Comparing Two Recent Particle Filter Implementations of Bayesian System Identification*

Andreas Svensson* Thomas B. Schon*

* Department of Information Technology, Uppsala University, Sweden
(e-mail: andreas.svensson@it.uu.se, thomas.schon@it.uu.se)

Abstract: Bayesian system identification is a theoretically well-founded and currently emerging area. We describe and evaluate two recent state-of-the-art sample-based methods for Bayesian parameter inference from the statistics literature, particle Metropolis-Hastings (PMH) and SMC^2, and apply them to a non-trivial real world system identification problem with large uncertainty present. We discuss their different properties from a user perspective, and conclude that they show similar performance in practice, while PMH is significantly easier to implement than SMC^2.

Keywords: System identification, particle filters, Monte Carlo methods, Bayesian inference

1. INTRODUCTION

In this paper, we are concerned with methods for learning unknown parameters in nonlinear state space models, i.e., gray box identification. We write the state space model as

\begin{align}
x_{t+1} | x_t & \sim f_\theta(x_{t+1} | x_t), \\
y_t | x_t & \sim g_\theta(y_t | x_t),
\end{align}

where \( y_t \in \mathbb{R}^{n_y} \) is the observed output, \( x_t \in \mathbb{R}^{n_x} \) is the state, and an exogenous input \( u_t \in \mathbb{R}^{n_u} \) may be included in \( f \). We let \( f \) and \( g \) denote probability densities (i.e., including stochastic noise), depending on an (unknown) parameter vector \( \theta \in \mathbb{R}^{n_\theta} \), and by \( \sim \) we mean distributed according to the density. We write \( p_\theta(y_{1:T}) \) to denote the likelihood of the model (1), and \( y_{1:T} \equiv \{y_1, \ldots, y_T\} \).

Traditionally, the system identification literature (Ljung, 1999; Söderström and Stoica, 1989) has mostly been focused on maximum likelihood (ML) point estimates \( \hat{\theta}_{\text{ML}} \) of the unknown parameter \( \theta \),

\[ \hat{\theta}_{\text{ML}} = \arg \max_\theta p_\theta(y_{1:T}) \tag{2} \]

rather than inferring its posterior distribution using Bayes’ theorem as

\[ p(\theta | y_{1:T}) \propto p_\theta(y_{1:T}) p(\theta), \tag{3} \]

where \( p(\theta) \) denotes the prior belief on \( \theta \). An excellent introduction to Bayesian system identification is given by Peterka (1981).

The traditional focus on the ML problem is indeed for a good reason: it is often computationally easier (and more natural) to handle a single parameter value than a full distribution. When the parameters are far from being uniquely determined by the data, however, the posterior distribution automatically gives a quantification of the amount of uncertainty present. Such situations may occur when, e.g., there is a model mismatch, the system was not excited enough during data collection, or the data record is very short.

The development of methods for Bayesian identification has made big advances in recent years, in particular methods based on sequential Monte Carlo (SMC) (Schön et al., 2015; Kantas et al., 2015). An important part of the advances has been published in the statistics literature, with a distinct focus on theoretical properties, such as consistency and convergence (Del Moral, 2004; Chopin, 2004). We believe these methods have a potential of being of great use also in practice. This paper compares two different state-of-the-art algorithms, and evaluate their properties from an engineering perspective.

We will focus on two popular methods, Particle Metropolis-Hastings (PMH, Andrieu et al., 2010) and SMC^2 (Chopin et al., 2013; Fulop and Li, 2013). Both methods are developed for being off-the-shelf methods for parameter learning problems, requiring nothing more than the ability to

(a) simulate from \( f_\theta(x_{t+1} | x_t) \),
(b) evaluate \( g_\theta(y_t | x_t) \).

The requirements (a) and (b) are fulfilled in most engineering applications. We will give a brief introduction (complete enough so the user can implement the methods on her/his own), and evaluate their performance on two examples. A more extensive introduction can be found in any of the recent tutorials by Schön et al. (2015), Kantas et al. (2015) or Dahlin and Schön (2015).

---

* This work was supported by the Swedish Research Council (VR), project Probabilistic modeling of dynamical systems with contract number 621-2013-5524.
The basic (bootstrap) particle filter

1: Draw \( x_0^{(i)} \sim p(x_0) \) and set \( w_0^{(i)} = 1 \).
2: for \( t = 1 \) to \( T \) do
3: \( a_{t-1}^{(i)} \) with \( \mathbb{P}(a_{t-1}^{(i)} = j) \propto w_{t-1}^{(j)} \) (resampling).
4: \( x_t^{(i)} \sim f_0(x_t|x_{t-1}^{(i)}) \) (propagation).
5: Set \( w_t^{(i)} = g_0(y_t|x_t^{(i)}) \) (weighting).
6: end for

All statements with \( (i) \) are for \( i = 1, \ldots, N_x \).

