
Backward Simulation Methods for Monte Carlo Statistical Inference

Backward Simulation Methods for Monte Carlo Statistical Inference

Fredrik Lindsten

*Linköping University
Sweden*

lindsten@isy.liu.se

Thomas B. Schön

*Linköping University
Sweden*

schon@isy.liu.se

now

the essence of **know**ledge

Boston – Delft



Documents
Document Sharing Hub

Foundations and Trends[®] in Machine Learning

Published, sold and distributed by:

now Publishers Inc.
PO Box 1024
Hanover, MA 02339
USA
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:

now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is F. Lindsten and T. B. Schön, Backward Simulation Methods for Monte Carlo Statistical Inference, Foundations and Trends[®] in Machine Learning, vol 6, no 1, pp 1–143, 2013.

ISBN: 978-1-60198-698-6

© 2013 F. Lindsten and T. B. Schön

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, mechanical, photocopying, recording or otherwise, without prior written permission of the publishers.

Photocopying. In the USA: This journal is registered at the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. Authorization to photocopy items for internal or personal use, or the internal or personal use of specific clients, is granted by now Publishers Inc. for users registered with the Copyright Clearance Center (CCC). The 'services' for users can be found on the internet at: www.copyright.com

For those organizations that have been granted a photocopy license, a separate system of payment has been arranged. Authorization does not extend to other kinds of copying, such as that for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale. In the rest of the world: Permission to photocopy must be obtained from the copyright owner. Please apply to now Publishers Inc., PO Box 1024, Hanover, MA 02339, USA; Tel. +1-781-871-0245; www.nowpublishers.com; sales@nowpublishers.com

now Publishers Inc. has an exclusive license to publish this material worldwide. Permission to use this content must be obtained from the copyright license holder. Please apply to now Publishers, PO Box 179, 2600 AD Delft, The Netherlands, www.nowpublishers.com; e-mail: sales@nowpublishers.com

**Foundations and Trends[®] in
Machine Learning
Volume 6 Issue 1, 2013
Editorial Board**

Editor-in-Chief:

Michael Jordan

Department of Electrical Engineering and Computer Science

Department of Statistics

University of California, Berkeley

Berkeley, CA 94720-1776

Editors

Peter Bartlett (UC Berkeley)

Yoshua Bengio (Université de Montréal)

Avrim Blum (Carnegie Mellon University)

Craig Boutilier (University of Toronto)

Stephen Boyd (Stanford University)

Carla Brodley (Tufts University)

Inderjit Dhillon (University of Texas at Austin)

Jerome Friedman (Stanford University)

Kenji Fukumizu (Institute of Statistical Mathematics)

Zoubin Ghahramani (Cambridge University)

David Heckerman (Microsoft Research)

Tom Heskes (Radboud University Nijmegen)

Geoffrey Hinton (University of Toronto)

Aapo Hyvarinen (Helsinki Institute for Information Technology)

Leslie Pack Kaelbling (MIT)

Michael Kearns (University of Pennsylvania)

Daphne Koller (Stanford University)

John Lafferty (Carnegie Mellon University)

Michael Littman (Brown University)

Gabor Lugosi (Pompeu Fabra University)

David Madigan (Columbia University)

Pascal Massart (Université de Paris-Sud)

Andrew McCallum (University of Massachusetts Amherst)

Marina Meila (University of Washington)

Andrew Moore (Carnegie Mellon University)

John Platt (Microsoft Research)

Luc de Raedt (Albert-Ludwigs Universitaet Freiburg)

Christian Robert (Université Paris-Dauphine)

Sunita Sarawagi (IIT Bombay)

Robert Schapire (Princeton University)

Bernhard Schoelkopf (Max Planck Institute)

Richard Sutton (University of Alberta)

Larry Wasserman (Carnegie Mellon University)

Bin Yu (UC Berkeley)

Editorial Scope

Foundations and Trends[®] in Machine Learning publishes survey and tutorial articles in the following topics:

- Adaptive control and signal processing
- Applications and case studies
- Behavioral, cognitive and neural learning
- Bayesian learning
- Classification and prediction
- Clustering
- Data mining
- Dimensionality reduction
- Evaluation
- Game theoretic learning
- Graphical models
- Independent component analysis
- Inductive logic programming
- Kernel methods
- Markov chain Monte Carlo
- Model choice
- Nonparametric methods
- Online learning
- Optimization
- Reinforcement learning
- Relational learning
- Robustness
- Spectral methods
- Statistical learning theory
- Variational inference
- Visualization

Information for Librarians

Foundations and Trends[®] in Machine Learning, 2013, Volume 6, 4 issues.
ISSN paper version 1935-8237. ISSN online version 1935-8245. Also available as a combined paper and online subscription.

Backward Simulation Methods for Monte Carlo Statistical Inference

Fredrik Lindsten¹ and Thomas B. Schön²

¹ *Division of Automatic Control, Linköping University, Linköping, 581 83, Sweden, lindsten@isy.liu.se*

² *Division of Automatic Control, Linköping University, Linköping, 581 83, Sweden, schon@isy.liu.se*

Abstract

Monte Carlo methods, in particular those based on Markov chains and on interacting particle systems, are by now tools that are routinely used in machine learning. These methods have had a profound impact on statistical inference in a wide range of application areas where probabilistic models are used. Moreover, there are many algorithms in machine learning which are based on the idea of processing the data sequentially, first in the forward direction and then in the backward direction. In this tutorial, we will review a branch of Monte Carlo methods based on the forward–backward idea, referred to as backward simulators. These methods are useful for learning and inference in probabilistic models containing latent stochastic processes. The theory and practice of backward simulation algorithms have undergone a significant development in recent years and the algorithms keep finding new applications. The foundation for these methods is sequential Monte Carlo (SMC). SMC-based backward simulators are capable of addressing smoothing problems in sequential latent variable models, such as

general, nonlinear/non-Gaussian state-space models (SSMs). However, we will also clearly show that the underlying backward simulation idea is by no means restricted to SSMs. Furthermore, backward simulation plays an important role in recent developments of Markov chain Monte Carlo (MCMC) methods. Particle MCMC is a systematic way of using SMC within MCMC. In this framework, backward simulation gives us a way to significantly improve the performance of the samplers. We review and discuss several related backward-simulation-based methods for state inference as well as learning of static parameters, both using a frequentistic and a Bayesian approach.

Contents

1	Introduction	1
1.1	Background and Motivation	1
1.2	Notation and Definitions	3
1.3	A Preview Example	4
1.4	State-Space Models	7
1.5	Parameter Learning in SSMs	9
1.6	Smoothing Recursions	11
1.7	Backward Simulation in Linear Gaussian SSMs	13
1.8	Outline	16
2	Monte Carlo Preliminaries	19
2.1	Sequential Monte Carlo	19
2.2	Markov Chain Monte Carlo	26
3	Backward Simulation for State-Space Models	35
3.1	Forward Filter/Backward Simulator	36
3.2	Analysis and Convergence	43
3.3	Backward Simulation with Rejection Sampling	49
3.4	Backward Simulation with MCMC Moves	56
3.5	Backward Simulation for Maximum Likelihood Inference	62

4 Backward Simulation for General Sequential Models	65
4.1 Motivating Examples	65
4.2 SMC Revisited	69
4.3 A General Backward Simulator	72
4.4 Rao–Blackwellized FFBSi	78
4.5 Non-Markovian Latent Variable Models	82
4.6 From State-Space Models to Non-Markovian Models	84
5 Backward Simulation in Particle MCMC	91
5.1 Introduction to PMCMC	91
5.2 Particle Marginal Metropolis–Hastings	93
5.3 PMMH with Backward Simulation	102
5.4 Particle Gibbs with Backward Simulation	106
5.5 Particle Gibbs with Ancestor Sampling	117
5.6 PMCMC for Maximum Likelihood Inference	122
5.7 PMCMC for State Smoothing	126
6 Discussion	129
Acknowledgments	133
Notations and Acronyms	135
References	137

1

Introduction

A basic strategy to address many inferential problems in machine learning is to process data sequentially, first in the forward direction and then in the backward direction. Examples of this approach are the well-known forward-backward algorithm for hidden Markov models (HMMs) and the Rauch-Tung-Striebel smoother [119] for linear Gaussian state-space models. Moreover, two decades of research on sequential Monte Carlo and Markov chain Monte Carlo have enabled inference in increasingly more challenging models. Many developments have been made in order to make use of the forward-backward idea together with these Monte Carlo methods, providing inferential techniques collectively referred to as backward simulation. This tutorial provides a unifying view of these methods. In this introductory section we review some relevant background materials and also derive a first backward simulator for the special case of linear Gaussian state-space models.

1.1 Background and Motivation

For over half a century, Monte Carlo methods have been recognized as potent tools for statistical inference in complex probabilistic

2 Introduction

models; see [103] for an early discussion. A continuous development and refinement of these methods have enabled inference in increasingly more challenging models. A key milestone in this development was the introduction of Markov chain Monte Carlo (MCMC) methods through the inventions of the Metropolis–Hastings algorithm [71, 102] and the Gibbs sampler [58]. Parallel to this, sequential importance sampling [70] and sampling/importance resampling [122] laid the foundation of sequential Monte Carlo (SMC). In its modern form, SMC was first introduced in [64, 129]. During the 1990s, several independent developments were made by, among others, [77, 83]. Recently, SMC and MCMC have been combined in a systematic manner through the developments of pseudo-marginal methods [6, 11] and particle MCMC [3].

Backward simulation is a strategy which is useful as a Monte Carlo method for learning of probabilistic models containing latent stochastic processes. In particular, we will consider inference in dynamical systems, i.e., systems that evolve over time. Dynamical systems play a central role in a wide range of scientific fields, such as signal processing, automatic control, epidemiology and econometrics, to mention a few.

