DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2016

Numerical Instability of Particle Learning: a case study

MAX KLINGMANN RÖNNQVIST

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Numerical Instability of Particle Learning: a case study

M A X K L I N G M A N N R Ö N N Q V I S T

Master's Thesis in Mathematical Statistics (30 ECTS credits)
Master Programme in Applied and Computational Mathematics (120 credits)
Royal Institute of Technology year 2016
Supervisor at KTH: Jimmy Olsson
Examiner: Jimmy Olsson

TRITA-MAT-E 2016:47
ISRN-KTH/MAT/E--16/47-SE

Royal Institute of Technology
School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden

URL: www.kth.se/sci

Numerical Instability of Particle Learning: a case study

Abstract

This master's thesis is about a method called *Particle Learning* (PL) which can be used to analyze so called *hidden Markov models* (HMM) or, with an alternative terminology, *state-space models* (SSM) which are very popular for modeling time series. The advantage of PL over more established methods is its capacity to process new datapoints with a constant demand on computational resources but it has been suspected to suffer from a problem known as particle path degeneracy. The purpose with this report is to investigate the degeneracy of PL by testing it on two examples. The results suggest that the method may not work very well for long time series.

Numerisk instabilitet i Particle Learning: en fallstudie

Sammanfattning

Detta examensarbete handlar om en metod som kallas *Particle Learning* (PL) som kan användas för att analysera *dolda Markovmodeller* eller *hidden Markov models* (HMM), vilka med en alternativ terminologi även kallas *tillståndsmodeller*, som är mycket populära för att modellera tidsserier. Fördelen med PL över mera etablerade metoder är dess förmåga att bearbeta nya datapunkter med konstant behov av beräkningskapacitet men den har även misstänkts lida av ett problem känt som är känt som degenerering av partikelbanorna. Syftet med denna rapport är att undersöka degenereringen av PL genom att testa den på två exempel. Resultaten tyder på att metoden inte fungerar så bra för långa tidsserier.

1 Introduction

A general *hidden Markov model* (HMM) consists of a Markov chain called the state process which is not directly observable. The Markov chain is however partially observed through a sequence of observations, y. The *states* are called x and the *observations* are called y in this report and a graphical representation of the model can be found in figure 1. An HMM is defined

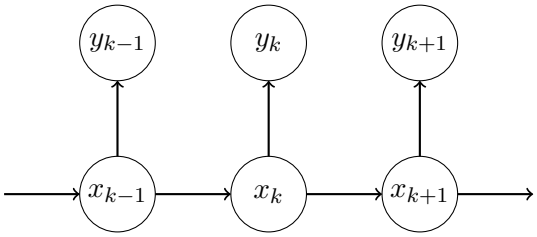

Figure 1: A graphical description of a Hidden Markov model or state space model.

by the transition densities $p(x_{k+1}|x_k)$ and $p(y_k|x_k)$ which often belong to a known parametric family. A simple case is when $x_{t+1} \sim N(x_t, \tau^2)$ and $(y_{t+1}|x_{t+1}) \sim N(x_{t+1}, \sigma^2)$ where σ and τ are unknown and needs to be estimated.

The focus in this report is on methods to estimate the unknown parameters using *Bayesian statistics*. Bayesian statistics requires that the experimenter formulates his or her belief about the unknown quantities in the form of a probability distribution before any data is received. The end result is then a probability distribution called the *posterior* which describes how probable it is that the unknown quantity is inside any interval conditionally on the data. It is often impossible to calculate the posterior analytically which means that a class of simulation techniques called *Markov chain Monte Carlo* (MCMC) must be used. A description of Bayesian statistics including MCMC can be found in [8] for example.

An analytically method called the *Kalman filter* can be used to estimate the hidden states in a specific type of HMMs called *dynamic linear model* (DLM) under the condition that there are no unknown parameters. A method known as *forward filtering backwards sampling* (FFBS) was developed in [1], [7] and [13] which can be used for DLMs with unknown parameters. FFBS is based on a type of *data augmentation* where the unknown parameters are treated as latent variables and an MCMC algorithm called a *Gibbs sampler* is used to estimate the joint posterior of the states and the

parameters. FFBS is however not suitable for applications where data needs to be processed sequentially. The reason for this is that all datapoints needs to be reprocessed from the beginning for each new datapoint.

Particle Learning (PL) is a method that can be used to estimate the parameters online which means that the CPU and memory requirement do not increase with each new data point. The method is applied to DLMs in [3] and a more general description can be found in [4]. PL is based on a class of methods called *particle filters* which are treated in great detail in [2]. Particle filters is a simulation technique which, like the Kalman filter, is typically used to estimate states in HMMs without unknown parameters but PL is a way to estimate the parameters as well. The advantage of particle filters over the kalman filter is that they can be used for a much wider range of models.

A problem with particle filters is that they typically give a poor estimate of the states that are in the beginning of the observed time period. This problem is referred to as particle path degeneracy and is discussed in [11] for example. The purpose with this master thesis is to test PL on two different DLMs to see how large impact the degeneracy have on the results. The results show that PL may indeed lead to a high variability between different runs of the algorithm which can be taken as a sign of degeneracy.

2 Background

This background section begins by giving a short introduction to Bayesian statistics. It then proceeds with a description of the models studied in this report followed by a section about the Kalman filter and another section about particle filters. Then follows a description about how to calculate the posterior of the parameters when the states are already known which is useful in the last section about Gibbs sampling and FFBS.

2.1 Bayesian Statistics

A fundamental problem in statistics is to estimate unknown quantity's from data involving random errors. There will always be some uncertainty in the estimation and Bayesian statistics describes this uncertainty in the form of random variables and their probability distributions. This is possible due to a formula called *Bayes's formula* which states that

$$f(x|y) = \frac{f(y|x)f(x)}{f(y)}, \tag{1}$$

where

$$f(y) = \int f(y|x)f(x)\mathrm{d}x.$$

Assume that a statistician has some data y and wish to estimate x using Bayesian statistics. The first step is to build a model that explains how x is related to y i.e. define the *likelihood* $f(y|x)$. This is not much different from the more common form of statistics which is referred to as *frequntist* statistics. The second step is to define the so called *prior* $f(x)$ which is the probability distribution of the unknown quantity x *before* any data is analyzed. In Bayesian statistics, the prior should reflect the belief of the experimenter before the current experiment is conducted and could for exemplar be based on previous experiments.

It is now possible to plug in the data y, the prior $f(x)$ and the likelihood $f(y|x)$ into Eqn. (1) in order to calculate the *posterior* $f(x|y)$. The posterior is a probability distribution describing the subjective belief of the experimenter regarding how likely it is that the parameter is in any given interval.

It can sometimes be very hard to calculate the posterior analytically and a very helpful tool is to use so called *conjugated priors*. For certain priors and likelihood functions, the posterior can be shown to be of the same family

as the prior but with different parameters. If this is the case then the prior is called a conjugated prior to the likelihood function and there exists tables with the most common of theses priors which saves a lot of effort. In many other cases however, it is necessary to use simulation based methods to calculate the posterior.

The description above is a very generic introduction to Bayesian statistics so the question now becomes how to apply it to HHMs. Let $y_{1:t}$ denote all observations in the time interval between 1 and t (1 and t are included in the interval) and similarly let $x_{1:t}$ denote all the hidden states in the interval between 1 and t. The prior for the first step must be chosen by the statistician. For the rest of the states however can the prediction density $f(x_{t+1}|y_{1:t})$ be used as a prior for x_{t+1}. It turns out that there are are two posteriors that are of particular interest and these are the *filtering distribution* and the *smoothing distribution*. Assume that T measurements have been recorded. The filter distribution $\psi(x_t)$ can intuitively be understood as the estimation of x_t given all data up to the time t. The smoothing distribution $\chi(X_t)$ on the other hand is an estimation of x_t given all data between 1 and T including data that arrives after t. This can be formulated by the following formulas:

$$\begin{aligned}\chi(X_t) &= f(X_t|y_{1:t}),\\ \psi(X_t) &= f(X_t|y_{1:T}).\end{aligned}$$

2.2 Dynamic Linear Models

DLMs is the only type of HHM treated in this thesis. In a DLM, the states evolve linearly in time with a normally distributed error with zero mean and the observations are also linear functions of the states with the same type of error. This can be expressed in formulas in the following way

$$\begin{aligned}y_{t+1}|x_{t+1},\theta &\sim N(F_t x_{t+1}, V),\\ x_{t+1}|x_t,\theta &\sim N(G_t x_t, W),\end{aligned} \tag{2}$$

where y_t and x_t are vectors and F_t, G_t, V, W are matrices in the general case. The index t will be dropped on G_t and F_t from here on if they do not change in time. The vector θ is defined as the unknown elements of V and W. The matrices F_t and G_t are assumed to be known in this thesis and section 2.3 includes examples of DLMs where this is indeed the case.

The prior for the first state x_0 is the normal distribution with mean m_0 and variance C_0. The prior for the variances V and W depends on the application. This thesis will only include models where V and W are

diagonal which makes it possible to treat each component separately. The *inverse gamma distribution* is used as a prior for each unknown parameter. The probability density function of the $IG(\alpha, \beta)$ distribution is

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{-\alpha-1} \exp\left(-\frac{\beta}{x}\right),$$

where α is known as the *shape parameter*, $\beta > 0$ is known as the *scale parameter* and Γ denotes the gamma function.

The inverse gamma prior is particularly useful for analyzing data which is sampled from a normal distribution with known mean μ but unknown variance. If the prior is $IG(a, b)$ and there are n such samples denoted $x_1, \ldots, x_n$, then the posterior is $IG(\hat{a}, \hat{b})$ where

$$\hat{a} = a + \frac{n}{2} \text{ , and } \hat{b} = b + \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{2} . \tag{3}$$

This relation will turn out to be useful in the context of DLMs.

