000	Yes

Table 2.3: MRF: Average Run Time (8 processor cores). Note that the Univariate SMRF model is extremely fast, and an efficient multi threaded implementation would have taken about 2 minutes. While in this instance the Joint Gibbs sampler is faster then the Univariate SMRF with Boost Smoothing, the direct sampling of the hidden states will eventually become computational more expensive the the SMRF model for larger MRFs.

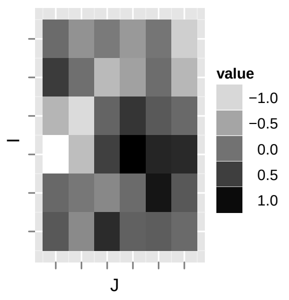
with filter smoothing to being relatively flat. For Multivariate SMRF, even with a large value of $K = 1000$, smoothing performs worse then the filter smoothing version. This suggests that multivariate samples, in contrast to univariate samples, from the filtering density make for a poor proposal for the smoothing distribution. This is a trend that is expected to get worse as dimensions increase, suggesting that this algorithm will not scale appropriately for larger MRFs.



(a) Low SNR: Data Y



(b) Medium SNR: Data Y



(c) High SNR: Data Y

Figure 2.3: 6×6 Observations. Three sets of observations for the 6×6 MRF where generated using $\tau^2 = \{.01, 1, 100\}$ with a common hidden state X and are show above.

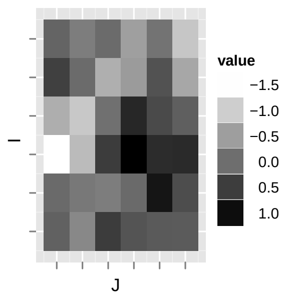
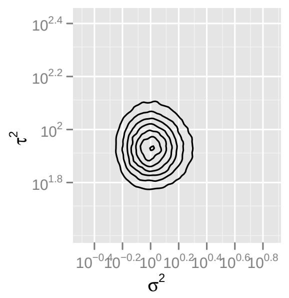
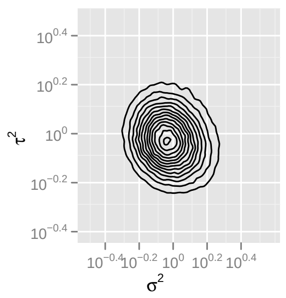


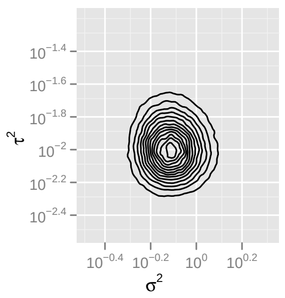
Figure 2.4: 6×6 Hidden States, used to generate observations for all three SNRs.



(a) Low SNR: $p(\sigma^2, \tau^2|Y)$

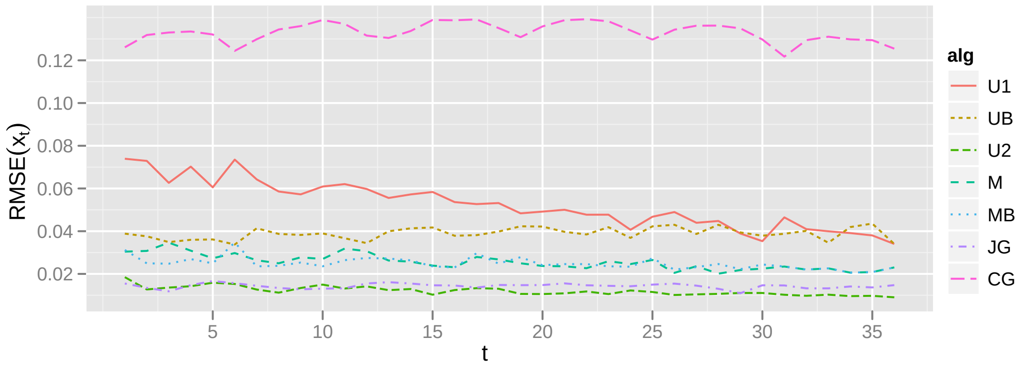


(b) Medium SNR: $p(\sigma^2, \tau^2|Y)$

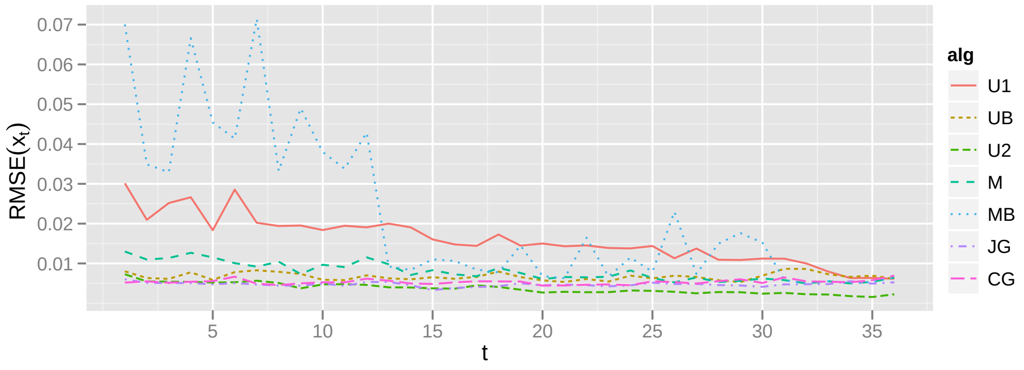


(c) High SNR: $p(\sigma^2, \tau^2|Y)$

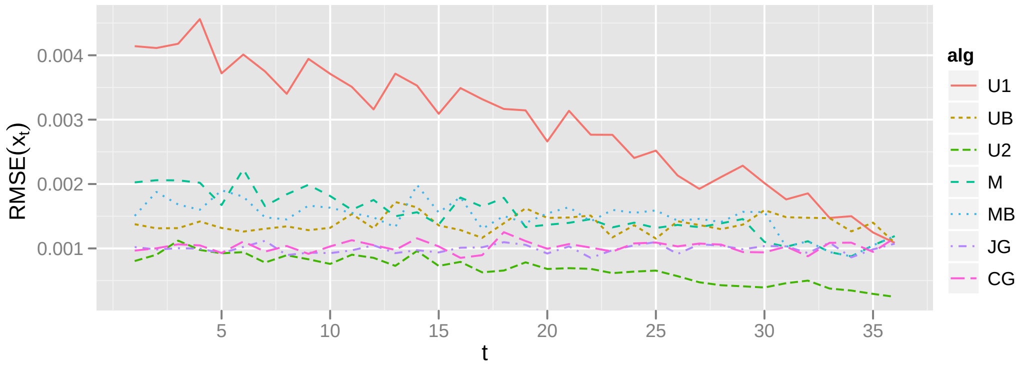
Figure 2.5: 6×6 Informative Priors: Posteriors. Contour plots of σ^2 and τ^2.



(a) Low SNR

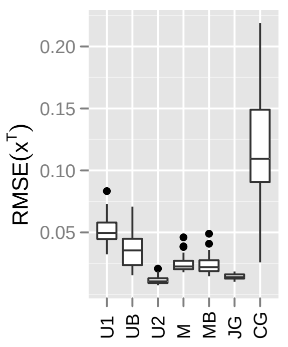


(b) Medium SNR

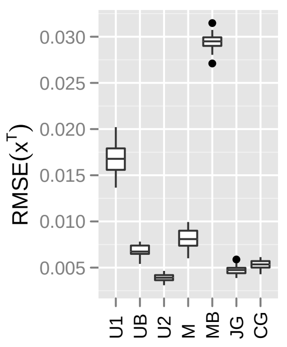


(c) High SNR

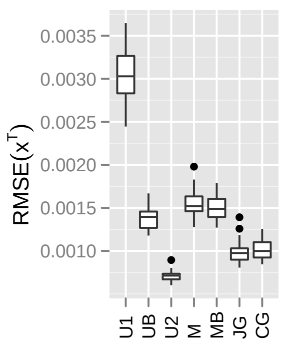
Figure 2.6: 6×6 Informative Priors: Hidden States RMSE(x_t). This figure shows that the RMSE for Boost Smoothing remains flat while SMRF models using filter smoothing show an increase in RMSE backward in time. The erratic results of the Multivariate SMRF model show the difficulties of SMC smoothing in high dimensions.



