Journal of Machine Learning Research 23 (2022) 1-39 Submitted 2/22; Revised 7/22; Published 8/22

De-Sequentialized Monte Carlo: a parallel-in-time particle smoother

Adrien Corenflos ADRIEN.CORENFLOS@AALTO.FI
Department of Electrical Engineering and Automation, Aalto University

Nicolas Chopin NICOLAS.CHOPIN@ENSAE.FR
ENSAE, Institut Polytechnique de Paris

Simo Särkkä SIMO.SARKKA@AALTO.FI
Department of Electrical Engineering and Automation, Aalto University

Editor: Anthony Lee

Abstract

Particle smoothers are SMC (Sequential Monte Carlo) algorithms designed to approximate the joint distribution of the states given observations from a state-space model. We propose dSMC (de-Sequentialized Monte Carlo), a new particle smoother that is able to process T observations in $\mathcal{O}(\log_2 T)$ time on parallel architectures. This compares favorably with standard particle smoothers, the complexity of which is linear in T. We derive $\mathcal{L}_p$ convergence results for dSMC, with an explicit upper bound, polynomial in T. We then discuss how to reduce the variance of the smoothing estimates computed by dSMC by (i) designing good proposal distributions for sampling the particles at the initialization of the algorithm, as well as by (ii) using lazy resampling to increase the number of particles used in dSMC. Finally, we design a particle Gibbs sampler based on dSMC, which is able to perform parameter inference in a state-space model at a $\mathcal{O}(\log_2 T)$ cost on parallel hardware.

Keywords: Sequential Monte Carlo; Parallel methods; Particle smoothing; Particle Gibbs

1. Introduction

State-space models (SSM), or hidden Markov models, are a class of statistical models that comprise unobserved (latent) Markovian states $X_t \in \mathcal{X}$ for $t \in \{0, 1, \dots, T\}$, and conditionally independent observations $Y_t \in \mathcal{Y}$. The models can be written in the form

$$\begin{aligned} X_t \mid x_{t-1} &\sim P_t(\mathrm{d}x_t \mid x_{t-1}), \\ Y_t \mid x_t &\sim P_t(\mathrm{d}y_t \mid x_t), \end{aligned} \tag{1}$$

with $X_0 \sim \mathbb{P}_0(\mathrm{d}x_0)$, where $P_t(\mathrm{d}x_t \mid x_{t-1})$ is the transition kernel of the Markov sequence X_t modeling the dynamics of the system, $P_t(\mathrm{d}y_t \mid x_t)$ is the conditional distribution of measurements Y_t, and $\mathbb{P}_0(\mathrm{d}x_0)$ is the prior distribution of the initial state X_0. For simplicity,

we assume that there exist h_t such that $P_t(\mathrm{d}y_t \mid x_t) = h_t(y_t \mid x_t)\,\mathrm{d}y_t$, where $\mathrm{d}y_t$ refers to a dominating measure over $\mathcal{Y}$, for example, the Lebesgue measure if $\mathcal{Y} = \mathbb{R}^{d_y}$.

In this paper, we consider the state-estimation problem which refers to the problem of inferring the states X_t from the measurements Y_t. In particular, we concentrate on the smoothing problem, where the aim is to infer the distribution of the whole trajectory of states $X_{0:T}$ given the whole set of measurements $Y_{0:T}$. A typical application of state estimation consists in target tracking, where the state X_t models the position (and possibly other physical quantities such as the speed) of a moving target, and the observation Y_t corresponds to some noisy and partial or indirect measurement of X_t (Jazwinski, 1970; Bar-Shalom et al., 2001). Additionally, we consider the parameter-estimation problem of inferring the unknown parameters appearing in the model. In addition to target tracking, state and/or parameter estimation problems also arise in various other applications such as in biomedicine, epidemiology, finance, audio signal analysis, and imaging. For an in-depth review of state-space models and their applications, see the books of Särkkä (2013) and Chopin and Papaspiliopoulos (2020).

In the signal processing setting, the solutions to the smoothing problem are focused on computing the marginal conditional distributions of X_t for $t = 0, \ldots, T$ given all the measurements $Y_{0:T}$. However, in the context of Monte Carlo methods – which we also concentrate on here – it is more natural to directly consider the joint distribution of all the states and measurements which can be written as

$$\mathbb{P}(\mathrm{d}x_{0:T}\,, \mathrm{d}y_{0:T}) \coloneqq \mathbb{P}_0(\mathrm{d}x_0) \left\{ \prod_{t=0}^{T} h_t(y_t \mid x_t)\,\mathrm{d}y_t \right\} \left\{ \prod_{t=1}^{T} P_t(\mathrm{d}x_t \mid x_{t-1}) \right\}. \tag{2}$$

In this notation, smoothing consists in representing the posterior distribution of the states conditionally on the observations, $\mathbb{Q}_T(\mathrm{d}x_{0:T}) \coloneqq \mathbb{P}(\mathrm{d}x_{0:T} \mid y_{0:T})$, and in particular being able to approximate expectations such as $\mathbb{Q}_T(\varphi) \coloneqq \mathbb{E}_{\mathbb{Q}_T}[\varphi(X_{0:T})]$ for some function φ of interest. When $\mathbb{P}_0(\mathrm{d}x_0) = \mathbb{P}_0(\mathrm{d}x_0 \mid \theta)$, $P_t(\mathrm{d}x_t \mid x_{t-1}) = P_t(\mathrm{d}x_t \mid x_{t-1}, \theta)$, and $h_t(y_t \mid x_t) = h_t(y_t \mid x_t, \theta)$ depend on a parameter θ, parameter estimation consists in computing estimates θ either as point estimates or in the form of a posterior distribution of the parameter. Formally, if θ is given a prior distribution $p(\mathrm{d}\theta)$, one can represent its posterior distribution as

$$p(\mathrm{d}\theta \mid y_{0:T}) \propto p(\mathrm{d}\theta) p(y_{0:T} \mid \theta), \tag{3}$$

where

$$p(y_{0:T} \mid \theta) = \int_{\mathcal{X}^{T+1}} \mathbb{P}_0(\mathrm{d}x_0 \mid \theta) \left\{ \prod_{t=0}^{T} h_t(y_t \mid x_t, \theta) \right\} \left\{ \prod_{t=1}^{T} P_t(\mathrm{d}x_t \mid x_{t-1}, \theta) \right\}. \tag{4}$$

Except in the case of finite-state SSMs (e.g. Rabiner, 1989), linear Gaussian SSMs (LGSSMs) (Kalman, 1960; Rauch et al., 1965), and certain other special cases, neither

the smoothing nor the parameter estimation problems admit a closed-form solution, and we need to resort to approximations. A successful class of such approximations comprise Gaussian approximation based filtering and smoothing approximations such as extended (Jazwinski, 1970), unscented (Julier et al., 2000), and cubature Kalman filters (Ito and Xiong, 2000; Arasaratnam and Haykin, 2009), as well as their corresponding smoothers (for a review, see, e.g., Särkkä, 2013). Another class of methods is sequential Monte Carlo (SMC) algorithms (see, e.g., Gordon et al., 1993; Doucet et al., 2000; Chopin and Papaspiliopoulos, 2020) such as particle filters and smoothers which are based on Monte Carlo sampling from the filtering and smoothing distributions. These algorithms can, more generally, also sample from the full distribution of Feynman–Kac models (Del Moral, 2004; Chopin and Papaspiliopoulos, 2020) given as a product of Markov kernels and potentials h_t as follows:

$$\mathbb{Q}_T(\mathrm{d}x_{0:T}) \propto \mathbb{P}_0(\mathrm{d}x_0) \left\{ \prod_{t=0}^{T} h_t(x_t) \right\} \left\{ \prod_{t=1}^{T} P_t(\mathrm{d}x_t \mid x_{t-1}) \right\}, \tag{5}$$

which recovers the case of (2) by setting $h_t(x_t) = h_t(y_t \mid x_t)$ in a slight abuse of notation.

The aforementioned finite-state methods, Gaussian approximations, and SMC methods are based on sequential forward and backward recursions which allow for computationally efficient algorithms that scale linearly in the number of time steps $\mathcal{O}(T)$. Although this computational complexity is (in a sense) optimal in classical single-core computers, it is not optimal in multi-core parallel computers which are capable of sub-linear time-complexity in terms of span-complexity (Cormen et al., 2009) – span-complexity referring here to the actual wall-clock time taken by a method when run on a parallel computer which can be less than $\mathcal{O}(T)$ even when the size of data is T. The sequential approximations for filtering and smoothing, in their standard formulation, have a linear time complexity in T even when run on a parallel computer, which is due to the inherent sequential nature of the computations.

However, it was recently shown in Särkkä and García-Fernández (2021) that Bayesian filtering and smoothing recursions (including, e.g., the Kalman filter and smoother) can be reformulated in terms of associative operators that can be time-parallelized to $\mathcal{O}(\log T)$ span-complexity by using a parallel scan algorithm. In Hassan et al. (2021), similar methods were developed for finite-state models, and Yaghoobi et al. (2021) developed Gaussian approximation based parallel methods for non-linear SSMs. These methods reduce the computational cost from linear to logarithmic in the number of time steps T on highly parallel hardware such as graphics processing units (GPUs). Unfortunately, the general formulation of Särkkä and García-Fernández (2021) is not directly applicable to SMC-based particle filters and smoothers, as propagating the associative operator appearing in Särkkä and García-Fernández (2021) is exactly what SMC offers to do in the first place. The aim of this article is to fix this shortcoming by proposing a parallel-in-time (PIT) formulation of SMC, the de-Sequentialized Monte Carlo (dSMC) method, that can be used – either

as a standalone method, or in combination with Gaussian approximations – in order to perform Monte Carlo inference in general SSMs. However, instead of using an associative operator formulation as in Särkkä and García-Fernández (2021), the method uses parallel merging of blocks in a tree structure.

1.1 Related work

Temporal parallelization of general Bayesian filters and smoothers have recently been discussed in Särkkä and García-Fernández (2021), Hassan et al. (2021), and Yaghoobi et al. (2021), but only in the contexts of Gaussian approximations and finite-state models. Parallelization methods for Kalman type of (ensemble) filters via parallel matrix computations over the state dimension are presented in Lyster et al. (1997) and Evensen (2003). In the context of SMC methods, parallelization over particles has been considered in Lee et al. (2010), Rosen and Medvedev (2013), and Murray et al. (2016), however, these methods do not address the time dimension and their computational complexity is still linear in T on parallel hardware. In the context of variational inference (see, e.g. Blei et al., 2017), it was also noted in Aitchison (2019) that operations akin to sequential importance sampling could easily be written as chaining matrix multiplications, allowing to parallelize these on a GPU, both in the time and particle dimensions. Singh et al. (2017) consider blocking strategies for particle Gibbs algorithm, using the Markov property to allow the treatment of non-contiguous time blocks in parallel. Their method, however, works better for larger blocks with a significant overlap, thereby reducing its parallelization properties, and they also do not consider parallelization of particle smoothing. Orthogonally to these direction, coupled smoothing methods, introduced in Jacob et al. (2019) and further developed in Middleton et al. (2019) and Lee et al. (2020), allow to compute unbiased estimates of particle smoothers. This allows to parallelize calculation of smoothing expectations by aggregating many unbiased smoothers together.

Closest to our work is Lindsten et al. (2017) who consider the case of already formed graphical models. In fact, once the tree structure of dSMC is built, our algorithm can be seen as a direct instance of divide and conquer SMC for one dimensional lattices (Lindsten et al., 2017, Section 3.4), which propagates and merges particle samples from children nodes to a parent node. In their article, Lindsten et al. (2017) show the consistency of their algorithm in terms of convergence in probability. This was further improved by Kuntz et al. (2021b) who derived additional theoretical properties of estimates computed from divide-and-conquer SMC. These results can be applied to dSMC as well. However, our method differs from both these articles in several ways. First, Lindsten et al. (2017) do not consider *modifying* the structure of a pre-existing graphical model to be able to parallelize it. Second, the bounds for $\mathcal{L}_p$ errors we derive in this article depend explicitly (and polynomially) on T. These results are specific to dSMC as a parallel algorithm. Third, we derive a parallel-in-time particle Gibbs algorithm for dSMC which can be more generally applied to Lindsten et al. (2017). Lastly, we introduce parallel-in-time initialization of the

algorithm and lazy resamplings as ways to speed up the algorithm and allow for better scalability in the number of particles used.

Finally, we note that Ding and Gandy (2018) introduced a smoothing algorithm leveraging the same binary tree. However, their method differs from ours in the following aspects. The main goal of Ding and Gandy (2018) is to reduce the variance of smoothing algorithms by computing adapted target distributions at each node of the tree. As a consequence, they do not directly address parallelization in time (our main motivation), and, in fact, do not allow for it as their algorithm requires to run a particle filter and a particle smoother *a priori*. They also do not discuss approximated LGSSM (PIT) initialization, lazy schemes, or particle Gibbs extensions.

1.2 Contributions

In Section 2, we introduce a formal divide-and-conquer formulation of the smoothing distribution for a class of Feynman–Kac models, which is then used to define dSMC. We then proceed to study the properties of dSMC and, in particular, we derive $\mathcal{L}_p$ error bounds that only scale polynomially in T for balanced tree representations of the smoothing distribution. Section 3 is concerned with introducing the conditional formulation of dSMC. This is then used to define a PIT particle Gibbs algorithm. In Section 4, we discuss how to construct adapted proposals without breaking the logarithmic scaling in T, and then show how parallel resampling methods can be used to lazily increase the number of particles used in dSMC. Finally, in Section 5, we experimentally demonstrate the statistical and computational properties of our method on a suite of examples. The article concludes with a discussion of the limitations and possible improvements of the de-Sequentialized Monte Carlo method.

2. De-Sequentialized Monte Carlo

We first introduce the core components required for building a parallel-in-time (PIT) particle smoother algorithm that we call de-Sequentialized Monte Carlo (dSMC). Our method relies on a divide-and-conquer approach, where we recursively stitch together partial smoothing distributions $\mathbb{Q}_{a:b}(\mathrm{d}x_{a:b})$ to form the final estimate. In order to do this, we first present the tree structure associated with smoothing in state-space models, and then we discuss how importance sampling-resampling can be leveraged to create joint samples from marginal ones. Finally, we describe the resulting algorithm and derive convergence bounds for it. For the sake of generality, we will consider the potential formulation $h_t(x_t)$ in (5), which possibly depends on y_t, but, by a slight abuse of language, we will still refer to $\mathbb{Q}_T$ as the smoothing distribution. To ensure that the model (5) is not degenerate, we will for simplicity assume that these potentials h_t are positive.

2.1 Tree structure

The recursive expressions for the smoothing distribution

$$\mathbb{Q}_T(\mathrm{d}x_{0:T}) = \frac{1}{L_T}\left[\mathbb{P}_0(\mathrm{d}x_0)\prod_{t=1}^{T} P_t(\mathrm{d}x_t \mid x_{t-1})\right]\prod_{t=0}^{T} h_t(x_t), \tag{6}$$

where L_T is a normalizing constant, are given by the forward Feynman–Kac recursion (see, e.g., Del Moral, 2004; Chopin and Papaspiliopoulos, 2020)

$$\mathbb{Q}_{t+1}(\mathrm{d}x_{0:t+1}) \propto \mathbb{Q}_t(\mathrm{d}x_{0:t})h_{t+1}(x_{t+1})P_t(\mathrm{d}x_{t+1} \mid x_t),$$

or the backward one

$$\mathbb{Q}_T(\mathrm{d}x_{t:T}) \propto h_{t+1}(x_{t+1})p_t(x_{t+1} \mid x_t)\,\mathrm{d}x_t\,\mathbb{Q}_T(\mathrm{d}x_{t+1:T}),$$

when $P_t(\mathrm{d}x_{t+1} \mid x_t)$ admits a density $p_t(x_{t+1} \mid x_t)$ with respect to a fixed (x_t-independent) measure $\mathrm{d}x_{t+1}$. Leveraging these recursions respectively corresponds to particle filtering and particle smoothing algorithms, and results in algorithms for sampling from $\mathbb{Q}_T$ that scale computationally in $\mathcal{O}(T)$.