2.3 Exemples of DLMs

The model presented in section 2.2 has many applications and the purpose of this section is to introduce the reader to some simple time series models which all can be reformulated into the type of model given in Eqn. (2). They are all taken from [6] or [12] which contain even more examples. More complex time series models can also be constructed by combining combining different smaller models to form large models.

2.3.1 Local Level Model

The most basic DLM is sometimes called the *local level model*. It is defined in the following way:

$$\begin{aligned} y_{t+1} &\sim N(x_{t+1}, \sigma^2), \\ x_{t+1} &\sim N(x_t, \tau^2). \end{aligned} \tag{4}$$

It is possible to make an analogy to linear regression here. The local level model would then be a regression model with only an intercept and no explanatory(independent) variables. It is useful on its own but is very often combined with other models.

2.3.2 Seasonal components

There are at least two ways to model seasonal effects in the DLM framework. In [12] they are called *Seasonal factor models* and *Fourier form seasonal models*. This report will only use the latter of the two.

Imagine a periodic function $g(t)$ defined at equidistant times. Also assume that each period consists of s different time points. All such functions form a vector space. One choice of basis for this vector space is s periodic functions that have the value zero at all time points on the interval 0 to s except for at one time. There are in total s such functions that are linearly independent. The vector space of all possible $g(t)$ will therefore have the dimension s. Fourier series use another base of this vector space.

A Fourier series is a way to represent periodic functions as a sum of sine and cosine waves with increasing frequency. The (angular) frequencies of the waves are denoted η_j and are defined as:

$$\eta_j = \frac{2\pi j}{s}.$$

The terms in the sum are denoted

$$S_j(t) = a_j \cos(t\eta_j) + b_j \sin(t\eta_j),$$

where a_j and b_j are constants that can be be calculated. $S_j(t)$ is called the jth harmonic of $g(t)$. It is now possible to represent every periodic function $g(t)$ defined on a discrete time set as

$$g(t) = \sum_j S_j(t), \tag{5}$$

since the functions $\cos(t\eta_j)$ and $\sin(t\eta_j)$ as well as $S_j(t)$ can be shown to be orthogonal (and hence linearly independent) with respect to a inner product. Only the first s harmonics are needed since the dimension of the vector space is s. The zeroth harmonic is

$$\begin{aligned} S_0(t) &= a_0 \cos(t\eta_0) + b_0 \sin(t\eta_0) \\ &= a_0 \cos(0) + b_0 \sin(0) \\ &= a_0, \end{aligned}$$

since $\eta_0 = \frac{2\pi \cdot 0}{s} = 0$. This is a constant not depending on t and the interest here is to model purely seasonal effects. Therefore, a_0 is set to zero.

It is possible to describe arbitrary functions in this way. In this context however, it is desired to find functions that are quite smooth. One way to archive this is to only include a few harmonics.

The hidden states in Fourier form seasonal models are the *harmonics* $S_j(t)$. The coefficients a_j and b_j are never calculated explicitly. The states in DLM must be linear functions of the previous state. This means that $S_j(t+1)$ must be calculated from $S_j(t)$. This is unfortunately not possible without some additional knowledge. For this reason, the *conjugated harmonics* are introduced. They are defined as:

$$S_j^*(t) = -a_j \sin(t\eta_j) + b_j \cos(t\eta_j).$$

It is now possible to express the time evolution of $S_j(t)$ and $S_j^*(t)$ as a linear function. To be more specific:

$$\begin{pmatrix} S_j(t+1) \\ S_j^*(t+1) \end{pmatrix} = \begin{pmatrix} \cos(\eta_j) & \sin(\eta_j) \\ -\sin(\eta_j) & \cos(\eta_j) \end{pmatrix} \begin{pmatrix} S_j(t) \\ S_j^*(t) \end{pmatrix}.$$

For a derivation of these equations see equation (3.20) in [12].

2.3.3 Regression components

This section deals with a modification of the standard linear regression problem. Assume that we are interested in how some variables $u_1, u_2, \ldots, u_n$ affect y. The standard linear regression model is then

$$y_t = \beta_1 u_{1,t} + \ldots + \beta_n u_{n,t} + v_t \quad v_t \sim N(0, \sigma^2).$$

The coefficients $\beta_1, \beta_2, \ldots, \beta_n$ are fixed in time and should be calculated, and the values of y_t as well as all $u_{1,t}, u_{2,t}, \ldots, u_{n,t}$ are known.

An alternative approach suggested in [12] and [6] is to model $\beta_1, \beta_2, \ldots, \beta_n$ as a random walk. This can be formulated as a DLM by setting $x_t = \begin{pmatrix} \beta_1 & \beta_2 & \ldots & \beta_n \end{pmatrix}^T$, $F_t = \begin{pmatrix} 1 & u_{1,t} & u_{2,t} & \ldots & u_{n,t} \end{pmatrix}$ and G is the identity matrix.

2.3.4 Combination of models

Several smaller models can be combined to form larger model. Assume for instance that a local level model defined by

$$(y_{t+1}|\beta_{0,t+1}) \sim N(\beta_{0,t+1}, V_1),$$
$$(\beta_{0,t+1}|\beta_{0,t}) \sim N(\beta_{0,t}, W_1),$$

should be combined with a regression model with only one explanatory variable $u = (u_1, \ldots, u_T)$ defined by:

$$(y_{t+1}|\beta_{1,t+1}) \sim N(u_t\beta_{1,t+1}, V_2),$$
$$(\beta_{1,t+1}|x_t) \sim N(\beta_{1,t}, W_2).$$

Theses models can be combined by defining the states x_t as

$$x_t = \begin{pmatrix} \beta_{1,t} \\ \beta_{2,t} \end{pmatrix},$$

which together with the matrices

$$F_t = \begin{pmatrix} 1 & u_t \end{pmatrix} \quad \text{and} \quad G = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

and the new variance matrices V and W given by

$$V = V_1 + V_2 \quad \text{and} \quad W = \begin{pmatrix} W_1 & 0 \\ 0 & W_2 \end{pmatrix},$$

defines a new DLM.

2.4 Kalman filter

An advantage with DLMs over other HHMs is that all relevant distribution including the filter and smoothing distribution can be shown to be Gaussian as long as there is no unknown parameters in the model (i.e F_t, G_t, V, W are known) which simplifies the computations significantly. It is sufficient to calculate the mean and variance of the distribution that should be calculated since the normal distribution is uniquely defined by those values.

The method for calculating the mean m_t and the variance of the filter distribution is called the Kalman filter. The derivation of the Kalman filter makes extensive use of the following property of Gaussian distributions. If a vector $\mathbf{x}$ has the distribution $N(\mu, \mathbf{\Sigma})$ and $\mathbf{y} = \mathbf{B}\mathbf{x} + \mathbf{c}$ then $\mathbf{y}$ will have the distribution $N(\mathbf{c} + \mathbf{B}\mu, \mathbf{B}\mathbf{\Sigma}\mathbf{B}^T)$.

The Kalman filter can be viewed as a three step procedure. The first step is to calculate the distribution of $X_t|Y_{1:t-1}$ which is Gaussian with mean a_t and variance R_t given by:

$$a_t = G_t m_{t-1},$$
$$R_t = G_t C_{t-1} G_t^T.$$

The value of m_{t-1} and C_{t-1} comes from the previous iteration of the Kalman filter. The second step is to calculate the distribution of $Y_t|y_{1:t-1}$ which is also Gaussian with mean f_t and variance Q_t given by:

$$\begin{aligned} f_t &= F_t a_t, \\ Q_{t+1} &= F R_{t+1} F^T + V. \end{aligned}$$

The third step is more complicated and involves the so-called Kalman gain $A_{t+1} = R_{t+1} F^T Q_{t+1}^{-1}$ and the forecasting error $e_t = Y_t - f_t$. The final result is provided by these formulas:

$$\begin{aligned} m_{t+1} &= G m_t + A_{t+1}(y_{t+1} - e_t), \\ C_{t+1} &= R_{t+1} - A_{t+1} F_{t+1} R_{t+1}. \end{aligned}$$

For the local level model can the Kalman filter be simplifyed to the following formulas:

$$\begin{aligned} A_{t+1} &= \frac{C_t + \tau^2}{C_t + \tau^2 + \sigma^2}, \\ m_{t+1} &= (1 - A_{t+1}) m_t + A_{t+1} y_{t+1}, \\ C_{t+1} &= A_{t+1} \sigma^2. \end{aligned} \tag{6}$$

Both the PL algorithm and FFBS require sampled paths from the smoothing distribution which is known as sampling from the *backward kernel*. Assume that $y_{1:T}$ has been filtered with the Kalman filter. The last state X_T can then be sampled from $N(m_T, C_T)$ since the filter and smoothing distribution is identical for the time T. The goal is then to recursively sample the rest of the states from the distribution of $X_t|X_{t+1}, Y_{1:T}$. Due to a property called *conditional independence* is $X_t|x_{t+1}, y_{1:T} = X_t|x_{t+1}, y_{1:t}$ which also has a Gaussian distribution with the mean m_t^T and variance C_t^T given by

$$\begin{aligned} m_t^T &= m_t + C_t G^T R_{t+1}^{-1}(x_{t+1} - G m_t), \\ C_t^T &= C_t - C_t G^T R_{t+1}^{-1} G C_t. \end{aligned}$$

2.5 Particle filters

Particle filters are used when the state process is nonlinear or the error terms are non-Gaussian. They do not solve all problems however. Both the Kalman filter and particle filters require that the parameters in the transition density and the observation density are fully known.