One of the most widely used models of a dynamical system is the state-space model (SSM), reviewed in more detail in Sections 1.4–1.6. The structure of an SSM can be seen as influenced by the notion of a physical system. At each time t , the system is assumed to be in a certain state x_t . The state contains all relevant information about the system, i.e., if we would know the state of the system we would have full insight into its internal condition. However, the state is typically not known. Instead, we measure some quantity y_t which depend on the state in some way. Given a sequence of observations $y_{1:T} \triangleq (y_1, \dots, y_T)$, we seek to draw inference about the latent state process $x_{1:T}$ (state inference), as well as about unknown static parameters of the model (parameter inference).

The class of SSMs will play a central role in this tutorial. Indeed, many of the inferential methods that we will review have been developed explicitly for SSMs. However, as will become apparent in Sections 4 and 5, most of the methods are more general and can be used for learning interesting models outside the class of SSMs.

Backward simulation is based on the forward–backward idea. That is, the data is processed first in the forward direction and then in the backward direction. In the backward pass, the state process is simulated backward in time, i.e., by first simulating x_T , then x_{T-1} etc., until a complete state trajectory $x_{1:T}$ is generated. This procedure gives us a tool to address the state smoothing problem in models for which no closed form solution is available. This is done by simulating multiple backward trajectories from the smoothing distribution, i.e., conditionally on the observations $y_{1:T}$, which can then be used for Monte Carlo integration. State smoothing is of key relevance, e.g., to obtain refined state estimates in offline settings. Furthermore, it lies at the core of many parameter inference methods (see Section 1.5) and it can be used to address problems in optimal control (see Section 4.1).

Backward simulation is also useful in MCMC, as a way of grouping variables to improve the mixing of the sampler. A common way to construct an MCMC sampler for an SSM is to sample the state variables x_t , for different t , one at a time (referred to as single-state sampling). However, since the states are often strongly dependent across time, this can lead to poor performance. Backward simulation provides a mean of grouping the state variables and sampling the entire trajectory $x_{1:T}$ as one entity. As we will illustrate in Section 1.3, this can lead to a considerable improvement upon the single-state sampler.

In Section 1.7 we will derive a first backward simulator for the class of linear Gaussian state-space (LGSS) models. Apart from LGSS models, exact backward simulation is tractable, basically only for finite state-space HMMs (see also Section 4.1.1). The main focus in this tutorial will be on models outside these restricted classes, for which exact backward simulation is not possible. Instead, we will make use of SMC (and MCMC) to enable backward simulation in challenging probabilistic models, such as nonlinear/non-Gaussian SSMs, as well as more general non-Markovian latent variable models.

1.2 Notation and Definitions

For any sequence $\{x_k\}_{k \in \mathbb{N}}$ and integers $m \leq n$ we write $x_{m:n} \triangleq (x_m, \dots, x_n)$. We let $\wedge$ be the minimum operator, i.e., $a \wedge b \triangleq \min(a, b)$.

For a matrix A , the matrix transpose is written as $A^\top$. For two probability distributions μ_1 and μ_2 , the total variation distance is given by $\|\mu_1 - \mu_2\|_{\text{TV}} \triangleq \sup_A |\mu_1(A) - \mu_2(A)|$. A Dirac point-mass located at some point x' is denoted as $\delta_{x'}(dx)$. We write $X \sim \mu$ to mean that the random variable X is either distributed according to μ , or sampled from μ . The uniform probability distribution on the interval $[a, b]$ is written as $\mathcal{U}([a, b])$. $\text{Cat}(\{p_i\}_{i=1}^n)$, with $\sum_{i=1}^n p_i = 1$, is the categorical (i.e., discrete) probability distribution on the set $\{1, \dots, n\}$, with probabilities $\{p_i\}_{i=1}^n$. Finally, $\mathcal{N}(m, \Sigma)$ and $\mathcal{N}(x; m, \Sigma)$ are the Gaussian (i.e., normal) probability distribution and density function, respectively, with mean vector m , covariance matrix Σ and argument x .

1.3 A Preview Example

Before we continue with this section on background theory, we consider an example to illustrate the potential benefit of using backward simulation. A simple stochastic volatility SSM is given by,

$$x_{t+1} = ax_t + v_t, \quad v_t \sim \mathcal{N}(0, q), \quad (1.1a)$$

$$y_t = e_t \exp\left(\frac{1}{2}x_t\right), \quad e_t \sim \mathcal{N}(0, 1), \quad (1.1b)$$

where the state process $\{x_t\}_{t \geq 1}$ is latent and observations are made only via the measurement process $\{y_t\}_{t \geq 1}$. Similar models have been used to generalize the Black–Scholes option pricing equation to allow for the variance to change over time [27, 101]. The same model was used by [30] to illustrate the poor mixing of a single-state Gibbs sampler; an example which is replicated here.

For simplicity, we assume that the parameters $a = 0.99$ and $q = 0.01$ are known. We seek the density $p(x_{1:T} | y_{1:T})$, i.e., the conditional density of the state process $x_{1:T}$ given a sequence of observations $y_{1:T}$ for some fixed final time point T . This conditional density is referred to as the joint smoothing density (JSD). For the model under study, the JSD is not available in closed form due to the nonlinear measurement Equation (1.1b). To remedy this, we construct an MCMC method to approximately sample from it. MCMC will be reviewed in more detail in Section 2.2. However, the basic idea is to simulate a Markov chain which is constructed in such a way that it admits the target

distribution as limiting distribution. The sample path from the Markov chain can then be used to draw inference about the target density $p(x_{1:T} \mid y_{1:T})$.

As an initial attempt, we try a single-state Gibbs sampler. That is, we sample each state x_t conditionally on $\{x_{1:t-1}, x_{t+1:T}\}$ (and the observations $y_{1:T}$). At each iteration of the Gibbs sampler we thus simulate according to,

$$\begin{aligned} x'_1 &\sim p(x_1 \mid x_{2:T}, y_{1:T}); \\ &\vdots \\ x'_t &\sim p(x_t \mid x'_{1:t-1}, x_{t+1:T}, y_{1:T}); \\ &\vdots \\ x'_T &\sim p(x_T \mid x'_{1:T-1}, y_{1:T}). \end{aligned}$$

This procedure will leave $p(x_{1:T} \mid y_{1:T})$ invariant (see Section 2.2 for more on Gibbs sampling) and it results in a valid MCMC sampler. The conditional densities $p(x_t \mid x_{1:t-1}, x_{t+1:T}, y_{1:T})$ are not available in closed form. However, for this model (Equation (1.1)), they are log-concave and we can employ the efficient rejection sampling strategy by [145] to sample exactly from these distributions.

The single-state Gibbs sampler will indeed converge to samples from $p(x_{1:T} \mid y_{1:T})$. However, it is well recognized that single-state samplers can suffer from poor mixing, due to the often strong dependencies between consecutive state variables. That is, the convergence can be slow in the sense that we need to iterate the above sampling scheme a large number of times to get reliable samples.

To analyze this, we generate $T = 100$ samples from the model (Equation (1.1)) and run the Gibbs sampler for 100 000 iterations (in each iteration, we loop over all the state variables for $t = 1, \dots, T$). The first 10 000 iterations are discarded, to avoid transient effects. We then compute the empirical autocorrelation function (ACF) of the state x_{50} , which is given in Figure 1.1. As can be seen, the ACF decreases very slowly, indicating a poorly mixing Gibbs kernel. This simply reflects the fact that, when the state variables are highly correlated, the single-state sampler will be inefficient at exploring the state-space. This is a

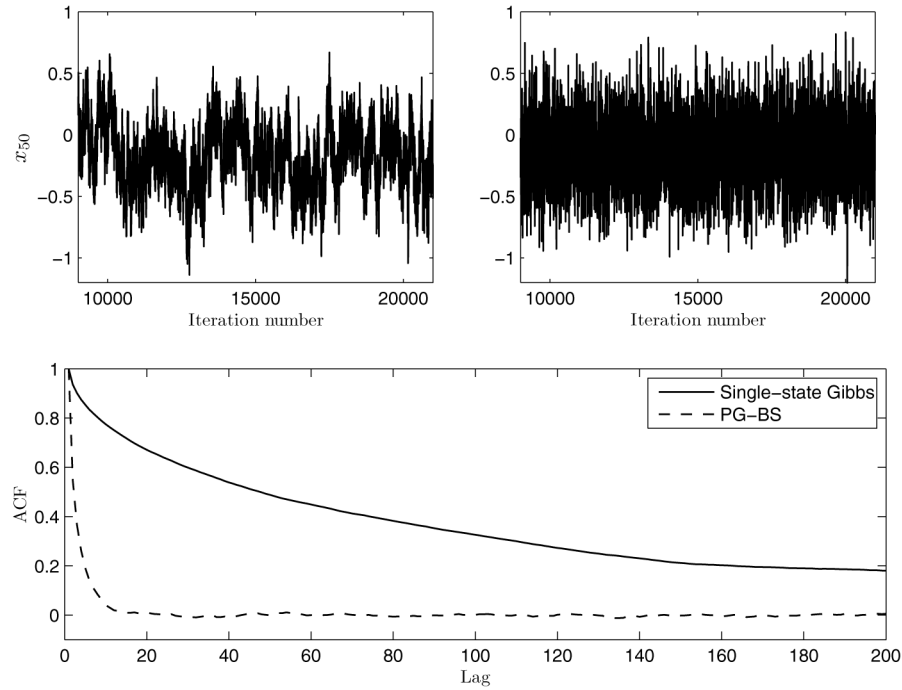


Fig. 1.1 (Top left) Part of sample path for the single-state Gibbs sampler; (Top right) Part of sample path for PGBS; (Bottom) Empirical ACF for x_{50} for the single-state Gibbs sampler and for PGBS using $N = 15$ particles.

common and well-recognized problem when addressing the state inference problem for SSMs.

One way to remedy this is to group the variables and sample a full state trajectory $x_{1:T}$ jointly. This is what a backward simulator aims to accomplish. Grouping variables in a Gibbs sampler will in general improve upon the mixing of the single-state sampler [97, Section 6.7], and in practice the improvement can be quite considerable.