In this section we instead propose a divide-and-conquer recursive construction of the smoothing density $\mathbb{Q}_T$. In order to do so, we introduce the concept of partial smoothing distributions.

Definition 1 *Let $(\nu_c(\mathrm{d}x_c))_{c=0}^T$ be a collection of probability measures, such that for all $c > 0$ and all $x_{c-1} \in \mathcal{X}$, $P_c(\mathrm{d}x_c \mid x_{c-1})$ is absolutely continuous with respect to $\nu_c(\mathrm{d}x_c)$. Then for any $0 \leq a \leq b \leq T$, we can define*

$$\mathbb{Q}^{\nu}_{a:b}(\mathrm{d}x_{a:b}) \coloneqq \frac{1}{L^{\nu}_{a:b}}\left[\nu_a(\mathrm{d}x_a)\prod_{t=a+1}^{b} P_t(\mathrm{d}x_t \mid x_{t-1})\right]\prod_{t=a+1}^{b} h_t(x_t), \tag{7}$$

where $L^{\nu}_{a:b}$ is a normalizing constant (assumed to be positive), and by convention the product over an empty set is 1, so that, for any a, $\mathbb{Q}^{\nu}_{a:a}(\mathrm{d}x_a) = \nu_a(\mathrm{d}x_a)$ and $L^{\nu}_{a:a} = 1$.

Provided that ν_0 defines the filtering posterior distribution of x_0, we can then recover the original $\mathbb{Q}_T$ from $\mathbb{Q}^{\nu}_{0:T}$. This corresponds to the following proposition.

Proposition 2 *For any family $\nu_{0:T}$ given by Definition 1, and such that the initial distribution verifies $\nu_0(\mathrm{d}x_0) \propto h_0(x_0)\mathbb{P}_0(\mathrm{d}x_0)$, we have*

$$\mathbb{Q}_T(\mathrm{d}x_{0:T}) = \mathbb{Q}^{\nu}_{0:T}(\mathrm{d}x_{0:T}). \tag{8}$$

The partial smoothing distributions $(\mathbb{Q}^{\nu}_{a:b})_{0\leq a<b,\leq T}$ can then be stitched together, forming a recursive structure for the smoothing operation.

Proposition 3 *For any $0 \leq a < c < b \leq T$, we have*

$$\mathbb{Q}^{\nu}_{a:b}(\mathrm{d}x_{a:b}) = \frac{L^{\nu}_{a:c-1} L^{\nu}_{c:b}}{L^{\nu}_{a:b}} \omega^{\nu}_{c}(x_{c-1}, x_c) \mathbb{Q}^{\nu}_{a:c-1}(\mathrm{d}x_{a:c-1}) \mathbb{Q}^{\nu}_{c:b}(\mathrm{d}x_{c:b}), \tag{9}$$

where ω^{ν}_{c} is defined as the following Radon–Nikodym derivative:

$$\omega^{\nu}_{c}(x_{c-1}, x_c) \coloneqq \frac{P_c(\mathrm{d}x_c \mid x_{c-1}) h_c(x_c)}{\nu_c(\mathrm{d}x_c)}. \tag{10}$$

Proof For all $0 \leq a < c < b \leq T$, we have:

$$\begin{aligned}
&\frac{L^{\nu}_{a:c-1} L^{\nu}_{c:b}}{L^{\nu}_{a:b}} \omega^{\nu}_{c}(x_{c-1}, x_c) \mathbb{Q}^{\nu}_{a:c-1}(\mathrm{d}x_{a:c-1}) \mathbb{Q}^{\nu}_{c:b}(\mathrm{d}x_{c:b}) \\
&= \frac{1}{L^{\nu}_{a:b}} \frac{P_c(\mathrm{d}x_c \mid x_{c-1}) h_c(x_c)}{\nu_c(\mathrm{d}x_c)} \\
&\times \left[\nu_a(\mathrm{d}x_a) \prod_{t=a+1}^{c-1} P_t(\mathrm{d}x_t \mid x_{t-1}) \right] \prod_{t=a+1}^{c-1} h_t(x_t) \\
&\times \left[\nu_c(\mathrm{d}x_c) \prod_{t=c+1}^{b} P_t(\mathrm{d}x_t \mid x_{t-1}) \right] \prod_{t=c+1}^{b} h_t(x_t) \\
&= \frac{1}{L^{\nu}_{a:b}} \left[\nu_a(\mathrm{d}x_a) \prod_{t=a+1}^{b} P_t(\mathrm{d}x_t \mid x_{t-1}) \right] \prod_{t=a+1}^{b} h_t(x_t) \\
&= \mathbb{Q}^{\nu}_{a:b}(\mathrm{d}x_{a:b}).
\end{aligned} \tag{11}$$

■

The recursive property exhibited by Proposition 3 allows us to construct an arbitrary tree structure on the smoothing distribution. This construction is illustrated in Figure 1. In practice, we could use any ordered binary tree structure on $\{0, 1, \ldots, T\}$ to define a well-posed recursive representation of $\mathbb{Q}_T$, but, as we will see in Sections 2.2 and 2.3, balanced representations offer better statistical and computational properties.

In practice, the stitching operation described by Proposition 3 is not tractable in closed-form, and we need to resort to Monte Carlo integration instead.

2.2 Sample stitching

For notational simplicity, in this section and all subsequent ones we do not emphasize the dependency of our estimates on ν. Suppose that we have two independent Monte Carlo

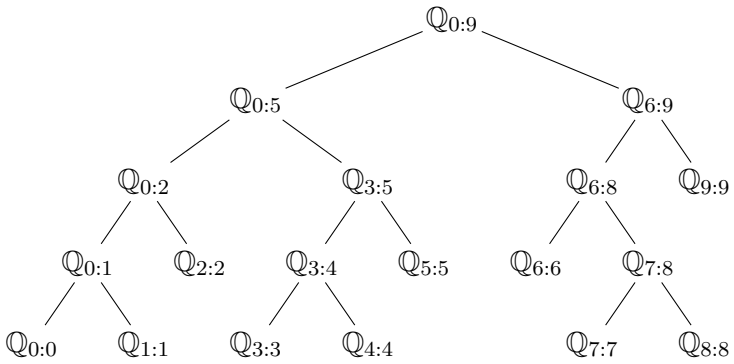

Figure 1: Example of a recursive tree structure for $\mathbb{Q}_{0:9}$.

approximations

$$\begin{aligned} \mathbb{Q}_{a:c-1} \approx \mathbb{Q}^N_{a:c-1} &\coloneqq \sum_{n=1}^N w^n_{c-1} \delta_{X^n_{a:c-1}} \\ \mathbb{Q}_{c:b} \approx \mathbb{Q}^N_{c:b} &\coloneqq \sum_{n=1}^N w^n_{c} \delta_{X^n_{c:b}}, \end{aligned} \tag{12}$$

following Lindsten et al. (2017) and Kuntz et al. (2021a), we can then form the "product-form" importance empirical density

$$\widetilde{\mathbb{Q}}^N_{a:b} \coloneqq \sum_{m,n=1}^N W^{m,n}_c \delta_{[X^m_{a:c-1}, X^n_{c:b}]}, \tag{13}$$

where

$$W^{m,n}_c = \frac{w^m_{c-1} w^n_c \omega_c(X^m_{c-1}, X^n_c)}{\sum_{i,j=1}^N w^i_{c-1} w^j_c \omega_c(X^i_{c-1}, X^j_c)}. \tag{14}$$

As described in Kuntz et al. (2021a), this estimator exhibits better statistical properties than the "naive" estimator

$$\sum_{n=1}^N \frac{w^n_{c-1} w^n_c \omega_c(X^n_{c-1}, X^n_{c-1})}{\sum_{m=1}^N w^m_{c-1} w^m_c \omega_c(X^m_{c-1}, X^m_{c-1})} \delta_{[X^n_{a:c-1}, X^n_{c:b}]}$$

when the $(X^n_{c-1})_{n=1}^N$, and $(X^n_c)_{n=1}^N$ have been sampled independently.

Moreover, the denominator of (14) directly provides us with an estimate of the normalizing constant increment, that is, if $L^N_{a:c-1}$ and $L^N_{c:b}$ are estimates of the normalizing

constants of $\mathbb{Q}_{a:c-1}$ and $\mathbb{Q}_{c:b}$, respectively, then, following Proposition 3, we know that

$$\begin{aligned} L_{a:b} &= L_{a:c-1}L_{c:b}\iint \omega_c(x_{c-1},x_c)\mathbb{Q}_{a:c-1}(\mathrm{d}x_{c-1})\mathbb{Q}_{c:b}(\mathrm{d}x_c) \\ &\approx L^N_{a:c-1}L^N_{c:b}\sum_{i,j=1}^N w^i_{c-1}w^j_c\omega_c(X^i_{c-1},X^j_{c-1}). \end{aligned} \tag{15}$$

Algorithm 1: Block combination

`// All operations on indices` m,n `are done in parallel`
Function COMBINE$\big(X^{1:N}_{a:c-1}, w^{1:N}_{c-1}, L^N_{a:c-1}, X^{1:N}_{c:b}, w^{1:N}_c, L^N_{c:b}\big)$
  `// In all but the initial step, the weights` w^m_{c-1} `and` w^n_c `are` $1/N$`.`
  $L^N_c \leftarrow \sum_{m,n=1}^N \omega_c(X^m_{c-1},X^n_c)w^m_{c-1}w^n_c$
  $W^{m,n}_c \leftarrow \omega_c(X^m_{c-1},X^n_c)w^m_{c-1}w^n_c/L^N_c$
  `// In parallel, using, e.g., multinomial or systematic resampling:`
  Sample N times from $\sum_{m,n} W^{m,n}_c\delta_{X^m_c}(\mathrm{d}x_c)\delta_{X^n_{c-1}}(\mathrm{d}x_{c-1})$ to get $(X^{l^n}_{c-1}, X^{r^n}_c)_{1\le n\le N}$
  `// The two loops below can be done in parallel:`
  for $c' = a,\dots,c-1$ **in parallel do**
    $\widetilde{X}^n_{c'} \leftarrow X^{l^n}_{c'}$
  for $c' = c,\dots,b$ **in parallel do**
    $\widetilde{X}^n_{c'} \leftarrow X^{r^n}_{c'}$
  return $\widetilde{X}^{1:N}_{a:b}$, $L^N_{a:c-1}L^N_{c:b}L^N_c$

When the importance estimator in (13) has been formed, we can then resample N pairs of partial smoothing paths $(l^n, r^n)_{n=1}^N$ according to the normalized weights $W^{m,n}_c$ to obtain a stitched Monte Carlo approximation $\mathbb{Q}^N_{a:b} \coloneqq \frac{1}{N}\sum_{n=1}^N \delta_{[X^{l^n}_{a:c-1}, X^{r^n}_{c:b}]}$. This construction is summarized in Algorithm 1, from which we can also compute the normalizing constant increment as a by-product. On the other hand, for all $t = 0,\dots,T$, we can define the initial self-normalized importance approximation $\mathbb{Q}^N_{t:t} \coloneqq \sum_{n=1}^N W^n_t\delta_{X^n_t}$, where, for all t, the X^n_t's are i.i.d. sampled from q_t, and the $W^n_t \propto \frac{\nu_t(X^n_t)}{q_t(X^n_t)}$ sum to 1.

Under this construction, we can show that the resulting Monte Carlo $\mathcal{L}_p$ error is well-behaved.

Proposition 4 *Let $p \ge 1$ be an integer, suppose that the $(l^n, r^n)_{n=1}^N$ are sampled according to a categorical distribution (i.e. multinomial resampling), that φ is a bounded measurable function, and that ω_c is bounded. If for any measurable bounded functions $\varphi_{a:c-1}$ and $\varphi_{c:b}$*

we have

$$\mathbb{E}\left[\left|\mathbb{Q}_{a:c-1}(\varphi_{a:c-1}) - \mathbb{Q}^N_{a:c-1}(\varphi_{a:c-1})\right|^p\right]^{1/p} \leq C^p_{a:c-1}\frac{\|\varphi_{a:c-1}\|_\infty}{N^{1/2}}, \tag{16}$$

$$\mathbb{E}\left[\left|\mathbb{Q}_{c:b}(\varphi_{c:b}) - \mathbb{Q}^N_{c:b}(\varphi_{c:b})\right|^p\right]^{1/p} \leq C^p_{c:b}\frac{\|\varphi_{c:b}\|_\infty}{N^{1/2}}, \tag{17}$$

for some constants $C^p_{a:c-1}, C^p_{c:b}$ independent of N, $\varphi_{a:c-1}$, and $\varphi_{c:b}$, then,

$$\mathbb{E}\left[\left|\mathbb{Q}_{a:b}(\varphi) - \mathbb{Q}^N_{a:b}(\varphi)\right|^p\right]^{1/p} \leq \left(4\min(C^p_{a:c-1}, C^p_{c:b})\|\bar{\omega}_c\|_\infty + 2^{(p+1)/p}\right)\frac{\|\varphi\|_\infty}{N^{1/2}}, \tag{18}$$

where $\bar{\omega}_c \coloneqq \omega_c/Z_c$ and $Z_c \coloneqq L_{a:b}/(L_{a:c-1}L_{c:b})$.

The proof of Proposition 4 may be found in Appendix C. Proposition 4 allows us to derive upper bounds to the total $\mathcal{L}_p$ error as a consequence.

Corollary 5 *Suppose that the ω_c/Z_c's and the $\frac{\nu_c}{q_c}$'s are uniformly bounded by some constant Ω independent of c, and let $\mathbb{Q}^N_{c:c} = \frac{1}{N}\sum_{n=1}^N W^n_c \delta_{X^n_c}$, where, for all c, the X^n_c's are i.i.d. sampled from q_c, then*

$$C^p_{0:T} = O\left((4\Omega)^D\right), \tag{19}$$

where D is the depth of the tree structure chosen for the smoothing operation (see Figure 1). In particular, if the tree is balanced, that is, if $D = \lceil \log_2 T \rceil$,

$$C^p_{0:T} = O\left(T^{2+\log\Omega}\right). \tag{20}$$

Proof The initial case comes from (Del Moral, 2004, Lemma 7.3.3) so that for all c, $C^p_{c:c} \leq 2^{(p+1)/p}$. The result then proceeds by using inequalities on the progression of arithmetico-geometric sequences (see, e.g., Riley et al., 2006, Section 4.2.3). ■

Remark 6 *The uniform bounding of the quantities*

$$\frac{\omega_c}{Z_c} = \frac{\omega_c}{\iint \omega_c(x_{c-1}, x_c)\mathbb{Q}_{a:c-1}(\mathrm{d}x_{c-1})\mathbb{Q}_{c:b}(\mathrm{d}x_c)}, \tag{21}$$

for all c, assumed in Corollary 5, is, for example, true as soon as the ω_c's are uniformly bounded below and above. This hypothesis, albeit strong, is typically assumed in proofs of the uniform convergence of particle filtering algorithms (Del Moral and Guionnet, 2001).