The key idea behind particle filters is to replace the true filter distribution $\phi(X_t)$ with an discrete representation. Assume that we have N samples x_t^i with an associated *weight* ω_t^i from the true filter distribution. Each sample is then called a *particle* and $\phi(X_t)$ is approximated with

$$\hat{\phi}(X_t) = \frac{1}{\sum_{i=1}^{N} \omega_t^i} \sum_{i=1}^{N} \omega_t^i \delta_{x_t^i}(X_t),$$

where $\delta(x)$ denotes the Dirac delta function.

Several algorithms exist for updating the particles as new data arrives and one of the most basic particle filters is described in [9]. The algorithm is initialized by sampling N particles from the prior and the algorithm consists of two steps that are repeated for each new data-point. Assume that there is a particle approximation available for the time t where all weights are equal to 1. The first step is then to sample x_{t+1}^i from the transition density $p(X_{t+1}^i | x_t^i)$ for all particles. Each new particle is then assigned the weight:

$$\omega_{t+1}^i = \frac{p(y_{t+1} | x_{t+1}^i)}{\sum_{j=1}^{N} p(y_{t+1} | x_{t+1}^j)}.$$

The new particles and their associated weights can be used to form a new particle approximation to the filtering distribution for the time $t + 1$.

It is possible to convert this weighted sample to a sample where the weights are equal to 1 through a process known as *resampling*. Resampling means that N particles are drawn with replacement from a weighted sample by randomly selecting particles with a probability equal to their weight. So if particle x_{t+1}^i has the weight ω_{t+1}^i, it should be selected with the probability ω_{t+1}^i and one particle can end up being selected several times. It is possible to construct algorithms without a resampling step but the problem is that most particles then end up with very low weight which negatively affects the performance. Resampling is introduced to solve this problem.

All particle filters perform worse than the Kalman filter for linear and Gaussian state-space models since these rely on an approximation and stochastic simulations whereas the Kalman filter is an analytical method. It is therefore not clear at first sight how they could be helpful in the linear and Gaussian setting.

2.6 Posterior of parameters

The goal with this section is to describe how to calculate the posterior of the parameters θ in a DLM if the states are known. This might not be

very realistic but the it will be useful as a building block for more advanced algorithms like a Gibbs sampler or PL. The key point here is that $x_{t+1} \sim N(Gx_t, W)$ and $y_{t+1} \sim N(Fx_{t+1}, V)$.

The variable y_t is a scalar in this report which means that $N(Fx_{t+1}, V)$ is a univariate Gaussian distribution and, since the inverse gamma prior is used, the posterior of V will be inverse gamma with the shape parameter a_n and scale parameter b_n given by

$$\begin{aligned} a_n &= a_0 + n/2, \\ b_n &= b_0 + \frac{\sum_{i=1}^{n}(y_i - F_i x_i)^2}{2}, \end{aligned}$$

where a_0 and b_0 is the parameters of the prior. The distribution $N(Gx_t, W)$ is a multivariate Gaussian distribution for a general DLM but since W is a diagonal matrix with independent inverse gamma priors, it is possible to use the same method to calculate the posteriors of each diagonal element of W. Let c_0 be vector containing all the shape parameters of the priors and let d_0 a vector containing all the scale parameters of the priors. The shape and scale of the posteriors will then be

$$\begin{aligned} c_n &= c_0 + 1 \cdot \frac{n-1}{2}, \\ d_n &= d_0 + \frac{1}{2} \cdot \sum_{i=1}^{n-1} \left(x_{n+1} - Gx_n\right)^2, \end{aligned}$$

where the exponent 2 means that each *element* of vector should be squared. Note that it is possible to update these posteriors recursively as new data arrives with the help of the following equations

$$\begin{aligned} a_{n+1} &= a_n + 1 \cdot \frac{1}{2}, \\ b_{n+1} &= b_n + \frac{(y_{n+1} - Fx_{n+1})^2}{2}, \\ c_{n+1} &= c_n + 1 \cdot \frac{1}{2}, \\ d_{n+1} &= b_n + \frac{(x_{n+1} - Gx_n)^2}{2}, \end{aligned} \tag{7}$$

where the exponent 2 once again should be taken elementwise.

2.7 Gibbs sampler and FFBS

The most basic Monte Carlo algorithm is used to calculate the expected value for random variables of the form $h(U)$ where h is a known function

and U is a random variable with the distribution $f(U)$ i.e calculate:

$$E(h(u)) = \int h(u)f(u)\ \mathrm{d}u. \tag{8}$$

The value of $E(h(U))$ can be approximated by sampling N independent samples of U denoted $u^1, u^2, \ldots, u^N$. The integral in Eqn. (8) can then be approximated by the sum:

$$E(h(U)) \approx \sum_{k=1}^{N} h(u^k)f(u^k). \tag{9}$$

It is often difficult to sample independent samples from the right distributions. MCMC methods is a way to sample Markov chains in such a way that Eqn. (9) still is a good approximation despite the dependence between the samples.

The Gibbs sampler is an MCMC algorithm used to obtain samples from stochastic variable of the form $U = (U_1, U_2, \ldots, U_n)$ where each sample is denoted $u^k = (u_1^k, u_2^k, \ldots, u_n^k)$. Each U_i is referred to as a *block* and it can be both a vector or a scalar. The Gibbs sampler is useful when it is possible to sample the individual blocks conditional on all the other blocks. The first sample is chosen arbitrary and the rest of the samples are sampled recursively using the following algorithm:

1. Sample u_1^{k+1} from $f(U_1|U_2 = u_2^k, U_3 = u_3^k \ldots, U_n = u_n^k)$.
2. Sample u_2^{k+1} from $f(U_2|U_1 = u_1^{k+1}, U_3 = u_3^k \ldots, U_n = u_n^k)$.
   $\vdots$
3. Sample u_n^{k+1} from $f(U_n|U_1 = u_1^{k+1}, U_2 = u_2^{k+1} \ldots, U_{n-1} = u_{n-1}^{k+1})$.

The first samples obtained this way are typically discard since they can be very influenced by the starting state. This is referred to as burn-in.

The FFBS algorithm for DLMs is a Gibbs sampler targeting the joint posterior of the states and the parameters denoted $f(x_{1:T}, \theta|y_{1:T})$. The Gibbs sampler is very useful since it is possible to sample both the states $X_{1:T}|\theta, y_{1:T}$ and the parameters $\theta|x_{1:T}$ using the methods in section 2.4 and 2.6 respectively. The FFBS algorithm has the following two steps.

1. Sample the states $x_{1:T}^k$ from $X_{1:T}|Y_{1:t} = y_{1:T}, \theta = \theta^{k-1}$ using the backwards kernel.
2. Sample the parameters θ^k from $\theta|X_{1:T} = x_{1:T}^k$.

3 Particle Learning

3.1 Basic idea

A suggestion for finding the parameters would be to treat them as a state and then perform filtering in a higher dimension. Unfortunately, particle filters can not handle states that do not change in time and the unknown variances are indeed fixed in time. Consider for example the particle filter in section 2.5 where new particles are sampled from the transition density $p(x_{t+1}^i|x_t^i)$. If $p(x_{t+1}^i|x_t^i)$ is designed in such a way that $x_t^i = x_{t+1}^i$ with probability 1, then the only source of randomness is then in the first time-step. There is no way of ending up with particles that have a value different then the original sample and there will be a smaller amount of unique values as the time goes on due to the resampling.

The PL algorithm circumvents this problem by propagating a set of sufficient statistics instead of the parameters and states themselves. All of the variables needed for updating each statistic are stored in what is called an *essential state vector* denoted z_t.

3.2 The essential state vector

The PL algorithm is basically a particle filter targeting the essential state vector z_t instead of the hidden state themselves. The vector z_t has three components in the case of DLMs and they are denoted s_t, s_t^x and θ_t. The vector θ_t is defined as the Bayesian estimate of θ, i.e. $\theta_t = \theta|x_{1:t}$, which is a random variable unlike the real θ. This makes it possible to estimate z_t with help of particle filters but it will be problematic when it comes to smoothing which will be discussed in section 3.6.

The vector s_t^x contains the mean and variance of the posterior for the hidden states conditional on θ at the time t provided by the Kalman filter. The vector s_t contains all information about the states that is required to calculate the posterior for the parameters and is define as $s_t = \begin{pmatrix} a_t & b_t & c_t & d_t \end{pmatrix}$ for the local level model.