2. THE PMH AND SMC\(^2\) ALGORITHMS

We assume the reader has some familiarity with particle filters (Doucet and Johansen, 2011), and summarize the bootstrap particle filter as Algorithm 1. The particle filter is perhaps most known in the signal processing literature as Algorithm 1. The particle filter filters (Doucet and Johansen, 2011), and summarize the likelihood of the parameters \( \theta \). The likelihood is estimated by sequentially adding more and more data.

The idea behind a Metropolis-Hastings sampler is to ‘stand’ at \( \theta[k] \), propose a new parameter value \( \theta’ \) from a proposal \( q(\theta’|\theta[k]) \) (e.g., a random walk). Then, accept the proposed value, i.e., set \( \theta[k+1] = \theta’ \), with probability

\[
\alpha = \min \left( 1, \frac{\tilde{p}_\theta(y_{1:T}) p(\theta’)}{\tilde{p}_\theta(y_{1:T}) p(\theta[k])} \right),
\]

otherwise ‘stay’ and set \( \theta[k+1] = \theta[k] \). To do this, a particle filter has to be run to evaluate (4) for each new proposed value \( \theta’ \). After repeating this procedure for sufficiently many iterations, a series of samples \( \{\theta[k]\}_{k=1}^K \) is obtained, consisting of (correlated) samples of the sought distribution \( p(\theta|y_{1:T}) \). The PMH algorithm is outlined as Algorithm 2.

There are in general two typical pitfalls with Metropolis-Hastings: It is hard to a priori tell how long the initial transient, the burn-in period, will be. A typical behavior is also that the chain gets ‘stuck’ for long periods of time, i.e., all proposals are rejected. One can intuitively understand SMC\(^2\), which we will detail in the next section, as addressing the first issue by ‘warm-starting’ the chain by sequentially adding more and more data.

Algorithm 1

Input: \( K \) (# steps), \( q \) (proposal), \( \theta[0] \) initial parameter
Output: \( \theta[1], \ldots, \theta[K] \) (samples from the posterior, including burn-in)

1: for \( k = 1 \) to \( K \) do
2: Draw \( \theta’ \sim q(\theta’|\theta[k-1]) \).
3: Estimate \( \tilde{p}_{\theta’}(y_{1:T}) \) by a particle filter, Algorithm 1.
4: Compute \( \alpha \) (5).
5: Draw \( d \sim \mathcal{U}(0, 1) \).
6: if \( d < \alpha \) then
7: \( \setminus \theta[k] = \theta’ \) (accept \( \theta’ \))
8: else
9: \( \setminus \theta[k] = \theta[k-1] \) (reject \( \theta’ \))
10: end if
11: end for

Algorithm 2

2.2 SMC\(^2\)

SMC\(^2\) was proposed independently by Chopin et al. (2013) and Fulop and Li (2013). In SMC\(^2\), an SMC sampler (Del Moral et al., 2006) is used instead of Metropolis-Hastings. The SMC sampler is inspired by the particle filter, but can be applied to sample from a general sequence of distributions, not necessarily originating from state space models. The SMC sampler is applied to the sequence \( \{p(\theta), p(\theta|y_1), p(\theta|y_{1:2}), \ldots, p(\theta|y_{1:T})\} \), a data-tempered sequence, evolving from the prior to the posterior.

The SMC sampler works in a similar fashion to the particle filter, with an iteration of weighting – resampling – propagation. The particles, \( \{\theta_{1:m}^{(i)}\}_{m=1}^{N_x} \), ‘live’ in \( \theta \)-space. However, the weighting for particle \( \theta_{1:m}^{(i)} \) should be done with respect to likelihood, so a standard particle filter has to be ‘attached’ to every single \( \theta_{1:m}^{(i)} \) particle to estimate its likelihood, hence the name SMC\(^2\). Thus, the weighting is with respect to

\[
\tilde{p}_{\theta_{1:m}^{(i)}}(y_{1:t-1}) = \frac{1}{N_i} \sum_{t=1}^{N_x} w_t^{(i)}.
\]

The resampling in the SMC sampler is identical to the particle filter, but the propagation has to be performed differently, as there is no equivalent to the function \( f \) for propagating the particles from \( p(\theta|y_{1:T}) \) to \( p(\theta|y_{1:T+1}) \). Instead PMH, Algorithm 2, is used to propagate the particles. To reduce the computational load, it is proposed by Chopin et al. (2013) to apply PMH only when some particle degeneracy criterion (e.g., the effective sample size, ESS) is fulfilled. This is summarized in Algorithm 3.