To illustrate this, we have included the ACF for a backward-simulation-based method in Figure 1.1. Since the model (Equation (1.1)) is nonlinear, exact backward simulation is not possible. Instead, the results reported here are from a backward simulator based on SMC, using (only) $N = 15$ particles. The specific method that we have used is denoted as particle Gibbs with backward simulation (PGBS), and it will be discussed in detail in Section 5.4. For the PGBS,

the ACF drops off much more rapidly, indicating a more efficient sampler. Furthermore, a key property of PGBS is that, despite the fact that it relies on a crude SMC approximation, it does not alter the stationary distribution of the Gibbs sampler, nor does it introduce any additional bias. That is, PGBS will, just as the single-state Gibbs sampler, target the exact JSD $p(x_{1:T} | y_{1:T})$. This property is known as *exact approximation*, a concept that we will return to in Section 5.

1.4 State-Space Models

State-space models (SSMs) are commonly used to model time series and dynamical systems. Additionally, many models that are not sequential “by nature” can also be written on state-space form. It is a comprehensive and important class of models, and it serves as a good starting point for introducing the concepts that will be discussed throughout this tutorial.

We consider here discrete-time SSMs on a general state-space $\mathbf{X}$. The system state is a Markov process $\{x_t\}_{t \geq 1}$ on $\mathbf{X}$, evolving according to a Markov transition kernel $F(dx_{t+1} | x_t)$ and with initial distribution $\nu(dx_1)$. The state x_t is assumed to summarize all relevant information about the system at time t . However, the state process is latent and it is observed only implicitly through the observations $\{y_t\}_{t \geq 1}$, taking values in some set $\mathbf{Y}$. Given x_t , the measurement y_t is conditionally independent of past and future states and observations, and it is distributed according to a kernel $G(dy_t | x_t)$. A graphical model, illustrating the conditional dependencies in an SSM, is given in Figure 1.2.

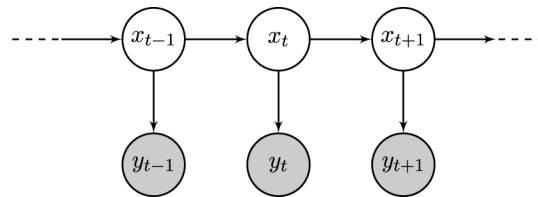


Fig. 1.2 Graphical model of an SSM. The white nodes represent latent variables and the gray nodes represent observed variables.

We shall assume that the observation kernel G admits a probability density g w.r.t. some dominating measure, which we simply denote dy . Such models are referred to as partially dominated. If, in addition, the transition kernel F admits a density f and the initial distribution ν admits a density μ , both w.r.t. some dominating measure dx , the model is called fully dominated. In summary, a fully dominated SSM can be expressed as,

$$x_{t+1} \sim f(x_{t+1} | x_t), \quad (1.2a)$$

$$y_t \sim g(y_t | x_t), \quad (1.2b)$$

and $x_1 \sim \mu(x_1)$. Two examples of SSMs follow below.

Example 1.1 (Finite state-space hidden Markov model). A finite state-space HMM, or simply HMM, is an SSM with $X = \{1, \dots, K\}$ for some finite K . The transition density (w.r.t. counting measure) can be summarized in a $K \times K$ transition matrix Π , where the (i, j) th entry is given by,

$$\Pi_{i,j} = P(x_{t+1} = j | x_t = i) = f(j | i).$$

Hence, $f(j | i)$ denotes the probability of moving from state i at time t , to state j at time $t + 1$.

Example 1.2 (Additive noise model). In engineering applications, SSMs are often expressed on functional form with additive noise,

$$x_{t+1} = a(x_t) + v_t,$$

$$y_t = c(x_t) + e_t,$$

for some functions a and c . Here, the noises v_t and e_t are commonly referred to as process noise and measurement noise, respectively. If the noise distributions admit densities w.r.t. dominating measures, then the model is fully dominated. The transition density is then given by $f(x_{t+1} | x_t) = p_{v_t}(x_{t+1} - a(x_t))$ and similarly for the observation density.

Throughout this tutorial, we will mostly be concerned with fully dominated SSMs and therefore do most of our derivations in terms of probability densities. There are, however, several examples of interesting models that are *degenerate*, i.e., that are not fully dominated. We will return to this in the sequel and discuss how it affects the methods presented in here.

1.5 Parameter Learning in SSMs

The basic inference problem for SSMs is typically that of state inference, i.e., to infer the latent states given measurements from the system. In fact, even when the actual task is to learn a model of the system dynamics, state inference tends to play a crucial role as an intermediate step of the learning algorithm. To illustrate this, assume that the SSM (Equation (1.2)) is parameterized by some unknown parameter $\theta \in \Theta$,

$$x_{t+1} \sim f_\theta(x_{t+1} \mid x_t), \quad (1.3a)$$

$$y_t \sim g_\theta(y_t \mid x_t), \quad (1.3b)$$

and $x_1 \sim \mu_\theta(x_1)$. Given a batch of measurements $y_{1:T}$, we wish to draw inference about θ . In the Bayesian setting, a prior distribution $\pi(\theta)$ is assigned to the parameter and the learning problem amounts to computing the posterior distribution $p(\theta \mid y_{1:T})$.

A complicating factor is that the likelihood $p(y_{1:T} \mid \theta)$ in general cannot be computed in closed form. To address this difficulty, it is common to make use of *data augmentation* [136, 132]. That is, we target the joint state and parameter posterior $p(\theta, x_{1:T} \mid y_{1:T})$, rather than the marginal posterior $p(\theta \mid y_{1:T})$. The latent states are thus viewed as auxiliary variables. This opens up for using Gibbs sampling (see Section 2.2), for instance by initializing $\theta[0] \in \Theta$ and iterating;

- (i) Draw $x_{1:T}[r] \sim p(x_{1:T} \mid \theta[r-1], y_{1:T})$;
- (ii) Draw $\theta[r] \sim p(\theta \mid x_{1:T}[r], y_{1:T})$.

Under weak assumptions, this procedure will generate a Markov chain $\{\theta[r], x_{1:T}[r]\}_{r \geq 1}$ with stationary distribution $p(\theta, x_{1:T} \mid y_{1:T})$. Consequently, the stationary distribution of the subchain $\{\theta[r]\}_{r \geq 1}$ will be the

marginal parameter posterior distribution $p(\theta \mid y_{1:T})$. Note that Step (i) of the above sampling scheme requires the computation of the JSD, for a fixed value of the parameter θ . That is, we need to address an intermediate smoothing problem in order to implement this Gibbs sampler.

Data augmentation is commonly used also in the frequentistic setting. Assume that we, instead of the posterior distribution, seek the maximum likelihood estimator (MLE),

$$\hat{\theta}_{\text{ML}} = \arg \max_{\theta \in \Theta} \log p_{\theta}(y_{1:T}), \quad (1.4)$$

where $p_{\theta}(y_{1:T})$ is the likelihood of the observed data for a given value of the system parameter θ . Again, since the log-likelihood $\log p_{\theta}(y_{1:T})$ is not available in closed form, direct maximization in Equation (1.4) is problematic. Instead, we can make use of the expectation maximization (EM) algorithm [33] (see also [100]). The EM algorithm is an iterative method, which maximizes $p_{\theta}(y_{1:T})$ by iteratively maximizing an auxiliary quantity,

$$Q(\theta, \theta') = \int \log p_{\theta}(x_{1:T}, y_{1:T}) p_{\theta'}(x_{1:T} \mid y_{1:T}) dx_{1:T}. \quad (1.5)$$

The EM algorithm is useful when maximization of $\theta \mapsto Q(\theta, \theta')$, for fixed θ' , is simpler than direct maximization of the log-likelihood, $\theta \mapsto \log p_{\theta}(y_{1:T})$. The procedure is initialized at some $\theta[0] \in \Theta$ and then iterates between two steps, expectation (E) and maximization (M);

(E) Compute $Q(\theta, \theta[r-1])$;

(M) Compute $\theta[r] = \arg \max_{\theta \in \Theta} Q(\theta, \theta[r-1])$.

The resulting sequence $\{\theta[r]\}_{r \geq 0}$ will, under weak assumptions, converge to a stationary point of the likelihood $p_{\theta}(y_{1:T})$ [148].

Using the conditional independence properties of an SSM, we can write the complete data log-likelihood as

$$\begin{aligned} & \log p_{\theta}(x_{1:T}, y_{1:T}) \\ &= \log \mu_{\theta}(x_1) + \sum_{t=1}^T \log g_{\theta}(y_t \mid x_t) + \sum_{t=1}^{T-1} \log f_{\theta}(x_{t+1} \mid x_t). \end{aligned} \quad (1.6)$$

From Equation (1.5), we note that the auxiliary quantity is defined as the expectation of expression (1.6) under the JSD. Hence, to carry out the E-step of the EM algorithm, we again need to address an intermediate smoothing problem for fixed values of the system parameters.

1.6 Smoothing Recursions

As noted above, the JSD is a quantity of central interest for learning and inference problems in SSMs. It summarizes all the information about the latent states which is available in the observations. Many densities that arise in various state inference problems are given as marginals of the JSD. There are a few that are of particular interest, which we summarize in Table 1.1. To avoid a cluttered notation, we now drop the (possible) dependence on an unknown parameter θ from the notation and write the JSD as $p(x_{1:T} | y_{1:T})$.

As in Equation (1.6), the conditional independence properties of an SSM implies that the complete data likelihood can be written as,

$$p(x_{1:T}, y_{1:T}) = \mu(x_1) \prod_{t=1}^T g(y_t | x_t) \prod_{t=1}^{T-1} f(x_{t+1} | x_t). \quad (1.7)$$

The JSD is related to the above expression by Bayes' rule,

$$p(x_{1:T} | y_{1:T}) = \frac{p(x_{1:T}, y_{1:T})}{\int p(x_{1:T}, y_{1:T}) dx_{1:T}}. \quad (1.8)$$

Despite the simplicity of this expression, it is of limited use in practice due to the high-dimensional integration needed to compute the normalization factor in the denominator. Instead, most practical methods

Table 1.1 Filtering and smoothing densities of particular interest.