Following Crisan and Doucet (2000, Proof of Lemma 5), for $p > 2$, Chebyshev's inequality and Borel–Cantelli lemma also provides the following corollary.

Corollary 7 *Under the same hypotheses, $\mathbb{Q}^N_{0:T}(\varphi)$ converges almost surely to $\mathbb{Q}_{0:T}(\varphi)$.*

An interesting point to notice is that for very unbalanced trees with depth of order T, Proposition 4 recovers the usual exponential scaling in T (Andrieu et al., 2001) of the mean squared error, instead of the polynomial scaling obtained when the tree is balanced.

Algorithm 2: Smoother initialization

`// All operations on indices` m, n `are done in parallel`
for $t = 0, \dots, T$ **in parallel do**
  $X_t^n \leftarrow$ sample from $q_t(\mathrm{d}x_t)$
  if $t = 0$ **then**
    $w_0^n \leftarrow h_0(X_0^n) \frac{\mathbb{P}_0(\mathrm{d}x_0)}{q_0(\mathrm{d}x_0)}(X_0^n)$
  else
    $w_t^n \leftarrow \frac{\nu_t(\mathrm{d}x_t)}{q_t(\mathrm{d}x_t)}(X_t^n)$
  $W_t^n \leftarrow w_t^n / \sum_{m=1}^N w_t^m$
  $L_t^N \leftarrow \frac{1}{N} \sum_{m=1}^N w_t^m$
return $X_{0:T}^{1:N}, W_{0:T}^{1:N}, L_{0:T}^N$

2.3 Algorithm

The Monte Carlo approximation of $\mathbb{Q}_{0:T}$ can be computed using a recursive algorithm which can be parallelized across all operations happening at each level of the tree depth. At initialization, we simply need to sample independently N times from $T+1$ proposal distributions q_t, $t = 0, \dots, T$, and then form the resulting importance sampling representation of all the distributions $\mathbb{Q}_{t:t}$, $t = 0, \dots, T$, following Definition 1 and Proposition 2. This is summarized in Algorithm 2. In order to obtain a balanced tree, we can recursively split at the midpoint of the partial smoothing interval, essentially recovering a binary tree when $T+1$ is a power or 2. This results in Algorithm 3.

The smoothing algorithm then simply consists in passing the output of Algorithm 2 to Algorithm 3. It is worth noting that while Algorithm 3 is correct, its recursive nature makes its implementation on parallel devices tedious if one wants to benefit from hardware acceleration. Moreover it does not consist in a tail recursion (see, e.g. Muchnick, 1997, Chap. 15), so that it cannot easily be transformed into a loop that would be easier to parallelize. However, the split-combine operations can be reformulated as a series of tensor reshaping operations, which is more amenable to parallelization. We provide this equivalent, albeit parallelizable, formulation of the algorithm in Appendix A.

Consider now the choice of tree partitioning given in Figure 1. The nodes correspond to the combination operation, while the edges correspond to the split happening in Algorithm 3. All the operations at a given depth can be run fully in parallel, each of them being entirely parallelizable too with respect to the particle samples, except for the resampling operation. The resampling operation requires normalizing the weights and running parallel search operations, which can be done with span complexity (Cormen et al., 2009) of $\mathcal{O}(\log N)$ on parallel architectures (using prefix-sum operations, see, e.g., Murray et al., 2016), so that each level of the tree has span complexity $\mathcal{O}(\log N)$. This results in a parallelized algorithm run time that globally scales linearly with the depth of the smoothing

Algorithm 3: Recursion

Function Recursion$\big(X^{1:N}_{a:c-1}, w^{1:N}_{a:c-1}, L^N_{a:c-1}, X^{1:N}_{c:b}, w^{1:N}_{c:b}, L^N_{c:b}\big)$
  if $a = c-1$ *and* $b = c$ **then**
    return Combine$\big(X^{1:N}_{a:a}, w^{1:N}_a, L^N_{a:a}, X^{1:N}_{b:b}, w^{1:N}_b, L^N_{b:b}\big)$
  else if $c-1 > a$ *and* $b = c$ **then**
    $c' \leftarrow \lfloor \frac{a+c-1}{2} \rfloor$
    $X^{1:N}_{a:c-1}, L^N_{a:c-1} \leftarrow$
     Recursion$\big(X^{1:N}_{a:c'-1}, w^{1:N}_{a:c'-1}, L^N_{a:c'-1}, X^{1:N}_{c':c-1}, w^{1:N}_{c':c-1}, L^N_{c':c-1}\big)$
    return Combine$\big(X^{1:N}_{a:c-1}, (1/N)^N_{n=1}, L^N_{a:c-1}, X^{1:N}_{b:b}, w^{1:N}_b, L^N_{b:b}\big)$
  else if $a = c-1$ *and* $b > c$ **then**
    $c' \leftarrow \lfloor \frac{c+b}{2} \rfloor$
    $X^{1:N}_{c:b}, L^N_{c:b} \leftarrow$ Recursion$\big(X^{1:N}_{c:c'-1}, w^{1:N}_{c:c'-1}, L^N_{c:c'-1}, X^{1:N}_{c':b}, w^{1:N}_{c':b}, L^N_{c':b}\big)$
    return Combine$\big(X^{1:N}_{a:a}, w^{1:N}_a, L^N_{a:a}, X^{1:N}_{c:b}, (1/N)^N_{n=1}, L^N_{c:b}\big)$
  else
    $c' \leftarrow \lfloor \frac{a+c-1}{2} \rfloor$
    $X^{1:N}_{a:c-1}, L^N_{a:c-1} \leftarrow$
     Recursion$\big(X^{1:N}_{a:c'-1}, w^{1:N}_{a:c'-1}, L^N_{a:c'-1}, X^{1:N}_{c':c-1}, w^{1:N}_{c':c-1}, L^N_{c':c-1}\big)$
    $c' \leftarrow \lfloor \frac{c+b}{2} \rfloor$
    $X^{1:N}_{c:b}, L^N_{c:b} \leftarrow$ Recursion$\big(X^{1:N}_{c:c'-1}, w^{1:N}_{c:c'-1}, L^N_{c:c'-1}, X^{1:N}_{c':b}, w^{1:N}_{c':b}, L^N_{c':b}\big)$
    return Combine$\big(X^{1:N}_{a:c-1}, (1/N)^N_{n=1}, L^N_{a:c-1}, X^{1:N}_{c:b}, (1/N)^N_{n=1}, L^N_{c:b}\big)$

tree considered, and logarithmically in the number of particles. As a consequence, we have the following proposition.

Proposition 8 *The total span complexity of dSMC is* $\mathcal{O}(\log_2(T)\log(N))$.

Remark 9 *It is worth noting that some alternative resampling methods exist that allow to parallelize the resampling operation, at the cost of biasing it, or at the cost of random execution time (Murray et al., 2016). We discuss these methods and the additional benefits they provide for dSMC in Section 4.2.*

3. Parallel-in-time particle Gibbs

We now focus on deriving a conditional formulation of dSMC (that we call c-dSMC) that we then use to build a PIT particle Gibbs algorithm. We quickly discuss its degeneracy properties, and in particular the fact that it may mix well even without the addition of a backward sampling step.

3.1 Conditional dSMC sampler

Particle Gibbs methods were introduced in Andrieu et al. (2010) in order to sample from the joint posterior $\mathbb{Q}_{0:T}(\mathrm{d}x_{0:T}, \mathrm{d}\theta)$ of a state-space model. It consists in successively applying two conditional sampling steps: (i) sampling θ conditionally on a given smoothing trajectory $x^*_{0:T}$, and (ii) sampling a smoothing trajectory $x'_{0:T}$ conditionally on $x^*_{0:T}$ and θ. Step (ii) needs to be understood as "conditionally to one of the trajectories sampled by the SMC algorithm being $x^*_{0:T}$".

Due to the arbitrary tree representation of the smoothing operation, it is complicated to manipulate the complete expression for the distribution of all the random variables generated during the course of the smoothing algorithm[1]. However, we can still provide a natural recursive expression that will serve as a support for understanding the behavior of the conditional distributions. In order to make notations simpler, we write $\sigma_{a:c-1}(k)$, $k = a, a+1, \ldots, c-1$ for the resampling array (i.e., the array of the resampling indices) applied to node k, and we write $\sigma^n_{a:c-1}(k)$ for its n-th element (and similarly for $\sigma_{c:b}(k)$).

Remark 10 *$\sigma_{a:c-1}$ is a function of the left-right resampling indices $l^{1:N}_{a+1:c-1}, r^{1:N}_{a+1:c-1}$ generated deeper in the recursion tree, and similarly for $\sigma_{c:b}$ via the recursion*

$$\begin{aligned} \sigma^m_{a:b}(k) &= \sigma^{l^m_c}_{a:c-1}(k), \quad \textit{for all } a \leq k < c, \\ \sigma^m_{a:b}(k) &= \sigma^{r^m_c}_{c:b}(k), \quad \textit{for all } c \leq k \leq b, \end{aligned} \tag{22}$$

and initial values

$$\sigma^m_{a:a}(a) = m, \tag{23}$$

for all $m \in \{1, 2, \ldots, N\}$.

Under this notation, if the initial Monte Carlo approximation for the $\mathbb{Q}_{c:c}$'s are given by

$$\mathbb{Q}^N_{c:c} = \frac{1}{N} \sum_{n=1}^{N} W^n_c \delta_{X^n_c},$$

then for all $a \leq k \leq b$, $a < b$, we have

$$\mathbb{Q}^N_{a:b}(\mathrm{d}x_k) = \frac{1}{N} \sum_{n=1}^{N} \delta_{X^{\sigma_{a:b}(k)}_k}(\mathrm{d}x_k).$$

That is, $\sigma_{a:b}$ encodes the subset (with repetitions) of particles that survived from initialization down to the partial smoothing distribution approximation $\mathbb{Q}^N_{a:b}$.

1. Although this was done for instance in Lindsten et al. (2017) to prove the unbiasedness of their resulting likelihood estimate.

For $a < c < b$, let $\psi_{a:c-1}(\mathrm{d}x_{a:c-1}^{1:N}, l_{a+1:c-1}^{1:N}, r_{a+1:c-1}^{1:N})$ and $\psi_{c:b}(\mathrm{d}x_{c:b}^{1:N}, l_{c+1:b}^{1:N}, r_{c+1:b}^{1:N})$ be the full distributions of all the random variables generated by dSMC for the partial smoothing distributions $\mathbb{Q}_{a:c-1}$ and $\mathbb{Q}_{c:b}$, respectively. The full distribution of all the random variables generated to form the resampled approximation $\mathbb{Q}_{a:b}^N$ of $\mathbb{Q}_{a:b}$ is given by

$$\psi_{a:b}(\mathrm{d}x_{a:b}^{1:N}, l_{a+1:b}^{1:N}, r_{a+1:b}^{1:N}) = \psi_{a:c-1}(\mathrm{d}x_{a:c-1}^{1:N}, l_{a+1:c-1}^{1:N}, r_{a+1:c-1}^{1:N}) \times \psi_{c:b}(\mathrm{d}x_{c:b}^{1:N}, l_{c+1:b}^{1:N}, r_{c+1:b}^{1:N}) \times \left\{ \prod_{n=1}^{N} W_c^{l_c^n, r_c^n} \right\}, \quad (24)$$

where for all $m, n \in \{1, \ldots, N\}$, we have

$$W_c^{m,n} \propto \omega_c \left(x_{c-1}^{\sigma_{a:c-1}^m(c-1)}, x_c^{\sigma_{c:b}^n(c)} \right) \quad (25)$$

so that $\sum_{m,n=1}^{N} W_c^{m,n} = 1$, and the initial distributions are given by $\psi_{a:a}(\mathrm{d}x_a^{1:N}) = \prod_{n=1}^{N} \nu_a(\mathrm{d}x_a^n)$ (for the sake of clarity, we restrict to the case $\nu_a = q_a$).

Equation (24) further allows to define the full distribution of all the random variables generated to form the weighted approximation $\widetilde{\mathbb{Q}}_{a:b}^N$ (13) as

$$\widetilde{\psi}_{a:b}(\mathrm{d}x_{a:b}^{1:N}, l_{a+1:b}^{1:N}, r_{a+1:b}^{1:N}) = \psi_{a:c-1}(\mathrm{d}x_{a:c-1}^{1:N}, l_{a+1:c-1}^{1:N}, r_{a+1:c-1}^{1:N}) \times \psi_{c:b}(\mathrm{d}x_{c:b}^{1:N}, l_{c+1:b}^{1:N}, r_{c+1:b}^{1:N}). \quad (26)$$

Similarly, the related estimate of the normalizing constant

$$L_{a:b}^N = L_{a:b}^N(x_{a:b}^{1:N}, l_{a+1:c-1}^{1:N}, l_{c+1:b}^{1:N}, r_{a+1:c-1}^{1:N}, r_{c+1:b}^{1:N}) \quad (27)$$

follows the recursion

$$L_{a:b}^N = L_{a:c-1}^N L_{c:b}^N \left\{ N^{-2} \sum_{i,j=1}^{N} \omega_c \left(x_{c-1}^{\sigma_{a:c-1}^i(c-1)}, x_c^{\sigma_{c:b}^j(c)} \right) \right\}. \quad (28)$$

Putting these together allows us to characterize recursively the invariant distribution of our specific version of the particle Gibbs kernel (Andrieu et al., 2010), that we will then use in order to express the related conditional dSMC distribution

$$\pi_{a:b}(\mathrm{d}x_{a:b}^{1:N}, l_{a+1:b}^{1:N}, r_{a+1:b}^{1:N}) = \frac{L_{a:b}^N}{L_{a:b}} \widetilde{\psi}_{a:b}(\mathrm{d}x_{a:b}^{1:N}, l_{a+1:b}^{1:N}, r_{a+1:b}^{1:N}). \quad (29)$$

Following Andrieu et al. (2010), define

$$\pi_{a:b}(\mathrm{d}x_{a:b}^{1:N}, l_{a+1:b}^{1:N}, r_{a+1:b}^{1:N}, m, n) = \pi_{a:b}(\mathrm{d}x_{a:b}^{1:N}, l_{a+1:b}^{1:N}, r_{a+1:b}^{1:N}) W_c^{m,n}, \quad (30)$$

corresponding to sampling once from (13). We have

$$\begin{aligned}
&\pi_{a:b}(\mathrm{d}x_{a:b}^{1:N}, l_{a+1:b}^{1:N}, r_{a+1:b}^{1:N}, m, n) \qquad (31)\\
&\quad= \frac{L_{a:c-1}^N L_{c:b}^N \left\{ N^{-2} \sum_{i,j=1}^N \omega_c \left(x_{c-1}^{\sigma_{a:c-1}^i(c-1)}, x_c^{\sigma_{c:b}^j(c)} \right) \right\}}{L_{a:c-1} L_{c:b} \int \omega_c(x_{c-1}, x_c) \mathbb{Q}_{a:c-1}(\mathrm{d}x_{c-1}) \mathbb{Q}_{c:b}(\mathrm{d}x_c)} W_c^{m,n}\\
&\qquad\times \psi_{a:c-1}(\mathrm{d}x_{a:c-1}^{1:N}, l_{a+1:c-1}^{1:N}, r_{a+1:c-1}^{1:N}) \times \psi_{c:b}(\mathrm{d}x_{c:b}^{1:N}, l_{c+1:b}^{1:N}, r_{c+1:b}^{1:N})\\
&\quad\propto \pi_{a:c-1}(\mathrm{d}x_{a:c-1}^{1:N}, l_{a+1:c-1}^{1:N}, r_{a+1:c-1}^{1:N}) \times \left\{ \prod_{i=1}^N W_{c_l}^{l_{c_l}^i, r_{c_l}^i} \right\} \qquad (32)\\
&\qquad\times \pi_{c:b}(\mathrm{d}x_{c:b}^{1:N}, l_{c+1:b}^{1:N}, r_{c+1:b}^{1:N}) \times \left\{ \prod_{i=1}^N W_{c_r}^{l_{c_r}^i, r_{c_r}^i} \right\}\\
&\qquad\times \omega_c \left(x_{c-1}^{\sigma_{a:c-1}^m(c-1)}, x_c^{\sigma_{c:b}^n(c)} \right),
\end{aligned}$$

where we assumed that we have $a < c-1$ and $c < b$ (as we otherwise recover the base case for $\psi_{a:c-1}$ and $\psi_{c:b}$ and the recursion can be stopped), that c_l and c_r are the indices for the stitching that formed $\psi_{a:c-1}$ and $\psi_{c:b}$ in Proposition 3, respectively, and where the constant of normalization is $N^2 \int \omega_c(x_{c-1}, x_c) \mathbb{Q}_{a:c-1}(\mathrm{d}x_{c-1}) \mathbb{Q}_{c:b}(\mathrm{d}x_c)$.

