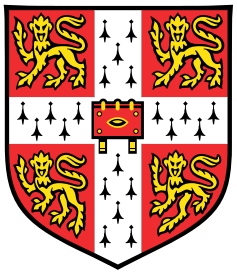
Monte Carlo Methods in Practice and Efficiency Enhancements via Parallel Computation



Alix Marie d'Avigneau

Supervisor: Dr Sumeetpal Singh

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Queens' College October 2020

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. I further state that no substantial part of my thesis has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee

Alix Marie d'Avigneau
October 2020

Abstract

Monte Carlo Methods in Practice and Efficiency Enhancements via Parallel Computation

Alix Marie d'Avigneau

Monte Carlo methods are crucial when dealing with advanced problems in Bayesian inference. Indeed, common approaches such as Markov chain Monte Carlo (MCMC) and sequential Monte Carlo (SMC) can be endlessly adapted to tackle the most complex problems. What is important then is to construct efficient algorithms, and significant attention in the literature is devoted to developing algorithms that mix well, have low computational complexity and can scale up to large datasets. One of the most commonly used and straightforward approaches is to speed up Monte Carlo algorithms by running them in parallel computing environments. The compute time of Monte Carlo algorithms is random and can vary depending on the current state of the Markov chain. Other computing-infrastructure related factors, such as competing jobs on the same processor, or memory bandwidth, which are prevalent in shared computing architectures such as cloud computing, can also affect this compute time. However, many algorithms running in parallel require the processors to communicate every so often, and for that we must ensure that they are simultaneously ready and any idle wait time is minimised. This can be done by employing a framework known as Anytime Monte Carlo, which imposes a real-time deadline on parallel computations.

The contributions in this thesis include novel applications of the Anytime framework to construct efficient Anytime MCMC and SMC algorithms which make use of parallel computing in order to perform inference for advanced problems. Examples of such problems investigated include models in which the likelihood cannot be evaluated analytically, and changepoint models, which are often used to model the heterogeneity of sequential data, but tricky to infer upon due to the unknown number and locations of the changepoints. This thesis also focuses on the difficult task of performing parameter inference in single-molecule microscopy, a category of models in which the arrival rate of observations is not uniformly distributed and measurement models have complex forms. These issues are exacerbated when molecules have trajectories described by stochastic differential equations.

The original contributions of this thesis are organised in Chapters 4-6. Chapter 4 shows the development of a novel Anytime parallel tempering algorithm and demonstrates the performance enhancements the Anytime framework brings to parallel tempering, an algorithm, which runs multiple interacting MCMC chains in order to more efficiently explore the state space. In Chapter 5, a general Anytime SMC sampler is developed for performing changepoint inference using reversible jump MCMC (RJ-MCMC), an algorithm that takes into account the unknown number of changepoints by including transdimensional MCMC updates. The workings of the algorithm are illustrated on a particularly complex changepoint model, and once again the improvements in performance brought by employing the Anytime framework are demonstrated. Chapter 6 moves away from the Anytime framework, and presents a novel and general SMC approach to performing parameter inference for molecules with stochastic trajectories.

À Maman et Papa …

Acknowledgements

First of all, I would like to thank my supervisor, Dr Sumeetpal Singh, for his invaluable help and guidance throughout these four years. I'd also like to thank my collaborators Dr Lawrence Murray and Prof Raimund Ober for their helpful insights and contributions to some of the most exciting subjects in this thesis.

I would also like to thank my examiners, Dr Ed Cohen and Prof Simon Godsill, for providing very helpful insights and comments that improved the quality of this thesis.

I've thoroughly enjoyed being part of the SigProc lab throughout my PhD. Thank you to IT whiz and excellent human Dr Phill Richardson for showing me the ropes at the start of my PhD and staying my friend even after leaving the lab. Thank you also to my fourth year neighbour Kuan Hsieh for all the chats trying to make our respective machines work faster, without which some of my experiments may have taken years to complete!

I'd like to thank the members of my Monte Carlo reading group Sam Duffield, Sam Power, Torben Sell and Jacob Vorstrup. I may be on the quiet side but nobody can deny how interesting and helpful these weekly meetings are! I hope we keep meeting for a long time.

Of course, these last few pandemic-filled, thesis-writing months would have been a lot bleaker if I hadn't had the privilege of sharing the house with resident plant, calligraphy and obscure etymology expert Louis Wilson.

Et enfin je voudrais remercier ma famille, et surtout mes parents Roger et Valérie Marie d'Avigneau, qui ont toujours été là pour moi et m'ont soutenue durant toutes mes études. J'ai beaucoup de chance d'être votre fille.

Table of contents

List of figures

List of tables

List of algorithms

Nomenclature

Roman Symbols

$\mathrm{Cov}[\cdot]$	Covariance
$\mathbb{E}[\cdot]$	Expectation
$\mathbb{I}$	Identity matrix. Variant: $\mathbb{I}_{n\times n}$ identity matrix of dimensions $n \times n$
J_a	Bessel function of the first kind of order a
$\mathbb{P}[\cdot]$	Probability

Distributions

$\mathrm{Exp}(\lambda)$	Exponential distribution with mean λ^{-1}
$\mathrm{Gamma}(\alpha,\beta)$	Gamma distribution with shape α and scale β
$\mathrm{Gamma}^{-1}(\alpha,\beta)$	Inverse Gamma distribution with shape α and scale β
$\mathscr{N}(\mu,\Sigma)$	Multivariate Gaussian distribution with mean vector μ and covariance matrix Σ
$\mathscr{N}(\mu,\sigma^2)$	Gaussian distribution with mean μ and variance σ^2
$T\mathscr{N}(\mu,\Sigma)$	Truncated Gaussian distribution with mean vector μ and covariance matrix Σ
$\mathscr{U}_A$	Uniform distribution over the set A

Greek Symbols

$\delta_{x_0}(x)$	Dirac delta mass located at x_0
$\Gamma(z)$	gamma function, i.e. $\Gamma(z) = \int_0^\infty x^{z-1}e^{-x}\mathrm{d}x$

Superscripts

(i) particle index

$(1:N)$ particle population, i.e. $X^{(1:N)} = \left\{X^{(i)}\right\}_{i=1}^{N}$

$\intercal$ matrix transpose

Other Symbols

$|a|$ absolute value of $a \in \mathbb{R}$

$[\![m,n]\!]$ $= m, m+1, \ldots, n-1, n$ if $n \leq m$ and $m, m-1, \ldots, n+1, n$ otherwise

$\lceil a \rceil$ ceiling function of $a \in \mathbb{R}$

$\|x\|^2$ vector norm, i.e. $\|x\|^2 = x^\intercal x$

$\mathbb{1}_E$ indicator function of event E

$\iota \sim \mathbb{P}\left(\omega^{(1:N)}\right)$ sample ι from the index set $[\![1,N]\!]$ with probabilities proportional to $\omega^{1:N}$

Acronyms / Abbreviations

ABC Approximate Bayesian computation

AMPF Auxiliary marginal particle filter

APF Auxiliary particle filter

APTMC Anytime parallel tempering Monte Carlo

CPU Central Processing Unit

EM Expectation-maximization

ESS Effective sample size

FFBSi Forward-filtering backward simulation

FFBSm Forward-filtering backward smoothing

FIM Fisher information matrix

GPU Graphics Processing Unit

IAT Integrated autocorrelation time

i.i.d independently and identically distributed

MA(q) Moving average process of order q

MCMC Markov chain Monte Carlo

ML Maximum likelihood

MPF Marginal particle filter

OIM Observed information matrix

PaRIS Particle-based rapid incremental smoother

RJ-MCMC Reversible jump Markov chain Monte Carlo

SDE Stochastic differential equation

SIR Sequential importance resampling

SIS Sequential importance sampling

SMC Sequential Monte Carlo

Chapter 1

Introduction

1.1 Monte Carlo methods

MONTE CARLO methods, coined as such in 1947 by N. Metropolis (Metropolis [1987]) and developed by S. Ulam and J. von Neumann (Eckhardt [1987]), are a widely used class of computational algorithms which rely on repeated random sampling to approximate quantities which are unavailable analytically. Notable applications include numerical integration and, as a direct result, estimating expectations. In this thesis, we focus on their application to Bayesian inference (Geman and Geman [1984]).

1.1.1 Developing efficient Monte Carlo methods for Bayesian inference

Consider a set of $\mathscr{Y}$-valued observations Y following a probability model with underlying states $X \in \mathscr{X}$ and associated *likelihood* $g(y|x)$. The unobserved random variables X describing the data Y are unknown with *prior* density $p(x)$. The aim of Bayesian inference is to use the information contained in the prior – representing prior belief – and the likelihood – representing evidence from the data – to obtain the *posterior* density $\pi(x) := p(x|y)$ of the parameters:

$$\pi(x) = \frac{p(x)g(y|x)}{\int_{\mathscr{X}} p(x)g(y|x)\mathrm{d}x}.$$

Except for the simplest cases, such as when the state space $\mathscr{X}$ is finite, or if the prior and likelihood are conjugate, this prior is intractable. Fortunately, as long as it is possible to sample from the posterior, we can obtain a Monte Carlo approximation of it. A vast array of methods have been developed in order to sample from the posterior. For example, in the case of *perfect Monte Carlo*, when it is possible to obtain i.i.d samples from π, its approximation

is given by

$$\hat{\pi}(x) = \frac{1}{N}\sum_{i=1}^{N} \delta_{X^{(1:N)}}(x).$$

This is the case of *inversion* and *rejection sampling* presented in Eckhardt [1987]. Being able to sample directly from π by inversion is unfortunately rare, and rejection sampling is often extremely inefficient. Other approaches have been developed such as *importance sampling* a variance reduction technique in which samples drawn from π are not i.i.d, but a Monte Carlo approximation can be constructed by assigning a weight to each sample. However, controlling the variance of importance sampling estimates is difficult, so the approach most often used to sample from an intractable posterior, and one of the main focus points of this thesis, is *Markov chain Monte Carlo* (MCMC) (Gilks et al. [1996]), presented in Section 2.3, in which a Markov chain is constructed whose invariant or target distribution is π.

An important issue that must then be addressed is the efficiency of MCMC algorithms. In other words, how fast is the Markov chain able to converge to its stationary distribution? Often, basic MCMC algorithms such as *Metropolis-Hastings* (Hastings [1970]; Metropolis et al. [1953]; Peskun [1981, 1973]) and Gibbs sampling (Gelfand and Smith [1990]; Geman and Geman [1984]) are slow to converge, and prone to getting stuck in local maxima when the target of interest is multimodal. An overview of methods to accelerate MCMC is given in Section 2.4, and includes tempering algorithms, which enable a more efficient exploration of the state space when constructing the Markov chain.

1.1.2 Sequential Monte Carlo inference

When dealing with sequential data such as in state space models, illustrated in Figure 1.1, which model dependencies between an unobserved or hidden latent process $(X_t)_{t=1}^{\infty}$, and the observed data $(Y_t)_{t=1}^{\infty}$ (see Section 3.2 for an introduction to state space models), MCMC can be particularly ineffective, as it involves evaluating the likelihood for the whole data at every iteration, which quickly becomes computationally expensive for state space models as the size of the data increases. Examples of state space models explored in this thesis include changepoint models (Chapter 5) and the fundamental data model for single-molecule microscopy (Chapter 6). Additionally, if a new data point arrives, the algorithm needs to be recomputed entirely, which adds to the computational cost. Another disadvantage of employing MCMC to approximate the posterior is the need to assess the convergence of the Markov chain to the desired distribution.

Fortunately, a class of Monte Carlo algorithms known as *sequential Monte Carlo* (SMC) or *particle filters*, described in Chapter 3, has been developed that allows for a sequential

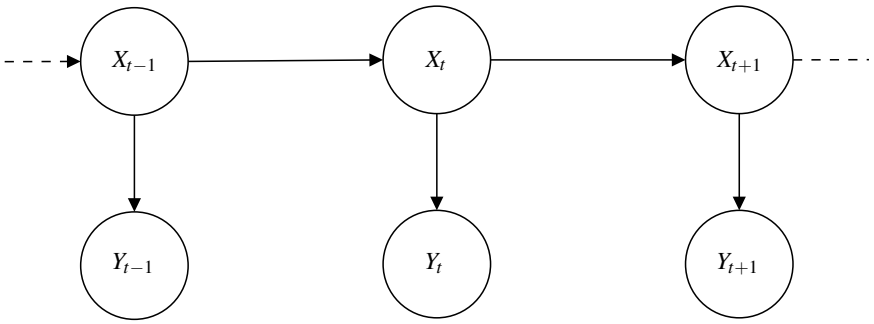


Fig. 1.1 Directed acyclic graph (DAG) for a state space model. The unobserved latent process $(X_t)_{t=1}^{\infty}$ constitutes a Markov chain and the observed process $(Y_t)_{t=1}^{\infty}$ is assumed to be conditionally independent given the latent process, i.e. $g(y_t|x_t, y_{-t}) = g(y_t|x_t)$ for $t = 1, 2, \ldots$ (see Section 3.2).

estimation of the posterior of interest. In SMC, a population of weighted particles initialised at $t = 1$ is propagated through the steps $t = 2, 3, \ldots$, and updated as new data points arrive, providing via particle filtering (Section 3.3) and smoothing (Section 3.4) algorithms a particle approximation of the posterior $\pi(x_{1:t})$ at step t.

To efficiently perform accurate inference, many common issues need to be addressed in SMC. Firstly, a resampling step must be included in order to avoid *weight degeneracy*, which as the algorithm progresses leads to the posterior being approximated by a single particle with weight one while all the others have weight zero. Secondly, while the resampling step is crucial, it introduces particle *path degeneracy*, in which the number of unique values representing the marginal at any fixed step t quickly falls to one as the particle smoother progresses (see Douc et al. [2014], Example 11.2), thus leading to increased variance in estimates. Finally, each step of an SMC algorithm should not be too computationally expensive, as this can render the algorithm very inefficient. The ultimate goal in SMC is to strike a good balance between addressing degeneracy issues and keeping the algorithm complexity low, and ideally linear. Chapter 6 deals with some of these issues.

1.2 Efficient use of parallel computing

Monte Carlo methods lend themselves well to parallel or distributed computing architectures (Murray [2010]; Wilkinson [2006]), from straightforward algorithms run on multiple CPU *processors* or *workers* to advanced experiments making use of the large number of processors provided by GPUs (Lee et al. [2010]). Several MCMC algorithms that make use of parallel computing have been developed in recent years. Some of these are known as *divide-and-conquer* algorithms, such as embarrassingly parallel MCMC (Neiswanger et al. [2013])

and consensus Monte Carlo (Scott et al. [2016]), and are particularly suitable for big data applications in which the data are divided into batches and MCMC is run on each batch separately before combining the results. *Population-based* algorithms (Laskey and Myers [2003]; Liang et al. [2011]) have also been developed, in which a population of MCMC chains are run in parallel and occasionally interact in order to improve convergence to the target distribution. An example of such algorithms, described in Section 2.4.1 and heavily featured in Chapter 4, is *parallel tempering* (Earl and Deem [2005]; Geyer [1991]; Swendsen and Wang [1986]). Similarly, the population-based nature of SMC algorithms makes them easily adaptable to parallel computing.

It must, however, be noted that in the majority of the Monte Carlo algorithms considered, while most updates are performed independently in parallel on the separate workers, some form of communication between those workers is regularly required. For example, in parallel tempering, exchange moves are performed between the various MCMC chains, and in SMC, all the particles must be combined for a resampling move. Before any communication can occur, all workers must complete the independent updates they are engaged in. At this point, an issue arises when algorithms take a varying and random time to complete their parallel updates. One reason for that is that the time taken to complete parallel moves may depend on the current state of the chain (see for example ABC in Chapter 4 and RJ-MCMC in Chapter 5). Another, more practical reason, is related to variations caused by the computing infrastructure, such as competing jobs on the same processor, I/O load, processor hardware, network traffic, system failures, all of which arise in shared computing environments such as cloud computing, for example. As a result, workers must sit idle, waiting for the slowest worker to complete their independent parallel updates before communication can occur. This idling issue is addressed at length in Murray et al. [2016b] and for various applications in Chapters 4 and 5.

1.3 Tackling complex models

One of the advantages of Monte Carlo methods is their versatility – they can be adapted to the most advanced and complex problems, and it is important to develop efficient algorithms to tackle such problems. This thesis explores a variety of advanced models and the algorithms that deal with them. For example, when dealing with a model for which the likelihood is unavailable (such as the Lotka-Volterra predator-prey model in Section 4.5.4), one can resort to ABC (Sisson and Fan [2011]), a class of algorithms described in Section 2.7 in which instead of evaluating the likelihood, artificial datasets are simulated and compared to the observed data.

Changepoint models, considered in Chapter 5, are particularly useful for modelling the heterogeneity of sequential observations. However, the number and locations of the changepoints are often unknown. The varying model dimensions that must be considered when inferring on the number of changepoints means that basic MCMC algorithms cannot be applied. Fortunately, when comparing models of varying dimensions, RJ-MCMC (Green [1995]), described in Section 2.8, generalises MCMC to allow for transdimensional updates in the Markov chain.

Another important branch of models explored in Chapter 6, which has recently garnered a lot of attention in the literature, is *single-molecule microscopy*. Indeed, it has allowed significant insight into the behaviour of single molecules in cellular environments using *fluorescence microscopy*. Single-molecule fluorescence microscopy (Moerner and Fromm [2003]; Shashkova and Leake [2017]) consists of using a suitable fluorophore to label the molecule of interest, exciting said fluorophore with a specific light source, and capturing the fluorescence emitted by the molecule through a microscope system onto a detector during a fixed acquisition time. Parameter inference such as being able to evaluate the Fisher information matrix for key hyperparameters is a very important aspect of single-molecule microscopy, as it is crucial for the effective design of experiments. However, these models are not straightforward. Firstly, according to optical diffraction theory (Born and Wolf [2013]), the locations of the photons on the detector are distributed according to difficult models such as the Airy profile (for an in-focus molecule) and the Born and Wolf model (for an out of focus molecule). Secondly, the arrival times of the photons on the detector are random and non-uniformly distributed, a factor which must be taken into account. And finally, if one considers molecules with stochastic trajectories, rather than static molecules or ones with deterministic trajectories, the problem is complicated even further. All three of these issues have been addressed separately, but they have rarely all been considered at once (Vahid et al. [2020]). Additionally, Gaussian approximations of the photon location profiles have often been employed in order to use exact filtering and smoothing techniques such as the Kalman filter (and smoother) to perform parameter inference, but it is possible to employ SMC methods and directly use the Airy profile and Born and Wolf model instead.

1.4 Outline

This thesis is organised as follows. Chapters 2 and 3 provide background information on two major types of Monte Carlo methods employed throughout the thesis, namely Markov chain Monte Carlo (MCMC) and particle filters, or sequential Monte Carlo (SMC) methods. The original contributions of this thesis are organised in Chapters 4-6.

Chapter 2: Markov Chain Monte Carlo and its Variants

We formally present MCMC and give an overview of methods to accelerate MCMC algorithms, with a focus on parallel tempering. Then, we introduce several extensions of MCMC such as SMC samplers for sampling from a sequence of distributions and Murray et al. [2016b]'s Anytime Monte Carlo framework, as well as variants such as ABC, that deals with applications where the likelihood is unavailable, and RJ-MCMC, that introduces transdimensional updates.

Chapter 3: Sequential Monte Carlo

This chapter presents various SMC methods for state space models. Starting from simple filtering and smoothing approaches for finite and linear Gaussian state space models and progressing into particle filtering and smoothing for general state space models. Several algorithms are described, which deal with the various degeneracy and efficiency issues raised in Section 1.1.2.

Chapter 4: Anytime Parallel Tempering

We introduce a novel algorithm called *Anytime Parallel Tempering Monte Carlo* (APTMC) which combines the commonly used parallel tempering algorithm to improve the efficiency of MCMC with the Anytime Monte Carlo framework. The APTMC algorithm's performance improvements are illustrated in multiple toy examples and real-life applications to ABC.

The work carried out in this chapter has been submitted for publication in collaboration with co-authors Dr Sumeetpal Singh and Dr Lawrence Murray. An arXiv preprint is available at Marie d'Avigneau et al. [2020].

Chapter 5: A General Anytime SMC Sampler for Changepoint Models using Reversible Jump MCMC

We present a novel general algorithm for performing changepoint inference using RJ-MCMC as part of an SMC sampler, implemented within the Anytime framework. The algorithm is presented in a general way that can easily be adapted to any changepoint model, and especially difficult ones. The workings of the algorithm are illustrated on a complex

changepoint model, and the performance gains obtained by the addition of the Anytime framework are demonstrated on simulated data.

Chapter 6: Particle Smoothing for Parameter Inference on Dynamic Single Molecules with Stochastic Trajectories

This chapter moves away from the Anytime framework to focus on single-molecule microscopy models in which the trajectory of the object of interest is described by a linear SDE, and the photons it emits onto a detector are observed for a fixed time interval. A difficulty of the model is that the arrival times of the photons on the detector are not uniformly distributed. However, in this chapter, we discretise the observation interval so that the model can be formulated as a straightforward state space model. As a result, we are able to apply common particle filtering and smoothing techniques in order to estimate the FIM of hyperparameters of interest relating to the SDE, and that for complex photon location measurement models such as the Airy profile and Born and Wolf model, which could not be done before.

The work carried out in this chapter has been submitted for publication in collaboration with co-authors Dr Sumeetpal Singh and Professor Raimund Ober. An arXiv preprint is available at Marie d'Avigneau et al. [2021].

Chapter 2

Markov Chain Monte Carlo and its Variants

2.1 Chapter overview

This chapter contains the first half of the literature review, focusing on Markov chain Monte Carlo (MCMC) methods. The chapter begins with an introduction to MCMC and its most widely used algorithms, before reviewing some approaches that have been developed to improve their efficiency. Then, some variants of MCMC are presented, which were introduced with aims ranging from further efficiency improvements to adapting MCMC to more complex and general applications. All the MCMC variants presented feature in some way or another in Chapters 4 and 5.

2.2 Introduction

Consider a set of $\mathscr{Y}$-valued observations Y following a probability model with underlying parameters $X \in \mathscr{X}$ and associated *likelihood* $g(y|x)$. The parameters x describing the data y are unknown and considered random variables with *prior* density $p(x)$. The aim of Bayesian inference is to use the information contained in the prior – representing prior belief – and the likelihood – representing evidence from the data – to obtain the *posterior* density $\pi(x) := p(x|y)$ of the parameters, following Equation 2.1. Summary statistics such as parameter estimates and credible intervals can subsequently be inferred from the posterior obtained.

$$\pi(x) = \frac{p(x)g(y|x)}{\int_{\mathscr{X}} p(x)g(y|x)\mathrm{d}x}. \tag{2.1}$$

In most cases however, the posterior π is intractable, or only known up to a proportionality constant, as the integral in the denominator of Equation 2.1 is unavailable.

More generally, the problem of computing the following integral often arises

$$\mathscr{H} = \mathbb{E}_{\pi}[h(x)] = \int_{\mathscr{X}} h(x), \pi(x) \mathrm{d}x,$$

where h is an arbitrary function. It is generally impossible to evaluate the integral, so it must be approximated using Monte Carlo (Robert and Casella [2004a]) methods, and consists of drawing samples from π. A commonly used method is known as *Markov chain Monte Carlo* (MCMC) (Gilks et al. [1996]), in which a Markov chain is constructed with π as its *target* or stationary distribution. The strength of MCMC methods is that they guarantee convergence to the target distribution of interest. They are also incredibly versatile and easily adaptable to applications such as likelihood-free or transdimensional inference. However, the construction of many MCMC algorithms often leads to a slow convergence, as they tend to only explore local areas rather than the whole support of the distribution. This issue only worsens as the dimensionality of the data and/or complexity of the problem increases. As a result, many techniques have been developed to accelerate the convergence of MCMC algorithms (Bardenet et al. [2015]; Robert et al. [2018]). In this chapter, we review some of the common methods employed to speed up and improve the efficiency of MCMC, as well as some variants that were developed for more complex applications such as likelihood-free inference and transdimensional problems.

2.3 Markov chain Monte Carlo (MCMC)

This section provides an overview of the basic workings of Markov chain Monte Carlo (MCMC) algorithms. A short history of MCMC is available in Robert and Casella [2011].

2.3.1 The Metropolis-Hastings algorithm

First introduced in Metropolis et al. [1953] and generalised in Hastings [1970]; Peskun [1981, 1973] the *Metropolis-Hastings* algorithm is the most generic illustration of how MCMC works. Let x be the current state in the Markov chain, and let $q(\cdot|x)$ be a conditional density, known as the *proposal density*. Propose to move the chain to a new state $x' \sim q(\cdot|x)$ and accept x' as the next sample in the chain with probability

$$a(x,x') = \min\left\{1, \frac{\pi(x')q(x|x')}{\pi(x)q(x'|x)}\right\}, \tag{2.2}$$

where a is known as the *acceptance probability*; otherwise, retain the current state x. As a result, a Markov chain $(X_n)_{n=0}^{\infty}$ is constructed with target distribution π and *Markov transition kernel* given by

$$\kappa(x_{n+1}|x_n) = a(x_n, x_{n+1})q(x_{n+1}|x_n) + \mathbb{1}_{x_{n+1}=x_n}\left(1 - \int_{\mathscr{X}} a(x, x_{n+1})q(x_{n+1}|x)\mathrm{d}x\right). \quad (2.3)$$

The Metropolis-Hastings algorithm is summarised in Algorithm 2.1.

Algorithm 2.1 Metropolis-Hastings algorithm

Input: current sample in the chain X_n.
1: Sample $X' \sim q(\cdot|X_n)$.
2: With probability $a(X_n, X')$, set $X_{n+1} := X'$, otherwise set $X_{n+1} := X_n$.
Output: updated sample X_{n+1}.

To establish that the stationary distribution is indeed π, it suffices to verify that the *detailed balance* condition (see Robert and Casella [2004b]) is verified, i.e.

$$\pi(x_n)\kappa(x_{n+1}|x_n) = \pi(x_{n+1})\kappa(x_n|x_{n+1}). \quad (2.4)$$

Plugging in the transition kernel from Equation 2.3, we have for the first term

$$\begin{aligned}\pi(x_n)a(x_n, x_{n+1})q(x_{n+1}|x_n) &= \min\{\pi(x_n)q(x_{n+1}|x_n), \pi(x_{n+1})q(x_n|x_{n+1})\} \\ &= \pi(x_{n+1})a(x_{n+1}, x_n)q(x_n|x_{n+1}),\end{aligned}$$

and the second term trivially cancels out when $x_n = x_{n+1}$. Therefore, the detailed balance condition is satisfied. Note that plugging Equation 2.1 into Equation 2.2 above is straightforward and avoids the need to evaluate the integral in order to sample from π.

When the proposal distribution q is symmetric, we have $q(x'|x) = q(x|x')$ for all $x, x' \in \mathscr{X}$ and the acceptance probability simplifies to

$$a(x, x') = \min\left\{1, \frac{\pi(x')}{\pi(x)}\right\}. \quad (2.5)$$

This version of the algorithm is known as *Metropolis* (Metropolis et al. [1953]). The simplicity of not having to evaluate the proposal density makes the use of symmetric versions very common. An example is the widely used *random walk Metropolis update* in which the proposed state is given by $x' = x + \varepsilon$ where $\varepsilon \sim \mathscr{N}(0, \sigma^2)$.

2.3.2 Gibbs sampling

The popular MCMC algorithm known as the *Gibbs sampler* (Gelfand and Smith [1990]; Geman and Geman [1984]) is particularly useful when X is multi-dimensional, i.e $X = (X_1, \ldots, X_d)$ where $d > 1$ is the number of components, and it is possible to sample directly from the full conditional $\pi_k(\cdot|x_{-k})$ where $x_{-k} := (x_{1:k-1}, x_{k+1:d})$ for $k = 1 \ldots d$.

2.3.2.1 Random scan Gibbs sampling

In the *random scan* Gibbs sampler, each iteration n of the algorithm proceeds as follows: update the current state $x_n := (x_{1,n}, \ldots, x_{d,n})$ by sampling a random index $k \sim \nu$ where the density $\nu(i) > 0$ for $1 \leq i \leq d$ and then sampling

$$x_{k,n+1} \sim \pi_k \left(\cdot|x_{-k,n}\right).$$

This version of the Gibbs sampler is simply a special case of the Metropolis-Hastings algorithm where the acceptance ratio is always 1:

$$\begin{aligned}\frac{\pi\left(x_{k,n+1}\right)\nu(k)\pi_k\left(x_{k,n}|x_{-k,n+1}\right)}{\pi\left(x_{k,n}\right)\nu(k)\pi_k\left(x_{k,n+1}|x_{n,-k}\right)} &= \frac{\pi\left(x_{k,n+1}|x_{-k,n+1}\right)\pi\left(x_{-k,n+1}\right)\pi_k\left(x_{k,n}|x_{-k,n+1}\right)}{\pi\left(x_{k,n}|x_{-k,n}\right)\pi\left(x_{-k,n}\right)\pi_k\left(x_{k,n+1}|x_{-k,n}\right)} \\ &= \frac{\pi_k\left(x_{k,n+1}|x_{-k,n}\right)\pi\left(x_{-k,n}\right)\pi_k\left(x_{k,n}|x_{-k,n+1}\right)}{\pi_k\left(x_{k,n}|x_{-k,n+1}\right)\pi\left(x_{-k,n}\right)\pi_k\left(x_{k,n+1}|x_{-k,n}\right)} = 1,\end{aligned}$$

where the second equality stems from the fact that $x_{-k,n+1} = x_{-k,n}$.

2.3.2.2 Multi-stage Gibbs sampling

Instead of only updating one random component of X per iteration, a *multi-stage* or *systematic scan* Gibbs sampler iterates through all components sequentially. In this case, each iteration of the algorithm proceeds as follows: let X^n be the current state of the chain, then update the state by sampling

$$x_{k,n+1} \sim \pi_k(\cdot|x_{1:k-1,n+1}, x_{k+1:d,n})$$

sequentially for $k = 1, \ldots, d$. The Markov transition kernel is then defined as $\kappa := \kappa_1 \kappa_2 \ldots \kappa_d$ where each transition kernel is given by

$$\kappa_k(x_{n+1}|x_n) = \pi_k(x_{k,n+1}|x_{1:k-1,n+1}, x_{k+1:d,n}).$$

This version of the Gibbs sampler does not satisfy detailed balance but it still converges to stationary distribution π, which can be established as follows:

$$\begin{aligned}\int_{\mathscr{X}} \pi(x)\kappa(y|x)\mathrm{d}x &= \int_{\mathscr{X}} \prod_{k=1}^{d} \pi(x_k|x_{1:k-1})\pi_k(y_k|y_{1:k-1},x_{k+1:d})\mathrm{d}x_{1:d} \\ &= \int_{\mathscr{X}} \prod_{k=1}^{d} \pi(x_k|x_{1:k-1}) \frac{\pi(y_k|y_{1:k-1})\pi(x_{k+1:d}|y_{1:k})}{\pi(x_{k+1:d}|y_{1:k-1})}\mathrm{d}x_{1:d} \\ &= \pi(y) \int_{\mathscr{X}} \prod_{k=1}^{d} \pi(x_k|y_{1:k-1},x_{k+1:d})x_{1:d} = \pi(y).\end{aligned}$$

For more details, see Mikusheva [2007]; Robert and Casella [2004a]. The multi-stage Gibbs sampler is summarised in Algorithm 2.2.

Algorithm 2.2 Multi-stage Gibbs sampler

Input: current sample in the chain $X_n = \left(X_{1,n},\ldots,X_{d,n}\right)$.
1: **for** $k = 1,\ldots,d$ **do**
2: Sample $X_{k,n+1} \sim \pi_k\left(\cdot|X_{1:k-1,n+1},X_{k+1:d,n}\right)$.
3: **end for**
Output: updated sample X_{n+1}.

2.3.2.3 Metropolis-within-Gibbs

When it isn't possible to sample directly from the full conditional $\pi_k\left(\cdot|x_{-k}\right)$ but a proposal distribution q_k is available for component k, it is straightforward to incorporate a Metropolis-Hastings algorithm within a Gibbs sampler. The acceptance ratio for the k-th component becomes

$$a_k\left(x_k,x_k'|x_{-k}\right) = \min\left\{1, \frac{\pi\left(x_k'\right)q_k\left(x_k|x_k',x_{-k}\right)}{\pi(x_k)q_k\left(x_k'|x_k,x_{-k}\right)}\right\}.$$

The resulting algorithm is known as *Metropolis-within-Gibbs* (Tierney [1994]). If the proposal q_k is close to the full conditional π_k, this approximates the Gibbs sampler well.

2.4 Accelerating Markov chain Monte Carlo

Despite being easily adaptable and convenient, basic MCMC algorithms can be very slow to converge to their stationary distribution due to the localised nature of their updates. Indeed, their lack of 'awareness' of the full support of the distribution can lead to them getting stuck in local maxima, as illustrated in Example 2.1.

Example 2.1. Consider the example from Wilkinson [2013] in which we wish to sample from the following density:

$$\pi(x) \propto \exp\left[-\chi(x^2-1)^2\right], \tag{2.6}$$

which is bimodal for $\chi > 0$. Obtaining 5×10^5 samples from π for $\chi = 8$ using a single chain with a basic random walk Metropolis update is inefficient and returns a posterior which tends to overestimate one of the modes, as evidenced in Figure 2.1.

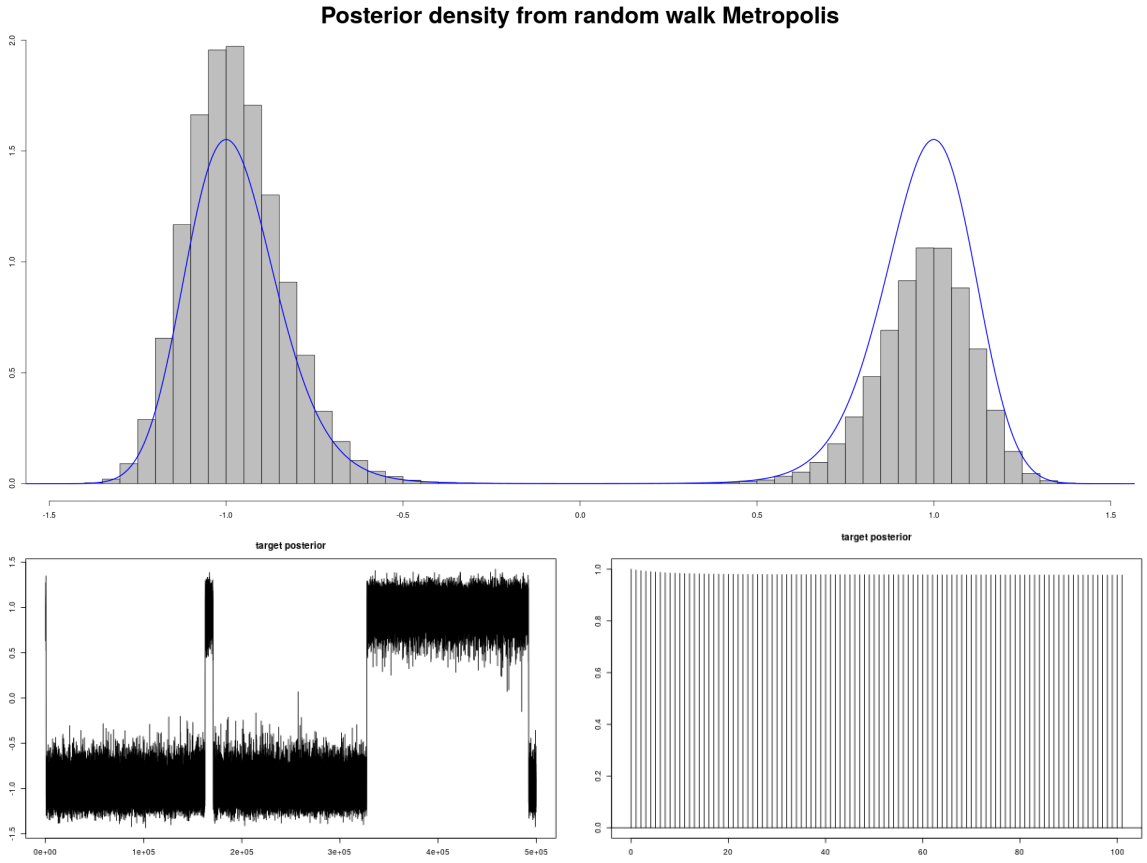


Fig. 2.1 Output from 5×10^5 samples of a random walk MCMC chain targeting the bimodal density in Equation 2.6. **Top**: histogram of random walk MCMC output compared to the true density (*blue line*), the histogram overestimates one of the modes and underestimates the other. **Bottom left**: MCMC chain output showing that random walk Metropolis tends to get stuck in local maxima for extended periods. **Bottom right**: sample autocorrelation (acf) function decaying extremely slowly, indicating that the samples are highly correlated and thus the algorithm is very inefficient.

This efficiency issue only worsens as the dimensionality of the problem increases. Fortunately, a vast range of algorithms have been developed to improve the performance of

MCMC algorithms. These can be organised in multiple categories, including algorithms that explore the state space more efficiently, such as tempering or gradient-based methods, scalable algorithms, which aim to manage the cost of individual computations for big data applications, and distributed computing approaches. Comprehensive reviews of how to speed up MCMC algorithms are available in Bardenet et al. [2017]; Robert et al. [2018]. In the rest of this section, we present a brief overview of some of the most commonly used algorithms to speed up MCMC, and present the key algorithms that will be employed throughout this thesis.

2.4.1 Parallel tempering and other schemes

Several tempering schemes have been developed to accelerate convergence and eliminate the risk of MCMC algorithms getting stuck exploring local maxima. These include parallel tempering (Earl and Deem [2005]; Geyer [1991]; Swendsen and Wang [1986]), employed in Chapter 4, as well as simulated or serial tempering (Geyer and Thompson [1995]; Marinari and Parisi [1992]), tempered transitions (Neal [1996]) and annealed importance sampling (AIS) (Neal [2001]). Tempering schemes consider a sequence of *tempered densities* $\pi_1, \ldots, \pi_\Lambda$ that allow for more exploration of the state space. For example, in *simulated tempering*, the densities become the components of a mixture as $\gamma(\lambda, x) = \pi_\lambda(x) c_\lambda$, where the c_λ are user-defined constants, as follows:

$$\gamma(x) = \sum_{\lambda=1}^{\Lambda} \gamma(\lambda, x),$$

where $\gamma(x) \propto \pi(x)$ is the unnormalised target distribution of interest. A Markov chain is run with state (λ, X) where λ is the mixture component index, indicating which of the tempered densities the algorithm is currently visiting.

Parallel tempering is more commonly used and intuitive to implement. It runs a global Markov chain $\left(X_n^{1:\Lambda}\right)_{n=1}^{\infty} := \left(X_n^1, \ldots, X_n^\Lambda\right)_{n=1}^{\infty}$ with target distribution

$$\pi(x^{1:\Lambda}) \propto \prod_{\lambda=1}^{\Lambda} \pi_\lambda(x^\lambda), \tag{2.7}$$

where the marginals $\pi_1, \ldots, \pi_\Lambda$ correspond to the target distributions of each of Λ Markov chains. One of these chains, for example the Λ-th chain, is commonly known as the *cold chain* and corresponds to the target density of interest π. The other, tempered or 'warmer' chains, are more diffuse in that they allow for more and more exploration of the state space as their 'temperature' increases. A parallel tempering algorithm is made of two types of update:

1. *Local moves*: generally a standard Gibbs or Metropolis-Hastings update applied to each tempered chain independently.

2. *Exchange moves*: propose to swap the states $x^{\lambda} \sim \pi_{\lambda}$ and $x^{\lambda'} \sim \pi_{\lambda'}$ of two chains and accept the swap with probability

$$a\left(x^{\lambda}, x^{\lambda'}\right) = \min\left\{1, \frac{\pi_{\lambda}\left(x^{\lambda'}\right)\pi_{\lambda'}\left(x^{\lambda}\right)}{\pi_{\lambda}\left(x^{\lambda}\right)\pi_{\lambda'}\left(x^{\lambda'}\right)}\right\}. \tag{2.8}$$

   Otherwise, the chains in the pair retain their current states. Assume that $\lambda' > \lambda$ for simplicity. An exchange move can be seen as a special case of the Metropolis algorithm where the current state is $(x^1, \ldots, x^{\lambda}, \ldots, x^{\lambda'}, \ldots, x^{\Lambda})$ and a new state $(x^1, \ldots, x^{\lambda'}, \ldots, x^{\lambda}, \ldots, x^{\Lambda})$ is proposed where the λ-th and λ'-th chains are swapped. Indeed, plugging in the joint density from Equation 2.7 into the Metropolis update in Equation 2.5 returns the swap probability in Equation 2.8.