	Density
Filtering ^a	$p(x_t y_{1:t})$
Joint smoothing	$p(x_{1:T} y_{1:T})$
Marginal smoothing ($t \leq T$)	$p(x_t y_{1:T})$
Fixed-interval smoothing ($s < t \leq T$)	$p(x_{s:t} y_{1:T})$
Fixed-lag smoothing (ℓ fixed) ^a	$p(x_{t-\ell+1:t} y_{1:t})$

^a The filtering and fixed-lag smoothing densities are marginals of the JSD at time t , $p(x_{1:t} | y_{1:t})$.

(and in particular the ones discussed in this tutorial) are based on a recursive evaluation of the JSD.

Again by using Bayes' rule, we get the following two-step procedure,

$$p(x_{1:t} | y_{1:t}) = \frac{g(y_t | x_t)p(x_{1:t} | y_{1:t-1})}{p(y_t | y_{1:t-1})}, \quad (1.9a)$$

$$p(x_{1:t+1} | y_{1:t}) = f(x_{t+1} | x_t)p(x_{1:t} | y_{1:t}). \quad (1.9b)$$

The above equations will be denoted as the forward recursion for the JSD, since they evolve forward in time. Step (1.9a) is often referred to as the measurement update, since the current measurement y_t is taken into account. Step (1.9b) is known as the time update, moving the density forward in time, from t to $t + 1$.

An interesting fact about SSMs is that, conditioned on $y_{1:T}$, the state process $\{x_t\}_{t=1}^T$ is an inhomogeneous Markov process. Under weak assumptions (see [23, Section 3.3.2] for details), the same holds true for the time-reversed chain, starting at time T and evolving backward in time according to the so-called backward kernel,

$$B_t(A | x_{t+1}) \triangleq P(x_t \in A | x_{t+1}, y_{1:T}). \quad (1.10)$$

Note that the backward kernel is time inhomogeneous. In the general case, it is not possible to give an explicit expression for the backward kernel. However, for a fully dominated model, this can always be done, and its density is given by

$$p(x_t | x_{t+1}, y_{1:T}) = \frac{f(x_{t+1} | x_t)p(x_t | y_{1:t})}{\int f(x_{t+1} | x_t)p(x_t | y_{1:t})dx_t}. \quad (1.11)$$

From the conditional independence properties of an SSM, it also holds that $p(x_t | x_{t+1}, y_{1:T}) = p(x_t | x_{t+1}, y_{1:t})$.

Using the backward kernel, we get an alternative recursion for the JSD, evolving backward in time,

$$p(x_{t:T} | y_{1:T}) = p(x_t | x_{t+1}, y_{1:t})p(x_{t+1:T} | y_{1:T}), \quad (1.12)$$

starting with the filtering density at time T , $p(x_T | y_{1:T})$. This is known as the backward recursion. At time $t = 1$, the JSD for the time interval $1, \dots, T$ is obtained.

The backward kernel density at time t depends only on the transition density $f(x_{t+1} | x_t)$ and on the filtering density $p(x_t | y_{1:t})$, a property which is of key relevance. Hence, to utilise the backward recursion (Equation (1.12)) for computing the JSD, the filtering densities must first be computed for $t = 1, \dots, T$. Consequently, this procedure is generally called forward filtering/backward smoothing.

1.7 Backward Simulation in Linear Gaussian SSMs

An important special case of Equation (1.2) is the class of linear Gaussian state-space models. A functional representation of an LGSS model is given by,

$$x_{t+1} = Ax_t + v_t, \quad v_t \sim \mathcal{N}(0, Q), \quad (1.13a)$$

$$y_t = Cx_t + e_t, \quad e_t \sim \mathcal{N}(0, R). \quad (1.13b)$$

Here, y_t is an n_y -dimensional vector of observations, x_t is an n_x -dimensional state vector and the system matrices A and C are of appropriate dimensions. The process and measurement noises are multivariate Gaussian with zero means and covariances Q and R , respectively.

Example 1.3 (Partially or fully dominated SSM). Assume that the measurement noise covariance R in Equation (1.13b) is full rank. Then, the observation kernel is Gaussian and dominated by Lebesgue measure. Hence, the model is partially dominated. If, in addition, the process noise covariance Q in Equation (1.13a) is full rank, then the transition kernel is also Gaussian and dominated by Lebesgue measure. In this case, the model is fully dominated.

However, for singular Q the model is degenerate (i.e., not fully dominated). Rank deficient process noise covariances arise in many applications, for instance if there is a physical connection between some of the states (such as between position and velocity).

A fully dominated LGSS model can equivalently be expressed as in Equation (1.2) with,

$$f(x_{t+1} | x_t) = \mathcal{N}(x_{t+1}; Ax_t, Q), \quad (1.14a)$$

$$g(y_t | x_t) = \mathcal{N}(y_t; Cx_t, R). \quad (1.14b)$$

LGSS models are without doubt one of the most important and well-studied classes of SSMs. There are basically two reasons for this. First, LGSS models provide sufficiently accurate descriptions of many interesting dynamical systems. Second, LGSS models are one of the few model classes, simple enough to allow for a fully analytical treatment.

When addressing inferential problems for SSMs, we are often asked to generate samples from the JSD, typically as part of an MCMC sampler used to learn a model of the system dynamics, as discussed above. For an LGSS model, the JSD is Gaussian and it can be computed using Kalman filtering and smoothing techniques (see e.g., [80]). Hence, we can make use of standard results for Gaussian distributions to generate a sample from $p(x_{1:T} | y_{1:T})$. This is possible for small T , but for increasing T it soon becomes infeasible due to the large matrix inversions involved.

To address this issue, it was recognized by [24, 56] that we can instead use the backward recursion (Equation (1.12)). It follows that the JSD can be factorized as,

$$p(x_{1:T} | y_{1:T}) = \left(\prod_{t=1}^{T-1} p(x_t | x_{t+1}, y_{1:t}) \right) p(x_T | y_{1:T}). \quad (1.15)$$

Initially, we generate a sample from the filtering density at time T ,

$$\tilde{x}_T \sim p(x_T | y_{1:T}). \quad (1.16a)$$

We then, successively, augment this *backward trajectory* by generating samples from the backward kernel,

$$\tilde{x}_t \sim p(x_t | \tilde{x}_{t+1}, y_{1:t}), \quad (1.16b)$$

for $t = T - 1, \dots, 1$. After a complete backward sweep, the backward trajectory $\tilde{x}_{1:T}$ is (by construction) a realization from the JSD (Equation (1.15)).

To compute the backward kernel, we first run a forward filter to find the filtering densities $p(x_t | y_{1:t})$ for $t = 1, \dots, T$. For an LGSS model, this is done by a standard Kalman filter [81]. It follows that the filtering densities are Gaussian according to,

$$p(x_t | y_{1:t}) = \mathcal{N}(x_t; \hat{x}_{t|t}, P_{t|t}), \quad (1.17)$$

for some tractable sequences of mean vectors $\{\hat{x}_{t|t}\}_{t \geq 1}$ and covariance matrices $\{P_{t|t}\}_{t \geq 1}$, respectively. From Equation (1.14a), we note that the transition density function is Gaussian and affine in x_t . Using Equations (1.11) and (1.17) and standard results on affine transformations of Gaussian variables, it then follows that

$$p(x_t | x_{t+1}, y_{1:t}) = \mathcal{N}(x_t; \mu_t, M_t), \quad (1.18a)$$

with

$$\mu_t = \hat{x}_{t|t} + P_{t|t} A^\top (Q + A P_{t|t} A^\top)^{-1} (x_{t+1} - A \hat{x}_{t|t}), \quad (1.18b)$$

$$M_t = P_{t|t} - P_{t|t} A^\top (Q + A P_{t|t} A^\top)^{-1} A P_{t|t}. \quad (1.18c)$$

Note that, if more than one sample is desired, multiple backward trajectories can be generated independently, without having to rerun the forward Kalman filter. We illustrate the backward simulator in the example below.

Example 1.4. To illustrate the possibility of generating samples from the JSD using backward simulation, we consider a first-order LGSS model,

$$\begin{aligned} x_{t+1} &= 0.9x_t + v_t, & v_t &\sim \mathcal{N}(0, 0.1), \\ y_t &= x_t + e_t, & e_t &\sim \mathcal{N}(0, 1), \end{aligned}$$

and $x_1 \sim \mathcal{N}(x_1; 0, 10)$. We simulate $T = 50$ samples $y_{1:T}$ from the model. Since the model is linear Gaussian, the marginal smoothing densities $p(x_t | y_{1:T})$ can be computed by running a Kalman filter followed by a Rauch–Tung–Striebel smoother [119]. However, we can also generate samples from the JSD $p(x_{1:T} | y_{1:T})$ by running a backward simulator. We simulate $M = 5000$ independent trajectories $\{\tilde{x}_{1:T}^j\}_{j=1}^M$, by first running a Kalman filter and then repeating the backward simulation procedure given by Equations (1.16) and (1.18) M times. Histograms over the simulated states at three specific time points, $t = 1$, $t = 25$ and $t = 50$, are given in Figure 1.3. As expected, the histograms are in close agreement with the true marginal smoothing distributions.

