Nonlinear System Identification
Learning while respecting physical models
using a sequential Monte Carlo method

Anna Wigren, Johan Wågberg, Fredrik Lindsten, Adrian G. Wills, Thomas B. Schön

Please cite this version:
Anna Wigren, Johan Wågberg, Fredrik Lindsten, Adrian G. Wills, Thomas B. Schön. “Nonlinear System Identification: Learning While Respecting Physical Models Using a Sequential Monte Carlo Method.” In: IEEE Control Systems Magazine 42.1 (2022). © 2022 IEEE, pp. 75–102

@article{Wigren2022,
  author={Wigren, Anna and Wågberg, Johan and Lindsten, Fredrik and Wills, Adrian G. and Schön, Thomas B.},
  journal={IEEE Control Systems Magazine},
  title={Nonlinear System Identification: Learning While Respecting Physical Models Using a Sequential Monte Carlo Method},
  year={2022},
  volume={42},
  number={1},
  pages={75-102},
  doi={https://doi.org/10.1109/MCS.2021.3122269},
}

A note on the structure of the article:
The published version of this article consists of a main text that provides the essential content and multiple sidebars with additional information, either in the form of examples or a background with further technical details. The same structure has been adopted in this version of the article. Sidebars are indicated by grey boxes and are referenced from the main text using double quotation marks, i.e. “Background: Markov chain Monte Carlo” refers to the sidebar on Markov chain Monte Carlo. The sidebars are placed at the end of the section where they are first referenced.
Nonlinear System Identification
Learning while respecting physical models using a sequential Monte Carlo method

Anna Wigren∗1, Johan Wågberg†2, Fredrik Lindsten‡3, Adrian Wills§4, and Thomas B. Schön¶5

1,2,5 Department of Information Technology, Uppsala University
3 Department of Computer and Information Science, Linköping University
4 School of Engineering, University of Newcastle

Abstract

Identification of nonlinear systems is a challenging problem. Physical knowledge of the system can be used in the identification process to significantly improve the predictive performance by restricting the space of possible mappings from the input to the output. Typically, the physical models contain unknown parameters that must be learned from data. Classical methods often restrict the possible models or have to resort to approximations of the model that introduce biases. Sequential Monte Carlo methods enable learning without introducing any bias for a more general class of models. In addition, they can also be used to approximate a posterior distribution of the model parameters in a Bayesian setting. This article provides a general introduction to sequential Monte Carlo and shows how it naturally fits in system identification by giving examples of specific algorithms. The methods are illustrated on two systems: a system with two cascaded water tanks with possible overflow in both tanks and a compartmental model for the spreading of a disease.

1 Introduction

The modern world contains an immense number of different and interacting systems, from the evolution of weather systems to variations in the stock market, autonomous vehicles interacting with their environment and the spread of diseases. For society to function, it is essential to understand the behavior of the world, so that informed decisions can be made that are based on likely future outcomes. For instance, consider the spread of a new disease like the coronavirus. It is of great importance to be able to predict the number of people that will be infected at different points in time to ensure that appropriate healthcare facilities are available. It is

∗anna.wigren@it.uu.se
†johan.wagberg@it.uu.se
‡fredrik.lindsten@liu.se
§adrian.wills@newcastle.edu.au
¶thomas.schon@it.uu.se
also of interest to be able to make decisions based on accurate information to best attenuate the spread of disease. Moreover, understanding specific attributes of the disease, such as the incubation time, the number of unreported cases, and how certain we are about this knowledge are also crucial.

These types of applications are examples of so-called dynamic systems, which are the focus of this article. Dynamic systems have the property that the future system response depends on the past system response [1]. Capturing these types of dynamic phenomena can be achieved using mathematical models, which offer a concrete mechanism for making predictions and supporting decisions. The extreme flexibility and versatility of mathematics affords modeling of highly disparate dynamic behavior. However, it also creates a challenge, since it is not always obvious how to choose an appropriate model. This diversity is perhaps best illustrated by contrasting examples.

Consider the modeling of rigid-body vehicle dynamics, such as the motion of a car or a plane. In this case, it is possible to exploit prior knowledge of the system and adopt a classical mechanics approach to derive Newton-Euler equations of motion for each application [2]. The mathematical model structure is largely determined by knowledge of the physical system, and the model will depend on certain parameters such as mass and inertia terms, and damping and friction coefficients. In many cases, these parameter values can be difficult to obtain based on first principles approaches alone. It is important to also note that some parameters may have feasible ranges, such as mass terms being nonnegative, which is also a form of prior knowledge.

Contrasting this type of model, it is also possible to employ highly flexible and general model structures to describe dynamic systems, such as deep neural networks (DNNs) or Gaussian processes (GPs) [3, 4, 5]. The flexibility of the DNN model class stems from the general construction of the model, which involves potentially many layers of interacting nonlinear functions. Importantly, these interactions are allowed to adapt for each new application, since they rely on coefficients/parameters that are free to change values. In the case of GP, the model structure is also highly flexible, nonparametric and adapted based on available data. For either of these flexible model classes, it is more challenging to impose prior system knowledge. However, some progress is being made along these lines [6, 7].

Irrespective of the type of model, there are unknown quantities that must be determined, which are often inferred from observations from the system that is being modeled. There are many different approaches for extracting or estimating these unknown values from observed system data [8, 9, 10, 11, 12]. Among the many possibilities, this article concentrates on two commonly used and complimentary approaches. In particular, the presented inference methods are grouped according to two main attributes: 1) the assumptions made about how to model unknown parameters, and, 2) what should be estimated in addition to the parameters.

More specifically, if the model parameters are assumed to be deterministic variables, then this results in a frequentist inference perspective, where the so-called maximum likelihood (ML) approach has proven to be highly successful in providing accurate point estimates of the parameters [9, 10]. Alternately, if uncertainty about the parameter values is incorporated by treating them as random variables, then this results in the so-called Bayesian perspective, where the posterior distribution of the parameters is the object of interest [11]. An attractive attribute of the Bayesian approach is that it provides quantification of uncertainty, which is essential when making decisions based on the associated models. Otherwise, decisions may be executed based on misplaced confidence. It is also worth mentioning that there is a connection between these two approaches by considering so-called maximum a posteriori methods [13].
Regardless of adopting the frequentist or Bayesian perspective, it is rare that the estimates can be provided analytically. This article provides computational tools for calculating these estimates in the remaining cases where analytical solutions are not available. Towards computing them, it is essential to both the frequentist and Bayesian approaches that certain integrals can be evaluated. While the details will be explained in subsequent sections, it suffices for now to mention that computing these integrals is generally intractable [14].

An overarching theme of this article is to approximate intractable integrals by employing carefully tailored Monte-Carlo integration techniques that result in tractable weighted sums. More precisely, the sequential nature of the dynamic models lends itself to the so-called sequential Monte Carlo (SMC) methods [15, 16], which will be explored in much more detail as the article progresses. Furthermore, these SMC methods are employed both within frequentist and Bayesian approaches, resulting in algorithms that are applicable to a wide range of modeling problems. An attractive property of the SMC methods is that they also offer asymptotic convergence guarantees, which are not offered by other approximation methods in general [14].

These SMC techniques are also highly suitable to the situation where prior knowledge of the system is available, such as knowledge of the physical system, model structure, and possibly feasible ranges for unknown parameter values. The main benefit of using SMC techniques is that they are applicable to general nonlinear systems, without modification of the prior knowledge or assumptions. This allows the separation of modeling from the inference method, which provides the modeler more freedom in adding domain-specific prior knowledge. In contrast, many alternative approaches require—either explicitly or implicitly—that the problem satisfies certain restrictive assumptions, such as Gaussian noise corruption. The aim of the article is to present computational tools for estimating general nonlinear dynamic systems, while adhering to prior system knowledge without modification.

The article first presents the type of models considered and provides two examples that illustrate how physical insight about the system can be transformed to a mathematical model suited for statistical inference. These two examples are used throughout to illustrate the various methods. Following this, the identification problem is introduced, and the key expressions needed for learning are highlighted. This leads to an introduction of SMC specifically targeted for offline system identification. The remainder of the article presents identification algorithms where SMC plays an integral part. Both optimization-based learning methods and probabilistic methods where posterior distributions are computed are considered and applied to the example models. The article also gives a short introduction to probabilistic programming, a tool that can significantly reduce the complexity of trying out different models and inference methods.

2 Modeling

Mathematical modeling is applicable to a wide range of problems spanning many areas of science and engineering. As such, it is important to restrict attention to the particular model class of interest to this article, namely, discrete-time state-space models for dynamic systems. These types of models have a long and fruitful history in the fields of physics and engineering, originating in the phase-space ideas from physics [17]. The essential idea is that the dynamic behavior of the model is determined by the current state of the model, which is a vector belonging to a so-called state space. It is important to mention that the states should be
associated with the model, rather than the real-world phenomena. The latter has no particular concern for states or any other modeling choices, including the model structure and associated parameters.

It is essential to connect observations from the real-world phenomena to the state-space model, since this is the primary purpose of modeling, and so that the model can be adapted to best match observations. These ideas are made more concrete in the subsequent section, which introduces the state-space model of interest in this article, and presents two concrete examples to illustrate this modeling approach.

2.1 Probabilistic formulation of the state-space model

To make the modeling ideas discussed above concrete, it is necessary to introduce some notation. To that end, the model state is denoted \( x_t \), where the subscript \( t \) indicates the current discrete time instant. Observations from the system are denoted \( y_t \), and inputs to the system are denoted \( u_t \). It is typical to express the connection between model and observations via the state-space equations

\[
\begin{align*}
    x_t &= f_t(x_{t-1}, u_t, w_t, \theta), \\
    y_t &= g_t(x_t, u_t, e_t, \theta).
\end{align*}
\]

In the above, the function \( f_t \) explains how the state evolves over time, and \( g_t \) relates the model state to the system observations. The parameter vector \( \theta \) allows the functions to depend on some possibly unknown parameters, and \( w_t \) and \( e_t \) are noise terms to account for uncertainty.

As an example, the functional form of a linear-Gaussian state-space model is

\[
\begin{align*}
    x_t &= Ax_{t-1} + Bu_t + w_t, \quad w_t \sim \mathcal{N}(0, Q), \\
    y_t &= Cx_t + Du_t + e_t, \quad e_t \sim \mathcal{N}(0, R),
\end{align*}
\]

where \( A \) is a transition matrix, \( B \) is an input matrix, \( C \) is an observation matrix, \( D \) is a feedforward matrix, and \( w_t \) and \( e_t \) are independent and identically distributed Gaussian noise with zero mean and covariance matrices \( Q \) and \( R \), respectively. The unknown parameters of this model are the transition matrix \( A \), the input matrix \( B \), the observation matrix \( C \), the feedforward matrix \( D \), and the covariance matrices \( Q \) and \( R \). The notation \( \mathcal{N}(z | \mu, \Sigma) \) is used to denote a multivariate Gaussian distribution with mean \( \mu \) and covariance matrix \( \Sigma \) for the variable \( z \).

This article uses a more general, probabilistic, form of the state-space model, where the essential idea remains the same: The state holds the information required to determine the state evolution. The main difference is the manner in which this is expressed. For probabilistic state-space models, the time evolution and measurement relationships are captured via the conditional probability distributions

\[
\begin{align*}
    x_t &\sim p(x_t | x_{t-1}, u_t, \theta), \\
    y_t &\sim p(y_t | x_t, u_t, \theta),
\end{align*}
\]

with transition density \( p(x_t | x_{t-1}, u_t, \theta) \) and observation density \( p(y_t | x_t, u_t, \theta) \), parameterized by an unknown parameter \( \theta \). A probabilistic state-space model can, equivalently, be represented graphically as a probabilistic graphical model. Figure 1 illustrates the graphical
The probabilistic modeling approach can be generalized to a rich class of systems that extends well beyond state-space models. A brief discussion of this general approach is available in “Background: Probabilistic models in general”.

Figure 1: A probabilistic graphical model representation of the state-space model. In a probabilistic graphical model, each random variable is denoted by a circle. If the variable is observed, the circle is shaded in gray. Each arrow indicates a dependency. For example, the state at time 2 depends on the state at time 1, and all states \( x_1:T \) and measurements \( y_1:T \) depend on the parameters \( \theta \).

Continuing the above linear-Gaussian example, the equivalent probabilistic form of the model (2) is

\[
\begin{align*}
x_t &\sim p(x_t \mid x_{t-1}, u_t, \theta) = \mathcal{N}(x_t \mid Ax_{t-1} + Bu_t, Q), \\
y_t &\sim p(y_t \mid x_t, u_t, \theta) = \mathcal{N}(y_t \mid Cx_t + Du_t, R).
\end{align*}
\]

(4) In addition to this model, the initial state \( x_1 \sim \mathcal{N}(x_1 \mid \mu_1, \Sigma_1) \), where \( \mu_1 \) is the mean and \( \Sigma_1 \) is a covariance matrix, must also be specified. Two examples of the functional as well as the probabilistic form for nonlinear state-space models are available in “Example: Cascaded water tanks” and “Example: Dengue fever”. In both cases, the model structure is adapted to the application at hand by incorporating “physical insight” about the application in the governing equations. This is useful, not only for obtaining a model that respects the physical properties of the system, but also for enabling more efficient learning of model parameters, compared to using generic black-box models.

When an appropriate model structure has been determined, which may be an iterative process, then the focus can be placed on learning (or estimating or identifying) the unknown parameters \( \theta \) based on observed data. This is the problem treated in the remainder of the article.

### Background: Probabilistic models in general

A probabilistic model describing the relation between two random variables \( A \in \mathcal{A} \) and \( B \in \mathcal{B} \) can be specified through the joint distribution of these variables \( p_{A,B}(a,b) \). The subscripts indicate which random variables that the distribution describes. Apart from in this section, the subscripts are left out in the interest of space and notational simplicity. The variable \( A \) can, for instance, be observations from a system of interest, and \( B \) can be parameters and/or state variables in the probabilistic model.

When working with probabilistic models, there are two basic operations that are
used extensively to form other distributions from the joint distribution. The first is the *marginalized* distribution of one of the random variables, which is obtained from the joint distribution by simply integrating out all other variables. Hence, for the variable $B$, the marginalized distribution is

$$p_B(b) = \int_A p_{A,B}(a,b)da.$$  (5)

If the random variable is discrete instead of continuous, the integral is replaced with a summation. The second basic operation is to form the *conditional* distribution of one of the random variables, which is obtained by factorizing the joint distribution into a product of a conditional and a marginal distribution. To form the conditional distribution of $A$ given $B$, the factorization is

$$p_{A,B}(a,b) = p_{A\mid B}(a\mid b)p_B(b),$$  (6)

where $p_{A\mid B}(a\mid b)$ is the sought-after conditional distribution. The two basic operations above can be combined to form

$$p_{B\mid A}(b\mid a) = \frac{p_{A,B}(a,b)}{p_A(a)} = \frac{p_{A,B}(a,b)}{\int_B p_{A,B}(a,b)db}$$

$$= \frac{p_{A\mid B}(a\mid b)p_B(b)}{\int_B p_{A\mid B}(a\mid b)p_B(b)db},$$  (7)

referred to as *Bayes’ theorem*.

**Example: Cascaded water tanks**

The cascaded water tank system, depicted in Figure 2, consists of two water tanks placed vertically so that water can flow between them through holes in the bottom of the tanks. A pump transports water from the reservoir below the tanks to the upper tank. From the upper tank, the water can flow into the lower tank through a hole in the bottom of the upper tank. Analogously, water can flow back into the reservoir through a hole in the bottom of the lower tank. The inflow of water to the upper tank can be controlled by adjusting the voltage supplied to the pump, and the water level in the lower tank is measured. The intended task is to control or predict the water level in the lower tank based on the voltage level supplied to the water pump. The water flow in the system can, under normal operating conditions, be considered a weakly nonlinear process. However, overflow can occur in both the upper and lower tank if the input signal is large enough, which corresponds to a hard nonlinearity.

**A state-space model of the cascaded water tanks based on physics**

The physical processes governing the behavior of fluid-flow systems like the water tanks are known and can be used to construct a discrete-time state-space model of the system.
Figure 2: The cascaded water tank system. The system consists of two water tanks placed vertically, so that water can flow between them. A pump transports water from the reservoir to the upper tank. From there, water flows into the lower tank through a hole in the bottom of the upper tank and back into the reservoir through a hole in the bottom of the lower tank. The inflow of water to the upper tank is controlled by adjusting the voltage $u$ supplied to the pump. The water level in the upper tank is $\hat{x}^u$, and the water level in the lower tank is $\hat{x}^l$. The observation $y$ is the measured water level in the lower tank. The water flow in the system is, under normal operating conditions, a weakly nonlinear process. However, both tanks can overflow if the input signal is large enough, which corresponds to a hard nonlinearity.