(a) Low SNR

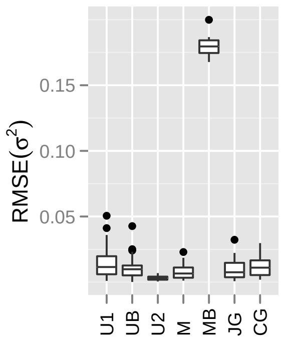


(b) Medium SNR

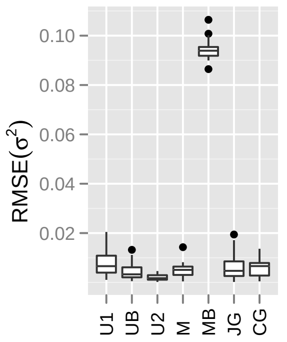


(c) High SNR

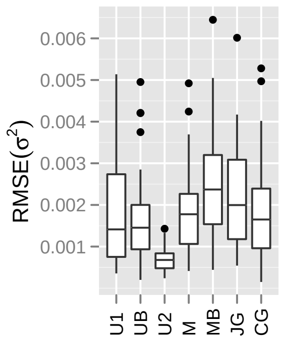
Figure 2.7: 6 $\times$ 6 Informative Priors: Boxplots For Hidden States. Note that in all SNRs, the Univariate SMRF model with 1 million particles provides the best performance. Conditional Gibbs sampling does particularly poorly in the low SNR case, while Multivariate SMRF with Boost Smoothing gives the worse performance for the medium SNR. For the high SNR case, Univariate SMRF gives the best and worse performance dependent upon the number of particles.



(a) Low SNR

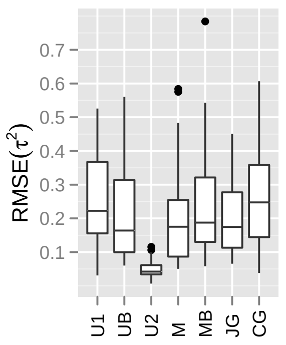


(b) Medium SNR

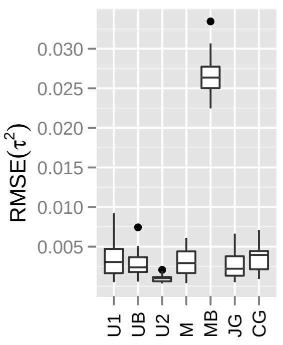


(c) High SNR

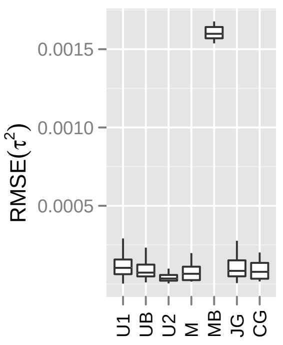
Figure 2.8: 6 $\times$ 6 Informative Priors: Boxplots For σ^2. All algorithms perform comparably when evaluating RMSE of the spatial smoothing parameter σ^2, except that Multivariate SMRF with Boost Smoothing performs noticeably worse in all cases. While the Univariate Boost Smoothing implementation actually improves RMSE results with $K = 250$, the Multivariate Boost Smoothing with $K = 1000$ performs worse then filter smoothing and thus is demonstrating the difficulty of SMC smoothing in high dimensions.



(a) Low SNR



(b) Medium SNR



(c) High SNR

Figure 2.9: 6×6 Informative Priors: Boxplots For τ^2. Again, as in the caption for the previous figure, the Univariate SMRF with 1 million particles gives the best performance in all cases and the use of Boost Smoothing for the Multivariate SMRF model actually increases RMSE.

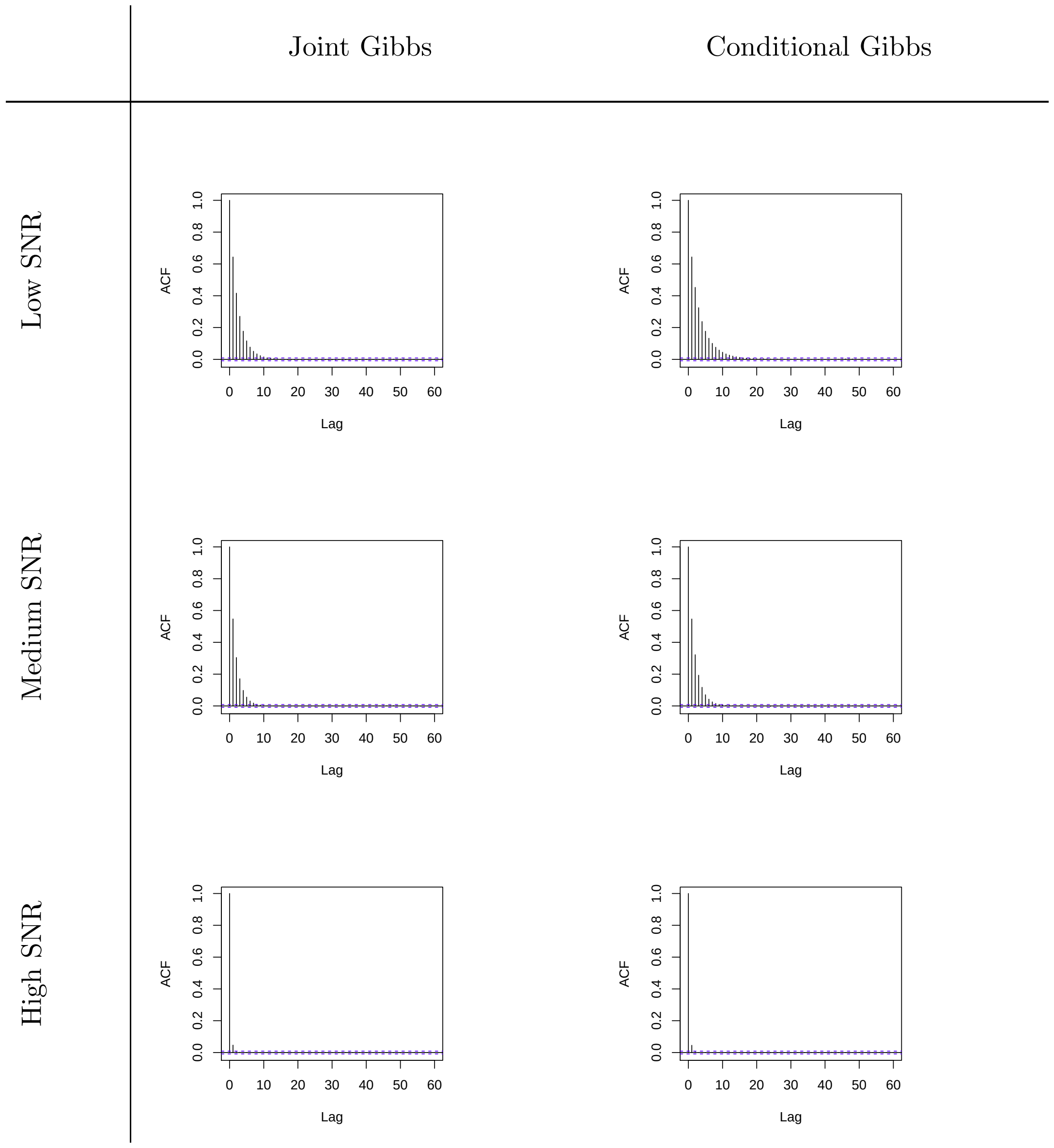
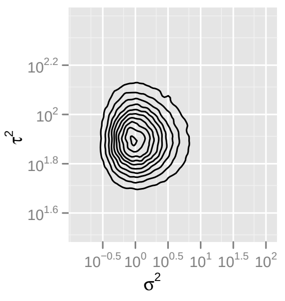
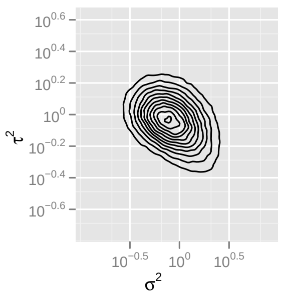


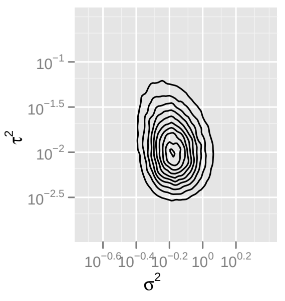
Figure 2.10: 6×6 Informative Priors: Joint And Conditional Gibbs ACF Plots For σ^2. Joint and Conditional Gibbs samplers provide adequate mixing rates for the spatial smoothing parameter σ^2 for this relatively small MRF.