3.3 The filter recursions

The question now is how to calculate the posterior of z_t conditional on all data. This should be made in a sequential manner, i.e., the posterior of z_{t+1} should be calculated from the posterior of z_t and y_{t+1}. The first step in the

derivation of the filter used in PL is given by the following formula:

$$p(z_{t+1}|y_{1:t+1}) = \int p(z_{t+1}|z_t, y_{1:t+1})p(z_t|y_{1:t+1}) \, \mathrm{d}z_t.$$

Now use that z_t is a sufficient statistic which leads to:

$$p(z_{t+1}|y_{1:t+1}) = \int p(z_{t+1}|z_t, y_{t+1})p(z_t|y_{1:t+1}) \, \mathrm{d}z_t.$$

Bayes's formula now gives that

$$\begin{aligned} p(z_t|y_{1:t+1}) &= p(z_t|y_{t+1}, y_{1:t}) \\ &\propto p(y_{t+1}|z_t, y_{1:t})p(z_t|y_{1:t}) \\ &\propto p(y_{t+1}|z_t)p(z_t|y_{1:t}), \end{aligned}$$

which in turn leads to the following formula:

$$p(z_{t+1}|y_{1:t+1}) \propto \int p(z_{t+1}|z_t, y_{t+1})p(y_{t+1}|z_t)p(z_t|y_{1:t}) \, \mathrm{d}z_t. \tag{10}$$

It is now assumed that there already exists a particle approximation of $p(z_t|y_{1:t})$ where all the weights are equal to 1 i.e:

$$\hat{p}(z_t|y_{1:t}) = \frac{1}{N}\sum_{i=1}^{N} \delta_{z_t^i}(z_t). \tag{11}$$

Eqn. (11) is now inserted into Eqn. (10):

$$\begin{aligned} p(z_{t+1}|y_{1:t+1}) &\propto \frac{1}{N}\int \left(p(z_{t+1}|z_t, y_{t+1})p(y_{t+1}|z_t) \sum_{i=1}^{N} \delta_{z_t^i}(z_t) \right) \, \mathrm{d}z_t \\ &\propto \frac{1}{N}\sum_{i=1}^{N} p(z_{t+1}|z_t^i, y_{t+1})p(y_{t+1}|z_t^i) \\ &\propto \sum_{i=1}^{N} p(z_{t+1}|z_t^i, y_{t+1})p(y_{t+1}|z_t^i). \end{aligned} \tag{12}$$

The goal is now to sample z_{t+1}^i from Eqn. (12) by using a technique from [10]. The trick is to sample from

$$p(z_{t+1}, i|y_{1:t+1}) \propto p(z_{t+1}|z_t^i, y_{t+1})p(y_{t+1}|z_t^i), \tag{13}$$

and discard the index i instead of sampling from $p(z_{t+1}|y_{1:t+1})$ directly. The marginal density of i can be calculated by using Eqn. (13) and the fact that integrals over probability density function must be equal to 1. The exact formulas are:

$$\begin{aligned} g(i) &\propto \int p(z_{t+1}, i|y_{1:t+1}) \, \mathrm{d}z_{t+1} \\ &\propto \int p(z_{t+1}|z_t^i, y_{t+1}) p(y_{t+1}|z_t^i) \, \mathrm{d}z_{t+1} \\ &\propto p(y_{t+1}|z_t^i) \underbrace{\int (z_{t+1}|z_t^i, y_{t+1}) \, \mathrm{d}z_{t+1}}_{=1} \\ &\propto p(y_{t+1}|z_t^i). \end{aligned}$$

This makes it possible to write Eqn.(13) as

$$p(z_{t+1}, i|y_{1:t+1}) \propto g(z_{t+1}|i)g(i), \tag{14}$$

where $g(z_{t+1}|i) = p(z_{t+1}|z_t^i, y_{t+1})$. The advantage with this formulation is that it provides a straight forward way to sample z_{t+1}. The first step is to sample i with a probability proportional to $g(i) = p(y_{t+1}|z_t^i)$ and the second step is then to use z_t^i to sample z_{t+1} from $p(z_{t+1}|z_t^i, y_{t+1})$. This of course requires that it is possible to sample from $y_{t+1}|z_t$ and $z_{t+1}|z_{t,m}, y_{t+1}$. A particle approximation of $p(z_{t+1}|y_{1:t+1})$ can finally be formed by sampling N new particles using this method which makes it possible to start the method all over again for the next data-point.

The process of producing N samples from $g(i)$ is very similar to the resampling step in the particle filter described in section 2.5. The entire algorithm can actually be summarized in the following way:

1. For each i, assign the particle z_t^i the weight $\omega_i = \frac{p(y_{t+1}|z_t^i)}{\sum_j p(y_{t+1}|z_t^j)}$.
2. Resample all particles according to their weight.
3. For each of the particles from step 2, sample a new particle from $p(z_{t+1}|z_t^j, y_{t+1})$ where j is the index of the old particle.
4. Create a particle approximation to $p(z_{t+1}|y_{1:t+1})$ by using the particles from step 3.

3.4 PL applied to DLMs

Section 3.3 provides an algorithm for updating z_t for a general essential state vector. It remains to adapt the method to the essential state vector used for DLMs which is described in section 3.2. The first step is to calculate the *prediction density* $p(y_{t+1}|z_t^i)$ which is used inside the Kalmanfilter. The second step, to sample z_{t+1} from $p(z_{t+1}|z_t^i, y_{t+1})$ is more difficult.

3.4.1 Resampling step

The prediction density $p(y_t|z_{t-1})$ for the general DLM can be found in section 2.4 and is given by

$$\begin{aligned} y_t|z_{t-1} &= y_t|m_{t-1}, C_{t-1}, \theta_{t-1} \\ &\sim N(FGm_{t-1}, FR_tF^T + V), \end{aligned}$$

where $R_t = GC_{t-1}G + W$. The formulas for the local level model are

$$\begin{aligned} y_t|z_{t-1} &= y_t|s_{t-1}^x, s_{t-1}, \theta_{t-1} \\ &\sim N(m_{t-1}, C_{t-1} + \tau_{t-1}^2 + \sigma_{t-1}^2), \end{aligned}$$

which is just a special case of the more general formula.

3.4.2 Propagation step

The problem in the propagation step is how to sample a new particle from y_t and z_{t-1}^i for a given i. This process is repeated for all i but the index i is dropped in this section in order to simplify the notation. Remember that $z_t = (s_t^x, s_t, \theta_t)$ where s_t^x is the output of the Kalman filter and s_t is the statistic for the fixed variances.

1. Calculate s_t^x from s_{t-1}^x, θ_{t-1} and y_t by using the Kalmanfilter.
2. Sample $(x_{t-1}, x_t|s_{t-1}^x, \theta_{t-1}, y_t)$.
3. Calculate s_t from x_{t-1}, x_t and s_{t-1}.
4. Sample $\theta_t|s_t$.

The values x_t and x_{t-1} are sampled by noting that $(x_{t-1}, x_t|s_{t-1}^x, \theta_{t-1}, y_t) = (x_{t-1}, x_t|s_{t-1}^x, s_t^x, \theta_{t-1})$ which means it is possible to first sample $(x_t|s_{t-1}^x, s_t^x, \theta_{t-1}) = x_t|s_t^x \sim N(m_t, C_t)$ and then sample from $x_{t-1}|x_t, s_{t-1}^x, s_t^x, \theta_{t-1}$ using the backward kernel. The values of x_{t-1} and x_t are only used in step 3 above and are then discarded. Step 3 is described closer in section 2.6.

3.5 Implementation details

The Kalman filter and smoother have a numerical problem which is not mentioned in [3] or [4]. There will allways be some truncation errors in the calculations since computers have a finite precision and the variance matrix C_t is particulary sensetive to these errors. The author of this master thesis has chosen to tackle this problem by using the methods described in [15] which are algebraically equivalent to the original Kalman filter and smoother but more numerically stable.

The filter in 2.4 takes m_t and C_t as inputs and then calculates m_{t+1} and C_{t+1}. The filter in [15] on the other hand works with the *Singular value Decomposition* (SVD) of C_t. The SVD of a (positive definit) covariance matrix A is a matrix decomposition of the form

$$A = UD^2U^T,$$

where D is a diagonal matrix. Suppose that the SVD of C_t is known i.e. the matrices U_t and D_t are known and $C_t = U_t D_t^2 U_t^T$. The SVD version of the Kalman filter then calculates U_{t+1} and D_{t+1} without the explicit use of C_t or C_{t+1}. The main steps of this process consist of calculating the SVD of some other auxiliary matrices. The full details can be found in [15].

The particle learning algorithm needs to be modified accordingly. The variance matrix C_t should be removed from the essential state vector z and be replaced with U_t and D_t.

3.6 Smoothing

The algorithm above is a way to jointly estimate the hidden states and the unknown variances θ in the model and the estimation of θ is changed in each step. The topic of this section is how to calculate the smoothed posterior for the hidden states. The smoothed estimate of θ is equal to the filter distribution at the last time since θ is a constant in the model and the smoothing distribution and filter distribution are identical at the last time.

Since the models in this thesis are linear and Gaussian it seams natural to try to use the Kalman smoother in some way. The problem is that the states have not yet been filtered using the correct parameters. Another way to approach the problem is to look at smothers used together with particle filters. The article [3] contain a short section about smoothing including a description of the algorithm from [14].

3.6.1 Smoothing and particle filters

The result of a particle filter is a set of particles, typically with weights, that approximates the filter distribution. Let the weight of particle i at time t be denoted $\omega_t^{(i)}$, The last time is denoted T. A sample path from the posterior can then be obtained by first randomly selecting particle i from the filter distribution at the time T with probability $\omega_t^{(i)}$. Then simulate backwards by sampling particles according to weights that are proportional to $\omega_t^{(i)} f(x_{t+1}|x_t)$.

This algorithm follows from Eqn. 4 in [14] which says that:

$$p(x_t|x_{t+1}, y_{1:T}) \propto p(x_t|y_{t:t}) f(x_{t+1}|x_t).$$

Note that the particle filter already provides an approximation to $p(x_t|y_{t:t})$ and that the transition density $f(x_{t+1}|x_t)$ is assumed to be known in this context.

The algorithm as it is described in [14] can be rather time consuming. Assume that there are N particles for each time. It is then necessary to calculate N weights for each time in order to get one sample path from the smoothing distribution and the algorithm has to be repeated N times in order to get the same number of samples as in the filter. The complexity is hence $O(N^2T)$ but there exist an imporved version of the algorithm with a lower complexity which can be found in [5].