One such model on functional form, taken from [18], is

$$
\begin{align*}
    x^u_t &= \hat{x}^u_{t-1} - k_1 \sqrt{\hat{x}^u_{t-1}} + k_5 u_t - k_3 \hat{x}^u_{t-1} + w^u_t, \\
    x^l_t &= \hat{x}^l_{t-1} + k_1 \sqrt{\hat{x}^u_{t-1}} - k_3 \sqrt{\hat{x}^l_{t-1}} \sqrt{\hat{x}^u_{t-1}} + k_2 \hat{x}^u_{t-1} - k_4 \hat{x}^l_{t-1} \\
        &\quad + k_6 \max(0, x^u_{t-1} - 10) + w^l_t, \\
    y_t &= \hat{x}^l_t + e_t,
\end{align*}
$$

(8a) (8b) (8c)

where the input $u_t$ is the voltage provided to the pump, and $\hat{x}^u_t$ and $\hat{x}^l_t$ are the water levels in the upper and lower tank, respectively (capped at $10$ — the height of the tanks). The states of the model, $x^u_t$ and $x^l_t$, are the water levels in the upper and lower tank, respectively, plus the inflow to each tank. The parameters $k_1, k_2, k_3, k_4, k_5, k_6$ encode physical properties of the system that are not known, for instance, the diameter of the tanks and holes, as well as different flow constants. The weakly nonlinear behavior of the tank system can be described using Bernoulli’s principle and conservation of mass [19]. The terms in (8a) and (8b) related to $k_1, k_3$, and $k_5$ capture this effect. Bernoulli’s principle is valid when losses in the system due to friction and the geometry of the tanks are low.
These losses are not negligible for the water tanks, so the linear terms $k_2 \ddot{x}_u^t$ and $k_4 \ddot{x}_l^t$ are added to the model to account for this effect \cite{20}. The hard nonlinearity introduced by overflow in the upper tank is accounted for through the term $k_6 \max(0, x_{u}^{t-1} - 10)$, which represents an extra inflow to the lower tank when the water level in the upper tank is above 10. The noise terms $w_u^t, w_l^t, e_t$ are assumed to be independent zero-mean, white Gaussian noise with unknown variances $\sigma_w^2$ and $\sigma_e^2$. The input voltage to the pump and the water level in the lower tank are measured. The initial water level in the upper tank is unknown and must be estimated along with the parameters of the model, which are collectively referred to as $\theta = \{k_1, k_2, k_3, k_4, k_5, k_6, \sigma_w^2, \sigma_e^2\}$.

**A probabilistic state-space model of the cascaded water tanks**

All randomness in (8) is due to the noise terms $w_u^t, w_l^t, e_t$, which are independent, additive, Gaussian random variables with mean value zero. The probabilistic form of (8) is therefore

\[
p(x_t | x_{t-1}, \theta) = N\left(\left[\mu_u(x_{u}^{t-1}, u_t), \mu_l(x_{l}^{t-1}, x_{u}^{t-1})\right], \sigma_w I\right),
\]

\[
p(y_t | x, \theta) = N\left(\tilde{x}_l^t, \sigma_e^2\right),
\]

where $I$ is the identity matrix and the mean values are

\[
\mu_u(x_{u}^{t-1}, u_t) = \ddot{x}_{u}^{t-1} - k_1 \sqrt{x_{u}^{t-1}} + k_5 u_t - k_2 \ddot{x}_{u}^{t-1},
\]

\[
\mu_l(x_{l}^{t-1}, x_{u}^{t-1}) = \ddot{x}_{l}^{t-1} + k_1 \sqrt{x_{u}^{t-1}} - k_3 \sqrt{x_{l}^{t-1}} + k_2 \ddot{x}_{u}^{t-1} - k_4 \ddot{x}_{l}^{t-1} + k_6 \max(0, x_{u}^{t-1} - 10).
\]

The covariance matrix is diagonal, since the noise sources are assumed to be independent of each other.

**The dataset**

For the cascaded water tank system in Figure 2 there exists a benchmark dataset developed at Uppsala University \cite{19} that consists of two data series: one for training and one for testing. Both series have multisine input signals of length 1024 with a frequency range of 0 to 0.0144 Hz. The water level in the lower tank was measured using a capacitive water-level sensor. Zeroth-order hold was used with a sampling period of 4 s.

---

**Example: Dengue fever**

Dengue fever is a tropical virus disease spread by mosquitoes that causes around 10,000 deaths every year worldwide \cite{21}. A human can become infected if bitten by an infected mosquito and, analogously, a mosquito can become infected if it bites an infected human. There is no direct transmission of the virus between individuals of the same species.
After the initial bite, the virus goes into incubation (reproduction inside the host body without onset of any symptoms) for 8-12 days if the host is a mosquito or 4-10 days if the host is a human [22]. When the incubation period is over, the infected individual becomes infectious and can infect other individuals. The infectious stage lasts 4-5 days on average, but it can be as long as 12 days for humans. Mosquitoes are infectious for the rest of their life [22]. Humans either die from the virus, or recover and become immune after 2-7 days of symptoms [22].

A state-space model for the spread of dengue fever based on prior knowledge

The dynamics of epidemiological diseases, like dengue fever or the coronavirus, can be described using compartmental models. In a compartmental model, the population is split into several disjoint compartments based on some disease-related characteristics (for example if an individual is infectious or have recovered from the disease). A simple example of a compartmental model is the susceptible–infectious–recovered (SIR) model visualized in Figure 3. It has three compartments: susceptible, which contains the individuals who are currently healthy but can become infected; infectious, which contains the currently infectious individuals; and recovered, which contains the individuals who have been sick but who are now either immune or dead. In each instance, some proportion of the susceptible moves to the infectious compartment, and some proportion of the infectious move to the recovered compartment.

Figure 3: The susceptible–infectious–recovered model is a compartmental model where the population is divided into three disjoint compartments: susceptible (healthy but can become infected), infectious (infectious and can infect others), and recovered (previously sick and is now either immune or dead). At each point in time, a proportion of the individuals in each compartment can transition from susceptible to infectious or from infectious to recovered. The transition is indicated by the arrows between the compartments.

To construct a model based on medical prior knowledge about the spread of dengue fever, two modifications to the SIR model are necessary. Firstly, dengue fever has an incubation time, implying that individuals do not move directly from susceptible to infectious. To take this into account, the compartment “exposed” is introduced, which contains individuals who have been infected but are not yet infectious. Such a compartmental model is referred to as a susceptible–exposed–infectious–recovered (SEIR) model. Secondly, both humans and mosquitoes go through the stages of being susceptible $S(t)$, exposed $E(t)$, infectious $I(t)$, and recovered $R(t)$. However, they transition between compartments with different rates depending on, for instance, different incubation times. Additionally, mosquitoes never enter the recovered compartment. To
incorporate the different dynamics for humans and mosquitoes, two coupled SEIR models are used—one for humans and one for mosquitoes. The final, coupled SEIR model is depicted in Figure 4.

Figure 4: A coupled susceptible–exposed–infectious–recovered (SEIR) model for an outbreak of dengue fever. To model the spread of the disease, two coupled SEIR models are used to capture that infectious mosquitoes can infect humans and infectious humans can infect mosquitoes. The coupling is illustrated with the red, dotted arrows between the two SEIR models. The SEIR model is a compartmental model with four compartments: susceptible, exposed, infectious, and recovered. The black, solid arrows indicate transitions between compartments. The difference, compared to the SIR model in Figure 3, is that the exposed compartment has been added to account for the incubation period of the dengue virus. The model for the mosquitoes lack the recovered compartment, since mosquitoes die before they recover from the disease.

A continuous-time state-space model corresponding to Figure 4 can be based on the system of coupled, ordinary differential equations (ODEs)

\[
\begin{align*}
\frac{dS^h}{dt}(t) &= -\lambda^h S^h(t), \\
\frac{dE^h}{dt}(t) &= \lambda^h S^h(t) - \delta^h E^h(t), \\
\frac{dI^h}{dt}(t) &= \delta^h E^h(t) - \gamma^h I^h(t), \\
\frac{dR^h}{dt}(t) &= \gamma^h I^h(t), \\
\frac{dS^m}{dt}(t) &= \nu^m - (\lambda^m + \mu^m)S^m(t), \\
\frac{dE^m}{dt}(t) &= \lambda^m S^m(t) - (\delta^m + \mu^m)E^m(t), \\
\frac{dI^m}{dt}(t) &= \delta^m E^m(t) - \mu^m I^m(t),
\end{align*}
\]

as described in [23]. The state is \( x = [S^h, E^h, I^h, R^h, S^m, E^m, I^m, R^m]^+ \), and the parameters are the force of infection \( \lambda \), the incubation rate \( \delta \), the recovery rate \( \gamma \), the birth rate \( \nu \), and the death rate \( \mu \). The superscripts \( h \) and \( m \) denote human and mosquito, respectively. The duration of the outbreak is assumed to be short enough to make human births and deaths negligible. The coupling between the two SEIR models stems from the force of infection parameters, which are modeled as \( \lambda^h = c_h I^m \) and \( \lambda^m = c_m I^h \), where the exact form of the constants \( c_h \) and \( c_m \) are available in [23]. The observations are Poisson-distributed with a rate that depends on the number of newly infectious humans and the reporting rate \( \rho \) to health centers.

A probabilistic state-space model of the spread of dengue fever

The continuous-time state-space model in [11] describes the dynamics using ODE. For compartmental models, this corresponds to a mean-field approximation of an underlying
stochastic process. Such an approximation is valid for large populations. However, it can be inaccurate for small populations, especially if the number of infectious drops below one \[24\]. A probabilistic, discrete-time, and discrete-state version of \[11\] was derived in \[25\]. The state transitions between compartments are now stochastic and are performed in two steps—a transition that keeps the population fixed, followed by a transition that accounts for births and deaths. Only the former transition is discussed here. The transitions between compartments (excluding births and deaths) are

\[
S_t^h = S_{t-1}^h - e_t^h, \quad S_t^m = S_{t-1}^m - e_t^m, \\
E_t^h = E_{t-1}^h + e_t^h - i_t^h, \quad E_t^m = E_{t-1}^m + e_t^m - i_t^m, \\
I_t^h = I_{t-1}^h + i_t^h - r_t^h, \quad I_t^m = I_{t-1}^m + i_t^m - r_t^m, \\
R_t^h = R_{t-1}^h + r_t^h,
\]

where lowercase letters indicate the stochastic number of newly exposed \(e_t\), infectious \(i_t\), and recovered \(r_t\). These random variables are assumed to be binomially-distributed. The binomial distribution, Bin(\(n, p\)), describes the number of successes in \(n\) independent trials with only two possible outcomes, success or failure, where success occurs with probability \(p\). The number of newly infectious humans in an SEIR model can be modeled as the number of (unfortunate) successful transitions from the exposed compartment to the infectious compartment. The number of exposed individuals that can transition at time \(t\) is \(E_{t-1}^h\), and a transition of one individual occurs with an infectious probability \(\delta^h\), independent of all other exposed individuals. Thus, the number of newly infectious is a binomial variable \(i_t^h \sim \text{Bin}(E_{t-1}^h, \delta^h)\). The transitions between all compartments are

\[
e_t^h \sim \text{Bin}(e_t^h | \tau_t^h, \lambda^h), \quad e_t^m \sim \text{Bin}(e_t^m | \tau_t^m, \lambda^m), \\
i_t^h \sim \text{Bin}(i_t^h | E_{t-1}^h, \delta^h), \quad i_t^m \sim \text{Bin}(i_t^m | E_{t-1}^m, \delta^m), \\
r_t^h \sim \text{Bin}(r_t^h | I_{t-1}^h, \gamma^h), \quad r_t^m \sim \text{Bin}(r_t^m | I_{t-1}^m, \gamma^m),
\]

where the parameters are the transmission probability \(\lambda\), the infection probability \(\delta\), and the recovery probability \(\gamma\). Mosquitoes never recover from the disease. Thus, \(\gamma^m\) is always zero, ensuring that \(r_t^m\) is zero. The number of susceptible humans that were bitten by an infectious mosquito, \(\tau_t^h\), and the number of susceptible mosquitoes that have bitten an infectious human, \(\tau_t^m\), are

\[
\tau_t^h \sim \text{Bin} \left( S_{t-1}^h, 1 - \exp \left( -I_{t-1}^m/n_{t-1}^h \right) \right), \\
\tau_t^m \sim \text{Bin} \left( S_{t-1}^m, 1 - \exp \left( -I_{t-1}^h/n_{t-1}^m \right) \right).
\]

The derivation of these quantities can be found in \[25\]. The observations are the number of newly infectious humans that sought medical care at a health center. Only a proportion of all that have been infected are expected to report to health centers. Thus, the observations are also modeled as binomially-distributed with reporting probability \(\rho\), that is,

\[
y_t \sim \text{Bin}(\tau_t^h, \rho).
\]

The complete parameter vector is \(\theta = \{\lambda^h, \delta^h, \gamma^h, \lambda^m, \delta^m, \gamma^m, \rho\}\) for this model.
3 Identification

Identification of the state-space model (3) is the process of learning all the unknown parameters so that the model best describes some measured input-output data. Let $D_T = \{y_t, u_t\}_{t=1}^T$ denote the collection of measured outputs and possible inputs up to time $T$. More precisely, the focus in this article is on the batch system identification problem, which amounts to finding a description for the unknown parameters $\theta$ of the state-space model (3) based on the available data $D_T$. For notational simplicity—without loss of generality—the known input $u_t$ is from now on dropped from the notation.

To proceed with identification, it is important to acknowledge any assumptions, or prior knowledge, on the unknown parameter values themselves. The two most commonly used assumptions on the unknown parameters $\theta$ are:

1. **Frequentistic (ML):** The parameters are assumed to be deterministic variables. The aim is to find a point estimate of the parameters.

2. **Bayesian:** The parameters are assumed to be random variables, implying that the model must be augmented with a prior distribution $\theta \sim p(\theta)$ for the parameters. The aim is to find the posterior distribution of the parameters.

Both formulations are treated in this article, without making any individual ranking between them. One of these may be better suited for a given problem, or perhaps a combination makes most sense—it depends on the task at hand. Which assumption to use is an important decision, since it influences which identification algorithms that can be applied.

ML is intuitive in the sense that it amounts to finding the point estimate of the unknown parameters that makes the observed data as likely as possible. This is done by selecting the parameter value that maximizes the marginal likelihood $p(y_{1:T} | \theta)$ of the observed data $y_{1:T} = (y_1, y_2, \ldots, y_T)$,

$$\hat{\theta}_{\text{ML}} = \arg \max_\theta p(y_{1:T} | \theta).$$

(16)

ML is one of the most common ways to formulate system identification problems. It has been extensively covered in the literature, see e.g. [26] [27].

In the Bayesian formulation, the aim is instead to find the posterior distribution of the parameters $p(\theta | y_{1:T})$. From an application of Bayes’ theorem (7), the parameter posterior is

$$p(\theta | y_{1:T}) = \frac{p(y_{1:T} | \theta)p(\theta)}{p(y_{1:T})}.$$  

(17)
Bayesian system identification is not as well-developed as its frequentistic counterparts. However, there has recently been more developments along this line of research. See [28] for early work and [29, 30] for newer initiatives.

It is interesting to note that the likelihood $p(y_{1:T} \mid \theta)$ is required for both the ML (16) and the Bayesian (17) formulation. For a state-space model, the likelihood can be computed via the integral

$$p(y_{1:T} \mid \theta) = \int p(y_{1:T}, x_{1:T} \mid \theta) \, dx_{1:T},$$

that is, by marginalizing out the hidden sequence of state variables $x_{1:T}$. This illustrates a central aspect of working with state-space models. In addition to the model parameters and the observed data, the model also contains hidden (unobserved) state variables that must be handled in some way, typically by marginalization. Indeed, the integrand in the expression above is the full probabilistic model for which there exists a tractable expression,

$$p(y_{1:T}, x_{1:T} \mid \theta) = p(x_1 \mid \theta) \prod_{t=1}^{T} p(y_t \mid x_t, \theta) \prod_{t=2}^{T} p(x_t \mid x_{t-1}, \theta).$$

(19)

However, computing the integral is challenging in general.

One interpretation of the likelihood calculation (18) is that it amounts to averaging the full probabilistic model $p(y_{1:T}, x_{1:T} \mid \theta)$ over all possible state trajectories $x_{1:T}$. For nonlinear system identification, the need for approximations —like the ones offered by SMC methods— is tightly linked to the intractability of (18) and the unknown state trajectory $x_{1:T}$. The two main strategies for addressing the unknown state trajectory are:

1. **Marginalization**: In this strategy, the state variables are marginalized (integrated out) according to (18), implying that the parameters are targeted directly. The identification problem is solved by first computing the integral appearing in (18), continuing by viewing $\theta$ as the only unknown quantity of interest. In the frequentistic problem formulation, the prediction error method and direct maximization of the likelihood belong to this strategy [26]. In the Bayesian formulation, the Metropolis–Hasitngs (MH) algorithm [31]-[32] can be used to approximate the posterior distribution of the parameters conditioned on the data.