This, in turn, can be rewritten to isolate a "star trajectory": developing line (32) we obtain

$$\begin{aligned}
&\pi_{a:c-1}(\mathrm{d}x_{a:c-1}^{1:N}, l_{a+1:c-1}^{1:N}, r_{a+1:c-1}^{1:N}) \times \left\{ \prod_{i \neq m} W_{c_l}^{l_{c_l}^i, r_{c_l}^i} \right\} \times W_{c_l}^{l_{c_l}^m, r_{c_l}^m}\\
&\quad= \frac{L_{a:c_l-1}^N L_{c_l:c-1}^N \left\{ N^{-2} \sum_{i,j=1}^N \omega_{c_l} \left(x_{c_l-1}^{\sigma_{a:c_l-1}^i(c_l-1)}, x_{c_l}^{\sigma_{c_l:c-1}^j(c_l)} \right) \right\}}{L_{a:c_l-1} L_{c_l:c-1} \iint \omega_{c_l}(x_{c_l-1}, x_{c_l}) \mathbb{Q}_{a:c_l-1}(\mathrm{d}x_{c_l-1}) \mathbb{Q}_{c_l:c-1}(\mathrm{d}x_{c_l})}\\
&\qquad\times \psi_{a:c_l-1}(\mathrm{d}x_{a:c_l-1}^{1:N}, l_{a+1:c_l-1}^{1:N}, r_{a+1:c_l-1}^{1:N}) \times \psi_{c_l:c-1}(\mathrm{d}x_{c_l:c-1}^{1:N}, l_{c_l+1:c-1}^{1:N}, r_{c_l+1:c-1}^{1:N}),
\end{aligned}$$

which can be further decomposed in

$$\begin{aligned}
&\frac{L_{a:c_l-1}^N}{L_{a:c_l-1}} \psi_{a:c_l-1}(\mathrm{d}x_{a:c_l-1}^{1:N}, l_{a+1:c_l-1}^{1:N}, r_{a+1:c_l-1}^{1:N})\\
&\qquad\qquad\times \frac{L_{c_l:c-1}^N}{L_{c_l:c-1}} \psi_{c_l:c-1}(\mathrm{d}x_{c_l:c-1}^{1:N}, l_{c_l+1:c-1}^{1:N}, r_{c_l+1:c-1}^{1:N})\\
&\qquad\qquad\qquad\times \left\{ \prod_{i \neq m} W_{c_l}^{l_{c_l}^i, r_{c_l}^i} \right\} \omega_{c_l} \left(x_{c_l-1}^{\sigma_{a:c_l-1}^{l_{c_l}^m}(c_l-1)}, x_{c_l}^{\sigma_{c_l:c-1}^{r_{c_l}^m}(c_l)} \right). \qquad (33)
\end{aligned}$$

The final structure of (33) mirrors that of (31), with the decomposition $\left\{\prod_{i\neq m} W_{c_l}^{l_{c_l}^i, r_{c_l}^i}\right\} \times \omega_{c_l}\left(x_{c_l-1}^{\sigma_{a:c_l-1}^{l_{c_l}^m}(c_l-1)}, x_{c_l}^{\sigma_{c_l:c-1}^{r_{c_l}^m}(c_l)}\right)$. This ensures that we can recursively decompose $\pi_{a:b}$ in a star trajectory and a remainder by defining $\sigma_{a:c-1}^* = \sigma_{a:c-1}^I$, $\sigma_{c:b}^* = \sigma_{c:b}^J$, where (I, J) is distributed according to a categorical distribution on $W_c^{m,n}$, and $\sigma_{a:b}^* = \left[\sigma_{a:c-1}^*, \sigma_{c:b}^*\right]$. This in turn defines $X_{a:b}^* \coloneqq X_{a:b}^{\sigma_{a:b}^*} = \left[X_{a:c-1}^*, X_{c:b}^*\right]$, and the related l_c^*, r_c^*, which correspond to the resampling indices pairs that eventually lead to the star trajectory. Following the recursive construction of (31) and (33), we are able to isolate the star trajectory from the rest of the variables appearing in $\pi_{a:b}$ to form the marginal distribution

$$\pi_{a:b}(\mathrm{d}x_{a:b}^*) \propto \prod_{c=a}^{b} \nu_c(\mathrm{d}x_c^*) \prod_{c=a}^{b} \omega_c(x_{c-1}^*, x_c^*), \tag{34}$$

which corresponds exactly to $\mathbb{Q}_{a:b}$. This allows us to formulate the following proposition.

Proposition 11 (Conditional dSMC) *Under $\pi_{a:b}$, the star trajectory $X_{a:b}^*$ is marginally distributed according to $\mathbb{Q}_{a:b}$, and the remaining variables admit as a conditional distribution, given the star trajectory, the distribution defined by Algorithm 4 (discussed in next section).*

Remark 12 *While we are concerned here with parallel-in-time smoothing, the construction above generalizes to the algorithm of Lindsten et al. (2017) which considers stitching independent SMC samplers by means of an operation akin to that of Proposition 3. This means that, provided that one is able to implement a conditional SMC for each individual SMC sampler in the tree structure – possibly in the form of an ancestor sampling algorihtm (Lindsten and Schön, 2012; Whiteley, 2010) – the conditional SMC properties can be preserved by a construction similar to the one we developed in this section for dSMC.*

3.2 Parallel-in-time particle Gibbs

The resulting algorithm resembles the classical conditional SMC algorithm of Andrieu et al. (2010), in that, similarly, we can implement it by simply enforcing that the first trajectory be preserved throughout the course of the recursion. In particular, only Algorithms 1 and 2 need to be modified. The conditional version of Algorithm 2 simply consists in prepending the star trajectory to the sampled proposal trajectories before computing the resulting weights. In other words, for a star trajectory $x_{0:T}^*$, if, for a given t, we are given $N-1$ i.i.d. samples $(X_t^n)_{n=2}^N$ from q_t, we set $X_t^1 = x_t^*$ and then proceed with computing the weights in Algorithm 2 *as if x_t^* had simply been sampled from q_t too.* On the other hand the conditional version of Algorithm 1 consists in preserving said star trajectory throughout the resampling steps and is given by Algorithm 4.

Algorithm 4: Conditional Block combination

`// All operations on indices` m, n `are done in parallel`

Result: Combine conditional particle representation of partial smoothing distributions

Function ConditionalCombine$\big(X_{a:c-1}^{1:N}, w_{c-1}^{1:N}, X_{c:b}^{1:N}, w_c^{1:N}\big)$

Set $W_c^{m,n} \propto \omega_c(X_{c-1}^m, X_c^n) w_{c-1}^m w_c^n$

Sample independently $N-1$ times from $\sum_{m,n} W_c^{m,n} \delta_{X_c^m}(\mathrm{d}x_c)\delta_{X_{c-1}^n}(\mathrm{d}x_{c-1})$ to get $(X_{c-1}^{l_n}, X_c^{r_n})_{1\leq n\leq N}$ `// in parallel, using multinomial resampling`

`// The two loops below can be done in parallel:`

for $c' = a, \ldots, c-1$ **in parallel do**

$\widetilde{X}_{c'}^1 \leftarrow X_{c'}^1$

$\widetilde{X}_{c'}^n \leftarrow X_{c'}^{l_n}$

for $c' = c, \ldots, b$ **in parallel do**

$\widetilde{X}_{c'}^1 \leftarrow X_{c'}^1$

$\widetilde{X}_{c'}^n \leftarrow X_{c'}^{r_n}$

return $\widetilde{X}_{a:b}^{1:N}$

Andrieu et al. (2010) considered implementing the conditional SMC step using a particle filter only, which resulted in lower mixing speeds for time steps further away from the last time step T. This was corrected by the introduction of the so-called backward sampling step (Whiteley, 2010; Lindsten and Schön, 2012), which enabled rejuvenating the conditional trajectories; see also Lindsten et al. (2014) for a related approach. A noteworthy point is that our proposed PIT particle Gibbs algorithm does not suffer from the classical genealogy degeneracy problem that prompted the development of the ancestor sampling step. This is due to the fact that the degeneracy arising in dSMC is essentially uniform across all time steps thanks to the balanced tree structure. Indeed, instead of the last time steps being resampled just a few times and the initial time steps being resampled around T times, as in standard SMC, all time steps in dSMC are resampled at most $\lceil \log_2 T \rceil$ times. This is also the reason why the $\mathcal{L}_p$ error in Proposition 4 scales as a polynomial of T and not exponentially. In practice, this means that the modified trajectories sampled from our conditional dSMC will mix similarly for initial timesteps and for final ones, provided that our proposal distributions q_t and auxiliary weight functions ν_t are adapted to the model and data at hand.

It is worth noting, however, that the backward sampling step of Whiteley (2010) additionally removes the need for scaling the number of particles, N, with the time horizon, T (at the cost of instead increasing the number of MCMC iterations required to converge, Lee et al., 2020). This property is likely not preserved by c-dSMC, and we expect that N needs to increase with T (at least logarithmically) in order to ensure proper ergodicity.

4. Variance reduction methods

A drawback of our method consists in the necessity to use independent proposals $q_{0:T}(\mathrm{d}x_{0:T}) = \prod_{t=0}^{T} q_t(\mathrm{d}x_t)$. It is well known that using such rough estimates increases the variance of the smoothing distribution estimates in particular in case of "sticky" processes which exhibit a strong time-dependency, or more precisely, when the conditional reverse Markov chain representing the smoothing distribution mixes slowly. However, this problem can be mitigated by using proposal distributions that are adapted to the model at hand. In Section 4.1 we describe how recently developed parallel-in-time Gaussian approximation based smoothing algorithms (Särkkä and García-Fernández, 2021; Yaghoobi et al., 2021) can be used to form such proposals. As these methods are also parallel in time, they do not relinquish the $\mathcal{O}(\log T)$ span complexity of the dSMC algorithm.

More prosaically, a natural way to reduce the variance of the smoothing estimators is to increase the number of particles used in the Monte Carlo representations. However, doing so in Algorithm 2.3 comes at a quadratic cost in memory and threads utilization. In Section 4.2 we discuss how we can leverage ideas from Murray et al. (2016) to lazily resample so as to keep a linear memory cost and reduce the computational burden.

4.1 Parallel-in-time Gaussian approximated smoothing solutions

It is well known that non-linear SSMs for which the state posterior distribution is unimodal can be approximated by LGSSMs. For example, consider an additive Gaussian noise transition model $p_t(x_t \mid x_{t-1}) = \mathcal{N}(x_t; f(x_{t-1}), Q_{t-1})\,\mathrm{d}x_t$. Under the Gaussian approximated assumption $p(x_t \mid y_{1:t}) \approx \mathcal{N}(x_t; m_t, P_t)$, we can use a Taylor linearization of the transition function f around the approximated mean m_t to form the linearized dynamics $x_{t+1} = f(m_t) + J[f](m_t)(x_t - m_t) + \epsilon_t$, where ϵ_t is a Gaussian random variable with mean 0 and covariance $J[f](m_t)Q_tJ[f](m_t)^\top$ and $J[f](m_t)$ is the Jacobian of f evaluated at m_t. By repeating this approximation for each time step and for the observation model, we obtain the extended Kalman filter algorithm (Jazwinski, 1970). Similarly, one can use Taylor expansion in order to compute Gaussian approximations of the smoothing distribution marginals $p(x_t \mid y_{1:T})$ for all t, yielding the extended Kalman smoother algorithm. Other linearization techniques exist, such as statistical linearization (Gelb, 1974), sigma-point (unscented) methods (Julier et al., 2000; Särkkä, 2008), and numerical integration based methods (Ito and Xiong, 2000; Särkkä and Hartikainen, 2010). For a review, we refer the reader to Särkkä (2013).

In practice it is worth noting that the reference point used to linearize the system at time t (m_t for the extended Kalman filter example above) is arbitrary, and could be optimized instead of taking the result of the previous time step. This remark led to development of iterated extended Kalman filters (Bell and Cathey, 1993), iterated sigma-point filters (Sibley et al., 2006; Zhan and Wan, 2007), and general iterated statistical linear regression methods called posterior linearization filters (García-Fernández et al., 2015). When considering smoothing problems, it is even better to iteratively linearize with

respect to the smoothing trajectory as is done in the iterated extended Kalman smoother (Bell, 1994). A general framework of iterated posterior linearization smoothers using this idea was developed in García-Fernández et al. (2017) and this was further generalized to more general state-space models in Tronarp et al. (2018). These methods result in Gaussian approximations to the marginals $p(x_t \mid y_{1:T}) \approx \mathcal{N}(x_t; m_t^l, P_t^l)$ which are optimal in a Kullback–Leibler sense (García-Fernández et al., 2015).

Recently, Särkkä and García-Fernández (2021) showed that by reformulating Bayesian filters and smoothers (including Kalman filters and smoothers) in terms of associative operators, it is possible to parallelize them along the time dimension by leveraging prefix-sum algorithms (Blelloch, 1989). This leads to logarithmic span-time complexity $\mathcal{O}(\log_2 T)$ instead of the conventional $\mathcal{O}(T)$ of sequential methods. Yaghoobi et al. (2021) then extended this framework to non-linear models by developing parallelized versions of the iterated extended Kalman smoothers as well as the more general iterated posterior linearization smoothers. This framework allows for computing the marginal approximations $p(x_t \mid y_{1:T}) \approx \mathcal{N}(x_t; m_t^l, P_t^l)$ in the $\mathcal{O}(\log_2 T)$ time complexity.

These Gaussian approximations to the smoothing distributions can now be used as proposal distributions q_t and/or weighting distributions ν_t in the proposed dSMC algorithm. The resulting method with $q_t = \nu_t$ is summarized in Algorithm 5.