(a) Low SNR: $p(\sigma^2, \tau^2|Y)$

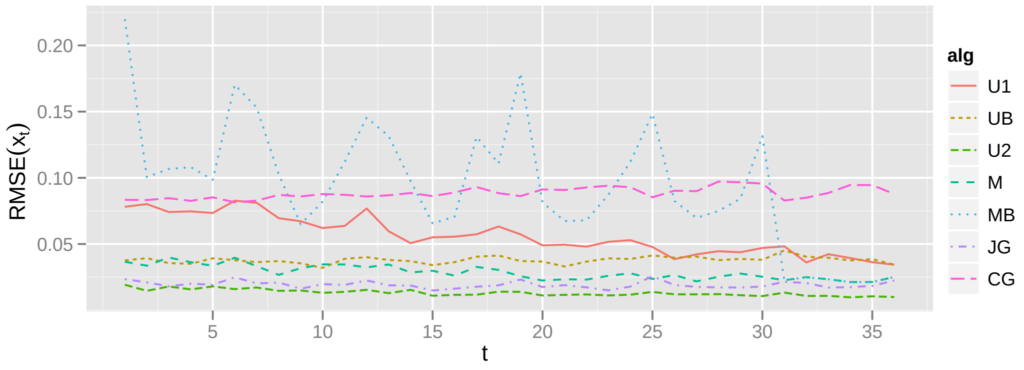


(b) Medium SNR: $p(\sigma^2, \tau^2|Y)$

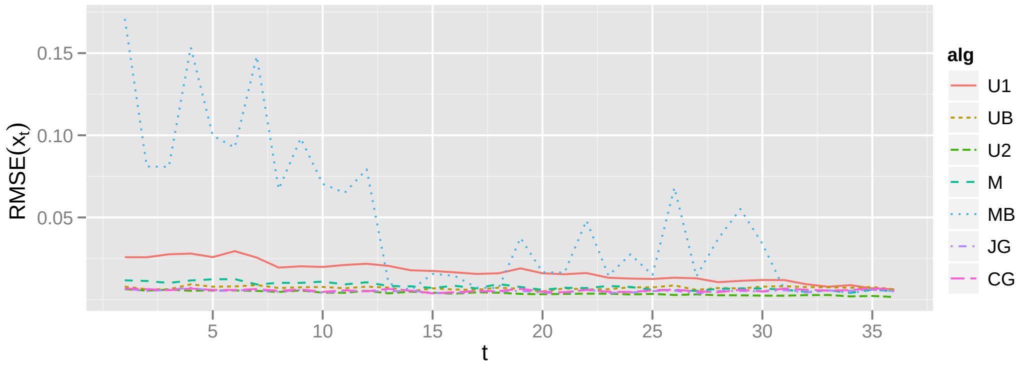


(c) High SNR: $p(\sigma^2, \tau^2|Y)$

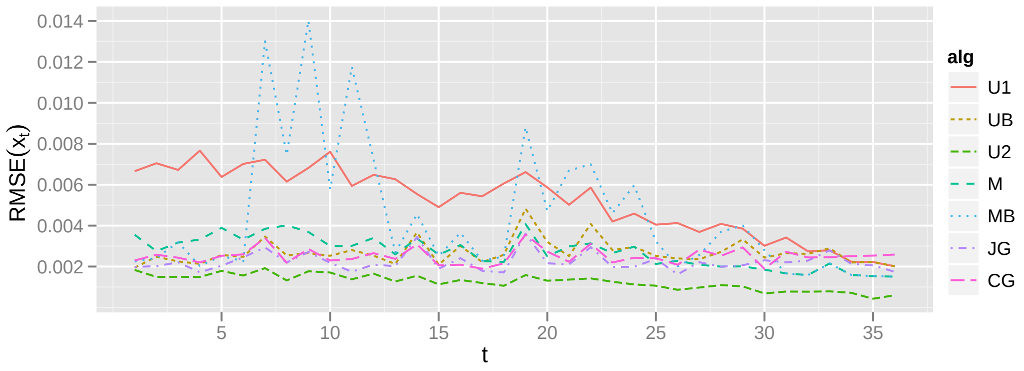
Figure 2.11: 6×6 Vague Priors: Posteriors. Contour plots of σ^2 and τ^2.



(a) Low SNR

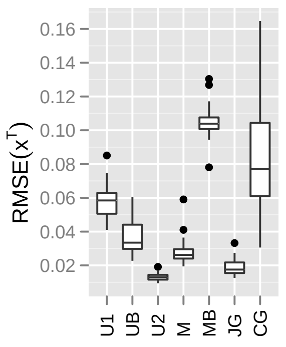


(b) Medium SNR

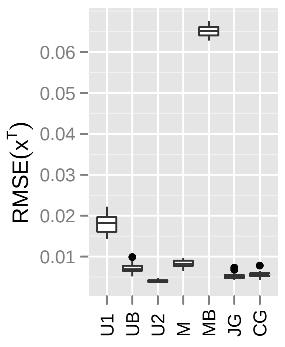


(c) High SNR

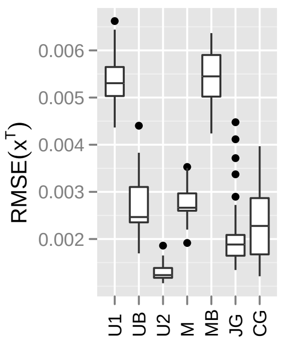
Figure 2.12: 6×6 Vague Priors: Hidden States RMSE(x_t). This figure shows that the RMSE for Boost Smoothing remains flat while SMRF models using filter smoothing show an increase in RMSE backward in time. The erratic results of the Multivariate SMRF model show the difficulties of SMC smoothing in high dimensions.



(a) Low SNR

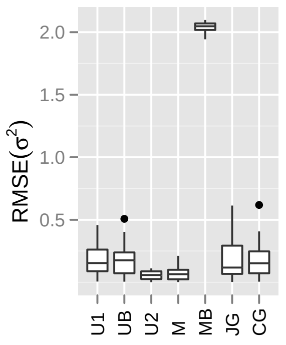


(b) Medium SNR

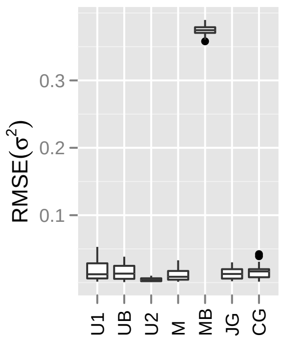


(c) High SNR

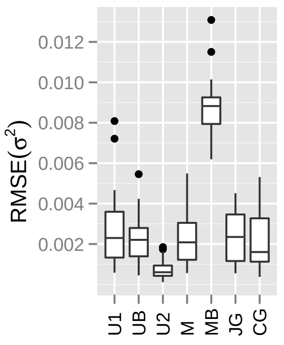
Figure 2.13: 6×6 Vague Priors: Boxplots For Hidden States. As in the informative prior case, note that in all SNRs, the Univariate SMRF model with 1 million particles provides the best performance. Conditional Gibbs sampling does particularly poorly in the low SNR case. Multivariate SMRF with Boost Smoothing gives the worse performance for the medium SNR, suggesting the need for an even larger value of K then 1000. For the high SNR case, Univariate SMRF gives the best and worse performance dependent upon the number of particles.



(a) Low SNR

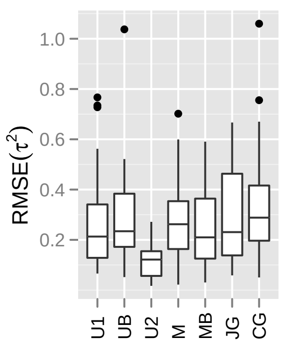


(b) Medium SNR

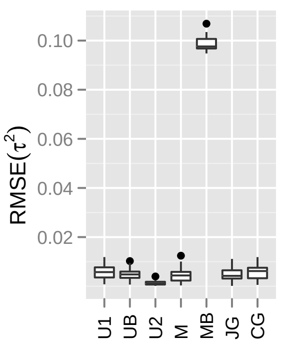


(c) High SNR

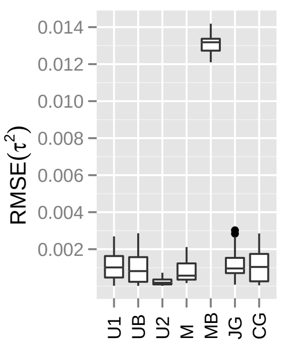
Figure 2.14: 6×6 Vague Priors: Boxplots For σ^2. Similarly to the case with informative priors, all algorithms perform comparably when evaluating RMSE of the spatial smoothing parameter σ^2, except that Multivariate SMRF with Boost Smoothing performs poorly in all SNR cases. Since the Univariate Boost Smoothing implementation actually improves RMSE results with $K = 250$, the Multivariate Boost Smoothing with $K = 1000$ is demonstrating the difficulty of SMC smoothing in high dimensions.



(a) Low SNR



(b) Medium SNR



(c) High SNR

Figure 2.15: 6×6 Vague Priors: Boxplots For τ^2. As has already been seen, the Univariate SMRF with 1 million particles gives the best performance in all cases and the use of Boost Smoothing for the Multivariate SMRF model actually increases RMSE in some cases. These results also show an overall increase in average RMSE compared to using more informative priors.