The strategy given by Equation (1.16), i.e., to sequentially sample (either exactly or approximately) from the backward kernel to generate a realization from the JSD, is what we collectively refer to as

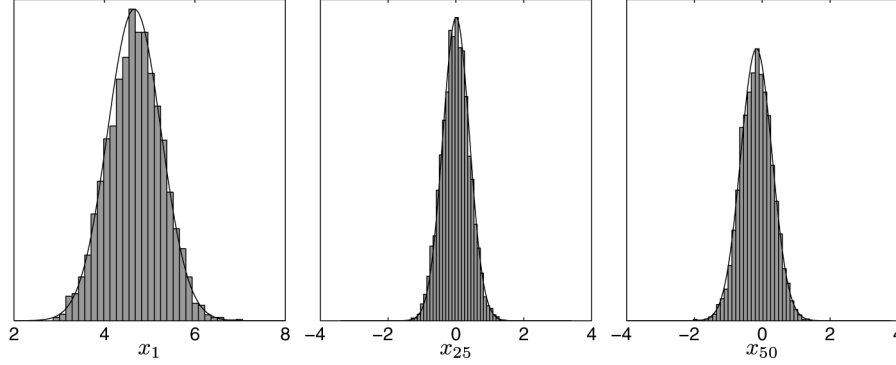


Fig. 1.3 Histograms of $\{\tilde{x}_t^j\}_{j=1}^M$ for $t = 1$, $t = 25$ and $t = 50$ (from left to right). The true marginal smoothing densities $p(x_t | y_{1:T})$ are shown as black lines.

backward simulation. We will now leave the world of LGSS models. In the remainder of this tutorial we address backward simulation for general nonlinear/non-Gaussian models. In these cases, the backward kernels will in general not be available in closed form. Instead, we will rely on SMC approximations of the kernels to carry out the backward simulation.

Before we leave this section, it should be noted that the backward simulator for LGSS models derived here is provided primarily to illustrate the concept. For LGSS models, more efficient samplers exist, e.g., based on disturbance simulation. See [30, 47, 146] for further details and extensions.

1.8 Outline

The rest of this tutorial is organized as follows. Section 2 reviews the two main Monte Carlo methods that are used throughout SMC and MCMC. The section is self-contained, but for obvious reasons it does not provide an in-depth coverage of these methods. Several references which may be useful for readers with no background in this area are given in Section 2.

Section 3 addresses SMC-based backward simulation for SSMs. The focus in this section is on smoothing in general nonlinear/non-Gaussian SSMs. More precisely, we discuss algorithms for generating

state trajectories, approximately distributed according to the joint smoothing distribution. These algorithms can be categorized as *particle smoothers*. Hence, readers with particular interest in smoothing problems may want to focus their attention on this section. However, smoothing is also addressed in Section 5 (see in particular Section 5.7), and the methods presented there can be useful alternatives to the particle smoothers discussed in Section 3.

Section 4 generalizes the backward simulation idea to latent variable models outside the class of SSMs. A general backward simulator is introduced and we discuss its properties and the type of models for which it is applicable. As a special case of the general backward simulator, we derive a Rao–Blackwellized particle smoother for conditionally linear Gaussian SSMs.

In Section 5, we discuss backward simulation in the context of so-called particle MCMC (PMCMC) methods. The focus in this section is on parameter inference, primarily in the Bayesian setting, but we also discuss PMCMC for maximum-likelihood-based inference. As mentioned above, the smoothing problem is also addressed. Finally, in Section 6 we conclude with a discussion about the various methods reviewed throughout this tutorial and outline possible directions for future work.

2

Monte Carlo Preliminaries

In this section, we review the two main Monte Carlo tools on which the subsequent development is based; sequential Monte Carlo (SMC) and Markov chain Monte Carlo (MCMC).

2.1 Sequential Monte Carlo

For general nonlinear non-Gaussian SSMs the backward kernel (Equation (1.10)) is not available on closed form. To address this issue, the basic idea that will be employed is to use an SMC approximation of the backward kernel. In this section, we review the basics of SMC applied to SSMs, also referred to as particle filtering.

Since the focus of this tutorial is on backward simulation, and not SMC, we will only present a basic SMC sampler and discuss some of its important properties. For a more in-depth treatment, we refer to one of the many tutorials and textbooks on the topic, see e.g., [10, 22, 38, 41, 66, 126]. A comprehensive collection of convergence results for SMC can be found in [31]. It should be emphasized that the backward simulators that are derived in the subsequent sections indeed rely on SMC approximations of the backward kernels, but not directly

on how these approximations are generated. Hence, it is straightforward to replace the basic sampler presented here with a more advanced SMC method, if desired.

2.1.1 The Particle Filter

SMC methods are based on importance sampling and resampling techniques to sequentially sample from a sequence of target densities. They can be seen as combinations of the sequential importance sampling [70] and the sampling/importance resampling [122] algorithms. The name particle filter (PF) is often used interchangeably with SMC, though here we reserve it for the case when the sequence of target densities is given by $p(x_{1:t} | y_{1:t})$ for $t = 1, 2, \dots$.

In a standard importance sampler, targeting $p(x_{1:t} | y_{1:t})$, we generate N independent samples $\{x_{1:t}^i\}_{i=1}^N$ from some proposal density $r_t(x_{1:t} | y_{1:t})$. These samples are then assigned importance weights,

$$w_t^i = \frac{p(x_{1:t}^i | y_{1:t})}{r_t(x_{1:t}^i | y_{1:t})}, \quad (2.1)$$

for $i = 1, \dots, N$. The weights can only be computed up to proportionality, since the normalization constant for the target density is unknown. However, this is easily coped with by normalizing the weights to sum to one. That is, we first compute the unnormalized importance weights

$$\bar{w}_t^i = \frac{p(x_{1:t}^i, y_{1:t})}{r_t(x_{1:t}^i | y_{1:t})}, \quad (2.2)$$

and then set $w_t^i = \bar{w}_t^i / \sum_l \bar{w}_t^l$.

Now, to allow for a sequential method, we construct the proposal density so that it factorizes according to,

$$\begin{aligned} r_t(x_{1:t} | y_{1:t}) &= r_t(x_t | x_{t-1}, y_t) r_{t-1}(x_{1:t-1} | y_{1:t-1}) \\ &= r_1(x_1 | y_1) \prod_{s=2}^t r_s(x_s | x_{s-1}, y_s). \end{aligned} \quad (2.3)$$

Hence, we can draw a sample from the proposal $x_{1:t}^i \sim r_t(x_{1:t} | y_{1:t})$ by first generating $x_1^i \sim r_1(x_1 | y_1)$, then $x_2^i \sim r_2(x_2 | x_1^i, y_2)$, etc. This

implies that the importance weights are given by,

$$w_t^i = \frac{p(x_{1:t}^i | y_{1:t})}{r_t(x_{1:t}^i | y_{1:t})} \propto \frac{g(y_t | x_t^i) f(x_t^i | x_{t-1}^i)}{r_t(x_t^i | x_{t-1}^i, y_t)} \frac{p(x_{1:t-1}^i | y_{1:t-1})}{r_{t-1}(x_{1:t-1}^i | y_{1:t-1})}. \quad (2.4)$$

Hence, if we define the weight function

$$W_t(x_{t-1}, x_t; y_t) = \frac{g(y_t | x_t) f(x_t | x_{t-1})}{r_t(x_t | x_{t-1}, y_t)}, \quad (2.5)$$

the importance weights can be computed as

$$\bar{w}_t^i = W_t(x_{t-1}^i, x_t^i; y_t) w_{t-1}^i, \quad (2.6)$$

and $w_t^i = \bar{w}_t^i / \sum_l \bar{w}_t^l$. We thus obtain a sequential updating formula also for the importance weights. The initial weight function at time $t = 1$ is given by $W_1(x_1; y_1) = g(y_1 | x_1) p(x_1) / r_1(x_1 | y_1)$. As indicated by the notation, the proposal at time t is allowed to depend on both the previous state x_{t-1} and on the current observation y_t . This is important in practice, to be able to make good use of the available information when designing the proposal.

In the SMC literature, the samples $\{x_{1:t}^i\}_{i=1}^N$ are called particles and the collection $\{x_{1:t}^i, w_t^i\}_{i=1}^N$ is referred to as a *weighted particle system*. This weighted sample defines an empirical point-mass distribution on $\mathbf{X}^t$,

$$\hat{p}^N(dx_{1:t} | y_{1:t}) \triangleq \sum_{i=1}^N w_t^i \delta_{x_{1:t}^i}(dx_{1:t}), \quad (2.7)$$

which is an approximation of the target distribution. Equivalently, for any test function $\varphi : \mathbf{X}^T \rightarrow \mathbb{R}$ we can construct the estimator

$$\hat{\varphi}_{t|t}^N \triangleq \sum_{i=1}^N w_t^i \varphi(x_{1:t}^i) \approx \mathbb{E}[\varphi(x_{1:t}) | y_{1:t}]. \quad (2.8)$$

This estimator is consistent, as $N \rightarrow \infty$, and a central limit theorem holds (see e.g., [31, 87, 36]).

The procedure resulting from sampling according to Equation (2.3) and computing the weights (Equation (2.6)) is the sequential importance sampler [70], dating back to the late 60's. Though applicable for short data lengths, a serious drawback with this method is that the

weight update (Equation (2.6)) is unstable. More precisely, the variance of the normalized importance weights increases over time [23, p. 232], typically at an exponential rate. This has the effect that all but one of the weights decreases to zero, and all emphasis is thus put on one of the particles (recall that the weights are normalized to sum to one). This does not come as a surprise, since we are in fact applying an importance sampler in a space with dimension increasing with t . In high dimensions, even what appears to be a small discrepancy between the proposal and the target densities will result in poor performance of the sampler (see e.g., [41] for an illustrative example).