Algorithm 5: PIT linearized proposal smoother

Function LinearizedSmoother($y_{1:T}$)
- **for** $t = 0, \ldots, T$ **in parallel do**
  - Initialize $q_t^0 = \mathcal{N}(x_t; m_t^0, P_t^0)$ `// for example, using the stationary distribution`
- Set $l \leftarrow 1$
- **while** *convergence criterion not verified* **do**
  - Linearize (2) around q_t^{l-1}, for $t = 0, 1, \ldots, T$ `// Done in parallel`
  - Run parallel Kalman filter and RTS smoothers on the linearized system as per Särkkä and García-Fernández (2021) or Yaghoobi et al. (2021)
  - Set $p(x_t \mid y_{1:T}) \approx q_t^l = \mathcal{N}(x_t; m_t^l, P_t^l)$, for $t = 0, 1, \ldots, T$ `// Done in parallel`
  - Set $l \leftarrow l + 1$
- Run the parallel smoother defined as per Algorithms 2 and 1

Similarly, we can tweak Algorithm 5 in order to define an efficient Gaussian proposal model for PIT pGibbs. Indeed, between two iterations of the d-cSMC described in Section 3.1, pGibbs typically proposes new parameters and we can expect the parameters of the state-space model to not have changed too much. Intuitively, this means that the optimum trajectory for the parallel IPLS method will not change much and we can therefore

reuse the optimum of the previous Gibbs iteration as initialization for the next one. The benefit of doing so is shown in the experiment of Section 5.2.

4.2 Parallel resampling for lazy evaluation of the weight matrix

Algorithm 1 presented in Section 2.3 requires to form a $N \times N$ matrix to then sample N elements from it. Doing so limits the scalability of dSMC in at least two ways:

1. The memory cost will increase quadratically with the required number of particles. This is particularly problematic on parallel hardware such as GPUs where the memory available is usually more limited than the main (random-access memory) memory accessible via a CPU. For a large number of time steps or particles, our algorithm may therefore simply fail to return a result.

2. The number of threads available on GPUs, while increasing year-on-year, is still limited, and our algorithm computational scalability, although theoretically logarithmic in both N and T, may be affected by threading bottlenecks. See Section 5.1 for an illustration of this.

In order to mitigate both these issues, we can leverage the parallel resampling schemes proposed by Murray et al. (2016). Indeed, these can be modified in order to sample N entries from a set of $N \times N$ unnormalized weights *without needing to evaluate the whole matrix*. This property, although not discussed in Murray et al. (2016) can crucially be utilized to design lazy resampling schemes for our $N \times N$ size importance density (13). Formally, suppose we want to sample N pairs (I_m, J_m) independently from a categorical distribution $\mathrm{Cat}((W_c^{i,j})_{i,j=1}^N)$, where for all i, j, $W_c^{i,j} \propto \omega_c(X_{c-1}^i, X_c^j)$ for some time index c. This can be done in parallel across the N pairs (I_m, J_m) by considering N independent instances of a Metropolis–Hastings (Code 2 in Murray et al., 2016) algorithm with proposal (after proper flattening of the $N \times N$ matrix) $\mathcal{U}(\{1, \dots, N^2\})$ and target $\propto \omega_c(X_{c-1}^i, X_c^j)$. Similarly, when an upper bound $\overline{\omega}_c$ to ω_c is available, an unbiased rejection sampling equivalent (Code 3 in Murray et al., 2016) can be implemented. Under this perspective, we only need to evaluate the term $\omega_c(X_{c-1}^i, X_c^j)$ for the proposed pairs (i, j). This allows us to never increase the memory and thread utilization beyond $\mathcal{O}(N)$ operations at any point in time. On the other hand, this also means that we may inefficiently re-evaluate the same pair several times. However, as shown in Section 5.3, the parallelization makes this trade-off beneficial. For the sake of completeness, we reproduce the resulting resampling algorithms in Appendix B.

Finally, while using these lazy resampling schemes comes at a price (biasedness in the case of the Metropolis–Hastings variation and random execution time in the case of the rejection sampling one), as discussed in Murray et al. (2016, Sections 3.2 and 3.4), this trade-off becomes better as the variance of the weights $\omega_c(X_{c-1}^i, X_c^j)$ decreases.

5. Experiments

In order to illustrate the computational and statistical properties of our proposed methods, we now consider a set of examples from the literature and compare with the sequential counterparts of our methods. All the results were obtained using an Nvidia® GeForce RTX 3090 GPU with 24GB memory and the code to reproduce them can be found at `https://github.com/AdrienCorenflos/parallel-ps`.

5.1 Comparison with FFBS

In this section, we compare dSMC to the classical forward filtering backward sampling (FFBS) algorithm (Godsill et al., 2004), both in terms of execution time and Monte Carlo error. To make the comparison fairer, we also implement FFBS on GPU; in this way, FFBS scale as $\mathcal{O}(T \log N)$ (see, e.g., the prefix-sum implementation of classical resampling operations in Murray et al., 2016), since the particle operations are parallelizable up to a logarithmic factor (corresponding to computing the sum of the importance weights, which can be done using a prefix-sum algorithm). We consider the same model as in Chopin and Singh (2015) (which is a simplified version of the model in Yu and Meng (2011) for photon emission):

$$\begin{aligned} x_0 &\sim \mathcal{N}\left(\mu, \frac{\sigma^2}{1-\rho^2}\right), \\ x_t &= \mu + \rho(\lambda x_{t-1} - \mu) + \epsilon_{t-1}, \quad \epsilon_{t-1} \sim \mathcal{N}(0, \sigma^2), \quad t \geq 1, \\ y_t &\sim \mathcal{P}(\exp(x_t)), \end{aligned} \tag{35}$$

where $\mathcal{P}(\exp(x_t))$ denotes a Poisson distribution with rate $\exp(x_t)$, and we want to estimate its Fisher score with respect to σ^2, and evaluated at σ^2:

$$\begin{aligned} &\mathbb{E}\left[\nabla_{\sigma^2} \ln p(X_{0:T}, y_{0:T}) \mid y_{0:T}\right] \\ &= \mathbb{E}\left[-\frac{T+1}{2\sigma^2} + \frac{1-\rho^2}{2\sigma^4}(X_0 - \mu)^2 + \frac{1}{2\sigma^4}\sum_{s=1}^{T}\left\{X_s - \mu - \rho(X_{s-1} - \mu)\right\}^2 \mid y_{0:T}\right]. \end{aligned} \tag{36}$$

Because of its additive nature, the variance of this expectation should increase as T increases, making it a good benchmark function to test our algorithm.

The stationary distribution of the underlying dynamics is $x_t \sim \mathcal{N}(\mu, \sigma^2/(1-\rho^2))$, so we take $q_t = \nu_t = \mathcal{N}(\mu, \sigma^2/(1-\rho^2))$ for all t.

In order to study the statistical and numerical properties of our algorithm we then generate 50 datasets $x_{0:T}, y_{0:T}$ from the model for $T = 32, 64, 128, 256, 512$ and repeat 100 dSMC and FFBS smoothing experiments on each dataset generated. The statistics we generate are then averaged over the datasets.

The resulting average running times (and 90% confidence intervals) of the corresponding algorithms are shown in Figure 2. Our algorithm is always faster than its sequential

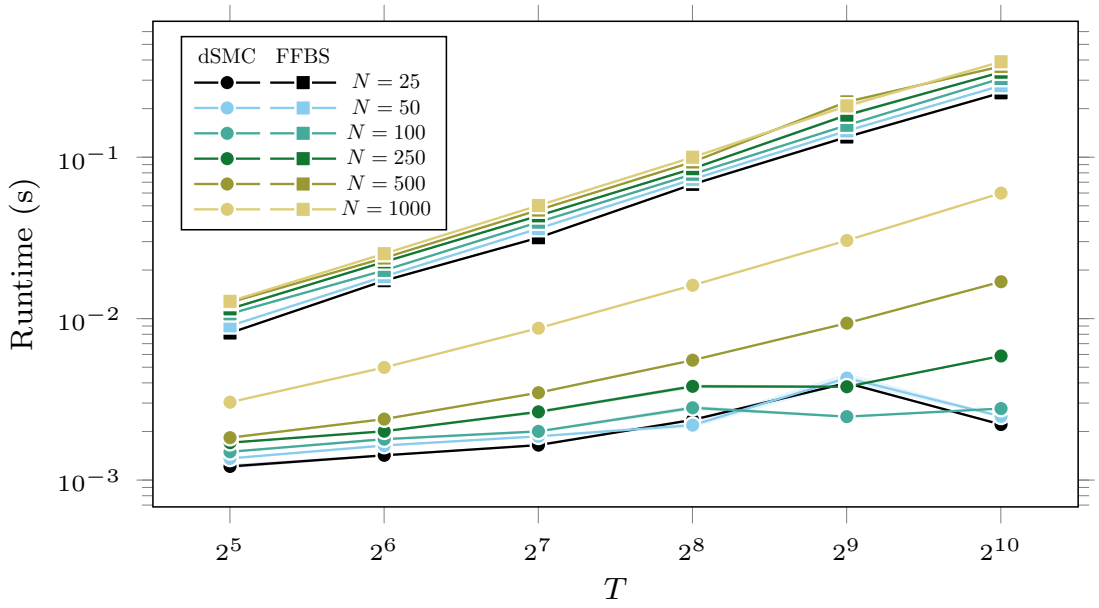

Figure 2: Average clock time of running a sequential FFBS vs dSMC. For T small enough, dSMC scales logarithmically, and then linearly when the parallelization threads have all been utilized. The effect is more pronounced for a higher number of particles. The 90% confidence intervals over the 50 datasets are also reported but hardly visible. The numerical artifact at $T = 2^9$ for $N = 25$ and $N = 50$ is not explained but may be due to the computational framework used.

FFBS counterpart. Due to the limited number of threads on our GPU, the logarithmic complexity scaling of our proposed method reaches a technical upper bound as we increase the number of sampled time steps. In particular the number of time steps that can effectively be parallelized is a decreasing function of the number of particles used. After the parallelization limit has been reached, dSMC scales linearly as further progress is blocked by waiting that a thread becomes free to use.

On the other hand, as can be expected from using independent proposals, our algorithm exhibits a larger error for estimating the Fisher score function, and this error increases with the number of time steps we want to sample. This effect is illustrated by Figure 3 where, for each dataset, we report the average relative error of computing the Fisher score for 100 runs of the particle smoothers, and report the average of this over all the datasets. There therefore exists a natural trade-off between speed and precision, which can be beneficial or not depending on the application. In the next section we show that the increase in variance does not necessarily affect sampling performance in practice.

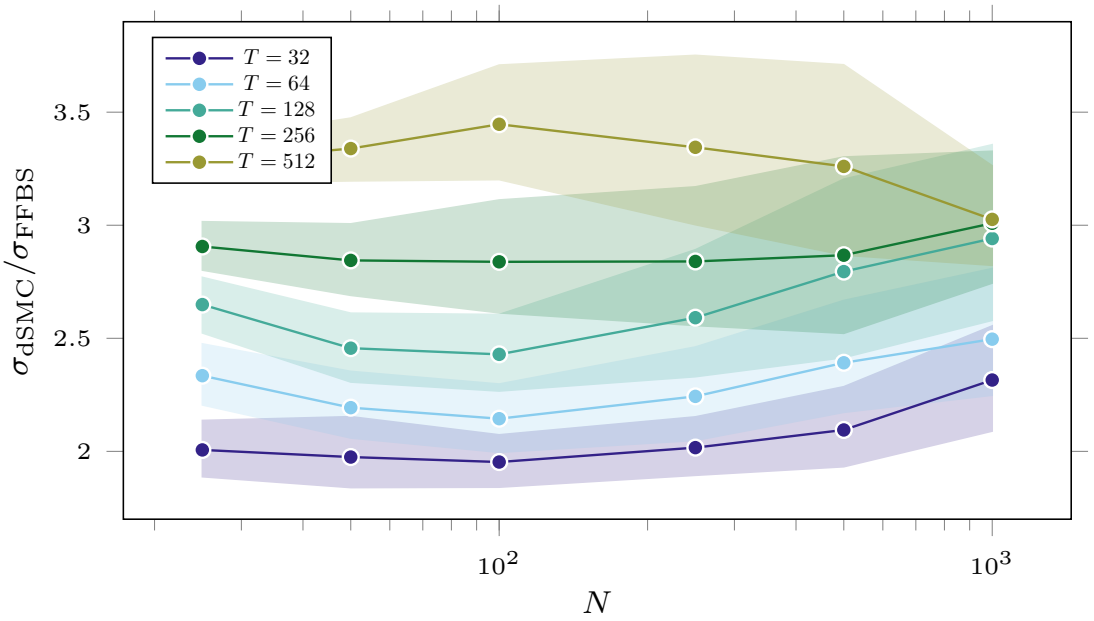

Figure 3: Average relative error (and 90% confidence interval thereof) of running a sequential FFBS vs dSMC. dSMC always exhibits a higher error than FFBS, the ratio between the two increasing as T increases.

5.2 Particle Gibbs sampling of theta-logistic model

The goal of this section is to show how the c-dSMC algorithm can be used to perform particle Gibbs sampling while not reducing its performance compared to the sequential version of cSMC. In order to illustrate the properties of this PIT pGibbs algorithm, we consider the following theta-logistic state-space model:

$$\begin{aligned} &x_0 \sim \mathcal{N}(0,1), \\ &x_t = x_{t-1} + \tau_0 - \tau_1 \exp(\tau_2 x_{t-1}) + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, q^2), t \geq 1 \\ &y_t = x_t + \gamma_t, \quad \gamma_t \sim \mathcal{N}(0, r^2), \quad t \geq 0. \end{aligned} \tag{37}$$

This model was originally proposed by Lande et al. (2003) in order to model population dynamics and has been used as a benchmark for PMCMC methods in, for example, Peters et al. (2010), Chopin and Papaspiliopoulos (2020, Chap. 16). We use the same prior and data (nutria, $T + 1 = 120$) as in these references. We run two sets of experiments: one informed, where Kalman approximations are used, and one uninformed.

For c-dSMC, the "informed proposal" method is defined as taking $q_t = \nu_t$ to be "locally adapted": $q_t(x_t) = p_{\text{EKS}}(x_t \mid y_{0:T}, \tau_0, \tau_1, \tau_2, q, r)$ given by the parallel extended Kalman smoother described in Section 4.1. More precisely, given an initial sample from the prior

$p(\tau_0, \tau_1, \tau_2, q, r)$, we compute the iterated EKS solution with 25 iterations and take the q_t's to be the resulting approximated smoothing marginal. For all subsequent steps, given new parameters, we run a single step of the iterated EKS, starting from the previous iterated EKS approximation, and use the updated Gaussian approximated smoothing marginals as our new proposal distributions q_t's. The "uninformed proposal" on the other hand, is taken to be a Gaussian proposal around the data: $q_t(x_t) = \mathcal{N}(x_t; y_t, r^2 + q^2)$.

For the classical cSMC, the "informed proposal" method is defined as a guided particle filter using the "locally optimal proposal" for (37) (see, e.g., Chopin and Papaspiliopoulos, 2020, Chap. 10.3.2). The "uninformed proposal" on the other hand, is taken to be the bootstrap proposal for the model. We use $N = 50$ particles for both the sequential and PIT versions of cSMC and report the run time of the experiments.