3.6.2 Smoothing and Particle Learning

There are two problems with applying the algorithm described in 3.6.1 to PL. The first problem is that the states have been marginalized away and the hidden states only enter the essential state vector z through the statistic s_t^x. This problem is not addressed at all in [3] but several alternatives are conceivable.

One way would be to treat the Kalman estimate as states ie. setting $x_t^i = m_t^{(i)}$, but the question is then what is meant by $f(x_{t+1}|x_t)$. It is possible to use the transition density of the hidden states x here but that would not really be true since m_t is not a hidden Markov chain itself.

It is however possible to sample x_t from each particle. This is actually done as a part of the algorithm and it is therefore possible to save those samples and use them for smoothing. A particle approximation of $p(x_t|y_{1:t}, \theta_t)$ can be obtained in this way but remember that θ_t changes for each time. It is however treated as a constant in [3]. The best estimate of θ is the estimate at time T and it therefore makes sense to calculate $f(x_{t+1}|x_t)$ using θ_t. The

problem is that PL gives a particle approximation of $p(x_t|y_{1:t}, \theta_t)$ instead of $p(x_t|y_{1:t}, \theta_T)$.

The smoothing algorithm will in any case not be online. It is therefore rather meaningless to use PL anyways if the smoothed states are of interest and smoothing is therefore not investigated further in this report.

4 Methods

The PL method is tested on simulated data since all quantities can be chosen by the experimenter and are hence are fully known whereas real data includes unknown parameters. Data is generated by first specifying a model including all parameters and the data is then simulated using a random number generator. The parameters are then estimated which makes it possible to compare the estimation with the true value.

To begin with, the results of PL is tested against a reference algorithm. This ensures that the implementation of PL used to make this report gives correct results and the authors of [12] have created a package for the statistical programming language R which can be used to analyze the types of models in this report. The package provides a Gibbs sampler which can be used as a reference and the output of the PL algorithm is expected to be very similar to the output of the Gibbs sampler.

Some values needs to specified in order to use a Gibbs sampler. To begin with, an initial guess most be provided, and the author of this report have chosen to initialize the Gibbs sampler with the true value. This eliminates the need of using a burn in since the real value should be included in the sample anyway.

The main experiment is however related to how PL performs for longer time series. A problem with particle filters that include a resampling step is that the early history of the particles tend to be very similar between different particles. The reason for this is that only some of the particles are propagated while others are removed from the sampled. The historical paths will therefore collapse to very few values for times that are far away from the current time which are not necessarily the correct ones. This process is called particle path degeneration.

PL is a way to estimate the most recent value of the essential state vector so it might not be obvious at first sight that the particle path degeneracy is a problem for the method. However, remember that the essential state vector includes a statistic s_t which includes all the states up to t.

A sign of degeneration is that the method generates different results for the same data each time the filter is used. The reason for this is that the resampling step is random and the historical paths will therefore collapse onto different paths each time. The degeneracy problem for PL is therefore tested by simulating a relative long data set and filter it several times. If the results are very different each time then the problem is large, on the other hand, if the same result can be reproduced then degeneracy is not a problem.

The number of particles will also have a effect on the degeneracy. If there are many particles to begin with, more paths will survive in the end and the problem will therefor be less visible. The degeneracy of PL will therefore be tested with few particles compared to the length of the data set. No comparison is made with a Gibbs sampler for this experiment but the result can be compared with the true values.

5 Results

In this section, the Particle Learning algorithm from section 3 is tested on some simple time series models. The models are described in detail in section 5.1. The data is generated using the same models as is used to analyze them.

5.1 Models

Two time series models are tested. Model number 1 is the local level model. Model number 2 combines model number 1 with a very simple seasonal factor model that only includes one harmonic. This might not be realistic but it is sufficient to demonstrate the concept.

Recall the definition of the local level model:

$$\begin{aligned} y_{t+1} &\sim N(x_{t+1}, \sigma^2), \\ x_{t+1} &\sim N(x_t, \tau^2). \end{aligned} \tag{15}$$

Data is simulated using $\sigma = \tau = 1$. The initial state of the state process is set to m_0. These parameters are sufficient in order to generate the data. To actually analyze the data, the priors need to be specified as well.

For the states the normal distribution is used with mean m_0 and variance C_0. The values on m_0 and C_0 can be found in Table 1 together with all other values needed for the other priors. The prior for σ is $IG(a_0, b_0)$ and the prior for τ is $IG(c_0, d_0)$.

a_0	b_0	c_0	d_0	m_0	C_0
1	3	1	3	0	1

Table 1: The values of the parameters of the priors for the local level model.

As can be seen in Table 1, the same prior is used for both σ and τ. The probability density function of this prior can be found in Figure 2. The value $x = 1$ is close to the peak but a little bit to the left.

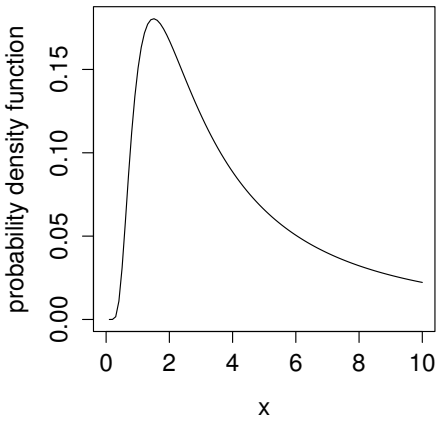

Figure 2: The inverse gamma distribution $IG(a, b)$ with $a = 1$ and $b = 3$

The second model is an extension of model 1. The local level model is combined with a seasonal component as described in section 2.3.2. The data is interpreted as being weekly data and a year is assumed to have exactly 52 weeks. This gives $s = 52$ with the notation in section 2.3.2. Only one harmonic is included with gives a DLM with 3 states. The matrices defining the model are

$$F_t = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} \qquad G = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\eta_1) & \sin(\eta_1) \\ 0 & -\sin(\eta_1) & \cos(\eta_1) \end{pmatrix},$$

where $\eta_1 = \frac{2\pi \cdot 1}{52}$.

An independent inverse gamma prior is used which means that W is a diagonal matrix. The same prior is used for all variances and the $IG(1, 3)$ distribution is used here too. The prior for the states is the $N(m_0, C_0)$ distribution but this time $m_0 = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$ and C_0 is the identity matrix.

5.2 Comparison with Gibbs sampler

The goal of this section is to make sure that the PL algorithm delivers correct results for the two test models. This is done by calculating the posterior of the variance parameters by PL and by a Gibbs sampler.

The data set contains 50 observations. The Gibbs sampler is set to produce 10000 samples and the Particle Learning uses the same number of particles. The histograms of the samples are then plotted side by side. The histograms for σ^2 can be found in Figure 3 and the histograms for τ^2 can be found in Figure 4. Both methods produce very similar results. This is not surprising since this model was tested in [3] in contrast to next model which is not tested in any of the articles about PL referenced in this report.

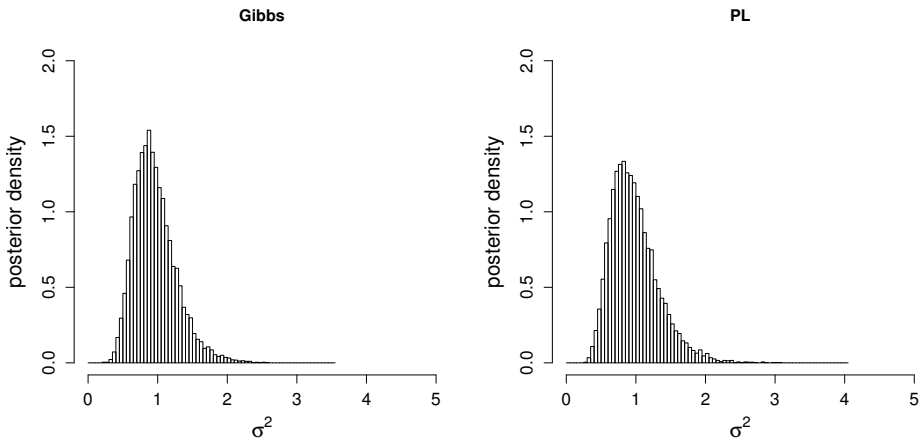

Figure 3: The left graph shows the computed posterior of σ^2 in form of a histogram computed by a Gibbs sampler. The right graph shows the same distribution computed with help of particle learning.

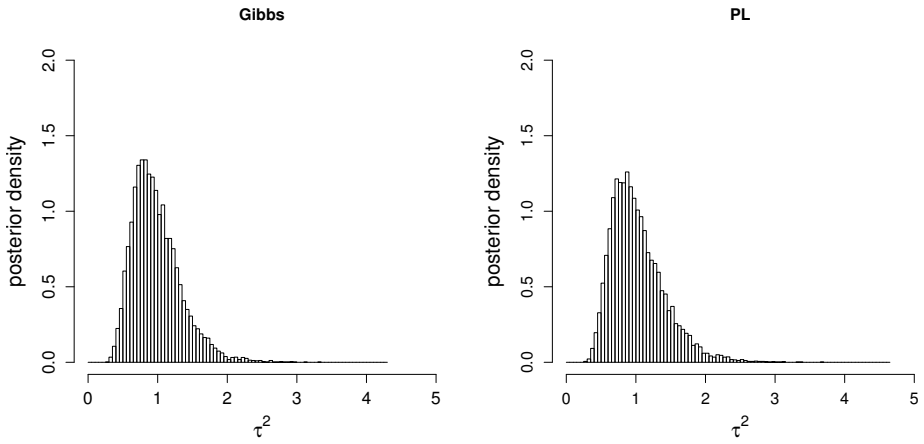

Figure 4: The left graph shows the computed posterior of τ^2 in form of a histogram computed by a Gibbs sampler. The right graph shows the same distribution computed with help of particle learning.