A way to mitigate this problem was proposed by [129, 64], resulting in the first functional SMC sampler. The idea is to rejuvenate the sample by replicating particles with high weights and discarding particles with low weights. This is done by resampling the particle system, similar to the sampling/importance resampling method [122]. Since the weighted particle system $\{x_{1:t}^i, w_t^i\}_{i=1}^N$ approximates the target according to Equation (2.7), we can generate a new, unweighted set of particles by sampling independently from Equation (2.7). That is, we set $\tilde{x}_{1:t}^j = x_{1:t}^i$ with probability w_t^i , for $j = 1, \dots, N$. The equally weighted particle system $\{\tilde{x}_{1:t}^j, \frac{1}{N}\}_{j=1}^N$ can then be used to construct an estimator similar to Equation (2.8),

$$\hat{\varphi}_{t|t}^N \triangleq \frac{1}{N} \sum_{j=1}^N \varphi(\tilde{x}_{1:t}^j) \approx \mathbb{E}[\varphi(x_{1:t}) \mid y_{1:t}]. \quad (2.9)$$

It should be noted that the resampling introduces some additional variance, so the estimator in Equation (2.9) will be dominated by the estimator in Equation (2.8). However, when applied sequentially, the resampling is critical since it allows us to put focus on the promising particles and discard the improbable ones. We summarize the PF in Algorithm 1.

A common choice in practice is to run the PF with $r_t(x_t \mid x_{t-1}, y_t) = f(x_t \mid x_{t-1})$, i.e., we propose samples according to the transition density function. This results in a vanilla method referred to as the *bootstrap filter*. If possible, it is recommended to use a proposal density which takes the current observation y_t into account. The *optimal proposal* is given by the conditional density $r_t(x_t \mid x_{t-1}, y_t) = p(x_t \mid x_{t-1}, y_t)$. This

Algorithm 1 Particle filter (all operations are for $i = 1, \dots, N$)

- 1: Draw $x_1^i \sim r_1(x_1 | y_1)$.
 - 2: Compute $\bar{w}_1^i = W_1(x_1^i; y_1)$.
 - 3: Normalize: set $w_1^i = \bar{w}_1^i / \sum_l \bar{w}_1^l$.
 - 4: **for** $t \geq 2$ **do**
 - 5: Resample with $P(\tilde{x}_{t-1}^i = x_{t-1}^j) = w_{t-1}^j$.
 - 6: Draw $x_t^i \sim r_t(x_t | \tilde{x}_{t-1}^i, y_t)$ and set $x_{1:t}^i = \{\tilde{x}_{1:t-1}^i, x_t^i\}$.
 - 7: Compute $\bar{w}_t^i = W_t(x_t^i, \tilde{x}_{t-1}^i; y_t)$.
 - 8: Normalize: set $w_t^i = \bar{w}_t^i / \sum_l \bar{w}_t^l$.
 - 9: **end for**
-

choice of proposal density will minimize the variance of the incremental importance weights at time t [41]. Unfortunately, it is often the case that this density is not available in practice, but various approximations can be used instead.

The resampling scheme outlined above is known as multinomial resampling. Alternative methods that introduce less variance exist, e.g., residual resampling [144, 98] and stratified resampling [83]. Furthermore, instead of resampling at every iteration, it is recommended to do so only when there is need for it. For this cause, it is common to introduce a measure of imbalance of the importance weights, such as the effective sample size (ESS) [97, Section 2.5]. Resampling is then only carried out when the ESS drops below some prespecified threshold.

Another modification of the resampling step is used in the auxiliary particle filter (APF) by [116]. The idea is to use an adjusted proposal distribution in the resampling step. This is accomplished by computing a set of weights $\{w_t^i \nu_t^i\}_{i=1}^N$, and using these (after normalization) as probabilities in the resampling. Here, the adjustment weights ν_t^i are user defined. They allow us to, for instance, take the observation y_{t+1} into account when resampling the particles at time t . The fact that we modify the proposal for the resampling is compensated for in the importance weight computation. If the APF is run with an optimal proposal kernel and with adjustment weights given by $\nu_t^i = p(y_{t+1} | x_t^i)$, the importance weights will be identically equal to one. In this case, the filter is said to be *fully adapted*. See [116, 79, 37] for more information about the APF and how the adjustment weights can be chosen.

2.1.2 Path Degeneracy

As pointed out in the previous section, the PF targets the sequence of JSDs, $p(x_{1:t} | y_{1:t})$. By keeping track of the *genealogy* of the particle filter, i.e., the ancestral dependence of the particles, we obtain weighted particle trajectories $\{x_{1:t}^i, w_t^i\}_{i=1}^N$. These provide an approximation, not only of the filtering distribution, but of the complete joint smoothing distribution as in Equation (2.7) [83]. It thus appears as if the PF solves the general state inference problem, since any smoothing density can be attained from $p(x_{1:t} | y_{1:t})$ by marginalization. However, this is not truly the case.

To see why, let s be some fixed time point and assume that we apply a PF to approximate the marginal smoothing density $p(x_s | y_{1:t})$ for $t \geq s$. This is given straightforwardly from Equation (2.7) by simply discarding everything except the s th time point from the particle trajectories $\{x_{1:t}^i\}_{i=1}^N$. Now, if $t = s$ (i.e., we are in fact looking at the filtering density) we have N unique particles in the point-mass approximation. However, each time we resample the particle system, the unique number of particles at time s will decrease. This is in fact the purpose of the resampling, to remove particles with small weights and duplicate particles with large weights. Consequently, for large enough $t \gg s$, all the particles $\{x_{1:t}^i\}_{i=1}^N$ will share a common ancestor at time s , due to the consecutive resampling steps. This problem, known as *path degeneracy*, is further illustrated in the example below.

Example 2.1 (Path degeneracy). A PF with $N = 30$ particles is applied to a one-dimensional Gaussian random walk process measured in Gaussian noise. At time $t = 50$ the JSD is targeted by a weighted particle system $\{x_{1:50}^i, w_{50}^i\}_{i=1}^{30}$. Figure 2.1 depicts the particle trajectories. As can be seen, all particles share a common ancestor at time $s = 32$.

Assume, for instance, that we are interested in the smoothed estimate of the initial state, $\mathbb{E}[x_1 | y_{1:50}]$. We thus construct an estimator,

$$\hat{x}_{1|50} = \sum_{i=1}^{30} w_{50}^i x_1^i,$$

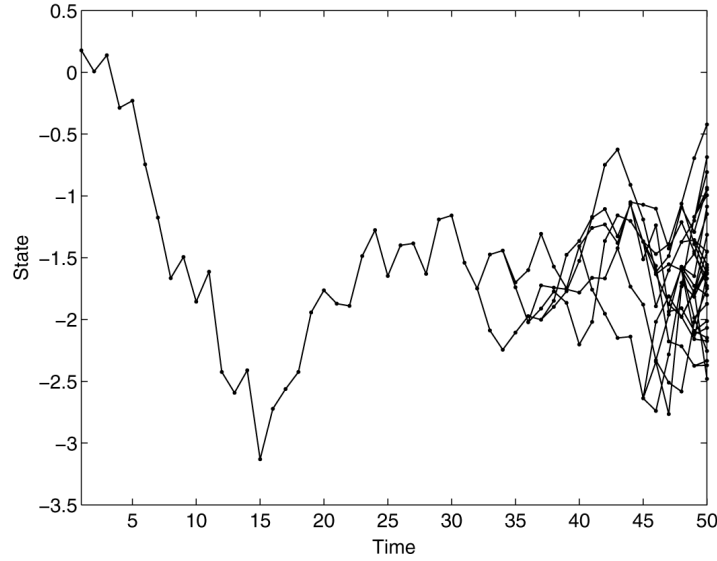


Fig. 2.1 Particle trajectories at time $t = 50$. For any time $s \leq 32$, all trajectories coincide.

but since x_1^i are identical for all $i = 1, \dots, 30$, this is in effect a Monte Carlo integration using a single sample. Hence, due to the path degeneracy, we cannot expect to obtain an accurate estimate of $\mathbb{E}[x_1 \mid y_{1:50}]$ from the PF.

Path degeneracy arises as an effect of resampling, but it should rather be thought of as a manifestation of a deeper problem, namely the degeneracy of the importance weights. In the sequential importance sampler, all weight will be put on a single sample for large enough t , having a similar effect as path degeneracy. This issue is in fact to a large extent mitigated by introducing resampling.

Due to path degeneracy, the PF is in practice used mostly for filtering (hence the name) and fixed-lag smoothing. At time t or $s \lesssim t$, the particle diversity is in general high, leading to accurate approximations of the filtering and fixed-lag smoothing distributions. How SMC can be used to address the marginal and joint smoothing problems will be the topic of Section 3.

2.2 Markov Chain Monte Carlo

Another comprehensive class of Monte Carlo methods is MCMC. In this section, we review some of the basic concepts of MCMC that are of key relevance. For further reading, see the tutorial papers [2, 128] or one of the many textbooks, e.g., [20, 97, 120].

2.2.1 Markov Chains and Limiting Distributions

MCMC methods allow us to approximately generate samples from an arbitrary target distribution $f(z)$ on some space Z . More precisely, an MCMC sampler will simulate a Markov chain, or Markov process, $\{z[r]\}_{r \geq 0}$ which is constructed in such a way that the limiting distribution of the chain is given by $f(z)$. The sample path of the Markov chain can then be used to draw inference about the target distribution. For instance, the target can be the JSD $p(x_{1:T} | y_{1:T})$, which means that MCMC can be used as an alternative to SMC for addressing the state inference problem. However, we will primarily make use of MCMC in a different way, either as a component of SMC or vice versa. Hence, in this section, the methods are presented in a general setting, which is specialized to different scenarios in the consecutive sections.

A Markov chain $\{z[r]\}_{r \geq 0}$ is completely specified by its initial distribution $\mu(z)$ and its transition kernel $K(z' | z)$ (for simplicity, we assume that there exists a dominating measure and express all distributions in terms of their probability densities). Here, $K(z' | z)$ encodes the probability density for the consecutive state of the chain, denoted as z' , given that the current state is z . That is, the chain can be simulated according to $z[0] \sim \mu(z)$ and $z[r] \sim K(z' | z[r-1])$ for $r = 1, 2, \dots$. As an example, the state process $\{x_t\}_{t \geq 1}$ of the SSM (Equation (1.2)) is a Markov chain. However, the view on this process is quite different, as $\{x_t\}_{t \geq 1}$ is thought of as the internal state of a physical system evolving over time. On the contrary, the Markov chain $\{z[r]\}_{r \geq 0}$ generated in an MCMC sampler is simulated as part of an inferential method, and it does not have a physical interpretation in the same sense.