2. **Data augmentation**: In this strategy, the states are treated as auxiliary variables that are estimated together with the parameters. Intuitively, this strategy can be thought of as a systematic way of separating one hard problem into two new and closely linked subproblems, each of which should be easier to solve than the original problem. The expectation maximization (EM) algorithm [33] solves the ML formulation in this way, and the Gibbs sampler [34] solves the Bayesian problem using this strategy.

The important difference between these strategies is that marginalization targets the parameters only, whereas data augmentation targets both parameters and states. Thus, marginalization, operates on a smaller dimensional space, but must somehow handle the intractable likelihood. On the other hand, data augmentation operates on a much larger space, but the (complete) likelihood (19) is available on closed form.

In this article, the identification methods have been associated with one of the two strategies based on where they fit most naturally. This division is visualized in Figure 5. For instance,
Gibbs sampling alternates between sampling the parameters and sampling the states; hence, it fits well with the data augmentation strategy. MH, on the other hand, can sample the parameters directly and fits more naturally with the marginalization strategy. It is important to note that many identification algorithms can be used for both strategies, even if they are better-suited for one of them. For example, MH could also be applied for the data augmentation strategy by sampling parameters and states jointly. However, it is well-known that MH can be inefficient for high-dimensional problems. Gibbs sampling, for example, typically performs better in this case.

Regardless of whether a marginalization or data augmentation approach is adopted, the marginal likelihood $p(y_{1:T} | \theta)$ plays a central role. While it is generally intractable to evaluate exactly, the next section discusses a class of SMC methods that offer tractable approximations of expectation integrals like (18). SMC forms the basis for the ensuing algorithms that employ either the maximum-likelihood or Bayesian assumption on the parameters, and either the marginalization or data augmentation strategy to address the unknown state trajectory.

### 4 Sequential Monte Carlo

SMC methods \cite{15,35,36} are generic algorithms for performing approximate inference in statistical models. This section first gives a brief description of the general SMC framework and then specializes to nonlinear state-space models.

SMC is based on importance sampling and uses weighted samples to iteratively approximate a sequence of probability distributions, referred to as the target distributions. The target distribution at iteration $t$, denoted $\pi_t(x_{1:t})$, is a joint probability density function (pdf) over the latent variables $x_{1:t} = (x_1, \ldots, x_t)$. The pdf is

$$\pi_t(x_{1:t}) = \frac{1}{Z_t} \gamma_t(x_{1:t}), \quad t = 1, \ldots, T, \quad (20)$$

where $\gamma_t$ is a positive integrable function that can be evaluated point-wise, and the normalization constant $Z_t = \int \gamma_t(x_{1:t}) \, dx_{1:t}$ ensures that $\pi_t$ is a pdf.

SMC approximates each target $\pi_t$ by a collection of $N$ weighted particles $\{x_{1:t}^i, w_t^i\}_{i=1}^N$ that are generated according to Algorithm 1.
Algorithm 1 Sequential Monte Carlo (all steps are for $i = 1, \ldots, N$)

Sample $x_i^t \sim q_i(x_i)$ independently, compute unnormalized weights $\tilde{w}_i^t = \gamma_i(x_i^t)/q_i(x_i)$ and normalize $w_i^t = \tilde{w}_i^t / \sum_{j=1}^{N} \tilde{w}_i^t$.

for $t = 2, \ldots, T$ do

(a) Resample: Simulate ancestor indices $a_i^t$ with probabilities $\{w_i^{t-1}\}_{i=1}^N$.
(b) Propagate: Simulate $x_i^t \sim q_i(x_t \mid x_i^{t-1})$ and set $x_{1:t}^i = \{x_i^{a_i^t}, x_i^t\}$.
(c) Weight: Compute $\tilde{w}_i^t = w_i(x_{1:t}^i)$ and normalize $w_i^t = \tilde{w}_i^t / \sum_{j=1}^{N} \tilde{w}_j^t$.

The algorithm initializes each particle by simulating independent samples from a user-specified proposal distribution $q_i(x_i)$ and computing the normalized weights $\{w_i^1\}_{i=1}^N$. How to choose the proposal distribution will be discussed later. For each iteration $t$, the particles are resampled by simulating ancestor indices $a_i^t$ for each particle, with probabilities given by the normalized weights $\{w_i^{t-1}\}_{i=1}^N$. This produces an unweighted approximation of $\pi_{t-1}$ by creating replicates of particles with high weights and discarding particles with low weights. Given this unweighted approximation, the particles are propagated by simulating the proposal $q_i(x_t \mid x_{t-1})$ for each particle and computing new, unnormalized weights according to the weighting function

$$w_i(x_{1:t}) = \frac{\gamma_i(x_{1:t})}{\gamma_{t-1}(x_{1:t-1}) q_i(x_t \mid x_{t-1})}.$$  \hfill (21)

The proposal can be chosen arbitrarily, as long as its support includes the support of the target $\pi_t(x_t \mid x_{1:t-1})$. One possible choice is the locally optimal proposal,

$$q_i(x_t \mid x_{t-1}) \propto \frac{\gamma_i(x_{1:t})}{\gamma_{t-1}(x_{1:t-1})},$$ \hfill (22)

which minimizes the conditional variance of the weights at iteration $t$. The weighted particles generated by Algorithm 1 can be used to approximate each intermediate target distribution $\pi_t$ by the empirical distribution $\tilde{\pi}_t(x_{1:t}) = \sum_{i=1}^{N} w_i \delta_{x_{1:t}^i}(x_{1:t})$. The algorithm also provides unbiased estimates of the normalizing constants $Z_t$,

$$\hat{Z}_t = \prod_{s=1}^{t} \frac{1}{N} \sum_{i=1}^{N} \tilde{w}_s^t,$$ \hfill (23)

see [37] and [38] for details. Textbook introductions to SMC are provided in [39] [40] [41].

4.1 Designing the target for state-space models

To enable inference in state-space models, the target distributions must be connected to a relevant sequence of distributions. Analog to (19), the joint pdf of the latent states and observations up to time $t$ is

$$p(y_{1:t}, x_{1:t}) = p(x_1) \prod_{s=1}^{t} p(y_s \mid x_s) \prod_{s=2}^{t} p(x_s \mid x_{s-1}),$$ \hfill (24)

where the dependence on the parameters has been dropped to simplify the notation. Letting the unnormalized target distributions be $\gamma_t(x_{1:t}) = p(y_{1:t}, x_{1:t})$ for $t = 1, \ldots, T$ is therefore a
natural choice that links the sequential structure of the algorithm with the sequential nature of the model. Indeed, this link has driven the development of SMC methods and is the foundation for the original particle filter (which is a special case of SMC). See “Background: The bootstrap particle filter” for further details and an alternative derivation of the particle filter. The targets $\gamma_t(x_{1:t}) = p(y_{1:t} | x_{1:t})$ recursively estimates the joint filtering distribution $p(x_{1:t} | y_{1:t})$ and produces an unbiased estimate of the marginal likelihood $Z_t = p(y_{1:t})$.

In offline system identification problems, it is often not important to estimate the filtering distributions. In fact, in most cases, only the final target $\pi_T(x_{1:T})$ is of interest. For the state-space model, this corresponds to the final filtering distribution, which equals the smoothing distribution, namely the distribution of all states given all observations $p(x_{1:T} | y_{1:T})$. Requiring only $\pi_T(x_{1:T}) = p(x_{1:T} | y_{1:T})$ gives a larger freedom in designing the sequence leading up to the final target. This freedom can be used to improve the final estimate, both for estimating the marginal likelihood and approximating the smoothing distribution.

A problem with using the filtering distributions as target distributions is that the support for the state $x_t$ at iteration $t$ is determined by the filtering approximation at iteration $t$. The marginals of the filtering distribution $p(x_t | y_{1:t})$ and the marginal smoothing distribution $p(x_t | y_{1:T})$ can differ a lot depending on the model and the observations. When the SMC algorithm propagates the particles from iteration $t - 1$ to $t$, the support for the states $x_{1:t-1}$ does not change. The new state $x_t$ has a distribution determined by the model, but $x_{1:t-1}$ has finite support. This leads to a situation where eventually all particles share a common ancestor and the marginal distribution of the states before that time is represented by a single sample. This is referred to as degeneracy.

The freedom in designing the intermediate target distributions leading up to the final target raises the question of an optimal sequence. For estimating the normalizing constant $Z_T$, the optimal target distributions are marginals of the final target

$$\pi_t(x_{1:t}) = \int \pi_T(x_{1:T}) \, dx_{t+1:T} = \pi_T(x_{1:t}).$$

(25)

The locally optimal proposal for the optimal target sequence is

$$q_t(x_t | x_{t-1}) \propto \frac{\gamma_t(x_{1:t})}{\gamma_{t-1}(x_{1:t-1})} = \frac{\gamma_T(x_{1:t})}{\gamma_T(x_{1:t-1})} = \gamma_T(x_t | x_{1:t-1}) = p(x_t | x_{1:t-1}, y_{1:T}),$$

(26)

where the last equality is true for the state-space model. This proposal is typically not possible to simulate, since it requires exact samples from the posterior distribution. However, using marginals of the final distribution as target distributions in combination with the locally optimal proposal leads to $Z_t = Z_T$ with probability 1 for all $t$, since

$$w_t(x_t) = \frac{\gamma_t(x_t)}{q_t(x_t)} = \frac{\gamma_t(x_t)}{\gamma_t(x_t) / \int \gamma_t(x_t) \, dx_t} = \int \gamma_T(x_{1:T}) \, dx_{t+1:T} = Z_T,$$

(27)

and $w_t(x_t) = \gamma_t(x_{1:t}) / (\gamma_{t-1}(x_{1:t-1}) q_t(x_t | x_{t-1})) \equiv 1$ for $t = 2, \ldots, T$.

For the optimal target, both the weight expression [21] and sampling or evaluating the proposal are intractable. It does, however, give useful guidance for the design of both the intermediate targets and the proposal. For a state-space model, the weight expression is

$$w_t(x_{1:t}) = \frac{p(x_{1:t}, y_{1:T})}{p(x_{1:t-1}, y_{1:T}) q_t(x_t | x_{t-1})} = \frac{p(x_t | x_{t-1}) p(y_t | x_t) p(y_{t+1:T} | x_t)}{q_t(x_t | x_{t-1}) p(y_{t:T} | x_{t-1})},$$

(28)
Let \( \psi_t(x_t) = p(y_{t+1:T} \mid x_t) \) and \( \psi_0 \equiv \psi_T \equiv 1 \). The functions \( \psi_t \) are called the twisting potentials. Evaluating the weight expression (28) is intractable for most models, since it would require evaluation of the conditional marginal likelihood. However, since the twisting potential for the final target \( \psi_T \) is constant, the choice of twisting potentials does not alter the final target (only the sequence leading up to it). From (28), the optimal twisting potential for state-space models is \( \psi^*_t(x_t) = p(y_{t+1:T} \mid x_t) \). One possibility is to approximate the conditional likelihood using deterministic inference methods such as the extended or the unscented Kalman filter. It is also possible to use an extended Rauch–Tung–Striebel or unscented smoother to construct the proposal from a deterministic approximation. An example where the likelihood \( p(y_{1:T} \mid \theta) \) is estimated using both a bootstrap particle filter and a twisted particle filter is available in “Example: Twisted particle filter for the cascaded water tanks”.

In this way, SMC can be seen as running on top of a biased inference method known to work well while also providing statistical guarantees such as unbiased estimates of the marginal likelihood.

### Background: The bootstrap particle filter

Sequential Monte Carlo (SMC) methods were initially known as particle filters and presented as an approximate solution to the optimal filtering problem for nonlinear state-space models. In this section, the SMC method is introduced via optimal filtering in state-space models.

The filtering problem amounts to computing the marginal filtering distribution \( p(x_t \mid y_{1:t}) \) sequentially in time for \( t = 1, \ldots, T \). By applying the laws of conditional distributions, the filtering distribution can be written

\[
p(x_t \mid y_{1:t}) = \frac{p(y_t \mid x_t)p(x_t \mid y_{1:t-1})}{p(y_t \mid y_{1:t-1})}, \tag{29}
\]

where the denominator is \( p(y_t \mid y_{1:t-1}) = \int p(y_t \mid x_t)p(x_t \mid y_{1:t-1}) \, dx_t \). The expression (29) is often referred to as the update step, since it updates the predictive distribution \( p(x_t \mid y_{1:t-1}) \) with the most recent observation \( y_t \). Using marginalization, the predictive distribution is

\[
p(x_t \mid y_{1:t-1}) = \int p(x_t \mid x_{t-1})p(x_{t-1} \mid y_{1:t-1}) \, dx_{t-1}, \tag{30}
\]

where the second term in the product is the filtering distribution at time \( t - 1 \). This is referred to as the prediction step, since it predicts the next state given the current filtering distribution.

Both (29) and (30) are typically intractable. Particle filters approximate these distributions using weighted samples. Assume that the filtering density at time \( t - 1 \) has been approximated by a set of \( N \) weighted samples \( \{w^i_{t-1}, x^i_{t-1}\}_{i=1}^N \) as \( \tilde{p}(x_{t-1} \mid y_{1:t-1}) = \sum_{i=1}^N w^i_{t-1} \delta_{x^i_{t-1}}(x_{t-1}) \), where \( \delta_x \) is the Dirac delta. The prediction step (30) then reduces to

\[
p(x_t \mid y_{1:t-1}) \approx \int p(x_t \mid x_{t-1})\tilde{p}(x_{t-1} \mid y_{1:t-1}) \, dx_{t-1} = \sum_{i=1}^N w^i_{t-1}p(x_t \mid x^i_{t-1}). \tag{31}
\]
If the distribution $p(x_t | x_{t-1})$ can be simulated, (31) is also easily simulated by first sampling ancestor indices $a_i^t$ from a categorical distribution with probabilities proportional to $\{w_i^{t-1}\}_{i=1}^N$, and then sampling $p(x_t | x_{a_i^t}^t)$. This provides an unweighted approximation of the predictive distribution $\hat{p}(x_t | y_{1:t-1}) = \sum_{i=1}^N \delta_{x_i^t}(x_t)$. Inserting this approximation in (29) yields a weighted approximation to the filtering distribution at time $t$ given by

$$\hat{p}(x_t | y_{1:t}) = \sum_{i=1}^N p(y_t | x_i^t) \delta_{x_i^t}(x_t) \frac{\sum_{j=1}^N p(y_t | x_j^t)}{\sum_{j=1}^N p(y_t | x_j^t)}.$$  \(32\)

Comparing (31) and (32), the normalized weights at time $t$ are

$$w_i^t = \frac{p(y_t | x_i^t)}{\sum_{j=1}^N p(y_t | x_j^t)}, \quad i = 1, \ldots, N. \hspace{1cm} (33)$$

The bootstrap particle filter is summarized in Algorithm 2. Note that this is a special case of Algorithm 1 with the unnormalized target distributions given by (24) and the proposal chosen as the state transition, $q_t(x_t | x_{t-1}) = p(x_t | x_{t-1})$.

**Algorithm 2** Bootstrap particle filter (all steps are for $i = 1, \ldots, N$)

Sample $x_i^1 \sim p(x_1)$ and compute the weights $w_i^1$ using (33).

for $t = 2, \ldots, T$ do

(a) Resample: Sample ancestor indices $\{a_i^t\}_{i=1}^N$ from a categorical distribution with probabilities $\{w_i^{t-1}\}_{i=1}^N$.

(b) Propagate: Simulate $x_i^t \sim p(x_t | x_{a_i^t}^{t-1})$.

(c) Weight: Compute weights $w_i^t$ using (33).

---

**Example: Twisted particle filter for the cascaded water tanks**

One important and useful property of sequential Monte Carlo is that it provides unbiased estimates of the normalizing constant. For models where the final unnormalized target distribution $\gamma_T(x_1:T)$ equals the full probabilistic model $p(x_1:T, y_1:T)$, the final estimate of the normalizing constant $\hat{Z}_T$ is an estimate of the likelihood $p(y_1:T)$.

Twisted particle filters give unbiased estimates for all twisting potentials. However, the choice of twisting potential affects properties of the likelihood estimate. Using the optimal twisting potential yields an estimator with zero variance, since it produces the true likelihood with probability one.

For the cascaded water tanks model, an extended Kalman filter works well for estimating the latent states, but it does not give accurate estimates of the likelihood. It is possible to combine the particle filter with an extended Kalman filter to approximate the optimal twisting potential. The resulting estimates of the likelihood are shown in Figure 6. The twisted filter is compared to a standard bootstrap filter, both use the transition distribution of the state-space model $p(x_t | x_{t-1})$ as proposal. Both filters were run 100 times with the number of particles varying from 10 to 1000. Increasing the number of particles decreases the variance of the estimate. The figure
shows that the twisted particle filter allows for a reduction of the number of particles by an order of magnitude.