With the cold chain providing more precision and the warmer chains more freedom of movement when exploring the parameter space, the combination of the two types of update allows all chains to mix much faster than any one of them would mix on its own. This provides a way to jump from mode to mode in far fewer steps than would be required under a basic Metropolis-Hastings algorithm, as illustrated in Example 2.2.

Note that the method in which the two candidate chains are selected in an exchange move must ensure that the proposal density is symmetric for the Metropolis-like update to be reversible. The most basic way of proposing two chains would be to choose the indices λ and λ' uniformly at random from the set $[\![1, \Lambda]\!]$ without replacement. This is however inefficient, as if the chains are too far apart, the rejection probability will be much higher. As a result, most parallel tempering algorithms privilege choosing pairs of adjacent chains for exchange moves. In Geyer [2011], it is suggested that one should first sample the index λ uniformly at random from the set $[\![1, \Lambda]\!]$ and then choose λ' uniformly at random from a user-specified set of neighbours of λ. Alternatively, one could select a pair of chain indices at random from the set of adjacent pairs $\{(1,2),(2,3),\ldots,(\Lambda-1,\Lambda)\}$. Since the exchange move is relatively cheap, it is also common to propose to swap multiple pairs of chains. To avoid selecting the same chain twice, they are divided into odd $O = \{(1,2),(3,4),\ldots\}$ and even $E = \{(2,3),(4,5),\ldots\}$ pairs of indices. In what Lingenheil et al. [2009] refer to as the stochastic even/odd (SEO) algorithm, the set O is selected with probability $\frac{1}{2}$ and exchange moves are performed on all the odd pairs, otherwise they are performed on the even pairs. It is however argued in detail in Syed et al. [2019] that the non-reversible alternative to the SEO

algorithm, in which we alternate between proposing to exchange all odd and all even pairs deterministically, known as the deterministic even/odd algorithm (DEO), can outperform its reversible counterpart.

Example 2.2. Revisiting Example 2.1, apply the following inverse tempering scheme to the density of interest

$$\pi_\lambda(x) \propto \pi(x)^{\frac{\lambda}{\Lambda}}$$

for $\lambda = 1, \ldots, \Lambda$. In this case, the density π_8 corresponds to the target density of interest π defined in Equation 2.6 and the densities get warmer as λ decreases (see Figure 2.2). Again, 5×10^5 samples from π are obtained using $\Lambda = 8$ chains, with the same random walk Metropolis update as in Example 2.1 for the local moves and adding exchange moves between pairs of adjacent chains. The algorithm is now significantly more efficient, as evidenced in Figure 2.3.

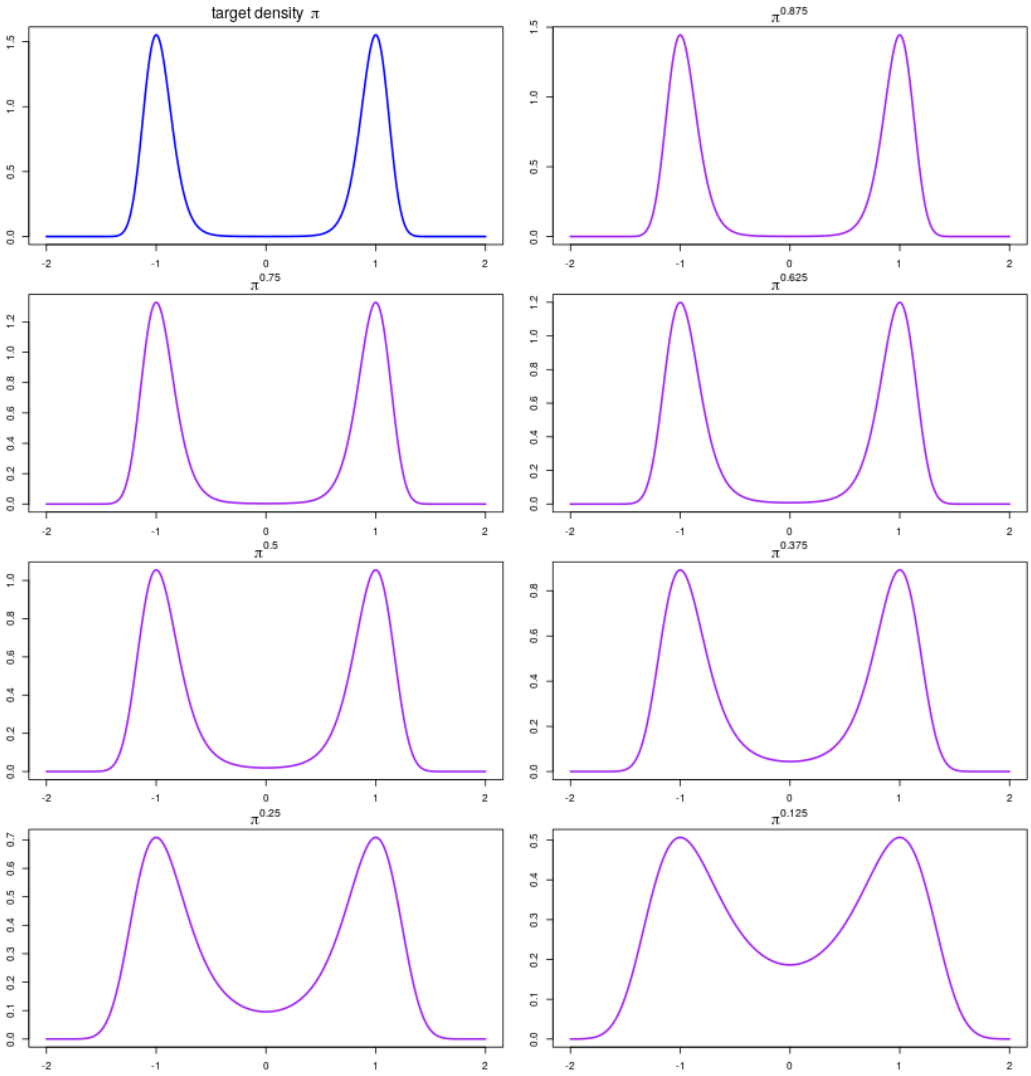


Fig. 2.2 Tempered target densities $\pi^{\frac{\lambda}{\Lambda}}$, $\lambda = 1, \ldots, \Lambda$ of the parallel tempering algorithm for $\Lambda = 8$ temperatures. The tempering is employed to improve convergence when sampling from the bimodal density π of Example 2.1. The top left plot corresponds to the target or *cold* density and displays a probability of almost zero in the dip between the two modes. As the densities get increasingly warmer, it becomes easier to switch between the two modes, as the probability in the dip between them increases.

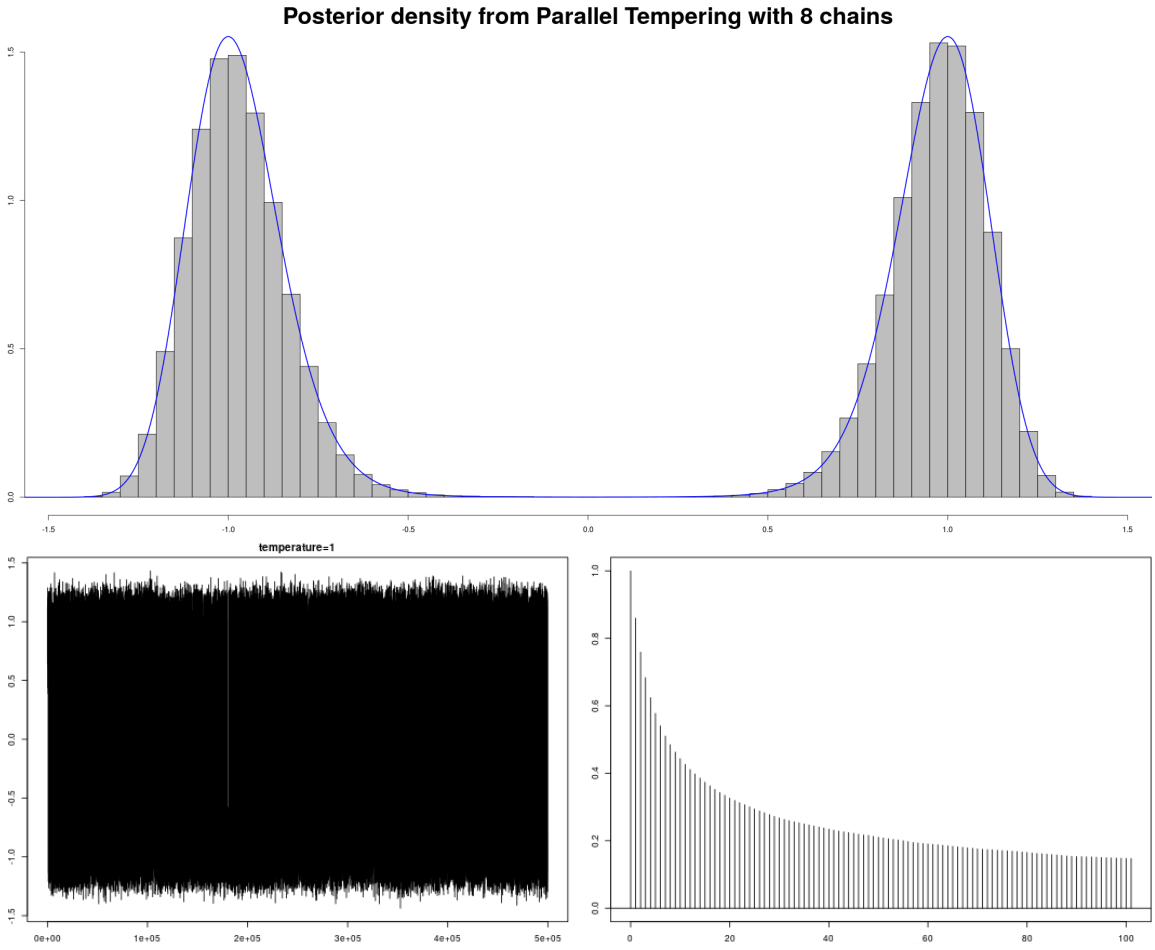


Fig. 2.3 Output from 5×10^5 samples of the cold parallel tempering chain targeting the bimodal density in Example 2.2. **Top**: histogram of the cold chain output compared to the true density (*blue line*); thanks to the exchange moves between chains, the algorithm has converged to its target distribution. **Bottom left**: cold chain MCMC output able to explore the whole state space thanks to the multi-chain parallel tempering construction and exchange moves. **Bottom right**: sample autocorrelation (acf) function for the cold chain decaying a lot faster than the basic random walk algorithm in Figure 2.1, indicating that parallel tempering yields a significant gain in efficiency.

2.4.2 Scalable and gradient-based methods

The most expensive part of MCMC algorithms is generally the computation of the likelihood, and scalable methods aim to make this cost more manageable. A comprehensive review of existing methods that aim to scale up the Metropolis-Hastings algorithm for big data applications and reduce computational costs is available in Bardenet et al. [2015]. The authors divide the available approaches into two categories: divide-and-conquer and subsampling methods.

First of all, *divide-and-conquer* methods (Minsker et al. [2014]; Neiswanger et al. [2013]; Wang and Dunson [2013]; Xu et al. [2014]) aim to divide the data into batches, then run the MCMC algorithm on each batch separately, before combining the subposteriors obtained to form an approximation of the full posterior. These methods therefore have an important advantage which is the possibility of running the algorithm in distributed computing environments, and thus significantly speeding up computations. This is the case of the consensus Monte Carlo algorithm presented by Scott et al. [2016]. However, some issues these methods must also address are how to keep communication between batches minimal and how to efficiently combine the subposteriors into a good approximation of the target posterior.

On the other hand, *subsampling* methods aim to reduce the number of likelihood evaluations to speed up computations. This approach includes *pseudo-marginal* MCMC methods (Andrieu et al. [2009]; Deligiannidis et al. [2018]) which employ unbiased estimators of the unnormalised target distribution. This usually means that only a subsample of the data is used at each iteration, which speeds up computations. A few examples of subsampling MCMC algorithms are the Bootstrap Metropolis-Hastings algorithm by Liang et al. [2016] and the confidence sampler developed by Bardenet et al. [2014] and extended in Bardenet et al. [2015] and in Kohn et al. [2016]. More examples of subsampling MCMC are available in Quiroz et al. [2016] and Korattikara et al. [2014]. It is also possible to use delayed acceptance MCMC, such as the Firefly algorithm in Maclaurin and Adams [2014]. The advantage of delayed acceptance MCMC is that it avoids computation of the likelihood if there is evidence that the proposal will be rejected; however, in general, it will compute the likelihood on the full dataset otherwise, which is not ideal when the likelihood itself is too computationally costly. This is avoided in Quiroz et al. [2017], where delayed acceptance is combined with subsampling.

Another major approach to accelerating MCMC is to exploit the geometry of the target. This in a way achieves a similar purpose as tempering, in that it allows for more efficient Metropolis-Hastings proposals. The most well-known approach is known as *Hamiltonian Monte Carlo* (HMC) described in Neal [2011]. Given target $\pi(x)$, let $\mathscr{L}(x) = \log \pi(x)$.

Taking advantage of Hamiltonian dynamics (Duane et al. [1987]), an auxiliary variable p known as the *momentum* is introduced so that we generally have $p \sim \mathcal{N}(0, M)$ where M is the target covariance. A continuous process $(X_t, P_t)_{t=0}^{\infty}$ can then be constructed targeting the joint density $\pi(x, p)$ so that $\frac{\partial}{\partial t} X_t = M^{-1} P_t$ and $\frac{\partial}{\partial t} P_t = \nabla \mathscr{L}(X_t)$. Essentially, at any time, the momentum is driven by the score of the log density of interest at the current state, and thus efficiently guides the process towards the joint density. Generally, the path of this continuous process is intractable, so a discretisation is employed instead (Betancourt [2017]) and the final state in the path is accepted or rejected following a Metropolis-Hastings step. A commonly used variant of HMC is known as the *no-U-turn sampler* (NUTS) (Hoffman and Gelman [2014]), which adapts the discretisation step size. Another approach is known as the *Metropolis-adjusted Langevin algorithm* (MALA) (Roberts et al. [1996]) and is equivalent to using HMC where the discrete path generated only contains a single step, effectively constructing an MCMC chain with informed proposals.

2.5 Sequential Monte Carlo (SMC) samplers

Sequential Monte Carlo (SMC) methods will be explored in detail in Chapter 3, but in this section we focus on a particular branch known as *Sequential Monte Carlo samplers*, introduced in Del Moral et al. [2006] as a complementary approach to MCMC sampling with a wide range of applications. The basic aim is to sample sequentially from a sequence of probability distributions $\pi_1, \ldots, \pi_n$ defined on a common space, though it can easily be extended to an instance where the π_k are defined on different spaces, where the index $k = 1, \ldots, n$ is referred to as the *step* throughout this thesis. Following the basic construction of a particle filter, a weighted particle population is propagated through the sequence so that at step k it approximates the distribution π_k.

There are many possible choices for the sequence of distributions. The most natural choice, in keeping with particle filters, is to have $\pi_k(x) = p(x|y_{1:k})$, i.e. where $\pi_k(x)$ corresponds to the posterior given the data $y_{1:k}$ collected until step k. If the whole dataset $y_{1:n}$ is already available, Chopin [2002] still suggests defining such a sequence of distributions as an alternative to basic MCMC in order to save computation time. This is the approach employed in Chapter 5. Another option is for the distribution to follow a tempering schedule, not unlike in Section 2.4.1, where π_1 corresponds to the 'warmest' distribution and π_n is the target. There are also many possible choices for the sequence of transition kernels $\kappa_1, \ldots, \kappa_n$ outlined in Del Moral et al. [2006]. For the purpose of this section, we focus on the natural choice of κ_k as an MCMC kernel targeting π_k.

Let π_k be a target density and define the unnormalised density $\gamma_k \propto \pi_k$. Denote as κ_k the

MCMC kernel targeting π_k with density $\kappa_k(x_k|x_{k-1})$ and introduce a user-defined, problem dependent *backward kernel* $\beta_{k-1}(x_{k-1}|x_k)$. Then, define the joint target

$$\tilde{\pi}(x_{1:k}) \propto \tilde{\gamma}_k(x_{1:k}),$$

where

$$\tilde{\gamma}_k(x_{1:k}) = \gamma_k(x_k) \prod_{l=1}^{k-1} \beta_k(x_{l+1}, x_l).$$

The density $\pi_k(x_k)$ is a marginal of $\tilde{\pi}_k(x_{1:k})$. The algorithm is initialised at step $k = 1$ and we assume that it is easy to approximate $\pi_1 = \tilde{\pi}_1$ by sampling $X_1^{(i)} \sim \eta_1$ for $i = 1, \ldots, N$, where the density η_1 is known as the *importance distribution*, and the unnormalised *importance weights* $w_1(X_1^{(i)})$, given by

$$w_1(x) = \frac{\gamma_1(x)}{\eta_1(x)} \tag{2.9}$$

can be computed exactly. This is known as *importance sampling* and is described in more detail in Section 3.3.1. At step $t-1$, we have the particle approximation $(X_{k-1}^{(1:N)}, \omega_{k-1}^{(1:N)})$ of the density $\tilde{\pi}_{k-1}$ defined as follows:

$$\tilde{\pi}_{k-1}^N(\mathrm{d}x_{1:k-1}) = \sum_{i=1}^N \omega_{k-1}^{(i)} \delta_{X_{1:k-1}^{(i)}}(\mathrm{d}x_{1:k-1}),$$

where the normalised importance weights are given by

$$\omega_{k-1}^{(i)} := \frac{w_{k-1}\left(X_{1:k-1}^{(i)}\right)}{\sum_{j=1}^N w_{k-1}\left(X_{1:k-1}^{(j)}\right)}.$$

At step k, propagate the particles according to transition kernel $\kappa_k(x_k|x_{k-1})$ and update the weights by computing the *incremental weights* given by

$$\tilde{w}(x_{k-1}, x_k) := \frac{\gamma_k(x_k)\beta_{k-1}(x_{k-1}|x_k)}{\gamma_{k-1}(x_{k-1})\kappa_k(x_k|x_{k-1})} \tag{2.10}$$

and setting $w_k(x_{1:k}) = w_{k-1}(x_{1:k-1})\tilde{w}(x_{k-1}, x_k)$.

As is common in particle filters, an additional step must be added to the algorithm to avoid the well-known issue of *weight degeneracy* (Doucet and Johansen [2009]; Kong et al. [1994]), in which the number of weights with significant weight drops rapidly, and we eventually end up with a single particle with weight equal to 1 while all other particles have zero weight. To circumvent this issue, the particles are resampled if their effective sample

size given by

$$ESS_k = \left[\sum_{i=1}^{N} \left(\omega_k^{(i)}\right)^2\right]^{-1} \tag{2.11}$$

falls below a certain threshold T_{ESS} (Liu and Chen [1998]). The resampling probabilities are proportional to the importance weights $\{\omega_k^{(i)}\}_{i=1}^N$. A summary of the algorithm is provided in Algorithm 2.3.

Algorithm 2.3 General SMC sampler

Where (i) *appears, the operation is performed for all* $i = 1,\ldots,N$.
Initialise at step $k = 1$:

1: Draw $X_1^{(i)} \sim \eta_1$.
2: Compute the importance weights $w\left(X_1^{(i)}\right)$ according to Equation 2.9, then normalise them to obtain $\omega_1^{(i)}$.
3: **for** $k = 2,\ldots,n$ **do**
4: (RESAMPLE) Obtain the ESS_{k-1} using Equation 2.11. If $ESS_{k-1} < T_{ESS}$, resample the particles and set $\omega_k^{(i)} = \frac{1}{N}$.
5: (PROPAGATE) Propagate the particles, i.e. sample $X_k \sim \kappa_k\left(\cdot|X_{k-1}^{(i)}\right)$.
6: (WEIGHT) Compute the incremental weights $\tilde{w}\left(X_{k-1}^{(i)}, X_k^{(i)}\right)$ according to Equation 2.10, then normalise the importance weights to obtain $\omega_k^{(i)}$ as follows:

$$\omega_k^{(i)} = \frac{\omega_{k-1}^{(i)} \tilde{w}\left(X_{k-1}^{(i)}, X_k^{(i)}\right)}{\sum_{j=1}^{N} \omega_{k-1}^{(j)} \tilde{w}\left(X_{k-1}^{(j)}, X_k^{(j)}\right)}.$$

7: **end for**

As is common in a particle filter, the SMC sampler in Algorithm 2.3 follows the standard *resample→propagate→weight* procedure (see Chapter 3). The backward kernel is artificial and its explicit form needs not be known but is instead implied by the computation of the incremental weights. Given we are working with an MCMC forward kernel κ_k targeting π_k, the backward kernel can be defined as follows:

$$\beta_{k-1}(x_{k-1}|x_k) = \frac{\pi_k(x_{k-1})\kappa_k(x_k|x_{k-1})}{\pi_k(x_k)},$$

which yields the incremental weight

$$\tilde{w}(x_{k-1}, x_k) = \frac{\gamma_k(x_{k-1})}{\gamma_{k-1}(x_{k-1})}.$$

Note that this incremental weight does not depend on the current state X_k. Therefore, when using this backward kernel (as is the case in Chapter 5), the order of the algorithm is slightly different, and proceeds as *weight→resample→propagate* instead (see Algorithm 5.2).

One of the main appeals of SMC samplers is that they allow for basic MCMC algorithms to be applied in an SMC setting. The presence of a population of particles means that the propagation step of the algorithm can be seen as 'local moves' and easily performed in parallel on multiple processors. A particular advantage this class of algorithms also has over parallel tempering is that while it can employ the same tempering schedule to efficiently explore the state space, there is no need to assess the convergence of the MCMC chains.

2.6 Anytime Monte Carlo

Typically, when running MCMC algorithms, the aim is to draw a fixed number, say n, of samples, which takes a random amount of real time $T(n)$. Instead, the *Anytime Monte Carlo* framework (Murray et al. [2016b]) fixes the real time t during which samples are drawn and the number of samples returned $N(t)$ becomes a random variable.

Let $(X_k)_{k=0}^{\infty}$ be a Markov chain with initial state X_0, evolving on state space $\mathscr{X}$, with transition kernel $\kappa(x_k|x_{n-1})$ and target distribution π. Define the *hold time* H_{k-1} as the random and positive real time required to complete the computations necessary to transition from state X_{k-1} to X_k via the kernel κ. Then let $H_{k-1}\,|\,x_{k-1} \sim \tau(h_{k-1}|x_{k-1})$ where τ is the hold time distribution.

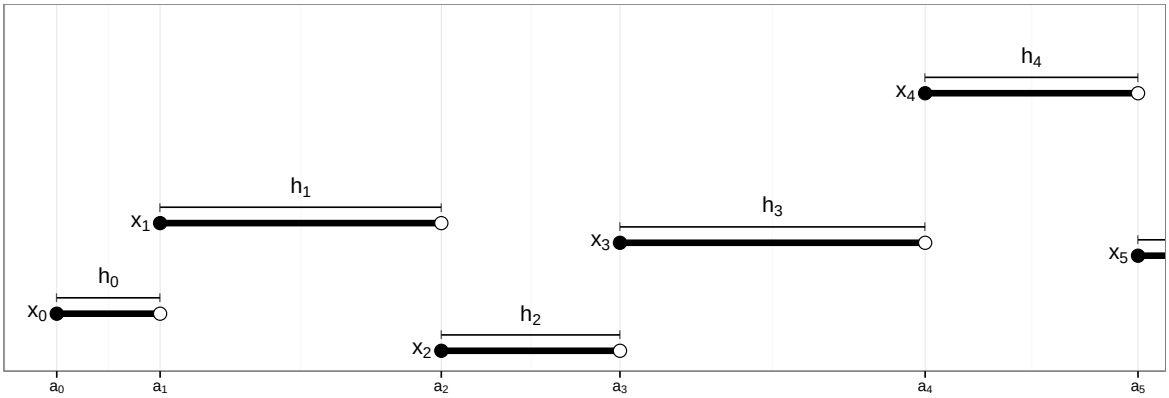


Fig. 2.4 (Murray et al. [2016b], Figure 1) Real-time realisation of a Markov chain with states $(X_n)_{n=0}^{\infty}$, arrival times $(A_n)_{n=0}^{\infty}$ and hold times $(H_n)_{n=0}^{\infty}$.

Assume that the hold time $H > \varepsilon > 0$ for minimal time ε, $\sup_{x\in\mathscr{X}} \mathbb{E}\,[H\,|\,x] < \infty$, and the hold time distribution τ is homogeneous in time. In general, nothing is known about the hold time distribution τ except how to sample from it, i.e. by recording the time taken by

the algorithm to simulate $X_k \,|\, x_{k-1}$. Let $\kappa(x_k, h_{k-1}|x_{k-1}) = \kappa(x_k|h_{k-1}, x_{k-1})\tau(h_{k-1}|x_{k-1})$ be a joint kernel. The transition kernel $\kappa(x_k|x_{k-1})$ is the marginal of the joint kernel over all possible hold times H_{k-1}. Denote by $(X_k)_{k=0}^{\infty}$ and $(H_k)_{k=0}^{\infty}$ the states and hold times of the joint process, and define the *arrival time* of the n-th state as

$$A_k := \sum_{i=0}^{k-1} H_i, \qquad k \geq 1,$$

where $a_0 := 0$. A possible realisation of the joint process is illustrated in Figure 2.4.

Let the process $N(t) := \sup\{k : A_k \leq t\}$ count the number of arrivals by time t. From this, construct a continuous Markov jump process $(X, L)(t)$ where $X(t) := X_{N(t)}$ and $L(t) := t - A_{N(t)}$ is the *lag time* elapsed since the last jump. This continuous process describes the progress of the computation in real time.

Proposition 2.1. The continuous Markov jump process $(X, L)(t)$ has stationary distribution given by

$$\alpha(x, l) = \frac{\bar{F}_\tau(l \,|\, x)}{\mathbb{E}[H]}\pi(x), \tag{2.12}$$

where $\bar{F}_\tau(l \,|\, x) = 1 - F_\tau(l \,|\, x)$, and $F_\tau(l \,|\, x)$ is the cumulative distribution function (cdf) of $\tau(h_k|x_k)$.

Corollary 2.1. The marginal $\alpha(x)$ of the density in Equation 2.12 is length-biased with respect to the target density $\pi(x)$ by expected hold time, i.e.

$$\alpha(x) = \frac{\mathbb{E}[H \,|\, x]}{\mathbb{E}[H]}\pi(x). \tag{2.13}$$

The proofs of Proposition 2.1 and Corollary 2.1 are given in Murray et al. [2016b].

The distribution α is referred to as the *anytime distribution* and is the stationary distribution of the Markov jump process. Note that Proposition 2.1 suggests that when the real time taken to draw a sample depends on the state of the Markov chain, i.e. $\mathbb{E}[H \,|\, x] \neq \mathbb{E}[H]$, a length bias with respect to computation time is introduced. In other words, when interrupted at real time t, the state of a Monte Carlo computation targeting π is distributed according to the anytime distribution α, which can essentially be seen as a length-biased target distribution. This bias diminishes with time, and when an empirical approximation or average over all post burn-in samples is required, it may be rendered negligible for a long enough computation. However, the bias in the final state does not diminish with time, and when this final state is important (which is the case e.g. in parallel tempering) the bias cannot be avoided by running the algorithm for longer.

We now discuss the approach in Murray et al. [2016b] to correct this bias. The main idea is to make it so expected hold time is independent of X, which leads to $\mathbb{E}[H\,|\,x] = \mathbb{E}[H]$ and hence $\alpha(x) = \pi(x)$, following Corollary 2.1. This is trivially the case for i.i.d sampling as $\kappa(x|x_{k-1}) = \pi(x)$, so the hold time H_{k-1} for X_{k-1} is the time taken to sample $X_k \sim \pi(x)$, and therefore independent of the state X_{k-1}.

For non-i.i.d sampling, assume that $\Lambda \in \mathbb{N}$ MCMC chains are being simulated sequentially on a single processor, and introduce an extra chain. The chains are denoted $(X_k^{1:\Lambda+1})_{k=0}^{\infty}$ and we assume that they have the same target distribution π, kernels κ and hold time distribution τ for $\lambda = 1,\ldots,\Lambda+1$. Their joint target is given by

$$\pi(x^{1:\Lambda+1}) = \prod_{\lambda=1}^{\Lambda+1} \pi(x^{\lambda}).$$

For simplicity, let $\Lambda+1$ always be the state of the chain currently simulating when an interruption occurs. Construct the joint continuous Markov jump process $(X^{1:\Lambda+1}, L)(t)$ where $L(t)$ is the lag time elapsed since the last jump. Generalising Proposition 2.1 and Corollary 2.1, Murray et al. [2016b] establish and prove the following results.

Proposition 2.2. The continuous Markov jump process $\left(X^{1:K+1}, L\right)(t)$ has stationary distribution given by

$$A(x^{1:\Lambda+1}, l) = \alpha(x^{\Lambda+1}, l) \prod_{\lambda=1}^{\Lambda} \pi(x^{\lambda}). \tag{2.14}$$

Corollary 2.2. The marginal $A(x^{1:\Lambda+1})$ of the density in Equation 2.14 is length-biased with respect to the target density $\pi(x^{1:\Lambda+1})$ by expected hold time on the state $X^{\Lambda+1}$ of the currently working chain, i.e.

$$A(x^{1:\Lambda+1}) = \alpha(x^{\Lambda+1}) \prod_{i=1}^{\Lambda} \pi(x^{\lambda}).$$

In other words, Corollary 2.2 states that if a set of $\Lambda+1$ chains being updated sequentially on a single processor are interrupted while chain $\Lambda+1$ is working, the first Λ chains have had a chance to complete their updates and therefore their states are distributed according to π, but the state of the $\Lambda+1$-th chain is distributed according to α, and thus introducing a length bias. Fortunately, it suffices to discard the $\Lambda+1$-th chain to recover a set of chains targeting π only and thus eliminate the length bias.

The Anytime framework is particularly useful in distributed computing experiments for two reasons. First, as we'll see in Chapters 4 and 5, depending on the MCMC implementation and the inference problem itself, each iteration of MCMC algorithms can easily have a non-

uniformly distributed hold time which depends on the part of the state space it is exploring. Many algorithms such as parallel tempering and SMC samplers can be run in parallel on multiple processors, but every so often the chains or particles must interact, e.g. for exchange moves in parallel tempering and resampling in SMC. Before any interaction can occur, all chains or particles *must* complete the independent parallel updates they are engaged in (e.g. local moves for parallel tempering or particle propagation for SMC) to avoid introducing a potentially substantial bias (see Proposition 2.1). This can result in processors sitting idle while the slowest of them completes its parallel updates. The second reason is more practical and relates to computing infrastructure induced variations due e.g. to variations in processor hardware, memory bandwidth, network traffic, I/O load, competing jobs on the same processors, as well as potential unforeseen interruptions e.g. due to system failures, all of which affect the compute time of parallel updates. For example, running experiments on cloud computing platforms is generally financially costly, and being able to manage the compute budget despite these infrastructure variations is desirable. The Anytime framework can mitigate both these situations and yield significant performance improvements, as will be seen in Chapters 4 and 5.

2.7 Approximate Bayesian computation

As Bayesian inference becomes increasingly advanced, it is not uncommon to encounter complex models for which the likelihood is either unavailable analytically or too computationally costly to compute. The notion of *Approximate Bayesian Computation* (ABC), also known as *likelihood-free inference* (Sisson and Fan [2011]), was developed by Tavaré et al. [1997] and Pritchard et al. [1999]. It can be seen as a likelihood-free way to perform Bayesian inference, using instead simulations from the model or system of interest, and comparing them to the observations available (Marjoram et al. [2003]; Ratmann et al. [2007]; Sisson et al. [2007]). Since its inception, it has been widely used in biological (Blum and Tran [2010]; Hamilton et al. [2005]; Jabot and Chave [2009]), signal processing (Nevat et al. [2009]; Peters et al. [2010]), archaeological (Wilkinson and Tavaré [2009]), and other theoretical (Drovandi and Pettitt [2011]; Peters et al. [2012]) applications.

Let $Y \in \mathscr{Y}$ be a set of observations described by underlying parameters $X \in \mathscr{X}$ through the analytically unavailable or computationally costly likelihood function $g(y|x)$. Let $p(x)$ denote the prior density on X. Assuming that it is possible to sample from the density $g(\cdot\,|\,x)$ for all $x \in \mathscr{X}$, approximate the likelihood by introducing an artificial likelihood (Lee [2012]) g^{ε} of the form

$$g^{\varepsilon}(y\,|\,x) = \mathrm{Vol}(\varepsilon)^{-1} \int_{B_{\varepsilon}(y)} g(z\,|\,x)\mathrm{d}z, \tag{2.15}$$

where $B_\varepsilon(y)$ denotes a metric ball centred at y of radius $\varepsilon > 0$ and $\mathrm{Vol}(\varepsilon)$ is its volume. The resulting approximate posterior is given by

$$\pi^\varepsilon(x) := p^\varepsilon(x\,|\,y) = \frac{p(x)g^\varepsilon(y\,|\,x)}{\int_{\mathscr{X}} p(x')g^\varepsilon(y\,|\,x')\mathrm{d}x'}.$$

The likelihood $g^\varepsilon(y\,|\,x)$ cannot be evaluated either, but an auxiliary variable z can be introduced and a kernel constructed to obtain samples from the approximate posterior $\pi^\varepsilon(x,z)$ defined as

$$\pi^\varepsilon(x,z) := p^\varepsilon(x,z\,|\,y) \propto p(x)g(z|x)\mathbb{1}_\varepsilon(z)\mathrm{Vol}(\varepsilon)^{-1},$$

where $\mathbb{1}_\varepsilon(z)$ is the indicator function for $z \in B_\varepsilon(y)$. This is referred to as *hitting* the ball $B_\varepsilon(y)$. The joint density $\pi^\varepsilon(x,z)$ admits the posterior $\pi^\varepsilon(x)$ as its marginal. The most basic version of ABC is known as rejection ABC (Pritchard et al. [1999]; Tavaré et al. [1997]) and is summarised in Algorithm 2.4. It consists of drawing a sample x from the prior, simulating a dataset z from the likelihood conditional on x and accepting x as a sample from the posterior $\pi^\varepsilon(x,z)$ if z hits the ball of radius y, as this means x is a good candidate to have generated the observed data y from this model. Note that for a high-dimensional Y, it may be difficult or

Algorithm 2.4 ABC Rejection sampling

1: Draw $x \sim p(x)$.
2: Simulate data from the likelihood $z \sim g(\cdot|x)$.
3: If $z \in B_\varepsilon(y)$, accept x as a sample from the posterior.

inefficient to directly compare y and z. Instead, it is possible to use summary statistics $s(z)$ and $s(y)$ for the comparison, i.e. $\mathbb{1}_\varepsilon(z)$ now corresponds to $s(z) \in B_\varepsilon(s(y))$. See Marin et al. [2014] for a discussion on the choice of statistics.

While it allows for independent sampling from the approximate posterior, the ABC rejection sampler can be extremely inefficient, since proposing candidates from the prior does not take into account the data or previously accepted candidates (see Section 4.5.4 for an example). To mitigate this issue, an MCMC kernel can be constructed to form a Markov chain $(X_n, Z_n)_{n=0}^\infty$ with invariant distribution $\pi^\varepsilon(x,z)$. In Marjoram et al. [2003] a Metropolis-Hastings MCMC kernel was introduced for sampling from the posterior. First, note that in Algorithm 2.4, Steps 1 and 2 are equivalent to proposing a sample from the density $p(x)g(z|x)$. Similarly, to construct a Metropolis-Hastings kernel, define the proposal density

$$q(x',z'|x,z) = q(x'|x)g(z'|x')$$

and accept the candidate x' as the next sample in the Markov chain with probability

$$\begin{aligned} a(x,z,x',z') &= \min\left\{1, \frac{\pi^{\varepsilon}(x',z')q(x,z|x',z')}{\pi^{\varepsilon}(x,z)q(x',z'|x,z)}\right\} \\ &= \min\left\{1, \frac{p(x')q(x|x')}{p(x)q(x'|x)}\mathbb{1}_{\varepsilon}(z')\right\}, \end{aligned} \tag{2.16}$$

where the second equality is obtained by cancelling out the likelihood densities and noticing that $\mathbb{1}_{\varepsilon}(z) = 1$. The Metropolis-Hastings kernel is summarised in Algorithm 2.5.

Algorithm 2.5 Metropolis-Hastings ABC kernel

Input: current state (X_n, Z_n).
1: Propose $X \sim q(\cdot|X_n)$.
2: Simulate data $Z \sim g(\cdot|X)$.
3: With probability $a(X_n, Z_n, X, Z)$ (Equation 2.16), set $(X_{n+1}, Z_{n+1}) := (X, Z)$, otherwise, $(X_{n+1}, Z_{n+1}) := (X_n, Z_n)$.
Output: updated state (X_{n+1}, Z_{n+1}).

The Metropolis-Hastings kernel is one of the simplest MCMC kernels that can be employed to sample from $\pi^{\varepsilon}(x,z)$. It is more efficient than rejection sampling, as the construction of the kernel ensures that more computational effort is spent in regions of $\mathscr{X}$ that have a high mass under $\pi^{\varepsilon}(x,z)$, as opposed to blindly proposing candidates from the prior only. However, in order to ensure that we have a good approximation of the true posterior, the ball radius ε should be small. A consequence of this is that the probability of hitting the ball is also small. As a result, this MCMC kernel is prone to high rejection rates and slow mixing. Adaptive approaches such as SMC samplers with tempering in the form of a sequence of distributions $\pi^{\varepsilon_1}, \ldots, \pi^{\varepsilon_d}$ where $\varepsilon_1 > \ldots > \varepsilon_d$ have been employed to alleviate this issue (Beaumont et al. [2009]; Del Moral et al. [2012a]; Sisson et al. [2007]). Alternatively, in Lee [2012]; Lee and Łatuszyński [2014], a group of 'robust' MCMC kernels is introduced to improve the mixing of ABC algorithms. The robustness comes from the fact that these algorithms retain a similar behaviour as $\varepsilon \to 0$. These include the 1-hit ABC-MCMC kernel, described in Section 4.4, which includes a 'race' step between the current and proposed states. A relevant feature of the 1-hit kernel is that due to the race step involved, its computational time depends on the value of the current state.

2.8 Reversible jump Markov chain Monte Carlo (RJ-MCMC)

Initially proposed in Green [1995] and further developed in Green [2003], *Reversible Jump Markov Chain Monte Carlo* (RJ-MCMC) is an extension of the Metropolis-Hastings algorithm to more general state spaces. In this framework, the reversible Markov chain constructed is able to jump between parameter subspaces of varying dimensions, which makes RJ-MCMC very useful for applications in model selection (Sisson [2005]) such as changepoint (Del Moral et al. [2006]; Fan and Brooks [2000]; Wyse and Friel [2010]), finite mixture models (Richardson and Green [1997]; Tadesse et al. [2005]) or time series models (Brooks et al. [2003]; Vermaak et al. [2004]) with an unknown number of components, as well as variable selection in regression models (Nott and Leonte [2004]) and knot selection in curve fitting (Denison et al. [1998]).

Let $Y \in \mathscr{Y}$ be a vector of observations and let $\{\mathsf{M}_1, \mathsf{M}_2, \ldots\}$ be a countable collection of candidate models. Each model M_k is associated with the vector of parameters $x_k \in \mathscr{X}^{d_k}$ where d_k is the dimension of the state space, and varies from model to model. Introduce the index $k \in \mathscr{K}$ as an auxiliary model indicator variable, such that we have the following hierarchical structure for the joint distribution of (k, x_k, y)

$$p(k, x_k, y) = p(k)p(x_k \,|\, k)g(y \,|\, x_k),$$

where $p(k)$ is the prior of model indicator k, or in other words the prior for model M_k, $p(x_k \,|\, k)$ is the prior distribution of x_k under model M_k and $g(y \,|\, x_k)$ is the likelihood of the data.