Model 2 is tested in the same way as model 1. The number of observation is the same as before as well as the number of iterations in the Gibbs sampler and the number of particles. Figure 5 shows the results regarding V. Figures 6 to 8 show the results for the diagonal elements of W. There are 3 such elements and consequently 3 figures with two histograms each. One histogram for the Gibbs sampler and one histogram for PL.

The results from the Gibbs sampler and PL are very similar for all states. The next step is to test what happens when the number of particles is reduced and more data points are analyzed.

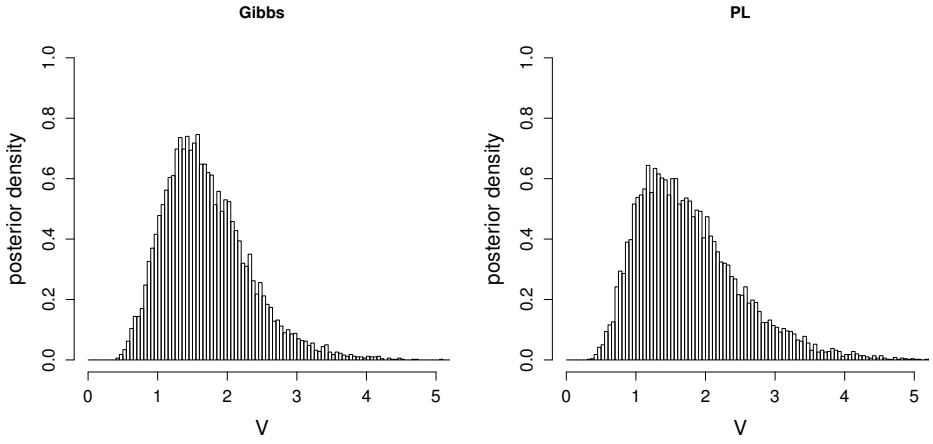

Figure 5: The left graph shows the computed posterior of V computed by a Gibbs sampler. The right graph shows the same distribution computed with help of particle learning.

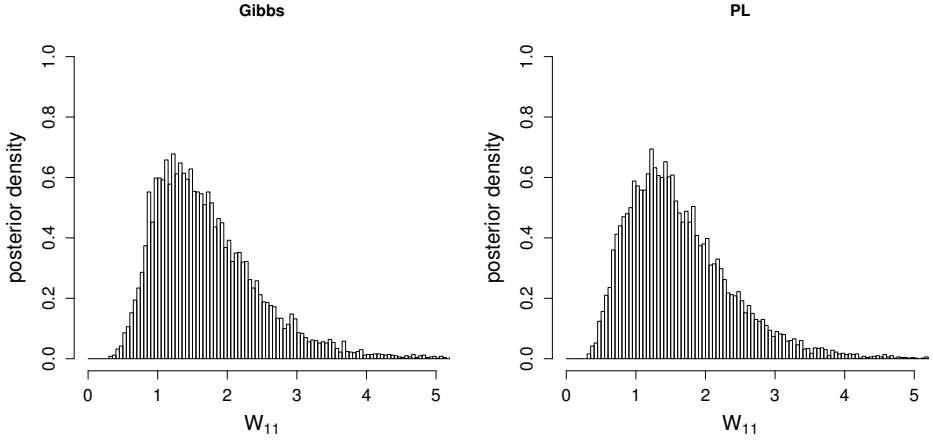

Figure 6: The left graph shows the computed posterior of W_{11} computed by a Gibbs sampler. The right graph shows the same distribution computed with help of particle learning.

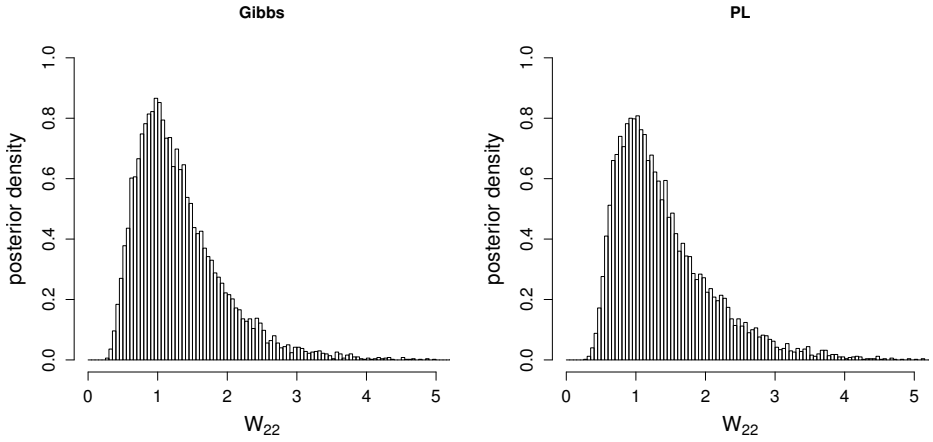

Figure 7: The left graph shows the computed posterior of W_{22} computed by a Gibbs sampler. The right graph shows the same distribution computed with help of particle learning.

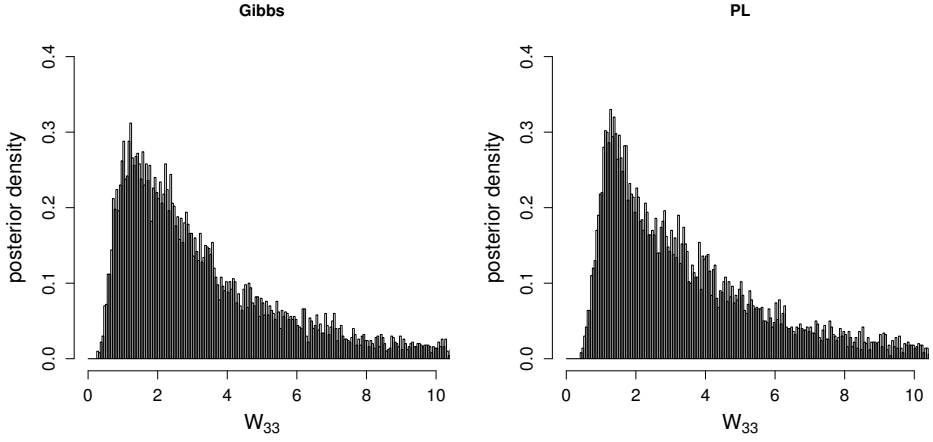

Figure 8: The left graph shows the computed posterior of W_{33} computed by a Gibbs sampler. The right graph shows the same distribution computed with help of particle learning.

5.3 Degeneracy

The degeneracy problem is studied by generating one dataset of the length 4000 for each model. Each dataset is then analyzed using only 500 particles. The posteriors are calculated using PL four times with the same data. The result is a large number of histograms which can be found in the appendix.

Model 1 is discussed first. The figures 9 and 10 show the posteriors of σ^2 and τ^2 respectively. The correct value is 1 in both cases. PL gives similar results for σ^2 in the first two calculations. The posteriors in calculation 3 and 4 however does not cover the true value and calculation 3 have peak lower then the true value whereas calculation 4 have a peak higher then the true value. The posteriors for τ^2 cover the true value in three of the four calculations but the peak is wandering around a little bit here as well.

Model 2 is more complex and hence performs worse. Both the width and the center of the peaks are different between different calculations. The maximal distance between peaks is about 2 as can be seen in Figure 12 which shows the posterior of the first diagonal element in W. This is a clear sign of the degeneration.

6 Discussions and Conclusions

In this report, PL is tested on two DLMs and the simplest of them, the local level model, has already been tested in [3]. In the first part of the experiment PL is compared against a more well established method called FFBS and the results agrees well with each other and with earlier results.

The main objective of this thesis is however to investigate the impact of particle path degeneracy on PL. This is done by analyzing time series that are 8 time as long as the number of particles and repeating the analysis four times which ideally should give the same result each time.

The local level model is discussed first. The calculated posteriors have very narrow peaks close to the true values but they seam to move around a bit. They may also have a to small width since the true value is not covered by the calculated posterior in all cases. This behavior is what one would expect if the method is starting to degenerate.

The second model performs much worse which is not surprising since it is more complex. PL does not seams to work at all for this model with the number of particles used in the experiment. Both the shape and and locations of the peaks vary between different calculations. This a clear sign that the method is degenerating.

It should also be noted that not the entire error can be attributed to the degeneracy since PL is a Monte Carlo method. Their will always be some variance with only 500 samples even if the samples are from the correct target distribution. In this case however it seams like at least some of the variance is caused by the degeneration. It would be better to study even longer time series with more particles but this is not practically possible with the implementation of PL used in this report. The results in [3] suggest that PL is considerably faster then a corresponding Gibbs sampler which could not be verified in this report. This can however be explained by the fact that the implementation of PL was not very optimized compered to the used Gibbs sampler.

The overall conclusion of this master's thesis is that particle path degeneracy can have a substantial influence on the results of PL. The problem can be circumvented by increasing the number of particles but this means that PL no longer can be seen as an online method which is its primary advantage over FFBS.

7 Appendix

This appendix contains all graphs used to study the degeneracy of Particle Learning. Figure 9 and Figure 10 belongs to model 1 and the rest to model 2. Each figure has 4 subfigures with the title calculation 1 to 4. Each of these subfigures comes from different runs of the PL algorithm.