![Figure 6: Log likelihood estimates for the parameters produced by a bootstrap particle filter and a twisted particle filter where the twisting potential has been computed using an extended Kalman filter. Both filters produce unbiased estimates of the likelihood. The log-likelihood is negatively biased, since the logarithm is a concave function. Large variance gives both wider spread and more negative bias. The twisted particle filter requires approximately an order of magnitude fewer particles to produce estimates comparable to the bootstrap particle filter.](image)

5 Identification using marginalization

This section briefly introduces the top row in Figure 5, that is, the marginalization strategies offered by direct gradient search and the MH algorithm when they are applied to state-space models.

5.1 Direct gradient-based approach

In many cases, the ML optimization problem can be approached using standard gradient-based search. The essential step of this approach involves iteratively updating the parameters via

$$\theta_{k+1} = \theta_k + \alpha_k d_k,$$

where $d_k$ is a search direction, and $\alpha_k > 0$ is a so-called step-length that is determined so that the iterates converge. For this strategy to be successful, it is enough to require that the search direction $d_k$ is an ascent direction, which can be guaranteed for $d_k = \nabla_\theta \ln p(y_{1:T} | \theta_k)$ —the gradient of the log-likelihood.

Ascent can be ensured more broadly by allowing any negative definite matrix $H_k$ to scale the gradient, and importantly, this can lead to significant improvements in convergence rate.
classical result along these lines reveals that the inverse Hessian matrix involving second-order curvature information is an ideal choice in this regard. Unfortunately, for the class of nonlinear state-space models considered in this article, it is not tractable to compute the log-likelihood, its gradient, or the associated Hessian matrix. The essential reasons stem from the intractable integrals required in evaluating the log-likelihood, see (29) and (30).

Given the success of SMC methods in this context, it is tantalizing to consider how they may be employed for approximating the gradient and Hessian as well. The gradient and Hessian can be approximated using SMC methods with complexity that grows linearly in data length [46], as captured in Algorithm 3. This is just one of the methods that can be employed to compute the gradient and Hessian. There are many SMC-based alternatives, see [46] and references therein for other approaches. While the method outlined in Algorithm 3 provides estimates of the gradient and Hessian, these estimates are based on SMC methods and are therefore stochastic. This raises a further issue in that standard gradient ascent methods are not intended to solve stochastic optimization problems.

Fortunately, due to their importance, stochastic optimization problems are rather well studied. The first known stochastic optimization algorithm was introduced in [47]. It uses only first-order (gradient) information, motivating the name stochastic gradient (SG) method. Importantly, the majority of SG algorithms are not ascent methods. Rather, they are Markov chain methods, since their update rule defines a Markov chain.

Following the landmark article [47], many extensions have been developed within the statistics and automatic control communities. Some notable works include convergence results [48, 49, 50], online parameter estimation and system identification in [51, 52], adaptive control strategies [53], and general books in the area [54, 55, 56]. The primary focus of current research activity is directed towards improving convergence rates. Two important aspects that impact the convergence rate are:

- Poor problem scaling, which can lead to slow convergence [57].
- Classical step-length formulas are conservative [58, 59].

Regarding the first aspect, incorporating second-order information can greatly improve convergence, as highlighted in [57]. Regarding the second aspect, the step-length can be determined using stochastic line-search procedures, modeled after the backtracking line-search with Armijo conditions [60, 61, 62]. Recent work along these lines can be found in [63, 64].

In “Example: Direct gradient-based search fr identification of cascaded water tank parameters”, the gradient-based search discussed above (with likelihood gradients and Hessians estimated using Algorithm 3) is applied for learning the parameters of the coupled-tank model in “Example: Cascaded water tanks”. Different combinations of the parameters, corresponding to different physical interpretations of the model, are also evaluated.

### 5.2 The Metropolis–Hastings algorithm

The MH algorithm is perhaps the most well-known and most commonly used Markov chain Monte Carlo (MCMC) method, largely due to its simplicity. A general, but brief, introduction to MCMC methods is available in “Background: Markov chain Monte Carlo”. For the MH algorithm, samples \( x \) from a target distribution \( \pi(x) \) are generated by iteratively executing two steps. First, a new candidate sample \( x' \) is generated from a proposal distribution \( q(x' | x(m)) \),
for the acceptance probability poses two difficulties: A proposal distribution must be chosen, if accepted, the proposed sample is kept and assigned as the next sample, or rejected according to the acceptance probability

where $p$ is the normalization constant.

Algorithm 3 Compute log-likelihood gradient and Hessian estimate

(a) Set $\alpha_i^0 = 0, \beta_i^0 = 0$, for $i = 1,\ldots,N$.
(b) Run Algorithm 1 to generate $\{x_i^t, a_i^t\}_{j=1}^N$ and $w_i^t$.

for $t = 1,\ldots,T$ do

Compute

$$v_t = \sum_{j=1}^N \gamma_i^j w_i^j, \quad B_t = v_t v_t^\top + \sum_{j=1}^N \left[ \phi_i^j + \gamma_i^j (\gamma_i^j)^\top \right] w_i^j,$$

where

$$\gamma_i^j = \nabla_{\theta} \log p(y_t \mid x_i^t, \theta) + \nabla_{\theta} \log p(x_i^t \mid x_{i-1}^t, \theta) + \alpha_i^{a_i^t}, \quad \alpha_i^t = \gamma_i^t - v_t,$$

$$\phi_i^t = \nabla_{\theta}^2 \log p(y_t \mid x_i^t, \theta) + \nabla_{\theta}^2 \log p(x_i^t \mid x_{i-1}^t, \theta) + \beta_i^{a_i^t}, \quad \beta_i^t = \phi_i^t - B_t.$$

(c) Approximate $\nabla_{\theta} \ln p(y_{1:T} \mid \theta)$ and $\nabla_{\theta}^2 \ln p(y_{1:T} \mid \theta)$ via

$$\nabla_{\theta} \ln p(y_{1:T} \mid \theta) \approx \sum_{t=1}^T v_t, \quad \nabla_{\theta}^2 \ln p(y_{1:T} \mid \theta) \approx \sum_{t=1}^T B_t.$$

where $x(m)$ is the most recently generated sample. The proposed sample $x'$ is then accepted or rejected according to the acceptance probability

$$\alpha = \min \left( 1, \frac{\pi(x') q(x(m) \mid x')}{\pi(x(m)) q(x' \mid x(m))} \right).$$

If accepted, the proposed sample is kept and assigned as the next sample, $x(m+1) = x'$. If rejected, the proposed sample is discarded and the most recently accepted sample is assigned as the next sample, $x(m+1) = x(m)$. See [65] for a more detailed description of the algorithm. An important observation is that the acceptance probability [38] contains a ratio of target distributions, implying that evaluation of the normalization constant is not required to compute the acceptance probability. This is essential for practical applications, since the normalization constant is typically intractable. Another observation is that the MH algorithm requires a proposal distribution, chosen by the user. The proposal affects the convergence rate of the algorithm. Ideally, the proposal should ensure fast exploration of the parameter space while maintaining a high acceptance probability. In practice, however, a high acceptance probability typically aligns with a slow exploration and vice versa. In particular, this is the case for high-dimensional problems.

For the system identification problems considered in this article, the target distribution of interest is the posterior distribution of the parameters, given in [17]. The acceptance probability for this target is, by insertion in [38],

$$\alpha = \min \left( 1, \frac{p(y_{1:T} \mid \theta') p(\theta')}{p(y_{1:T} \mid \theta(m)) p(\theta(m))} \frac{q(\theta(m) \mid \theta')}{q(\theta'( \mid \theta(m))} \right),$$

where the normalization constant $p(y_{1:T})$ in the target distribution [17] cancels. The expression for the acceptance probability poses two difficulties: A proposal distribution must be chosen,
and the likelihood \( p(y_{1:T} \mid \theta) \) must be evaluated. The former (choosing a proposal) is in a sense simpler, since it is possible to run the algorithm for any proposal whose support contains the support of the target distribution. The evaluation of the likelihood will, if intractable, prevent even running the algorithm. An appealing idea is to replace the likelihood, when intractable, with an estimate. This modification can be used to construct an algorithm that yields samples from the desired target distribution, provided the likelihood estimate is part of the generated Markov chain (likelihood estimate and parameters are accepted or rejected collectively), and the likelihood estimator is nonnegative and unbiased. Such a likelihood estimator can be obtained from running SMC and evaluating \((23)\). The resulting method belongs to the particle Markov chain Monte Carlo (PMCMC) family and is referred to as particle marginal Metropolis–Hastings (PMMH) \([29]\).

The claim that any likelihood estimator that is unbiased and nonnegative can be used, deserves some motivation. The most intuitive argument stems from deriving PMMH using a pseudo-marginal approach \([66]\), where the target distribution is extended to include the likelihood estimate \( \hat{Z} \). A proposal distribution for the MH algorithm that yields the acceptance probability \((39)\) with \( p(y_{1:T} \mid \theta) \) replaced with its estimated value, can be constructed by first sampling the parameters from some suitable proposal distribution, and then sampling \( \hat{Z} \) from its distribution. This approach is valid only if the extended target distribution is a valid probability distribution and the marginal distribution of the extended target with respect to the parameters is the true target distribution. These conditions are fulfilled only if the likelihood estimator is nonnegative and unbiased. In \([30]\) a tutorial-style derivation of this result is presented that is based on the original derivation in \([66]\).

For the case considered here, where the likelihood estimate is obtained using SMC, an alternative proof that the resulting algorithm is valid is provided in \([29]\). It too is based on extending the target distribution. However, the extension this time is by including all random variables that are generated in the SMC algorithm. The PMMH sampler can then be obtained by designing a proposal distribution for this extended target distribution.

**Example: Direct gradient-based search for identification of cascaded water tank parameters**

In this section, the gradient-based search approach is demonstrated using the the cascaded water tank model described in \((8a)-(8c)\) and the data from \([19]\). The data has \( T = 1024 \) input–output measurements available for estimation and a further \( T_v = 1024 \) input–output measurements available for validation purposes.

Recall that the parameters are \( \theta = \{k_1, k_2, k_3, k_4, k_5, k_6, \sigma_w^2, \sigma_e^2\} \), and for the purposes of illustration, they are initialized to

\[
\theta_1 = \{0.2, 0, 0.2, 0, 0.2, 0.2, 0.1, 0.1\}.
\]  

While it is possible to also add the initial states as unknown parameters, they are simply chosen as \( x_1 = [6, y_1]^\top \) here. It is instructive to consider several other model structure choices to demonstrate their effect on the predictive performance. Different model structures can be explored by eliminating selected terms in the model (by setting the accompanying coefficients to zero), and observing the predictive performance in each case. It is important to notice the relative simplicity in changing the model structure,
and its physical interpretation. The latter is often difficult or impossible in more general black-box models. In all cases considered, $N = 50$ particles were used.

Consider the model where coefficients $k_2$, $k_4$, and $k_6$ are all zero. This has the effect of removing the linear loss terms and the overflow event from the model. Without the loss and overflow terms, the parameters are $\theta = \{k_1, k_3, \sigma_w^2, \sigma_e^2\}$. The gradient-search algorithm was allowed to run for 100 iterations. Using the obtained estimate in combination with the validation data input sequence, a simulated output, $\hat{y}_t$, can be computed by iterating (8a)-(8c). The simulated output can be compared with the observed validation data output, $y_t$, to provide a measure of predictive performance by computing the root-mean-square simulation error

$$e_{RMS} = \sqrt{\frac{1}{T} \sum_{t=1}^{T} \|y_t - \hat{y}_t\|^2}. \quad (41)$$

The predictive metric $e_{RMS}$ for the estimated parameters and their log-likelihood is captured in Table 1.

Inspired by the prospect that the model could benefit from including more terms, the linear loss coefficients $k_2$ and $k_4$ are added to the estimated parameter list. After 100 iterations, the predictive metric $e_{RMS}$ for the estimated parameters is captured in Table 1. While the log-likelihood value has increased, the predictive performance has degraded with these parameter additions. This highlights the often-observed phenomena that improved training cost does not necessarily lead to improved predictive performance on validation data.

The reverse situation is considered next, where $k_2$ and $k_4$ are set to zero, and the overflow event coefficient $k_6$ is allowed to be estimated. Table 1 reveals that the log-likelihood increased, and that the predictive performance also improved relative to both previous cases. The implication is that the overflow event should be modeled.

Finally, the case that includes all parameters was tested, revealing the estimated values

$$\hat{\theta} = \{0.0392, 0.0016, 0.0637, -0.0059, 0.0414, 0.2572, 0.0012, 0.0001\} \quad (42)$$

The estimates of the two loss coefficients $k_2$ and $k_4$ are both an order of magnitude less than the estimated $k_1$ and $k_3$ coefficients. The log-likelihood and predictive metric are recorded in Table 1 revealing that the log-likelihood has increased compared to the

| Table 1: Log likelihood and predictive performance for different parameter combinations for the cascaded water tank example. Higher log likelihood values indicate improved fit between the model predicted and the measured output. Lower $e_{RMS}$ values indicate reduced simulation error on validation data. |
|---------------------------------------------------------------|
| Parameters | Log likelihood | $e_{RMS}$ |
| $k_{1,3,5}, \sigma_w^2, \sigma_e^2$ | 1388 | 0.64 |
| $k_{1,2,3,4,5}, \sigma_w^2, \sigma_e^2$ | 1540 | 0.90 |
| $k_{1,3,5,6}, \sigma_w^2, \sigma_e^2$ | 1810 | 0.49 |
| $k_{1,6}, \sigma_w^2, \sigma_e^2$ | 1979 | 0.28 |
previously considered models, and the predictive performance on the validation dataset was best of all models tested. A plot of the measured and simulated outputs is provided in Figure 7.

Figure 7: Measured and simulated outputs for direct gradient-based search on the cascaded water tank example. The validation data was not used for estimating the model, and the simulated output relies on the input signal and model only.

Background: Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) methods are useful for evaluating expectation integrals that are otherwise challenging to compute. In the context of Bayesian identification, MCMC methods can be employed to compute expectations relative to the posterior distribution $p(\theta | y_{1:T})$ of the form

$$ I = \int f(\theta) p(\theta | y_{1:T}) \, d\theta, \quad (43) $$

where the function $f(\cdot)$ may be quite general. For example, $f$ could be the indicator function that $\theta$ belongs to a given set. If $f(\theta) = \theta$, the above integral computes the conditional expected mean, which is also the minimum-mean-squared estimator 67.

Solving these types of integrals is generally intractable in closed-form, and classical quadrature methods are limited to low dimensions only. An alternative computational approach relies on the law of large numbers to provide estimates of (43) via so-called Monte-Carlo integration, where $M$ samples $\{\theta(m)\}_{m=1}^M$ from $p(\theta | y_{1:T})$ are used to compute a sample average of $f(\cdot)$ according to

$$ \hat{I}_M = \frac{1}{M} \sum_{m=1}^M f(\theta(m)), \quad \theta(m) \sim p(\theta | y_{1:T}). \quad (44) $$
6 Identification using data augmentation

This section briefly introduces the bottom row in Figure 5, that is, the data augmentation strategies offered by expectation maximization and Gibbs sampling when they are applied to state-space models.

6.1 Expectation maximization

The EM approach is based on the idea that if (in addition to the output sequence \(y_{1:T}\)) the state sequence \(x_{1:T}\) were known, then \(\theta\) could be estimated by solving the ML problem

\[
\hat{\theta} = \arg \max_{\theta} \ln p(y_{1:T}, x_{1:T} | \theta),
\]

over the joint state–output data \(p(y_{1:T}, x_{1:T} | \theta)\) in (19). The above ML problem does not involve any troubling integration terms, and in principle, this problem could be approached using standard optimization tools. Furthermore, the solution \(\hat{\theta}\) can be expressed in closed form in some cases, such as linear time-invariant state-space models.