The Markov chains encountered in MCMC have, by construction, a special stability property, namely that of a *stationary distribution*. That is, there exist some distribution g such that, if $z[r]$ is marginally

distributed according to g , then so is $z[r + 1]$ (and therefore all $z[m]$ for $m \geq r$). If g is a stationary distribution for the Markov kernel K , then we say that K leaves g *invariant*. Furthermore, if the marginals of the chain approach the stationary distribution (in total variation), it is referred to as the *limiting distribution* of the chain. An important consequence of this convergence property is that the sample path average,

$$\frac{1}{R} \sum_{r=0}^R \varphi(z[r]), \quad (2.10)$$

converges almost surely to the expectation $\mathbb{E}_g[\varphi(z)] = \int \varphi(z)g(z) dz$. This result, known as the ergodic theorem (see e.g., [120, Section 6.6]), ensures that the Markov chain can be used to compute Monte Carlo estimators of expectations under the stationary distribution.

There are two properties of the chain that are sufficient for a stationary distribution to be also a limiting distribution; irreducibility and aperiodicity. The Markov chain is said to be *g-irreducible* if, for any initial state, it has positive probability of entering any set which has a positive probability under g . The chain is periodic if, with probability 1, certain subsets only can be visited at regularly spaced intervals. If the chain is not periodic, it is said to be *aperiodic*. For an irreducible and aperiodic chain with stationary distribution g , the marginals converge to the stationary distribution.

Theorem 2.1. Let $\{z[r]\}_{r \geq 0}$ be a g -irreducible and aperiodic Markov chain with stationary distribution g . Then, for g -almost all starting points,

$$\|\mathcal{L}(z[r] \in \cdot) - g(\cdot)\|_{\text{TV}} \rightarrow 0,$$

as $r \rightarrow \infty$, where $\mathcal{L}(z[r] \in \cdot)$ is the marginal distribution of $z[r]$ and $\|\cdot\|_{\text{TV}}$ is the total variation norm.

Proof. See [134, Theorem 1]. □

Stronger forms of convergence are also common in the MCMC literature, such as geometric ergodicity and uniform ergodicity. It is

also possible to establish central limit theorems for the estimator in Equation (2.10). There is by now a well-developed theory on Markov chains, safeguarding the theoretical validity of MCMC methods, see e.g., [104, 120, 134].

2.2.2 Metropolis–Hastings Algorithm

A sample path from a Markov chain with a specific limiting distribution can be used to compute estimators according to Equation (2.10). However, since we are interested in computing expectations under a given target distribution $f(z)$, it remains to construct the chain so that $f(z)$ is its limiting distribution. To accomplish this, we have freedom in the choice of transition kernel $K(z' | z)$, as long as the kernel is chosen in such a way that it is possible to simulate the Markov chain.

A property which is sufficient for $f(z)$ to be a stationary distribution of the chain is that the kernel satisfies *detailed balance*,

$$f(z)K(z' | z) = f(z')K(z | z') \quad \text{for } (z, z') \in \mathbb{Z}^2. \quad (2.11)$$

Based on this criterion, it is possible to construct a Markov kernel with the correct stationary distribution using an accept/reject procedure. This results in the Metropolis–Hastings (MH) algorithm [102, 71], which is the main tool in the MCMC toolbox.

The idea behind the MH algorithm is to generate samples from some arbitrary proposal kernel $q(z' | z)$. The simulation is followed by an accept/reject decision to make sure that detailed balance holds for the given target distribution. The MH acceptance probability, for a move from z to z' , is given by

$$\rho(z, z') = 1 \wedge \frac{f(z') q(z | z')}{f(z) q(z' | z)}, \quad (2.12)$$

where $a \wedge b = \min(a, b)$. That is, with $z[r - 1]$ being the previous state of the Markov chain at iteration r , we proceed by sampling a candidate value $z' \sim q(z' | z[r - 1])$. With probability $\rho(z[r - 1], z')$, the sample is accepted and we set $z[r] = z'$. Otherwise, the sample is rejected and we set $z[r] = z[r - 1]$. It is interesting to note that the acceptance probability depends on the target density only through the ratio $f(z')/f(z)$.

Algorithm 2 Metropolis–Hastings sampler

-
- 1: Set $z[0]$ arbitrarily.
 - 2: **for** $r \geq 1$ **do**
 - 3: Draw $z' \sim q(z' \mid z[r-1])$.
 - 4: With probability

$$\rho(z[r-1], z') = 1 \wedge \frac{f(z')}{f(z[r-1])} \frac{q(z[r-1] \mid z')}{q(z' \mid z[r-1])},$$

set $z[r] = z'$, otherwise set $z[r] = z[r-1]$.

- 5: **end for**
-

Hence, it is possible to run the MH algorithm even when the normalizing constant of the target density is unknown, which is key to its applicability in practice. The MH sampler is summarized in Algorithm 2.

The transition kernel given by the sampling procedure above is,

$$K(dz' \mid z) = \rho(z, z')q(dz' \mid z) + (1 - \eta(z))\delta_z(dz'), \quad (2.13)$$

with $\eta(z) = \int \rho(z, z')q(z' \mid z)dz'$. It can be verified that this kernel satisfies the detailed balance condition (Equation (2.11)). Consequently, $f(z)$ is a stationary distribution of the chain. Under additional, mild, assumptions (basically, positivity of the proposal kernel), it can also be verified that the chain is irreducible and aperiodic, which by Theorem 2.1 implies that $f(z)$ is the limiting distribution of the Markov chain.

In practice, the choice of the proposal kernel is critical. If the proposal is not chosen appropriately, the average acceptance probability can be very low. This results in that the chain gets “stuck” at the same value for many iterations. We say that the chain suffers from poor *mixing*. Furthermore, a high acceptance probability does not necessarily imply good mixing. Indeed, a proposal kernel which only makes small, local moves can yield a high acceptance probability, but at the same time a very poor exploration of the state-space. To construct a proposal kernel with good mixing properties is particularly challenging in high dimensions. To address this issue, sophisticated proposal mechanisms have been developed for the MH algorithm. One approach is to construct a proposal kernel based on the simulation of

a Hamiltonian dynamical system, leading to the so-called Hamiltonian Monte Carlo method [44] (see also [109]). See [61, 78, 124] for applications and extensions of this method. Another approach is to make use of SMC within MCMC, which is the idea that underlies the particle MCMC framework [3]. These methods will be discussed in more detail in Section 5.

2.2.3 Gibbs Sampling

The Gibbs sampler [58, 57] is an MCMC method of particular interest. It can be seen as a special case of the MH algorithm. However, it differs both in terms of basic methodology and historical motivation. Assume that the random variable of interest can be partitioned according to $z = \{z_1, \dots, z_p\}$, where the z_i s can be either uni- or multidimensional. A Markov kernel is then constructed by sampling each component from its full conditional under f , i.e.,

$$z'_i \mid z_1, \dots, z_{i-1}, z_{i+1}, \dots, z_p \sim f_i(z_i \mid z_1, \dots, z_{i-1}, z_{i+1}, \dots, z_p). \quad (2.14)$$

The sampling procedure given above leaves $f(z)$ invariant, i.e., if z is distributed according to f , then so is $\{z_1, \dots, z_{i-1}, z'_i, z_{i+1}, \dots, z_p\}$. Hence, sampling each of the components in this manner, either in a deterministic or in a random order, defines a Markov kernel on $\mathbf{Z}$, with stationary distribution $f(z)$. Since only the conditionals $f_1, \dots, f_p$ are used in the sampling procedure, all of the simulations can be univariate (or low-dimensional), even for high-dimensional problems, which is a key property of the Gibbs sampler. We summarize the method in Algorithm 3.

The specific parameterization that is used in the construction of a Gibbs sampler can have very significant effects on its performance, see e.g., [72, 121]. If the individual components $z_1, \dots, z_p$ are strongly dependent, sampling from the full conditionals can lead to insignificant updates of the components, resulting in a poorly mixing Gibbs sampler. An example of this phenomena, which is of particular interest to the present work, is single-state Gibbs sampling for SSMs. Assume that the target distribution is the JSD $p(x_{1:T} \mid y_{1:T})$. A natural parameterization for a Gibbs sampler is to draw the individual states at different time

Algorithm 3 Gibbs sampler (deterministic scan)

-
- 1: Set $z[0]$ arbitrarily.
 - 2: **for** $r \geq 1$ **do**
 - 3: Sample according to

$$\begin{aligned}
& z_1[r] \sim f_1(z_1 \mid z_2[r-1], \dots, z_p[r-1]). \\
& \quad \vdots \\
& z_i[r] \sim f_i(z_i \mid z_1[r], \dots, z_{i-1}[r], z_{i+1}[r-1], \dots, z_p[r-1]). \\
& \quad \vdots \\
& z_p[r] \sim f_p(z_p \mid z_1[r], \dots, z_{p-1}[r]).
\end{aligned}$$

- 4: **end for**
-

points from their full conditional, i.e.,

$$x'_t \sim p(x_t \mid x_{1:t-1}, x_{t+1:T}, y_{1:T}), \quad (2.15)$$

for $t = 1, \dots, T$. This leads to the single-state sampler which was used in the illustrating example in Section 1.3. As was experienced in this example, the resulting Gibbs sampler can suffer from very poor mixing. The reason is that there is often a strong dependence between consecutive states in an SSM. Hence, when x'_t is sampled, the resulting value will be in close agreement with the conditioning on x_{t-1} and x_{t+1} , resulting in small updates at each iteration.