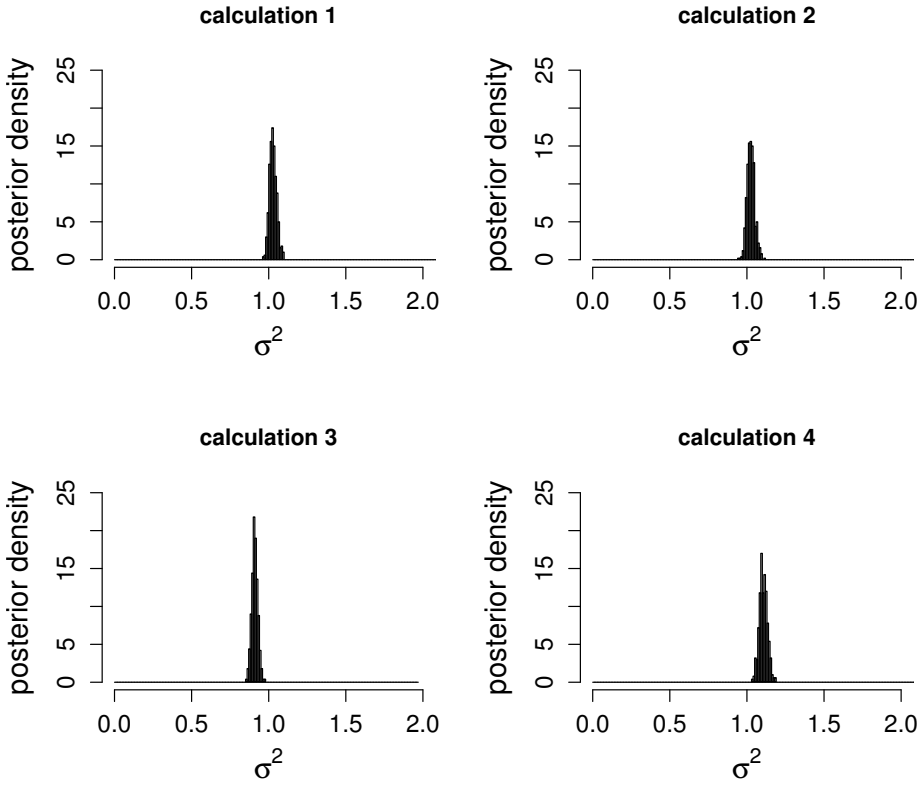

Figure 9: The posterior of σ^2 in model 1 at time 4000. The posterior is calculated 4 different times using PL with 500 particles.

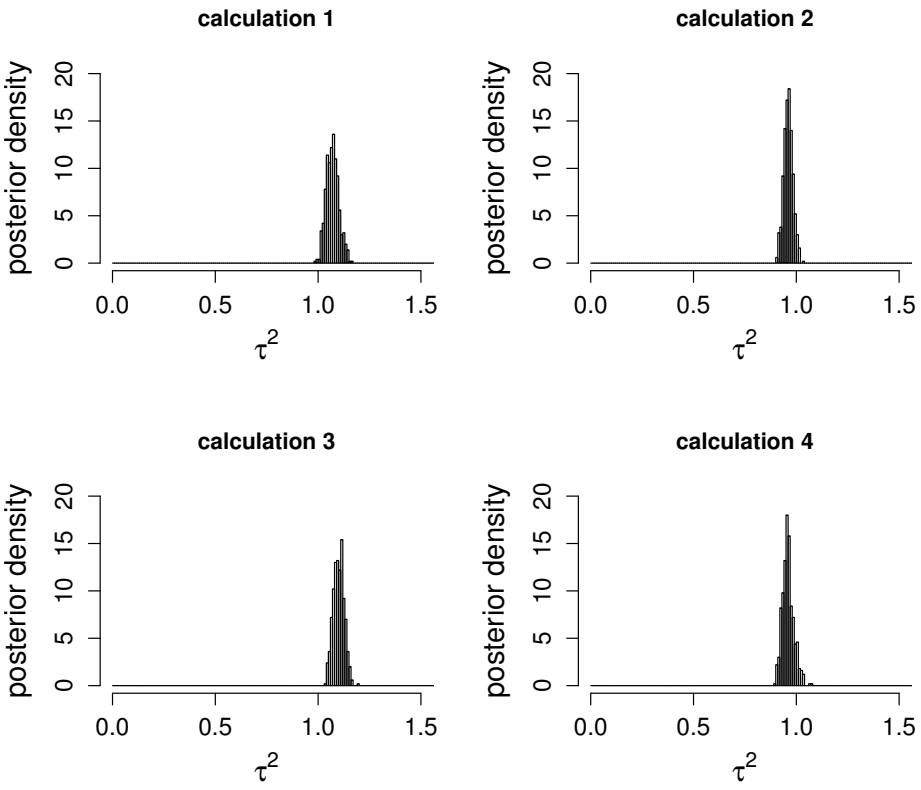

Figure 10: The posterior of τ^2 in model 1 at time 4000. The posterior is calculated 4 different times using PL with 500 particles.

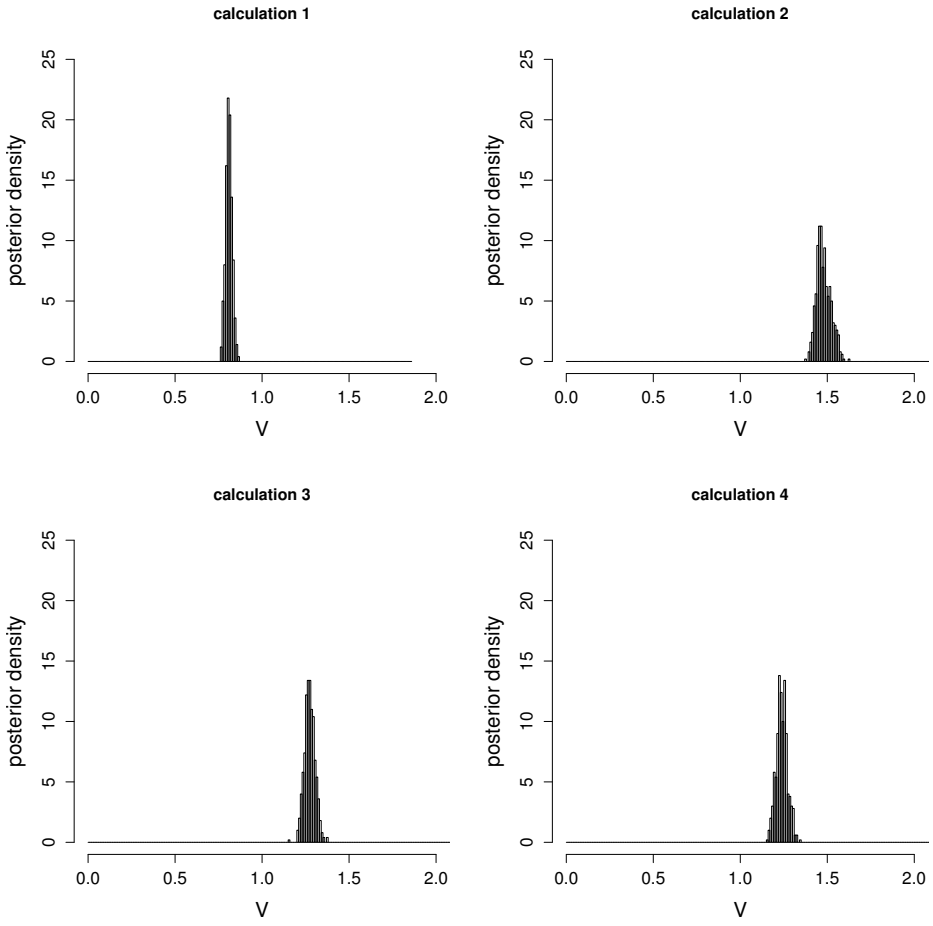

Figure 11: The posterior of V in model 2 at time 4000. The posterior is calculated 4 different times using PL with 500 particles.

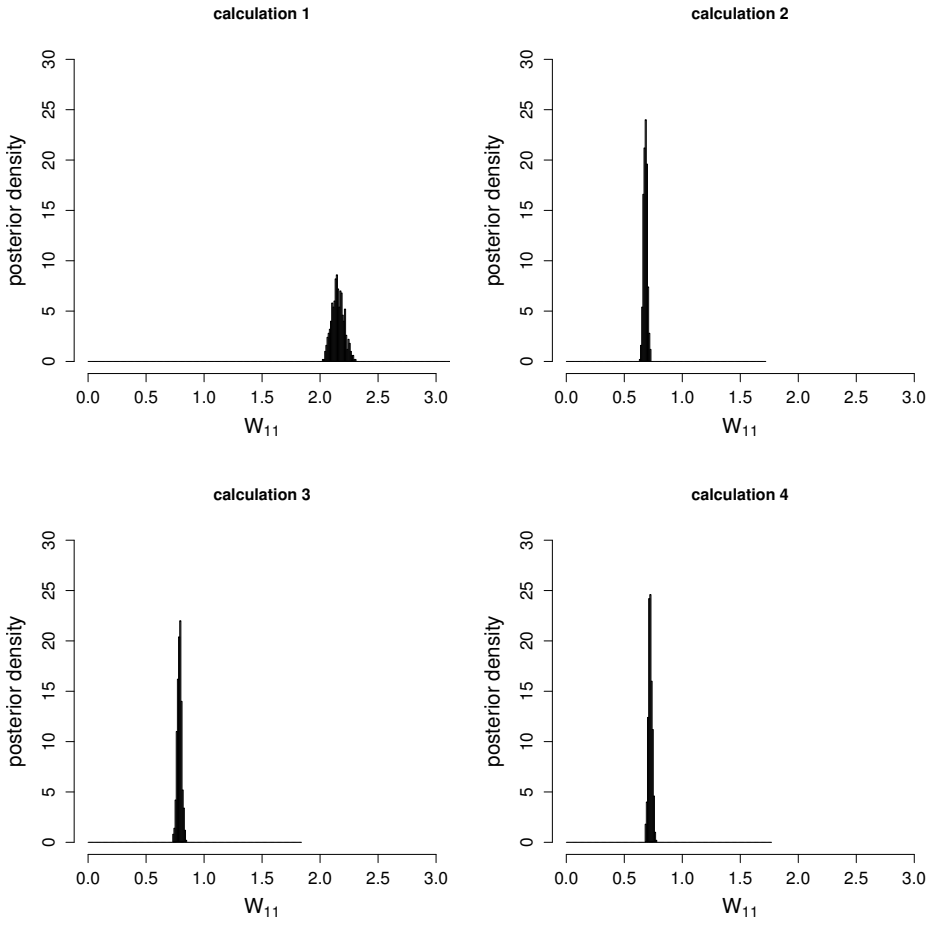

Figure 12: The posterior of W_{11} in model 2 at time 4000. The posterior is calculated 4 different times using PL with 500 particles.