By construction, SMC\(^2\) is an ‘online algorithm’ in the sense that it evolves along the time index, and if stopped prematurely at \( t^* \), samples from the posterior \( p(\theta|y_{1:t^*}) \) are obtained. Similarly, if another data point \( y_{T+1} \) is added, the algorithm does not have to start over from scratch, as opposed to the PMH. However, the computational load of SMC\(^2\) is increasing with \( t \), prohibiting use in online implementations with real-time requirements.
3. NUMERICAL COMPARISON

In this section, we will first apply PMH and SMC$^2$ to a small simulated example, and then to the problem of learning parameters in a water tank model from real data. We will also discuss their different properties from a user perspective. The Matlab code for all examples can be found on the first authors homepage\(^1\).

3.1 A simulated example

We will start the comparison by a simulated numerical example. Consider the one-dimensional state space model
\[
x_{t+1} = |x_t|^\beta + u_t + w_t, \quad w_t \sim \mathcal{N}(0,1) \quad (7a)
\]
\[
y_t = x_t + e_t, \quad e_t \sim \mathcal{N}(0,1), \quad (7b)
\]
where \(u_t\) is a known input signal, drawn from a Gaussian. We want to learn the parameter \(\beta\), which we believe is drawn from \(\mathcal{N}(0,1)\). This is thus our prior \(p(\beta)\).

We simulate \(T = 200\) data points \(y_{1:T}\) from the model with \(\beta = 0.4\). As the data record is relatively short in this example, we expect the posterior distribution to contain a non-negligible amount of uncertainty.

We apply the PMH algorithm to the problem, with proposal \(q(\theta'|\theta)\) taken as the random walk \(\theta' \sim \mathcal{N}(\theta', 0.01)\). The trace plot for 2000 iterations is shown in Figure 1a. Note the burn-in period; the Markov chain starts at 1.5, but moves eventually to the relevant part of the parameter space. In more complicated problems, the burn-in period is typically much longer. Also note the sequences of some hundred consecutive samples, where all proposals are rejected, a typical Metropolis-Hastings behavior.

Also SMC$^2$ was applied to the problem. A resulting trace plot is shown as Figure 1b. To evaluate the methods, we explore three different settings (in PMH the number of iterations \(K\) and particles \(N_x\) in each particle filter, for SMC$^2$ the number of \(\theta\)-particles \(N_\theta\) and PMH step \(K\)) and plot the samples as histograms in Figure 1c. The reported time are for a standard desktop computer\(^2\).

The methods only show similar results in the last row of Figure 1c, and the poor results in the first and second row are due to shortcomings of the methods when not run with sufficiently many particles/iterations. In this example, SMC$^2$ seems to have a slight advantage as it runs faster.

Fig. 1. Trace plots for PMH (a) and SMC$^2$ (b), and final samples (c) for Example 3.1, for different settings of the methods.

\(^1\) http://www.it.uu.se/katalog/andsv164

\(^2\) Intel i7-4600 2.1 GHz CPU
3.2 A real world example

We also evaluate the methods in a real-world scenario. We use the first \( T = 40 \) data samples from the two water tank data presented by Wigren and Schoukens (2013), to learn a discretized version of the model with 6 unknown parameters

\[
\begin{align*}
(y_t^{(1)}, y_t^{(2)}) &= \left( -k_1 \sqrt{x_t^{(1)}}, k_2 \sqrt{x_t^{(1)} - k_3}, k_4 w_t, 0 \right) + w_t, \quad (8a)
\end{align*}
\]

\( y_t = x_t + e_t, \quad w_t \sim \mathcal{N}(0, k_5 I_2), \quad e_t \sim \mathcal{N}(0, k_6 I_2). \quad (8b) \)

\( (I_2) \) is the 2-dimensional unit matrix, and \( k_5 \) and \( k_6 \) are scalar.) From the physical interpretation of the parameters (Wigren and Schoukens, 2013), our prior assumptions on \( k_1, k_2 \) and \( k_3 \) are a uniform distribution on \([0,1]\), and \( k_4 \) is \( \mathcal{N}(0,1) \). As the noise parameters \( k_5 \) and \( k_6 \) are strictly positive, we formulate priors on their logarithms as \( \log k_5 \sim \mathcal{N}(0,0.1) \) and \( \log k_6 \sim \mathcal{N}(-1,0.1) \) respectively.