Two important techniques, used to improve the mixing of the Gibbs sampler, are *grouping* and *collapsing*. To illustrate these concepts, assume that the variable of interest is partitioned into three parts, $z = \{z_1, z_2, z_3\}$. The basic Gibbs sampler would then, at each iteration, simulate according to

$$z'_1 \sim f_1(z_1 \mid z_2, z_3), \quad z'_2 \sim f_2(z_2 \mid z'_1, z_3), \quad z'_3 \sim f_3(z_3 \mid z'_1, z'_2).$$

Grouping, or blocking, amounts to sample a group of variable, say $\{z_2, z_3\}$, jointly. That is, we simulate according to

$$z'_1 \sim f_1(z_1 \mid z_2, z_3), \quad \{z'_2, z'_3\} \sim f_{2,3}(z_2, z_3 \mid z'_1).$$

This corresponds to a reparameterization of the Gibbs sampler, with only two components instead of three. If z_2 and z_3 are strongly dependent, sampling them jointly can result in much larger updates, improving the mixing of the basic Gibbs sampler.

Collapsing is another word for marginalization. In the simplest case, one component of the model, say z_3 , is marginalized in all the steps of the sampler. That is, we simulate according to

$$z'_1 \sim \bar{f}_1(z_1 | z_2), \quad z'_2 \sim \bar{f}_2(z_2 | z'_1),$$

where $\bar{f}_i, i = 1, 2$, are the full conditionals under the marginal distribution $\bar{f}(z_1, z_2) = \int f(z_1, z_2, z_3) dz_3$. This corresponds to a standard Gibbs sampler, targeting the marginal distribution $\bar{f}$ instead of f .

Perhaps more interestingly, the Gibbs sampler does not have to be fully collapsed, as above, to attain an improvement in mixing. In a *partially collapsed* Gibbs sampler, we marginalize a certain variable only in some of the Gibbs steps. For instance, we could simulate according to

$$z'_1 \sim f_1(z_1 | z_2, z_3), \quad z'_2 \sim \bar{f}_2(z_2 | z'_1), \quad z'_3 \sim f_3(z_3 | z'_1, z'_2).$$

This partially collapsed Gibbs sampler results in a valid MCMC method, with stationary distribution $f(z_1, z_2, z_3)$. Again, if z_2 and z_3 are strongly dependent, sampling z_2 unconditionally on z_3 enables a larger update, which in turn results in a larger update also of z_3 .¹

However, care needs to be taken when constructing a partially collapsed sampler, since the order of the Gibbs steps in fact can affect the stationary distribution. If, for instance, we would collapse all the Gibbs steps to a maximum degree and sample each of the variables z_1 , z_2 and z_3 from their respective marginals under f , then we would lose all correlation between the variables. A more intricate example of an improperly collapsed sampler is,

$$z'_1 \sim \bar{f}_1(z_1 | z_2), \quad z'_2 \sim f_2(z_2 | z'_1, z_3), \quad z'_3 \sim f_3(z_3 | z'_1, z'_2).$$

This sampling scheme is seemingly similar to the one above, but this latter scheme does not leave $f(z_1, z_2, z_3)$ invariant. Again, the reason is

¹ In fact, in this specific example, the partially collapsed Gibbs sampler is equivalent to the grouped Gibbs sampler.

that the sampler does not take the dependences between the variables into account in a correct way. To see why this is the case and how to construct a *properly collapsed* Gibbs sampler, see [137].

Grouping and collapsing will in general improve the mixing properties of the Gibbs sampler (see [97, Section 6.7]). In practice, the gain can be quite considerable. In the motivating example in Section 1.3, we pointed out that a backward simulator aims at sampling the state sequence $x_{1:T}$ jointly from the JSD, instead of resorting to single-state sampling of the individual state components. In light of the above discussion, we see that the backward simulator thus is a way of grouping the state variables to alleviate the strong dependencies between consecutive states.

A different technique which can also be useful is to reparameterize the model by making a change of variables. If the new variables are chosen in a way which decreases the posterior interdependence, then the mixing of the Gibbs sampler can be improved; see [114] and Section 4.6.3.

To be able to implement the Gibbs samplers described above, we need to be able to sample from all the involved conditionals under the target distribution f . While this is indeed possible for many interesting problems, it is not always the case. To alleviate this, one possibility is to use a mixed strategy, in which Gibbs steps are used whenever possible and MH steps are used for the intractable conditionals. Since the MH sampler leaves its target distribution invariant, each step of the mixed sampler will leave f invariant, resulting in a valid MCMC kernel. This approach is sometimes referred to as *Hastings-within-Gibbs*.

3

Backward Simulation for State-Space Models

SMC provides an approximation of the joint smoothing distribution, given by Equation (2.7). However, as discussed in Section 2.1.2, path degeneracy makes this approximation unreliable for anything but filtering or fixed-lag smoothing with a short enough lag. In the present section, we will see how the joint smoothing problem can be addressed by complementing the forward filter with a second recursion, evolving in the time-reversed direction. In particular, we will make use of SMC to enable backward simulation for general SSMs, allowing us to generate samples approximately distributed according to the JSD.

A related approach is the two-filter smoother [17], which is based on one filter moving forward in time and one filter moving backward in time. When the two filters meet “in the middle”, the information is merged, enabling computation of smoothed estimates. This approach will not be considered here, and for further reading on SMC implementations of the two-filter smoother we refer to [18, 51].

3.1 Forward Filter/Backward Simulator

As was recognized in Section 1.7, it is possible to make use of the backward recursion (Equation (1.12)) to generate samples from the JSD. The key ingredient is the backward kernel. Let us assume that the model under study is fully dominated. The backward kernel density is then given by Equation (1.11). As previously pointed out, this expression depends explicitly on the filtering density $p(x_t | y_{1:t})$. The basic idea, underlying the particle-based forward filter/backward simulator (FFBSi) [62, 40], is to make use of a PF to approximate the backward kernel.

Assume that we have recorded a sequence of observations $y_{1:T}$ up to some final time point T . Assume also that we have applied a PF to this batch of data. For each time $t = 1, \dots, T$ we thus have a weighted particle system $\{x_t^i, w_t^i\}_{i=1}^N$ approximating the filtering distribution at time t ,

$$\hat{p}^N(dx_t | y_{1:t}) \triangleq \sum_{i=1}^N w_t^i \delta_{x_t^i}(dx_t). \quad (3.1)$$

These approximations are given by marginalization of Equation (2.7), or, put differently, by discarding everything from the particle trajectories except for the last time point. Using Equation (3.1) in Equation (1.11), we obtain an approximation of the backward kernel,

$$\hat{B}_t^N(dx_t | x_{t+1}) \triangleq \sum_{i=1}^N \frac{w_t^i f(x_{t+1} | x_t^i)}{\sum_l w_t^l f(x_{t+1} | x_t^l)} \delta_{x_t^i}(dx_t). \quad (3.2)$$

It is important to note that the particles $\{x_t^i, w_t^i\}_{i=1}^N$ that are used to approximate the filtering distribution in Equation (3.1) are those that were generated at time t in the PF. Hence, there is in general a rich diversity of particles in this system, and the approximations in Equations (3.1) and (3.2) do not suffer (directly) from path degeneracy. This is also the reason for why FFBSi can succeed where the PF fails. Even though the PF results in a degenerate approximation of the JSD, it can generally provide accurate approximations of the filtering distributions, which is all that is needed to compute an approximation of the backward kernel.

We can now make use of Equations (3.1) and (3.2) to generate a backward trajectory by sampling (cf. Equation (1.16)),

$$\tilde{x}_T \sim \hat{p}^N(dx_T | y_{1:T}), \quad (3.3a)$$

$$\tilde{x}_t \sim \hat{B}_t^N(dx_t | \tilde{x}_{t+1}), \quad (3.3b)$$

for $t = T - 1, \dots, 1$. The backward trajectory $\tilde{x}_{1:T}$ is an approximate realization from $p(x_{1:T} | y_{1:T})$. How close the distribution of $\tilde{x}_{1:T}$ is to the JSD, clearly depends on how accurate the PF approximations in Equations (3.1) and (3.2) are. The convergence properties of FFBSi will be the topic of Section 3.2.2.

Sampling according to Equation (3.3a) simply consists of drawing among the filter particles at time T with $P(\tilde{x}_T = x_T^i) = w_T^i$. Similarly, the empirical backward kernel has finite support and it can be written as

$$\hat{B}_t^N(dx_t | \tilde{x}_{t+1}) = \sum_{i=1}^N \tilde{w}_{t|T}^i \delta_{x_t^i}(dx_t), \quad (3.4)$$

where we have defined the smoothing weights

$$\tilde{w}_{t|T}^i = \frac{w_t^i f(\tilde{x}_{t+1} | x_t^i)}{\sum_l w_t^l f(\tilde{x}_{t+1} | x_t^l)}. \quad (3.5)$$

Hence, to sample according to Equation (3.3b), we evaluate the smoothing weights $\{\tilde{w}_{t|T}^i\}_{i=1}^N$ and draw among the forward filter particles at time t with $P(\tilde{x}_t = x_t^i) = \tilde{w}_{t|T}^i$. Note that, in general, $\tilde{x}_t$ will differ from the ancestor particle of $\tilde{x}_{t+1}$, resulting in an increased particle diversity.

We emphasize that Equation (3.2) relies on the assumption that the model under study is fully dominated. Applying backward simulation to degenerate (non fully dominated) models, turns out to be much more tricky. We will return to this issue in Section 4.

When using FFBSi to address the smoothing problem, we typically generate multiple backward trajectories $\{\tilde{x}_{1:T}^j\}_{j=1}^M$, by repeating the backward simulation (Equation (3.3)) M times. Hence, conditionally on the forward filter particles, the collection $\{\tilde{x}_{1:T}^j\}_{j=1}^M$ are i.i.d. samples. The conditional distribution of these samples will be given explicitly in Section 3.2.1. The backward trajectories define an unweighted