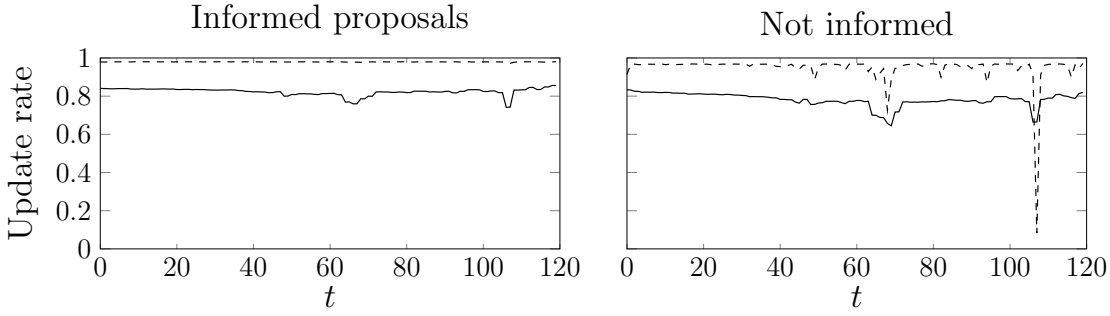

Figure 4: Average update rate of the star trajectory X_t^* for each time t. The average update rate of cSMC with backward sampling (- - -) is higher than that of c-dSMC (——), but in the case of uninformed proposals, the latter is more homogeneous across time steps. Using an adapted proposal marginally improves the resulting update rate in both cases, mostly by smoothing out the "dip" in the model.

In Figure 4, we report the update rate for the sampled trajectory, defined as the empirical probability that the star trajectory X_t^* is updated by running a conditional SMC. It varies between 70% and 80% for the uninformed c-dSMC, and 80% and 85% for the informed version, homogeneously across all time steps *without any explicit backward sampling step*. This is to be compared with the non-uniform renewal rates ($\approx 90\%$) of the uninformed standard pGibbs algorithm, and the almost ideal behavior of the informed standard pGibbs, when a backward sampling step (Whiteley, 2010; Lindsten and Schön, 2012) is implemented.

Moreover, obtaining 10^5 samples from the Gibbs chain took around 400 seconds with both proposal versions of c-dSMC, while it took around 4 000 seconds for both sequential cSMC samplers with backward sampling. Finally, the ACFs (auto-correlation functions) of the Markov chains formed by the parameters posterior samples are virtually identical, as illustrated by Figure 5 (we only report these for the adapted proposals, as there is no major difference with the uninformed ones).

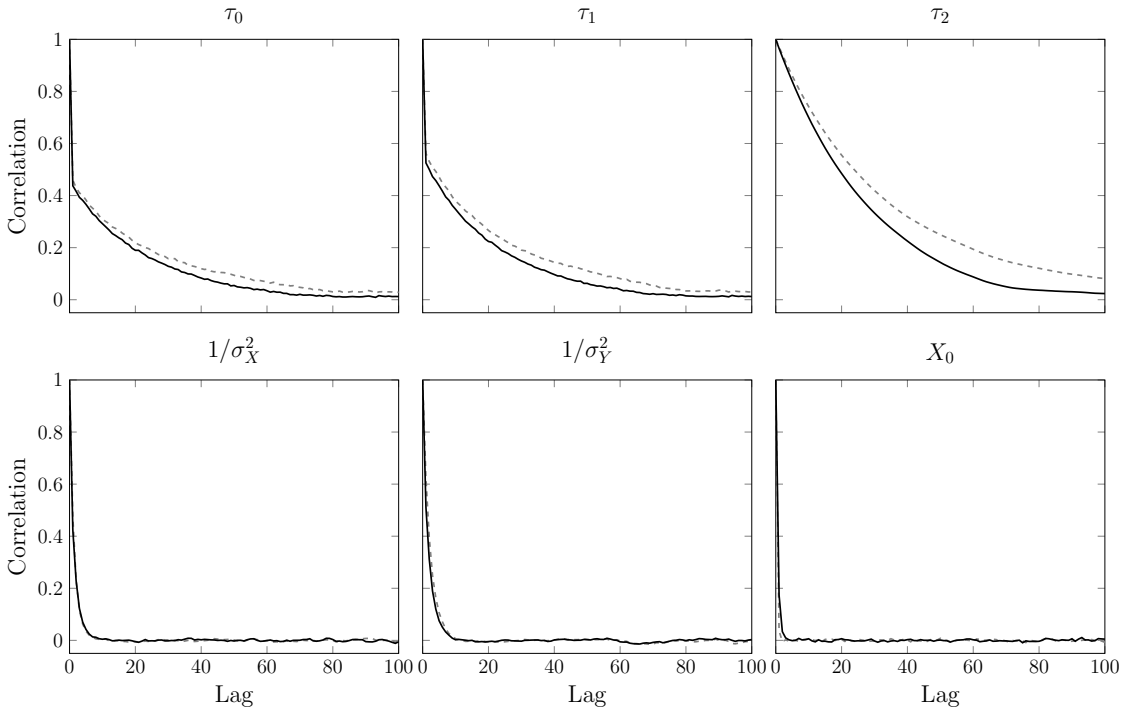

Figure 5: Auto-correlation plots for the parameters τ_0, τ_1, τ_2, $1/\sigma_X^2$, $1/\sigma_Y^2$, and the initial state X_0 posterior samples. Using cSMC with backward sampling (- - -) or c-dSMC (——) results in similar auto-correlation functions for the posterior samples.

5.3 Speed-up and variance reduction via lazy resampling

We now show how the lazy resampling methods introduced in Section 4.2 can help speed up dSMC significantly, while at the same time retaining the same variance as the original method. In order to do so, similarly to Deligiannidis et al. (2020, Section 4.1), we consider a constrained random walk model studied, for example, in Del Moral and Doucet (2004) and Adorisio et al. (2018). While, contrarily to these works, we are not concerned with exact simulation, this model is helpful in understanding the impact of the weights variance on the total runtime and variance of dSMC with lazy resampling. Indeed, the model is controlled by a single parameter σ which represents the noise of the constrained random walk, and directly impacts the variance of the weights in dSMC. Decreasing this parameter will increase the variance of the importance weights, reducing the performance of Monte Carlo methods. This type of behavior is akin to what happens when one increases the dimension of the state (or of the observation). Furthermore, this model is not easily approximated by an LGSSM, and therefore, the variance reduction method of Section 4.1 does not apply here.

Formally, the model is defined as follows:

$$\begin{aligned} &x_0 \sim \mathcal{N}(0,1), \\ &x_t = x_{t-1} + \sigma\epsilon_{t-1}, \quad \epsilon_{t-1} \sim \mathcal{N}(0,1), \end{aligned} \tag{38}$$

and we want to sample from $p(x_{0:T} \mid -1 \leq x_t \leq 1, t = 0, \dots, T)$. This model corresponds to the transition kernel $P_t(\mathrm{d}x_t \mid x_{t-1}) \sim \mathcal{N}(x_{t-1}, \sigma^2)$ with potential function $h_t(x_t) = \mathbb{1}_{[-1,1]}(x_t)$. Following Deligiannidis et al. (2020), we consider the proposal $q_t = \mathcal{U}([-1,1])$; the weights ω_t are then upper-bounded by $(2\pi\sigma)^{-1/2}$. As σ gets higher, we expect the lazy resampling schemes in Section 4.2 to perform better. In order to compare the different smoothing algorithms, we estimate $\mathbb{E}\left[\varphi(X_{0:T}) \mid -1 \leq X_t \leq 1, t = 0, \dots, T\right]$, where

$$\varphi(x_{0:T}) = \log(\sigma) + \frac{1}{\sigma^3}\sum_{t=1}^{T}\left(x_t - x_{t-1}\right)^2, \tag{39}$$

which corresponds (up to a multiplicative constant) to the expected Fisher's score estimate of this model.

For the sake of simplicity, we only consider the rejection version of our lazy resampling methods. Recall that the Metropolis–Hastings version is biased, and thus our convergence theorems do not apply. On the other hand, Murray et al. (2016) find that it works better than the rejection counterpart in all the examples they consider.

In Figure 6, we take σ to be in $\{0.3, 0.4, 0.5\}$, this set being taken to be around the value when using lazy resampling starts to outperform FFBS, and we report the average (over 100 experiments) run times of FFBS, dSMC with systematic resampling (sys-dSMC), and dSMC with rejection-resampling (rs-dSMC) of computing Fisher's score estimates. In Figure 7, we report the respective variance of the resulting 100 score estimates.

For low N's, sys-dSMC is the fastest, with fairly high variance estimates of the Fisher score, as previously discussed in Section 5.1. However, for larger N values, despite its random run time, rs-dSMC completely outperforms both FFBS and sys-dSMC in terms of speed. Moreover, for $\sigma = 0.4$ and $\sigma = 0.5$ and all T's, the slowest running rs-dSMC ($N = 5000$) is faster than the fastest running FFBS ($N = 25$) and exhibits a lower Fisher score estimate variance than FFBS with more particles than $N = 25$. Finally, this improved performance becomes better as the number of time steps T increases, therefore confirming the appeal of dSMC for high values of T.

6. Discussion

In this article we have introduced de-Sequentialized Monte Carlo, the first fully parallel-in-time particle smoother. This algorithm exhibits $\mathcal{L}_p$ error bounds that scale polynomially in the number of times steps and inverse proportionally to the number of particles used. Futhermore, we have shown how one can build a conditional version of dSMC, to be used,

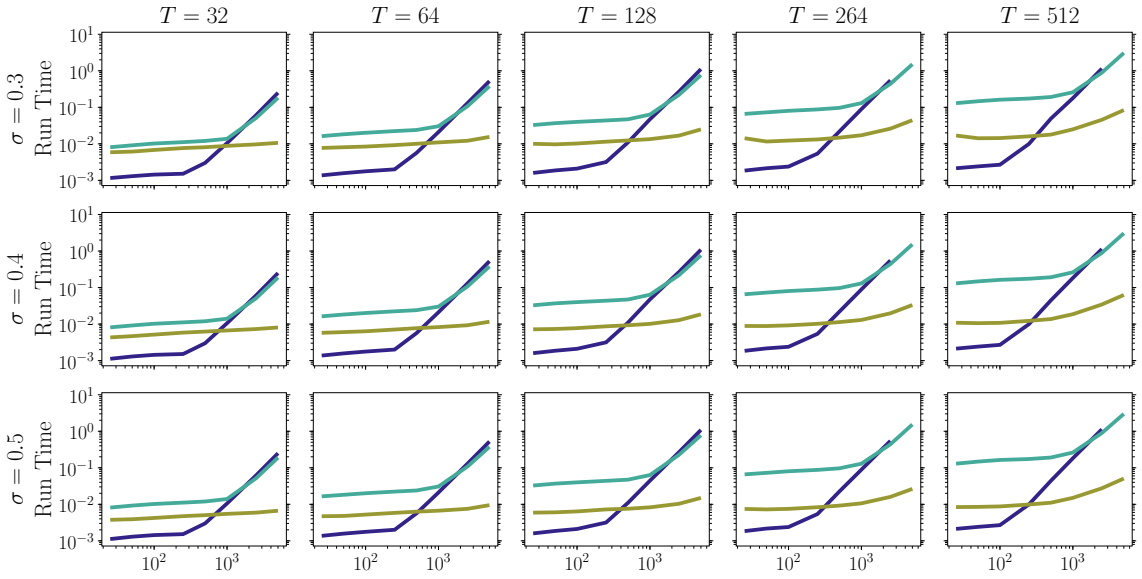

Figure 6: Average run times to compute Fisher's score estimate (39) as a function of the number of particles N for FFBS ( ), sys-dSMC ( ), and rs-dSMC ( ), and different values of T and σ. We can see that for lower variance weights regimes (higher σ), rs-dSMC runs largely faster than both FFBS and sys-dSMC.

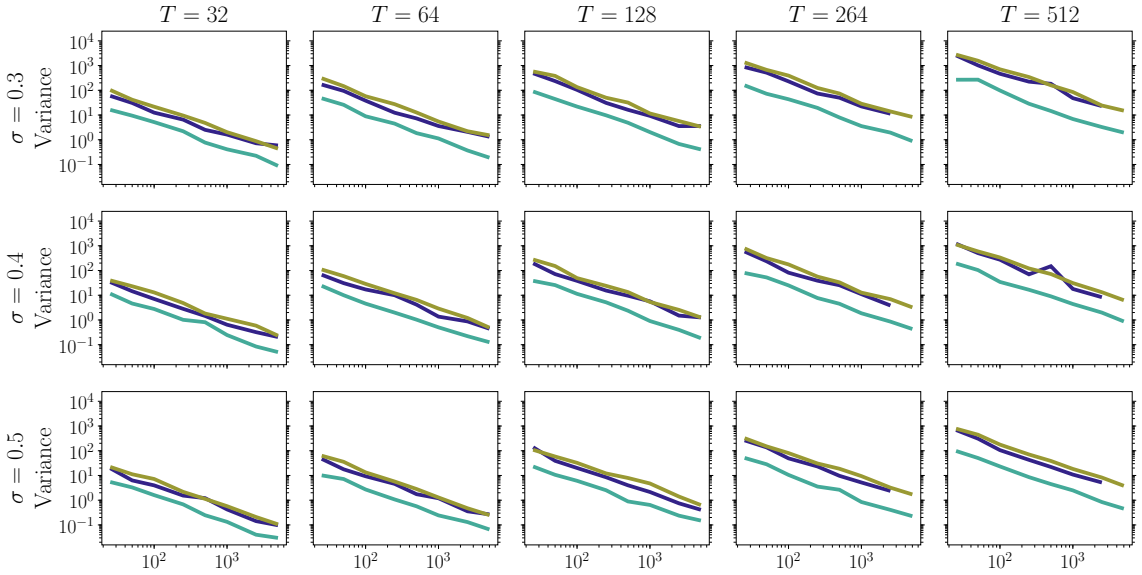

Figure 7: Variances of Fisher's score estimate (39) as a function of the number of particles N for FFBS ( ), sys-dSMC ( ), and rs-dSMC ( ), and different values of T and σ. We can see that rs-dSMC retains roughly the same variance as sys-dSMC throughout.

for example, in particle Gibbs algorithms. Furthermore, we discussed two variance reduction schemes based on parallel-in-time linear Gaussian state-space models approximants, as well as lazy resampling schemes. The resulting algorithms have then been shown to be competitive with standard sequential methods in different non-trivial regimes.

While the Gaussian approximations recover a lot of practical use cases, their nature makes them inadequate to approximate, for example, multi-modal posteriors. Designing proposals with more modeling capacity, and fully utilizing the additional degree of freedom offered by the different roles of ν and q is an important direction of future work. This could be done, for instance, using direct gradient methods (Corenflos et al., 2021; Naesseth et al., 2018; Maddison et al., 2017; Le et al., 2018) or more iterative methods (Guarniero et al., 2017; Heng et al., 2020).

Our parallel smoother exhibits good statistical and computational properties in non-trivial regimes, and allows faster inference at the cost of some precision. The loss of precision comes from the need to use independent proposal distributions, and hinders inference in high variance regimes (such as high dimensional spaces). We believe future research should maybe directed towards using pathwise proposals instead, for example by further leveraging the LGSSM approximants of Yaghoobi et al. (2021).

Because we developed a conditional version of dSMC, our algorithm can be used *mutatis mutandis* within the unbiased coupled smoothing framework of Jacob et al. (2019). While (non-lazy) dSMC exhibits higher variance than its sequential counterparts (for the same number of particles), the framework of Jacob et al. (2019) allows to average independent such estimates to increase the precision of the resulting estimate arbitrarily, making the gain of speed particularly attractive in this context.