We run the PMH sampler for \( K = 50000 \) iterations with \( N = 40 \) particles, and SMC\(^2\) with \( N_\theta = 1000 \) and \( K = 30 \) (requiring 12 and 27 minutes on a standard desktop computer, respectively). The samples obtained by the algorithms are plotted in Figure 2a. The results are quite similar, indicating similar ranges of possible values, but one can guess (by comparing the results for, e.g., \( k_3 \)) that SMC\(^2\) would benefit from an even bigger number \( N_\theta \).

In this example, PMH seems to be the preferred option, as it requires less computational time, still not suffering from the particle degeneracy problem as SMC\(^2\) with, e.g., \( k_3 \).

3.3 Computational load and tuning

The computational load of PMH is governed by \( O(KT N_x) \), i.e., proportional to the number of iterations \( K \), the number of data points \( T \) and the number of particles \( N_x \). The PMH is an anytime algorithm, in the sense that \( K \) does not have to be determined a priori, but can be run as long as the time permits. It has been shown that the choice of \( N_x \) in PMH affects the acceptance rate, and that \( N_x \) should be chosen \( \propto T \) to keep an acceptance rate not varying with \( T \) (Chopin et al., 2013). Following this rule of thumb, the computational load of PMH becomes \( O(KT^2) \).

SMC\(^2\) has the computational load \( O(KT^2 N_x N_\theta) \). The rule of thumb for \( N_\theta \) for PMH also applies to the PMH within SMC\(^2\), giving the load \( O(KT^3 N_\theta) \). Although SMC\(^2\) provides the user with \( p(\theta|y_{1:t}) \) for all \( t \) from 0 to \( T \), it is not suited for online problems with real-time requirements, as it grows forbiddingly fast with increasing \( T \).

The time for implementation does usually not count as a part of the computational load, but is nevertheless of big relevance in practice. Our experience from implementing the two examples is that SMC\(^2\) is significantly harder to implement, due to the quite involved interaction between the two nested levels of SMC. PMH, on the other hand, only requires a particle filter algorithm, computation of the acceptance probability \( \alpha \) and some storage. The implementation of PMH therefore also has the advantage of being easier to debug than SMC\(^2\).

(b) The input-output data in the watertank data. The first 40 samples (200 s) are used as the training data, for inferring the posterior distribution of the parameters. The model learned with PMH (including the uncertainties) was here used to simulate the behavior of the tanks for another 800 s. (SMC\(^2\) yields similar results.) The more intensive gray color, the more likely simulated output.

As an alternative for the user to implement the methods on her/his own, there are currently a few software packages designed for off-the-shelf use of PMH, SMC\(^2\) and related methods available, such as LibBi\(^3\) (Murray, 2013) and Biips\(^4\) (Todeschini et al., 2014).

The tuning of both algorithms is indeed crucial for their performance. A perhaps subtle, but important, design choice is the proposal \( q \) inside the PMH. Suggestions on how to improve the proposal have been proposed in the literature, e.g Dahlin et al. (2015) for PMH, and Fearnhead and Taylor (2013) for SMC samplers. The number of particles, \( N_x \) and \( N_\theta \), are of course also of great importance, and a recent work on automatic adaption of \( N_\theta \) in SMC\(^2\) is presented by Chopin et al. (2015). Further, the rule of thumb \( N_x \propto T \) applies, and a recent work with more extensive guidelines is Doucet et al. (2015).

\( ^3 \) http://libbi.org
\( ^4 \) http://alea.bordeaux.inria.fr/biips/
4. CONCLUSIONS

PMH has a clear computational advantage over SMC\(^2\) for large \(T\). For small \(T\), as illustrated in the numerical examples, the methods are more comparable. In the examples, there are no clear advantage neither for PMH nor SMC\(^2\); they both perform similarly, with a computational time of the same magnitude. SMC\(^2\) does indeed offer more flexibility in the tuning, and might thus have a greater potential in adaption to certain problems. If all sequential posteriors \(p(\theta|y_1), \ldots, p(\theta|y_1:\hat{T})\) are of interest, SMC\(^2\) is indeed preferable. However, as discussed, SMC\(^2\) is considerably more challenging to implement and debug than PMH.

REFERENCES