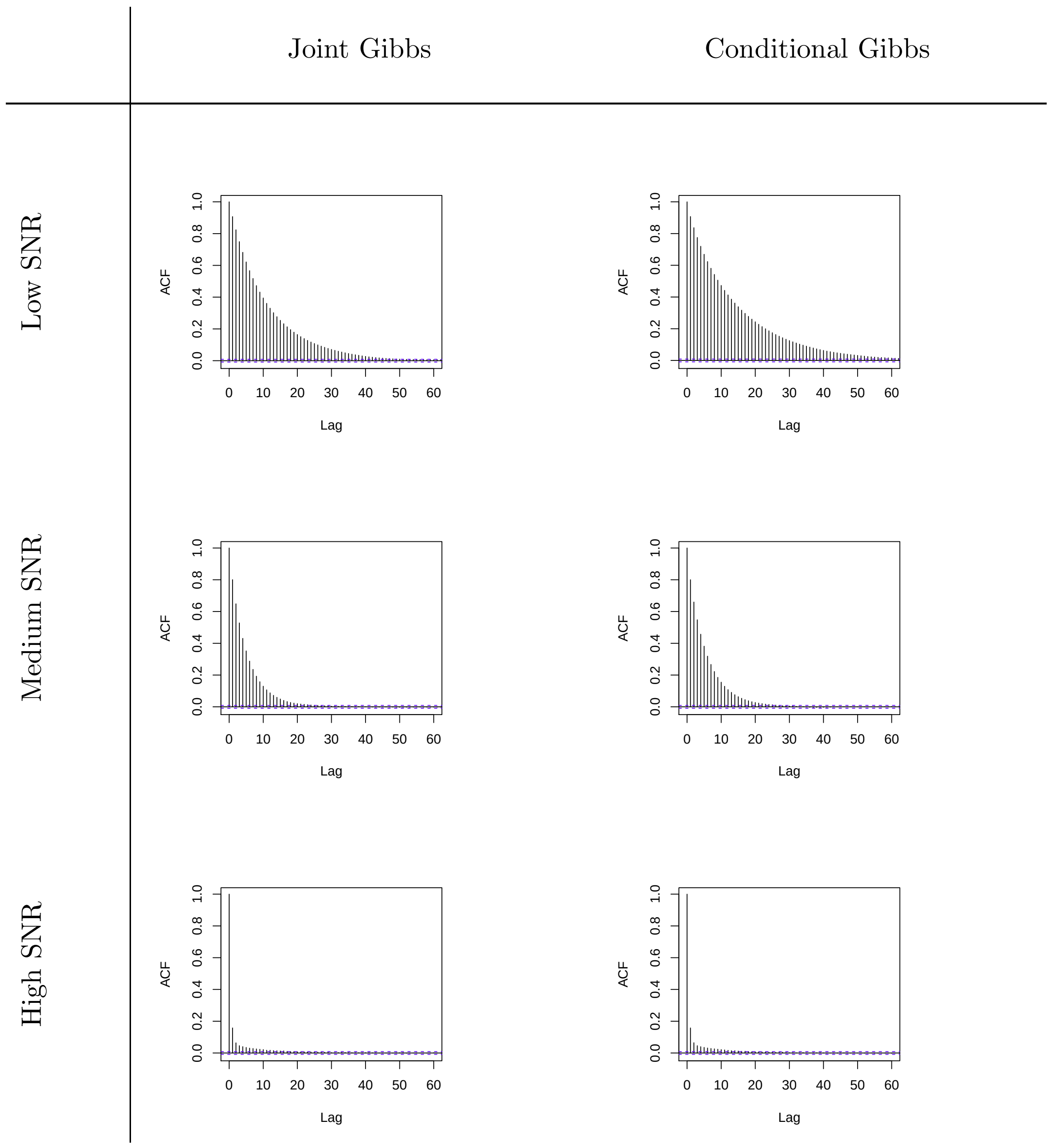
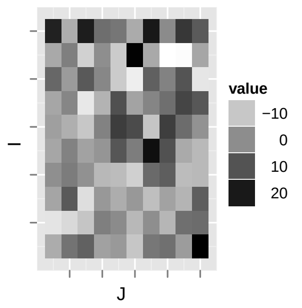
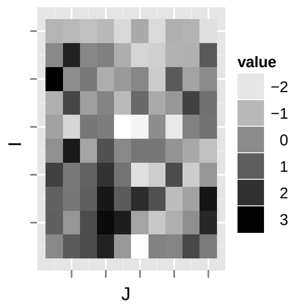


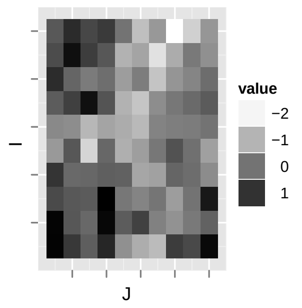
Figure 2.16: 6×6 Vague Priors: Joint And Conditional Gibbs ACF Plots For σ^2. Joint and Conditional Gibbs samplers provide worse mixing rates with vague priors then was seen with informative priors for the spatial smoothing parameter σ^2 for this relatively small MRF.



(a) Low SNR: Data Y



(b) Medium SNR: Data Y



(c) High SNR: Data Y

Figure 2.17: 10 × 10 Observations. Three sets of observations for the 10 × 10 MRF where generated using $\tau^2 = \{.01, 1, 100\}$ and a common hidden state X and are show above.

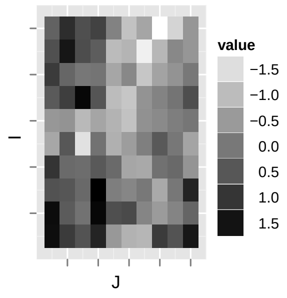
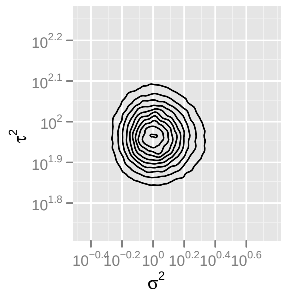


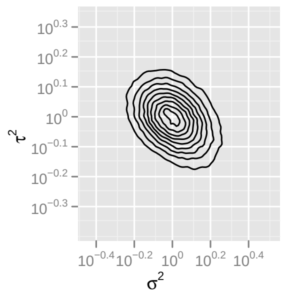
Figure 2.18: 10 × 10 Hidden States, used to generate observations for all three SNRs.

2.4 Univariate SMRF Improvements

Univariate SMRF appears to be competitive with the Joint Gibbs sampler, and results suggest it might scale better for larger MRFs. Additionally, a number of modifications can be made to the backward pass of the Univariate SMRF model to make it computationally more efficient. Proposed modifications are as follows.

- Since forward filtering is so much faster then Boost Smoothing, reduce the MC error associated with the forward filter step by running with more particles.


(a) Low SNR: $p(\sigma^2, \tau^2|Y)$


(b) Medium SNR: $p(\sigma^2, \tau^2|Y)$

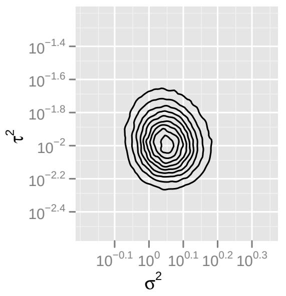
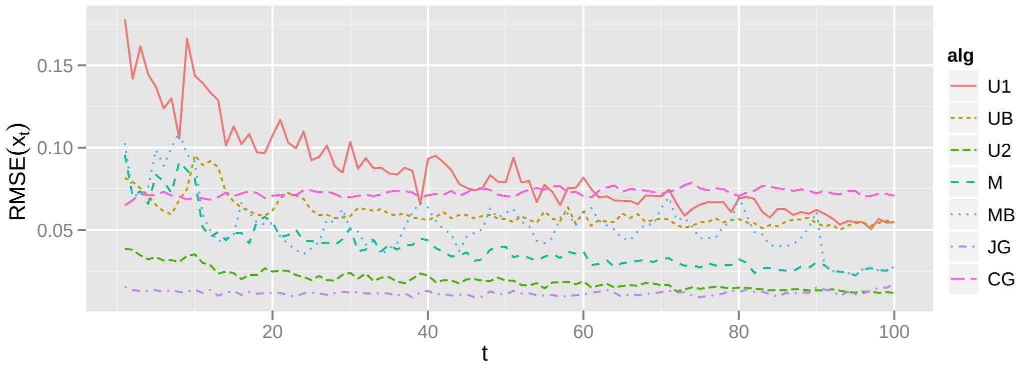