Another avenue of future work is to study the ergodic properties of the parallel-in-time particle Gibbs we developed in Section 3, in particular, how one needs to choose the number of particles N as a function of T. We believe that, contrary to conditional SMC with backward sampling (Lee et al., 2020), N needs to increase with T. Understanding the exact relationship between T and N in our case requires a careful examination and is an interesting direction of research.

An important technical limitation of our methodology is the necessity, at each level of the recursion, to explicitly form several $N \times N$ matrices. While this does not impact the theoretical logarithmic properties of our algorithm, this clearly limits the number of particles that we can use in at least two ways: the memory footprint will scale quadratically with it, and the number of threads being limited, a processing bottleneck may appear (as illustrated in Figure 2). We mitigated these issues by utilizing the parallel resampling perspective of Murray et al. (2016) as a lazy resampling scheme, never computing more than N weights at once, which allowed us to improve the scalability of dSMC in the low weights variance regime. We believe that this method can be further improved by using non-uniform proposals on the indices pairs (I, J) to target specific pairs that have a higher *a priori* chance of resulting in a high weight. It was also suggested in Corenflos and Särkkä (2022) that using ensemble techniques in parallel resampling schemes may result in an

improved performance at the cost of a slightly higher memory consumption. Both these extensions deserve more investigation.

On the computational resource perspective, over the years parallel processing hardware have continually increased both the memory and number of threads, so we expect our algorithm to become increasingly competitive in the future. Similarly, it is also possible to distribute the computations across several processors (be it GPUs or CPUs), which in turn would result in making the algorithm scale better with the number of time steps or particles, provided that the communication cost between processors remains limited. Combining this technical solution with the lazy resampling approach of Section 4.2 in particular would likely result in a very competitive smoothers.

Finally, it was recently suggested in Deligiannidis et al. (2020) that it is possible to perform perfect sampling of SSMs smoothing distributions provided we use independent proposals. While our algorithm does not sample exactly from the same proposal distribution, it is our hope that the methods developed here could be applied to sampling from their proposal distribution too, thereby making their sampling algorithm scale logarithmically in time.

Acknowledgments

Adrien Corenflos and Simo Särkkä gratefully acknowledge the support of Academy of Finland (project 321900). The authors would also like to thank Francesca R. Crucinio for spotting a typo in the proof of Proposition 3, as well as the two anonymous referees and the editor for their useful comments and suggestions on this article.

Individiual contributions

The original idea for this article comes from discussions between Adrien Corenflos and Simo Särkkä. The methodology of dSMC was developed by Adrien Corenflos in collaboration with Nicolas Chopin. The pGibbs and lazy resampling extensions are both due to Adrien Corenflos while the LGSSM approximants are jointly due to Simo Särkkä and Adrien Corenflos. The original proofs of this article's results are due to Adrien Corenflos, the convergence rate of Proposition 4 being subsequently improved with the help of Nicolas Chopin. The experimental results are all due to Adrien Corenflos. The first version of this article was written by Adrien Corenflos, after which all authors contributed to the writing.

Appendix A. Parallel combination algorithm

We now reproduce a parallel equivalent to Algorithm 3. It can generally be thought of a divide-and-conquer algorithm akin to prefix-sum algorithms, but not requiring associativity of the operator. Algorithm 6 is phrased in terms of generic operators and elements which, in the particular case of parallel particle smoothing, need to be taken to be, respectively, the operator defined in Algorithm 1 and the set of particles, weights and partial normalizing constants.

Algorithm 6: Generic parallel combination via array reshaping

Result: Combined array

Function ParallelReshapeCombination($Z_{1:K}$, Operator)

Find L such that $2^{L-1} < K \leq 2^L$

for $t = 1, \dots, K$ **in parallel do**

`// Flag that says if we should use the value or not`

$b_t \leftarrow 1$

`// Pad` Z `to the next power of 2 using some NULL value`

for $t = K+1, \dots, 2^L$ **in parallel do**

$Z_t \leftarrow$ NULL

$b_t \leftarrow 0$

for $l = 0, \dots L-1$ **do**

for $n = 1, \dots 2^{L-l}$ **in parallel do**

`/* Join the` Z`'s block by block, this corresponds to reshaping the array and do not result in creating a new array. */`

$Y_n \leftarrow \left[Z_{1+(n-1)2^l}, Z_{2+(n-1)2^l}, \dots Z_{n2^l}\right]$

for $n = 1, \dots 2^{L-l}$ **in parallel do**

`/* Combine the adjacent odd and even` Y`'s if we have not reached the padding threshold, otherwise, just leave the data unchanged. */`

if $b_{1+n2^l} = 1$ **then**

$\left[Z_{1+(n-1)2^l}, Z_{2+(n-1)2^l}, \dots Z_{n2^l}\right], \left[Z_{1+n2^l}, Z_{2+n2^l}, \dots Z_{(n+1)2^l}\right] \leftarrow$ CombinationOperator(Y_n, Y_{n+1})

return $Z_{1:K}$

It is worth noting that Algorithm 6 and Algorithm 3 are not strictly equivalent. This is because the combination operator used for smoothing is random and depends on the state of a random number generator. In fact two reasons make these two algorithm differ:

1. The order in which the nodes at a given depth of Algorithm 3 are handled is arbitrary. Similarly for the order in which we combine adjacent blocks in Algorithm 6.

2. The splitting of Algorithm 6, although corresponding to a balanced tree, will not correspond to the mid-point splitting of Algorithm 3 except when $T+1$ is a power of 2.

However, both algorithms are consistent and can be analyzed by Proposition 4 in the same way.

Appendix B. Lazy resampling algorithms

We now describe the lazy resampling algorithms introduced in Section 4.2. The Metropolis–Hastings version is given by Algorithm 7, while the rejection sampling one is given by Algorithm 8. In Algorithm 7, B is a user defined parameter corresponding to the assumed number of MCMC steps required for the Markov chain to converge to the categorical distribution $\mathrm{Cat}((W_c^{i,j})_{i,j=1}^N)$ (Murray et al., 2016, Section 2.1). Some guidance on how to choose B is provided in Murray et al. (2016), and directly applies to Algorithm 7.

Algorithm 7: Metropolis–Hastings lazy resampling algorithm

Result: Resampling indices (I_m, J_m) for $m = 1, \dots, N$.
Function MHResampling$\big(X_{c-1}^{1:N}, X_c^{1:N}, \omega_c, B\big)$
  for $m = 1, \dots, N$ **in parallel do**
    $(I_m, J_m) \leftarrow (m, m)$
    for $b = 1, \dots, B$ **do**
      Sample $u \sim \mathcal{U}([0,1])$
      Sample $I^*, J^* \sim \mathcal{U}(\{1, \dots, N\})$, independently
      if $u < \omega_c(X_{c-1}^{I^*}, X_c^{J^*}) / \omega_c(X_{c-1}^{I_m}, X_c^{J_m})$ **then**
        $(I_m, J_m) \leftarrow (I^*, J^*)$
  return $I_{1:N}, J_{1:N}$

It is worth noting that, contrarily to Algorithm 3 (Code 3) in Murray et al. (2016), the initial proposal in Algorithm 8 is random and not deterministic. This is because the deterministic starting point of Murray et al. (2016) would result in a bias when subsampling N candidates from the $N \times N$ entries in the weight matrix.

Appendix C. Proof of Proposition 4

For simplicity we only consider the case when $w_{c-1}^n = w_c^n = 1/N$, for all $n \in \{1, \dots, N\}$. The general case follows from the same lines. Using Minkowski's inequality, we have

$$\mathbb{E}\left[\left|\mathbb{Q}_{a:b}(\varphi) - \mathbb{Q}_{a:b}^N(\varphi)\right|^p\right]^{1/p} \leq \mathbb{E}\left[\left|\mathbb{Q}_{a:b}(\varphi) - \widetilde{\mathbb{Q}}_{a:b}^N(\varphi)\right|^p\right]^{1/p} + \mathbb{E}\left[\left|\widetilde{\mathbb{Q}}_{a:b}^N(\varphi) - \mathbb{Q}_{a:b}^N(\varphi)\right|^p\right]^{1/p}. \tag{40}$$

Algorithm 8: Rejection-sampling lazy resampling algorithm

Result: Resampling indices (I_m, J_m) for $m = 1, \dots, N$.
Function RSResampling$\left(X_{c-1}^{1:N}, X_c^{1:N}, \omega_c, \overline{\omega}_c\right)$
  // $\overline{\omega}_c$ is such that $\omega_c(x, y) \leq \overline{\omega}_c$ for all x, y.
  for $m = 1, \dots, N$ **in parallel do**
    Sample I_m, $J_m \sim \mathcal{U}(\{1, \dots, N\})$, independently
    Sample $u \sim \mathcal{U}([0, 1])$
    while $u > \omega_c(X_{c-1}^{I_m}, X_c^{J_m})/\overline{\omega}_c$ **do**
      Sample $I_m, J_m \sim \mathcal{U}(\{1, \dots, N\})$, independently
  return $I_{1:N}, J_{1:N}$

The second term of (40), corresponding to the resampling error, can be controlled as a Monte Carlo error via Del Moral (2004, Lemma 7.3.3). Indeed, let us first notice that we have $\mathbb{E}\left[\mathbb{Q}_{a:b}^N(\varphi) \mid X_{a:b}^{1:N}\right] = \sum_{m,n=1}^N W_c^{m,n} \varphi\left(X_{a:c-1}^m, X_{c:b}^n\right)$, and that, given that we are considering the multinomial resampling case, conditionally on $X_{a:b}^{1:N}$, the variables $(l_n, r_n)_{n=1}^N$ are independent. In this case,

$$\mathbb{E}\left[\left|\widetilde{\mathbb{Q}}_{a:b}^N(\varphi) - \mathbb{Q}_{a:b}^N(\varphi)\right|^p \mid X_{a:b}^{1:N}\right]^{1/p} \leq d(p) \frac{\|\varphi\|_\infty}{N^{1/2}} \tag{41}$$

for some constant $d(p) \leq 2^{(p+1)/p}$, so that the tower law ensures that

$$\mathbb{E}\left[\left|\widetilde{\mathbb{Q}}_{a:b}^N(\varphi) - \mathbb{Q}_{a:b}^N(\varphi)\right|^p\right]^{1/p} \leq 2^{(p+1)/p} \frac{\|\varphi\|_\infty}{N^{1/2}} \tag{42}$$

is verified too.

On the other hand, the first term of (40), corresponding to the self-normalization error, requires more attention. In order to simplify notations, let us introduce the following quantities:

$$\begin{aligned} \widehat{\mathbb{Q}}_{c:b}^N(\varphi) &\coloneqq \frac{1}{N} \sum_{n=1}^N \int \bar{\omega}_c\left(X_{c-1}^n, x_c\right) \varphi\left(X_{a:c-1}^n, x_{c:b}\right) \mathbb{Q}_{c:b}(\mathrm{d}x_{c:b}), \\ \breve{\mathbb{Q}}_{a:b}^N(\varphi) &\coloneqq \frac{1}{N^2} \sum_{m,n=1}^N \bar{\omega}_c\left(X_{c-1}^m, X_c^n\right) \varphi\left(X_{a:c-1}^m, X_{c:b}^n\right). \end{aligned} \tag{43}$$

Using Minkowski's inequality again, twice, we can now decompose the first term of (40) as

$$\mathbb{E}\left[\left|\mathbb{Q}_{a:b}(\varphi) - \widetilde{\mathbb{Q}}_{a:b}^N(\varphi)\right|^p\right]^{1/p} \leq \mathbb{E}\left[\left|\mathbb{Q}_{a:b}(\varphi) - \breve{\mathbb{Q}}_{a:b}^N(\varphi)\right|^p\right]^{1/p} + \mathbb{E}\left[\left|\breve{\mathbb{Q}}_{a:b}^N(\varphi) - \widetilde{\mathbb{Q}}_{a:b}^N(\varphi)\right|^p\right]^{1/p}, \tag{44}$$

so that, splitting once more, we have

$$\begin{aligned}\mathbb{E}\left[\left|\mathbb{Q}_{a:b}(\varphi)-\breve{\mathbb{Q}}^N_{a:b}(\varphi)\right|^p\right]^{1/p} \leq\ & \mathbb{E}\left[\left|\mathbb{Q}_{a:b}(\varphi)-\widehat{\mathbb{Q}}^N_{c:b}(\varphi)\right|^p\right]^{1/p} \\ & + \mathbb{E}\left[\left|\widehat{\mathbb{Q}}^N_{c:b}(\varphi)-\breve{\mathbb{Q}}^N_{a:b}(\varphi)\right|^p\right]^{1/p}.\end{aligned} \tag{45}$$

Let us first remark that

$$\mathbb{Q}_{a:b}(\varphi) = \mathbb{Q}_{a:c-1}\left(x_{a:c-1} \mapsto \int \bar{\omega}_c\left(x_{c-1}, x_c\right)\varphi\left(x_{a:c-1}, x_{c:b}\right)\mathbb{Q}_{c:b}(\mathrm{d}x_{c:b})\right). \tag{46}$$

Then, the integrand $x_{a:c-1} \mapsto \int \bar{\omega}_c\left(x_{c-1}, x_c\right)\varphi\left(x_{a:c-1}, x_{c:b}\right)\mathbb{Q}_{c:b}(\mathrm{d}x_{c:b})$ is upper bounded by $\|\bar{\omega}_c\|_\infty\|\varphi\|_\infty$, so that we can apply the recursion hypothesis to get

$$\mathbb{E}\left[\left|\mathbb{Q}_{a:b}(\varphi)-\widehat{\mathbb{Q}}^N_{c:b}(\varphi)\right|^p\right]^{1/p} \leq C^p_{a:c-1}\|\bar{\omega}_c\|_\infty\frac{\|\varphi\|_\infty}{N^{1/2}}. \tag{47}$$

On the other hand, using the tower law, the second term of (45) becomes

$$\mathbb{E}\left[\left|\widehat{\mathbb{Q}}^N_{c:b}(\varphi)-\breve{\mathbb{Q}}^N_{a:b}(\varphi)\right|^p\right] = \mathbb{E}\left[\mathbb{E}\left[\left|\widehat{\mathbb{Q}}^N_{c:b}(\varphi)-\breve{\mathbb{Q}}^N_{a:b}(\varphi)\right|^p \mid X^{1:N}_{a:c-1}\right]\right]. \tag{48}$$

Noting that

$$\widehat{\mathbb{Q}}^N_{c:b}(\varphi) = \mathbb{Q}_{c:b}\left(x_{c:b} \mapsto N^{-1}\sum_{n=1}^N \bar{\omega}_c\left(X^n_{c-1}, x_c\right)\varphi\left(X^n_{a:c-1}, x_{c:b}\right)\right), \tag{49}$$

and that, for all $n = 1, \ldots, N$ and all $x_{c:b}$, $N^{-1}\sum_{n=1}^N \bar{\omega}_c\left(X^n_{c-1}, x_c\right)\varphi\left(X^n_{a:c-1}, x_{c:b}\right) \leq \|\bar{\omega}_c\|_\infty\|\varphi\|_\infty$, we can leverage the recursion hypothesis one more time to obtain

$$\mathbb{E}\left[\left|\widehat{\mathbb{Q}}^N_{c:b}(\varphi)-\breve{\mathbb{Q}}^N_{a:b}(\varphi)\right|^p \mid X^{1:N}_{a:c-1}\right]^{1/p} \leq C^p_{a:c-1}\|\bar{\omega}_c\|_\infty\frac{\|\varphi\|_\infty}{N^{1/2}} \tag{50}$$

and, applying the tower law again,

$$\mathbb{E}\left[\left|\widehat{\mathbb{Q}}^N_{c:b}(\varphi)-\breve{\mathbb{Q}}^N_{a:b}(\varphi)\right|^p\right]^{1/p} \leq C^p_{a:c-1}\|\bar{\omega}_c\|_\infty\frac{\|\varphi\|_\infty}{N^{1/2}}. \tag{51}$$