To construct the MCMC sampler, consider $\xi = (k, x_k)$ to be the state of the Markov chain where $p(k, x_k) = p(k)p(x_k \,|\, k)$ is the *joint prior* distribution and the *joint posterior*

$$\pi(\xi) := p(k, x_k | y) = \frac{p(k, x_k)g(y \,|\, x_k)}{\sum_{j \in \mathscr{K}} \int_{\mathscr{X}^{d_j}} p(j, x'_j, y) \mathrm{d}x'_j}$$

is the target or invariant distribution over joint state space $\Xi = \bigcup_{k \in \mathscr{K}} \left(\{k\} \times \mathscr{X}^{d_k} \right)$. Following the Metropolis-Hastings algorithm, propose a new state $\xi' = (k', x_{k'})$ according to proposal density $q_{\mathfrak{M}}(\xi' \,|\, \xi)$ where $\mathfrak{M}$ is the type of move and accept ξ' as the next sample in

the Markov chain with probability

$$a(\xi,\xi') = \min\left\{1, \frac{\pi(\xi')q_{\mathfrak{M}}(\xi\,|\,\xi')}{\pi(\xi)q_{\mathfrak{M}}(\xi'\,|\,\xi)}\right\}$$
$$= \min\left\{1, \frac{p(k',x_{k'})g(y\,|\,x_{k'})}{p(k,x_k)g(y\,|\,x_k)}\frac{q_{\mathfrak{M}}(k,x_k\,|\,k',x_{k'})}{q_{\mathfrak{M}}(k',x_{k'}\,|\,k,x_k)}\right\}, \tag{2.17}$$

otherwise, retain the current state ξ. At each iteration of the RJ-MCMC samples, Fan and Sisson [2011] divide the types of moves $\mathfrak{M}$ into two major categories:

1. *Within-model moves*: fix the model index k and update the parameters x_k following standard Gibbs or Metropolis-Hastings algorithms, for example, in which case the proposal distribution $q_{\mathfrak{M}}$ is standard.

2. *Between-model moves*: jointly update the state $\xi = (k,x_k)$ by proposing a new state $\xi' = (k',x_{k'}) \sim q_{\mathfrak{M}}(\xi'\,|\,\xi)$ and matching dimensions before accepting with probability $a(\xi,\xi')$.

While the within-model moves are straightforward, the between-model moves are more involved as they include a 'dimension matching' element. For example, assume that under the current model M_k, the current state ξ has dimension d_k and the proposed state ξ' under model $\mathsf{M}_{k'}$ has dimension $d_{k'}$ where $d_k \neq d_{k'}$. We must ensure that the dimensions of the numerator and denominator in Equation 2.17 match. For that, introduce an auxiliary variable for the transition $\mathfrak{M}$ from model M_k to model $\mathsf{M}_{k'}$ denoted $u \sim h_{\mathfrak{M}}$ and of dimension r_k where the density $h_{\mathfrak{M}}$ is known. The proposal becomes a function of this joint state, i.e. $\xi' = s_{\mathfrak{M}}(\xi,u)$ where $s_{\mathfrak{M}}$ is the change of variable function. Similarly, for the reverse transition (i.e. model $\mathsf{M}_{k'}$ to M_k), introduce $u' \sim h'_{\mathfrak{M}}$ of dimension $r_{k'}$ such that we recover $\xi = s'_{\mathfrak{M}}(\xi',u') = s'_{\mathfrak{M}}(s_{\mathfrak{M}}(\xi,u),u')$. We now have $d_k + r_k = d_{k'} + r_{k'}$ and hence the dimensions of the joint states (ξ,u) and (ξ',u') match.

When there are multiple possible moves $\mathfrak{M}$, we must generally also include the probability $j_{\mathfrak{M}}$ of choosing a specific move. For example, in mixture models, $j_{\mathfrak{M}}$ can correspond to the probability of performing a 'death' move (i.e. remove a component from the mixture) on a specific mixture component, say component l, and will take the form (probability of death move) $\times$ (probability of selecting component l).

A direct consequence of the detailed balance condition for the Metropolis-Hastings algorithm (Green [2003]) is that we must satisfy the following equality

$$\pi(\xi)q_{\mathfrak{M}}(\xi\,|\,\xi')a(\xi,\xi') = \pi(\xi')q_{\mathfrak{M}}(\xi'\,|\,\xi)a(\xi',\xi). \tag{2.18}$$

Introducing the dimension matching element and its accompanying change of variable, Equation 2.18 becomes

$$\pi(\xi)j_{\mathfrak{M}}(\xi)h_{\mathfrak{M}}(u)a(\xi,\xi') = \pi(\xi')j_{\mathfrak{M}}(\xi')h'_{\mathfrak{M}}(u')a(\xi',\xi)\left|\frac{\partial s_{\mathfrak{M}}(\xi,u)}{\partial(\xi,u)}\right|,$$

where the last factor corresponds to the Jacobian for the change of variable between (ξ,u) and (ξ',u'). Thus, the generalised acceptance probability for transdimensional moves becomes

$$a(\xi,\xi') = \min\left\{1, \frac{\pi(\xi')j_{\mathfrak{M}}(\xi')h'_{\mathfrak{M}}(u')}{\pi(\xi)j_{\mathfrak{M}}(\xi)h_{\mathfrak{M}}(u)}\left|\frac{\partial s_{\mathfrak{M}}(\xi,u)}{\partial(\xi,u)}\right|\right\}.$$

A particular feature of the transdimensionality in RJ-MCMC is that the computational complexity of within-model moves is affected by the dimension d_k of the current model M_k. Therefore, RJ-MCMC is an example of an algorithm in which the compute time depends on the state of the Markov chain, as mentioned in Section 2.6. This will be explored further in Chapter 5.

Chapter 3

Sequential Monte Carlo

3.1 Chapter overview

This chapter continues the literature review and presents sequential Monte Carlo (SMC) methods, or particle filters and smoothers, which will be heavily featured in Chapters 5 and 6. The chapter begins by introducing state space models and the main inference goals when dealing with such models, then reviews some of the most commonly used particle filtering and smoothing algorithms in the literature.

3.2 Introduction to state space models

A *state space model* is one of the most widely used models in the literature, with applications such as biosciences (Boys et al. [2000]; Fearnhead and Vasileiou [2009]; Vahid et al. [2020]), climatology (Hughes et al. [1999]; Zucchini and Guttorp [1991]), signal processing (Li et al. [2000]; Xie and Evans [1991]), computer vision (Carmi et al. [2012]), and finance (Rydén et al. [1998]). See Cappé [2001] for a more complete list of applications.

Let $(X_t)_{t=1}^{\infty}$ and $(Y_t)_{t=1}^{\infty}$ be discrete-time processes taking values in $\mathscr{X}$ and $\mathscr{Y}$, respectively. The *hidden states* or *latent variables* $(X_t)_{t=1}^{\infty}$ form a Markov process with initial distribution $\nu_\theta(x_1)$ and transition density $f_\theta(x_t|x_{t-1})$ where $\theta \in \Theta$ is the vector of *hyperparameters*. The observations $(Y_t)_{t=0}^{\infty}$ are assumed to be independent given the hidden states, with density $g_\theta(y_t|x_t)$. To summarise, the state space model is defined as

$$\begin{aligned} X_1 &\sim \nu_\theta, \qquad X_t|(X_{t-1}=x_{t-1}) \sim f_\theta(\cdot|x_{t-1}), \\ Y_t|(X_t=x_t) &\sim g_\theta(y_t|x_t), \end{aligned}$$

and its *joint density* is given by

$$p_\theta(x_{1:n}, y_{1:n}) = \nu_\theta(x_1) \prod_{t=2}^{n} f_\theta(x_t|x_{t-1}) \prod_{t=1}^{n} g_\theta(y_t|x_t),$$

where $n \in \mathbb{N}$. The first aim is generally to sequentially derive or estimate the posterior $p_\theta(x_t|y_{1:s})$ of the latent state at step t given data observed up to step s. The inference aims can be divided into three categories (Douc et al. [2014]). These are performed keeping the hyperparameters θ fixed, so we henceforth drop dependence on the hyperparameters from the notation for simplicity. The three categories are:

Filtering when $s = t$, for which it is straightforward to derive the following recursion (Doucet et al. [2001]) for $t = 2, \ldots, n$

$$\text{Predict}: \quad p(x_t|y_{1:t-1}) = \int_{\mathscr{X}} f(x_t|x_{t-1}) p(x_{t-1}|y_{1:t-1}) \mathrm{d}x_{t-1}, \tag{3.1}$$

$$\text{Update}: \quad p(x_t|y_{1:t}) = \frac{g(y_t|x_t) p(x_t|y_{1:t-1})}{\int_{\mathscr{X}} g(y_t|x_t) p(x_t|y_{1:t-1}) \mathrm{d}x_t}, \tag{3.2}$$

where $p(x_t|y_{1:t-1})$ is known as the *predictive density* and $p(x_t|y_{1:t})$ is known as the *filtering distribution*.

Smoothing when $s > t$; it is also possible to derive a backward recursion (Briers et al. [2010]) for $t = s-1, \ldots, 1$,

$$p(x_t|y_{1:s}) = p(x_t|y_{1:t}) \int_{\mathscr{X}} \frac{p(x_{t+1}|y_{1:s}) f(x_{t+1}|x_t)}{p(x_{t+1}|y_{1:t})} \mathrm{d}x_{t+1}, \tag{3.3}$$

where $p(x_t|y_{1:s})$ is known as the *marginal smoothing distribution*.

Forecasting when $s < t$.

A filtering pass through the full data $y_{1:n}$ only provides posteriors $p(x_t|y_{1:t})$ based on observations up to $t \leq n$. As a result, it must generally be followed by a backward smoothing pass in order to obtain posteriors $p(x_t|y_{1:n})$ based on all observations for $t = n, \ldots, 1$. In this case, the filtering step is a *forward pass* through the data and the smoothing a *backward pass* in what is known as the *forward filtering-backward smoothing recursion* introduced in Kitagawa [1987].

It is also of interest in particle filtering and smoothing to derive the full posterior $p(x_{1:t}|y_{1:s})$ up to step $s \geq t$, for which it is possible to obtain a recursion for $s = t$:

$$p(x_{1:t}|y_{1:t}) = p(x_{1:t-1}|y_{1:t-1}) \frac{g(y_t|x_t) f(x_t|x_{t-1})}{p(y_t|y_{1:t-1})}.$$

This is referred to as the *joint smoothing distribution*. A backward decomposition can also be obtained for $s > t$ given the recursion in Equation 3.3:

$$p(x_{1:t}|y_{1:s}) = p(x_s|y_{1:s}) \prod_{t=1}^{s-1} \frac{p(x_t|y_{1:t}) f(x_{t+1}|x_t)}{\int_{\mathscr{X}} p(x_t|y_{1:t}) f(x_{t+1}|x_t)}.$$

Finally, it is often also of interest to evaluate *smoothing expectations*

$$\mathscr{S}_t := \mathbb{E}\left[S_t\left(X_{1:t}\right)|y_{1:t}\right] = \int_{\mathscr{X}^t} S_t\left(x_{1:t}\right) p(x_{1:t}|y_{1:t}) \mathrm{d}x_{1:t}, \tag{3.4}$$

a task for which filtering and smoothing are both needed. In this chapter, we focus on various filtering and smoothing methods that have been developed.

3.2.1 Finite state space models

Consider the special case in which the state space is finite, e.g. $\mathscr{X} = \{1, \ldots, K\}$. In this case, the state space model is sometimes referred to as a *Hidden Markov Model* (HMM)[1]. For finite state spaces, filtering and smoothing can be performed exactly by rewriting Equations 3.1-3.3 as follows:

Filtering for discrete state space,

$$p(x_t|y_{1:t-1}) = \sum_{x_{t-1} \in \mathscr{X}} f(x_t|x_{t-1}) p(x_{t-1}|y_{1:t-1}),$$
$$p(x_t|y_{1:t}) = \frac{g(y_t|x_t) p(x_t|y_{1:t-1})}{\sum_{x_t \in \mathscr{X}} g(y_t|x_t) p(x_t|y_{1:t-1})},$$

Smoothing for discrete state space and $s > t$,

$$p(x_t|y_{1:s}) = p(x_t|y_{1:t}) \sum_{x_{t+1} \in \mathscr{X}} \frac{p(x_{t+1}|y_{1:s}) f(x_{t+1}|x_t)}{p(x_{t+1}|y_{1:t})}.$$

For the rest of this chapter, we denote the marginal posterior as follows: for $s, t \in [\![1, n]\!]$,

$$\pi_s(x_t) := p_t(x_t|y_{1:s})$$

for notational simplicity. The forward and backward passes to obtain the marginal smoothing distribution $p(x_t|y_{1:n}) = \pi_n(x_t)$ are summarised in Algorithms 3.1 and 3.2. Similar construc-

[1] Though we note that the terms 'general state space model' and 'HMM' are often used interchangeably, regardless of whether the state space is finite.

tions of these algorithms, which achieve the same result, are also available, such as the Baum-Welch algorithm (Baum et al. [1970]) and in Douc et al. [2014]; Scott [2002].

Algorithm 3.1 Forward Filtering

Initialise $\pi_0(x_1) := p(x_1) = \nu(x_1)$.
for $t = 1, \ldots, n$ **do**
    Forward recursion

$$C_t = \sum_{x_t \in \mathscr{X}} \pi_{t-1}(x_t) g(y_t|x_t),$$
$$\pi_t(x_t) = \frac{1}{C_t} \pi_{t-1}(x_t) g(y_t|x_t),$$

    if $t < n$ **do**

$$\pi_t(x_{t+1}) = \sum_{x_t \in \mathscr{X}} \pi_t(x_t) f(x_{t+1}|x_t).$$

end for
Output: $\{\pi_t(x_t)\}_{t=1}^n$.

Algorithm 3.2 Backward Smoothing

Input: $\{\pi_t(x_t)\}_{t=1}^n$ output from Algorithm 3.1.
Initialise with $\pi_n(x_n)$.
for $t = n-1, \ldots, 1$ **do**
    Backward smoothing

$$D_t(x_{t+1}) = \sum_{x_t \in \mathscr{X}} \pi_t(x_t) f(x_{t+1}|x_t),$$
$$\pi_n(x_t) = \pi_t(x_t) \sum_{x_{t+1} \in \mathscr{X}} \pi_n(x_{t+1}) \frac{f(x_{t+1}|x_t)}{D_t(x_{t+1})}.$$

end for
Output: $\{\pi_n(x_t)\}_{t=1}^n$.

Note that the density $\pi_s(x_t)$ is a vector of length K (one component per possible state). The need to sum over the state space $\mathscr{X}$ in order to update $\pi_s(x_t)$ makes each iteration of these algorithms $\mathscr{O}(K^2)$ in complexity, which can become computationally prohibitive for large K. Additionally, the need to store $\{\pi_t(x_t)\}_{t=1}^n$ in order to perform backward smoothing can be costly in terms of memory for a particularly large n. An alternative way of implementing the filtering and smoothing algorithms is to approximate them via particle filtering and smoothing (see Sections 3.3 and 3.4).

3.2.2 Linear Gaussian state space models

One of the most basic families of infinite state space models is known as the *linear Gaussian state space models*, in which both f and g are assumed to be Gaussian densities, i.e. let $\mathscr{X} = \mathbb{R}^c$ and $\mathscr{Y} = \mathbb{R}^d$, then for $t > 1$

$$\begin{aligned} X_t &= A_t X_{t-1} + V_t, & V_t &\sim \mathscr{N}(0, Q_t), \\ Y_t &= B_t X_t + W_t, & W_t &\sim \mathscr{N}(0, R_t), \end{aligned}$$

where the matrices $A_t \in \mathbb{R}^{c\times c}$ and $B_t \in \mathbb{R}^{d\times c}$. We also assume the initial state X_1 is Gaussian with mean x_0 and covariance Σ_0. The Gaussian assumption makes it possible to evaluate the recursions in Equations 3.1 and 3.2 directly via the *Kalman filter* (Kalman [1960]; Kalman and Bucy [1961]). Denote the predicted state $X_{t|s} := \mathbb{E}[X_t|Y_{1:s}]$ and covariance $P_{t|s} := \mathrm{Cov}[X_t|Y_{1:s}] = \mathbb{E}\left[(X_t - X_{t|s})(X_t - X_{t|s})^\intercal | Y_{1:s}\right]$. For a sample $y_{1:n}$ of n observations, the Kalman filter to sequentially evaluate the filtering distribution $\pi_t(x_t) := p(x_t|y_{1:t})$ for $t = 1,\ldots,n$ is described in Algorithm 3.3. Derivations of the Kalman filter are available in Byron et al. [2004]; Meinhold and Singpurwalla [1983]; West and Harrison [2006].

Algorithm 3.3 The Kalman Filter

1: Initialise $X_{1|1} = x_0$ and $P_{1|1} = \Sigma_0$.
2: **for** $t = 2,\ldots,n$ **do**
3: *Predict* state and covariance estimates

$$\begin{aligned} X_{t|t-1} &= A_t X_{t-1|t-1}, \\ P_{t|t-1} &= A_t P_{t-1|t-1} A_t^\intercal + Q_t. \end{aligned}$$

4: *Update* state and covariance estimates

$$\begin{aligned} K_t &= P_{t|t-1} B_t \left(B_t P_{t-1|t-1} B_t^\intercal + R_t\right)^{-1}, \\ X_{t|t} &= X_{t|t-1} + K_t \left(Y_t - B_t X_{t|t-1}\right), \\ P_{t|t} &= \left(\mathbb{I} - K_t B_t\right) P_{t|t-1}, \end{aligned}$$

where K_t is known as the *Kalman gain*.
5: **end for**
Output: $\left\{(X_{t|t}, P_{t|t})\right\}_{t=1}^n$.

Once the Kalman filter has been run, it is also possible to perform the backward pass in Equation 3.3 exactly in order to obtain the marginal smoothing distribution $\pi_n(x_t)$ for $t = n,\ldots,1$. The backward smoothing pass is described in Algorithm 3.4. See Ansley and Kohn [1982]; Byron et al. [2004]; Douc et al. [2014] for a derivation.

Algorithm 3.4 The Kalman Smoother

Input: $\left\{(X_{t|t}, P_{t|t})\right\}_{t=1}^{n}$ obtained in Algorithm 3.3.

1: Initialise with $(X_{n|n}, P_{n|n})$.

2: **for** $t = n-1, \ldots, 1$ **do**

3:

$$\begin{aligned} J_t &= P_{t|t} A_{t+1}^{\mathsf{T}} \left(A_{t+1} P_{t|t} A_{t+1}^{\mathsf{T}} + Q_{t+1}\right)^{-1}, \\ X_{t|n} &= X_{t|t} + J_t \left(X_{t+1|n} - A_{t+1} X_{t|t}\right), \\ P_{t|n} &= P_{t|t} + J_t \left(P_{t+1|n} - A_{t+1} P_{t|t} A_{t+1}^{\mathsf{T}} - Q_{t+1}\right) J_t^{\mathsf{T}}. \end{aligned}$$

4: **end for**

Output: $\left\{(X_{t|n}, P_{t|n})\right\}_{t=1}^{n}$.

The Kalman filter and smoother are very convenient, as they are exact and easy to implement. However, for large c and d, the matrix multiplications and inversions can easily become computationally prohibitive. Additionally, the Gaussian assumption limits the range of models that can be explored, which is why the alternative approach known as particle filtering and smoothing is often employed nowadays.

3.3 Particle filtering

Apart from very simple cases, as described in Sections 3.2.1 and 3.2.2, the recursions in Equations 3.1-3.4 are impossible to evaluate. Fortunately, a number of methods, known as *Sequential Monte Carlo* (SMC) have been developed, which allow particle approximations to be obtained of the form

$$\hat{\pi}_t(x_t) = \sum_{i=1}^{N} \omega_t^{(i)} \delta_{X_t^{(i)}}(x_t), \tag{3.5}$$

where $\delta_{x_0}(x)$ denotes the Dirac delta mass located at x_0, $X_t^{(1:N)}$ denotes the particles and $\omega_t^{(1:N)}$ their corresponding normalised weights, i.e. $\omega_t^{(i)} > 0$ and $\sum_{i=1}^{N} \omega_t^{(i)} = 1$ for $t = 1, \ldots, n$. Combined, $(X_t^{(1:N)}, \omega_t^{(1:N)})$ makes up the particle approximation of the posterior $\pi_t(x_t)$. See Cappé et al. [2006]; Chopin and Papaspiliopoulos [2020]; Douc et al. [2014]; Doucet et al. [2001] for comprehensive reviews of SMC methods. In this section, we focus on various SMC algorithms that have been developed to perform filtering, known as *particle filters*. The particle smoothers, developed for smoothing purposes, will be explored in Section 3.4.

3.3.1 Preliminaries

When it is possible to sample directly from the filtering distribution $\pi_t(x_t)$, the weights are uniform, i.e. $\omega_t^{(i)} = \frac{1}{N}$ for all $i \in [\![1,N]\!]$ and the Monte Carlo approximations in Equations 3.5 are straightforward to obtain by marginalising the Monte Carlo approximation of the joint posterior. This is known as *perfect Monte Carlo*. However, this is very rarely the case.

Another basic approach is known as *importance sampling*. It consists of introducing an *importance distribution* $\eta_t(x_t) := q(x_t|y_{1:t})$ from which is easy to sample, and then evaluating the (unnormalised) *importance weights* given by

$$w_t(x_t) := \frac{\pi_t(x_t)}{\eta_t(x_t)},$$

which can then be normalised as follows:

$$\omega_t^{(i)} = \frac{w_t\left(X_t^{(i)}\right)}{\sum_{j=1}^N w_t\left(X_t^{(j)}\right)},$$

for all $i \in [\![1,N]\!]$ in order to obtain a particle approximation of the posterior. For more advanced problems, approaches such as MCMC can also be employed if needed. However, importance sampling under this basic form is not convenient for recursive estimation, as it simulates and weights the filtering distribution with respect to the full data $y_{1:t}$ up to step t. If one wishes to obtain the filtering distribution for $y_{1:t+1}$, they must restart computations from scratch, and the computational complexity of evaluating $\pi_t(x_t)$ increases with $t = 1,\ldots,n$. The same observation can be made for MCMC methods, which is why recursive approaches are generally preferable for sequential data.

3.3.2 Sequential importance sampling (SIS)

It is possible to implement importance sampling sequentially in order to sample from the filtering distribution $\pi_t(x_t)$, in what is known as *Sequential Importance Sampling* (SIS). Rewrite the recursion on the filtering distribution in Equations 3.1 and 3.2 as follows:

$$\pi_t(x_t) \propto g(y_t|x_t) \int_{\mathscr{X}} f(x_t|x_{t-1})\pi_{t-1}(x_{t-1})\mathrm{d}x_{t-1}. \tag{3.6}$$

Since the integral in Equation 3.6 is generally intractable, we can employ a Monte Carlo approximation. Let $(X_{1:t-1}^{(1:N)}, \omega_{t-1}^{(1:N)})$ be the particle approximation of the filtering distribution

$\pi_{t-1}(x_{t-1})$. Approximate Equation 3.6 as follows:

$$\hat{\pi}_t(x_t) \propto g(y_t|x_t) \sum_{i=1}^{N} \omega_{t-1}^{(i)} f\left(x_t|X_{t-1}^{(i)}\right). \tag{3.7}$$

This can be done via importance sampling with *proposal density* $\eta_t(x_t|x_{t-1}) := q(x_t|x_{t-1}, y_t)$. Note that a proposal density of the form $q(x_t|x_{t-1}, y_{1:t})$ could also be used; our choice here stems from the fact that the optimal proposal is of the form $q(x_t|x_{t-1}, y_t)$. From this, we can update the importance weights as follows:

$$w_t(x_t) \propto w_{t-1}(x_{t-1}) \underbrace{\frac{g(y_t|x_t) f(x_t|x_{t-1})}{\eta_t(x_t|x_{t-1})}}_{\tilde{w}(x_{t-1}, x_t)}, \tag{3.8}$$

where $\tilde{w}(x_{t-1}, x_t)$ is known as the *incremental weight*. Note that the fact that the recursion is only up to a constant is not an issue, as the normalising constants cancel out when normalising the weights. The SIS algorithm is summarised in Algorithm 3.5.

Algorithm 3.5 Sequential Importance Sampling (SIS)

Where (i) *appears, the operation is performed for all* $i \in [\![1, N]\!]$.

Initialise at $t = 1$:

1: Sample $X_1^{(i)} \sim \eta_1$.

2: Compute the importance weights $w_t\left(X_1^{(i)}\right) = \dfrac{\pi_1\left(X_1^{(i)}\right)}{\eta_1\left(X_1^{(i)}\right)}$ and normalise them to obtain $\omega_1^{(i)}$.

Given weighted particle sample $\left(X_{t-1}^{(1:N)}, \omega_{t-1}^{(1:N)}\right)$.

3: **for** $t = 2, \ldots, n$ **do**

4: Sample $X_t^{(i)} \sim \eta_t\left(\cdot|X_{t-1}^{(i)}\right)$.

5: Compute the incremental weights $\tilde{w}\left(X_{t-1}^{(i)}, X_t^{(i)}\right)$ according to Equation 3.8, then normalise the importance weights to obtain $\omega_t^{(i)}$ as follows:

$$\omega_t^{(i)} = \frac{\omega_{t-1}^{(i)} \tilde{w}\left(X_{t-1}^{(i)}, X_t^{(i)}\right)}{\sum_{j=1}^{N} \omega_{t-1}^{(j)} \tilde{w}\left(X_{t-1}^{(j)}, X_t^{(j)}\right)}.$$

6: **end for**

It is also straightforward to implement SIS to sequentially sample from the joint smoothing posterior $\pi_t(x_{1:t}) := p(x_{1:t}|y_{1:t})$ directly. Indeed, most of the particle filtering algorithms in this section can easily be adapted to sequentially sample from the joint smoothing posterior

instead of the filtering distribution (Cappé et al. [2007]), using the same incremental weights. However, in practice, sampling from a very high dimensional state space causes the weights to degenerate very quickly, so that after relatively few steps t, very few particles contribute to the approximation. As a result, the low-dimensional filtering distribution is generally privileged in particle filtering.

3.3.3 Sequential importance resampling (SIR)

When using SIS, a well-known issue becomes apparent as the algorithm progresses through the steps $t = 1, \ldots, n$. The weights of a few particles will become much larger than the rest, and in the end, the algorithm will likely return a single particle with weight 1 while all the other particles have zero weight (Del Moral and Doucet [2003]). This is known as *weight degeneracy* (Doucet et al. [2000]; Doucet and Johansen [2009]; Kong et al. [1994]). Fortunately, it is possible to avoid this issue by introducing a *resampling step* (Gordon et al. [1993]). This consists of drawing indices $\iota_t^{(1:N)}$ with probabilities corresponding (or proportional) to the normalised weights $\omega_t^{(1:N)}$, which we denote as follows:

$$\iota_t^{(i)} \sim \mathbb{P}\left(\omega_t^{(1:N)}\right), \qquad i = 1, \ldots, N.$$

Many resampling methods are available. See Gerber et al. [2019]; Murray et al. [2016a] for reviews and theoretical results on the available resampling schemes. However, while crucial, resampling leads to an increase in the variance of Monte Carlo approximations, and doing it too often leads to a lower number of distinct particles, so SMC algorithms often opt for an adaptive resampling strategy (Del Moral et al. [2012b]). This strategy consists of only resampling the particles when necessary, i.e. when the quality of the particle approximation is poor. A common method for assessing this is to evaluate the effective sample size given by

$$ESS_t = \left[\sum_{i=1}^{N} \left(\omega_t^{(i)}\right)^2\right]^{-1} \tag{3.9}$$

and to resample if the effective sample size falls below a user-defined threshold T_{ESS}. The adaptive resampling step is summarised in Algorithm 3.6.

For the rest of this chapter, we refer to the adaptive resampling step of algorithms as

$$\left(\iota_t^{(1:N)}, \omega_t^{(1:N)}\right) := \mathsf{resample}\left(\omega_t^{(1:N)}\right),$$

where resampling at every iteration is simply a special case of the adaptive resampling scheme where $T_{ESS} = N$.

Algorithm 3.6 Adaptive Resampling

Where (i) *appears, the operation is performed for all* $i \in [\![1, N]\!]$.

Input: At step t, normalised weights $\omega_t^{(1:N)}$.

Evaluate effective sample size ESS_t using Equation 3.9.
if $ESS_t < T_{ESS}$ **then** ▷ *Resample*
    Sample indices $\iota_t^{(i)} \sim \mathbb{P}\left(\omega_t^{(1:N)}\right)$.
    Reset weights $\tilde{\omega}_t^{(i)} := \frac{1}{N}$.
else ▷ *Do not resample*
    $\iota_t^{(i)} := i$
    Keep weights $\tilde{\omega}_t^{(i)} := \omega_t^{(i)}$.
end if

Output: $\left(\iota_t^{(1:N)}, \tilde{\omega}_t^{(1:N)}\right)$.

Introducing resampling to the SIS algorithm results in the *Sequential Importance Resampling* (SIR) algorithm summarised in Algorithm 3.7. Note that the SIR algorithm displays the three key steps of particle filters, namely *resample*→*propagate*→*weight*. These three key steps will be present in all particle filtering algorithms in some form or another for the rest of this chapter.

Algorithm 3.7 Sequential Importance Resampling (SIR)

Where (i) *appears, the operation is performed for all* $i \in [\![1, N]\!]$.

Initialise at $t = 1$:
Sample $X_1^{(i)} \sim \eta_1$.
Compute the importance weights $w_1\left(X_1^{(i)}\right) = \dfrac{\pi_1\left(X_1^{(i)}\right)}{\eta_1\left(X_1^{(i)}\right)}$ and normalise them to obtain $\omega_1^{(i)}$.

Given weighted particle sample $\left(X_{t-1}^{(1:N)}, \omega_{t-1}^{(1:N)}\right)$,
for $t = 2, \ldots, n$ **do**
    (RESAMPLE) $\left(\iota_t^{(1:N)}, \omega_t^{(1:N)}\right) := \mathsf{resample}\left(\omega_t^{(1:N)}\right)$.
    (PROPAGATE) Sample $X_t^{(i)} \sim \eta_t\left(\cdot | X_{t-1}^{(\iota_t^{(i)})}\right)$.
    (WEIGHT) Compute the incremental weights $\tilde{w}\left(X_{t-1}^{(\iota_t^{(i)})}, X_t^{(i)}\right)$ according to Equation 3.8, then normalise the importance weights to obtain $\omega_t^{(i)}$.
end for

3.3.4 The bootstrap filter

Now that we have described the basic form of a particle filter, we can begin exploring a few of the most commonly used particle filters. The *bootstrap filter* was introduced in Gordon et al. [1993] and can be seen as a special case of SIR where each proposal of the importance distribution corresponds to the Markov state transition density, i.e. $\eta_t(x_t|x_{t-1}) = f(x_t|x_{t-1})$. It is straightforward to establish from Equation 3.8 that as a result of this, the incremental weights are given by

$$\tilde{w}(x_{t-1}, x_t) = g(y_t|x_t).$$

The bootstrap filter (of which a typical step is summarised in Algorithm 3.8) is one of the most convenient and easy to implement algorithms. It is very flexible, as it only suffices to adapt the state transition $f(x_t|x_{t-1})$ and observation $g(y_t|x_t)$ densities to apply it to a new state space model in which the former can be sampled from and the latter can be evaluated pointwise. It is therefore widely used in the literature (Doucet et al. [2001]).

Algorithm 3.8 The Bootstrap filter

Where (i) *appears, the operation is performed for all* $i \in [\![1, N]\!]$.

Input: weighted particle sample $\left(X_{t-1}^{(1:N)}, \omega_{t-1}^{(1:N)}\right)$.

1: $\left(\iota_t^{(1:N)}, \omega_t^{(1:N)}\right) := \mathsf{resample}\left(\omega_t^{(1:N)}\right)$.

2: Sample $X_t^{(i)} \sim f\left(\cdot|X_{t-1}^{(\iota_t^{(i)})}\right)$.

3: Compute the incremental weights $\tilde{w}\left(X_t^{(i)}\right) = g\left(y_t|X_t^{(i)}\right)$ then normalise the importance weights to obtain $\omega_t^{(i)}$.

Output: updated particle sample $\left(X_t^{(1:N)}, \omega_t^{(1:N)}\right)$.

An issue the bootstrap filter faces is that its proposal density $f(x_t|x_{t-1})$ is not 'aware' of the quality of its proposals. For example, a model in which the likelihood density $g(y_t|x_t)$ at step t has a peak in a small area of the state space will yield few particles with high importance weights and many particles with very low weights, thus increasing the variance of the weights. One possible solution is to simply increase the size N of the particle population, though this comes at a computational cost. An alternative solution is for the proposal density to suggest an 'educated' proposal based on the current observation y_t. If the proposal matches the target closely, the values of weights will be closer to equal and thus their variance reduced. Note that we can write

$$g(y_t|x_t)f(x_t|x_{t-1}) = p(x_t, y_t|x_{t-1}) = p(x_t|y_t, x_{t-1})p(y_t|x_{t-1}), \tag{3.10}$$

so setting the proposal $\eta_t^{\text{opt}}(x_t|x_{t-1}) = p(x_t|y_t, x_{t-1})$ yields the optimal incremental weights $\tilde{w}^{\text{opt}}(x_{t-1}) = p(y_t|x_{t-1})$ which do not depend on the value of the state X_t. This was employed in Kong et al. [1994]; Liu and Chen [1995] and is the optimal choice of proposal as it minimises the variance of the incremental weights (see Nemeth [2014] Proposition 3.4.1 for a proof). The issue then is that this optimal filter does require for it to be possible to sample from the posterior $p(x_t|y_t, x_{t-1})$, which is rarely the case.

3.3.5 The auxiliary particle filter (APF)

First of all, approximate the optimal proposal by defining the *auxiliary density* as follows:

$$\rho(x_t, y_t|x_{t-1}) = \rho(x_t|y_t, x_{t-1})\rho(y_t|x_{t-1}), \tag{3.11}$$

where $\rho(x_t|y_t, x_{t-1})$ is easy to sample from and $\rho(y_t|x_{t-1})$ can be evaluated point-wise.

Given weighted particle sample $(X_{t-1}^{(1:N)}, \omega_{t-1}^{(1:N)})$, rewrite the particle approximation of the filtering distribution in Equation 3.7 using the equality in Equation 3.10:

$$\hat{\pi}(x_t) \propto \sum_{i=1}^{N} \underbrace{\omega_{t-1}^{(i)} p\left(y_t|X_{t-1}^{(i)}\right)}_{\bar{\omega}_{t-1}^{(i)}} p\left(x_t|y_t, X_{t-1}^{(i)}\right),$$

which can be seen as a mixture of densities $p(x_t|y_t, X_{t-1}^{(i)})$ with mixture weights $\bar{\omega}_{t-1}^{(i)}$. At step t, the *auxiliary particle filter* (APF) introduced by Pitt and Shephard [1999] (see also Carpenter et al. [1999]) targets the augmented density

$$\begin{aligned}\pi_t(x_t, i) := p(x_t, i|y_{1:t}) &\propto \omega_{t-1}^{(i)} p\left(y_t|X_{t-1}^{(i)}\right) p\left(x_t|y_t, X_{t-1}^{(i)}\right) \\ &= \omega_{t-1}^{(i)} g\left(y_t|x_t\right) f\left(x_t|X_{t-1}^{(i)}\right),\end{aligned}$$

where i is the index for the i-th mixture component (and also the i-th particle in the population). The APF also modifies the importance distribution by simulating an index for each particle according to a distribution whose weights $\bar{\omega}_{t-1}^{(1:N)}$ take into account its 'compatibility' with the new observation y_t (Whiteley and Johansen [2010]). As a result, the proposal density for the i-th particle is of the form

$$\eta_t(x_t, i) := q_t(x_t, i|y_{1:t}) = \omega_{t-1}^{(i)} \rho\left(y_t|X_{t-1}^{(i)}\right) \rho\left(x_t|y_t, X_{t-1}^{(i)}\right),$$

which is equivalent to resampling the particle population using the weights $\bar{\omega}_{t-1}^{(i)}$ and then

propagating the particles by sampling from the density $\rho(\cdot|y_t, x_{t-1})$. This results in the following importance weights:

$$w_t(x_{t-1}, x_t) = \frac{\pi(x_t, i)}{\eta(x_t, i)} \propto \frac{g(y_t|x_t) f(x_t|x_{t-1})}{\rho(y_t|x_{t-1}) \rho(x_t|y_t, x_{t-1})}. \tag{3.12}$$

Given particle population $X_t^{(1:N)}$, the importance weights can be normalised to obtain $\omega_t^{(1:N)}$. A typical step of the APF incorporating adaptive resampling (Papaspiliopoulos [2010]) is summarised in Algorithm 3.9.

Algorithm 3.9 Auxiliary particle filter (APF)

Where (i) *appears, the operation is performed for all* $i \in [\![1, N]\!]$.

Input: weighted particle sample $\left(X_{t-1}^{(1:N)}, \omega_{t-1}^{(1:N)}\right)$.

1: Evaluate effective sample size ESS_{t-1} using Equation 3.9.
2: **if** $ESS_{t-1} < T_{ESS}$ **then**
3: Sample indices $\iota_t^{(i)} \sim \mathbb{P}\left(\omega_{t-1}^{(1:N)} \rho\left(y_k | X_{t-1}^{(1:N)}\right)\right)$.
4: Set $\upsilon_{t-1}^{(i)} := \rho\left(y_t | X_{t-1}^{(\iota_t^{(i)})}\right)^{-1}$.
5: **else**
6: Keep $\iota_t^{(i)} := i$.
7: Set $\upsilon_{t-1}^{(i)} := \omega_{t-1}^{(i)}$.
8: **end if**
9: Sample $X_t^{(i)} \sim \rho\left(\cdot | y_t, X_{t-1}^{(\iota_t^{(i)})}\right)$.
10: Update weights $w_t\left(X_{t-1}^{(\iota_t^{(i)})}, X_t^{(i)}\right) \propto \upsilon_{t-1}^{(i)} \tilde{w}_t\left(X_{t-1}^{(\iota_t^{(i)})}, X_t^{(i)}\right)$, where

$$\tilde{w}_t(x_{t-1}, x_t) = \frac{g(y_t|x_t) f(x_t|x_{t-1})}{\rho(x_t|y_t, x_{t-1})},$$

and normalise to obtain $\omega_t^{(1:N)}$.

Output: updated particle sample $\left(X_t^{(1:N)}, \omega_t^{(1:N)}\right)$.

A few observations can be made. Firstly, note that in the optimal case where $p(x_t|y_t, x_{t-1})$ can be sampled from and $p(y_t|x_{t-1})$ can be evaluated pointwise, the importance weights in Equation 3.12 are all equal. This is referred to in Pitt and Shephard [1999] as the *fully adapted APF*. Secondly, by setting $\rho(y_t|x_{t-1}) = 1$, we simply recover the SIR algorithm. Lastly, if in addition to that, we set $\rho(x_t|y_t, x_{t-1}) = f(x_t|x_{t-1})$, then we recover the bootstrap filter. The last two observations illustrate the fact that the APF can be thought of as a generalised SIR algorithm.