Unfortunately, the state sequence \(x_{1:T}\) is rarely available. The EM approach combats this by replacing the joint log-likelihood \(\ln p(y_{1:T}, x_{1:T} | \theta)\) with its expected value over the unobserved (or missing) state sequence \(x_{1:T}\), conditioned on the measured outputs \(y_{1:T}\)—the so-called expectation or E-step. It then aims to solve the surrogate maximization problem (the M-step)

\[
\theta^* = \arg \max_{\theta'} \int \ln p(y_{1:T}, x_{1:T} | \theta)p(x_{1:T} | y_{1:T}, \theta') \, dx_{1:T}
\]

instead. The conditional distribution \(p(x_{1:T} | y_{1:T}, \theta')\) relies on a different parameter \(\theta'\), so that the variables being optimized over only appear in the joint log-likelihood term \(\ln p(y_{1:T}, x_{1:T} | \theta)\). The EM method progresses by solving for \(\theta^*\), then updating \(\theta' \leftarrow \theta^*\) and repeating. This can
be captured succinctly by indexing the parameters with iteration number \( k \), so that starting with \( \theta_1 \) repeatedly solves the problem

\[
\theta_{k+1} = \arg \max_\theta \int \ln p(y_{1:T}, x_{1:T} \mid \theta) p(x_{1:T} \mid y_{1:T}, \theta_k) \, dx_{1:T},
\]

(48)

for \( k = 1, 2, \ldots \). Importantly (and perhaps surprisingly), the sequence of \( \theta_k \)'s is guaranteed not to decrease the log-likelihood \([33]\), that is

\[
\ln p(y_{1:T} \mid \theta_{k+1}) \geq \ln p(y_{1:T} \mid \theta_k),
\]

(49)

which explains the primary mechanism for solving the maximum-likelihood problem \([16]\) using EM. Unfortunately, the expectation integral in \([48]\) is not generally tractable. Nevertheless, in keeping with the theme of this article, this integral can be approximated using SMC methods that target smoothed distributions \( p(x_{1:T} \mid y_{1:T}, \theta) \). In essence, the integral is approximated by a finite sum

\[
\sum_{i=1}^N w_i(\theta_k) \ln p(y_{1:T}, x_{1:T}^i(\theta_k) \mid \theta) \approx \int \ln p(y_{1:T}, x_{1:T} \mid \theta) p(x_{1:T} \mid y_{1:T}, \theta_k) \, dx_{1:T}
\]

(50)

where \( x_{1:T}^i(\theta_k) \) are particles and associated weights \( w_i(\theta_k) \) from a particle smoother that depends on \( \theta_k \). This approach has been explored by several authors \([41, 70, 71]\). It gives rise to so-called particle expectation maximization (pEM) methods, where

\[
\theta_{k+1} = \arg \max_\theta \sum_{i=1}^N w^i(\theta_k) \ln p(y_{1:T}, x_{1:T}^i(\theta_k) \mid \theta)
\]

(51)

is iterated. Different particle smoothing algorithms can be used to address the underlying smoothing problem in the E-step of the EM algorithm, resulting in different variants of the pEM method. When working with state-space models, one property that can be exploited is that the joint likelihood \([19]\) factorizes over the time steps. Therefore, even though \([47]\) seems to require the solution to a joint smoothing problem, it is enough to compute marginal smoothing estimates to implement the EM algorithm. This enables, for example, fixed-lag smoothers to be used, see \([70]\) for details. There are also extensions of pEM methods that couple the E-step and M-step to make more efficient use of the generated particles. One such method is explained in “Background: Particle stochastic approximation expectation maximization”, and demonstrated on the coupled-tank experiment in “Example: Particle stochastic approximation expectation maximization for identification of cascaded water tank parameters”.

### 6.2 Gibbs sampling

The Gibbs sampler \([34]\) is one of the most commonly used algorithms in the MCMC family. Like all MCMC algorithms, it generates samples of some variable of interest \( x \) from its target distribution \( \pi(x) \). On a high level, the main idea behind the Gibbs sampler is to turn a potentially high-dimensional sampling problem into several subproblems of lower dimension that are, hopefully, simpler to sample from. This is achieved by splitting the variable of interest \( x \) into smaller components, then updating these components by iteratively sampling them, one at a time, from their full conditional distributions while keeping all other components fixed. A
more mathematical description of the Gibbs sampler as well as a simple example is described in “Example: Gibbs sampling”.

For the Bayesian formulation of the system identification problem considered in this article, the target distribution of interest is the parameter posterior, \( p(\theta \mid y_{1:T}) \), given in (17), with the parameters \( \theta \) being the variable of interest. Targeting this distribution directly with a Gibbs sampler is difficult, since the intractable likelihood \( p(y_{1:T} \mid \theta) \) is required. Instead, the Gibbs sampler targets the joint distribution \( p(\theta, x_{1:T} \mid y_{1:T}) \), where the states have been introduced as auxiliary variables. Gibbs sampling is therefore a method that most naturally belongs to the data augmentation strategy in Figure 5. It is important to note that given that samples \( \{\theta(m), x_{1:T}(m)\}_{m=1}^{M} \) from the augmented target distribution \( p(\theta, x_{1:T} \mid y_{1:T}) \) are available, the sequence of only the parameter samples \( \{\theta(m)\}_{m=1}^{M} \) provides an approximation of the parameter posterior \( p(\theta \mid y_{1:T}) \), since the parameter posterior is a marginal of the augmented target. A Gibbs sampler that generates samples from the augmented target \( p(\theta, x_{1:T} \mid y_{1:T}) \) can be designed to alternate between sampling a new state trajectory given the current parameters, \( x_{1:T}(m+1) \sim p(x_{1:T} \mid \theta(m), y_{1:T}) \), and sampling new parameters given the current state trajectory, \( \theta(m+1) \sim p(\theta \mid x_{1:T}(m+1), y_{1:T}) \). It is typically feasible to sample the parameters. However, sampling the state trajectory implies sampling from a high-dimensional, and in many cases intractable, distribution. The distribution for generating new states can be recognized as the final target used in SMC for nonlinear state-space models. However, simply applying Algorithm 1 to sample new states does not yield samples from the correct target distribution \( 29 \). In the following section on particle Gibbs (PG), a solution to this problem is discussed.

In contrast to other MCMC methods, like MH, no design of a proposal distribution is necessary in Gibbs sampling. This is a clear advantage, especially when sampling from high-dimensional distributions where it can be difficult to find a suitable proposal. On the other hand, Gibbs sampling performs poorly when there are strong dependencies between some of the components.

6.2.1 Particle Gibbs

The PG sampler generates samples from the joint posterior, \( p(\theta, x_{1:T} \mid y_{1:T}) \), by alternately sampling new states conditioned on the current parameters and sampling new parameters conditioned on the current states. To generate samples that have the correct target distribution, the sampling of new state trajectories must be done using a modified version of the SMC method in Algorithm 1 referred to as conditional SMC \( 29 \). Conditional SMC is similar to ordinary SMC, but it takes a reference trajectory \( x_{1:T}' \) as input. The reference trajectory is guaranteed to survive all resampling steps in the SMC algorithm, and at the final iteration, a new reference trajectory is generated according to the weights \( w_{i} \). Intuitively, the reference trajectory can be thought of as a guide for the other trajectories that directs them to suitable parts of the state space. A more detailed motivation and description of conditional SMC is provided in “Background: Conditional sequential Monte Carlo”, and the complete PG sampler is outlined in Algorithm 4. In “Example: Particle Gibbs for identification of dengue fever parameters”, PG is applied to learn the posterior distribution for the parameters of the model describing the spread of dengue fever (introduced in “Example: Dengue fever”) for an outbreak of dengue fever in Micronesia \( 23 \). It also provides an illustration of how parameter priors for a Bayesian identification method can be designed based on attributes of the parameters and prior knowledge about the disease from previous studies.
**Algorithm 4** The particle Gibbs sampler

Initialize: Set $x_{1:T}(1)$ and $\theta(1)$ arbitrarily.

for $m = 1 \ldots M - 1$ do
  Run conditional SMC, inputs $x_{1:T}(m)$ and $\theta(m)$, to draw $x_{1:T}(m + 1)$.
  Draw $\theta(m + 1) \sim p(\theta \mid x_{1:T}(m + 1), y_{1:T})$.

PG is part of a larger group of methods that combine MCMC methods with SMC, referred to as PMCMC methods, that were very briefly introduced in Section 5.2. These methods are *exact approximations* of the corresponding MCMC method in the sense that they return a sequence of samples with the desired target distribution, even though they use SMC to approximate some of the intermediate distributions [29]. Any PMCMC method will, asymptotically as the number of MCMC iterations increases, yield samples from the correct target for any number of particles $N$ in the SMC component. However, the performance usually improves with a larger number of particles. For PG, a too low number of particles in the conditional SMC sampler leads to path degeneracy, that is, all samples share the same ancestry up to some time $t$. This implies that all state trajectories collapse to the reference trajectory and, consequently, the state trajectory is rarely updated. The result is a slow exploration of the space of state trajectories, and the PG sampler is said to “mix poorly”. The number of particles in the SMC component must be at least proportional to the number of time steps for good mixing, which is often prohibitive computation-wise in practice, since the complete algorithm then scales quadratically with time. Even with proportionality between the number of particles and the time steps, the sampler can mix slowly for certain models [72].

Fortunately, there are several extensions of the PG sampler that can significantly improve the mixing without significantly increasing the number of particles. These extensions are typically based on trying to explore the state space around the reference trajectory more efficiently than is done in standard PG. One example is particle Gibbs with backward sampling [73, 74] which, as the name suggests, uses backward sampling to draw a new reference trajectory. Another example is particle Gibbs with ancestor sampling (PGAS) [75], which instead explores by sampling new ancestor indices for the reference trajectory. PGAS is explained in more detail in “Background: Particle Gibbs with ancestor sampling”. A third possibility is to use blocking strategies, as suggested by [76]. This not only reduces the effect of path degeneracy, with improved theoretical and empirical stability as an effect, but also enables parallelization and adaptations of the algorithm for additional performance boosts.

---

**Background: Particle stochastic approximation expectation maximization**

The class of particle expectation maximization (pEM) methods are intuitively appealing, since they make use of particle smoothing algorithms in a natural way to address the intractable smoothing problem arising in the expectation maximization (EM) algorithm. However, it has been realized that they sometimes make inefficient use of the particles that are generated. At iteration $k$, $N$ particles are generated to approximately solve the E-step, followed by an update of the parameter from $\theta_k$ to $\theta_{k+1}$. The generated particles are then discarded, and $N$ new particles must be generated at iteration
However, the parameter update is often minor: \( \theta_{k+1} \approx \theta_k \). In particular when the algorithm starts to converge to an optimal. Based on this observation, an appealing idea is to reuse the particles from iteration \( k \) at iteration \( k + 1 \) and so on. One way to accomplish this is through stochastic approximation EM \([77]\) and its particle-based counterpart, particle stochastic approximation EM (PSAEM) \([18, 78]\).

Without going into details, PSAEM tracks and updates smoothing estimates that are based on particles generated at all previous iterations up to the current iteration \( k \). The “old” particles are gradually down-weighted based on a decreasing step-size sequence. The method also makes use of a particle Markov chain Monte Carlo technique (reviewed in “Particle Gibbs with ancestor sampling”) to create a dependence between particles generated at consecutive EM iterations. When combined, this results in an algorithm that converges (as always, under some conditions) to a local maximum of the likelihood when \( k \to \infty \), despite using a finite number of particles \( N \) at each iteration. To borrow a phrase from \([18]\), the PSAEM algorithm “entangles the convergence” of the EM algorithm with the convergence of the underlying particle smoother. The practical implication is that a convergent particle-based EM algorithm is obtained at a much lower computational cost than conventional EM methods.

Example: Particle stochastic approximation expectation maximization for identification of cascaded water tank parameters

In this section, the particle stochastic approximation expectation maximization (PSAEM) approach is demonstrated on the cascaded water tank model described in \([8a]-[8c]\) using the data available from \([19]\). The setup is identical to that used in “Example: Direct Gradient-Based Search for Identification of Cascaded Water Tank Parameters”. The PSAEM algorithm requires the user to select the number of particles \( N \). For this example, \( N = 50 \) was used. The PSAEM method was allowed to run for 50 iterations, yielding the final parameter estimates

\[
\theta_{50} = \{0.0603, -0.0038, 0.0702, -0.0065, 0.0457, 0.2272, 0.0021, 0.0137\}. \tag{52}
\]

Similar to the direct approach, this estimate indicates that the two loss coefficients \( k_2 \) and \( k_4 \) are both slightly negative, and also an order of magnitude smaller than the estimated \( k_1 \) and \( k_3 \) coefficients. For the estimated parameters, the root-mean-square simulation error \([41]\) was \( e_{\text{RMS}} = 0.29 \). A plot of the measured and simulated outputs is provided in Figure \([8]\).
Figure 8: Measured and simulated outputs for particle stochastic approximation expectation maximization on the cascaded water tank example. The validation data was not used for estimating the model, and the simulated output relies on the input signal and model only.

Example: Gibbs sampling

Assume that samples of some random variable of interest $x$ must be generated from a target distribution $\pi(x)$. In addition, assume that the variable of interest can be split into $d$ components, $x = (x_1, x_2, \ldots, x_d)$. Gibbs sampling can then be employed to iteratively generate samples from the target distribution by updating each component $x_j$ in $x$ separately according to its full conditional distribution $\pi(x_j \mid x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_d)$, while keeping all other components fixed. Updating one component at a time in this way ensures that the sample is still distributed according to the desired target distribution $\pi(x)$ after each update.

To make this a bit more concrete, consider the case when $x$ consists of three components $x_1$, $x_2$, and $x_3$, and the target distribution of interest is $\pi(x) = \pi(x_1, x_2, x_3)$. The Gibbs sampler is first initialized by assigning $x(1) \sim p(x(1))$, where $p$ is some initial distribution chosen by the user. Gibbs sampling now proceeds by first sampling a new value for $x_1$ conditioned on the current values for $x_2$ and $x_3$, that is, $x_1(2) \sim \pi(x_1 \mid x_2(1), x_3(1))$. Next, given the current values for $x_1$ and $x_3$, a new value for the second component, $x_2(2)$, is generated from $\pi(x_2 \mid x_1(2), x_3(1))$. Finally, a new sample $x_3(2)$ is generated, given the current values of $x_1$ and $x_2$, from $\pi(x_3 \mid x_1(2), x_2(2))$. The new sample is $x(2) = (x_1(2), x_2(2), x_3(2))$. Gibbs sampling then proceeds by repeating these three steps to generate more samples. Eventually, as the number of samples increases, the samples will be distributed according to the target distribution. The procedure is summarized in Algorithm 5.
Algorithm 5 The Gibbs sampler for the target \( \pi(x) = \pi(x_1, x_2, x_3) \)

Initialize: For \( x = (x_1, x_2, x_3) \), set \( x(1) \sim p(x(1)) \)

for \( m = 1 \ldots M - 1 \) do

\[
\begin{align*}
& x_1(m + 1) \sim \pi(x_1 \mid x_2(m), x_3(m)), \\
& x_2(m + 1) \sim \pi(x_2 \mid x_1(m + 1), x_3(m)), \\
& x_3(m + 1) \sim \pi(x_3 \mid x_1(m + 1), x_2(m + 1)).
\end{align*}
\]

Background: Conditional sequential Monte Carlo

Conditional sequential Monte Carlo (SMC) is a version of the method presented in Algorithm 1 that takes a reference trajectory \( x'_{1:T} \) as input and generates a new reference trajectory as output. The name conditional SMC stems from the fact that the method generates samples like SMC, but it conditions on the reference trajectory surviving all resampling steps. In each step of conditional SMC, \( N - 1 \) ancestor indices are generated according to the weights of each particle in the same way as in SMC. The last ancestor index, however, is always set to be the index of the reference trajectory. In the propagation step, \( N - 1 \) new states are simulated according to the proposal distribution. The last state is set deterministically to the reference trajectory. Finally, the weights are updated using (21) and normalized. After the final iteration, a new reference trajectory is generated based on the weights \( w_T \). Conditional SMC is summarized in Algorithm 6.

Algorithm 6 Conditional SMC (all steps for \( i = 1, \ldots, N \))

Input: Reference trajectory \( x'_{1:T} \), parameters \( \theta \), observations \( y_{1:T} \).

Draw \( x_i' \sim q_i(x_i) \) independently, set \( x_N' = x_i' \).

Compute weights \( \tilde{w}_i = \gamma_i(x_i')/q_i(x_i) \) and normalize \( w_i = \tilde{w}_i/\sum_{j=1}^N \tilde{w}_j \).

for \( t = 2 \ldots T \) do

\( \begin{align*}
& (a) \text{ Resample: Draw ancestor indices } \{a_i^t\}_{i=1}^{N-1} \text{ with probabilities } \{w_i^{t-1}\}_{i=1}^N \text{ and set } a_N^t = N. \text{ Set } w_{t-1}^t = 1/N. \\
& (b) \text{ Propagate: Simulate } x_i^t \sim q(x_t \mid x_{a_i}^{t-1}) \text{ and set } x_N^t = x_i^t. \text{ Set } x_{1:t}^i = \{x_{a_i}^{t-1}, x^t_i\}. \\
& (c) \text{ Weight: Set } \tilde{w}_i^t = w_t(x_{1:t}^i) \text{ and normalize } w_i^t = \tilde{w}_i^t/\sum_{j=1}^N \tilde{w}_j^t. \\
\end{align*} \)

Output: Draw \( j \) with probability \( \{w_T^i\}_{i=1}^N \), output new reference trajectory \( x'_{1:T} = x_{1:T}^j \).