(c) High SNR: $p(\sigma^2, \tau^2|Y)$

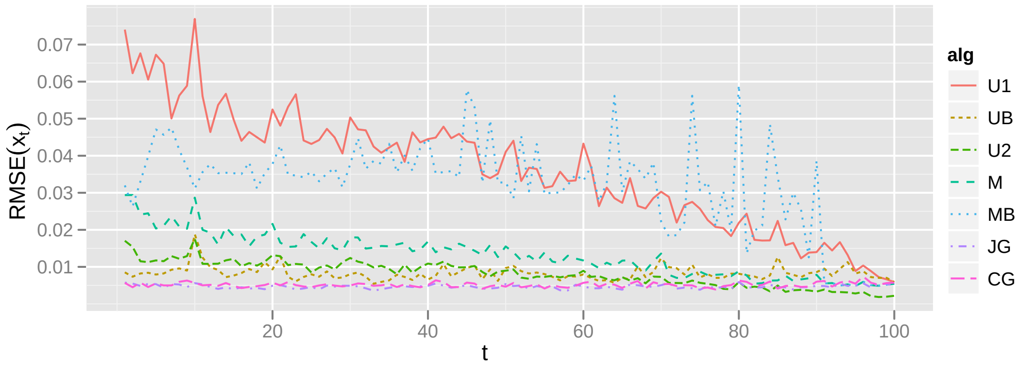
Figure 2.19: 10×10 Informative Priors: Posteriors. Contour plots of σ^2 and τ^2.

- Since filter smoothing works very well for a few time steps, only perform Boost Smoothing periodically or adaptively as needed.
- Results from the nonlinear model of Section 1.4 suggest that larger values of K are only needed at certain time steps, hinting at the usefulness of adaptive ways of choosing K.

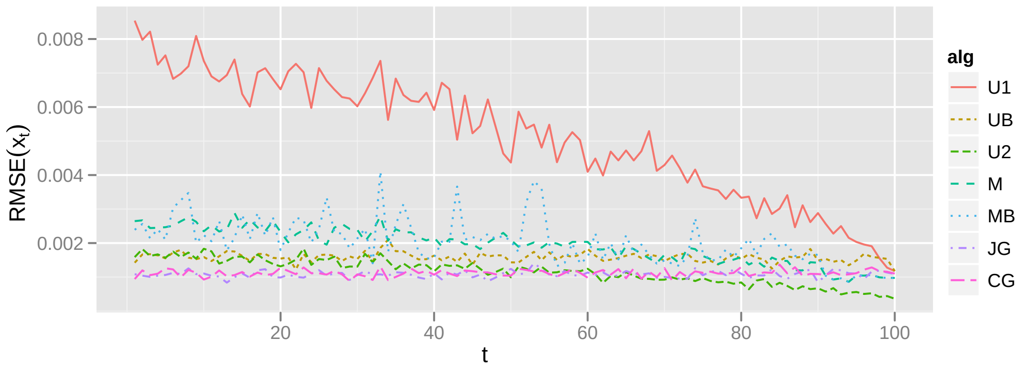
Another approach is to use the results of the Univariate SMRF model as a massively parallel burn in procedure for a multiple chain MCMC algorithm, using ideas from population simulation methods. To mitigate accumulation of MC error in the SMC smoothing process, each particle trajectory can be treated as 1 of N MCMC chains. By performing a series of Gibbs or Metropolis-Hastings MCMC update steps on each chain in parallel, the quality of the SMC particles can be improved and verified that they are at a limiting distribution by using MCMC convergence diagnostics for multiple chain MCMC. Since the particles should already be very near to the posterior distribution, the convergence of this MCMC should be a fraction of the auto correlation of a single



(a) Low SNR

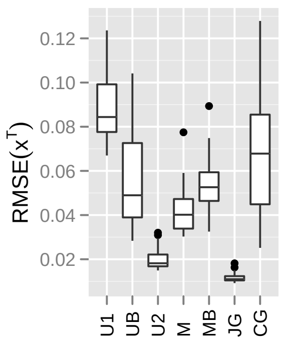


(b) Medium SNR

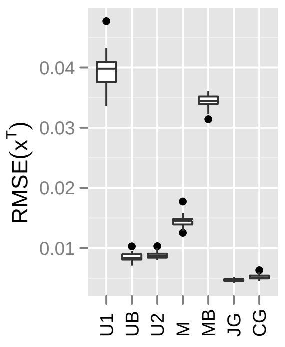


(c) High SNR

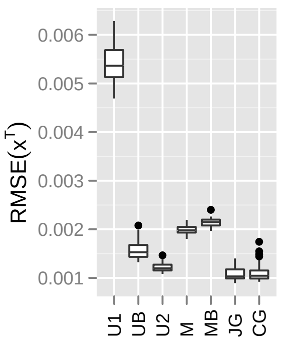
Figure 2.20: 10×10 Informative Priors: Hidden States RMSE(x_t). The larger Univariate SMRF model (U2) initially estimates the last state with more precision then the Joint Gibbs sampler. This eventually changes as only the Boost algorithms and Gibbs samplers and algorithms have a flat RMSE plot while filter smoothing does not.



(a) Low SNR

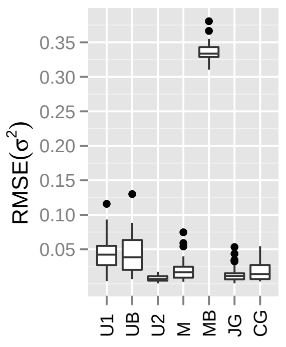


(b) Medium SNR

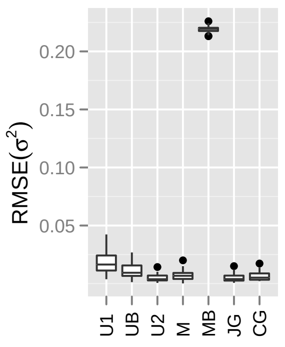


(c) High SNR

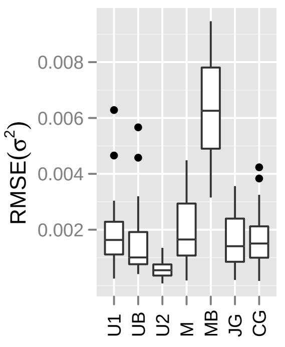
Figure 2.21: 10×10 Informative Priors: Boxplots For Hidden States. Here the Joint Gibbs Sampler performs slightly better then the larger Univariate SMRF model. Also of note is the size of the improvement Boost Smoothing offers over filter smoothing for the smaller sized Univariate SMRF model (U1). Secondly, it seems that both the Joint and Conditional Gibbs samplers perform about equally for the medium and low SNRs, suggesting that the added expense of direct sampling is not always worth the computation cost.



(a) Low SNR

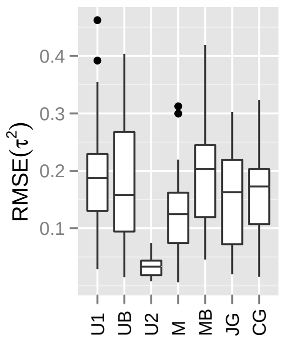


(b) Medium SNR

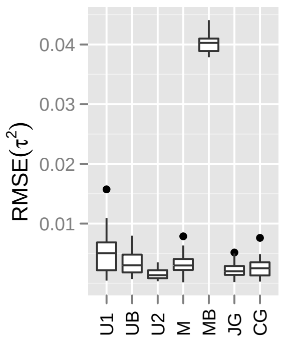


(c) High SNR

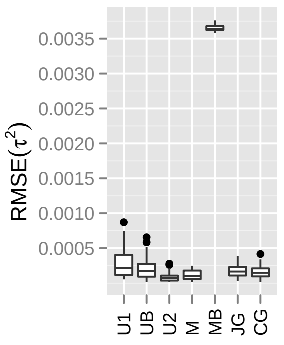
Figure 2.22: 10×10 Informative Priors: Boxplots For σ^2. Here most algorithms perform about the same except again for the Multivariate SMRF model with Boost Smoothing, due to the need of an even larger value of K for smoothing in high dimensions.



(a) Low SNR



(b) Medium SNR



(c) High SNR

Figure 2.23: 10×10 Informative Priors: Boxplots For τ^2. In this case the larger Univariate SMRF model (U2) does a better job estimate the intervals for τ^2 then the Joint Gibbs sampler.

MCMC chain, not not the relatively slow convergence of MCMC burn in step. Hence, these SMC smoothing algorithms can be view as an advanced burn in procedure for exceedingly multiple chain MCMC, with much better posterior exploration of disjoint modes. Another form of this approach is motivated by the field of Genetic algorithm. Crossover steps can be used to share information between chains to allow for mixing between disjoint state spaces. This will also allow for more precise estimation of the relative density associated with each disjoint mode. Perhaps, with so many samples, this would be a better way to replenish the particles then using MCMC or PMCMC steps. A recent review of population simulation methods can be found in [17].