3.3.6 The marginal particle filter (MPF)

Another approach to reduce the variance of the importance weights is known as the *marginal particle filter* (MPF). It was first introduced in Klaas et al. [2012] as a method to sample directly from the filtering distribution $\pi_t(x_t)$ instead of the joint smoothing distribution $\pi(x_{1:t})$. The MPF approximates the recursion in Equation 3.6 directly by defining a 'marginal' proposal density as follows. Given weighted particle population $(X_{t-1}^{(1:N)}, \omega_{t-1}^{(1:N)})$,

$$\eta_t(x_t) := \sum_{i=1}^{N} \omega_{t-1}^{(i)} \eta_t\left(x_t | X_{t-1}^{(i)}\right),$$

and the importance weights are defined as

$$w_t(x_t) \propto \frac{g(y_t|x_t) \sum_{j=1}^{N} \omega_{t-1}^{(j)} f\left(x_t | X_{t-1}^{(j)}\right)}{\sum_{j=1}^{N} \omega_{t-1}^{(j)} \eta_t\left(x_t | X_{t-1}^{(j)}\right)}.$$

The optimal choice of proposal here is to take $\eta_t^{\text{opt}}(x_t|x_{t-1}) = p(x_t|y_t, x_{t-1}) p(y_t|x_{t-1})$ as this results in equal weights. If the densities $p(x_t|y_t, x_{t-1})$ and $p(y_t|x_{t-1})$ are unavailable, it is straightforward to adapt the APF into what is referred to in Klaas et al. [2012] as the *auxiliary marginal particle filter* (AMPF), by adapting the proposal density to approximate the optimal proposal

$$\eta_t(x_t) = \sum_{i=1}^{N} \omega_{t-1}^{(i)} \rho\left(y_t | X_{t-1}^{(i)}\right) \rho\left(x_t | y_t, X_{t-1}^{(i)}\right),$$

where $\rho(y_t|x_{t-1})$ and $\rho(x_t|y_t, x_{t-1})$ are defined in Equation 3.11. The importance weights are similarly adapted to return

$$w_t(x_t) \propto \frac{g(y_t|x_t) \sum_{j=1}^{N} \omega_{t-1}^{(j)} f\left(x_t | X_{t-1}^{(j)}\right)}{\sum_{j=1}^{N} \omega_{t-1}^{(j)} \rho\left(y_t | X_{t-1}^{(j)}\right) \rho\left(x_t | y_t, X_{t-1}^{(j)}\right)}.$$

A typical step of the AMPF is summarised in Algorithm 3.10. For Step 1 of the algorithm, Klaas et al. [2012] suggest stratified sampling, and Gustafsson [2013] discusses how to perform this step. An important observation to make is that the computational cost of MPF algorithms is $\mathcal{O}(N^2)$, so the choice of application should ensure that the variance reducing benefits of MPF are not outweighed by the performance costs. It is also possible to reduce this complexity to linear, as discussed in Gustafsson [2013].

Note that once again the bootstrap particle filter can be recovered by setting $\rho(x_t|y_t, x_{t-1}) = f(x_t|x_{t-1})$ and $\rho(y_t|x_{t-1}) = 1$.

Algorithm 3.10 Auxiliary marginal particle filter (AMPF)

Where (i) *appears, the operation is performed for all* $i \in [\![1,N]\!]$.

Input: weighted particle sample $\left(X_t^{(1:N)}, \omega_t^{(1:N)}\right)$.

1: Sample $X_t^{(i)} \sim q(\cdot|y_{1:t})$ where

$$q(x_t|y_{1:t}) \propto \sum_{i=1}^{N} \omega_{t-1}^{(i)} \rho\left(y_t|X_{t-1}^{(i)}\right) \rho\left(X_t|y_t, X_{t-1}^{(i)}\right).$$

2: Update weights

$$w_t\left(X_t^{(i)}\right) \propto \frac{g\left(y_t|X_t^{(i)}\right) \sum_{j=1}^{N} \omega_{t-1}^{(j)} f\left(X_t^{(i)}|X_{t-1}^{(j)}\right)}{\sum_{j=1}^{N} \omega_{t-1}^{(j)} \rho\left(y_t|X_{t-1}^{(j)}\right) \rho\left(X_t^{(i)}|y_t, X_{t-1}^{(j)}\right)}$$

and normalise to obtain $\omega_t^{(1:N)}$.

Output: updated particle sample $\left(X_t^{(1:N)}, \omega_t^{(1:N)}\right)$.

3.4 Particle smoothing

Recall that particle filtering only provides estimates for the filtering distribution $\pi_t(x_t)$ up to step t. If one wishes to estimate the joint posterior, or joint smoothing distribution, $\pi_s(x_{1:t}) := p(x_{1:t}|y_{1:s})$ for $t \leq s \leq n$, particle smoothing must also be employed. Some of the main motivations for particle smoothing are: obtaining a 'corrected' or smoothed posterior for state X_t in light of new observations since step t, and parameter inference such as MLE via EM or the estimation of the score and OIM.

3.4.1 Preliminaries

Let $(X_t^{(1:N)}, \omega_t^{(1:N)})$ be the particle population at steps $t = 1, \ldots, n$ obtained using any particle filter. The naïve approach to performing particle smoothing, referred to in Douc et al. [2014] as the *poor man's smoother*, is to keep track of the resampled indices over time as the particle filter runs so that it is possible to reconstruct a particle's *ancestral path*. If $\iota_t^{(i)}$ is the index of the particle at step $t-1$, known as the *ancestor*, from which particle $X_t^{(i)}$ was generated for $t = 1, \ldots, n$, define the sequence of *ancestor variables* $\zeta_{1:n}^{(i)}$ so that for $t = 1, \ldots, n-1$,

$$\zeta_n^{(i)} = i \qquad \zeta_t^{(i)} = \iota_{t+1}^{(\zeta_{t+1}^{(i)})},$$

where $\zeta_t^{(i)}$ is the index of the ancestor at step t of what is now (at step n) the i-th particle. Then, at any step t, the ancestral path of particle $X_t^{(i)}$ can be recovered as

$$Z_t^{(i)} := \left(X_1^{(\zeta_1^{(i)})}, \ldots, X_t^{(\zeta_t^{(i)})} \right).$$

from this, a particle approximation $(Z_t^{(1:N)}, \omega_t^{(1:N)})$ of the joint smoothing distribution $\pi_t(x_{1:t})$ can be recovered, and from this, it is straightforward to marginalise in order to obtain particle approximations of the marginal smoothing distributions $\pi_s(x_t)$ for $t \leq s \leq n$. However, unless the number N of particles is asymptotically large, this is a very poor approximation. Indeed, because of the resampling step, the number of unique values representing the marginal at any fixed step t quickly falls to 1 as the particle smoother progresses (see Douc et al. [2014], Example 11.2). This problem is known as *path degeneracy*.

While it is essential to keep the resampling step in particle filters, it makes path degeneracy unavoidable in naïve implementations such as the poor man's smoother. Fortunately, a variety of particle smoothers have been developed to mitigate, and even eliminate, this issue.

3.4.2 Forward-filtering backward smoothing (FFBSm)

The *forward-filtering backward smoothing* (FFBSm) algorithm (Doucet et al. [2000]; Hürzeler and Künsch [1998]; Kitagawa [1987]) is simply an extension of the forward backward approach employed for a finite state space model in Section 3.2.1 to particle methods and does not suffer from path degeneracy. A particle approximation of filtering distributions $\pi_t(x_t)$ for steps $t = 1, \ldots, n$ can be obtained in a forward pass by employing any available particle filter. For the backward pass, recall the recursion for the marginal smoothing distribution in Equation 3.3: for $t = n-1, \ldots, 1$,

$$\pi_n(x_t) = \pi_t(x_t) \int_{\mathscr{X}} \frac{\pi_n(x_{t+1}) f(x_{t+1}|x_t)}{\int_{\mathscr{X}} \pi_n(x_{t+1}) f(x_{t+1}|x_t) \mathrm{d}x_t} \mathrm{d}x_{t+1}. \tag{3.13}$$

The FFBSm algorithm simply aims to employ the Monte Carlo approximation of Equation 3.13. Given particle populations $(X_t^{(1:N)}, \omega_t^{(1:N)})$ approximating the filtering distributions $\pi_t(x_t)$ for $t = 1, \ldots, n$, the backward smoothing pass updates or 'smooths out' the particle weights in order to return the marginal smoothing distributions as follows. Note first of all that the filtering and marginal smoothing distributions coincide at step n, i.e. for $\pi_n(x_n)$. Start

from the particle population $(X_n^{(1:N)}, \omega_n^{(1:N)})$ of $\pi_n(x_n)$, then for steps $t = n-1, \ldots, 1$ we have

$$\hat{\pi}_n(x_t) = \sum_{i=1}^{N} \upsilon_t^{(i)} \delta_{X_t^{(i)}}(x_t),$$

where $\upsilon_t^{(1:N)}$ denotes the self-normalising, *marginal smoothing weights* given by

$$\upsilon_t^{(i)} = \omega_t^{(i)} \sum_{j=1}^{N} \frac{\upsilon_{t+1}^{(j)} f\left(X_{t+1}^{(j)} | X_t^{(i)}\right)}{\sum_{k=1}^{N} \omega_t^{(k)} f\left(X_{t+1}^{(j)} | X_t^{(k)}\right)}, \tag{3.14}$$

and $\upsilon_n^{(1:N)} = \omega_n^{(1:N)}$. Then, the weighted sample $(X_t^{(1:N)}, \upsilon_t^{(1:N)})$ is a particle approximation of the marginal smoothing distribution $\pi_n(x_t)$. This particular implementation of the backward pass is sometimes known as marginal backward smoothing and is summarised in Algorithm 3.11. Note that using the backward decomposition in Equation 3.2, it is also possible to directly obtain a particle approximation of the joint smoothing distribution $\pi_n(x_{1:t})$ given by

$$\hat{\pi}_n(x_{1:t}) = \sum_{i_1=1}^{N} \cdots \sum_{i_t=1}^{N} \left(\prod_{k=1}^{t-1} \frac{\omega_k^{(i_k)} f\left(X_{k+1}^{(i_{k+1})} | X_k^{(i_k)}\right)}{\sum_{j=1}^{N} \omega_k^{(j)} f\left(X_{k+1}^{(i_{k+1})} | X_k^{(j)}\right)} \right) \omega_t^{(i_t)} \delta_{X_1^{i_1}, \ldots, X_t^{i_t}}(x_{1:t}). \tag{3.15}$$

However, computing Equation 3.15 is a hugely intensive task of computational complexity $\mathcal{O}(N^n)$, so the motivation behind the implementation of marginal backward smoothing is justified. While more practical and certainly cheaper, each iteration of the FFBSm algorithm is still $\mathcal{O}(N^2)$, and the algorithm requires that the full history of estimated filtering distributions for $t = 1, \ldots, n$ be retained, so also comes at a memory cost.

Algorithm 3.11 Forward-Filtering Backward Smoothing (FFBSm)

Where (i) *appears, the operation is performed for all* $i \in [\![1,N]\!]$.

Input: weighted particle sample $\left(X_t^{(1:N)}, \omega_t^{(1:N)}\right)$ approximating the filtering distribution for $t = 1,\dots,n$ from a particle filter.

1: Set $\upsilon_n^{(i)} = \omega_n^{(i)}$.
2: **for** $t = n-1,\dots,1$ **do**
3: Set

$$D_t^{(j)} = \sum_{k=1}^{N} \omega_t^{(k)} f\left(X_{t+1}^{(j)} | X_t^{(k)}\right).$$

4: Update

$$\upsilon_t^{(i)} = \omega_t^{(i)} \sum_{j=1}^{N} \frac{\upsilon_{t+1}^{(j)}}{D_t^{(j)}} f\left(X_{t+1}^{(j)} | X_t^{(i)}\right).$$

5: **end for**

Output: weighted particle sample $\left(X_t^{(1:N)}, \upsilon_t^{(1:N)}\right)$ approximating the marginal smoothing distribution for $t = 1,\dots,n$.

3.4.3 Forward-filtering backward simulation (FFBSi)

An alternative to FFBSm is known as *forward-filtering backward simulation* (FFBSi) (Godsill et al. [2004]) and is more straightforward to implement. It consists of sampling backwards from the sequence of filtering distribution approximations in order to obtain a realisation from the joint smoothing distribution.

At step t, given weighted particle samples $\left(X_t^{(1:N)}, \omega_t^{(1:N)}\right)$ approximating the filtering distributions for $t = 1,\dots,n$ output by any particle filter, define the Markov transition matrix over the set of particle indices:

$$\Lambda_t^{(i,j)} = \frac{\omega_t^{(j)} f\left(X_{t+1}^{(i)} | X_t^{(j)}\right)}{\sum_{k=1}^{N} \omega_t^{(k)} f\left(X_{t+1}^{(i)} | X_t^{(k)}\right)}, \quad (i,j) \in [\![1,N]\!]^2.$$

Starting at step n, sample an index $J_n \sim \mathbb{P}(\omega_n^{(1:N)})$. Then, progressing backwards, i.e. for steps $t = n-1,\dots,1$, sample the next index $J_n \sim \mathbb{P}(\Lambda_t^{(J_{t+1},1:N)})$ from the J_{t+1}-th row of the matrix $\Lambda_t := (\Lambda_t^{(1:N,1:N)})$. The FFBSi algorithm is summarised in Algorithm 3.12. By sampling N indices $J_t^{(1:N)}$ at each step t instead of a single one, it is possible to construct an unbiased estimator of FFBSm, which can essentially be viewed as a Rao-Blackwellised version of FFBSi.

Algorithm 3.12 Forward-Filtering Backward Simulation (FFBSi)

Where (i,j) *appears, the operation is performed for all* $(i,j) \in [\![1,N]\!]^2$.

Input: weighted particle sample $\left(X_t^{(1:N)}, \omega_t^{(1:N)}\right)$ approximating the filtering distribution for $t = 1,\dots,n$ from a particle filter.

1: Sample $J_n \sim \mathbb{P}\left(\omega_n^{(1:N)}\right)$.

2: **for** $t = n-1,\dots,1$ **do**.

3: Set for $j = 1,\dots,N$

$$D_t^{(j)} = \sum_{k=1}^{N} \omega_t^{(k)} f\left(X_{t+1}^{(j)} | X_t^{(k)}\right).$$

4: Compute the matrix

$$\Lambda_t^{(i,j)} = \frac{\omega_t^{(j)}}{D_t^{(j)}} f\left(X_{t+1}^{(i)} | X_t^{(j)}\right).$$

5: Sample

$$J_t \sim \mathbb{P}\left(\Lambda_t^{(J_{t+1},1:N)}\right).$$

6: **end for**

Output: simulated sample $\left(X_t^{(J_t)}\right)$ for $t = 1,\dots,n$ from the joint smoothing distribution.

The need to recompute the matrix Λ_t for all steps t makes each iteration of the FFBSi algorithm of $\mathscr{O}(N^2)$ complexity. Under mild assumptions, it is however possible to reduce this complexity to $\mathscr{O}(N)$ by employing an Accept-reject scheme, described next.

Given weighted particle samples $(X_t^{(1:N)}, \omega_t^{(1:N)})$ approximating the filtering distributions for $t = 1,\dots,n$ output by a particle filter, assume that the Markov transition density for the state f is uniformly bounded, i.e.

Assumption 3.1. There exists an upper bound $\varepsilon > 0$ such that

$$f(x|x') \leq \varepsilon$$

for all $(x,x') \in \mathscr{X} \times \mathscr{X}$.

Given Assumption 3.1, it is possible to sample the indices J_t for $t = n,\dots,1$ as described in Algorithm 3.13. It is straightforward to extend the Accept-reject FFBSi algorithm so that N indices $J_t^{(1:N)}$ are sampled at each step t, effectively constructing an $\mathscr{O}(N)$ approximation to FFBSm.

Algorithm 3.13 Accept-reject FFBSi

Input: weighted particle sample $\left(X_t^{(1:N)}, \omega_t^{(1:N)}\right)$ approximating the filtering distribution for $t = 1, \ldots, n$ from a particle filter.

Sample $J_n \sim \mathbb{P}\left(\omega_n^{(1:N)}\right)$.
for $t = n-1, \ldots, 1$ **do**
    REJECT := TRUE
    while REJECT **do**
        Sample $J_t \sim \mathbb{P}\left(\omega_t^{(1:N)}\right)$.
        Draw $u \sim \mathscr{U}_{[0,1]}$.
        if $u \leq f\left(X_{t+1}^{J_t} | X_t^{J_t}\right) / \varepsilon$ **then**
            REJECT := FALSE
        else
            REJECT := TRUE
        end if
    end while
end for

Output: simulated sample $\left(X_t^{(J_t)}\right)$ for $t = 1, \ldots, n$ from the joint smoothing distribution.

3.5 Forward smoothing for additive functionals

We now turn our attention to the general problem of evaluating *smoothing expectations*. Recall their definition in Equation 3.4 for function $S_t : \mathscr{X}^t \to \mathbb{R}$,

$$\mathscr{S}_t := \mathbb{E}\left[S_t\left(X_{1:t}\right) | y_{1:t}\right] = \int_{\mathscr{X}^t} S_t\left(x_{1:t}\right) \pi_t(x_{1:t}) \mathrm{d}x_{1:t}. \tag{3.16}$$

Let us consider the situation in which the function S_t is an *additive functional*, i.e. there exists a sequence of *sufficient statistics* $s_k : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ for $k = 1, \ldots, t$ such that

$$S_t(x_{1:t}) = \sum_{k=1}^{t} s_k(x_{k-1}, x_k),$$

where $s_0(x_0, x_1) := s_1(x_1)$. Note that the sufficient statistics can be defined to depend on the current data point y_k. From this, a recursion is straightforward to establish:

$$S_t(x_{1:t}) = S_{t-1}(x_{1:t-1}) + s_t(x_{t-1}, x_t), \tag{3.17}$$

starting from $S_1(x_1) = s_1(x_1)$. As seen in Section 3.2, the smoothing expectation can only be computed exactly for a linear Gaussian state space model or when the state space is finite.

Otherwise, a particle approximation $(X_{1:t}^{(1:N)}, \omega_t^{(1:N)})$ of Equation 3.16 given by

$$\hat{\mathscr{S}}_t := \sum_{i=1}^{N} \omega_t^{(i)} \hat{S}_t\left(X_{1:t}^{(i)}\right)$$

can be obtained using any of the particle smoothing algorithms above. For example, note that Equation 3.16 can be rewritten as

$$\mathscr{S}_t = \sum_{k=1}^{t} \int_{\mathscr{X}^t} s_k\left(x_{k-1}, x_k\right) \pi_t(x_{1:t}) \mathrm{d}x_{1:t} = \sum_{k=1}^{t} \int_{\mathscr{X}^2} s_k\left(x_{k-1}, x_k\right) \pi_t(x_{k-1:k}) \mathrm{d}x_{k-1:k}.$$

Taking the FFBSm algorithm, a particle approximation of $\pi_t(x_{k-1:k})$ can be obtained in order to estimate $\mathscr{S}_t$ and is given by

$$\hat{\pi}_t(x_{k-1:k}) := \sum_{i=1}^{N} \sum_{j=1}^{N} \upsilon_k^{(j)} \delta_{X_{k-1}^{(i)}, X_k^{(j)}}(x_{k-1:k}),$$

where the marginal smoothing weights $\upsilon_k^{(1:N)}$ are defined in Equation 3.14.

Currently, most particle smoothing algorithms described so far require both a forward and a backward pass, but when S_t is an additive functional, smoothing can be done in a single forward pass. This is particularly beneficial for large datasets, or when multiple passes through the data are required, e.g. when estimating the hyperparameters via offline EM (see Section 6.5.6 for an example).

Before describing forward smoothing, we give a brief overview of the *path space* approach to estimating smoothing expectations of additive functionals without a backward pass, which is intimately related to the poor man's smoother from Section 3.4.1. Thanks to the additive form of S_t, its estimate can be updated at each step t of the algorithm without needing to store the ancestral paths of the particles, by taking advantage of the recursion in Equation 3.17 as follows. Given particle approximations $(X_t^{(1:N)}, \omega_t^{(1:N)})$ and $(X_{t-1}^{(1:N)}, \omega_{t-1}^{(1:N)})$ for the current and previous steps, and denoting by $\hat{S}_{t-1}(X_{1:t-1}^{(i)})$ the i-th particle approximating $S_{t-1}(x_{1:t-1})$, we have

$$\hat{S}_t\left(X_{1:t}^{(i)}\right) = \hat{S}_{t-1}\left(X_{1:t-1}^{(i)}\right) + s_t\left(X_{t-1}^{(i)}, X_t^{(i)}\right),$$

starting from $\hat{S}_1(X_1^{(i)}) = s_1(X_1^{(i)})$ for $i = 1, \ldots, N$. Despite being relatively simple, of $\mathscr{O}(N)$ complexity and the only smoothing algorithm so far that doesn't require a backward pass through the data, the path space approach is often not suitable, since it suffers from path degeneracy. This leads to poor estimates of $\mathscr{S}_t$, which have an asymptotic variance that increases at least quadratically with t (Poyiadjis et al. [2011]).

3.5.1 Fixed-lag smoothing

One way of reducing the asymptotic variance of the estimates while still avoiding a backward pass is to employ the *fixed-lag smoother* introduced by Kitagawa and Sato [2001] and discussed in Olsson et al. [2008, 2011]. Fixed-lag smoothing consists of taking advantage of the 'forgetting properties' of the state space model, i.e. for $\ell \geq 0$,

$$\pi_t(x_{1:t}) \approx \pi_{\min\{t+\ell,n\}}(x_{1:t}), \tag{3.18}$$

which essentially means that any observations arriving in the interval $t+\ell < k \leq n$ bring little additional information to $X_{1:t}$. As a result, this version of the smoother works by only updating the estimate of the smoothing expectation until step $t+\ell$. The issue then is to select a suitable value of ℓ. Too small and the approximation in Equation 3.18 will be poor, too large and path degeneracy will arise. Additionally, the storage cost for this approach is higher, since the ℓ ancestor particles of $X_t^{(i)}$ must be retained for all $i \in [\![1,N]\!]$.

3.5.2 Forward-only FFBSm

Forward smoothing was introduced in Del Moral et al. [2010] in the form of a forward-only version of FFBSm, as another way to both fight the issue of path degeneracy and avoid the addition of a backward pass, and without needing to specify an additional parameter ℓ. Define the following auxiliary function $T_t : \mathscr{X} \to \mathbb{R}$,

$$T_t(x_t) := \int_{\mathscr{X}^{t-1}} S_t(x_{1:t})\pi_{t-1}(x_{1:t-1}|x_t)\mathrm{d}x_{1:t-1},$$

where $\pi_{t-1}(x_{1:t-1}|x_t) := p(x_{1:t-1}|y_{1:t-1},x_t)$. It is straightforward to establish a recursion on T_t known as the *forward smoothing recursion* as follows:

$$\begin{aligned}
T_t(x_t) &= \int_{\mathscr{X}^{t-1}} \left[S_{t-1}(x_{1:t-1}) + s(x_{t-1},x_t)\right] \pi_{t-1}(x_{1:t-1}|x_t)\mathrm{d}x_{1:t-1} \\
&= \int_{\mathscr{X}} \pi_{t-1}(x_{t-1}|x_t) \underbrace{\int_{\mathscr{X}^{t-2}} S_{t-1}(x_{1:t-1})\pi_{t-2}(x_{1:t-2}|x_{t-2})\mathrm{d}x_{1:t-2}}_{T_{t-1}(x_{t-1})} \mathrm{d}x_{t-1} \\
&\quad + \int_{\mathscr{X}} s(x_{t-1},x_t)\pi_{t-1}(x_{t-1}|x_t)\mathrm{d}x_{t-1} \\
&= \int_{\mathscr{X}} \left[T_{t-1}(x_{t-1}) + s(x_{t-1},x_t)\right] \pi_{t-1}(x_{t-1}|x_t)\mathrm{d}x_{t-1},
\end{aligned} \tag{3.19}$$

and the smoothing expectation of interest is given in terms of the auxiliary function by

$$\mathscr{S}_t = \int_{\mathscr{X}} T_t(x_t)\pi_t(x_t)\mathrm{d}x_t. \tag{3.20}$$

Now what remains is to obtain an expression for the density $\pi_{t-1}(x_{t-1}|x_t)$:

$$\pi_{t-1}(x_{t-1}|x_t) = \frac{f(x_t|x_{t-1})\pi_{t-1}(x_{t-1})}{\int_{\mathscr{X}} f(x_t|x_{t-1})\pi_{t-1}(x_{t-1})\mathrm{d}x_{t-1}}. \tag{3.21}$$

None of these quantities can be computed exactly for a non-Gaussian, non-finite state space model, but particle approximations of them can be obtained by incorporating the expressions in Equations 3.19, 3.20 and 3.21 within any particle filter. The resulting algorithm is known as *forward smoothing SMC* (SMC-FS) and is summarised in Algorithm 3.14.

The SMC-FS algorithm does not require a backward pass and is capable of yielding estimates whose variances increase only linearly with t (Poyiadjis et al. [2011]), thus reducing path degeneracy. However, its computational complexity is still $\mathscr{O}(N^2)$ which can be prohibitive in some cases. Klaas et al. [2012] propose 'N-body' (Gray and Moore [2001]) approaches to reduce the complexity to $\mathscr{O}(N \log N)$ and more recently, Olsson et al. [2017] introduced an importance sampling-based algorithm described in Section 6.5.3 which is capable of reducing the complexity to linear.

Algorithm 3.14 Forward smoothing SMC (SMC-FS)

Where (i) *or* (j) *appears, the operation is performed for all* $i, j \in [\![1, N]\!]$.

At $t = 1$

1: Initialise the particle filter to obtain the weighted particle sample $\left(X_1^{(1:N)}, \omega_1^{(1:N)}\right)$.

2: Set $\hat{T}_1\left(X_1^{(i)}\right) := 0$.

3: **for** $t = 2, \ldots, n$ **do**

4: Use the particle filter to update the weighted particle sample, i.e.

$$\left(X_t^{(1:N)}, \omega_t^{(1:N)}\right) := \mathsf{PF}\left(X_{t-1}^{(1:N)}, \omega_{t-1}^{(1:N)}\right).$$

5: Set

$$\Psi_t^{(i,j)} := \frac{\omega_{t-1}^{(j)} f\left(X_t^{(i)} | X_{t-1}^{(j)}\right)}{\sum_{j=1}^N \omega_{t-1}^{(j)} f\left(X_t^{(i)} | X_{t-1}^{(j)}\right)}.$$

6: Update the auxiliary function estimate

$$\hat{T}_t\left(X_t^{(i)}\right) = \sum_{j=1}^N \Psi_t^{(i,j)} \left[\hat{T}_{t-1}\left(X_{t-1}^{(j)}\right) + s_t\left(X_{t-1}^{(j)}, X_t^{(i)}\right)\right].$$

7: Update the smoothing expectation estimate

$$\hat{\mathscr{S}}_t = \sum_{i=1}^N \omega_t^{(i)} \hat{T}_t\left(X_t^{(i)}\right).$$

8: **end for**

Output: smoothing expectation estimate $\hat{\mathscr{S}}_n$.

Chapter 4

Anytime Parallel Tempering

4.1 Chapter overview

Developing efficient and scalable Markov chain Monte Carlo (MCMC) algorithms is indispensable in Bayesian inference. To improve their performance, a possible solution is parallel tempering, which runs multiple interacting MCMC chains to more efficiently explore the state space. The multiple MCMC chains are advanced independently in what is known as local moves, and the performance enhancement steps are the exchange moves, where the chains pause to exchange their current sample amongst each other. To reduce the real time taken to perform the independent local moves, they may be performed simultaneously on multiple processors. Another problem is then encountered: depending on the MCMC implementation and the inference problem itself, the local moves can take a varying and random amount of time to complete, and there may also be computing infrastructure induced variations, such as competing jobs on the same processors, an issue one must contend with in a cloud computing setting, for example. Thus before the exchange moves can occur, all chains must complete the local move they are engaged in so as to avoid introducing a potentially substantial bias (Proposition 2.1). To solve this problem of randomly varying local move completion times when parallel tempering is implemented on a multi-processor computing resource, we adopt the Anytime Monte Carlo framework of Murray et al. [2016b]: we impose real-time deadlines on the parallelly computed local moves and perform exchanges at these deadlines without any processor idling. We show our methodology for exchanges at real-time deadlines does not introduce a bias and leads to significant performance enhancements over the naïve approach of idling until every processor's local moves complete.

The work done in this chapter is the subject of a journal paper Marie d'Avigneau et al. [2020] written in collaboration with co-authors Dr Sumeetpal Singh and Dr Lawrence Murray and submitted for publication.

4.2 Introduction

Consider a set of n observations $y = \{y_1, \ldots, y_n\} \in \mathscr{Y}$ following a probability model with underlying parameters $\theta \in \Theta$ and associated *likelihood* $f(y_1, \ldots, y_n \,|\, \theta)$ which we abbreviate to $f(y \,|\, \theta)$. In most cases, the posterior $\pi(\theta)$ of interest is intractable and must be approximated using computational tools such as the commonly used Metropolis-Hastings algorithm (Robert and Casella [2004b]) with random walk proposals. However, as models become more complex, the exploration of the posterior using such basic methods quickly becomes inefficient (Beskos et al. [2009]). Furthermore, the model itself can pose its own challenges such as the likelihood becoming increasingly costly or even impossible to evaluate (Tavaré et al. [1997]); the Lotka-Volterra predator-prey model of Section 4.5 is a concrete example.

Parallel tempering, initially proposed by Swendsen and Wang [1986] and further developed under the name Metropolis-coupled Markov chain Monte Carlo (MC)^3 by Geyer [1991], is a generic method for improving the efficiency of MCMC that can be very effective without significantly altering the original MCMC algorithm, for example by designing more efficient proposals. The parallel tempering algorithm runs multiple interacting MCMC chains to more efficiently explore the state space. The multiple MCMC chains are advanced independently, in what is known as the local moves, and the performance enhancement steps are the exchange moves, where the chains pause and attempt to swap their current sample amongst each other. Parallel tempering allows for steps of various sizes to be made when exploring the parameter space, which makes the algorithm effective, even when the distribution we wish to sample from has multiple modes. In order to reduce the real time taken to perform the independent local moves, they may be performed simultaneously on multiple processors, a feature we will focus on in this work.

Let the parallel tempering MCMC chain be $\left(X_k^{1:\Lambda}\right)_{k=1}^{\infty} = \left(X_k^1, \ldots, X_k^\Lambda\right)_{k=1}^{\infty}$ with initial state $\left(X_0^{1:\Lambda}\right)$ and target distribution

$$\pi(x^{1:\Lambda}) \propto \prod_{\lambda=1}^{\Lambda} \pi_\lambda(x^\lambda), \tag{4.1}$$

where the $\pi_\lambda(\,\cdot\,)$ are independent marginals corresponding to the target distribution of each of Λ chains, running in parallel at different temperatures indexed by λ. One of these chains, say $\lambda = \Lambda$, is the *cold* chain, and its target distribution $\pi_\Lambda = \pi$ is the posterior of interest. At each step n of parallel tempering (Geyer [2011]), one of two types of updates is used to advance the Markov chain $X_k^{1:\Lambda}$ to its next state:

1. Independent *local moves*: for example, a standard Gibbs or Metropolis-Hastings update, applied to each tempered chain X_k^λ in parallel.

2. Interacting *exchange moves*: propose to swap the states $x \sim \pi_\lambda$ and $x' \sim \pi_{\lambda'}$ of one or more pairs of adjacent chains. For each pair, accept a swap with probability

$$\min\left\{1, \frac{\pi_\lambda(x')\pi_{\lambda'}(x)}{\pi_\lambda(x)\pi_{\lambda'}(x')}\right\}, \tag{4.2}$$

   otherwise, the chains in the pair retain their current states.

With the cold chain providing the desired precision and the warmer chains more freedom of movement when exploring the parameter space, the combination of the two types of update allows all chains to mix much faster than any one of them would mix on its own. This provides a way to jump from mode to mode in far fewer steps than would be required under a standard non-tempered implementation using, say, the Metropolis-Hastings algorithm.

A particular advantage of parallel tempering is that it is possible to perform the independent local moves in parallel on multiple processors in order to reduce the real time taken to complete them. Unfortunately, this gives rise to the following problem: depending on the MCMC implementation and the inference problem itself, the local moves can take a *varying and random* amount of time to complete, which depends on the part of the state space it is exploring (see the Lotka-Volterra predator-prey model in Section 4.5 for a specific real example). Thus, before the exchange moves can occur, all chains *must* complete the local move they are engaged in to avoid introducing a potentially substantial bias (see Proposition 2.1). Additionally, the time taken to complete local moves may also reflect computing infrastructure induced variations, for example, due to variations in processor hardware, memory bandwidth, network traffic, I/O load, competing jobs on the same processors as well as potential unforeseen interruptions such as system failures, all of which affect the compute time of local moves. Local moves in parallel tempering algorithms can also have temperature-dependent completion times. This is the case of the ABC application in Section 4.4.

Firstly, to solve the problem of randomly distributed local move completion times when parallel tempering is implemented on a multi-processor computing resource, we adopt the Anytime Monte Carlo framework of Murray et al. [2016b]: we guarantee the simultaneous readiness of all chains by imposing real-time deadlines on the parallelly computed local moves, and perform exchange moves at these deadlines without any idling, i.e. without waiting for the slowest of them to complete their local moves. Idling has a financial cost, for example in a cloud computing setting, and can also significantly reduce the effective Monte Carlo sample size returned. We show that hard deadlines introduce a bias which we mitigate using the Anytime framework (see Proposition 4.1).

Secondly, we illustrate our gains through detailed numerical work. The first experiment

considered is a multi-modal Gamma mixture model where the biased and de-biased target distributions can be characterised for ease of comparison with the numerical results. We then apply our Anytime parallel tempering methodology in the context of ABC (Pritchard et al. [1999]; Tavaré et al. [1997]). In ABC, simulation is used instead of likelihood evaluations, which makes it particularly useful for Bayesian problems where the likelihood is unavailable or too costly to compute. In Lee [2012], a more efficient MCMC kernel for ABC (as measured by the effective sample size), called the *1-hit MCMC kernel*, was devised to significantly improve the probability that a good proposal in the direction of a higher posterior density is accepted, thus more closely mimicking exact likelihood evaluations. This new MCMC kernel was subsequently shown in Lee and Łatuszyński [2014] to also theoretically outperform competing ABC methods. The 1-hit kernel has a random execution time that depends on the part of the parameter space being explored, and is thus a good candidate for our Anytime parallel tempering method. In this paper, we show that we can improve the performance of the 1-hit MCMC kernel by introducing tempering and exchange moves, and embed the resulting parallel tempering algorithm within the Anytime framework to mitigate processor idling due to random local move completion times. Parallel tempering for ABC has been proposed by Baragatti et al. [2013], but hasn't been studied in the Anytime context as we do for random local move completion times, nor has the more efficient 1-hit MCMC kernel been employed. We perform a detailed numerical study of the Lotka-Volterra predator-prey model, which has an intractable likelihood and is a popular example used to contrast methods in the ABC literature (Fearnhead and Prangle [2012]; Prangle et al. [2017]; Toni et al. [2009]). The time taken to simulate from the Lotka-Volterra model is random and parameter value dependent; this randomness is in addition to that induced by the 1-hit kernel.

The Anytime parallel tempering framework can be applied in several contexts. For example, another candidate for our framework is RJ-MCMC by Green [1995], which is a variable-dimension Bayesian model inference algorithm. An instance of RJ-MCMC within a parallel tempering algorithm is given in Jasra et al. [2007b], where multiple chains are simultaneously updating states of variable dimensions (depending on the model currently considered on each chain), and the real completion time of local moves depends on the dimension of the state space under the current model. Additionally, in the fixed dimension parallel tempering setting, if the local moves use any of the following MCMC kernels, then they have a parameter dependent completion time and thus could benefit from an Anytime formulation: the no-U-turn sampler (NUTS) (Hoffman and Gelman [2014]) and elliptical slice sampling (Murray et al. [2010]; Nishihara et al. [2014]). Even if the local moves do not take a variable random time to complete by design (Calderhead and Girolami [2009]; Friel and Pettitt [2008]), computer infrastructure induced variations, such as memory bandwidth,

competing jobs, etc. can still affect the real completion time of local moves in a parallel tempering algorithm, such as in Rodinger et al. [2006]. In the statistical mechanics literature, there are also parallel tempering-based simulation problems where the local move completion time is temperature- and parameter-dependent as well as random, e.g. see Earl and Deem [2004]; Hritz and Oostenbrink [2007]; Karimi et al. [2011]; Wang and Jordan [2003], and thus could benefit from our Anytime formulation. Finally, the Anytime framework has so far not been tested beyond the SMC^2 example of Murray et al. [2016b], but it can be applied to any parallelisable population-based MCMC algorithm which includes local moves and interacting moves where all processors must communicate, such as sequential Monte Carlo (SMC) samplers (Del Moral et al. [2006]), considered in Chapter 5, or parallelised generalised elliptical slice sampling (Nishihara et al. [2014]).

This chapter is structured as follows. Section 4.3 develops our Anytime Parallel Tempering Monte Carlo (APTMC) algorithm and then Section 4.4 extends our framework further for the 1-hit MCMC kernel of Lee [2012] for approximate Bayesian computation. Experiments are run in Section 4.5 and include a carefully constructed synthetic example to demonstrate the workings and salient features of Anytime parallel tempering. Section 4.5 also presents an application of Anytime parallel tempering to the problem of estimating the parameters of a stochastic Lotka-Volterra predator-prey model. Finally, Section 4.6 provides a summary and some concluding remarks.

4.3 Anytime Parallel Tempering Monte Carlo

4.3.1 Overview

Consider the problem in which we wish to sample from target distribution $\pi(x)$. In a parallel tempering framework, construct Λ Markov chains where each individual chain λ targets the tempered distribution

$$\pi_\lambda(x) \propto \pi(x)^{\frac{\lambda}{\Lambda}}$$

and is associated with kernel $\kappa_\lambda(x_k \,|\, x_{k-1})$ and hold time distribution $\tau_\lambda(h_k \,|\, x_k)$. In this setting, the hold time distribution is not assumed to be homogeneous across all chains, and may be temperature-dependent. As per Corollary 2.1, each chain also has its associated anytime distribution denoted $\alpha_\lambda(x_k)$. Assume that all Λ chains are running concurrently on Λ processors. We aim to interrupt the computations on a real-time schedule of times $t_1, t_2, t_3, \ldots$ to perform exchange moves between adjacent pairs of chains before resuming the local moves. To illustrate the challenge of his task, we discuss the case where $\Lambda = 2$. Let π_2 be the desired posterior and π_1 the 'warm' chain, with associated hold time distributions τ_1

and τ_2, respectively. When the two chains are interrupted at some time t, assume that the current sample on chain 1 is X_j^1 and that of chain 2 is X_k^2. It follows from Corollary 2.1 that

$$X_j^1 \sim \alpha_1(x) = \frac{\mathbb{E}[H_1 \,|\, x]}{\mathbb{E}[H_1]}\pi_1(x) \neq \pi_1(x),$$

and similarly for X_k^2. Exchanging the samples using the acceptance probability in Equation 4.2 is incorrect. Indeed, exchanging using the current samples X_j^1 and X_k^2, if accepted, will result in the sample sets $\{X_1^1, X_2^1, \ldots\}$ and $\{X_1^2, X_2^2, \ldots\}$ being corrupted with samples which arise from their respective length-biased, anytime distributions α_1 and α_2, as opposed to being exclusively from π_1 and π_2. Furthermore, the expressions for α_1 and α_2 will most often be unavailable, since their respective hold time distributions τ_1 and τ_2 are not explicitly known but merely implied by the algorithm used to simulate the two chains. Finally, we could wait for chains 1 and 2 complete their computation of X_{j+1}^1 and X_{k+1}^2 respectively, and then accept/reject the exchange $\left(X_{j+1}^1, X_{k+1}^2\right) \to \left(X_{k+1}^2, X_{j+1}^1\right)$ according to Equation 4.2. This approach won't introduce a bias but can result in one processor idling while the slower computation finishes. We show this can result in significant idling in numerical examples.

In the next section, we describe how to correctly implement exchange moves within the Anytime framework.

4.3.2 Anytime exchange moves

Here, we adapt the multi-chain construction devised to remove the bias present when sampling from Λ Markov chains, where each chain λ targets the distribution π_λ for $\lambda = 1, \ldots, \Lambda$. Associated with each chain is MCMC kernel $\kappa_\lambda(x_k^\lambda \,|\, x_{k-1}^\lambda)$ and hold time distribution $\tau_\lambda(h \,|\, x)$.