Example: Particle Gibbs for identification of dengue fever parameters

In this sidebar, particle Gibbs (PG) is used to identify the posterior distribution of the parameters in the model describing the spread of dengue fever introduced in “Example: Dengue fever”, using the dataset from the dengue outbreak on Yap [23]. In PG, the Bayesian view on the parameters is adopted, which requires the specification of prior
Choosing a parameter prior

The unknown model parameters are \( \theta = \{ \lambda^h, \delta^h, \gamma^h, \lambda^m, \delta^m, \gamma^m, \rho \} \). All parameters are probabilities, which implies that they can only take values in the range \([0, 1]\). The beta distribution \( \text{Beta}(\alpha, \beta) \), with shape parameter \( \alpha \) and scale parameter \( \beta \), is a common choice for modeling such a random variable. To fully specify the beta prior, values for the distribution’s parameters \( \alpha \) and \( \beta \) must be selected. The prior should reflect the beliefs about the parameters before seeing the data and can, for instance, be based on results from previous studies of the disease.

The incubation and infection times for humans, as well as the incubation time for mosquitoes are well-studied and can be used to specify priors for the infection and recovery probabilities \( \delta \) and \( \gamma \). In [23], the rate parameters in (13) are specified through Gaussian priors on the incubation and infection times, which are related to the corresponding transition rates as rate=1/time. Mode-matching can be used to incorporate the information from these Gaussian priors into the beta priors. The mode for a Gaussian distribution is simply its mean value. For the beta distribution, the mode is \( m = \frac{\alpha - 1}{\alpha + \beta - 2} \). Assuming that the mean transition time is \( \mu_0 \), the mean transition rate is \( 1/\mu_0 \), and mode-matching results in the relation

\[
\frac{\alpha - 1}{\alpha + \beta - 2} = \frac{1}{\mu_0} \tag{53}
\]

One choice fulfilling (53) is \( \alpha = 1 + \frac{2}{\mu_0} \) and \( \beta = 3 - \frac{2}{\mu_0} \). From [23], \( \mu_0 = 4.4 \) for the human incubation time, \( \mu_0 = 4.5 \) for the human infectious time, and \( \mu_0 = 6.5 \) for the mosquito incubation time. Hence, the beta prior is \( \delta^h \sim \text{Beta}(1 + \frac{2}{4.5}, 3 - \frac{2}{4.5}) \) for the human infection probability, and \( \delta^m \sim \text{Beta}(1 + \frac{2}{6.5}, 3 - \frac{2}{6.5}) \) for the mosquito infection probability.

To understand why the rates in [23] can be directly related to the corresponding probabilities in the probabilistic formulation, (12) and (13), note that the binomial transitions imply geometrically distributed transition times. A geometric distribution with mean transition time \( z \) has a mean transition probability \( 1/z \). Hence, the transition probabilities, for which a prior is designed, are related to the mean transition time in exactly the same way as the transition rates in [23] are. This motivates using the mean transition rate as a proxy for centering the beta prior on the transition probabilities.

The transmission times are unknown for both humans and mosquitoes. The prior on the transmission probabilities \( \lambda^h \) and \( \lambda^m \) is therefore chosen to have parameters \( \alpha = \beta = 1 \), which yields a uniform distribution between 0 and 1 to reflect this uncertainty. The reporting probability is also unknown a priori. Following the same reasoning as for the transmission probabilities, the prior on the reporting probability is chosen to have \( \alpha = \beta = 1 \). Finally, it was noted in the model description that the mosquitoes never recover from the disease. Based on this, the prior for the recover probability for
mosquitoes is set to zero. In conclusion, the parameter priors are

\[
\begin{align*}
\lambda^h &\sim \text{Beta}(1,1), & \lambda^m &\sim \text{Beta}(1,1), \\
\delta^h &\sim \text{Beta}(1 + \frac{2}{4.4}, 3 - \frac{2}{4.4}), & \delta^m &\sim \text{Beta}(1 + \frac{2}{6.5}, 3 - \frac{2}{6.5}), \\
\gamma^h &\sim \text{Beta}(1 + \frac{2}{4.5}, 3 - \frac{2}{4.5}), & \gamma^m &= 0, \\
\rho &\sim \text{Beta}(1,1).
\end{align*}
\]

The parameter priors are visualized in Figure 9.

The initial values are set to

\[
\begin{align*}
E^h_1 &\sim \text{Poisson}(5), & E^m_1 &= 0, \\
I^h_1 - 1 &\sim \text{Poisson}(5), & I^m_1 &= 0, \\
R^h_1 &\sim \text{Uniform}(0, 7370 - E^h_1 - I^h_1), & R^m_1 &= 0, \\
S^h_1 &\sim 7370 - E^h_1 - I^h_1 - R^h_1, & S^m_1 &= 10^u \cdot 7370,
\end{align*}
\]

with \(u \sim \mathcal{U}(-1, 2)\). The Poisson distribution for the initial number of exposed and infectious humans reflects the belief that, on average, five people are expected to be exposed and infectious initially, and the infections occur independent of each other. The \(-1\) in the expression for the initial number of infectious ensures that there is always at least one infectious individual. There have been previous outbreaks of dengue fever on Yap, so the number of immune individuals in the human population is not known apriori. Therefore, the initial number of recovered is sampled from a uniform distribution. The
mosquitoes are assumed to not recover from the disease, which motivates setting the number of recovered to zero initially. Additionally, the lifespan of mosquitoes is short in comparison with that for humans. Thus, it is reasonable to assume that initially the spread of the disease originates from the human population. Hence, the number of exposed and infectious mosquitoes are initially set to zero. Finally, it is unknown how large the mosquito population is. Consequently, it is initialized to be in a large range, allowing for it to be both smaller and much larger than the human population.

**Inferring the parameters from data using particle Gibbs**

A PG sampler was run for $M = 10,000$ iterations with $N = 1,024$ particles in the SMC component. The parameter priors and initial values were selected according to (54) and (55). The simulation was repeated four times. A histogram for the reporting probability $\rho$ (average of four runs) is shown in Figure 10. It can be seen that the reporting probability is likely to be somewhere between 0.2 and 0.5, implying that a large proportion of the dengue cases are never reported to a health center. It is interesting to note that the observed data from the outbreak has led to an update of the prior beliefs about the distribution of the reporting rate from a uniform prior distribution to the posterior in Figure 10. Similar histograms can be generated for the other parameters of the model.

![Figure 10: Histogram for the reporting probability $\rho$ in the dengue model. A particle Gibbs sampler (Algorithm 4) was run for $M = 10,000$ iterations using $N = 1,024$ particles in the sequential Monte Carlo component. The parameter prior is a uniform distribution between 0 and 1, and the initial values are chosen according to (55). The histogram shows samples obtained over 4 such runs. It can be seen that the reporting probability is likely to be somewhere between 0.2 and 0.5, implying that a large proportion of the dengue cases are never reported to a health center.](image-url)
Particle Gibbs with ancestor sampling (PGAS) is an extension of particle Gibbs (PG) that improves the mixing of the sampler by selecting a new ancestor for the reference trajectory at each point in time, instead of setting the ancestor deterministically (as is done in PG). The ancestor of the reference trajectory $a_t^N$ is sampled according to the ancestor weights

$$w_{t-1|T} \propto w_{t-1}^i \frac{\gamma_T([x_{1:t-1}^i, x_{t:T}^i] | \theta)}{\gamma_{t-1}(x_{1:t-1}^i | \theta)},$$

where $w_{t-1}^i$ is the weight of the possible ancestor trajectory $i$, and $[x_{1:t-1}^i, x_{t:T}^i]$ is a concatenation of the possible ancestor trajectory and the remaining part of the reference trajectory at time $t$. Apart from the sampling of ancestor weights, the conditional sequential Monte Carlo (SMC) part of the PGAS algorithm, given in Algorithm 7, is identical to Algorithm 6.

In spite of being such a small change algorithm-wise, the sampling of new ancestor indices has a significant impact on the mixing of the sampler. Figure 11 shows an example of the improved performance over standard PG offered by PGAS in terms of the autocorrelation of the generated samples for the linear-Gaussian state-space model in (4) with no input signal and unknown noise variances $Q$ and $R$. The prior on the noise variances is chosen to be an inverse-gamma distribution, which is a standard choice. Details on the experiments are in the figure caption.

**Algorithm 7** Conditional SMC with ancestor sampling (all steps for $i = 1, \ldots, N$)

**Input:** Reference trajectory $x_{1:T}^i$, parameters $\theta$, observations $y_{1:T}$.

Draw $x_1^i \sim q(x_1)$ independently, set $x_{1}^N = x_1^i$.

Compute weights $\tilde{w}_1^i = \gamma_1(x_1^i)/q_1(x_1^i)$ and normalize $w_t^i = \tilde{w}_t^i/\sum_{j=1}^N \tilde{w}_t^j$.

For $t = 2 \ldots T$ do

(a) Resample: Draw ancestor indices $\{a_t^i\}_{t=1}^{N-1}$ with probabilities $\{w_{t-1}^i\}_{t=1}^N$ and draw $a_t^N$ with probabilities $w_{t-1|T}^i$ in (56). Set $w_{t-1}^i = 1/N$.

(b) Propagate: Simulate $x_t^i \sim q(x_t | x_{t-1}^{a_t^i})$ and set $x_t^N = x_t^i$. Set $x_{1:t}^i = \{x_{1:t-1}^{a_t^i}, x_t^i\}$.

(c) Weight: Set $\tilde{w}_t^i = w_t^i(x_{1:t}^i)$ and normalize $w_t^i = \tilde{w}_t^i/\sum_{j=1}^N \tilde{w}_t^j$.

**Output:** Draw $j$ with probability $\{w_T^i\}_{i=1}^N$, output new reference trajectory $x_{1:T}' = x_{1:T}^j$.

The improved performance is, perhaps surprisingly, not due to avoidance of path degeneracy. In fact, path degeneracy also occurs for PGAS. However, in PGAS the reference trajectory at each point in time is assigned different ancestors, which causes a collapse of the state trajectory to a trajectory that (with high probability) is different from the current reference trajectory. Because of this small change, PGAS can perform well with a much lower number of particles $N$ in the SMC component than what is required for the corresponding PG sampler.
Figure 11: The autocorrelation function of the generated samples for particle Gibbs (PG) (red) and particle Gibbs with ancestor sampling (PGAS) (blue) for some different number of particles $N$ in the sequential Monte Carlo component. The figure shows the autocorrelation function of the noise covariance $Q$ in the linear-Gaussian state-space model (2a) and (2b) for the one-dimensional case with no input $u$ and all parameters known except for the noise variances $Q$ and $R$. An inverse-gamma prior was used for both noise covariances, and there were $T = 150$ observations. Both samplers were allowed to run for $M = 10000$ iterations for two different choices of the number of particles $N$ for each sampler. The area under the autocorrelation function is related to the Monte Carlo variance of the generated samples. A smaller area corresponds to a smaller Monte Carlo variance and better mixing of the sampler. Thus, the figure indicates that PGAS performs better than standard PG, despite using a lower number of particles.

7 Probabilistic programming

To solve the nonlinear identification problems discussed in this article, both the physically motivated state-space model and the SMC-based inference method must be implemented in some programming language. Implementing these inference methods in a standard programming language—like Matlab, C, or Python—can be a tedious and error-prone process. Moreover, the end result is in many cases a model-specific implementation of the inference method that cannot straightforwardly be reused for another model. Probabilistic programming is a recent tool that has been tailored to facilitate the implementation process by providing the user with a framework for encoding probabilistic models and performing inference in them in an automated way. In a probabilistic program, the model specification and the inference algorithm are separated, which enables the user to focus on the modeling—the inference algorithm is already implemented.

7.1 Representing the model as a program

When specifying a model of a process, some type of modeling “language” is used to communicate the assumptions made about the behavior of that process. In this article, models have
so far mainly been described using the language of mathematics. Figure 1 illustrates another modeling language—the graphical model. Such a model explicitly shows the conditional dependence between the variables. A probabilistic program is yet another modeling language, where the model is encoded as a computer program written in a Turing-complete probabilistic programming language (PPL). Writing the model as a program allows for using programmatic constructs, such as stochastic branching and recursion, in the model formulation. Constructs like these make probabilistic programs a much more expressive class of models than, for example, graphical models. Consider Figure 12, which depicts a probabilistic program that incorporates stochastic branching. Despite its seemingly simple form, this model cannot be formulated as a graphical model using any conventional representation.

1: x ∼ Gaussian(1,2);
2: if (x>2) then
3:   y ∼ Student(x,3,2);
4: else
5:   y ∼ Gaussian(x,2);

**Figure 12:** A probabilistic program with stochastic branching. The program first samples $x$ from a Gaussian distribution with mean value 1 and variance 2. Sampling is indicated by the symbol $\sim$. Depending on the sampled value for $x$, the variable $y$ is either sampled from a Student’s t distribution or a Gaussian distribution.

The PPLs used to encode probabilistic programs are often based on already existing programming languages. However, they differ from standard programming languages in two aspects: They have special constructs for conditional distributions, and they have an inference engine that is in control of the execution of the program. To make the distinction between a standard computer program and a probabilistic program clearer, consider a standard computer program. It is provided with some inputs and is then executed deterministically, step by step, to produce some output. A probabilistic program instead takes observations as inputs and is executed deterministically only until it reaches a so-called *checkpoint*, where the execution is paused. At the checkpoint, the inference engine takes the current state of the program, manipulates it as specified in the program, returns the updated state and resumes deterministic execution of the program. The checkpoints, where all randomness is introduced into the program, can be of different types. A typical setup is to have at least two basic checkpoints: *sample*, which creates a random variable and can trigger sampling of that variable; and *observe*, which triggers conditioning on observed data [79]. Figure 13 shows a probabilistic program of a linear-Gaussian state-space model that incorporates both of these checkpoints. A more complex example is provided in “Example: Dengue fever in the probabilistic programming language Birch”, where the model describing the spread of dengue fever in “Example: Dengue fever” is implemented as a probabilistic program in the PPL Birch [80].

### 7.2 Inference in a probabilistic program

When a probabilistic program is executed, it produces a set of random variables. Each time the program is run, it might encounter a different set of variables due to the randomness inherent in the program from entering different stochastic branches. For instance, the probabilistic program in Figure 12 might sample $x = 1$ the first time it is run, and $y$ is consequently
1: $y[1] = -0.45$;  
2: $y[2] = 3.56$;  
3: $x[1] \sim \text{Gaussian}(0,2)$;  
4: $\text{observe}(\text{Gaussian}(x[1],2), y[1])$;  
5: $x[2] \sim \text{Gaussian}(2x[1],1)$;  
6: $\text{observe}(\text{Gaussian}(x[2],2), y[2])$;  
7: $\text{print}(x[1])$;  

Figure 13: A probabilistic program describing a linear-Gaussian state-space model with two observations, $y[1]$ and $y[2]$, and two states, $x[1]$ and $x[2]$. This program incorporates two types of checkpoints. A sample checkpoint is implemented using the symbol $\sim$, followed by the distribution. An observe checkpoint is denoted $\text{observe}(\text{distribution}, \text{value})$.

sampled from a Gaussian distribution. During the second run, it might sample $x = 4$, and $y$ is in this case sampled from a Student’s t distribution. The randomness present in the probabilistic program makes the design of inference methods more complicated for probabilistic programs than for other types of models. For instance, different executions of the same program may encounter the observations in different orders. For an inference method like SMC, it is not immediately clear how to compute the weights when such a situation occurs. Despite these difficulties, many inference algorithms have been adapted for use in probabilistic programs. To provide an idea of how SMC can be implemented in a PPL, “Background: The bootstrap particle filter in a probabilistic programming language” describes a PPL version of the bootstrap particle filter. Some existing PPLs that implement versions of SMC are Anglican [81], Birch [80], Figaro [82], Gen [83], LibBi [84], Pyro [85], Turing [86], Venture [87], and WebPPL [88]. Examples of gplppl that are based on other inference methods—such as variational inference, Hamiltonian Monte Carlo and Gibbs sampling—are Church [89], Edward [90], Infer.NET [91], JAGS [92], Stan [93], and WinBUGS [94].

An ideal PPL should implement a variety of different inference methods. It should also support the user by choosing an appropriate inference method for the model automatically and tune it to obtain the best possible performance. The choice of inference method can be based on the structure of the model [80]. For example, if the model is a (physically motivated) nonlinear state-space model like those in this article, the PPL might suggest to use SMC, provided that SMC has been implemented. On the other hand, if the model is linear-Gaussian, the PPL should suggest using a Kalman filter/smother. The PPLs of today are not yet at this level of automation, where they can select a suitable inference algorithm among a large selection of algorithms and then tune it for an arbitrary model. Nonetheless, many of the existing PPLs implement more than one inference algorithm. Although the user must specify which inference method to use, just being able to try several different methods without having to reimplement the model is a huge benefit. Some PPLs of today even provide analytic improvements like marginalization for variance reduction [95, 96, 97] automatically for models where such relations exists. There are also some PPLs that can perform adaptive tuning of the inference method without any intervention from the user [98].
### Example: Dengue fever in the probabilistic programming language Birch

This section introduces the reader to a probabilistic programming language (PPL) that uses sequential Monte Carlo (SMC) as the base method for inference, and illustrates how a physically motivated state-space model can be implemented as a probabilistic program in that PPL.