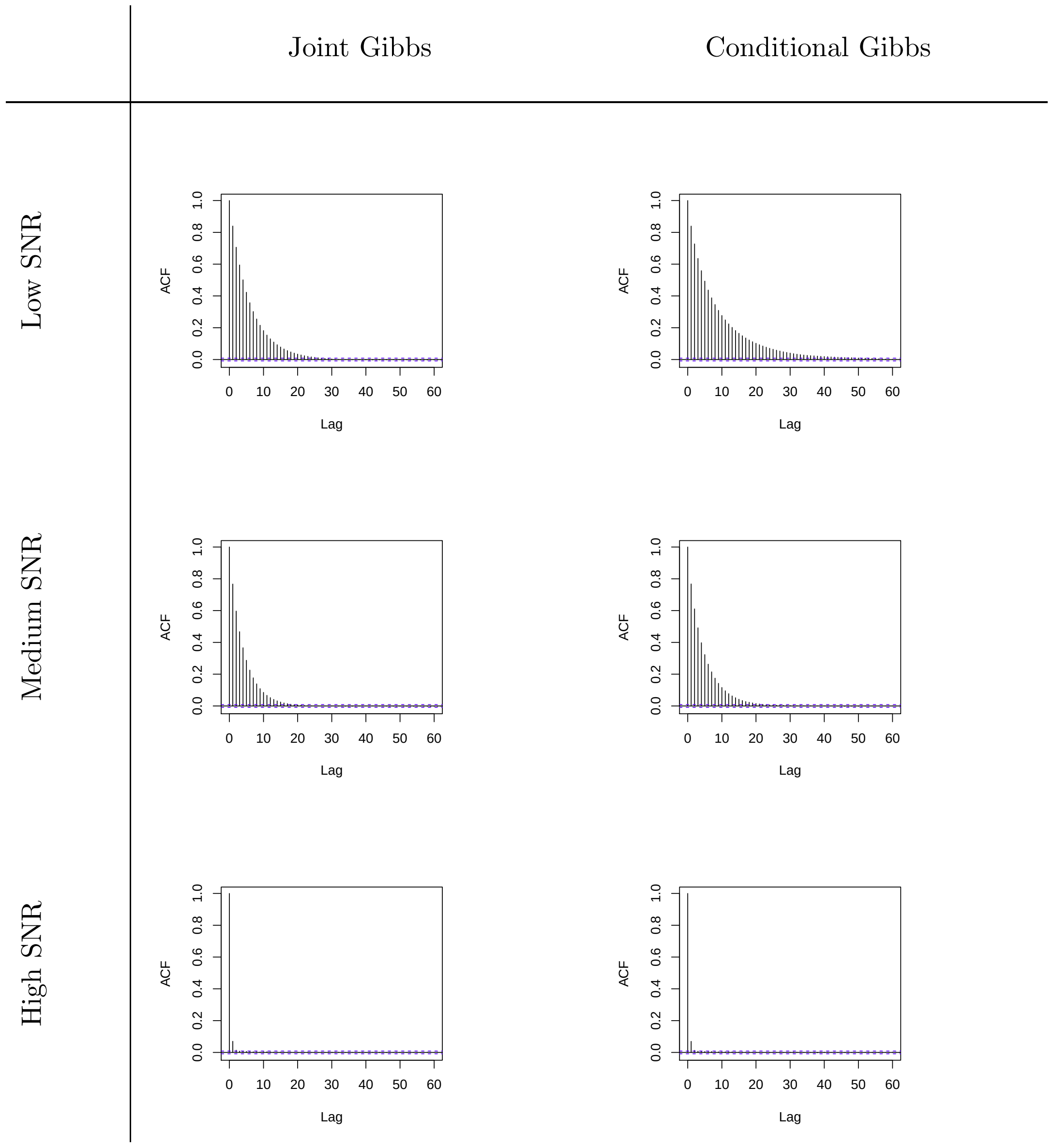


Figure 2.24: 10×10 Informative Priors: Joint And Conditional Gibbs ACF Plots For σ^2. The auto correlation increases with the size of the MRF and becomes more of a problem in low SNR case. Conditional Gibbs shows slower mixing then Joint Gibbs, but when considering the differences in computational complexity, the natural preference for the Joint Gibbs sampler should not be automatic.

Chapter 3

Future Work

Of lessons learned about the computation of multivariate statistical models, the first is to avoid multivariate sampling or density estimates by converting to a univariate state space model. This was the problem with Kernel Smoothing, in that high dimensional KDE is not practical. The second lesson, highly related to the first for this class of MRFs, is the avoidance of any sort of matrix operation, and specifically matrix inversion or decomposition.

The computational complexity of the Univariate SMRF model is $\mathcal{O}(NIJ^2)$, and this suggests that this algorithm is ideally suited for situations where $J << I$. This can happen in time series where high dimensional states are on the order of 10 while the number of time steps can be 1,000 or more. The filtering distributions of a multivariate time series model can be recovered from the univariate transformed model by using either Boost Smoothing or filter smoothing to obtain equivalent fixed lag smoothing distribution. The conditional dependence structure may no longer be sparse, but it may still outperform competing algorithms that find sampling in high dimensions difficult, similarly to the difficulties experienced with the Multivariate SMRF model. The factorizing of the posterior seen in the Univariate MRF is not specific to MRFs, and this idea can be applied to other posterior likelihoods with a suitable factorization.

Lastly, the whole field of Population SMC is infused with a predilection that SMC Smoothing is computationally inefficient. These results show that SMC can be competitive with MCMC methods on a much larger class of problems in Bayesian inference then previously thought.

Appendix A

Proofs

A.1 Proof Of Convergence For Boost Smoothing

This proof builds upon the results of GDW04 [15], by showing how the results of M smoothers with K particles can be averaged to provide an estimate whose rate of convergence is equivalent to one smoother with $N = MK$ particles with a substantial improvement in computational complexity. Currently, this proof does not incorporate static parameter estimation.

The formal measure-theoretic notation used in GDW04 is now described. Let (Ω, F, P) be a probability space defined on two vector real-valued stochastic processes $X = \{X_t, t \in \mathbb{N}*\}$ and $Y = \{Y_t, t \in \mathbb{N}*\}$ of dimensions n_x and n_y respectively. Assuming Lebesgue measures, let the unobserved hidden states X follow a Markov process with transition kernel $f(dx_t|x_{t-1}) = f(x_t|x_{t-1})dx_t$ and initial distribution $X_1 \sim f(dx_1)$, with observed states Y following $g(dy_t|x_t) = g(y_t|x_t)dy_t$. For measure μ, function φ, and Markov kernel K, the following notation is defined.

$$(\mu, \varphi) \triangleq \int \varphi d\mu \qquad \mu K(A) \triangleq \int \mu(dx)K(A|x) \qquad K\varphi(x) \triangleq \int K(dz|x)\varphi(z)$$

The posterior distributions are defined using the following measures.

$$\begin{aligned} \pi_{t|t-1}(dx_t) &\triangleq P(dx_t|Y_{1:t-1} = y_{1:t=1}) \\ \pi_{t|t}(dx_t) &\triangleq P(dx_t|Y_{1:t} = y_{1:t}) \\ \pi_{t_1:t_2|t_3}(dx_{t_1:t_2}) &\triangleq P(dx_{t_1:t_2}|Y_{1:t_3} = y_{1:t_3}) \end{aligned}$$

The normalized weighted measure of the particle filtering densities and the unweighted measure generated from N joint samples are defined next.

$$\pi^N_{t|t}(dx_t) \triangleq \sum_{i=1}^{N} w_t^{(i)} \delta_{x_t^{(i)}}(dx_t) \qquad \pi^N_{1:T|T}(dx_{1:T}) \triangleq \sum_{i=1}^{N} \delta_{x_{1:T}^{(i)}}(dx_{1:T})$$

The fixed lag smoothing measure $\pi_{t:T|T}$ for $t \in 1:T$ and the joint smoothing measure $\pi_{1:T|T}$ representing the truth are defined by choosing a measure $\rho_t(dx_{t-1:t})$ that is absolutely continuous with respect to $\pi_{t-1|t-1}(dx_{t-1:t})$ and admits h_t as a strictly positive Radon-Nykodym derivative as defined below.

$$\pi_{1:T|T} \triangleq \pi_{t-1|t-1}(dx_{t-1})q(dx_t|y_t, x_{t-1:t})$$
$$h_t(y_t, x_{t-1}, x_t) \propto \frac{\pi_{t-1|t-1}(dx_{t-1:t})}{\rho_t(dx_{t-1:t}})$$

Sufficient conditions for convergence are then satisfied by assuming that φ is a function in the space of bounded, Borel-measurable functions such that $\varphi \in B(\mathbb{R}^n)$,

$$||\varphi|| \triangleq \sup_{x \in \mathbb{R}^n} \varphi(x)$$

Assumption A.1.1. *[15]. $\pi_{t-1:t|t}$ is absolutely continuous with respect to ρ_t. Moreover, $g(y_t|x_t)h_t(y_t, x_{t-1}, x_t)$ is positive and bounded in argument $(x_{t-1}, x_t) \in (\mathbb{R}^{n_x})^2$*

Theorem A.1.1. *[10]. Under Assumption A.1.1, for all $t > 0$, there exists $c_{t|t}$ independent of N such that for any $\phi \in B(\mathbb{R}^{n_x})$,*

$$E[((\pi^N_{t|t}, \phi) - (\pi_{t|t}, \phi))^2] \leq c_{t|t}\frac{||\phi||^2}{N},$$

where the expectation is over all realizations of the random particle method.