Proposition 4.1. Let $\pi_\lambda(x)$, $\lambda = 1 \ldots, \Lambda$ be the stationary distributions of Λ Markov chains with associated MCMC kernels $\kappa_\lambda(x_k^\lambda \,|\, x_{k-1}^\lambda)$ and hold time distributions $\tau_\lambda(h \,|\, x)$. Assume the chains are updated sequentially and let j be the index of the currently working chain. The joint anytime distribution is the following generalisation of Proposition 2.1:

$$A(x^{1:\Lambda}, l, j) = \frac{1}{\Lambda}\frac{\mathbb{E}[H \,|\, j]}{\mathbb{E}[H]}\alpha_j(x^j, l) \prod_{\lambda=1, \lambda\neq j}^{\Lambda} \pi_\lambda(x^\lambda).$$

The proof of Proposition 4.1 is a straightforward adaptation of the proof of Proposition 5 in Murray et al. [2016b]. Conditioning on x^j, j and l we obtain

$$A(x^{1:\Lambda\setminus j} \,|\, x^j, l, j) = \prod_{\lambda=1, \lambda\neq j}^{\Lambda} \pi_\lambda(x^\lambda). \tag{4.3}$$

Therefore, if exchange moves on the conditional $A(x^{1:\Lambda\setminus j} \,|\, x^j, l, j)$ are performed by 'eliminating' the j-th chain to obtain the expression in Equation 4.3, they are being performed involving only chains distributed according to their respective targets π_λ and thus the bias is eliminated.

4.3.3 Implementation

On a single processor, the algorithm may proceed as in Algorithm 4.1, where in Step 3 the Λ chains are simulated one at a time in a serial schedule. Figure 4.1 provides an illustration of how the algorithm works.

Algorithm 4.1 Anytime Parallel Tempering Monte Carlo on one processor

1: Initialise real-time Markov jump process $\left(X^{1:\Lambda}, L, J\right)(0) = \left(x_0^{1:\Lambda}, 0, 1\right)$.
2: **for** $i = 1, 2, \ldots$ **do**
3: Simulate real-time Markov jump process $\left(X^{1:\Lambda}, L, J\right)(t)$ until real time t_i.
4: Perform exchange steps on the conditional in Equation 4.3.
5: **end for**

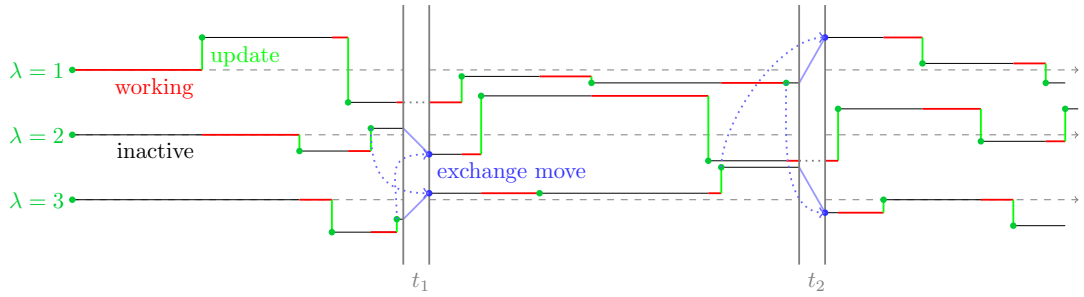


Fig. 4.1 Illustration of the progression of three chains in the Anytime Parallel Tempering Monte Carlo (APTMC) algorithm on a single processor. The *green* (local move) and *blue* (exchange move) dots represent samples from the posterior being recorded as their respective local and exchange moves are completed. When exchange moves occur at t_1, chain $\lambda = 1$ is currently moving cannot participate in exchange moves without introducing a bias. Therefore it is ignored, and the exchange moves are performed on the remaining (inactive) chains. Similarly, a time t_2 chain $\lambda = 2$ is excluded from the exchange. The widths of intervals t_1 and t_2 are for illustrating the exchange procedure only.

When multiple processors are available, the Λ chains may be run in parallel. However, running a single chain on each processor means that when the real-time deadline occurs, all chains will be distributed according to their respective anytime distributions α_λ, and thus be biased as exchange moves occur. Therefore, all processors must contain at least two chains.

The implementation is defined as described in Algorithm 4.2. Note that the multiple chain construction eliminates the intractable densities in the acceptance ratio for the exchange step when τ differs between processors, since exchange moves are performed between chains that are not currently working (i.e. on the density in Equation 4.3 for a single processor and that in Equation 4.4 for multiple processors), so the hold time distribution does not factor in.

Algorithm 4.2 Anytime Parallel Tempering Monte Carlo on multiple processors

1: On worker w, initialise the real-time Markov jump process $\left(X_w^{1:C}, L_w, J_w\right)(0) = \left(x_0^{1:C}, 0, 1\right)$.
2: **for** $i = 1, 2, \ldots$ **do**
3: On each worker, simulate the real-time Markov jump process $\left(X_w^{1:C}, L_w, J_w\right)(t)$ until real time t_i.
4: Across all workers, perform exchange steps on the conditional

$$A(\boldsymbol{x}^{1:C\setminus \boldsymbol{j}} \,|\, \boldsymbol{x}^{\boldsymbol{j}}, \boldsymbol{l}, \boldsymbol{j}) = \prod_{w=1}^{W} \prod_{c=1, c\neq j_w}^{C} \pi_w(x_w^c), \tag{4.4}$$

where $\boldsymbol{x}^{1:C\setminus \boldsymbol{j}} = \left(x_1^{1:C\setminus j_1}, \ldots, x_W^{1:C\setminus j_W}\right)$, $\boldsymbol{x}^{\boldsymbol{j}} = \left(x_1^{j_1}, \ldots, x_W^{j_W}\right)$, $\boldsymbol{l} = l_{1:W}$ and $\boldsymbol{j} = j_{1:W}$.
5: **end for**

Depending on the problem at hand and computing resources available, there are various approaches to distributing the chains across workers. We distinguish three possible scenarios. The first is an ideal scenario, where the number of processors exceeds Λ and the communication overhead between workers is negligible. In this scenario, each worker implements $K = 2$ chains running at the same temperature. For example, with $W = \Lambda$ workers, worker $w = \lambda$ contains 2 chains targeting π_λ. The second scenario arises when the number of workers available is limited, but communication overhead is still negligible. In this case, the chains, sorted in increasing order of temperature, are divided evenly among workers. For example, with $W = \frac{\Lambda}{2}$ workers, worker w could contain two chains, one with target π_{2w-1} and one with target π_{2w}. The third scenario deals with non-negligible inter-processor communication overhead (which only affects the exchange moves). To account for this, exchange moves are divided into two types:

1. *Within-worker* exchange move: performed on each individual worker in parallel, between a pair of adjacent chains. No communication between workers is necessary in this case.

2. *Between-worker* exchange move: performed by selecting a pair of adjacent workers and exchanging between the warmest eligible chain from the first worker and coldest

from the second. Thus, an exchange move between two adjacent chains is effectively being performed, except this time communication between workers is required.

4.3.4 Tuning considerations

In this section we discuss the issue of tuning Anytime parallel tempering by drawing on various ideas from the literature. The main concerns are the selection of the number of chains and their temperatures, the tuning of the local moves for each chain, and the selection of appropriate deadlines for the exchange moves to occur. In our setting, the computational budget determines the number of chains Λ, and for such a fixed budget we aim to improve the sampling of the cold chain through the adoption of parallel tempering stages. The issue of determining the temperature of adjacent chains has been considered in Atchadé et al. [2011]; Kone and Kofke [2005]; Rathore et al. [2005] where it was shown that an exchange success rate of approximately 20-25% for adjacent chains is optimal, in an appropriate sense, and is demonstrated to confer the most benefit to sampling the coldest chain. However, the optimality curve (Atchadé et al. [2011]; Kone and Kofke [2005]) has a broad mode, and even 40% seems appropriate. To achieve this 25% acceptance rate of exchange moves, other than employing pilot runs, adaptive tuning is possible and Miasojedow et al. [2013] use a Robbins-Munro scheme to adjust the temperatures to target a 25% acceptance rate during runtime. The next issue is local proposals, and how large a change of state one should attempt (for the local accept/reject step). This subject has received ample attention in the literature following the seminal paper by Roberts et al. [2001], where a 25% local move acceptance rate is again optimal. The local proposal can be a Gaussian proposal whose mean and covariance matrix are again tuned online (Miasojedow et al. [2013]) via a Robbins-Munro scheme to achieve the 25% acceptance rate. Tuning of a Gaussian proposal for MCMC in general was popularised by the seminal paper of Haario et al. [2001].

Recalling from Section 2.4.1, when performing exchange moves, rather than selecting a single pair of adjacent chains from $\{(1,2),(2,3),\ldots,(\Lambda-1,\Lambda)\}$ for an exchange move, it is common to propose to swap multiple pairs of chains simultaneously, as the exchange move is relatively cheap. To avoid selecting the same chain twice, they are divided into odd $\{(1,2),(3,4),\ldots\}$ and even $\{(2,3),(4,5),\ldots\}$ pairs of indices in Lingenheil et al. [2009], and all odd or even pairs are selected for exchange with equal probability. It is however shown in Syed et al. [2019] that it is better to deterministically cycle between exchanging odd and even pairs.

Although thus far we have suggested tuning the number of chains and annealing schedule for APTMC as if one were tuning a standard parallel tempering algorithm, we highlight the following caveats. Selecting chains for exchange moves can be applied by omitting the

currently working chains and relabelling the indices of the remaining, inactive or *eligible* chains. However, note that by the nature of the Anytime exchange moves, the Anytime version of an optimised parallel tempering algorithm can be suboptimal, since one or more temperature(s) might be missing from exchange moves. Considering the example in Figure 4.1 and assuming the chains are all running at increasing temperatures, at t_2, chain 2 is working, so the exchange move is performed between chains 1 and 3. In a practical example, these chains would be further apart, which would lead to a lower exchange move acceptance rate. Selecting adjacent chains to target a slightly higher successful exchange rate, say 40%, would mitigate this issue; noting that even 40% is close to optimal (Atchadé et al. [2011]; Kone and Kofke [2005]). In our implementation, we only experienced a small drop in acceptance rate caused by attempting to swap two eligible chains that are not immediately adjacent, and this event becomes less likely as the number of chains increases.

Another important facet of tuning APTMC is the issue of determining the real-time schedule $t_1, t_2, \ldots$ of exchange moves. Let δ be the real-time interval or deadline between exchange moves, so that $t_i = i\delta$ for $i = 1, 2, \ldots$ We now present guidelines for calibrating δ. Let K be the number of chains, labelled $k = 1, \ldots, K$, on the slowest processor w_s (generally the one containing the cold chain), our experiments have shown that exchange moves should occur once every chain on this processor has completed at least one local move. The expected hold time of one set of local moves on processor w_s, denoted $\mathsf{H} := \sum_{k=1}^{K} \mathbb{E}[H_k]$, can be estimated by repeatedly measuring the time taken for one set of local moves to complete, and averaging across all measurements. Using a pilot run, an estimate $\widehat{\mathsf{H}}$ of this expected hold time can be obtained first, before setting $\delta = \widehat{\mathsf{H}}$ for the APTMC algorithm run. This δ value can also be calibrated in real time, denoted $\delta(t)$ where t is the real time. At $t = 0$, initialise $\delta(0) = \delta_0$ such that $\delta_0 > 0$ is an initial, user-defined guess. Similarly as before, record a hold time sample every time a set of local move occurs on processor w_s, then after every exchange move, recompute $\widehat{\mathsf{H}}$ and update $\delta(t) = \widehat{\mathsf{H}}$. An advantage of this second approach is that $\delta(t)$ then adapts to a potentially time-inhomogeneous hold time, due e.g. to competing jobs on the processors starting mid-algorithm and suddenly slowing down the computation time of local moves.

A scenario we encountered in our experiments was non-negligible communication overhead between workers when executing the exchange moves, and this overhead was comparable to the local move times which were themselves lengthy. To mitigate the communication overhead, as described in Section 4.3.3, exchange moves are divided into within- and between-workers. On a given worker with K chains, a set of worker-specific moves is performed before inter-worker exchanges. These were K local moves, one (set of) within-worker exchange moves, then K more local moves, before inter-worker communication occurs for between-

worker exchange moves. Given that within-worker exchanges are instant, this amounts in real time to performing $2K$ local moves on this worker before inter-worker communication occurs. Therefore, the real-time deadline in the Anytime version of the algorithm for this scenario is set to be $\delta = 2\mathsf{H}$ and can be determined as above. See Section 4.5.4.2 for an example.

Finally, Section 4.5.1.3 details other, empirical tools that help with tuning by assessing the efficiency of each chain. These include evaluating the sample autocorrelation function (acf), as well as the integrated autocorrelation time (IAT) and effective sample size (ESS).

4.4 Application to approximate Bayesian computation (ABC)

In this section we adapt the Anytime parallel tempering Monte Carlo framework to ABC.

4.4.1 The 1-hit MCMC kernel

The notion of ABC was developed by Tavaré et al. [1997] and Pritchard et al. [1999]. It can be seen as a likelihood-free way to perform Bayesian inference, using instead simulations from the model or system of interest, and comparing them to the observations available.

Let $y \in \mathscr{Y}$ be some data with underlying unknown parameters θ, where $p(\theta)$ denotes the prior for $\theta \in \Theta$. Suppose we are in the situation in which the likelihood $f(y \,|\, \theta)$ is either intractable or too computationally expensive, which means that MCMC cannot be performed as normal. Assuming that it is possible to sample from the density $f(\cdot \,|\, \theta)$ for all $\theta \in \Theta$, approximate the likelihood by introducing an artificial likelihood f^{ε} of the form

$$f^{\varepsilon}(y \,|\, \theta) = \mathrm{Vol}(\varepsilon)^{-1} \int_{B_{\varepsilon}(y)} f(x \,|\, \theta) \mathrm{d}x, \tag{4.5}$$

where $B_{\varepsilon}(y)$ denotes a metric ball centred at y of radius $\varepsilon > 0$ and $\mathrm{Vol}(\varepsilon)$ is its volume. The resulting approximate posterior is given by

$$p^{\varepsilon}(\theta \,|\, y) = \frac{p(\theta) f^{\varepsilon}(y \,|\, \theta)}{\int p(\vartheta) f^{\varepsilon}(y \,|\, \vartheta) \mathrm{d}\vartheta}.$$

The likelihood $f^{\varepsilon}(y \,|\, \theta)$ cannot be evaluated either, but an MCMC kernel can be constructed to obtain samples from the approximate posterior $\pi^{\varepsilon}(\theta, x)$ defined as

$$\pi^{\varepsilon}(\theta, x) = p^{\varepsilon}(\theta, x \,|\, y) \propto p(\theta) f(x|\theta) \mathbb{1}_{\varepsilon}(x) \mathrm{Vol}(\varepsilon)^{-1}, \tag{4.6}$$

where $\mathbb{1}_\varepsilon(x)$ is the indicator function for $x \in B_\varepsilon(y)$. This is referred to as *hitting* the ball $B_\varepsilon(y)$. In the MCMC kernel, one can propose $\theta' \sim q(\cdot \mid \theta)$ for some proposal density q, simulate the dataset $x \sim f(\cdot \mid \theta')$ and accept θ' as a sample from the posterior if $x \in B_\varepsilon(y)$.

The *1-hit MCMC kernel*, proposed by Lee [2012] and of which a typical iteration is described in Algorithm 4.3, introduces local moves in the form of a 'race': given current and proposed parameters θ and θ', respectively simulate corresponding datasets x and x' sequentially. The state associated with the first dataset to hit the ball $B_\varepsilon(y)$ 'wins' and is accepted as the next sample in the Markov chain. The proposal θ' is also accepted if both x and x' hit the ball at the same time. It is proven in Lee [2012] (Proposition 1) that the 1-hit kernel is a valid MCMC kernel targeting π^ε which satisfies detailed balance.

Algorithm 4.3 ABC: 1-hit MCMC kernel

Input: current state (θ_k, x_k).
for $i := 1, 2, \ldots$ **do**.
  Propose $\theta' \sim q(\cdot \mid \theta_k)$. ▷ *propose a local move*
  Compute preliminary acceptance probability ▷ *prior check*

$$\alpha(\theta_k, \theta') = \min\left\{1, \frac{p(\theta')q(\theta_k \mid \theta')}{p(\theta_k)q(\theta' \mid \theta_k)}\right\}.$$

  Sample $u \sim \text{Uniform}(0,1)$.
  if $u < \alpha(\theta_k, \theta')$ **then**
    RACE := TRUE
  else
    RACE := FALSE
    Retain $(\theta_{k+1}, x_{k+1}) := (\theta_k, x_k)$. ▷ *reject θ' as it is unlikely to win race*
  end if
  while RACE **do**
    Simulate $x \sim f(\cdot \mid \theta_n)$ and $x' \sim f(\cdot \mid \theta')$.
    if $x \in B_\varepsilon(y)$ **or** $x' \in B_\varepsilon(y)$ **then** ▷ *stop the race once either x or x' hits the ball.*
      RACE := FALSE
    end if
  end while
  if $x' \in B_\varepsilon(y)$ **then** ▷ *accept or reject move*
    Set $(\theta_{k+1}, x_{k+1}) := (\theta', x')$.
  else
    Retain $(\theta_{k+1}, x_{k+1}) := (\theta_k, x)$.
  end if
  $k := k + 1$
end for
Output: updated state (θ_{k+1}, x_{k+1}).

4.4.2 ABC Anytime Parallel Tempering Monte Carlo (ABC-APTMC)

Including the 1-hit kernel in the local moves of a parallel tempering algorithm is straightforward. Exchange moves must, however, be adapted to this new likelihood-free setting. Additionally, the race that occurs takes a random time to complete, thus providing good motivation for the use of Anytime Monte Carlo.

4.4.2.1 Exchange moves

Let (θ, x) and (θ', x') be the states of two chains targeting π^{ε} and $\pi^{\varepsilon'}$, respectively, where $\varepsilon' > \varepsilon$. Here, this is equivalent to saying θ' is the state of the 'warmer' chain. We already know that x' falls within ε' of the observations y, i.e. $x' \in B_{\varepsilon'}(y)$. Similarly, we also know that $x \in B_{\varepsilon}(y)$, and that $x \in B_{\varepsilon'}(y)$. If x' also falls within ε of y, then swap the states, otherwise do not swap. The odds ratio is

$$\begin{aligned}\frac{\pi^{\varepsilon'}(\theta,x)\pi^{\varepsilon}(\theta',x')}{\pi^{\varepsilon}(\theta,x)\pi^{\varepsilon'}(\theta',x')} &= \frac{p(\theta)f(x\,|\,\theta)\mathrm{Vol}(\varepsilon')p(\theta')f(x'\,|\,\theta')\mathbb{1}_{\varepsilon}(x')\mathrm{Vol}(\varepsilon)}{p(\theta)f(x\,|\,\theta)\mathrm{Vol}(\varepsilon)p(\theta')f(x'\,|\,\theta')\mathrm{Vol}(\varepsilon')} \\ &= \mathbb{1}_{\varepsilon}(x'),\end{aligned}$$

so the probability of the swap being accepted is the probability of x' also hitting the ball of radius ε centred at y. This type of exchange move is summarised in Algorithm 4.4.

4.4.2.2 Implementation

The full implementation of the ABC Anytime Parallel Tempering Monte Carlo (`ABC-APTMC`) algorithm on a single processor is described in Algorithm 4.5. The multi-processor algorithm can similarly be modified to reflect these new exchange moves.

Algorithm 4.4 ABC: exchange move between two chains

Input: $\xi_k = ((\theta,x),(\theta',x'))$ where $\theta \sim \pi$, $x \sim f(\cdot\,|\,\theta)$ and $\theta' \sim \pi'$, $x' \sim f(\cdot\,|\,\theta')$.
▷ *both* (θ,x) *and* (θ',x') *are outputs from Algorithm 4.3 for different* $\varepsilon' > \varepsilon$

1: **if** $x' \in B_{\varepsilon}(y)$ **then** ▷ *accept or reject swap depending on whether* x' *also hits the ball of radius* ε
2: Set $\xi_{k+1} := ((\theta',x'),(\theta,x))$.
3: **else**
4: Retain $\xi_{k+1} := \xi_k$.
5: **end if**
6: $k := k+1$

Output: updated state ξ_{k+1}.

Algorithm 4.5 ABC: Anytime Parallel Tempering Monte Carlo Algorithm

1: Initialise the real-time Markov jump process $(\theta^{1:\Lambda}, L, J) = (\theta_0^{1:\Lambda}, 0, 1)$.
2: Set $k := 0$.
3: **for** $i := 1, 2, \ldots$ **do**
SIMULATE THE REAL-TIME MARKOV JUMP PROCESS $(\theta, L, J)(t)$ UNTIL REAL TIME t_i
4: Perform local moves on $\left(\theta_k^j, x_k^j\right)$ according to Algorithm 4.3.
5: $j := j + 1$
6: **if** $j > \Lambda$ **then**
7: $j := 1$
8: **end if**
PERFORM EXCHANGE STEPS ON THE CONDITIONAL:

$$A(\theta^{1:\Lambda} \mid \theta^j, l, j) = \prod_{\lambda=1, \lambda \neq j}^{\Lambda} \pi_\lambda(\theta^\lambda).$$

9: Perform exchange moves on $\xi_k = \left((\theta_k^\lambda, x_k^\lambda), (\theta_k^{\lambda'}, x_k^{\lambda'})\right)$ according to Algorithm 4.4.
10: **end for**

4.5 Numerical experiments

In this section, we first illustrate the workings of the algorithms presented in Section 4.3.3 on a simple model, in which real-time behaviour is simulated using virtual time and an artificial hold distribution. The model is also employed to demonstrate the gain in efficiency provided by the inclusion of exchange moves. Then, the ABC version of the algorithms, as presented in Section 4.4, is applied to three case studies. The first case is a simple Gaussian model and serves to verify the workings of the ABC algorithm, including bias correction. The second case considers the problem of estimating the parameters of a moving average problem, and serves to illustrate the performance improvements brought by the addition of ABC exchange moves to the 1-hit ABC kernel on a single processor. The third case is more advanced and considers the problem of estimating the parameters of a stochastic Lotka-Volterra predator-prey model − in which the likelihood is unavailable − and serves to evaluate the performance of the Anytime parallel tempering version of the ABC-MCMC algorithm, as opposed to the standard versions (with and without exchange moves) on both a single and multiple processors. The exchange moves are set up so that multiple pairs could be swapped at each iteration. All experiments in this paper were run on MATLAB and the code is available at https://github.com/alixma/ABCAPTMC.git.

4.5.1 Toy example: Gamma mixture model

In this example we attempt to sample from an equal mixture of two Gamma distributions using the Anytime Parallel Tempering Monte Carlo (`APTMC`) algorithm. Define the target $\pi(\mathrm{d}x)$ and an 'artificial' hold time $\tau(\mathrm{d}h\,|\,x)$ distribution as follows:

$$\begin{aligned} X &\sim \phi\,\mathrm{Gamma}(k_1,\theta_1) + (1-\phi)\,\mathrm{Gamma}(k_2,\theta_2), \\ H\,|\,x &\sim \psi\,\mathrm{Gamma}\left(\frac{x^p}{\theta_1},\theta_1\right) + (1-\psi)\,\mathrm{Gamma}\left(\frac{x^p}{\theta_2},\theta_2\right), \end{aligned}$$

with mixture coefficients $\phi = \frac{1}{2}$ and ψ, where $\mathrm{Gamma}(\cdot,\cdot)$ denotes the probability density function of a Gamma distribution, with shape and scale parameters (k_1,θ_1) and (k_2,θ_2) for each component, respectively, and with polynomial degree p, assuming it remains constant for both components of the mixture.

In the vast majority of experiments, the explicit form of the hold time distribution τ it not known, but observed in the form of the time taken by the algorithm to simulate X. For this example, so as to avoid external factors such as competing jobs affecting the hold time, we assume an explicit form for τ is known and simulate virtual hold times. This consists of simulating a hold time $h \sim \tau(\cdot\,|\,x)$ and advancing the algorithm forward for h units of virtual time without updating the chains, effectively 'pausing' the algorithm. These virtual hold times are introduced so that what in a real-time example would be the effects of constant $(p=0)$, linear $(p=1)$, quadratic $(p=2)$ and cubic $(p=3)$ computational complexities can be studied. Another advantage is that the anytime distribution α_Λ of the cold chain can be computed analytically and is the following mixture of two Gamma distributions

$$\begin{aligned} \alpha_\Lambda(x) = {} & \varphi(p,k_{1:2},\theta_{1:2})\,\mathrm{Gamma}\,(k_1+p,\theta_1) \\ & + [1-\varphi(p,k_{1:2},\theta_{1:2})]\,\mathrm{Gamma}\,(k_2+p,\theta_2)\,, \end{aligned} \tag{4.7}$$

where

$$\varphi(p,k_{1:2},\theta_{1:2}) = \left(1+\frac{\Gamma(k_1)\Gamma(p+k_2)\theta_2^p}{\Gamma(k_2)\Gamma(p+k_1)\theta_1^p}\right)^{-1}.$$

We refer the reader to Appendix 4.A for the proof of Equation 4.7. In the anytime distribution, one of the components of the Gamma distribution will have an associated mixture coefficient $\varphi(p,k_{1:2},\theta_{1:2})$ or $1-\varphi(p,k_{1:2},\theta_{1:2})$ which increases with p while the coefficient of the other component decreases proportionally. Note that for constant $(p=0)$ computational complexity, the anytime distribution is equal to the target distribution π.

4.5.1.1 Implementation

On a single processor, the Anytime Parallel Tempering Monte Carlo (`APTMC-1`) algorithm is implemented as follows: simulate $\Lambda = 8$ Markov chains, each targeting the distribution $\pi_\lambda(x) = \pi(x)^{\frac{\lambda}{\Lambda}}$. To construct a Markov chain $(X_k^\lambda)_{k=0}^\infty$ with target distribution

$$\pi_\lambda(x) \propto \left[\frac{1}{2}\text{Gamma}(k_1,\theta_1) + \frac{1}{2}\text{Gamma}(k_2,\theta_2)\right]^{\frac{\lambda}{\Lambda}}$$

for $\lambda = 1,\ldots,\Lambda$, use a *random walk Metropolis* update, i.e. symmetric Gaussian proposal distribution $\mathcal{N}(x_k^\lambda,\sigma^2)$ with mean x_k^λ and standard deviation $\sigma = 0.5$. Set $(k_1,k_2) = (3,20)$, $(\theta_1,\theta_2) = (0.15,0.25)$ and use $p \in \{0,1,2,3\}$. The single processor algorithm is run for $T = 10^8$ units of virtual time with exchange moves alternating between occurring on all even $(1,2),(3,4),(5,6)$ and all odd $(2,3),(4,5),(6,7)$ pairs of inactive chains every $\delta = 5$ units of virtual time. When the algorithm is running, a sample is recorded every time a local or exchange move occurs.

On multiple processors, the Anytime Parallel Tempering Monte Carlo (`APTMC-W`) algorithm is implemented similarly. A number of $W = \Lambda = 8$ processors is used, where each worker $w = \lambda$ contains $K = 2$ chains, all targeting the same π_λ for $\lambda = 1,\ldots,\Lambda$. The multiple processor algorithm is run for $T = 10^7$ units of virtual time, with exchange moves alternating between occurring on all odd $(1,2),(3,4),(5,6),(7,8)$ and all even $(2,3),(4,5),(6,7)$ pairs of workers every $\delta = 5$ units of virtual time. On each worker, the chain which was not working when calculations were interrupted is the one included in the exchange moves.

4.5.1.2 Verification of bias correction

To check that the single and multiple processor algorithms are successfully correcting for bias, they are also run *uncorrected*, i.e. not excluding the currently working chain. This means that several exchange moves are performed on samples distributed according to α instead of π, thus causing the algorithm to yield biased results. Since the bias is introduced by the exchange moves (when they are performed on α), we attempt to create a 'worst case scenario', i.e. maximise the amount of bias present when the single processor algorithm is uncorrected. The algorithm is further adjusted such that local moves are not performed on the cold chain and it is instead solely made up of samples resulting from exchange moves with the warmer chains. The fact that exchange moves occur every $\delta = 5$ units of virtual time also means that a high proportion of the samples in a warmer chain come from exchange moves. The multi-processor `APTMC-W` algorithm is not run in a 'worst case scenario', so local moves on the cold chain of the multi-processor algorithm are therefore allowed. This means

that the bias caused by failing to correct when performing exchange moves across workers is still apparent, if less strongly.

Figure 4.2 shows kernel density estimates of the post burn-in cold chains resulting from runs of the `APTMC-1` and `APTMC-W` algorithms, uncorrected and corrected for bias. As expected, when the hold time does not depend on x, which corresponds to the case where $p = 0$, no bias is returned. On the other hand, the cold chains for the single-processor algorithm with computational complexity $p \in \{1, 2, 3\}$ have been corrupted by biased samples and converged to a shifted distribution which puts more weight on the second Gamma mixture component, instead of an equal weight. Additionally, the bias becomes stronger as computational complexity p increases. A similar observation can be made for the cold chains from the multi-processor experiment − which display a milder bias due to local moves occurring on the cold chain. The green dashed densities indicate that when the algorithms are corrected, i.e. when the currently working chain is not included in exchange moves, it successfully eliminates the bias for all $p \in \{1, 2, 3\}$ to return the correct posterior π − despite this being the 'worst-case scenario' in the case of the `APTMC-1` algorithm. Note that the uncorrected density estimates do not exactly correspond to the anytime distributions. This has nothing to do with burn-in, but with the proportion of biased samples (from exchange moves) present in the chain.

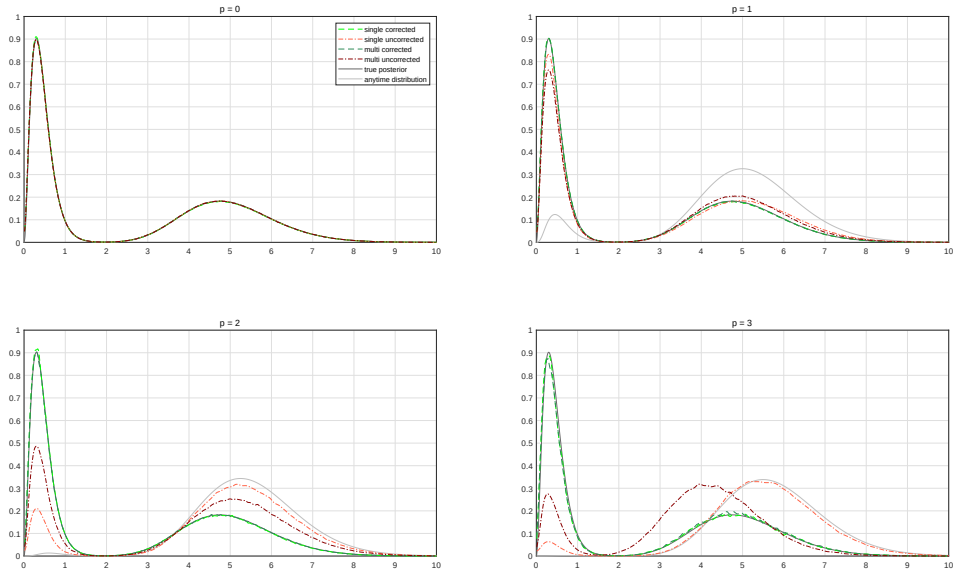


Fig. 4.2 Density estimates of the cold chain for bias corrected and uncorrected runs of the single (`APTMC-1`) and multi-processor (`APTMC-W`) algorithms on various hold time distributions $p \in \{0,1,2,3\}$. In the single-processor case, the cold chain is made up entirely of updates resulting from exchange moves. The solid *dark grey* line represents the true posterior density π and the solid *light grey* line the anytime distribution α. The case $p = 0$ represents an instance in which, in a real-time situation, the local moves do not take a random time to complete, and therefore all densities are identical. The two *green* dashed lines represent bias corrected densities and the *red* dot-dashed lines represent uncorrected densities. For $p \geq 1$, the two corrected densities are identical to the posterior, indicating that the bias correction was successful.

4.5.1.3 Performance evaluation

Next we verify that introducing the parallel tempering element to the Anytime Monte Carlo algorithm improves performance. A standard `MCMC` algorithm is run for computational complexities $p \in \{0,1,2,3\}$, applying the random walk Metropolis update described in Section 4.5.1.1. The single and multiple processor APTMC algorithms are run again for the same amount of virtual time, with exchange moves occurring every $\delta_{0:2} = 5$ units of virtual time for $p \leq 2$ and every $\delta_3 = 30$ units when $p = 3$. The single processor version is run on $\Lambda_s = 8$ chains, and the multi-processor on $W = 8$ workers, with $K = 2$ chains per worker, so $\Lambda_m = 16$ chains in total. This time, local moves are performed on the cold chain of the single processor `APTMC-1` algorithm.

To compare results, kernel density estimates of the posterior are obtained from the post burn-in cold chains for each algorithm using the `kde` function in MATLAB [2019], developed by Botev et al. [2010]. It is also important to note that even though all algorithms run for the same (virtual) duration, the standard `MCMC` algorithm is performing local moves on a single chain uninterrupted until the deadline, while the `APTMC-1` algorithm has to update $\Lambda = 8$ chains in sequence, and each worker w of the `APTMC-W` algorithm has to update $K = 2$ chains in sequence before exchange moves occur. Therefore, by time T the algorithms will not have returned samples of similar sizes. For a fair performance comparison, the sample autocorrelation function (acf) is estimated first of all. When available, the acf is averaged over multiple chains to reduce variance in its estimates. Other tools employed are

- *Integrated Autocorrelation Time* (*IAT*), the computational inefficiency of an MCMC sampler. Defined as

$$IAT_s = 1 + 2\sum_{\ell=1}^{\infty} \rho_s(\ell),$$

  where $\rho_s(\ell)$ is the autocorrelation at the ℓ-th lag of chain s. It measures the average number of iterations required for an independent sample to be drawn, or in other words the number of correlated samples with same variance as one independent sample. Hence, a more efficient algorithm will have lower autocorrelation values and should yield a lower *IAT* value. Here, the *IAT* is estimated using a method initially suggested in Sokal [1997] and Goodman and Weare [2010], and implemented in the Python package `emcee` by Foreman-Mackey et al. [2013] (Section 3). Let

$$\hat{IAT}_s = 1 + 2\sum_{\ell=1}^{M} \hat{\rho}_s(\ell),$$

  where M is a suitably chosen cutoff, such that noise at the higher lags is reduced. Here,

the smallest M is chosen such that $M \geq D\hat{\rho}_s(M)$ where $D \approx 6$. More information on the choice of D is available in Sokal [1997].

- *Effective Sample Size* (*ESS*), the amount of information obtained from an MCMC sample. It is closely linked to the *IAT* by definition:

$$ESS_s = \frac{N_s}{IAT_s},$$

where N_s is the size of the current sample s. The *ESS* measures the number of independent samples obtained from MCMC output.

As per Foreman-Mackey et al. [2013], when multiple repeat runs of an experiment are performed (see Section 4.5.4), the *IAT* for a given algorithm is obtained by averaging the acf returned by this algorithm over the repeat runs, and the resulting *ESS*s of each run are summed to obtain a cumulative *ESS* for this algorithm.

The resulting *ESS* and *IAT* for different algorithms and computational complexities are computed and shown in Table 4.1. If an exchange move is accepted, the new state of the chain does not depend on the value of the previous state. This means that the autocorrelation in a chain containing a significant proportion of (accepted) samples originating from exchange moves will be lower. For low p, significantly more local moves occur before each deadline, as hold times are short, while for a higher p, the hold times are longer and hence fewer local moves are able to occur. Therefore, higher values of p will yield a higher proportion of samples from exchange moves, and thus a more notable increase in efficiency.

In Figure 4.3 we observe that the quality of the posterior estimates decreases as p increases. As a matter of fact, 10^7 units of virtual time tend to not be enough for the some of the posterior chains to completely converge. Indeed, while the standard `MCMC` algorithm performs reasonably well for $p = 0$, it becomes increasingly harder for it to fully converge for higher computational complexities. Similarly, the single processor `APTMC-1` algorithm returns reasonably accurate posterior estimates for $p \leq 2$ but then visibly underestimates the first mode of the true posterior for $p = 3$. In general, the multi-processor `APTMC-W` algorithm returns results closest to the true cold posterior for all p.

As for efficiency, Table 4.1 displays a much lower *IAT* and much higher *ESS* for both `APTMC` algorithms, indicating that they are much more efficient than the standard `MCMC` algorithm. This is further supported by the sample autocorrelation decaying much more quickly for `APTMC` algorithms than for the `MCMC` algorithm for all p in Figure 4.4. The multi-processor `APTMC-W` algorithm also yields *IAT* values that are lower than those returned by the single processor `APTMC-1` algorithm for $p < 3$, and similarly yields effective sample sizes that are higher for all p. The *ESS* and *IAT* values for chains that have not converged to

their posterior (their resulting kernel density estimates significantly under or overestimate modes in Figure 4.3) have been omitted from the table.

p	Multi-processor APTMC		Single-processor APTMC		Standard MCMC	
	IAT	*ESS*	*IAT*	*ESS*	*IAT*	*ESS*
0	53.925	12049	81.156	1202.2	1739.0	287.46
1	45.942	5888.3	95.104	708.74	2818.2	64.047
2	80.871	1168.4	132.79	448.92	-	-
3	131.91	116.51	-	-	-	-

Table 4.1 Integrated autocorrelation time (*IAT*) and effective sample size (*ESS*) for runs of the single, multi-processor Anytime parallel tempering and standard MCMC algorithms. The algorithms were run for 10^6 units of virtual time for computational complexity $p = 0$ and 10^7 units for $p \geq 1$, and the resulting *ESSs* were scaled down for consistency with $p = 0$. The *ESS* and *IAT* values for chains that have not converged to their posterior (their resulting kernel density estimates significantly under or overestimate modes in Figure 4.3) have been omitted.

Next, we consider an application of the APTMC framework to ABC, a class of algorithms that are well-adapted to situations in which the likelihood is either intractable or computationally prohibitive. ABC features a real hold time at each MCMC iteration, making it an ideal candidate for adaptation to the Anytime parallel tempering framework.

4.5.2 ABC toy example: univariate Normal distribution

To validate the results of Section 4.4.2, consider another simple example, initially featured in Lee [2012], and adapted here within the APTMC framework. Let Y be a Gaussian random variable, i.e. $Y \sim \mathcal{N}(\theta, \sigma^2)$, where the standard deviation σ is known but the mean θ is not. The ABC likelihood here is

$$f^{\varepsilon}(y \,|\, \theta) = \Phi\left(\frac{y+\varepsilon-\theta}{\sigma}\right) - \Phi\left(\frac{y-\varepsilon-\theta}{\sigma}\right)$$

for $\varepsilon > 0$. Using numerical integration tools in MATLAB, it is possible to obtain a good approximation of the true posterior for any ε for visualisation. Let $y = 3$ be an observation of Y and $\sigma^2 = 1$, and put the prior $p(\theta) = \mathcal{N}(\theta; 0, 5)$ on θ. In this example, the exact posterior distribution for θ can easily be shown to be $\mathcal{N}\left(\theta; \frac{5}{2}, \frac{5}{6}\right)$.

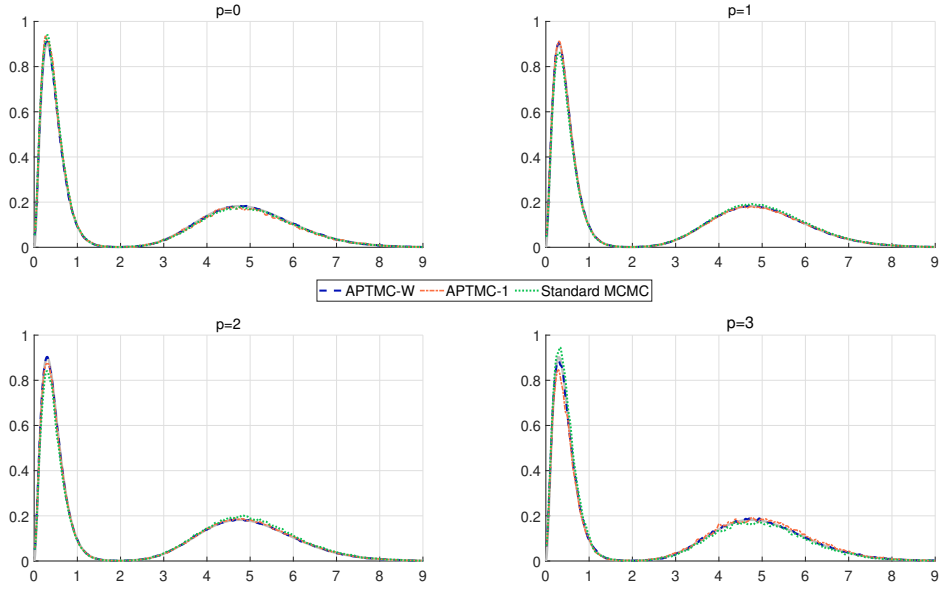


Fig. 4.3 Density estimates of the cold posterior for runs of the single (*orange*) and multiple (*blue*) processor APTMC algorithms (`APTMC-1` and `APTMC-W`, respectively) as well as the standard (*green*) `MCMC` algorithm. The *grey* line represents the true posterior density π. Each plot corresponds to a different hold time distribution $p \in \{0,1,2,3\}$. While the multi processor density has successfully converged for all p − as evidenced by the perfect overlap between the grey and dark blue lines −, the other two algorithms tend to increasingly struggle to estimate the first mode of the posterior as p increases.