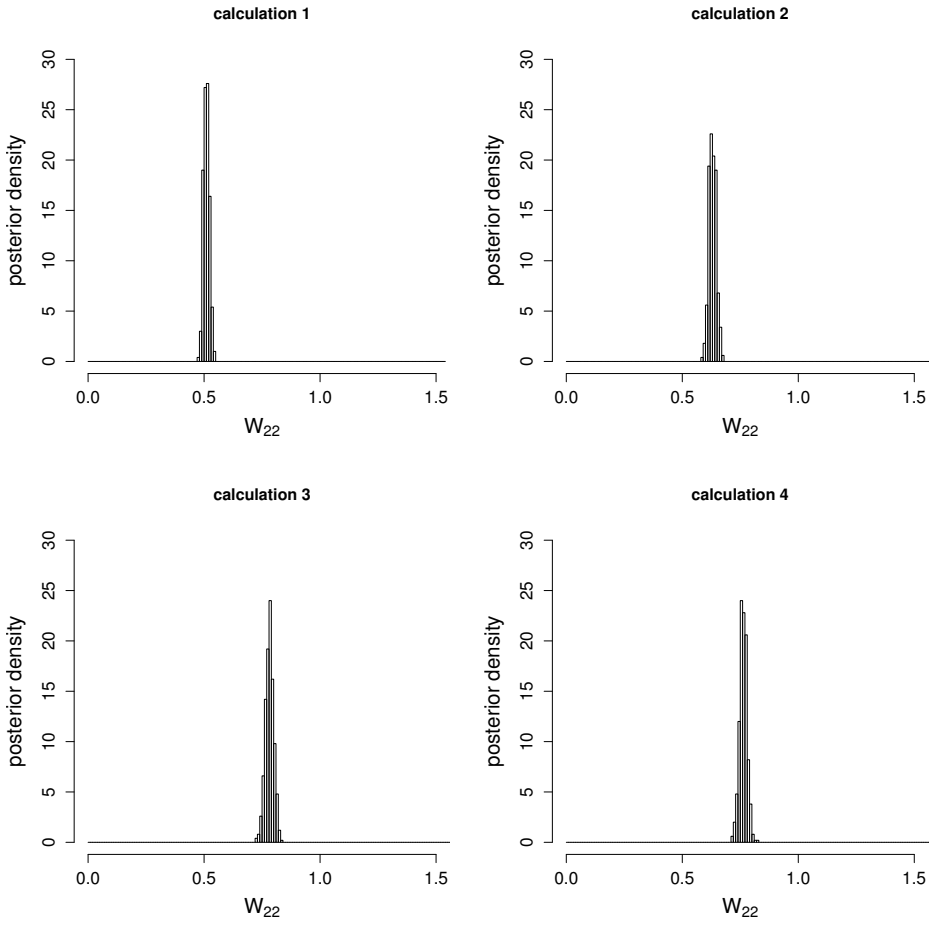

Figure 13: The posterior of W_{22} in model 2 at time 4000. The posterior is calculated 4 different times using PL with 500 particles.

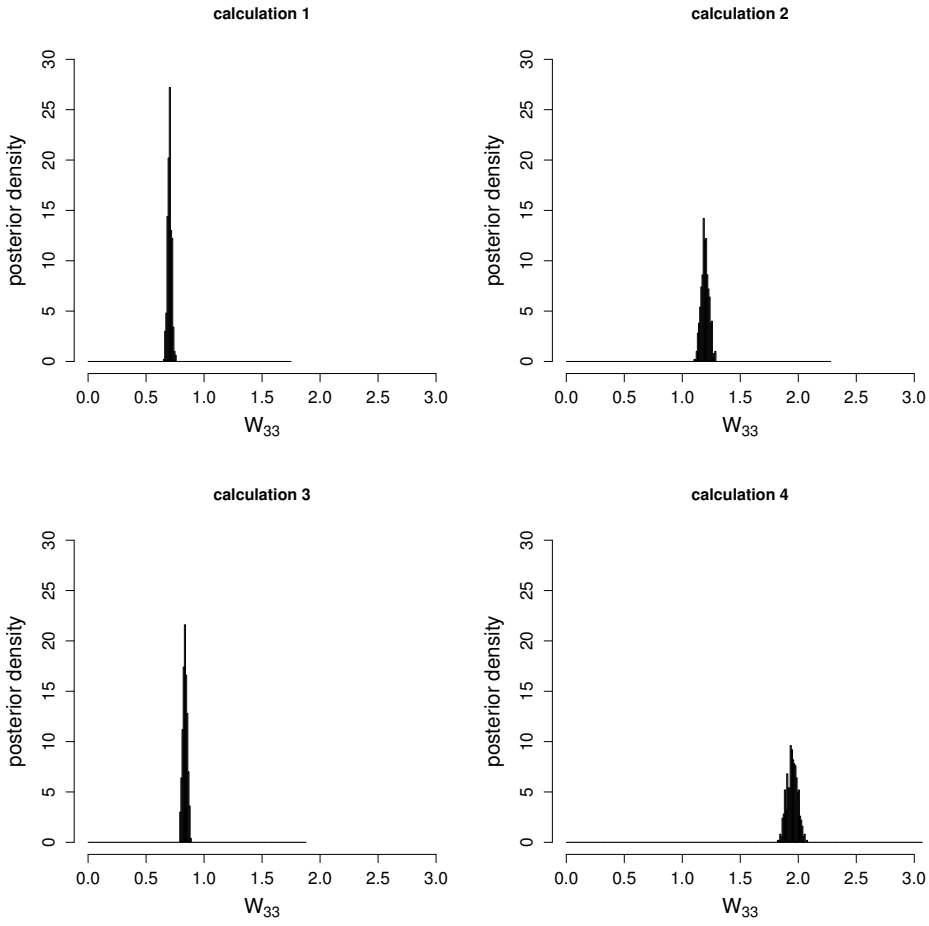

Figure 14: The posterior of W_{33} in model 2 at time 4000. The posterior is calculated 4 different times using PL with 500 particles.

8 References

[1] R. Kohn C. K. Carter. On gibbs sampling for state space models. *Biometrika*, 81(3):541–553, 1994.

[2] Olivier Cappé, Eric Moulines, and Tobias Ryden. *Inference in Hidden Markov Models (Springer Series in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[3] Carlos M. Carvalho, Michael S. Johannes, Hedibert F. Lopes, and Nicholas G. Polson. Particle learning and smoothing. *Statistical Science*, 25(1):pp. 88–106, 2010.

[4] Carlos M. Carvalho, Michael S. Johannes, Hedibert F. Lopes, and Nicholas G. Polson. Particle learning for sequential bayesian computation. *Bayesian Statistics*, 9:pp. 317–360, 2011.

[5] Randal Douc, Aurélien Garivier, Eric Moulines, and Jimmy Olsson. Sequential monte carlo smoothing for general state space hidden markov models. *Ann. Appl. Probab.*, 21(6):2109–2145, 12 2011.

[6] J. Durbin and S.J. Koopman. *Time Series Analysis by State Space Methods: Second Edition*. Oxford Statistical Science Series. OUP Oxford, 2012.

[7] Sylvia Frühwirth-Schnatter. Data augmentation and dynamic linear models. *Journal of Time Series Analysis*, 15(2):183–202, 1994.

[8] A. Gelman, J.B. Carlin, H.S. Stern, and D.B. Rubin. *Bayesian Data Analysis, Second Edition*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis, 2003.

[9] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings F - Radar and Signal Processing*, 140(2):107–113, April 1993.

[10] Neil Shephard Michael K. Pitt. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 94(446):590–599, 1999.

[11] Jimmy Olsson, Olivier Cappé, Randal Douc, and Éric Moulines. Sequential monte carlo smoothing with application to parameter estimation in nonlinear state space models. *Bernoulli*, 14(1):155–179, 02 2008.

[12] G. Petris, S. Petrone, and P. Campagnoli. *Dynamic Linear Models with R*. Use R! Springer New York, 2009.

[13] Neil Shephard. Partial non-gaussian state space. *Biometrika*, 81(1):115–131, 1994.

[14] Mike West Simon J. Godsill, Arnaud Doucet. Monte carlo smoothing for nonlinear time series. *Journal of the American Statistical Association*, 99(465):156–168, 2004.

[15] Youmin Zhang and X. R. Li. Fixed-interval smoothing algorithm based on singular value decomposition. In *Control Applications, 1996., Proceedings of the 1996 IEEE International Conference on*, pages 916–921, Sep 1996.

TRITA -MAT-E 2016:47
ISRN -KTH/MAT/E--16/47-SE