Now, define $\bar{f}(x) \triangleq ||f(x|.)||$ and consider the next assumption.

Assumption A.1.2. *[15]. For any $t \in (1, ..., T)$, we have*

$$\left(\pi_{t|T}, \left(\frac{\bar{f}}{\pi_{t|t-1}}\right)^2\right) < \infty.$$

Theorem A.1.2. *[15]. Under Assumptions A.1.1 and A.1.2, for all $t \in (1, ..., T)$, there exists $c_{t|T}$ independent of N such that for any $\phi \in B(\mathbb{R}^{n_x T})$*

$$E[((\pi^N_{1:T|T}, \phi) - (\pi_{1:T|T}, \phi))^2] \leq c_{1|T}\frac{||\phi||^2}{N}$$

where $c_{1|T}$ can be computed using the backward recursion,

$$c_{t|T} = \left((c_{t+1|T})^{1/2} + 2(c_{t|t})^{1/2}\left(\pi_{t+1|T}, \left(\frac{\bar{f}}{\pi_{t+1|t}}\right)^2\right)^{1/2}\right)^2,$$

and $c_{t|t}$ is given by Theorem A.1.1.

Now using Theorem A.1.2, the Theorem A.1.3 can be stated which requires samples from each of M groups of K particles where $\pi_{t|t}^{K,(i)}$ is obtained using the $GDW04$ algorithm for $i \in 1 : M$.

$$\pi_{1:T|T}^{K,M}(dx_{1:T}) \triangleq \frac{1}{M}\sum_{i=1}^{M} \pi_{1:T|T}^{K,(i)}(dx_{1:T})$$

Theorem A.1.3. *Under Assumptions A.1.1 and A.1.2, for all* $t \in (1, ..., T)$*, there exists* $c_{t|T}$ *independent of* K *and* M *such that for any* $\phi \in B(\mathbb{R}^{n_x T})$

$$E[((\pi_{1:T|T}^{K,M}, \phi) - (\pi_{1:T|T}, \phi))^2] \leq c_{1|T}\frac{||\phi||^2}{KM} + E^2[((\pi_{1:T|T}^{K}, \phi) - (\pi_{1:T|T}, \phi))^2]$$

with a worse case bound when $V[(\pi_{1:T|T}^{K}, \phi) - (\pi_{1:T|T}, \phi)] = 0$ *as shown below.*

$$E[((\pi_{1:T|T}^{K,M}, \phi) - (\pi_{1:T|T}, \phi))^2] \leq c_{1|T}\frac{||\phi||^2}{KM} + c_{1|T}\frac{||\phi||^2}{K}$$

Proof. Let

$$X_i = \left(\pi_{1:T|T}^{K,(i)}, \phi\right) - \left(\pi_{1:T|T}, \phi\right)$$

such that $E(X_i^2) = V(X_i) + E^2(X_i^2) \leq c_{1|T}\frac{||\phi||^2}{K}$. Then for M and K large, $X_i \sim$ i.i.d.,

$$\begin{aligned}
E[((\pi_{1:T|T}^{K,M}, \phi) - (\pi_{1:T|T}, \phi))^2] &= E\left[\left(\frac{1}{M}\sum_{i=1}^{M}\left(\pi_{1:T|T}^{K,(i)}, \phi\right) - (\pi_{1:T|T}, \phi)\right)^2\right] \\
&= V\left(\frac{1}{M}\sum_{i=1}^{M} X_i\right) + E^2\left(\frac{1}{M}\sum_{i=1}^{M} X_i\right) \\
&\leq \frac{1}{M}E(X_i^2) + E^2(X_i^2) \\
&\leq c_{1|T}\frac{||\phi||^2}{MK} + E^2(X_i^2) \\
&= c_{1|T}\frac{||\phi||^2}{KM} + E^2[((\pi_{1:T|T}^{K}, \phi) - (\pi_{1:T|T}, \phi))^2]
\end{aligned}$$

Which completes the proof for Theorem A.1.3 $\square$

A.2 Proof Of Convergence For Marginal Kernel Smoothing

Assume that the forward filtering densities $\dot{p}(x_t|y^t) = \{w_t^{(i)}, x_t^{(i)}\}_{i=1}^N$ for $t = 1 : T$ are known. Then starting with $p(x^T|y^T)$ and proceeding recursively for $t = T-1 : 1$ while assuming the Markov property, the recursive structure is as follows.

$$p(x_{t:t+1}|y^T) = p(x_t|x_{t+1}, y^T)\int p(x_{t+1:t+2}|y^T)\mathrm{d}x_{t+2} = p(x_t|x_{t+1}, y^t)p(x_{t+1}|y^T)$$

Since $p(x_t|x_{t+1}, y^t)$ in general cannot be sampled directly, a convenient importance or backward propagation distribution $q(x_t|x_{t+1}, y^t)$ is chosen such that,

$$p(x_{t:t+1}|y^T) = \lim_{N\to\infty} \dot{p}(x_{t:t+1}|y^T) = \lim_{N\to\infty}\sum_{i=1}^N w_{t|T}^{(i)}\delta_{x_t^{(i)}}(x_t)\delta_{x_{t+1}^{(i)}}(x_{t+1})$$

where $x_{t+1}^{(i)} \sim \dot{p}(x_{t+1}|y^T)$ and $x_t^{(i)} \sim q(x_t|x_{t+1}^{(i)}, y^t)$, $\sum_{i=1}^N w_{t|T}^{(i)} = 1$, and

$$w_{t|T}^{(i)} \propto \frac{p(x_t^{(i)}|x_{t+1}^{(i)}, y^t)}{q(x_t^{(i)}|x_{t+1}^{(i)}, y^t)} = \frac{p(x_t^{(i)}, x_{t+1}^{(i)}|y^t)}{p(x_{t+1}^{(i)}|y^t)q(x_t^{(i)}|x_{t+1}^{(i)}, y^t)} = \frac{p(x_{t+1}^{(i)}|x_t^{(i)})p(x_t^{(i)}|y^t)}{p(x_{t+1}^{(i)}|y^t)q(x_t^{(i)}|x_{t+1}^{(i)}, y^t)}$$

The densities $p(x_t|y^t)$ and $p(x_{t+1}|y^t)$ are estimated using weighted samples from the forward filtering pass and kernel density estimation to allow for density estimation at new particle locations.

$$p(x_t^{(i)}|y^t) = \lim_{\substack{h\to 0\\ Nh\to\infty}} \sum_{j=1}^N w_t^{(j)} K_h(x_t^{(j)} - x_t^{(i)})$$
$$p(x_{t+1}^{(i)}|y^t) = \lim_{\substack{h\to 0\\ Nh\to\infty}} \sum_{j=1}^N w_t^{(j)} K_h(x_{t+1}^{(j)} - x_{t+1}^{(i)}) = \lim_{N\to\infty}\sum_{j=1}^N w_t^{(j)} p(x_{t+1}^{(i)}|x_t^{(j)})$$

While the choice of kernel is arbitrary, an ideal choice for estimating $p(x_{t+1}^{(i)}|y^t)$ is to use the system equation such that $K_h(x_{t+1}^{(j)} - x_{t+1}^{(i)}) = p(x_{t+1}^{(i)}|x_t^{(j)})$. This greatly reduces MC error in the weights by ensuring that tail densities are more precisely estimated in the denominator.