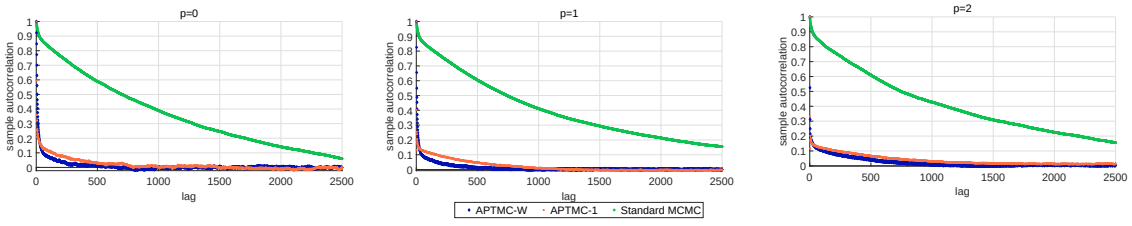


Fig. 4.4 Plots of the sample autocorrelation function up to lag 2500 of the post burn-in cold chain for runs of the single (*orange*) and multiple (*blue*) processor APTMC algorithms (`APTMC-1` and `APTMC-W`, respectively) as well as for the output of the standard Anytime Monte Carlo (`MCMC`) algorithm (*green*). Each plot corresponds to a different computational complexity $p \in \{0,1,2\}$. The two `APTMC` algorithms perform considerably better than standard `MCMC` for all p. The sample acf plot for $p = 3$ has been omitted due to both the `APTMC` and `MCMC` chains not having fully converged to their posterior.

When performing local moves (Algorithm 4.3), use a Gaussian random walk proposal with standard deviation $\xi = 0.5$. The real-time Markov jump process is run using $\Lambda = 10$ chains. The algorithm is run on a single processor for one hour or $T = 3600$ seconds in real time after a 30-second burn-in, with exchange moves occurring every $\delta_T = 5 \times 10^{-4}$ seconds (or 0.5 milliseconds). The radii of the balls $\varepsilon^{1:\Lambda}$ are defined to vary between $\varepsilon^1 = 0.1$ and $\varepsilon^\Lambda = 1.1$.

We verify that bias correction must be applied for all chains to converge to the correct posterior. This is done by visually comparing density estimates of each of the post burn-in chains to the true corresponding posterior (obtained by numerical integration). When bias correction is not applied, the `ABC-APTMC` algorithm does not exclude the currently working chain j in its exchange moves. In this case, every chain converges to an erroneous distribution which overestimates the mode of its corresponding posterior, as is visible in Figure 4.5. On the other hand, correcting the algorithm for such bias ensures that every chain converges to the correct corresponding posterior.

Next, we compare the performance of the `ABC-APTMC` algorithm to that of a standard `ABC` algorithm. For that, a more applied parameter estimation example is considered, for which the adoption of a likelihood-free approach is beneficial.

4.5.3 Moving average process

To illustrate a possible application of the `ABC-APTMC` algorithm, we now consider a common parameter estimation example taken from Marin et al. [2012]. A *moving average* process of order q, or MA(q) process, is used in time series analysis to model serial autocorrelation for the stochastic process $y = (y_k)_{k \in \mathbb{N}}$ up to lag q. Consider the expression

$$y_k = u_k + \sum_{i=1}^{q} \theta_i u_{k-i}, \tag{4.8}$$

where $u_k \overset{\text{iid}}{\sim} \mathcal{N}(0,1)$ for $k = 1, 2, \ldots$. In this example, we aim to estimate the posterior of the parameters $\theta = (\theta_1, \ldots, \theta_q)$.

In time series analysis, a standard invertibility condition is the following:

Condition 4.1. The roots of the polynomial

$$Q(x) = 1 - \sum_{i=1}^{q} \theta_i x^i$$

all lie outside the unit circle in the complex plane.

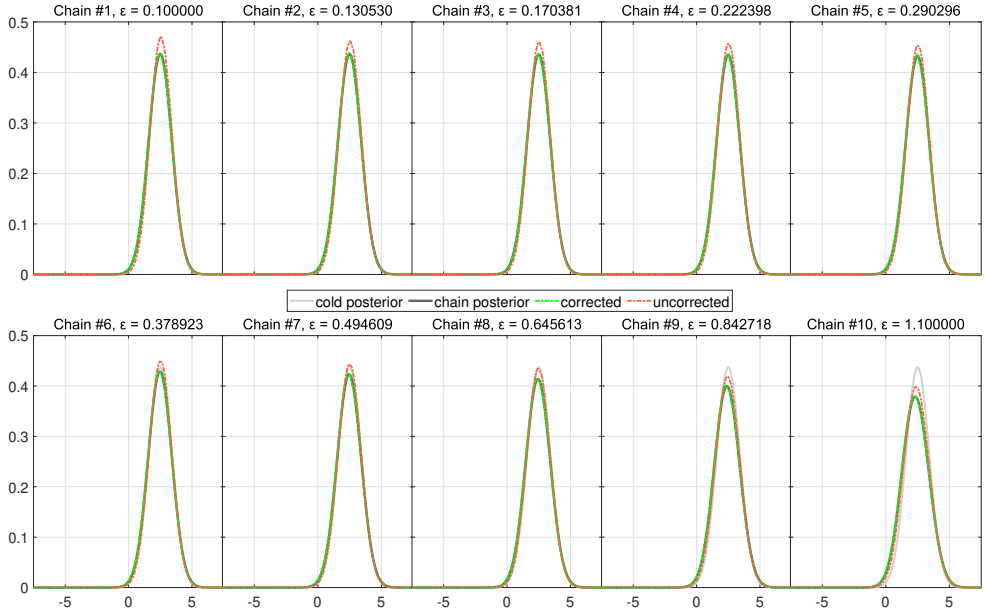


Fig. 4.5 Kernel density estimates of all chains for corrected and uncorrected runs of the single processor `ABC-APTMC` algorithm. In each subplot, the *light grey* line is fixed and represents the cold posterior for reference, the *dark grey* line represents each chain's target posterior (obtained by numerical integration), the dot-dashed *green* lines are kernel density estimates of the chain's posterior returned by the corrected algorithm and are indistinguishable from the dark grey line. The *orange* lines are kernel density estimates for the uncorrected algorithm, and do not agree with the dark grey line, as expected.

We can therefore define a uniform prior over the permitted range of θ_i's. In the case $q = 2$ this corresponds to sampling uniformly from the triangle

$$-2 < \theta_1 < 2, \qquad \theta_1 + \theta_2 > -1, \qquad \theta_1 - \theta_2 < 1.$$

From Hamilton [1994], the likelihood of the observed sequence $y = (y_1, \ldots, y_n)$ originating from a MA(q) process with parameters $\theta = (\theta_1, \ldots, \theta_q)$ is available as a multivariate Gaussian of the form $\mathcal{N}(0, \Omega)$ where Ω is the $n \times n$ variance-covariance matrix

$$\Omega = \mathbb{E}\left[y^\top y\right] = \begin{pmatrix} \gamma_0 & \gamma_1 & \gamma_2 & \ldots & \gamma_q & \ldots & 0 \\ \gamma_1 & \gamma_0 & \gamma_1 & \ddots & \gamma_{q-1} & \ddots & \vdots \\ \gamma_2 & \gamma_1 & \ddots & \ddots & & \ddots & \gamma_q \\ \vdots & \ddots & \ddots & & & & \gamma_{q-1} \\ \gamma_q & & & & & \gamma_1 & \vdots \\ \vdots & \ddots & & & \gamma_1 & \gamma_0 & \gamma_1 \\ 0 & \ldots & \gamma_q & \ldots & \gamma_2 & \gamma_1 & \gamma_0 \end{pmatrix},$$

where setting $\ell = |r|$ for $r = -q, \ldots, q$ and $\theta_0 = 1$, the covariance for lag ℓ is

$$\gamma_\ell = \sum_{i=0}^{q-\ell} \theta_i \theta_{i+\ell}.$$

In practice, when n is large, the computational cost of dealing with the matrix Ω becomes prohibitive. Other means of computing the likelihood have been devised, notably in Marin and Robert [2007], but the use of ABC provides an easy, likelihood-free approach to estimate the parameters θ.

Instead of evaluating the full likelihood at each iteration of local and exchange moves, we simulate the MA(q) process $(X_k)_{k=1}^n$ and evaluate its distance to the observations $(Y_k)_{k=1}^n$. There are multiple ways to evaluate such a distance: it can for example be the raw distance between the two datasets, but in general it is better to consider the distance between conveniently chosen summary statistics. More on the choice of summary statistics can be found in Marin et al. [2014]. To obtain the vector $\rho(y) = (\rho_1(y), \ldots, \rho_q(y))$ of summary statistics, we evaluate the quadratic distance between the first q sample autocorrelations, i.e. for $j = 1, \ldots, q$ compute

$$\rho_j(y) = \frac{1}{\tau_0} \sum_{k=j+1}^{n} M y_k y_{k-j}, \tag{4.9}$$

where $\tau_0 = \sum_{k=1}^{n} y_k^2$. Therefore, the vector $\rho(x)$ 'hitting' the ball $B_\varepsilon(\rho(y))$ of radius ε is here equivalent to $\|\rho(y) - \rho(x)\|_2 \leq \varepsilon$.

4.5.3.1 Methods and settings

In this example, the MA(2) process is considered for easy visualisation, i.e. $q = 2$. A sample of $n = 500$ observations $y = (y_1, \ldots, y_n)$ is simulated using parameters $\theta = (\theta_1, \theta_2) = (0.6, 0.2)$. In this case, the true likelihood can be computed without too much computational effort, and hence the true posteriors can be approximated using numerical integration.

This example serves to illustrate a possible application of the `ABC-APTMC-1` algorithm, and especially the computational benefits of introducing ABC exchange moves. Therefore, we include the standard version of the parallel tempering algorithm (denoted `ABC-PTMC-1`), in which exchange moves are performed after a fixed amount of local moves instead of following a real-time schedule. The `ABC-APTMC-1` and standard 1-hit kernel ABC algorithm with (`ABC-PTMC-1`) and without (standard `ABC`) exchange moves are run three times on a single processor for $T = 28800$ seconds (or 8 hours) of real time after an initial $t_{\min} = 1800$-second period of burn-in. The algorithms use a Gaussian random walk proposal with standard deviation $\xi = 0.05$ in the standard ABC case, and $\xi^{1:\Lambda}$ varying between $\xi^1 = 0.05$ and $\xi^\Lambda = 1$ in the `ABC-APTMC-1` and `ABC-PTMC-1` cases. Additionally, the `ABC-PTMC-1` and `ABC-APTMC-1` algorithms are run on $\Lambda = 12$ chains with exchange moves occurring every 12 local moves in the standard version and every 0.07 seconds in the anytime version. These values are chosen in order to ensure both algorithms spend the same median time performing local moves, following the tuning guidance in Section 4.3.4. The radii of the balls $\varepsilon^{1:\Lambda}$ are set to vary between $\varepsilon^1 = 0.02$ and $\varepsilon^1 = 1$. The standard `ABC` algorithm is run on a single chain, with the ball radius equal to ε^1, i.e. the radius corresponding to the cold chain in the parallel tempering algorithms. We compare the efficiency of the parallel tempering ABC algorithms to that of the standard, single-chain one. For that, a sample acf plot averaged over all runs is drawn in Figure 4.7 and the *IAT* and cumulative *ESS* over all runs are computed in Table 4.2 for comparison.

4.5.3.2 Performance evaluation

The scatter plots in Figure 4.6 illustrate the workings of the `ABC-APTMC-1` algorithm, with each chain targeting a distribution increasingly close to the true posterior as ε decreases. An equivalent set of plots can be obtained for the `ABC-PTMC-1` algorithm as well. As was already the case in Marin et al. [2012], the ABC approximation fails to reconstruct the posterior perfectly, even for ε as low as 0.02. Several ways to improve results are discussed

in Marin et al. [2012], including decreasing ϵ further and considering alternative summary statistics for comparison with the observations, as well as applying corrections to the ABC output such as in Beaumont et al. [2002]. After running for 8 hours, all three algorithms return indistinguishable posteriors. What remains to be compared is the performance of each algorithm.

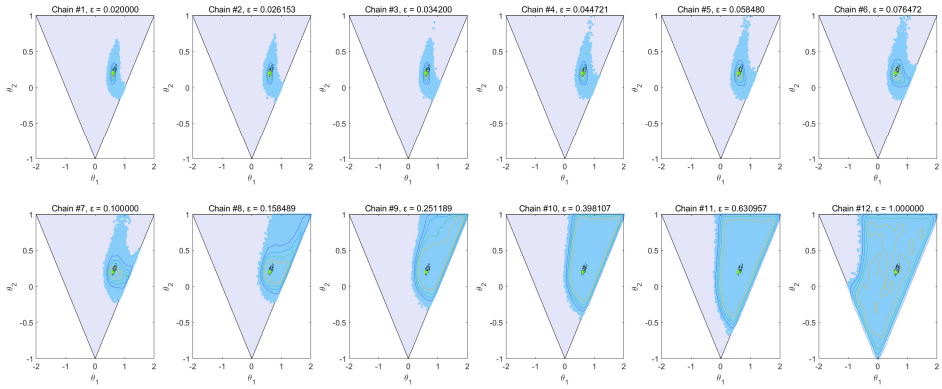


Fig. 4.6 Scatter plots of post burn-in chains for a run of the single processor `ABC-APTMC-1` algorithm. In each subplot, the *black* contour lines represent the level sets of the true posterior and the multi-coloured contour lines a bivariate Gaussian kernel density estimate from the samples returned for each chain by the algorithm, obtained using the `gkde2` function in MATLAB. The *green* star represents the true value $\theta = (0.6, 0.2)$ and the triangle is the range of acceptable values of θ. These plots illustrate the workings of ABC parallel tempering algorithms, with each chain targeting a distribution increasingly close to the true posterior as ϵ decreases.

It should be noted that while all three algorithms run for the same amount of time, the standard `ABC` algorithm only has one chain to update while the parallel tempering algorithms must perform local moves on 12 chains in sequence, and will hence return a cold chain with fewer samples despite the addition of exchange moves. This means that algorithms must be carefully tuned so that any improvement in performance is not offset by the reduction in the number of cold chain samples returned. Here again, a relevant comparison tool is the integrated autocorrelation time (IAT), and it is also important to verify that the parallel tempering algorithms return larger average effective sample sizes (ESS) after running for 8 hours.

As evidenced by the IAT values for both θ_1 and θ_2 in Table 4.2, the most inefficient of the three algorithms is the standard `ABC` algorithm, as it needs on average 1.5 to 2.9 times more samples to obtain the equivalent of an independent draw than are needed for the parallel tempering algorithms. This is further supported by the sample acf plots in Figure 4.7, which display a steeper decay in sample acf for the two parallel tempering algorithms. As a result, Table 4.2 also indicates that in three 8-hour runs, the addition of exchange moves has allowed

the ABC-PTMC-1 and ABC-APTMC-1 algorithms to double the *ESS* output compared to the standard ABC algorithm.

	ABC-APTMC-1		ABC-PTMC-1		Standard ABC	
	IAT	*ESS*	*IAT*	*ESS*	*IAT*	*ESS*
θ_1	16.448	64549	8.0068	63901	20.043	38023
θ_2	21.45	49497	11.139	45933	36.306	20991

Table 4.2 Effective sample size (*ESS*) and integrated autocorrelation time (*IAT*) over three 8-hour runs of the standard ABC and single processor ABC-APTMC-1 and ABC-PTMC-1 algorithms to estimate the posterior distributions of the parameters $\theta = (\theta_1, \theta_2)$ of a MA(2) process. The addition of exchange moves has effectively doubled the *ESS* for both components of θ.

This example serves to illustrate the improvements in performance brought by the addition of exchange moves. Indeed, the benefits of the Anytime framework are most evident in a multi-processor setting. In this single processor experiment, the ABC-PTMC-1 and ABC-APTMC-1 parallel tempering algorithms returned a similar *ESS* in Table 4.2, though the *IAT* for the ABC-APTMC-1 algorithm is slightly higher. This is due to the fact, mentioned in Section 4.3.4, that Anytime algorithms cannot always exchange the samples of adjacent chains, because they must exclude the chain that is currently computing; this causes a slightly higher rejection rate compared to the standard version. Fortunately the nature of the Anytime framework also allows the algorithm to return more samples, thus still enabling the algorithm to return a higher overall *ESS*. We note that the higher exchange move rejection rates issue can be mitigated by increasing the number of chains.

The multi-processor parallel tempering algorithms were not considered in this example. The next section considers a more advanced case study in which the likelihood is unavailable, and the benefits of performing exchange moves within the Anytime framework are illustrated both on a single and multiple processors.

4.5.4 Stochastic Lotka-Volterra model

In this section, we consider the stochastic Lotka-Volterra predator-prey model (Lotka [1926], Volterra [1927]), a model which has been explored in the past (Boys et al. [2008]; Wilkinson [2011]), including in an ABC setting (Fearnhead and Prangle [2012]; Lee and Łatuszyński [2014]; Prangle et al. [2017]; Toni et al. [2009]). Let $X_{1:2}(t)$ be a bivariate, integer-valued pure jump Markov process with initial values $X_{1:2}(0) = (50, 100)$, where $X_1(t)$ represents the number of preys and $X_2(t)$ the number of predators at time t. For small time interval Δt, we describe the predator-prey dynamics in the following way:

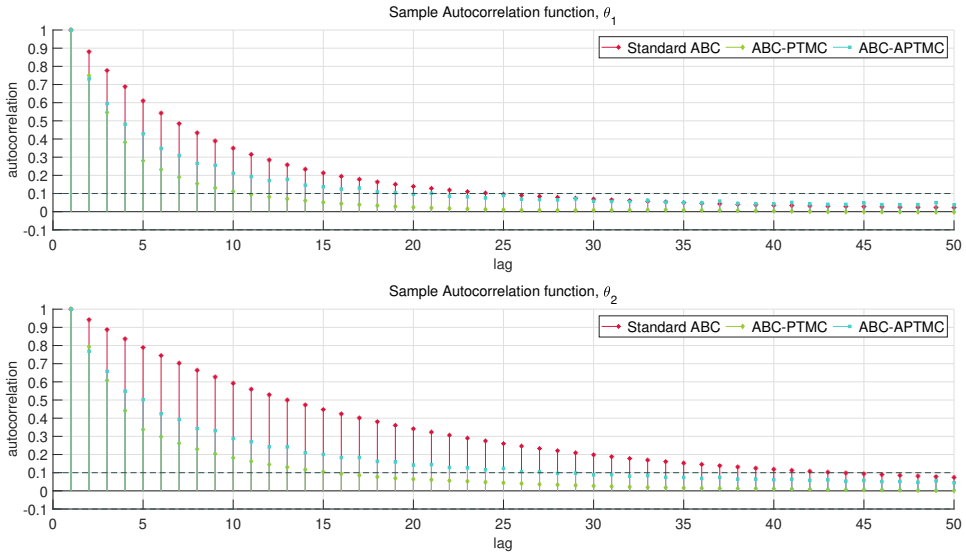


Fig. 4.7 Plots of the average sample autocorrelation function up to lag 100 of the post burn-in cold chain for eight 3-hour long runs of the standard `ABC` (*red*) and single processor `ABC-PTMC-1` (*green*) and `ABC-APTMC-1` (*blue*) algorithms to estimate the posterior distributions of the parameters $\theta = (\theta_1, \theta_2)$ of a MA(2) process. The two parallel tempering algorithms (`ABC-PTMC-1` and `ABC-APTMC-1`) display a steeper acf decay as opposed to the standard 1-hit kernel (standard `ABC`).

$$\mathbb{P}\{X_{1:2}(t+\Delta t) = z_{1:2} \,|\, X_{1:2}(t) = x_{1:2}\} = \begin{cases} \theta_1 x_1 \Delta t + o(\Delta t), & \text{if } z_{1:2} = (x_1+1, x_2), \\ \theta_2 x_1 x_2 \Delta t + o(\Delta t), & \text{if } z_{1:2} = (x_1-1, x_2+1), \\ \theta_3 x_2 \Delta t + o(\Delta t), & \text{if } z_{1:2} = (x_1, x_2-1), \\ o(\Delta t), & \text{otherwise.} \end{cases}$$

In this example, the only observations available are the number of preys, i.e. X_1 at 10 discrete time points. Following theory in Wilkinson [2011] (Chapter 6), the process can be simulated and discretised using the Gillespie [1977] algorithm, in which the inter-jump times follow an exponential distribution. The observations employed were simulated in Lee and Łatuszyński [2014] with true parameters $\theta = (1, 0.005, 0.6)$, giving $y = \{88, 165, 274, 268, 114, 46, 32, 36, 53, 92\}$ at times $\{1, \ldots, 10\}$.

This case study presents various challenges. The first challenge is that the posterior is intractable, and some of the components of the parameters $\theta := \theta_{1:3}$ (namely θ_2 and θ_3) exhibit strong correlations. For ABC, the 'ball' considered takes the following form for $\varepsilon > 0$

$$B_\varepsilon(y) = \{X_1(t) : |\log[X_1(i)] - \log[y(i)]| \leq \varepsilon, \forall i = 1, \ldots, 10\};$$

therefore, a set of simulated $X_1(t)$ is considered as 'hitting the ball' if all 10 simulated data points are at most e^{ε} times (and at least $e^{-\varepsilon}$ times) the magnitude of the corresponding observation in y.

In Lee and Łatuszyński [2014] (Algorithm 3), the 1-hit MCMC kernel (`ABC`), is shown to return the most reliable results by comparison with other MCMC kernels which are not considered here. The second challenge is that while this algorithm can be reasonably fast, it is highly inefficient as it has a very low acceptance rate, and thus the autocorrelation between samples for low lags is very high.

We have established that the race step in Algorithm 4.3 takes a random time to complete. In addition to that, its hold time distribution for the Lotka-Volterra model is a mixture between quick and lengthy completion times, as the simulation steps within the 1-hit kernel race are capable of taking a considerable amount of time despite often being almost instant. Indeed, when simulating observations using the discretised Gillespie algorithm, if the number of predators is low, prey numbers will flourish and the simulation will take longer. Hence, the third challenge in this particular model is that the race can sometimes get stuck for extended periods of time if the number of preys to simulate is especially high. Therefore, we aim to first of all improve performances by introducing exchange moves on a single processor (`ABC-PTMC`). Then − and most importantly − we further improve the algorithm by implementing it within the Anytime framework (`ABC-APTMC`), a method which works especially well on multiple processors.

4.5.4.1 One processor

The first part of this case study is run on a single processor and serves to demonstrate the gain in efficiency introduced by the exchange moves described in Algorithm 4.4. Departing slightly from the example in Lee and Łatuszyński [2014], define the prior on $\theta \in [0,\infty)^3$ for the single processor experiment to be $p(\theta) = \exp\{-\theta_1 - \theta_2 - \theta_3\}$, i.e. three independent exponential priors, all with mean 1. The proposal distribution is a truncated normal, i.e. $\theta' \,|\, \theta \sim T\mathcal{N}(\theta, \Sigma)$, $\theta' \in (0,10)$ with mean θ and covariance $\Sigma = \text{diag}\,(0.25, 0.0025, 0.25)$. The truncated normal is used in order to ensure that all proposals remain non-negative. For reference, 2364 independent samples from the posterior are obtained via ABC rejection sampling with $\varepsilon = 1$ and the density estimates in Figure 6 of Lee and Łatuszyński [2014] are reproduced. To obtain these posterior samples, 10^7 independent samples from the prior were required, yielding the very low 0.024% acceptance rate. This method of sampling from the posterior is therefore extremely inefficient, and the decision to resort to MCMC kernels in order to improve efficiency is justified.

On a single processor, the three algorithms considered are the vanilla 1-hit MCMC

kernel (`ABC`) defined in Algorithm 4.3, the single-processor version of the algorithm with added exchange moves (`ABC-PTMC-1`) and the same but within the Anytime framework (`ABC-APTMC-1`). They are run nine times for 100800 seconds (28 hours) − after 3600 seconds (1 hour) of burn-in − and their main settings are summarised in Table 4.3.

Given the aim is to compare the performance of these algorithms, it is also important to note that the parallel tempering algorithms, having to deal with updating multiple chains sequentially, are likely to return cold chains with fewer samples. The algorithms must therefore be properly set up such that the gain in efficiency introduced by exchange moves is not overshadowed by the greater number of chains and computational cost of having to update them all. In this experiment, the parallel tempering algorithms are run on $\Lambda = 6$ chains, each targeting posteriors associated with balls of radii $\varepsilon^{1:6} = \{1, 1.1447, 1.3104, 1.5, 11, 15\}$ and the proposal distribution has covariance $\Sigma^{1:6}$ where $\Sigma^{\lambda} = \text{diag}\left(\sigma^{\lambda}, \sigma^{\lambda} 10^{-2}, \sigma^{\lambda}\right)$ and $\sigma^{1:6} = \{0.008, 0.025, 0.05, 0.09, 0.25, 0.5\}$. Exchange moves are performed as described in Algorithm 4.4 and alternate between odd $(1,2),(3,4),(5,6)$ (excluding $(5,6)$ in the Anytime version) and even $(2,3),(4,5)$ pairs of eligible chains. As per Section 4.3.4, exchange moves for the `ABC-PTMC-1` algorithm are performed every $\Lambda = 6$ local moves, and the real-time deadline δ for exchange moves in the `ABC-APTMC-1` algorithm is determined by repeatedly measuring the times taken by the `ABC-PTMC-1` algorithm to perform these 6 moves and setting δ to be the median over all measured times.

Label	***Workers*** W	***Chains*** Λ	***Chains per worker*** K	***Exchange moves (every)***	***Anytime***
`ABC`	1	1	1	none	No
`ABC-PTMC-1`	1	6	6	6 local moves	No
`ABC-APTMC-1`	1	6	6	2.59 seconds	Yes

Table 4.3 Algorithm information and settings for the estimation of the parameters of a stochastic Lotka-Volterra predator-prey model by ABC on a single processor.

4.5.4.2 Multiple processors

Next, we demonstrate the gain in efficiency introduced by running the parallel tempering algorithm within the Anytime framework on multiple processors. The algorithms considered are the multi-processor `ABC-PTMC-W` and `ABC-APTMC-W` algorithms and their single processor counterparts `ABC-PTMC-1` and `ABC-APTMC-1`. This time, instead of relying on an informative, exponential prior on θ, we define a uniform prior between 0 and 3. The proposal distribution is still a truncated normal, but with tighter limits (corresponding to the prior) i.e. $\theta' \,|\, \theta \sim$

$T\mathcal{N}(\theta,\Sigma)$, $\theta' \in (0,3)$. Again, 1988 independent samples from the posterior are obtained for reference. They are generated via ABC rejection sampling with $\varepsilon = 1$. To obtain these posterior samples, 10^8 independent samples from the prior were required, yielding the even lower 0.002% acceptance rate.

The two algorithms are run on $\Lambda = 20$ chains, each targeting posteriors associated with balls of radii ranging from $\varepsilon^1 = 1$ to $\varepsilon^{20} = 11$ and proposal distribution covariances $\Sigma^{1:21}$ where $\Sigma^k = \text{diag}\left(\sigma^k, \sigma^k 10^{-2}, \sigma^k\right)$ for chain k and where values range from $\sigma^1 = 0.008$ to $\sigma^{20} = 0.5$. These are tuned so that the acceptance rates of exchange moves between adjacent chains remain on average greater than 70%. The algorithms are run four times for 864000 seconds (24 hours) and their main settings are summarised in Table 4.4. Given the non-negligible 1.1 second communication overhead, this experiment is run according to the third scenario from Section 4.3.3, i.e. dividing exchange moves into within- and between-worker exchange moves. As described in Section 4.3.4, a full set of worker-specific moves here consists of $K = 5$ local moves, followed by within-worker exchange moves between a pair of adjacent chains selected at random, followed by 5 more local moves. The between-worker exchange moves are performed after a full set of parallel moves on the master by selecting a pair of adjacent workers at random and exchanging between the warmest eligible chain from the first worker and coldest from the second so that they are adjacent.

Label	***Workers*** W	***Chains*** Λ	***Chains per worker*** K	***Communication overhead***	***Exchange moves (every)***	***Anytime***
`ABC-PTMC-1`	1	20	20	-	20 local moves	No
`ABC-APTMC-1`	1	20	20	-	11 seconds	Yes
`ABC-PTMC-W`	4	20	5	1.1 seconds	5 local moves	No
`ABC-APTMC-W`	4	20	5	1.1 seconds	5 local moves (*within* workers) 15.3 seconds (*between* workers)	Yes

Table 4.4 Algorithm information and settings for stochastic Lotka-Volterra predator-prey model on multiple processors.

4.5.4.3 Performance evaluation

All algorithms returned density estimates that were close to those obtained via rejection sampling. In order to compare the performance of the algorithms, as stated above, all algorithms compared are set to run for the same real time period. The *IAT* and cumulative *ESS* over all repeat runs are computed for all algorithms. While the *IAT* and sample autocorrelation plots are good tools for comparing efficiency, they do not take into account the computational cost of running 6 chains instead of a single chain. The *ESS*, on the other hand, gives us how many effective samples the different algorithms can return within a fixed

time frame. For example, a very fast algorithm could still return a higher *ESS* even if it has a much higher *IAT*. Additionally, to illustrate how the Anytime version of the parallel tempering algorithms works compared to standard `ABC-PTMC`, the real times all algorithms take to perform local and exchange moves are measured and their timelines plotted in Figure 4.9.

One processor Both the `ABC-PTMC-1` and `ABC-APTMC-1` algorithm display an improvement in performances: they return *IAT*s that are respectively 3.2 and 1.6 times lower on average than those of the standard `ABC` algorithm in Table 4.5, and display a steeper decay in sample autocorrelation in Figure 4.8. In the 28 hours (post burn-in) during which the algorithms ran, both parallel algorithms also yielded an increased *ESS*. The effect of the Anytime framework on the behaviour of the parallel tempering algorithm is demonstrated in Figure 4.9. The timeline of local moves for the `ABC-PTMC-1` algorithm illustrates the fact that local moves take a random amount of time to complete. This is mitigated since a deadline was implemented in the Anytime version of the algorithm. As a result, the bottom plot in Figure 4.9 displays more consistent local move times.

Note that in Table 4.5, while the improvement in *IAT* is significant, the increase in *ESS* after 28 hours is not particularly large. This is due to the previously mentioned erratic behaviour of the hold time distribution for this example. Other examples explored such as the moving average example of Section 4.5.3 yielded a much more significant increase in *ESS* after introducing exchange moves. We also note that in this example, the `ABC-PTMC-1` algorithm returned a lower *IAT* than its Anytime counterpart. There are two potential reasons to account for the *IAT*. The first is the many swaps which are cycling the same samples among the held chains in the Anytime case. The second, as mentioned in Section 4.3.4, is that Anytime algorithms cannot always exchange the samples of adjacent chains, because they must exclude the chain that is currently computing; this causes a slightly higher rejection rate compared to the standard version (in the multi-processor example with more chains, this is mitigated). However, the many more exchange moves of the Anytime algorithm do then result in more returned samples, which leads to a higher *ESS*. The single-processor experiment was mainly designed to demonstrate the performance improvements brought by adding exchange moves to the 1-hit MCMC kernel (referred to as standard `ABC`) and to show that Anytime does match the performance of parallel tempering on a single processor. Since a single processor is never idling, the strength of the Anytime framework is best illustrated in a multi-processor setting.

	Standard ABC		ABC-PTMC-1		ABC-APTMC-1	
	IAT	*ESS*	*IAT*	*ESS*	*IAT*	*ESS*
θ_1	69.476	7018.1	22.404	7618.6	44.071	7963.8
θ_2	122.73	3973	35.381	4824.2	69.803	5028
θ_3	150.74	3234.6	50.035	3411.3	98.929	3547.7

Table 4.5 Integrated autocorrelation time (*IAT*) and cumulative effective sample size (*ESS*) over nine 28-hour runs of the ABC, ABC-PTMC-1 and ABC-APTMC-1 algorithms to estimate the posterior distributions of the parameters $\theta = (\theta_1, \theta_2, \theta_3)$ of a stochastic Lotka-Volterra model. Improvements in performance are modest in this example.

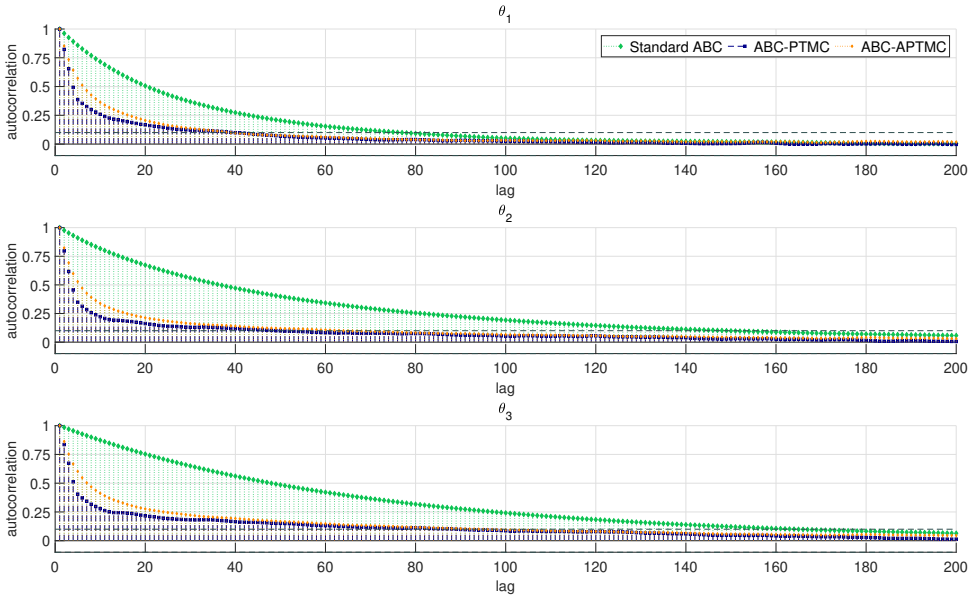


Fig. 4.8 Plots of the sample autocorrelation function up to lag 200 of the cold chain for runs of the standard ABC (*green*), single processor ABC-PTMC-1 (*blue*) and multi-processor ABC-PTMC-1 (*orange*) algorithms to estimate the posterior distributions of the parameters $\theta = (\theta_1, \theta_2, \theta_3)$ of a stochastic Lotka-Volterra model. The inclusion of exchange moves boosts efficiency and leads to a steeper decay in the parallel tempering algorithms.

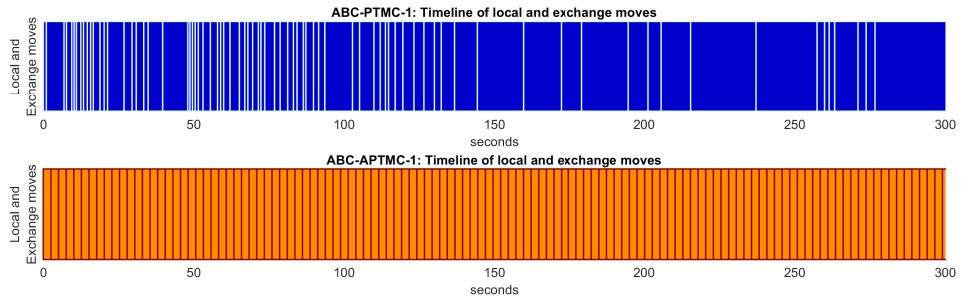


Fig. 4.9 Timeline of local and exchange moves for the `ABC-PTMC-1` and `ABC-APTMC-1` algorithms for the first 300 seconds. The exchange moves are represented by the *white* and *red* lines and the local moves by the *dark blue* and *orange* coloured blocks. Local moves take a random amount of time to complete, as illustrated by the times consumed by local moves for the `ABC-PTMC-1` algorithm. The anytime (`ABC-APTMC-1`) version effectively implements a hard deadline for the exchange moves (without introducing a bias), as can be seen by the regularity of local move times in the bottom figure.

Multiple processors In the multi-processor case study, both the `ABC-PTMC-1` and `ABC-PTMC-W` were set so that on each worker, an exchange move occurred after all chains had been updated locally once, as described in Table 4.4. The total number of samples returned by the `ABC-PTMC-W` algorithm is higher for all chains (see Table 4.7). However, the `ABC-PTMC-W` algorithm is just as affected by the distribution of the hold times being a mixture of quick and lengthy completion times as its single processor counterpart, and is just as prone to getting stuck in a race for an extended period. During this time, all processors sit idle while waiting for the race to complete, as illustrated in Figure 4.10. Therefore, the `ABC-PTMC-W` algorithm struggles to properly boost the total sample size output by the cold chain, and the *ESS* is not markedly higher on average in Table 4.6. On the other hand, thanks to the real time deadlines implemented, the `ABC-APTMC-W` algorithm is able to more than double the total size of the samples output (see Table 4.7), and the *ESS*s for the cold chain returned in Table 4.6 are on average 3.41 times higher than those of the single processor version.

As for the main comparison − namely Anytime vs standard ABC with exchange moves − both single and multi-processor Anytime algorithms return an *ESS* larger than their respective standard versions in Table 4.6. While the improvement on a single processor is not significantly large, the *ESS* has more than quadrupled on multiple processors. Figures 4.10 and 4.11 illustrate well the advantage of implementing a real-time deadline to local moves. At most local moves, the issue in which all workers sit idle waiting for the slowest to finish arises for the `ABC-PTMC-W` algorithm. On the other hand, Figure 4.11 shows that the Anytime version of the algorithm is making better use of the allocated computational

resources. The Anytime framework ensures that none of the workers need to wait for the slowest among them to finish, allowing for more exploration of the sample space in the faster workers. Additionally, the real time deadline ensures that even if chain k on Worker w remains stuck in a race for an extended period of time, the other workers are still updating. Therefore, while the remaining four chains on Worker w wait for chain k to complete its race, they also continue to be updated at regular intervals thanks to the exchange moves with other workers.

The addition of ABC exchange moves in his case study proved fruitful, as the effective sample size for the parameters of the Lotka-Volterra model was increased. However this required fine-tuning of the settings. The benefits of ABC parallel tempering will be stronger and more easily visible in a problem in which the parameters to be estimated have a multi-modal distribution, as a single chain may get stuck in local optima while multiple tempered chains will explore more of the sample space. Nonetheless, the introduction of the Anytime framework is a significant an important improvement. It ensures the various chains in ABC parallel tempering continue to be updated (via exchange moves) even when one of them is stuck performing local moves for longer than expected and encourages the algorithm to make better use of its allocated resources on multiple processors.

	One processor				**Multiple processors**			
	`ABC-PTMC-1`		`ABC-APTMC-1`		`ABC-PTMC-W`		`ABC-APTMC-W`	
	IAT	*ESS*	*IAT*	*ESS*	*IAT*	*ESS*	*IAT*	*ESS*
θ_1	39.535	269.89	72.475	362.62	48.621	266.7	39.898	1452.5
θ_2	72.908	146.35	88.446	297.14	67.395	192.4	72.79	796.14
θ_3	82.464	129.39	138.56	189.68	87.635	147.97	101.57	570.57

Table 4.6 Integrated autocorrelation time (*IAT*) and cumulative effective sample size (*ESS*) over four 24-hour runs of the `ABC-PTMC-1`, `ABC-APTMC-1`, `ABC-PTMC-W` and `ABC-APTMC-W` algorithms to estimate the posterior distributions of the parameters $\theta = (\theta_1, \theta_2, \theta_3)$ of a stochastic Lotka-Volterra model.