**Birch**

Birch \[80, 99\] is a Turing-complete, open-source PPL that compiles to C++. The model and the inference methods are both encoded in the Birch language. However, they are implemented separate from each other, so the user need only specify the model and decide on which pre-implemented inference algorithm to use. Birch is an object-oriented programming language, which allows for using classes to encode common model structures. By using a specific model class, the user can let the inference engine know something about the structure of the model, so that it can choose a suitable inference method. SMC is the main inference method in Birch. Markov chain Monte Carlo methods are also implemented, and by combining these with SMC, particle Markov chain Monte Carlo methods are also supported \[95, 25\]. Birch utilizes three types of checkpoints to perform inference: the \textit{assume} checkpoint, which initializes a random variable $X$ to have some distribution $p$; the \textit{observe} checkpoint, which conditions a random variable $X$ on the value of some other variable $Y$ being $y$; and the \textit{value} checkpoint, which triggers sampling of a random variable $X$ that was previously only assumed.

**The dengue fever model in Birch**

The results for particle Gibbs (PG) presented in “Example: Particle Gibbs for identification of dengue fever parameters” for the model described in “Example: Dengue fever” were, in fact, obtained using probabilistic programming in Birch. To replicate these results, the user must only implement the model as a probabilistic program written in Birch code, since the PG algorithm is already supported in Birch through implementations of Gibbs sampling and conditional SMC. If the object-oriented nature of Birch is utilized, a probabilistic program for the dengue model can be based on Birch’s special model class \textit{MarkovModel}. The \textit{MarkovModel} class requires three internal functions to be defined: \textit{parameter}, which defines the parameters of the probabilistic program; \textit{initial}, which initializes the probabilistic program; and \textit{transition}, which specifies the transitions and observations in the probabilistic program. The program for the dengue model inherits from \textit{MarkovModel} and must also implement these functions. Here, a simplified version of only the parts of the transition function that illustrates the connection to the model in “Example: Dengue fever” is presented, in the interest of space and clarity. The complete code for the Birch implementation of the dengue model is available in the Github repository \textit{VectorBorneDisease} \[100\].

The probabilistic program is implemented in three levels. At the lowest level is the implementation of a basic susceptible-exposed-infectious-recovered (SEIR) model, shown in Birch code \[1\] which can be used to describe the transitions for both humans...
and mosquitoes. The symbol \( \sim \) indicates simulation from the distribution to the right of the symbol, implying that this is a value checkpoint in the program. The symbol \( \leftarrow \) indicates the standard assignment of a value to a variable. The first three rows in the program samples the number of newly exposed, infectious, and recovered, respectively, according to (13). Similarly, row 4-7 update the number of susceptible, exposed, infectious, and recovered according to (13).

### Birch code 1 SEIRModel

```java
final class SEIRModel < MarkovModel<SEIRParameter,SEIRState> {
    function transition(x':SEIRState,x:SEIRState,θ:SEIRParameter ,ne:Integer,ni:Integer,ne:Integer) {
        x'.Δe \sim \text{Binomial}(ne, θ.λ);
        x'.Δi \sim \text{Binomial}(ni, θ.δ);
        x'.Δr \sim \text{Binomial}(nr, θ.γ);
        x'.s <- x.s - x'.Δe;
        x'.e <- x.e + x'.Δe - x'.Δi;
        x'.i <- x.i + x'.Δi - x'.Δr;
        x'.r <- x.r + x'.Δr;
        x'.n <- x'.s + x'.e + x'.i + x'.r;
    }
}
```

The next level implements the interaction between the human and mosquito SEIR models, see Birch code 2. It first defines one SEIR model (Birch code 1) for humans and mosquitoes, respectively. It then samples the number of susceptible humans and mosquitoes that have been exposed to the virus (bitten) according to (14). The last two rows call the transition function for each of the SEIR models as implemented in Birch code 1.

### Birch code 2 VBDModel

```java
final class VBDModel < MarkovModel<VBDParameter,VBDState> {
    h:SEIRModel;
    m:SEIRModel;
    function transition(x':VBDState,x:VBDState,θ:VBDParameter) {
        ne_h:Integer;
        ne_m:Integer;
        ne_h \sim \text{Binomial}(x.h.s, 1.0 - \exp(-x.m.i/\text{Real}(x.h.n)));
        ne_m \sim \text{Binomial}(x.m.s, 1.0 - \exp(-x.h.i/\text{Real}(x.h.n)));
        h.transition(x'.h, x.h, θ.h, ne_h, x.h.e, x.h.i);
        m.transition(x'.m, x.m, θ.m, ne_m, x.m.e, x.m.i);
    }
}
```

The highest level, shown in Birch code 3, implements the disease- and outbreak-specific parameters (not shown here) and the observation model. First, the coupled SEIR models are defined, and the transition function in Birch code 2 is called. Then, the total number of newly infectious humans since the last observation is computed and stored in the variable \( x'.z \). Finally, if there is an observation \( x'.y \), the number
of newly infectious humans is conditioned on that observation according to \(15\). The
observe checkpoint is indicated with the symbol \(\sim\). Note that there is no example
of an assume checkpoint, indicated by the symbol \(\sim\), in Birch code [1] [2] and [3]. The
assume checkpoint is typically used to initialize the parameters to have a particular
distribution. For an example, see the complete code at [100].

\[
\text{Birch code 3 YapDengueModel}
\]

```java
final class YapDengueModel < MarkovModel<YapDengueParameter,YapDengueState> {
  v: VBDModel;

  function transition(x':YapDengueState,x:YapDengueState ,θ:YapDengueParameter ) {
    v.transition(x', x, θ);
    x'.z <- x.z + x'.h.Δi;
    if x'.y? {
      x'.y! \(\sim\) Binomial(x'.z, θ.ρ);
      x'.z <- 0;
    }
  }
}
```

\[
\text{Background: The bootstrap particle filter in a probabilistic program-
ming language}
\]

Here, a high-level description of how inference using a bootstrap particle filter can work
in a probabilistic programming language (PPL) is presented. This description highlights
the actions that must be performed by the inference engine. However, it does not go
into details on how these are implemented, since that is language-specific and beyond
the scope of this article. The interested reader is referred to the documentation for one
of the PPL implementing sequential Monte Carlo (SMC) mentioned in Section 7.2.

Recall from “Bootstrap filter” that the bootstrap particle filter is a special case of
SMC, where the state transition probabilities \(p(x_t \mid x_{t-1})\) are used as proposal distributions.
In short, the bootstrap filter estimates the filtering distributions \(p(x_t \mid y_1:t)\) using
a set of weighted particles that are updated at each time step according to step (a)-(c) in Algorithm 2. In PPLs, a particle corresponds to an execution of a probabilistic program.
The inference engine keeps track of \(N\) parallel executions of the program
that together represent the particle set. Analogous to the standard bootstrap particle
filter, where each particle has a weight, each execution of the probabilistic program has
an associated weight that describes how likely the observations are, given the random
choices made by that particular execution.

Each of the \(N\) different executions of the program is executed deterministically from
the first line of the program until a checkpoint is encountered. When an execution of
the program reaches a sample checkpoint, the inference engine simply samples from the
specified distribution—the proposal is just the state transition for a bootstrap particle
filter. This corresponds to step (b), propagate, in Algorithm 2. The execution of the
program then continues deterministically until the next checkpoint is encountered. At
an *observe* checkpoint, the execution of the program is paused while the inference engine performs the required actions. To implement a bootstrap particle filter, the inference engine must perform two actions. First, it updates the weight of the execution by multiplying it with the likelihood of the new observation. This corresponds to step (c), *weight*, in Algorithm 2. Once all executions (particles) have updated their weights, resampling takes place. This corresponds to step (a), *resample*, in Algorithm 2. The resampling leads to termination of some executions and duplication (or cloning) of other executions. After resampling, all weights are reset. The surviving executions (and their potential clones) are then resumed.

To make the above description a bit more concrete, consider the probabilistic program in Figure 13. Inference in this program using a bootstrap particle filter proceeds in the following way. First $N$ parallel executions are initiated, and the observations are stored as $y[1]$ and $y[2]$ (lines 1 and 2). When an execution reaches the *sample* checkpoint on line 3, a value for $x[1]$ is drawn from the specified Gaussian distribution, and execution of the program is resumed. Next, an *observe* checkpoint is encountered on line 4. The execution is paused and the inference engine updates the weight for the execution by computing the likelihood of the observation $y[1]$. When weights have been computed for all $N$ parallel executions, resampling based on the weights is performed. The surviving executions (and their eventual clones) are then resumed. On line 5, the program encounters a new *sample* checkpoint, which is handled in the same way as line 3. Similarly, on line 6, a new *observation* checkpoint is encountered where the same actions as for line 4 are performed before resuming the executions of the program.

At a first glance, it can seem like there is not much difference between a probabilistic program and a standard implementation of the particle filter. However, note that the inference engine typically implements more than one inference method that can easily be applied to the same model. Additionally, the execution of the probabilistic program is stochastic and might therefore encounter different variables and checkpoints in different executions. In particular, this can occur when stochastic branching (see Figure 12 for an example) is present in the probabilistic program. Different executions of the program can then pause for weighting and resampling while being in different branches that may have a different number of observe checkpoints—the executions are not aligned. This is something that cannot be handled by an implementation of the bootstrap particle filter in a standard programming language. It is problematic also for many PPLs, in the sense that many PPLs do not handle the alignment problem explicitly. However, some work on how to automatically align observe checkpoints for SMC is available in [101], and it is handled in a more manual fashion by processing a complete branch before resampling in Birch [80].

8 Future directions

There are several interesting avenues for future work, a few of these are outlined here. When it comes to modeling, an interesting direction is to develop constructions that allow for combining prior knowledge about the process with the highly successful, flexible, black-box, deep neural networks and Gaussian processes. This would create a model where the existing in-
formation about the phenomenon under study is combined with the data in such a way that new knowledge can be gained. A key question is how to strike the right balance between prior knowledge and new knowledge via the data. Concrete examples of constructions of this kind are available for Gaussian processes [6] and neural networks [7] when it comes to the incorporation of linear operator constraints. It remains a challenge to formulate models of this kind that are suitable for dynamical systems.

A subtle, but vitally important aspect in the modeling and identification techniques outlined in this article is that the model structure is assumed given. The modeling step can require significant user input, and for many new applications it is often tedious to construct an appropriate structure. It is interesting that many of the physically inspired models rely on relatively simple combinations of mathematical functions, in conjunction with a calculus for combining them to form models that describe the observed phenomena. A provocative idea is to encode these atomic functions and the allowed calculus, so that the structure is learned from the data (which covers the parameter estimation idea as a special case). Promising research in this direction can be found in [102].

When it comes to the learning and inference algorithms, it remains a major challenge to handle high-dimensional systems using SMC. Interesting developments in this direction include [103, 104, 105].

The validity of the SMC framework relies on an assumed forgetting in the dynamical system, which prevents errors from accumulating over time. Therefore, these methods tend to struggle when there are strong and long-ranging temporal dependencies in the data (and the model). Handling such cases is largely an open problem, although the use of “twisting” as discussed above can provide part of a solution. See [44, 106] for developments in this direction. Another possibility is to learn an efficient SMC algorithm conditionally on all the available data using variational inference; see [107, 108, 109].

For modeling in general, the problem of using the wrong model class remains challenging, since the assumptions made at the start have a strong impact on the end result. There are currently very few methods available that perform automatic validation of modeling assumptions, making this a highly relevant avenue for future research. The work in [110], where an SMC-based method is developed for model validation, provides a possible starting point. The idea there is to make use of the generative capabilities of the model to gauge its ability to generate data that is similar to the observed data.

Finally, learning from very long observation sequences is computationally challenging, since most algorithms for identifying nonlinear state-space models require processing all observations between each parameter update, both for direct likelihood optimization and data augmentation methods. Furthermore, some (but not all) SMC-based methods require the number of particles to grow with the number of time steps, making this computational issue even more pronounced. Methods based on “mini-batching” has also been proposed for state-space identification [111]. However, the temporal dependencies makes this more challenging than when learning from independent observations.
Acknowledgments

This research was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by Knut and Alice Wallenberg Foundation; the projects ASSEMBLE (contract number: RIT15-0012) and Probabilistic Modeling and Inference for Machine Learning (contract number: ICA16-0015), funded by the Swedish Foundation for Strategic Research (SSF); the projects NewLEADS - New Directions in Learning Dynamical Systems (contract number: 621-2016-06079), Handling Uncertainty in Machine Learning Systems (contract number: 2020-04122), and Learning flexible models for nonlinear dynamics (contract number: 2017-03807), funded by the Swedish Research Council; ELLIIT; and Kjell och Märta Beijer Foundation.
References

[1] Lennart Ljung. System identification, Theory for the user. second. System sciences series. Upper Saddle River, NJ, USA: Prentice Hall, 1999.

[2] Herbert Goldstein. Classical Mechanics. Second. Addison-Wesley, 1980.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT Press, 2016.

[4] Lennart Ljung, Carl Andersson, Koen Tiels, and Thomas B. Schön. “Deep learning and system identification”. In: Proceedings of the IFAC World Congress. Berlin, 2020.

[5] Ryan Turner, Marc P. Deisenroth, and Carl E. Rasmussen. “System identification in Gaussian Process Dynamical Systems”. In: Proceedings of the Neural Information Processing Systems (NIPS) Conference. Whistler, Canada, 2009.

[6] Carl Jidling, Niklas Wahlström, Adrian Wills, and Thomas B. Schön. “Linearly constrained Gaussian processes”. In: Advances in Neural Information Processing Systems (NIPS). Long Beach, CA, USA, 2017.

[7] Johannes Hendriks, Carl Jidling, Adrian Wills, and Thomas B. Schön. Linearly Constrained Neural Networks. Feb. 5, 2020. arXiv: 2002.01600 [stat.ML].

[8] Donald B. Rubin. “Bayesianly justifiable and relevant frequency calculations for the applied statistician”. In: The Annals of Statistics 12.4 (1984), pp. 1151–1172.

[9] Ronald A. Fisher. “On an absolute criterion for fitting frequency curves”. In: Messenger of Mathematics 41 (1912), pp. 155–160.

[10] Ronald A. Fisher. “On the Mathematical Foundations of Theoretical Statistics”. In: Philosophical Transactions of the Royal Society Series A 222 (1922), pp. 309–368.

[11] Thomas Bayes. “An Essay Towards Solving a Problem in the Doctrine of Chances”. In: The Philosophical Transactions 53 (1763), pp. 370–418.

[12] Vidyadhar P. Godambe. Estimating functions. Oxford University Press, 1991.

[13] Robert Bassett and Julio Deride. “Maximum a posteriori estimators as a limit of Bayes estimators”. In: Mathematical Programming 174.1 (2019), pp. 129–144.

[14] Nikolas Kantas, Arnaud Doucet, Sumeetpal S. Singh, Jan M. Maciejowski, and Nicolas Chopin. “On particle methods for parameter estimation in state-space models”. In: Statistical Science 30.3 (2015), pp. 328–351.

[15] Neil J. Gordon, David J. Salmond, and Adrian F. M. Smith. “Novel approach to nonlinear/non-Gaussian Bayesian state estimation”. In: IEE Proceedings on Radar and Signal Processing. Vol. 140. 1993, pp. 107–113.

[16] Pierre Del Moral. “Non-linear filtering: Interacting particle solution”. In: Markov processes and related fields 2.4 (1996), pp. 555–580.

[17] David D. Nolte. “The tangled tale of phase space”. In: Physics today 63.4 (2010), pp. 33–38.

[18] Andreas Lindholm and Fredrik Lindsten. Learning dynamical systems with particle stochastic approximation EM. June 25, 2018. arXiv: 1806.09548v2 [stat.CO]
[19] Maarten Schoukens, Per Mattsson, Torbjörn Wigren, and Jean-Philippe Noël. “Cascaded tanks benchmark combining soft and hard nonlinearities”. In: 20th World Congress of the International Federation of Automatic Control 1 (2017), pp. 448–453.

[20] Tim J. Rogers, Geoffrey R. Holmes, Elizabeth J. Cross, and Keith Worden. “On a Grey Box Modelling Framework for Nonlinear System Identification”. In: Special topics in structural dynamics. 2017, pp. 167–178.

[21] Jeffrey D. Stanaway, Donald S. Shepard, Eduardo A. Undurraga, Yara A. Halasa, Luc E. Coffeng, Oliver J. Brady, Simon I. Hay, Neeraj Bedi, Isabela M. Bensenor, Carlos A. Castañeda-Orjuela, Ting-Wu Chuang, Katherine B. Gibney, Ziad A. Memish, Anwar Rafay, Kingsley N. Ukwaja, Naohiro Yonemoto, and Christopher J. L. Murray. “The global burden of dengue: An analysis from the Global Burden of Disease Study 2013.” In: The Lancet. Infectious diseases 16.6 (2016), pp. 712–723.