To show the convergence of this kernel density estimate, let $\{w^{(i)}, x^{(i)}\}_{i=1}^N$ be the weighted sample defined by the discrete distribution and the kernel density estimate respectively as follows.

$$\dot{p}(x) = \sum_{i=1}^N w^{(i)}\delta_{x^{(i)}}(x) \qquad\qquad \hat{p}_h(x) = \sum_{i=1}^N w^{(i)} K_h(x - x^{(i)})$$

By holding the bandwidth h constant, take the limit as $N \to \infty$ to show that

$$\lim_{N\to\infty} \hat{p}_h(x^*) = \lim_{N\to\infty} \sum_{i=1}^{N} w^{(i)} K_h(x^* - x^{(i)}) = \int K_h(x^* - x)p(x)\mathrm{d}x$$

and hence for constant h the KDE is a convolution of the kernel and the target density function. Next, taking the limit of this convolution as $h \to 0$, observe that

$$\lim_{h\to 0} \hat{p}_h(x^*) = \lim_{h\to 0} \int K_h(x - x^*)p(x)\mathrm{d}x = \int \delta(x - x^*)p(x)\mathrm{d}x = p(x)$$

This completes the proof of convergence for Marginal Kernel Smoothing.

A.3 Proof For Binding Separate Marginal Distributions

Using concomitants of order statistics it can be shown that samples from the joint probability $\{x_{t:t+1}^{(i)}\}_{i=1}^{N} \sim p(x_{t:t+1}|y^T)$ can be replenished with a new sample $\{\tilde{x}_t^{(i)}\}_{i=1}^{N} \sim p(x_t|Y^T)$ by sorting both samples by x_t or $\tilde{x}_t$ respectively. First, decompose $p(x_{t:t+1}|y^T)$ into a mixture of bivariate order statistics $p(x_{t+1}^{[i]}, x_t^{(i)}|y^T)$, where $x_t^{(i)}$ is the i^{th} order statistic of a sample of size N, $x_{t+1}^{[i]}$ is its concomitant, $P_t^{-1}(p)$ is the inverse CDF of $p(x_t)$, and $\xi_p = \lim_{N\to\infty} x_t^{(pN)} = P^{-1}(p)$ for $p \in (0, 1)$ is the limiting value of the central order statistic. Only central order statistics are considered since the accumulated probability mass of the non-central order statistics is 0 in the limit.

$$\begin{aligned}
p(x_t, x_{t+1}|y^T) &= \lim_{N\to\infty} \frac{1}{N} \sum_{i\in\text{central}} p(x_t^{(i)}, x_{t+1}^{[i]}|y^T) \\
&= \lim_{N\to\infty} \frac{1}{N} \sum_{i\in\text{central}} p(x_t^{(i)}|y^T)p(x_{t+1}^{[i]}|x_t^{(i)}, y^T) \\
&= \lim_{N\to\infty} \frac{1}{N} \sum_{i\in\text{central}} \delta_{\xi_{i/N}}(x_t)p(x_{t+1}^{[i]}|\xi_{i/N}, y^T) \\
&= \lim_{N\to\infty} \frac{1}{N} \sum_{i\in\text{central}} p(\tilde{x}_t^{(i)}|y^T)p(x_{t+1}^{[i]}|\xi_{i/N}, y^T)
\end{aligned}$$

Hence, as $N \to \infty$, $\{x_{t:t+1}^{(i)}\}_{i=1}^{N} \sim \{\tilde{x}_t^{(i)}, x_{t+1}^{[i]}\}_{i=1}^{N}$.

For the purposes of Kernel Smoothing, this allows the binding together of one step filter smoothing distributions $\dot{p}(x_{t:t+1}|y^T)$ for $t = 1 : T-1$ to form $\dot{p}(x^T|y^T)$. Also, while the proof here implies that x_t is univariate, marginal binding does generalize to multivariate distributions by replenishing each dimension of x_t one at a time.

Bibliography

[1] Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle Markov Chain Monte Carlo Methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

[2] S. Banerjee, B. P. Carlin, and A. E. Gelfand. *Hierarchical Modeling and Analysis for Spatial Data*. Chapman & Hall, London, 2004.

[3] Julian Besag. Spatial Interaction and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):192–236, 1974.

[4] Julian Besag and Charles Kooperberg. On Conditional and Intrinsic Autoregressions, 1995.

[5] Mark Briers, Arnaud Doucet, and Simon Maskell. Smoothing Algorithms for State-Space Models. Technical report, in Submission IEEE Transactions on Signal Processing, 2004.

[6] O. Cappe, S.J. Godsill, and E. Moulines. An Overview of Existing Methods and Recent Advances in Sequential Monte Carlo. *Proceedings of the IEEE*, 95(5):899 –924, May 2007.

[7] C. K. Carter and R. Kohn. On Gibbs Sampling for State Space Models. *Biometrika*, 1994.

[8] Carlos M. Carvalho, Michael Johannes, and Hedibert F. Lopes. Particle Learning and Smoothing. *Statistical Science*, 25(1):88–106, 2010.

[9] M. Carvalho, Hedibert F. Lopes, Nicholas G. Polson, and Matthew A. Taddy. Particle Learning for General Mixtures. *Bayesian Analysis*, 5(4):709–740, 2010.

[10] Dan Crisan and Arnaud Doucet. Convergence of Sequential Monte Carlo Methods. Technical report, Sequential Monte Carlo Methods in Practice, 2000.

[11] R. Douc, E. Moulines, and J. Olsson. On the Auxiliary Particle Filter. *unknown*, 2007.

[12] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On Sequential Monte Carlo Sampling Methods for Bayesian Filtering. *Statistics and Computing*, 10(3):197–208, 2000.

[13] Paul Fearnhead, David Wyncoll, and Jonathan Tawn. A Sequential Smoothing Algorithm with Linear Computational Cost. *Biometrika*, 97(2):447–464, 2010.

[14] Sylvia Fruhwirth-Schnatter. Data Augmentation and Dynamic Linear Models. *Journal of Time Series Analysis*, 1994.

[15] Simon J. Godsill, Arnaud Doucet, and Mike West. Monte Carlo Smoothing for Nonlinear Time Series. *Journal of the American Statistical Association*, 99(465):156–168, 2004.

[16] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation. *Radar and Signal Processing, IEE Proceedings F*, 140(2):107 –113, apr 1993.

[17] Ajay Jasra, David A. Stephens, and Christopher C. Holmes. On Population-Based Simulation for Static Inference. *Statistics and Computing*, 17:263–279, September 2007.

[18] Michael Johannes and Nicholas Polson. Particle Filtering and Parameter Learning. 2007.

[19] Mark S. Kaiser and Noel Cressie. The Construction of Multivariate Distributions from Markov Random Fields. *J. Multivar. Anal.*, 73(2):199–220, 2000.

[20] Genshiro Kitagawa. Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models. *Journal of Computational and Graphical Statistics*, 5(1):1–25, 1996.

[21] Michael Lavine. Another Look at Conditionally Gaussian Markov Random Fields. In *Bayesian Statistics 6: Proceedings of the Sixth Valencia International Meeting*, pages 371–387. Clarendon Press, Oxford, 1999.

[22] Jane Liu and Mike 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*, 2001.

[23] Jun S. Liu and Rong Chen. Blind Deconvolution via Sequential Imputations. *Journal of the American Statistical Association*, 90(430):567–576, 1995.

[24] Hedibert F. Lopes, Carlos M. Carvalho, Michael S. Johannes, and Nicholas G. Polson. Particle Learning for Sequential Bayesian Computation, with Discussion. In *Bayesian Statistics 9: Proceedings of the Ninth Valencia International Meeting.* Oxford University Press (in preparation), Oxford, 2011.

[25] Simon Maskell, Matthew Orton, and Neil Gordon. Efficient Inference for Conditionally Gaussian Markov Random Fields.

[26] Giovanni Petris, Sonia Petrone, and Patrizia Campagnoli. *Dynamic Linear Models with R.* useR! Springer-Verlag, New York, 2009.

[27] Michael K. Pitt and Neil Shephard. Filtering via Simulation: Auxiliary Particle Filters. *Journal of the American Statistical Association*, 94(446):590–599, 1999.

[28] Raquel Prado and Mike West. *Time Series: Modeling, Computation, and Inference.* Chapman & Hall/CRC Texts in Statistical Science, 2010.

[29] R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.

[30] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[31] H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Applications*, volume 104 of *Monographs on Statistics and Applied Probability.* Chapman & Hall, London, 2005.

[32] Havard Rue. Fast Sampling of Gaussian Markov Random Fields. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 63(2):325–338, 2001.

[33] Geir Storvik. Particle Filters for State Space Models With the Presence of Static Parameters, 2002.

[34] Mike West and Jeff Harrison. *Bayesian Forecasting and Dynamic Models (2nd ed.).* Springer-Verlag New York, Inc., New York, NY, USA, 1997.