4.6 Conclusion

In an effort to increase the efficiency of MCMC algorithms, in particular for use on distributed computing, and for situations in which the likelihood is unavailable and/or compute times of the algorithms depend on their current states, the APTMC algorithm was developed for

the fist time. The algorithm combines the enhanced exploration of the state space, provided by the between-chain exchange moves in parallel tempering, with control over the real-time budget and robustness to interruptions available within the Anytime Monte Carlo framework.

Initially, the construction of the Anytime Monte Carlo algorithm with the inclusion of exchange moves on a single and multiple processors was verified on a toy Gamma mixture example. The performance improvements they brought were then demonstrated by comparing the algorithm to a standard MCMC algorithm. Subsequently, the exchange moves were adapted for pairing with the 1-hit ABC-MCMC kernel, a simulation-based algorithm within the ABC framework, which provides an attractive, likelihood-free approach to MCMC. The construction of the adapted ABC algorithm was verified using a simple univariate normal example. Then, the increased efficiency of the inclusion of exchange moves was demonstrated in comparison to that of a standard ABC algorithm on two parameter estimation problems. The first problem involved the parameters of a moving average process and demonstrated a significant increase in effective sample size brought by introducing exchange moves to the ABC 1-hit kernel. The second problem considered the estimation of the parameters of a stochastic Lotka-Volterra predator-prey model based on partial and discrete data, and the likelihood of this model is intractable. On a single processor, it was shown that introducing exchange moves provides an improvement in performance and an increase in the *ESS* compared to that of the standard, single chain ABC algorithm. The Anytime results for a single processor match the efficiency of standard parallel tempering, which is to be expected since the single processor is never idling in both the Anytime and standard versions. The *ESS* gains of Anytime become significant in a multi-processor setting, since one slow processor will lead to all the others idling in standard parallel tempering.

One major class of applications with local moves that have state-dependent real completion times and would benefit from the APTMC framework are transdimensional applications, such as RJ-MCMC (Green [1995]), implemented by Jasra et al. [2007b] within a parallel tempering algorithm. The performance of parallel tempering algorithms with temperature-dependent completion times, as addressed in Earl and Deem [2004], can also be improved by the Anytime framework. Examples of such algorithms occur in Hritz and Oostenbrink [2007]; Karimi et al. [2011]; Wang and Jordan [2003]. From a purely computing infrastructure related perspective, exogenous factors such as processor hardware, competing jobs, memory bandwidth, network traffic or I/O load also affect the completion time of local moves even if they are not state-dependent within the algorithm itself. This is the case in Rodinger et al. [2006]. In a more general setting, any population-based MCMC algorithm such as parallel tempering, SMC samplers (Del Moral et al. [2006]) considered in Chapter 5, or parallelised generalised elliptical slice sampling (Nishihara et al. [2014]), which combines a parallel

updates step (e.g. local moves) and an inter-processor communication step (e.g. exchange moves, resampling) can benefit from the APTMC framework in future implementations.

Conversely, as noted above, in our single-processor experiments, the Anytime results only matched the efficiency of standard parallel tempering, which is expected since there is no idling of the processor. Therefore, *ESS* gains of the Anytime framework are only truly significant in a multi-processor setting. Additionally, if communication overhead is negligible and one has access to a large number of high performance, uncontested processors, parallel tempering algorithms which do not include state-dependent local move completion times such as in Calderhead and Girolami [2009]; Friel and Pettitt [2008] will not benefit as much from the Anytime framework. Finally, embarrasingly parallel algorithms which do not require any inter-processor communication before completion will also only marginally benefit from the Anytime framework.

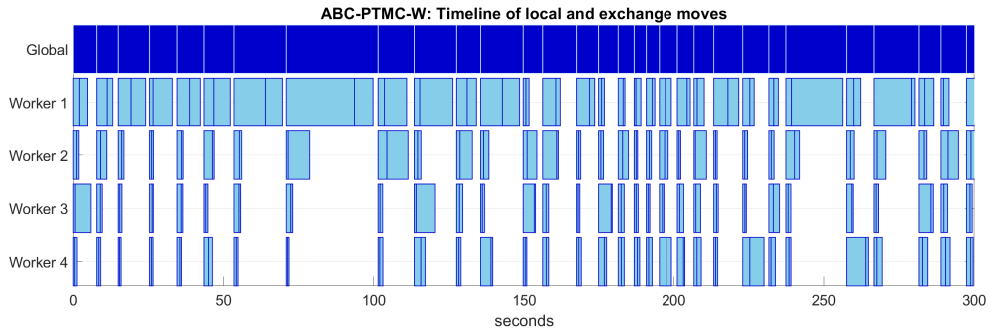


Fig. 4.10 Timeline of local and exchange moves for the `ABC-PTMC-W` algorithm for the first 300 seconds. Within and between worker exchange moves are represented by the *white lines* on the Global timeline and *blue lines* on the various Worker timelines, respectively. Local moves on each worker are represented by the *light blue* coloured blocks and the *dark blue* coloured blocks correspond to the global time all workers spend running in parallel, including communication overhead. Significant idle time is apparent on all workers as they always have to wait for the slowest among them to complete its set of local moves.

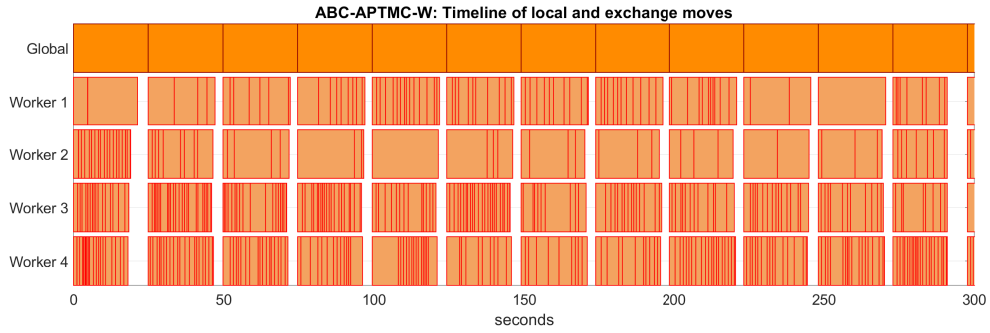


Fig. 4.11 Timeline of local and exchange moves for the `ABC-APTMC-W` algorithm for the first 300 seconds. Within and between worker exchange moves are represented by the *red lines*. Local moves on each worker are represented by the various *orange* coloured blocks, with the brighter blocks corresponding to the global time all workers spend running in parallel, including communication overhead. The significant idle time from Figure 4.10 has been greatly reduced thanks to the deadlines implemented.

Chain k	ε^k	σ^k	ABC-PTMC-1	ABC-PTMC-W	ABC-APTMC-1	ABC-APTMC-W
1	1	0.008	2667.5	3241.8	6570.3	14488
2	1.046	0.009	2790	3564.8	6934.8	16902
3	1.094	0.011	2796.8	3567.5	6941	16837
4	1.145	0.012	2797.3	3604.5	6924.8	17264
5	1.197	0.014	2793.8	3719.8	6931.3	15710
6	1.253	0.016	2786.3	3748.8	6951.5	17157
7	1.31	0.019	2784.8	3629.3	6961.3	18947
8	1.371	0.022	2795.5	3615	6941.5	18551
9	1.434	0.025	2805.5	3608.5	6950.8	18759
10	1.5	0.029	2803.5	3711.5	6962.8	17276
11	1.661	0.034	2798.5	3799.8	6962.3	46350
12	1.84	0.039	2803.3	3693.3	6983	53716
13	2.038	0.045	2814.8	3656.5	6995.3	53289
14	2.257	0.052	2799	3658.3	7008.8	53458
15	2.5	0.06	2783.5	3796.3	7029.8	46597
16	3.362	0.092	2787.5	4054.5	7038.5	68953
17	4.522	0.14	2783.8	3936.5	7009.5	79231
18	6.082	0.214	2781	3912.5	6982.5	78917
19	8.179	0.327	2780.8	3919.5	7002.8	79038
20	11	0.5	2665.8	3598.5	6604.3	67725

Table 4.7 Average sample sizes per chain returned over four 24-hour runs of the ABC-PTMC-1, ABC-APTMC-1, ABC-PTMC-W, ABC-APTMC-W algorithms to estimate the posterior distributions of the parameters θ of a stochastic Lotka-Volterra model on multiple processors in Section 4.5.4.3. The ball radius ε^k and proposal distribution covariance $\text{diag}\left(\sigma^k, \sigma^k 10^{-2}, \sigma^k\right)$ associated with each chain k are displayed for information.

Appendix 4.A Anytime distribution of the cold chain

To obtain the anytime distribution in the Gamma mixture example in Section 4.5.1, compute the three components of the expression in Equation 4.7:

1. The density of X

$$\pi(x) = \frac{x^{k_1-1}}{2\Gamma(k_1)\theta_1^{k_1}} e^{-\frac{x}{\theta_1}} + \frac{x^{k_2-1}}{2\Gamma(k_2)\theta_2^{k_2}} e^{-\frac{x}{\theta_2}},$$

   where $\Gamma(\cdot)$ is the gamma function.

2. The expectation of $H \,|\, x$ given by

$$\mathbb{E}[H \,|\, x] = \psi x^p + (1-\psi)x^p = x^p.$$

   The ψ factors cancel out, meaning that the anytime distribution is independent of ψ and therefore its value can be chosen to be 1 for convenience.

3. To compute $\mathbb{E}[H]$, use a property of conditional expectation and the honesty conditions of the Gamma(k_1+p, θ_1) and Gamma(k_2+p, θ_2) distributions:

$$\begin{aligned}
\mathbb{E}[H] &= \mathbb{E}[\mathbb{E}(H \,|\, x)] = \mathbb{E}[x^p] \\
&= \int \frac{x^{p+k_1-1}}{2\Gamma(k_1)\theta_1^{k_1}} e^{-\frac{x}{\theta_1}} \,\mathrm{d}x + \int \frac{x^{p+k_2-1}}{2\Gamma(k_2)\theta_2^{k_2}} e^{-\frac{x}{\theta_2}} \,\mathrm{d}x \\
&= \frac{\Gamma(k_2)\Gamma(p+k_1)\theta_1^p + \Gamma(k_1)\Gamma(p+k_2)\theta_2^p}{2\Gamma(k_1)\Gamma(k_2)} \\
&= \frac{C}{2\Gamma(k_1)\Gamma(k_2)},
\end{aligned}$$

   letting $C = \Gamma(k_2)\Gamma(p+k_1)\theta_1^p + \Gamma(k_1)\Gamma(p+k_2)\theta_2^p$.

Combining the three components,

$$\begin{aligned}\alpha(x) &= \frac{2\Gamma(k_1)\Gamma(k_2)}{C}\left(\frac{x^{p+k_1-1}}{2\Gamma(k_1)\theta_1^{k_1}}e^{-\frac{x}{\theta_1}} + \frac{x^{p+k_2-1}}{2\Gamma(k_2)\theta_2^{k_2}}e^{-\frac{x}{\theta_2}}\right) \\ &= \underbrace{\frac{\Gamma(k_2)\Gamma(p+k_1)\theta_1^{p+k_1}}{C\theta_1^{k_1}}}_{\varphi(p,k_{1:2},\theta_{1:2})}\underbrace{\frac{x^{p+k_1-1}}{\Gamma(p+k_1)\theta_1^{p+k_1}}e^{-\frac{x}{\theta_1}}}_{\text{Gamma}(p+k_1,\theta_1)} \\ &+ \underbrace{\frac{\Gamma(k_1)\Gamma(p+k_2)\theta_2^{p+k_2}}{C\theta_2^{k_2}}}_{\varphi'(p,k_{1:2},\theta_{1:2})}\underbrace{\frac{x^{p+k_2-1}}{\Gamma(p+k_2)\theta_2^{p+k_2}}e^{-\frac{x}{\theta_2}}}_{\text{Gamma}(p+k_2,\theta_2)}.\end{aligned}$$

And now substituting back the expression C in φ:

$$\varphi(p,k_{1:2},\theta_{1:2}) = \frac{1}{1+\frac{\Gamma(k_1)\Gamma(p+k_2)\theta_2^p}{\Gamma(k_2)\Gamma(p+k_1)\theta_1^p}}.$$

Similarly, we can obtain $\varphi'(p,k_{1:2},\theta_{1:2}) = 1-\varphi(p,k_{1:2},\theta_{1:2})$. Therefore, the anytime distribution $\alpha(\mathrm{d}x)$ is the following mixture of two Gamma distributions:

$$\alpha(\mathrm{d}x) = \varphi(p,k_{1:2},\theta_{1:2})\,\text{Gamma}\,(k_1+p,\theta_1) + [1-\varphi(p,k_{1:2},\theta_{1:2})]\,\text{Gamma}\,(k_2+p,\theta_2)\,.$$

Chapter 5

A General Anytime Sequential Monte Carlo Sampler for Changepoint Models using Reversible Jump MCMC

5.1 Chapter overview

Multiple changepoint problems are very useful for modelling the heterogeneity of observed data, as the changepoints divide the data into multiple segments corresponding to different models and/or latent parameters. A commonly used and easily adaptable approach to changepoint detection and inference is reversible jump MCMC (RJ-MCMC). However, RJ-MCMC can suffer from slow mixing, and in any MCMC algorithm, the need to assess whether the Markov chain has converged arises. To mitigate this issue, the RJ-MCMC algorithm is integrated into an SMC sampler, in which a population of particles is propagated through the data, improving efficiency and eliminating the need to assess convergence. As in any particle filter, distributed computing can be used to update the particles via RJ-MCMC in parallel on separate workers, but every once in a while, all workers must communicate for the resampling step. The transdimensional nature of RJ-MCMC can lead to parallel updates taking varying and random amounts of real time to complete, as they contain differing numbers of changepoints. This is accentuated when a latent parameter of interest is particularly expensive to update (See Section 5.6). As we explored in the previous chapter, the Anytime framework can be employed to ensure that all workers are synchronised for the resampling step and no idling occurs. In this chapter, we formulate a new general framework for changepoint detection using an RJ-MCMC algorithm within an Anytime SMC sampler. We present an application of the methodology to provide new insights and demonstrate the

gain in efficiency brought by employing the Anytime framework on a complex changepoint model.

5.2 Introduction

Multiple changepoint models are widely used in the literature to model the heterogeneity of data, with many applications (Aminikhanghahi and Cook [2017]), including biology (Fearnhead and Vasileiou [2009]; Xing et al. [2012]), signal processing (Chowdhury et al. [2012]; Punskaya et al. [2002]; Ureten and Serinken [2005]), physical sciences (Jarrett [1979]; Ó Ruanaidh and Fitzgerald [1996]; Raftery and Akman [1986]), climate analysis (Itoh and Kurths [2010]) and finance Dias and Embrechts [2002]. Indeed, in many applications, the observed data do not follow a single model but are divided into an often unknown number of segments, where each segment follows a different model. The changepoints correspond to the locations in the data at which the current model switches to the next (see Figure 5.2 for an illustration).

Often, a Bayesian approach is employed to estimate the number and position of the changepoints, as well as any latent parameters of the models describing the data in each segment. A variety of such approaches have been developed to estimate the changepoint posterior. For example, Chib [1998] employs Bayes factors to compare various changepoint configurations, Fearnhead [2006]; Fearnhead and Liu [2007] derive the commonly used filtering recursions to sample exactly from the changepoint posterior, and Lavielle and Lebarbier [2001] use Markov chain Monte Carlo (MCMC) along with a binary sequence of indicator variables to estimate the posterior changepoint number and locations. A commonly used and flexible method introduced in Green [1995] and applied in Benson et al. [2018]; Boys and Henderson [2004]; Wyse and Friel [2010] makes use of RJ-MCMC updates to explore the various changepoint configurations.

The RJ-MCMC algorithm tends to suffer from slow mixing (Brooks et al. [2003]), and as for any MCMC method, the need to assess whether the Markov chain has converged to its stationary distribution (i.e. the posterior of interest) arises. To mitigate both issues, Del Moral et al. [2006] developed sequential Monte Carlo (SMC) samplers, in which a population of particles evolves through the data, thus allowing to sequentially approximate the posterior of interest. An SMC sampler naturally progresses through the data like a particle filter, and can therefore be run in a distributed computing environment (Lee et al. [2010]), where the RJ-MCMC updates are performed in parallel on multiple processors. Another issue then arises: all processors must be synchronised before the resampling step can occur, but the time taken by some of the RJ-MCMC updates to complete depends on the current number of

changepoints on a given particle. As a result, the processors will sit idle until the slowest among them finishes working. To address this issue, just like in Chapter 4, the Anytime Monte Carlo framework introduced in Murray et al. [2016b] can be employed. In this chapter, we present a general method for changepoint inference using RJ-MCMC updates within an Anytime SMC sampler.

The chapter is organised as follows. In Section 5.3, we formally introduce multiple changepoint models in a general setting. In Section 5.4, the common changepoint inference aims are presented, as well as a quick overview of some of the main approaches to sample from the changepoint posterior, including RJ-MCMC. Section 5.5 introduces the focus of this chapter: an Anytime SMC sampler that uses RJ-MCMC for changepoint inference. In Section 5.6, we illustrate how to apply the algorithm for a complex changepoint model, and demonstrate the gain in efficiency brought by employing the Anytime framework. Finally, conclusions are discussed in Section 5.7.

5.3 Multiple changepoint model specification

5.3.1 Model overview

Consider a set of $n \in \mathbb{N}$ sequential observations $y := \{y_1, \ldots, y_n\}$ obtained during the time interval $[0, L]$, and taking values in $\mathscr{Y}$ such that each observation y_j depends on an unobserved parameter $\phi_j \in \Phi \subset \mathbb{R}$ for $j = 1, \ldots, n$. Let $\zeta := \{\zeta_i\}_{i=1}^{m+1}$, $m+1 < n$ denote a process of unobserved *latent states* taking values in the set $\mathscr{Z}$ and let $\tau := \{\tau_i\}_{i=1}^{m}$ denote the $\mathscr{T}$-valued *changepoint locations* $\tau_0 := 0 < \tau_1 < \ldots < \tau_m < n =: \tau_{m+1}$ such that the latent state ζ_i corresponds to the interval $(\tau_{i-1}, \tau_i]$ for $i = 1, \ldots, m+1$. The changepoint locations are the instances at which the values of the parameters ϕ_j change, i.e. the value of ϕ_j at any observation index j can be given by the following step function

$$\phi_j = \sum_{i=1}^{m+1} \varphi_i \mathbb{1}_{(\tau_{i-1}, \tau_i]}(j), \qquad j = 1, \ldots, n,$$

where $\mathbb{1}_{(\tau_{i-1}, \tau_i]}(j)$ denotes the indicator function for $j \in (\tau_{i-1}, \tau_i]$ and φ_i corresponds to the value of the latent parameters in the i-th interval. For the rest of this chapter, we refer to $\varphi := \{\varphi_i\}_{i=1}^{m+1}$ as the *latent parameters*. The latent states ζ may or may not be present, and generally provide more information on the state of the corresponding segment. For example, in Fearnhead and Vasileiou [2009]; Yıldırım et al. [2013], ζ_i corresponds to the family or model to which the i-th segment belongs, and therefore we have $\mathscr{Z} = [\![1, K]\!]$ where K is the

total number of families. The observations can be partitioned into $m+1$ disjoint segments

$$\left\{y_{1:T(\tau_1)}, y_{T(\tau_1)+1:T(\tau_2)}, \ldots, y_{T(\tau_{m-1})+1:T(\tau_m)}, y_{T(\tau_m)+1:n}\right\},$$

where the i-th segment $y_{T(\tau_{i-1})+1:T(\tau_i)} := \left\{y_{T(\tau_{i-1})+1}, \ldots, y_{T(\tau_i)}\right\}$ depends on the latent parameter φ_i for $i = 1, \ldots, m+1$ and $T(\tau_i)$ corresponds to the maximum observation index before the changepoint at τ_i occurs, i.e. map τ_i to the integers by defining the function $T : \mathscr{T} \to [\![0, n]\!]$ which finds its largest integer part as follows:

$$T(\tau_i) = \max_{j \in [\![0,n]\!]} \{j \leq \tau_i\}, \qquad i = 0, \ldots, m+1. \tag{5.1}$$

Note that if the changepoint locations are assumed to be discrete as in Benson et al. [2018]; Fearnhead [2006], then $T(\tau_i) = \tau_i$. Deciding whether to assume the changepoint locations are discrete or continuous depends on the model and algorithms employed for inference, as both approaches are able to simplify different aspects of the implementation slightly. For example, assuming continuous changepoint locations makes it straightforward to propose candidate locations for new changepoints in RJ-MCMC moves (see Section 5.4.2) without running the risk of selecting the location of an already existing changepoint, while assuming discrete changepoint locations makes the task of evaluating the likelihood easier.

Example 5.1. From Fearnhead and Vasileiou [2009]. Let $y = (y_1, \ldots, y_n)$ be the real-valued ($\mathscr{Y} = \mathbb{R}$) data obtained in the interval $[0, n]$. The data are divided into $m+1$ segments separated by m discrete-valued ($\mathscr{T} = \mathbb{N}$) changepoints $\tau_0 < \tau_1 < \ldots < \tau_m < \tau_{m+1}$ where $\tau_0 := 0$ and $\tau_{m+1} := n$. The i-th segment contains observations $\left(y_{\tau_{i-1}+1}, \ldots, y_{\tau_i}\right)$ and is associated with family $\zeta_i \in \mathscr{Z} = [\![1, K]\!]$ and latent parameters $\varphi_i = (\mu_i, \sigma_i^2) \in \Phi = \mathbb{R} \times (0, \infty)$. Figure 5.1 illustrates the observed data.

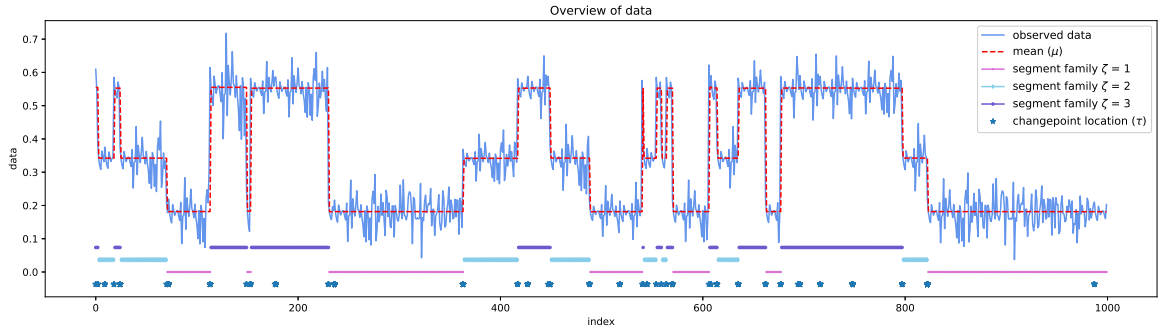


Fig. 5.1 Observed data (*light blue line*) simulated according to the changepoint model from Fearnhead and Vasileiou [2009]. The true mean μ parameter is displayed (*dashed red line*), as well as the true changepoint locations τ (*dark blue stars*) and the true families ζ to which each segment of the data belongs (*pink line* for $\zeta = 1$, *turquoise line* for $\zeta = 2$, *purple line* for $\zeta = 3$).

Let θ denote the *hyperparameters* of the model, and define a prior on the latent states and changepoint locations $p_\theta(\tau, \zeta)$. Assume that the latent parameters φ for each segment are conditionally independent given the latent states ζ with their density defined as $p_\theta(\varphi|\zeta)$, and that the observations $Y := \{Y_j\}_{j=1}^n$ are conditionally independent within a segment given the latent parameter and latent process values for that segment. The likelihood of the observations across all segments is then defined as

$$g(y|\varphi,\tau,\zeta) = \prod_{i=1}^{m+1} \prod_{j=T(\tau_{i-1})+1}^{T(\tau_i)} g(y_j|\varphi_i,\zeta_i),$$

and the joint density is defined as

$$\begin{aligned} p_\theta(y,\varphi,\tau,\zeta) &= p_\theta(\tau,\zeta)p_\theta(\varphi|\zeta)g(y|\varphi,\tau,\zeta) \\ &= p_\theta(\tau,\zeta)\prod_{i=1}^{m+1} p_\theta(\varphi_i|\zeta_i) \prod_{j=T(\tau_{i-1})+1}^{T(\tau_i)} g(y_j|\varphi_i,\zeta_i). \end{aligned} \tag{5.2}$$

It is often possible to integrate or 'collapse' out dependence on the latent parameters φ. Given latent state ζ_i and corresponding current segment $(\tau_{i-1},\tau_i]$, let $y_{s:t}$ be a set of observations such that $\tau_{i-1} < s \leq t \leq \tau_i$ belong to the segment with latent parameter φ_i. Integrate (or *collapse*) out the latent parameter to define the *evidence function* $R_\theta : [\![1,n]\!]^2 \times \mathscr{Z} \to [0,\infty)$ as

$$R_\theta(s,t,\zeta_i) = \int_\Phi \prod_{j=s}^{t} g(y_j|\varphi_i,\zeta_i)p_\theta(\varphi_i|\zeta_i)\mathrm{d}\varphi_i. \tag{5.3}$$

From this, we can also define the *potential function* $G_i^\theta : \mathscr{T}^2 \times \mathscr{Z} \to [0,\infty)$ of the i-th segment as follows:

$$G_i^\theta(\tau_{i-1},\tau_i,\zeta_i) := R_\theta\left(T(\tau_{i-1})+1, T(\tau_i), \zeta_i\right) = \int_\Phi \prod_{j=T(\tau_{i-1})+1}^{T(\tau_i)} g(y_j|\varphi_i,\zeta_i) p_\theta(\varphi_i|\zeta_i)\mathrm{d}\varphi_i. \tag{5.4}$$

The potential function G_i^θ essentially corresponds to the probability of the data observed in the i-th segment. Using Equation 5.2, the *collapsed joint density* of y, ζ and τ can be written as

$$p_\theta(y,\tau,\zeta) = p_\theta(\tau,\zeta)\prod_{i=1}^{m+1} G_i^\theta(\tau_{i-1},\tau_i,\zeta_i). \tag{5.5}$$

5.3.2 Choice of priors

The priors selected for the changepoint locations τ and latent states ζ depend on the inference aims, type of data and model specification. Assume first that either there is no latent state i.e. $\zeta = \emptyset$ or the latent state and changepoint locations are independent, i.e. $p_\theta(\tau,\zeta) = p_\theta(\tau)p_\theta(\zeta)$. If the changepoint locations τ are assumed to be discrete (Benson et al. [2018]; Fearnhead [2006]; Fearnhead and Vasileiou [2009]), a common choice of prior is

$$p_\theta(\tau) = \left[1 - F_\theta(n-\tau_m)\right]\upsilon_\theta(\tau_1)\prod_{i=2}^{m} f_\theta(\tau_i - \tau_{i-1}), \tag{5.6}$$

where the density $F_\theta(t) = \sum_{j=1}^t f_\theta(j)$ corresponds to the distance in time between successive changepoints, υ_θ is the density of the time to the first changepoint τ_1 and the density f_θ follows the negative binomial distribution given by

$$f_\theta(t) = \binom{t-s}{s-1}\lambda^s(1-\lambda)^{t-s}, \quad \upsilon_\theta = \frac{1}{s}\sum_{i=1}^{s}\binom{t-i}{i-1}\lambda^i(1-\lambda)^{t-i}. \tag{5.7}$$

A few observations can be made. First, note that if the hyperparameter $s = 1$, Equation 5.7 becomes the probability mass function for the geometric distribution, as used in Fearnhead and Vasileiou [2009]; Yao [1984]. We also note that setting a prior on the distance between two changepoints also implies a prior on the total number m of changepoints (Fearnhead and Liu [2007]). If the changepoints are assumed to occur at continuous time points (Del Moral et al. [2006]; Green [1995]), their prior is generally defined conditional on the number of changepoints m. In this case, a prior is also defined on the number of changepoints, i.e. $p_\theta(m)$. In Green [1995], given m changepoints, the changepoint locations are distributed as even-numbered order statistics from $2m+1$ points uniformly distributed on the observation

interval $[0, L]$. In Del Moral et al. [2006] a similar order statistic from m points uniformly distributed on the observation interval is employed. Order statistic based priors can also be straightforwardly employed for discrete-valued changepoints. An advantage of employing order statistics as priors is that even when assuming discrete-valued changepoint locations, it avoids the instance of two adjacent observations being selected as changepoints.

Relaxing the assumption that the changepoint locations and latent state are independent, we can instead assume that the changepoint locations depend on the latent states. In this case, define the prior as follows:

$$p_\theta(\tau, \zeta) = p_\theta(\tau|\zeta)p_\theta(\zeta)$$

and, for example, rewrite the density in Equation 5.6 as

$$p_\theta(\tau|\zeta) = [1 - F_\theta(n - \tau_m|\zeta_{m+1})]\, \upsilon_\theta(\tau_1|\zeta_1) \prod_{i=2}^{m} f_\theta(\tau_i - \tau_{i-1}|\zeta_i),$$

where the density f_θ in Equation 5.7 is given by

$$f_\theta(\tau_i - \tau_{i-1} = t|\zeta_i = k) = \binom{t-s}{s-1} \lambda_k^s (1-\lambda_k)^{t-s},$$

and the densities F_θ and υ_θ are rewritten similarly.

As for the latent parameters φ, in order to ensure that the integral in Equation 5.4 is tractable, an appropriate conjugate prior should be selected for the latent parameters φ. In the case that a conjugate prior is unavailable, Fearnhead [2006] suggests that if φ is low-dimensional, the integral can be calculated numerically.

5.4 Bayesian changepoint inference

The first aim of changepoint inference is to identify the number and location of the changepoints τ. In order to achieve this, the posterior probability density function of τ and ζ given the observations Y must be estimated. The posterior is given by $p_\theta(\tau, \zeta|y) \propto p_\theta(y, \tau, \zeta)$ where $p_\theta(y, \tau, \zeta)$ is the joint density defined in Equation 5.5. In more complex models, the latent states ζ are also of particular interest, such as in Fearnhead and Vasileiou [2009]; Yıldırım et al. [2013] where ζ_i represents the model to which the segment $(\tau_{i-1}, \tau_i]$ belongs. We take a closer look at the most commonly used methods to estimate the latent state posterior density $p_\theta(\tau, \zeta|y)$, namely filtering distributions by Fearnhead [2006] and RJ-MCMC by Green [1995].

5.4.1 Filtering recursions

We give a brief overview of a popular approach to changepoint analysis presented in Fearnhead [2006]; Fearnhead and Liu [2007] that enables exact sampling from the posterior $p_\theta(\tau, \zeta|y)$. Similar work has also been done by Barry and Hartigan [1992]; Liu and Lawrence [1999]. First of all, assume that the integral for the evidence function R_θ in Equation 5.3 is tractable or cheap to compute numerically. For simplicity, we also assume that the changepoints occur at discrete time points, i.e. $T(\tau_i) = \tau_i$ where T is defined in Equation 5.1 and that their prior is given by the negative binomial in Equation 5.7. To illustrate the filtering recursion method, we solely focus on sampling from the posterior $p_\theta(\tau|y)$ for simplicity, but Fearnhead and Liu [2007, 2011] extended the definition of the filtering recursions to the possibility of including ζ as a parameter labelling the model or family of each segment. Define the probability

$$H_\theta(j) = \mathbb{P}_\theta(y_{j:n}|\text{changepoint at } j-1).$$

A backward recursion can be derived to obtain the probabilities $H_\theta(j)$ as follows (more details in Fearnhead [2006]). For $j = 1, \ldots, n$,

$$H_\theta(j) = \sum_{k=j}^{n-1} R_\theta(j,k) H_\theta(k+1) f_\theta(k+1-j) + R_\theta(j,n)(1 - F_\theta(n-j)),$$

where f_θ is the negative binomial density defined in Equation 5.7 and R_θ the evidence function defined in Equation 5.3. Then, it is possible to obtain the posterior density of τ_i given the previous changepoint τ_{i-1} as

$$p_\theta(\tau_i|\tau_{i-1}, y) = \frac{G_i^\theta(\tau_{i-1}, \tau_i) H_\theta(\tau_i + 1) f_\theta(\tau_i - \tau_{i-1})}{H_\theta(\tau_{i-1} + 1)}. \tag{5.8}$$

It is therefore possible to recursively sample exactly from the posterior density of the changepoints $p_\theta(\tau|y)$. A similar method can also be applied if a prior conditional on the number of changepoints is defined instead of the negative binomial. Sampling the changepoints one at a time would make the complexity of the algorithm $\mathcal{O}(n^2)$, but fortunately, Fearnhead [2006] devised an efficient algorithm that reduces the complexity so that it is linear in n.

Filtering recursions are an attractive approach to changepoint inference, as they allow for independent sampling from the posterior. However, Benson et al. [2018]; Wyse and Friel [2010] comment on a few drawbacks, such as the fact that the hyperparameters θ are fixed, and including a hyperprior would render the algorithm $\mathcal{O}(n^2)$ again, that reducing the algorithm complexity to linear makes it inexact, and that the transition densities in Equation

5.8 can become unstable for large datasets. Additionally, we note that the recursion relies on the assumption that the integral in Equation 5.3 is either tractable or cheap to compute numerically, which is not always the case (Quan et al. [2019]), and as of yet no method has been developed to sample from $p_\theta(\tau, \zeta|y)$ for a general ζ. Finally, each iteration of the filtering recursion takes the same real time to complete, meaning that this algorithm construction will not benefit as much from the Anytime framework. As a result, an alternative, RJ-MCMC approach to changepoint inference is preferred by some, which benefits from the performance improvements brought by the Anytime framework.

5.4.2 Reversible jump Markov chain Monte Carlo (RJ-MCMC) for changepoint inference

In Green [1995], an RJ-MCMC scheme was developed to obtain samples from the posterior in Equation 5.5. Depending on inference aims and whether the integral in Equation 5.4 is tractable, five possible updates can occur: add a changepoint (*birth*, $\mathfrak{M}_{m\to m+1}$), remove a changepoint (*death*, $\mathfrak{M}_{m\to m-1}$), change the positions τ of the changepoints ($\mathfrak{M}_\tau$), update the latent states ζ ($\mathfrak{M}_\zeta$) and finally, update the latent parameters φ ($\mathfrak{M}_\varphi$). The birth and death updates are transdimensional, or *between-model* updates, as they affect the number of changepoints, while the $\mathfrak{M}_\tau$, $\mathfrak{M}_\zeta$ and $\mathfrak{M}_\varphi$ updates are *within-model* moves, where the number of changepoints remains fixed. A typical iteration of the RJ-MCMC algorithm follows a Metropolis-within-Gibbs (Gilks et al. [1995]) scheme described in Algorithm 5.1.

Algorithm 5.1 RJ-MCMC moves for changepoint inference

Input: current sample $\left(\tau^k_{1:m^k}, \zeta^k_{1:m^k+1}, \varphi^k_{1:m^k+1}\right)$ in the Markov chain.

1: ($\mathfrak{M}_\varphi$) Update the latent parameters: $\varphi^{k+1}_{1:m^k} \sim p_\theta\left(\cdot|\tau^k_{1:m^k}, \zeta^k_{1:m^k+1}, y\right)$.

2: ($\mathfrak{M}_\zeta$) Update the latent state: $\zeta^{k+1}_{1:m^k} \sim p_\theta\left(\cdot|\varphi^{k+1}_{1:m^k+1}, \tau^k_{1:m^k}, y\right)$.

3: ($\mathfrak{M}_\tau$) Update the changepoint locations: $\tau^{k+1}_{1:m^k} \sim p_\theta\left(\cdot|\varphi^{k+1}_{1:m^k+1}, \zeta^{k+1}_{1:m^k+1}, y\right)$.

4: With probability b_{m^k}, perform a birth move, otherwise, perform a death move, i.e $\mathfrak{M}_{m^k\to m^{k+1}}$ update where

$$m^{k+1} = \begin{cases} m^k + 1 & \text{with probability } b_{m^k}, \\ m^k - 1 & \text{with probability } 1 - b_{m^k}. \end{cases}$$

Output: updated sample $\left(\tau^{k+1}_{1:m^{k+1}}, \zeta^{k+1}_{1:m^{k+1}+1}, \varphi^{k+1}_{1:m^{k+1}+1}\right)$.

Note that the latent states ζ may not be present, in which case the $\mathfrak{M}_\zeta$ update can be

skipped. To update τ, we proceed by proposing to move c changepoints τ_{C} where C is the c-dimensional set of indices of the changepoints selected. While the existence and construction of the updates in Steps 1 and 2 of the algorithm depends on the model specification and inference aims (see Section 5.4.3 for more details and Section 5.6.3.3 for an example of Step 2), the updates in Steps 3 and 4 generally follow the same pattern, described next.

5.4.2.1 Moving a changepoint

The simplest way of updating the changepoint locations (update $\mathfrak{M}_\tau$) is by a local Metropolis update. Let $\tau := \tau_{1:m}$ be the current set of changepoints, $\zeta := \zeta_{1:m+1}$ the current set of latent states and $\varphi := \varphi_{1:m+1}$ the current set of latent parameters. If τ_i is a changepoint we wish to move, propose a new location as follows: sample τ_i' uniformly from the set (τ_{i-1}, τ_{i+1}) if τ_i is continuous-valued and from the set $[\![\tau_{i-1}+1, \tau_{i+1}-1]\!] \setminus \tau_i$ if it is discrete. Denote the proposed set of changepoint locations $\tau' := (\tau_1, \ldots, \tau_i', \ldots, \tau_m)$. Then, accept the move with probability $\min\{1, A_\tau\}$ where the acceptance ratio is given by

$$A_\tau = \frac{p_\theta(\tau', \zeta)}{p_\theta(\tau, \zeta)} \frac{P_g(\tau_{i-1}, \tau_i', \zeta_i, \varphi_i)}{P_g(\tau_{i-1}, \tau_i, \zeta_i, \varphi_i)} \frac{P_g(\tau_i', \tau_{i+1}, \zeta_{i+1}, \varphi_{i+1})}{P_g(\tau_i, \tau_{i+1}, \zeta_{i+1}, \varphi_{i+1})}, \tag{5.9}$$

where $P_g(\tau_{i-1}, \tau_i, \zeta_i, \varphi_i) = \prod_{j=T(\tau_{i-1})+1}^{T(\tau_i)} g(y_j|\varphi_i, \zeta_i)$. If the integral in Equation 5.4 is tractable or cheap to compute numerically, the acceptance ratio in Equation 5.9 takes a different, simplified form, given by

$$A_\tau = \frac{p_\theta(\tau', \zeta)}{p_\theta(\tau, \zeta)} \frac{G_i^\theta(\tau_{i-1}, \tau_i', \zeta_i)}{G_i^\theta(\tau_{i-1}, \tau_i, \zeta_i)} \frac{G_i^\theta(\tau_i', \tau_{i+1}, \zeta_{i+1})}{G_i^\theta(\tau_i, \tau_{i+1}, \zeta_{i+1})}.$$

5.4.2.2 Birth and death of a changepoint

Let $\tau := \tau_{1:m}$ be the current set of m changepoints, $\zeta := \zeta_{1:m+1}$ the current set of latent states and $\varphi := \varphi_{1:m+1}$ the current set of latent parameters. A typical transdimensional move is accepted with probability $\min\{1, A\}$ where the acceptance ratio

$$A = \text{proposal ratio} \times \text{prior ratio} \times \text{likelihood ratio} \times \text{Jacobian}.$$

We first define the proposal ratio. Let b_m be the probability of a birth move occurring, and $d_m = 1 - b_m$ the probability of a death occurring. In both updates, the changepoint location τ^* will be selected with probability $r_m(\tau^*)$. For a birth move, the new changepoint is selected at random from a set of available changepoint locations, e.g. $r_m(\tau^*) = \frac{1}{n-m-1}$. So that the dimensions of the ratios match, a new ζ^* and φ^* must be proposed as well,

which involves sampling random variables u with density $q(u)$ and performing a change of variable to obtain ζ^* and φ^*. The purpose of the Jacobian $|J|$ is to take into account this change of variable, see Green [1995]; Richardson and Green [1997] and Section 2.8 for more details. For a death move, τ^* is similarly selected from the set of existing changepoints, e.g. $r_m(\tau^*) = \frac{1}{m}$. Thus, the (unnormalised) proposal density for adding a changepoint is given by $Q(m, m+1) = b_m r_m(\tau^*) q(u)$ and the density for removing a changepoint is $Q(m, m-1) = d_m r_m(\tau^*)$.