This ensures that

$$\mathbb{E}\left[\left|\mathbb{Q}_{a:b}(\varphi)-\breve{\mathbb{Q}}^N_{a:b}(\varphi)\right|^p\right]^{1/p} \leq 2C^p_{a:c-1}\|\bar{\omega}_c\|_\infty\frac{\|\varphi\|_\infty}{N^{1/2}}. \tag{52}$$

Similarly, instead of introducing $\widehat{\mathbb{Q}}^N_{c:b}(\varphi)$, we could have introduced the similar quantity

$$\widehat{\mathbb{Q}}^N_{a:c-1}(\varphi) = \frac{1}{N}\sum_{n=1}^{N}\int \bar{\omega}_c\left(x_{c-1}, X^n_c\right)\varphi\left(x_{a:c-1}, X^n_{c:b}\right)\mathbb{Q}_{a:c-1}(\mathrm{d}x_{a:c-1})$$

to obtain:

$$\mathbb{E}\left[\left|\mathbb{Q}_{a:b}(\varphi) - \breve{\mathbb{Q}}^N_{a:b}(\varphi)\right|^p\right]^{1/p} \leq 2C^p_{c:b}\|\bar{\omega}_c\|_\infty \frac{\|\varphi\|_\infty}{N^{1/2}}. \tag{53}$$

This finally ensures that

$$\mathbb{E}\left[\left|\mathbb{Q}_{a:b}(\varphi) - \breve{\mathbb{Q}}^N_{a:b}(\varphi)\right|^p\right]^{1/p} \leq 2\min(C^p_{a:c-1}, C^p_{c:b})\|\bar{\omega}_c\|_\infty \frac{\|\varphi\|_\infty}{N^{1/2}}. \tag{54}$$

Now the term $\mathbb{E}\left[\left|\breve{\mathbb{Q}}^N_{a:b}(\varphi) - \widetilde{\mathbb{Q}}^N_{a:b}(\varphi)\right|^p\right]^{1/p}$ can be controlled in a way similar to the one used in (Chopin and Papaspiliopoulos, 2020, Lemma 11.2). Indeed we first note that $\widetilde{\mathbb{Q}}^N_{a:b}(\varphi) - \breve{\mathbb{Q}}^N_{a:b}(\varphi) = \widetilde{\mathbb{Q}}^N_{a:b}(\varphi)\left(1 - \breve{\mathbb{Q}}^N_{a:b}(1)\right)$, so that

$$\mathbb{E}\left[\left|\breve{\mathbb{Q}}^N_{a:b}(\varphi) - \widetilde{\mathbb{Q}}^N_{a:b}(\varphi)\right|^p\right]^{1/p} \leq \|\varphi\|_\infty \mathbb{E}\left[\left|1 - \breve{\mathbb{Q}}^N_{a:b}(1)\right|^p\right]^{1/p}. \tag{55}$$

Moreover, $\mathbb{Q}_{a:b}(1) = 1$ by definition, so that we can rewrite

$$\mathbb{E}\left[\left|1 - \breve{\mathbb{Q}}^N_{a:b}(1)\right|^p\right]^{1/p} = \mathbb{E}\left[\left|\mathbb{Q}_{a:b}(1) - \breve{\mathbb{Q}}^N_{a:b}(1)\right|^p\right]^{1/p} \tag{56}$$

which can be bounded similarly to (54), giving

$$\mathbb{E}\left[\left|1 - \breve{\mathbb{Q}}^N_{a:b}(1)\right|^p\right]^{1/p} \leq 2\min(C^p_{a:c-1}, C^p_{c:b})\|\bar{\omega}_c\|_\infty \frac{1}{N^{1/2}}. \tag{57}$$

This results in the following inequality

$$\mathbb{E}\left[\left|\breve{\mathbb{Q}}^N_{a:b}(\varphi) - \widetilde{\mathbb{Q}}^N_{a:b}(\varphi)\right|^p\right]^{1/p} \leq 2\min(C^p_{a:c-1}, C^p_{c:b})\|\bar{\omega}_c\|_\infty \frac{\|\varphi\|_\infty}{N^{1/2}}. \tag{58}$$

Putting everything together, we obtain

$$\mathbb{E}\left[\left|\mathbb{Q}_{a:b}(\varphi) - \mathbb{Q}^N_{a:b}(\varphi)\right|^p\right]^{1/p} \leq \left(4\min(C^p_{a:c-1}, C^p_{c:b})\|\bar{\omega}_c\|_\infty + 2^{(p+1)/p)}\right)\frac{\|\varphi\|_\infty}{N^{1/2}}. \tag{59}$$

References

M. Adorisio, A. Pezzotta, C. de Mulatier, C. Micheletti, and A. Celani. Exact and efficient sampling of conditioned walks. *Journal of Statistical Physics*, 170(1):79–100, 2018.

L. Aitchison. Tensor Monte Carlo: Particle methods for the GPU era. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

C. Andrieu, A. Doucet, and E. Punskaya. Sequential Monte Carlo methods for optimal filtering. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, pages 79–95. Springer New York, New York, NY, 2001.

C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

I. Arasaratnam and S. Haykin. Cubature Kalman filters. *IEEE Transactions on Automatic Control*, 54(6):1254–1269, 2009.

Y. Bar-Shalom, X.-R. Li, and T. Kirubarajan. *Estimation with Applications to Tracking and Navigation*. Wiley, New York, NY, 2001.

B. M. Bell. The iterated Kalman smoother as a Gauss–Newton method. *SIAM Journal on Optimization*, 4(3):626–636, 1994.

B. M. Bell and F. W. Cathey. The iterated Kalman filter update as a Gauss–Newton method. *IEEE Transactions on Automatic Control*, 38(2):294–297, 1993.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: a review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

G. E. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11):1526–1538, 1989.

N. Chopin and O. Papaspiliopoulos. *An Introduction to Sequential Monte Carlo*. Springer International Publishing, 2020.

N. Chopin and S. S. Singh. On particle Gibbs sampling. *Bernoulli*, 21(3):1855–1883, 2015.

A. Corenflos and S. Särkkä. The coupled rejection sampler. *arXiv preprint arXiv:2201.09585 (version 1)*, 2022.

A. Corenflos, J. Thornton, G. Deligiannidis, and A. Doucet. Differentiable particle filtering via entropy-regularized optimal transport. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2100–2111. PMLR, 2021.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

D. Crisan and A. Doucet. Convergence of sequential Monte Carlo methods. *Signal Processing Group, Department of Engineering, University of Cambridge, Technical Report CUEDIF-INFENGrrR38*, 1, 2000.

P. Del Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Springer New York, New York, NY, 2004.

P. Del Moral and A. Doucet. Particle motions in absorbing medium with hard and soft obstacles. *Stochastic Analysis and Applications*, 22(5):1175–1207, 2004.

P. Del Moral and A. Guionnet. On the stability of interacting processes with applications to filtering and genetic algorithms. *Annales de l'Institut Henri Poincaré (B) Probability and Statistics*, 37(2):155–194, 2001.

G. Deligiannidis, A. Doucet, and S. Rubenthaler. Ensemble rejection sampling. *arXiv preprint arXiv:2001.09188 (version 1)*, 2020.

D. Ding and A. Gandy. Tree-based particle smoothing algorithms in a hidden Markov model. *arXiv preprint arXiv:1808.08400 (version 1)*, 2018.

A. Doucet, S. J. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000.

G. Evensen. The ensemble Kalman filter: Theoretical formulation and practical implementation. *Ocean dynamics*, 53(4):343–367, 2003.

Á. F. García-Fernández, L. Svensson, M. R. Morelande, and S. Särkkä. Posterior linearization filter: principles and implementation using sigma points. *IEEE Transactions on Signal Processing*, 63(20):5561–5573, 2015.

Á. F. García-Fernández, L. Svensson, and S. Särkkä. Iterated posterior linearization smoother. *IEEE Transactions on Automatic Control*, 62(4):2056–2063, 2017.

A. Gelb. *Applied Optimal Estimation*. MIT press, 1974.

S. J. Godsill, A. Doucet, and M. West. Monte Carlo smoothing for nonlinear time series. *Journal of the American Statistical Association*, 99(465):156–168, 2004.

N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEEE Proceedings on Radar and Signal Processing*, volume 140, pages 107–113, 1993.

P. Guarniero, A. M. Johansen, and A. Lee. The iterated auxiliary particle filter. *Journal of the American Statistical Association*, 112(520):1636–1647, 2017.

S. Hassan, S. Särkkä, and A. F. García-Fernández. Temporal parallelization of inference in hidden Markov models. *IEEE Transactions on Signal Processing*, 69:4875–4887, 2021.

J. Heng, A. N. Bishop, G. Deligiannidis, and A. Doucet. Controlled sequential Monte Carlo. *The Annals of Statistics*, 48(5):2904–2929, 2020.

K. Ito and K. Xiong. Gaussian filters for nonlinear filtering problems. *IEEE Transactions on Automatic Control*, 45(5):910–927, 2000.

P. E. Jacob, F. Lindsten, and T. B. Schön. Smoothing with couplings of conditional particle filters. *Journal of the American Statistical Association*, 2019.

A. H. Jazwinski. *Stochastic Processes and Filtering Theory*. Academic Press, New York, NY, 1970.

S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte. A new method for the nonlinear transformation of means and covariances in filters and estimators. *IEEE Transactions on Automatic Control*, 45(3):477–482, 2000.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82 (Series D):35–45, 1960.

J. Kuntz, F. R. Crucinio, and A. M. Johansen. Product-form estimators: exploiting independence to scale up Monte Carlo. *arxiv preprint arXiv:2102.11575 (version 3)*, 2021a.

J. Kuntz, F. R. Crucinio, and A. M. Johansen. The divide-and-conquer sequential Monte Carlo algorithm: theoretical properties and limit theorems. *arXiv preprint arXiv:2110.15782 (version 1)*, 2021b.

R. Lande, S. Engen, and B.-E. Saether. *Stochastic population dynamics in ecology and conservation*. Oxford University Press on Demand, 2003.

T. A. Le, M. Igl, T. Rainforth, T. Jin, and F. Wood. Auto-encoding sequential Monte Carlo. In *ICLR*, 2018.

A. Lee, C. Yau, M. B. Giles, A. Doucet, and C. C. Holmes. On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. *Journal of Computational and Graphical Statistics*, 19(4):769–789, 2010.

A. Lee, S. S. Singh, and M. Vihola. Coupled conditional backward sampling particle filter. *The Annals of Statistics*, 48(5):3066–3089, 2020.

F. Lindsten and T. B. Schön. On the use of backward simulation in the particle Gibbs sampler. In *Proceedings of the 37th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Kyoto, Japan*, 2012.

F. Lindsten, M. I. Jordan, and T. B. Schön. Particle Gibbs with ancestor sampling. *Journal of Machine Learning Research*, 15:2145–2184, 2014.

F. Lindsten, A. M. Johansen, C. A. Naesseth, B. Kirkpatrick, T. B. Schön, J. Aston, and A. Bouchard-Côté. Divide-and-conquer with sequential Monte Carlo. *Journal of Computational Statistics and Graphics*, 26:445–458, 2017.

P. M. Lyster, S. E. Cohn, R. Ménard, L.-P. Chang, S.-J. Lin, and R. G. Olsen. Parallel implementation of a Kalman filter for constituent data assimilation. *Monthly Weather Review*, 125(7):1674–1686, 1997.

C. J. Maddison, D. Lawson, G. Tucker, N. Heess, M. Norouzi, A. Mnih, A. Doucet, and Y. W. Teh. Filtering variational objectives. In *Advances in Neural Information Processing Systems*, 2017.

L. Middleton, G. Deligiannidis, A. Doucet, and P. E. Jacob. Unbiased smoothing using particle independent Metropolis-Hastings. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2378–2387. PMLR, 2019.

S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann, 1997.

L. M. Murray, A. Lee, and P. E. Jacob. Parallel resampling in the particle filter. *Journal of Computational and Graphical Statistics*, 25(3):789–805, 2016.

C. A. Naesseth, S. W. Linderman, R. Ranganath, and D. M. Blei. Variational sequential Monte Carlo. In *AISTATS*, 2018.

G. W. Peters, G. R. Hosack, and K. R. Hayes. Ecological non-linear state space model selection via adaptive particle Markov chain Monte Carlo (AdPMCMC). *arXiv preprint arXiv:1005.2238 (version 1)*, 2010.

L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

H. E. Rauch, F. Tung, and C. T. Striebel. Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 3(8):1445–1450, 1965.

K. F. Riley, M. P. Hobson, and S. J. Bence. *Mathematical Methods for Physics and Engineering: A Comprehensive Guide*. Cambridge University Press, 2006.

O. Rosen and A. Medvedev. Efficient parallel implementation of state estimation algorithms on multicore platforms. *IEEE Transactions on Control Systems Technology*, 21(1):107–120, 2013.

S. Särkkä. Unscented Rauch-Tung-Striebel smoother. *IEEE Transactions on Automatic Control*, 53(3):845–849, 2008.

S. Särkkä. *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.

S. Särkkä and Á. F. García-Fernández. Temporal parallelization of Bayesian smoothers. *IEEE Transactions on Automatic Control*, 66(1):299–306, 2021.

S. Särkkä and J. Hartikainen. On Gaussian optimal smoothing of non-linear state space models. *IEEE Transactions on Automatic Control*, 55(8):1938–1941, 2010.

G. Sibley, G. S. Sukhatme, and L. H. Matthies. The iterated sigma point Kalman filter with applications to long range stereo. *Robotics: Science and Systems*, 8(1):235–244, 2006.

S. S. Singh, F. Lindsten, and E. Moulines. Blocking strategies and stability of particle Gibbs samplers. *Biometrika*, 104(4):953–969, 2017.

F. Tronarp, Á. F. García-Fernández, and S. Särkkä. Iterative filtering and smoothing in nonlinear and non-Gaussian systems using conditional moments. *IEEE Signal Processing Letters*, 25(3):408–412, 2018.

N. Whiteley. Discussion of 'Particle Markov chain Monte Carlo methods' by Andrieu et al. *J. R. Statist. Soc. B*, 72(3):306–307, 2010.

F. Yaghoobi, A. Corenflos, S. Hassan, and S. Särkkä. Parallel iterated extended and sigma-point Kalman smoothers. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5350–5354, 2021.

Y. Yu and X.-L. Meng. To center or not to center: that is not the question—an ancillarity-sufficiency interweaving strategy (ASIS) for boosting MCMC efficiency. *Journal of Computational and Graphical Statistics*, 20(3):531–570, 2011.

R. Zhan and J. Wan. Iterated unscented Kalman filter for passive target tracking. *IEEE Transactions on Aerospace and Electronic Systems*, 43(3):1155–1163, 2007.