[22] World Health Organization (WHO). Dengue and severe dengue. https://www.who.int/news-room/fact-sheets/detail/dengue-and-severe-dengue. Accessed: 2020-10-23. 2020.

[23] Sebastian Funk, Adam J. Kucharski, Anton Camacho, Rosalind M. Eggo, Laith Yakob, Lawrence M. Murray, and W. John Edmunds. “Comparative analysis of dengue and Zika outbreaks reveals differences by setting and virus”. In: PLOS Neglected Tropical Diseases 10.12 (2016), pp. 1–16.

[24] Thomas G. Kurtz. “Solutions of ordinary differential equations as limits of pure jump Markov processes”. In: Journal of Applied Probability 7.1 (1970), pp. 49–58.

[25] Lawrence M. Murray, Daniel Lundén, Jan Kudlicka, David Broman, and Thomas B. Schön. “Delayed sampling and automatic Rao-Blackwellization of probabilistic programs”. In: Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics. Playa Blanca, Lanzarote, Canary Islands, 2018, pp. 1037–1046.

[26] Lennart Ljung and Torkel Glad. Control Theory – Multivariable and Nonlinear Methods. London: Taylor and Francis, 2000.

[27] Torsten Söderström and Petre Stoica. System identification. Systems and Control Engineering. Prentice Hall, 1989.

[28] Václav Peterka. “Bayesian System Identification”. In: Automatica 17.1 (1981), pp. 41–53.

[29] Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. “Particle Markov chain Monte Carlo methods”. In: Journal of the Royal Statistical Society. Series B (Methodological) 72.2 (2010), pp. 1–33.

[30] Thomas B. Schön, Andreas Svensson, Lawrence M. Murray, and Fredrik Lindsten. “Probabilistic learning of nonlinear dynamical systems using sequential Monte Carlo”. In: Mechanical Systems and Signal Processing (MSSP) 104 (2018), pp. 866–883.

[31] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. “Equations of state calculations by fast computing machine”. In: Journal of Chemical Physics 21.6 (1953), pp. 1087–1092.

[32] Wilfred K. Hastings. “Monte Carlo simulation methods using Markov Chains and their applications”. In: Biometrika 57 (1970), pp. 97–109.
[33] Arthur P. Dempster, Nan Laird, and Donald B. Rubin. “Maximum likelihood from incomplete data via the EM algorithm”. In: *Journal of the Royal Statistical Society, Series B* 39.1 (1977), pp. 1–38.

[34] Stuart Geman and Donald D. Geman. “Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6.6 (1984), pp. 731–741.

[35] Genshiro Kitagawa. “A Monte Carlo filtering and smoothing method for non-Gaussian nonlinear state space models”. In: *Proceedings of the 2nd US-Japan joint Seminar on Statistical Time Series Analysis*. 1993, pp. 110–131.

[36] Leland Stewart and Perry McCarty Jr. “The use of Bayesian belief networks to fuse continuous and discrete information for target recognition and discrete information for target recognition, tracking, and situation assessment”. In: *Proceedings of SPIE Signal Processing, Sensor Fusion and Target Recognition*. Vol. 1699. 1992, pp. 177–185.

[37] Pierre Del Moral. “Feynman-Kac formulae”. In: *Feynman-Kac Formulae*. Springer, 2004, pp. 47–93.

[38] Michael K. Pitt, Ralph dos Santos Silva, Paolo Giordani, and Robert Kohn. “On some properties of Markov chain Monte Carlo simulation methods based on the particle filter”. In: *Journal of Econometrics* 171.2 (2012), pp. 134–151.

[39] Nicolas Chopin and Omiros Papaspiliopoulos. *An introduction to Sequential Monte Carlo*. Springer, 2020.

[40] Simo Särkkä. *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.

[41] Olivier Cappé, Eric Moulines, and Tobias Rydén. *Inference in hidden Markov models*. Berlin, Germany: Springer, 2005.

[42] Nick Whiteley and Anthony Lee. “Twisted particle filters”. In: *The Annals of Statistics* 42.1 (2014), pp. 115–141.

[43] Jeremy Heng, Adrian N Bishop, George Deligiannidis, Arnaud Doucet, et al. “Controlled sequential Monte Carlo”. In: *Annals of Statistics* 48.5 (2020), pp. 2904–2929.

[44] Pieralberto Guarniero, Adam M. Johansen, and Anthony Lee. “The iterated auxiliary particle filter”. In: *Journal of the American Statistical Association* 112.520 (2017), pp. 1636–1647.

[45] Simo Särkkä. *Bayesian filtering and smoothing*. Vol. 3. Cambridge University Press, 2013.

[46] Christopher Nemeth, Paul Fearnhead, and Lyudmila Mihaylova. “Particle Approximations of the Score and Observed Information Matrix for Parameter Estimation in State–Space Models With Linear Computational Cost”. In: *Journal of Computational and Graphical Statistics* 25.4 (2016), pp. 1138–1157.

[47] Herbert Robbins and Sutton Monro. “A Stochastic Approximation Method”. In: *Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407.

[48] Jack Kiefer and Jacob Wolfowitz. “Stochastic estimation of the maximum of a regression function”. In: *The Annals of Mathematical Statistics* 23.3 (1952), pp. 462–466.
[49] Lennart Ljung. “Analysis of recursive stochastic algorithms”. In: *IEEE transactions on automatic control* 22.4 (1977), pp. 551–575.

[50] Lennart Ljung. “Strong convergence of a stochastic approximation algorithm”. In: *The Annals of Statistics* (1978), pp. 680–696.

[51] Lennart Ljung. “Asymptotic Behavior of the Extended Kalman Filter as a Parameter Estimator for Linear Systems”. In: *IEEE Transactions on Automatic Control* AC–24.1 (1979), pp. 36–50.

[52] Lennart Ljung and Torsten Söderström. *Theory and Practice of Recursive Identification*. Cambridge, MA, USA: The MIT Press, 1983.

[53] Graham C. Goodwin, Peter J. Ramadge, and Peter E. Caines. “Discrete time stochastic adaptive control”. In: *SIAM Journal on Control and Optimization* 19.6 (1981), pp. 829–853.

[54] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-dynamic programming*. Vol. 5. Athena Scientific Belmont, MA, 1996.

[55] James C. Spall. *Introduction to stochastic search and optimization: estimation, simulation, and control*. John Wiley & Sons, 2005.

[56] Lennart Ljung, Georg Pflug, and Harro Walk. *Stochastic approximation and optimization of random systems*. Birkhäuser, 2012.

[57] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. “Optimization methods for large-scale machine learning”. In: *SIAM Review* 60.2 (2018), pp. 223–311.

[58] Hilal Asi and John C. Duchi. “Stochastic (approximate) proximal point methods: convergence, optimality, and adaptivity”. In: *SIAM Journal on Optimization* 29.3 (2019), pp. 2257–2290.

[59] Eric Moulines and Francis Bach. “Non-asymptotic analysis of stochastic approximation algorithms for machine learning”. In: *Advances in Neural Information Processing Systems (NIPS)*. Granada, Spain, 2011.

[60] Larry Armijo. “Minimization of functions having Lipschitz continuous first partial derivatives”. In: *Pacific Journal of Mathematics* 16.1 (1966), pp. 1–3.

[61] Philip Wolfe. “Convergence conditions for ascent methods”. In: *SIAM Review* 11.2 (1969), pp. 226–235.

[62] Philip Wolfe. “Convergence conditions for ascent methods II: some corrections”. In: *SIAM Review* 13.2 (1971), pp. 185–188.

[63] Adrian G. Wills and Thomas B. Schön. “Stochastic quasi-Newton with line-search regularisation”. In: *Automatica* 127 (2021), p. 109503.

[64] Adrian Wills, Thomas B. Schön, and Carl Jidling. “A fast quasi-Newton-type method for large-scale stochastic optimisation”. In: *Proceedings of the IFAC World Congress*. Online, 2020.

[65] George Casella and Christian P. Robert. *Monte Carlo statistical methods*. 2004.

[66] Christophe Andrieu and Gareth O. Roberts. “The pseudo-marginal approach for efficient Monte Carlo computations”. In: *The Annals of Statistics* 37.2 (2009), pp. 697–725.
[67] Steven M. Kay. *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory*. Upper Saddle River, NJ, USA: Prentice Hall, 1993.

[68] Luke Tierney. “Markov chains for Exploring Posterior Distributions”. In: *The Annals of Statistics* 22.4 (1994), pp. 1701–1728.

[69] Stuart Gibson and Brett Ninness. “Robust maximum-likelihood estimation of multivariable dynamic systems”. In: *Automatica* 41.10 (2005), pp. 1667–1682.

[70] Johan Olsson, Olivier Cappé, Randal Douc, and Eric Moulines. “Sequential Monte Carlo smoothing with application to parameter estimation in nonlinear state-space models”. In: *Bernoulli* 14.1 (2008), pp. 155–179.

[71] Thomas B. Schön, Adrian Wills, and Brett Ninness. “System Identification of Nonlinear State-Space Models”. In: *Automatica* 47.1 (Jan. 2011), pp. 39–49.

[72] Fredrik Lindsten, Randal Douc, and Eric Moulines. “Uniform ergodicity of the Particle Gibbs sampler”. In: *Scandinavian Journal of Statistics* 42.3 (2015), pp. 775–797.

[73] Nick Whiteley. “Discussion on particle Markov chain Monte Carlo methods”. In: *Journal of the Royal Statistical Society: Series B* 72.3 (2010), pp. 306–307.

[74] Anthony Lee, Sumeetpal S. Singh, and Matti Vihola. “Coupled conditional backward sampling particle filter”. In: *Annals of Statistics* 48.5 (2020), pp. 3066–3089.

[75] Fredrik Lindsten, Michael I. Jordan, and Thomas B. Schön. “Particle Gibbs with ancestor sampling”. In: *Journal of Machine Learning Research (JMLR)* 15 (2014), pp. 2145–2184.

[76] Sumeetpal S. Singh, Fredrik Lindsten, and Eric Moulines. “Blocking strategies and stability of particle Gibbs samplers”. In: *Biometrika* 104.4 (2017), pp. 953–969.

[77] Bernard Delyon, Marc Lavielle, and Eric Moulines. “Convergence of a stochastic approximation version of the EM algorithm”. In: *The Annals of Statistics* 27.1 (1999), pp. 94–128.

[78] Fredrik Lindsten. “An efficient stochastic approximation EM algorithm using conditional particle filters”. In: *Proceedings of the 38th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Vancouver, Canada, 2013.

[79] David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank Wood. “Design and implementation of probabilistic programming language Anglican”. In: *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages*. 2016.

[80] Lawrence M. Murray and Thomas B. Schön. “Automated learning with a probabilistic programming language: Birch”. In: *Annual Reviews in Control* 46 (2018), pp. 29–43.

[81] Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. “A new approach to probabilistic programming inference”. In: *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*. 2014, pp. 1024–1032.

[82] Avi Pfeffer. *Practical probabilistic programming*. Manning, 2016.
[83] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. “Gen: A General-purpose Probabilistic Programming System with Programmable Inference”. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. Phoenix, AZ, USA, 2019, pp. 221–236.

[84] Lawrence M. Murray. “Bayesian State-Space Modelling on High-Performance Hardware Using LibBi”. In: *Journal of Statistical Software, Articles* 67.10 (2015), pp. 1–36.

[85] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. “Pyro: Deep universal probabilistic programming”. In: *Journal of Machine Learning Research* 20.28 (2019), pp. 1–6.

[86] Hong Ge, Kai Xu, and Zoubin Ghahramani. “Turing: A language for flexible probabilistic inference”. In: *International Conference on Artificial Intelligence and Statistics*. 2018, pp. 1682–1690.

[87] Vikash Mansinghka, Daniel Selsam, and Yura Perov. *Venture: a higher-order probabilistic programming platform with programmable inference*. Apr. 1, 2014. arXiv: 1404.0099 [cs.AI].

[88] Noah D. Goodman and Andreas Stuhlmüller. *The Design and Implementation of Probabilistic Programming Languages*. http://dippl.org. Accessed: 2020-3-23. 2014.

[89] Noah Goodman, Vikash Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. “Church: a language for generative models”. In: *Uncertainty in Artificial Intelligence*. Vol. 2008. Jan. 2008, pp. 220–229.

[90] Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja Rudolph, Dawen Liang, and David M. Blei. *Edward: A library for probabilistic modeling, inference, and criticism*. Oct. 31, 2016. arXiv: 1610.09787 [stat.CO].

[91] Tom. Minka, John M. Winn, John P. Guiver, Yordan Zaykov, Dany Fabian, and John Bronskill. *Infer.NET 0.3*. Microsoft Research Cambridge. http://dotnet.github.io/infer. 2018.

[92] Martyn Plummer. “JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling”. In: *Proceedings of the 3rd international workshop on distributed statistical computing*. 125.10. 2003, pp. 1–10.

[93] Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. “Stan: A Probabilistic Programming Language”. In: *Journal of Statistical Software, Articles* 76.1 (2017), pp. 1–32.

[94] David J. Lunn, Andrew Thomas, Nicky Best, and David Spiegelhalter. “WinBUGS - A Bayesian modelling framework: Concepts, structure, and extensibility”. In: *Statistics and Computing* 10 (2000), pp. 325–337.

[95] Anna Wigren, Riccardo Sven Risuleo, Lawrence M. Murray, and Fredrik Lindsten. “Parameter elimination in particle Gibbs sampling”. In: *Advances in Neural Information Processing Systems*. Vancouver, Canada, 2019.
[96] Fritz Obermeyer, Eli Bingham, Martin Jankowiak, Du Phan, and Jonathan P. Chen. *Functional Tensors for Probabilistic Programming*. Oct. 23, 2019. arXiv: [1910.10775 [stat.ML]].

[97] Matthew D. Hoffman, Matthew J Johnson, and Dustin Tran. “Autoconj: Recognizing and Exploiting Conjugacy Without a Domain-Specific Language”. In: *Advances in Neural Information Processing Systems*. 2018.

[98] Matthew D Hoffman and Andrew Gelman. “The no-u-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo.” In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1593–1623.

[99] Lawrence M. Murray. *Birch*. [https://www.birch.sh/](https://www.birch.sh/) 2020.

[100] Lawrence M. Murray. *Vector-borne disease package*. [https://github.com/lawmurray/Birch/tree/master/examples/VectorBorneDisease](https://github.com/lawmurray/Birch/tree/master/examples/VectorBorneDisease) Accessed: 2020-10-23. 2020.

[101] Daniel Lundén, David Broman, Fredrik Ronquist, and Lawrence M. Murray. *Automatic alignment of sequential Monte Carlo inference in higher-order probabilistic programs*. Dec. 18, 2018. arXiv: [1812.07439 [cs.PL]].

[102] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. “Discovering governing equations from data by sparse identification of nonlinear dynamical systems”. In: *Proceedings of the National Academy of Sciences* 113.15 (2016), pp. 3932–3937.

[103] Patrick Rebeschini, Ramon Van Handel, et al. “Can local particle filters beat the curse of dimensionality?” In: *The Annals of Applied Probability* 25.5 (2015), pp. 2809–2866.

[104] Christian A. Naesseth, Fredrik Lindsten, and Thomas B. Schön. “High-dimensional Filtering using Nested Sequential Monte Carlo”. In: *IEEE Transactions on Signal Processing* 67.16 (2019), pp. 4177–4188.

[105] Tianshi Chen, Thomas B. Schön, Henrik Ohlsson, and Lennart Ljung. “Decentralized Particle Filter with Arbitrary State Decomposition”. In: *IEEE Transactions on Signal Processing* (2011).

[106] Fredrik Lindsten, Jouni Helske, and Matti Vihola. “Graphical model inference: Sequential Monte Carlo meets deterministic approximations”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 8190–8200.

[107] Christian Naesseth, Scott Linderman, Rajesh Ranganath, and David Blei. “Variational sequential Monte Carlo”. In: *Proceedings of Machine Learning Research*. Vol. 84. Playa Blanca, Lanzarote, Canary Islands, 2018, pp. 968–977.

[108] Tuan Anh Le, Maximilian Igl, Tom Rainforth, Tom Jin, and Frank Wood. *Auto-encoding sequential Monte Carlo*. May 29, 2017. arXiv: [1705.10306 [stat.ML]].

[109] Chris J. Maddison, John Lawson, George Tucker, Nicolas Heess, Mohammad Norouzi, Andriy Mnih, Arnaud Doucet, and Yee-Whye Teh. “Filtering variational objectives”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 6573–6583.

[110] Andreas Lindholm, Dave Zachariah, Petre Stoica, and Thomas B. Schön. “Data consistency approach to model validation”. In: *IEEE Access*. Vol. 7. 2019, pp. 59788–59796.

[111] Christopher Aicher, Yi-An Ma, Nicholas J. Foti, and Emily B. Fox. “Stochastic gradient MCMC for state space models”. In: *SIAM Journal on Mathematics of Data Science* 1.3 (2019), pp. 555–587.