If a new changepoint is proposed at τ^* such that $\tau_{i-1} < \tau^* < \tau_i$, as well as a new ζ^* and φ^*, relabel the proposed set of changepoint locations $\tau' := \tau'_1, \ldots, \tau'_{m+1}$ so that $\tau'_i = \tau^*$ and $\tau'_{i+1} = \tau_i$. Similarly relabel the proposed set of latent states $\zeta' := \zeta'_1, \ldots, \zeta'_{m+2}$ and parameters $\varphi' = \varphi'_1, \ldots, \varphi'_{m+2}$. The opposite relabelling occurs for a death move. The prior and likelihood ratio are straightforward to obtain, and the acceptance ratio for a birth move is given by

$$A_b := \frac{Q(m, m+1)}{Q(m+1, m)} \times \frac{p_\theta(\tau', \zeta')}{p_\theta(\tau, \zeta)} \frac{p_\theta(\varphi'_i | \zeta'_i) p_\theta(\varphi'_{i+1} | \zeta'_{i+1})}{p_\theta(\varphi_i | \zeta_i)} \times \frac{P_g(\tau'_{i-1}, \tau'_i, \zeta'_i, \varphi'_i) P_g(\tau'_i, \tau'_{i+1}, \zeta'_{i+1}, \varphi'_{i+1})}{P_g(\tau_{i-1}, \tau_i, \zeta_i, \varphi_i)} \times |J| . \tag{5.10}$$

If the integral in Equation 5.4 is tractable or cheap to compute numerically, the acceptance ratio in Equation 5.10 takes the following different, simpler form:

$$A_b := \frac{Q(m, m+1)}{Q(m+1, m)} \times \frac{p_\theta(\tau', \zeta')}{p_\theta(\tau, \zeta)} \times \frac{G_\theta(\tau'_{i-1}, \tau'_i, \zeta'_i) G_\theta(\tau'_i, \tau'_{i+1}, \zeta'_{i+1})}{G_\theta(\tau_{i-1}, \tau_i, \zeta_i)} \times |J| .$$

The acceptance ratio for a death move can similarly be obtained and is given by $A_d := A_b^{-1}$.

5.4.2.3 Viability for the Anytime Monte Carlo framework

The fact that the total number of changepoints m is random due to the inclusion of the transdimensional moves in RJ-MCMC means that performing the updates $\mathfrak{M}_\zeta$, $\mathfrak{M}_\varphi$ and $\mathfrak{M}_\tau$ (in the case where a location change is proposed for all changepoints) is an $\mathcal{O}(m)$ task. As a result, every iteration of the RJ-MCMC algorithm will take a random amount of time to complete depending on the state of τ at the beginning of the iteration, thus making it an ideal candidate for the Anytime Monte Carlo framework. Additionally, being an MCMC algorithm, RJ-MCMC can easily be incorporated into parallelisable algorithms. For changepoint models, as we'll see in Section 5.5, it is possible to improve the performance of the RJ-MCMC algorithm by applying it within an SMC sampler.

5.4.3 Parameter inference

Most of the time, the latent parameters φ are also of interest. For example, in the case of the coal-mining disaster data introduced in Jarrett [1979] and widely explored in the literature (Del Moral et al. [2006]; Green [1995]; Wyse and Friel [2010]), the data consist of events occurring sequentially during a fixed timeline, and are presented as an inhomogeneous Poisson process, where the rate of the i-th segment $(\tau_{i-1}, \tau_i]$ is λ_i, so we have $\varphi_i = \lambda_i$ for $i = 1, \ldots, m+1$. Similarly, in Gaussian changepoint models such as the Well-log data (Benson et al. [2018]; Fearnhead [2006]; Ó Ruanaidh and Fitzgerald [1996]) or DNA sequence segmentation models (Boys and Henderson [2004]; Fearnhead and Vasileiou [2009]; Yıldırım et al. [2013]), a given data point y_j in the segment $(\tau_{i-1}, \tau_i] \ni j$ is assumed to be Gaussian distributed with mean μ_i and variance σ_i^2. In this case, $\varphi_i = (\mu_i, \sigma_i^2)$ for $i = 1, \ldots, m+1$. The second aim in both these types of models is to estimate the *latent parameter posterior* density $p_\theta(\varphi|y, \zeta)$ of the latent parameters φ given the latent states ζ and the observations Y. It may be possible to sample directly from the posterior (Boys and Henderson [2004]; Fearnhead and Vasileiou [2009]) if the prior $p_\theta(\varphi|\zeta)$ is defined to be conjugate. Otherwise, methods such as Metropolis-Hastings (Del Moral et al. [2006]; Green [1995]; Quan et al. [2019]), as described in the previous section, may be employed.

Finally, depending on the case study, the hyperparameters θ may also need to be estimated. One possible method to estimate them is maximum likelihood estimation (MLE) (Eckley et al. [2011]). For example, in Yıldırım et al. [2013], an online expectation-maximization (EM) algorithm was developed to obtain maximum likelihood estimates of the hyperparameters in a long sequence of observations. Alternatively, the hyperparameters can be estimated as in Boys and Henderson [2002]; Quan et al. [2019]; Wyse and Friel [2010], by defining a hyperprior $p(\theta)$ and obtaining their posterior distribution $p(\theta|\tau, \zeta, y)$ in a similar fashion as for the latent parameters φ.

Next, we present a novel changepoint detection algorithm which makes use of SMC samplers within the Anytime framework.

5.5 An Anytime sequential Monte Carlo (SMC) sampler for changepoint detection

Developing RJ-MCMC algorithms that mix well is not a straightforward task, as discussed in Brooks et al. [2003]. The transdimensional jumps presented in Section 5.4.2.2 are known as *nested jumps*, in which the proposal only has one additional or one fewer changepoint. This ensures that the probability that the Markov chain moves is not prohibitively small. However,

it is still necessary to assess whether the chain has reached its stationary distribution, and for example, a lack of prior knowledge on the true number of changepoints may lead to slow mixing, especially if the number of changepoints is very different from any initial estimates. In Del Moral et al. [2006], SMC samplers were developed that circumvent the need to assess convergence.

5.5.1 The sequential Monte Carlo (SMC) sampler

Sequential Monte Carlo (SMC) samplers, described in Section 2.5, were first introduced in Del Moral et al. [2006] as a method to sample sequentially from a series of probability distributions over a single space (or a sequence of nested transdimensional spaces). To approximate these distributions, a population of weighted samples or *particles* evolves over the data like a particle filter.

For notational simplicity we drop the index θ representing dependence on the hyperparameters, we also define $x := (\tau_{1:m}, \zeta_{1:m+1}, \varphi_{1:m+1})$. In changepoint inference, the posterior of interest is $\pi_n(x) := p(x|y_{1:n})$. Define a sequence of target distributions $\{\pi_t(x)\}_{t=1}^n$, where $\pi_0(x) = p(x)$ corresponds to the prior and $\pi_t(x) \propto p(x|y_{1:t})$ for $t = 1, \ldots, n$. The distributions are defined over a sequence of nested transdimensional spaces

$$\mathscr{E}_t = \bigcup_{m_t \in [\![1,t]\!]} \left[\{m_t\} \times \mathscr{T}^{m_t} \times \mathscr{Z}^{m_t+1} \times \Phi^{m_t+1} \right],$$

where m_t is the number of changepoints at step t and we have $\mathscr{E}_{t-1} \subset \mathscr{E}_t$ for $t = 1, \ldots, n$. Let κ_t denote the RJ-MCMC kernel targeting distribution π_t. Each target distribution π_t can be approximated by a population $\hat{\pi}_t$ of N weighted samples as follows:

$$\hat{\pi}_t(x) = \sum_{i=1}^{N} \omega_t^{(i)} \delta_{X_t^{(i)}}(x),$$

where $X_t^{(i)} := (\tau_{1:m_t}^{(i)}, \zeta_{1:m_t+1}^{(i)}, \varphi_{1:m_t+1}^{(i)})$ is the i-th particle in the population at step t, and $\omega_t^{(i)}$ is its corresponding normalised weight. A typical step t of a basic SMC sampler for changepoint inference is given in Algorithm 5.2. As discussed in Lee et al. [2010]; Murray [2013], Steps 1 and 4 of the algorithm are straightforward to implement on parallel processors. However, the resampling step (Step 3) still requires all the processors to communicate. Before resampling can occur, all processors must be synchronised, but as observed in Section 5.4.2.3, the real time taken for RJ-MCMC updates to complete for a given particle depends on the current number of changepoints on that particle. As a result, idle time becomes apparent, as resampling cannot take place before the slowest processor has finished computing. To

suppress this idle time, the SMC algorithm can be run within the Anytime framework (Murray et al. [2016b]).

Algorithm 5.2 SMC sampler with RJ-MCMC moves for changepoint inference

Where (i) *appears, the operation is performed for all* $i \in [\![1, N]\!]$.

Input: weighted particle population $\left(X_{t-1}^{(1:N)}, \omega_{t-1}^{(1:N)}\right)$ from step $t-1$.

1: Compute the incremental weights

$$\tilde{w}\left(X_{t-1}^{(i)}\right) = \frac{\pi_t\left(X_{t-1}^{(i)}\right)}{\pi_{t-1}\left(X_{t-1}^{(i)}\right)} \propto p_\theta\left(y_t | X_{t-1}^{(i)}, y_{1:t-1}\right).$$

2: Update and normalise the weights

$$\omega_t^{(i)} = \frac{\omega_{t-1}^{(i)} \tilde{w}\left(X_{t-1}^{(i)}\right)}{\sum_{j=1}^N \omega_{t-1}^{(j)} \tilde{w}\left(X_{t-1}^{(j)}\right)}.$$

3: Resample $\left(\iota_t^{(1:N)}, \omega_t^{(1:N)}\right) := \mathsf{resample}\left(\omega_t^{(1:N)}\right)$ according to Algorithm 3.6 and then set $X_t^{(i)} := X_{t-1}^{\left(\iota_t^{(i)}\right)}$.

4: Update $X_t^{(i)}$ for r_t RJ-MCMC iterations according to Algorithm 5.1, i.e.

$$X_t^{(i)} \sim \kappa_t\left(\cdot | X_t^{(i)}\right), \quad \text{for } r_t \text{ iterations.}$$

Output: updated particle population $\left(X_t^{(1:N)}, \omega_t^{(1:N)}\right)$.

5.5.2 The Anytime SMC sampler

The adaptation of SMC samplers to the Anytime framework was introduced in Murray et al. [2016b]. At step t, in the updating part of the SMC algorithm (Algorithm 5.2, Step 4), each particle $X_t^{(i)}$ is updated for a fixed number r_t of iterations of the RJ-MCMC algorithm (Algorithm 5.1). In the Anytime version of the algorithm, this is replaced with a real-time budget a_t. Additionally, as seen in the previous chapter (Chapter 4), the Anytime framework is at its most useful when implemented on distributed computing.

Define the anytime distribution α_t associated with target π_t and RJ-MCMC kernel κ_t. At step t, we wish to resample our particle population $(X_t^{(1:N)}, \omega_t^{(1:N)})$. To illustrate the problem, assume that the N particles are distributed across $P := \frac{N}{D}$ different processors,

i.e. D particles per processor denoted $\{X_t^{(p,d)}\}_{d=1}^D$ for processors $p = 1,\ldots,P$. When the RJ-MCMC updates are interrupted at the real-time deadline a_t, denote by j the index of the particle that was computing on each processor, meaning that we have $X_t^{(p,d)} \sim \pi_t$ for $d \in [\![1,D]\!] \setminus j$ and $X_t^{(p,j)} \sim \alpha_t \neq \pi_t$ for all $p = 1,\ldots P$. Therefore, if particle $X_t^{(p,j)}$ is selected for resampling, a length bias is introduced in the particle population. In Murray et al. [2016b], multiple ways are suggested to correct for the length bias. A straightforward method to implement is to discard particle $X_t^{(p,j)}$ from every processor $p = 1,\ldots P$ and draw a new $X_{t+1}^{(p,j)}$ from $\hat{\pi}_t$. In practice, this is done by sampling N indices from the population of $N-P$ unbiased samples. A typical step of the Anytime SMC samplers on multiple processors for changepoint inference is detailed in Algorithm 5.3.

Algorithm 5.3 Anytime SMC sampler with RJ-MCMC moves for changepoint inference

Input: weighted particle population $\left(X_{t-1}^{(1:N)}, \omega_{t-1}^{(1:N)}\right)$ from step $t-1$.

1: On each processor p, discard the currently working particle $X_{t-1}^{(p,j)}$ and compute the remaining incremental weights

$$\tilde{w}\left(X_{t-1}^{(p,d)}\right) = \frac{\pi_t\left(X_{t-1}^{(p,d)}\right)}{\pi_{t-1}\left(X_{t-1}^{(p,d)}\right)} \propto p_\theta\left(y_t | X_{t-1}^{(p,d)}, y_{1:t-1}\right),$$

for $d \in [\![1,D]\!] \setminus j$.

2: Collectively update and normalise the weights

$$\omega_t^{(i)} = \frac{\omega_{t-1}^{(i)} \tilde{w}\left(X_{t-1}^{(i)}\right)}{\sum_{k=1}^N \omega_{t-1}^{(k)} \tilde{w}\left(X_{t-1}^{(j)}\right)},$$

for all $i \in \mathbb{N}_D$ where $\mathbb{N}_D$ is the set of all eligible (not currently working) particles, i.e. $\mathbb{N}_D := [\![1,N]\!] \setminus \{l(p,j)\}_{p=1}^P$ where $l(p,j) = (p-1)D + j$.

3: Collectively resample $\left(\iota_t^{(1:N)}, \omega_t^{(1:N)}\right) := \mathsf{resample}\left(\omega_t^{(\mathbb{N}_D)}\right)$ and set $X_t^{(i)} := X_{t-1}^{\left(\iota_t^{(i)}\right)}$ for all $i = [\![1,N]\!]$.

4: On each processor p, update $X_t^{(p,d)}$ according to the RJ-MCMC kernel (Algorithm 5.1) for some real time a_t, i.e.

$$X_t^{(p,d)} \sim \kappa_t\left(\cdot | X_t^{(p,d)}\right) \quad \text{until real-time deadline } a_t,$$

for all $d \in [\![1,D]\!]$.

Output: updated population $\left(X_t^{(1:N)}, \omega_t^{(1:N)}\right)$.

Since updates on all the processors are interrupted at each step t following the same real-time budget a_t, the idle time is suppressed and all processors are automatically synchronised for the resampling step. This results in a more efficient, and often faster algorithm. The improvements in performance are demonstrated in the next section.

We now offer some guidance on how to distribute our particle population $(X_t^{(1:N)}, \omega_t^{(1:N)})$ between processors. Assume that we have access to P processors. For convenience, we recommend dividing the total number of particles N evenly among the processors. This means that N should be a multiple of P. By the nature of Anytime Monte Carlo, each processor must contain at least two particles for bias correction, but the total number N of particles required to efficiently explore the state space and obtain a good approximation of the posterior depends on the problem at hand and resources available (see Jasra et al. [2007a] for guidance on the choice of N). We observed in numerical experiments that the more particles are spread across processors, the more variation there is between real completion times of RJ-MCMC moves, and hence the more the algorithm will benefit from the Anytime framework. This means that an SMC sampler run on a large number P of processors, with $N = 2P$ particles will be more efficient and benefit far more from the Anytime framework than if only a single processor is available, i.e. $P = 1$, in which case no idling occurs and the benefits of the Anytime framework are not as significant. Note that by the nature of SMC samplers, each step t of the algorithm considers an increasing sample of observations $y_{1:t}$. This means that the real time budget a_t increases linearly over time (see Section 5.6.5 for an example), and apart from very low t, it is generally safe to assume that communication overhead between processors is negligible. Note also that while in Chapter 4, parallel tempering exchange moves occurred between chains running at adjacent temperatures, such specific moves do not occur in SMC samplers, so the particles on a given processor do not need to be sorted in any particular order.

Finally, the increment for t does not need to be 1. In other words, for a dataset of size n, it is possible to progress through the data faster than one data point at a time. For example, if progressing through the data Δ data points at a time, we have $t = \Delta, 2\Delta, 3\Delta, \ldots, (N_\Delta - 1)\Delta, n$ where $N_\Delta = \lceil n/\Delta \rceil$ is the number of target distributions in the SMC sampler. In this case, Algorithms 5.2 and 5.3 can be adjusted by replacing $t-1$ by $t-\Delta$ where relevant.

5.6 Illustration

In this section, we demonstrate the gain in efficiency brought by applying the SMC algorithm from the previous section within the Anytime framework on a complex changepoint detec-

tion model explored in Fearnhead and Vasileiou [2009]; Yıldırım et al. [2013] and briefly presented in Example 5.1.

Let $y = (y_1, \ldots, y_n)$ be the real-valued ($\mathscr{Y} = \mathbb{R}$) data obtained in the interval $[0, n]$ and let $y_{1:t}$ be the data observed up to t where $1 \leq t \leq n$. At step t, the data are divided into $m_t + 1$ segments separated by m_t discrete-valued ($\mathscr{T} = \mathbb{N}$) changepoints $\tau_0 < \tau_1 < \ldots < \tau_{m_t} < \tau_{m_t+1}$ where $\tau_0 := 0$ and $\tau_{m_t+1} := n$. The i-th segment contains observations $\left(y_{\tau_{i-1}+1}, \ldots, y_{\tau_i}\right)$ and is associated with family $\zeta_i \in \mathscr{Z} = [\![1, K]\!]$ and latent parameters $\varphi_i = (\mu_i, \sigma_i^2) \in \Phi = \mathbb{R} \times (0, \infty)$. The main application of this model is segmentation and estimation of mean GC (Guanine+Cytosine) content in a string of DNA. The main inference aims are therefore to estimate the mean μ of the data over time and to identify the family ζ to which each segment belongs.

5.6.1 Prior specification

Define the joint prior on the family of the segments and position of the changepoints up to step t as follows:

$$p_\theta(\tau_{1:m_t}, \zeta_{1:m_t+1}) = p_\theta(\zeta_1) p_\theta(\tau_{m_t+1} \,|\, \tau_{m_t}, \tau_{m_t+1}) \prod_{i=1}^{m_t} p_\theta(\tau_i \,|\, \tau_{i-1}, \zeta_i) p_\theta(\zeta_{i+1} \,|\, \zeta_i),$$

where conditional on the family $\zeta_i = k$, the length of the i-th segment follows a Geometric distribution with density given by

$$p_\theta(\tau_i \,|\, \tau_{i-1}, \zeta_i = k) = f_\theta(\tau_i - \tau_{i-1} | \zeta_i = k) = \lambda_k (1 - \lambda_k)^{\tau_i - \tau_{i-1} - 1},$$

and the segment families evolve according to the $K \times K$ Markov transition matrix P so that $p_\theta(\zeta_i = l | \zeta_{i-1} = k) = P_{kl}$. Finally, conditional on the family $\zeta_i = k$, the priors on latent parameters (μ_i, σ_i^2) for the i-th segment are given by

$$\mu_i \,|\, \sigma_i^2, \zeta_i = k \sim \mathscr{N}\left(\xi_k, \frac{\sigma_i^2}{\delta_k}\right), \qquad \sigma_i^2 \sim \text{Gamma}^{-1}(\nu, \gamma), \tag{5.11}$$

where $\mathscr{N}(\cdot, \cdot)$ denotes the Gaussian distribution and $\text{Gamma}^{-1}(\cdot, \cdot)$ the Inverse Gamma distribution.

5.6.2 Likelihood

The model at hand is a Gaussian changepoint model, so within the i-th segment of family ζ_i, between the changepoints τ_{i-1} and τ_i, the likelihood of observations y_j conditional on the

latent parameters (μ_i, σ_i^2) is given by

$$y_j \,|\, \mu_i, \sigma_i^2 \sim \mathcal{N}(\mu_i, \sigma_i^2) \text{ for } j \in [\![\tau_{i-1}+1, \tau_i]\!]. \tag{5.12}$$

The prior defined in Equation 5.11 is conjugate, so within a single segment of family $\zeta_i = k$, the latent parameters can be collapsed out following Equation 5.3 as follows. Let $\tau_{i-1}+1 \leq s \leq t \leq \tau_i$, then the evidence function for this interval is given by

$$R_\theta(s,t,k) = \pi^{-\frac{n(s,t)}{2}} \sqrt{\frac{\delta_k}{\delta_k'}} \frac{\Gamma(\nu')}{\Gamma(\nu)} \frac{(2\gamma)^\nu}{(2\gamma')^{\nu'}}, \tag{5.13}$$

where $n(s,t) = t-s+1$ corresponds to the number of observations in the interval, $\nu' = \nu + \frac{n(s,t)}{2}$, $\delta_k' = n(s,t) + \delta_k$ and $2\gamma' = 2\gamma + B_2(s,t) + \delta_k \xi_k^2 - \frac{(B_1(s,t)+\delta_k\xi_k)^2}{\delta_k'}$. The proof of Equation 5.13 is given in Appendix 5.A. As pointed out in Benson et al. [2018], the computations of the sums $B_1(s,t) = \sum_{i=s}^t y_i$ and $B_2(s,t) = \sum_{i=s}^t y_i^2$ can be sped up by noting that $B_1(s,t) = B_1(1,t) - B(1,s-1)$ and similarly for $B_2(s,t)$. By precomputing $B_1(1,j)$ and $B_2(1,j)$ for $j = 1,\ldots,n$, the computational and storage costs of evaluating the evidence function are greatly reduced. From Equation 5.13, we can obtain the potential function for the i-th segment

$$G_i^\theta(\tau_{i-1}, \tau_i, \zeta_i) = R_\theta(\tau_{i-1}+1, \tau_i, \zeta_i). \tag{5.14}$$

5.6.3 RJ-MCMC updates

In this section, we describe how the various within-model and between-model updates introduced as part of the RJ-MCMC sampler in Section 5.4.2 are carried out for this particular model. Denote by $\tau_{1:m_t}$ and $\zeta_{1:m_t+1}$ the changepoint locations and segment families given there are m_t changepoints at step $t \in [\![1,n]\!]$ of the SMC sampler. The fact that the latent parameters have a conjugate prior means that they are not necessary to update $\tau_{1:m_t}$ and $\zeta_{1:m_t+1}$ or perform any of the transdimensional moves, as they have been integrated out of the potential function in Equation 5.14. As a result, when performing RJ-MCMC updates, we are sampling from the collapsed posterior $p_\theta(\tau_{1:m_t}, \zeta_{1:m_t+1} | y_{1:t}) \propto p_\theta(y_{1:t}, \tau_{1:m_t}, \zeta_{1:m_t+1})$ where $p_\theta(y_{1:t}, \tau_{1:m_t}, \zeta_{1:m_t+1})$ is defined in Equation 5.5.

5.6.3.1 Birth and death of a changepoint

Recall that a birth move $(\mathfrak{M}_{m_t \to m_t+1})$ is proposed with probability b_{m_t} and a death move $(\mathfrak{M}_{m_t \to m_t-1})$ with probability $d_{m_t} = 1 - b_{m_t}$. The birth move is performed as follows: first of all, select a data point index τ^* uniformly at random from the indices that are not changepoints

(and excluding t), i.e. from the set $[\![1,t]\!] \setminus \tau_{1:m_t+1}$ where $\tau_{m_t+1} := t$. The candidate location τ^* is such that it lies between two existing changepoints, i.e. $\tau_{i-1} < \tau^* < \tau_i$. Then, propose new families ζ'_{i-1} and ζ'_i on the sub-intervals $[\tau_{i-1}+1, \tau^*]$ and $[\tau^*+1, \tau_i]$, respectively by defining the following change of variables:

$$(\zeta'_i, \zeta'_{i+1}) = (\zeta_i, u),$$

where the new family u is sampled uniformly from the set $[\![1,K]\!]$. The resulting Jacobian is 1. Finally, accept the birth move with probability $\min\{1, A_{m_t \to m_t+1}\}$ where

$$\begin{aligned} A_{m_t \to m_t+1} = & \frac{G_i^\theta(\tau_{i-1}, \tau^*, \zeta_i) G_i^\theta(\tau^*, \tau_i, u)}{G_i^\theta(\tau_{i-1}, \tau_i, \zeta_i)} && \text{likelihood ratio} \\ & \times \frac{P_{\zeta_i u} P_{u \zeta_{i+1}}}{P_{\zeta_i \zeta_{i+1}}} \frac{\lambda_u (1-\lambda_u)^{\tau_i - \tau^* - 1}}{(1-\lambda_{\zeta_i})^{\tau_i - \tau^*}} && \text{prior ratio} \\ & \times \frac{d_{m_t+1} K (t - m_t - 1)}{b_{m_t}(m_t+1)} && \text{proposal ratio and Jacobian.} \end{aligned}$$

The death of a changepoint must be constructed so that it satisfies detailed balance with the corresponding birth move. Select a changepoint τ_i to delete uniformly at random from the set of existing changepoints τ. The new segment becomes $[\tau'_{i-1}+1, \tau'_i] = [\tau_{i-1}+1, \tau_{i+1}]$. To propose a new family ζ'_i on the segment $[\tau'_{i-1}+1, \tau'_i]$, reverse the birth move calculation by inverting the previous change of variables, i.e. $\zeta'_i = \zeta_i$. Finally, accept the death move with probability $\min\{1, A_{m_t \to m_t-1}\}$ where

$$\begin{aligned} A_{m_t \to m_t-1} = & \frac{G_i^\theta(\tau_{i-1}, \tau_{i+1}, \zeta_i)}{G_i^\theta(\tau_{i-1}, \tau_i, \zeta_i) G_i^\theta(\tau_i, \tau_{i+1}, \zeta_{i+1})} && \text{likelihood ratio} \\ & \times \frac{P_{\zeta_i \zeta_{i+2}}}{P_{\zeta_i \zeta_{i+1}} P_{\zeta_{i+1} \zeta_{i+2}}} \frac{(1-\lambda_{\zeta_i})^{\tau_{i+1} - \tau_i}}{\lambda_{\zeta_{i+1}} (1-\lambda_{\zeta_{i+1}})^{\tau_{i+1} - \tau_i - 1}} && \text{prior ratio} \\ & \times \frac{b_{m_t - 1} m_t}{d_{m_t}(t - m_t) K} && \text{proposal ratio and Jacobian,} \end{aligned}$$

and note that $A_{m_t \to m_t+1} = A^{-1}_{m_t+1 \to m_t}$ is satisfied.

5.6.3.2 Updating the changepoint locations

As described in Section 5.4.2.1, it is straightforward to update the changepoint locations conditional on all other parameters (update $\mathfrak{M}_\tau$). In the i-th segment, draw the candidate location τ'_i uniformly from the set $[\![\tau_{i-1}+1, \tau_{i+1}]\!] \setminus \tau_i$ and accept to move to the new locations

with probability $\min\{1, A_\tau\}$ where

$$A_\tau = \left(\frac{1-\lambda_{\zeta_i}}{1-\lambda_{\zeta_{i+1}}}\right)^{\tau_i'-\tau_i} \times \frac{G_i^\theta(\tau_{i-1}, \tau_i', \zeta_i) G_i^\theta(\tau_i', \tau_{i+1}, \zeta_{i+1})}{G_i^\theta(\tau_{i-1}, \tau_i, \zeta_i) G_i^\theta(\tau_i, \tau_{i+1}, \zeta_{i+1})}.$$

In this example, all the changepoint locations are updated sequentially at each iteration of the RJ-MCMC algorithm.

5.6.3.3 Updating the families by forward-filtering backward simulation for a finite state space model (F-FFBSi)

Updating the segment families $\zeta_{1:m_t+1}$ conditional on all other parameters (update $\mathfrak{M}_\zeta$) is not as straightforward as the previous moves. To be able to sample from the posterior $p_\theta(\zeta_{1:m_t+1}|y_{1:t}, \tau_{1:m_t})$ at step t of the SMC sampler, we must reformulate this particular problem as a finite state space model conditional on the changepoint locations. Then, filtering and smoothing methods can be employed. These include forward-filtering backward smoothing or simulation (Baum et al. [1970]) for finite state space models, a combination of both (Scott [2002]), or forward smoothing methods (Del Moral et al. [2010]).

We now detail how to sample from $p_\theta(\zeta_{1:m_t+1}|y_{1:t}, \tau_{1:m_t})$. Given the state space is finite, this can be done using an exact version of the forward-filtering backward simulation algorithm (Algorithm 3.12) for finite state space models, presented in Scott [2002] and denoted F-FFBSi, for this example. Conditional on the changepoint locations $\tau_{1:m_t}$, denote as z_i the i-th segment of the data, i.e. $z_i := (y_{\tau_{i-1}+1}, \ldots, y_{\tau_i})$. Define the following finite state space model with Markov transition density

$$f(\zeta_i|\zeta_{i-1}) := P_{\zeta_{i-1}\zeta_i}, \quad f(\zeta_1) := \rho_{\zeta_1},$$

where the vector ρ is the initial distribution of ζ_1. Often, ρ is defined as the stationary distribution of the matrix P. Define the potential function

$$G_i(\zeta_i) := G_i^\theta(\tau_{i-1}, \tau_i, \zeta_i),$$

where G_i^θ is defined in Equation 5.14. The joint density is given by

$$p(z_{1:m_t+1}, \zeta_{1:m_t+1}) = f(\zeta_1) G_1(\zeta_1) \prod_{i=1}^{m_t+1} f(\zeta_i|\zeta_{i-1}) G_i(\zeta_i).$$

The aim is to obtain a sample from the posterior $p(\zeta_{1:m_t+1}|z_{1:m_t+1})$. To achieve this, we first

introduce the matrix density $a_i(\zeta_{i-1}, \zeta_i) := p(\zeta_{i-1}, \zeta_i | z_{1:i})$ and its marginal, known as the *forward density*, $\alpha_i(\zeta_i) := p(\zeta_i | z_{1:i})$. The following recursion is straightforward to establish

$$a_i(\zeta_{i-1}, \zeta_i) = \frac{G_i(\zeta_i) f(\zeta_i|\zeta_{i-1}) \alpha_i(\zeta_{i-1})}{\sum_{\zeta_{i-1:i} \in \mathscr{Z}^2} G_i(\zeta_i) f(\zeta_i|\zeta_{i-1}) \alpha_i(\zeta_{i-1})},$$
$$\alpha_i(\zeta_i) = \sum_{\zeta_{i-1} \in \mathscr{Z}} a_i(\zeta_{i-1}, \zeta_i),$$

for $i = 2, \ldots, m_t + 1$ (see Scott [2002]). The recursion is initialised at $\alpha_1(\zeta_1) = \frac{G_1(\zeta_1) f(\zeta_1)}{\sum_{\zeta_1 \in \mathscr{Z}} G_1(\zeta_1) f(\zeta_1)}$. To sample from the posterior, a backward simulation step is performed starting from the last segment $m_t + 1$ as follows: initialise by drawing $\zeta'_{m_t+1} \sim \alpha_{m_t+1}(\cdot)$, then for $i = m_t, \ldots, 1$, sample the previous segment family from the ζ'_{i+1}-th column of the matrix a_{i+1}, i.e. $\zeta'_i \sim a_{i+1}(\cdot, \zeta'_{i+1})$.

The F-FFBSi algorithm performs both a forward and backward pass through the segments, and its complexity is $\mathscr{O}(K^2 m_t)$ where K is the total number of possible segment families. As a result, the $\mathfrak{M}_\zeta$ update is the most computationally expensive and will have a strong effect on the different times taken by each processor to complete RJ-MCMC updates.

5.6.4 Estimating the latent parameters

For this changepoint model, recall the fact that the prior in Equation 5.11 for the latent parameters $(\mu_{1:m_t+1}, \sigma^2_{1:m_t+1})$ is conjugate. This means that they are not required for any of the RJ-MCMC updates. In a standard RJ-MCMC algorithm, we would still need to sample from the latent parameters posterior at each iteration by sampling from the sequence of densities

$$\mu_i | \sigma_i^2, y_{\tau_{i-1}+1:\tau_i}, \zeta_i = k \sim \mathscr{N}\left(\frac{B_1(\tau_{i-1}+1, \tau_i) + \delta_k \xi_k}{\delta'_k}, \delta'_k \right),$$
$$\sigma_i^2 | y_{\tau_{i-1}+1:\tau_i} \sim \text{Gamma}^{-1}\left(\nu', \gamma'\right),$$

for $i = 1, \ldots, m+1$, where δ'_k, ν', γ' and B_1 are defined in Section 5.6.2 and m is the total number of changepoints in the data at the current RJ-MCMC iteration. However, given the construction of the SMC samplers, a particle approximation of the posterior $p_\theta(\tau_{1:m_n}, \zeta_{1:m_n+1} | y_{1:n})$ is obtained at the final, n-th step of the algorithm. As a result, to obtain a particle approximation of the latent parameters posterior, it suffices to sample from $p_\theta(\tau_{1:m_n}, \zeta_{1:m_n+1} | y_{1:n})$ for each particle once the SMC sampler has completed its pass through the data.

5.6.5 Numerical experiments

In order to compare the performance of the SMC sampler with RJ-MCMC updates for changepoint detection with and without anytime moves, an experiment is run on a sample of 1000 data points simulated according to the changepoint model presented in this section. There are $K = 3$ segment families and the true hyperparameter values are as follows: $\lambda_{1:3} = (0.02, 0.05, 0.07)$, $P_{kl} = 0.5$ for $k, l = 1, 2, 3$, $\xi_{1:3} = (0.18, 0.34, 0.55)$, $\delta_{1:3} = (1.5, 0.7, 0.4)$, $\nu = 5$ and $\gamma = 0.007$. To facilitate visual comparison with posterior results, an overview of the data is available again in the top plot of Figure 5.2.

We first run the standard SMC sampler (`RJMCMC-SMC`) with $r_t = 5$ RJ-MCMC iterations per particle for each SMC step. Then, we run the SMC sampler with anytime RJ-MCMC updates (`RJMCMC-ASMC`), setting a linearly increasing budget (as suggested in Murray et al. [2016b]) of $a + C \approx 178$ minutes as follows:

$$a_t = C + \frac{2t}{N_\Delta(N_\Delta + 1)}a,$$

where N_Δ is the number of target distributions in the SMC sampler, C roughly corresponds to the time taken by the `RJMCMC-SMC` algorithm to complete its first SMC step, and the parameter a corresponds to the overall, user-defined compute budget. In this experiment, a is selected by trial and error so that the real-time budget follows a similar (linear) trend as the standard algorithm, as evidenced in Figure 5.4. The algorithms are both run using 1005 particles distributed across 15 CPU workers (so 67 particles per worker) with no contesting processes. For a fair comparison, the particles are resampled at every iteration instead of only when their effective sample size is below a certain threshold (Del Moral et al. [2006]).

Both algorithms are able to recover an accurate posterior estimate of the mean μ and segment families ζ as evidenced in the bottom plot of Figure 5.2. Compute profiles for both runs are given in Figure 5.3 and display the working and idle times of the 15 workers while the algorithms were running. The top plot, corresponding to the `RJMCMC-SMC` algorithm, displays visible idle times on all processors throughout its run. The compute profile for the bottom plot, on the other hand, shows that the introduction of anytime moves has significantly reduced all the waiting times. This is further evidenced in Figure 5.4. Indeed, while the compute times of local RJ-MCMC moves on the workers follow a similar increasing trend for both algorithms, all workers took the same time to complete their RJ-MCMC moves for the `RJMCMC-ASMC` algorithm, as expected, while there is significant variation in the compute times of the RJ-MCMC moves for the `RJMCMC-SMC` algorithm. As a result, the `RJMCMC-ASMC` algorithm makes better use of its allocated resources and is able to save nearly an hour of computation time. Note that increasing the number of particles and processors would lead to

even more variation in the compute times of RJ-MCMC moves in the `RJMCMC-SMC` algorithm, and it would benefit even more from the Anytime framework.

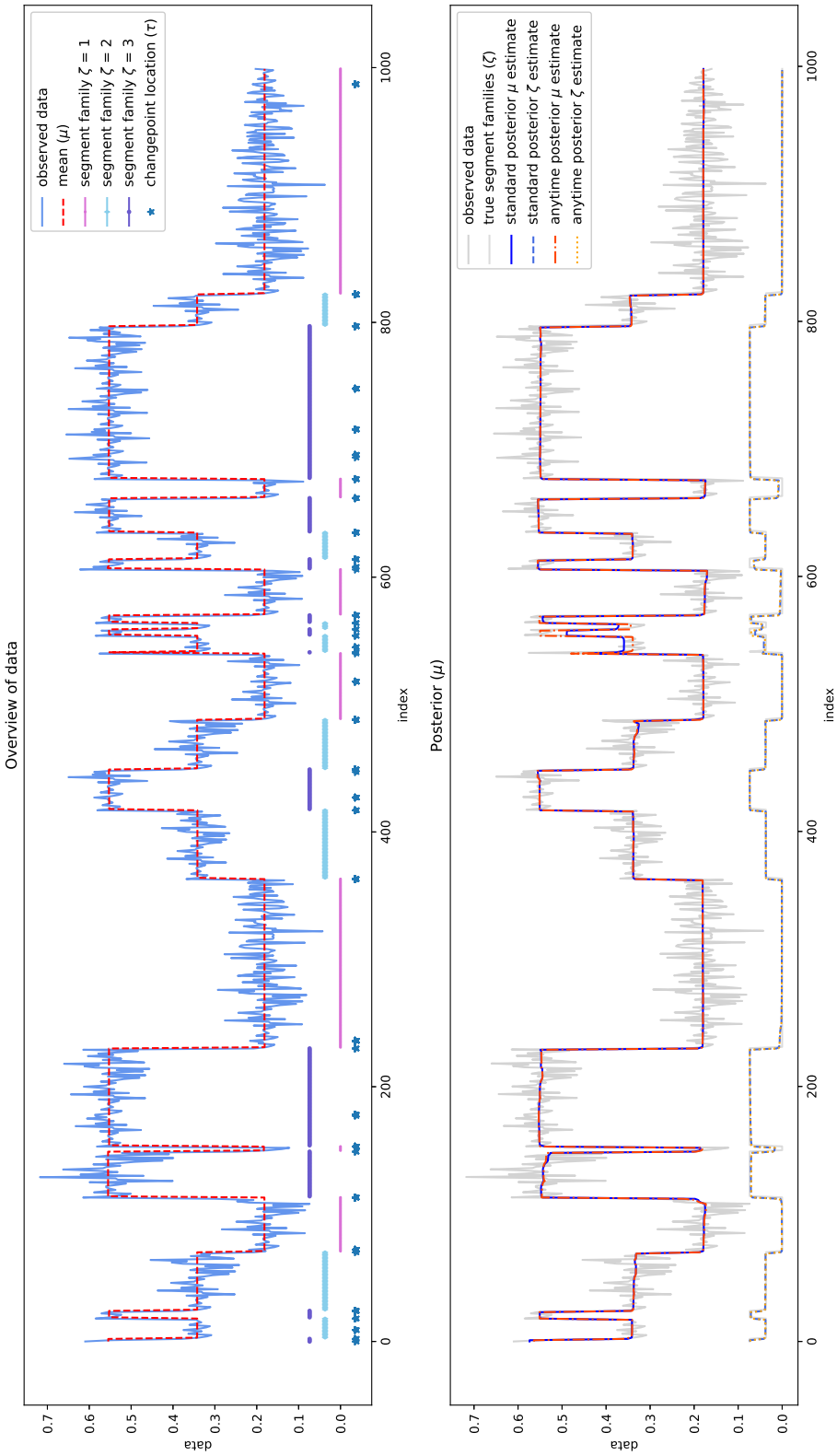


Fig. 5.2 **Top plot**: Observed data (*light blue line*) simulated according to the changepoint model from Fearnhead and Vasileiou [2009]. The true mean μ parameter is displayed (*dashed red line*), as well as the true changepoint locations τ (*dark blue stars*) and the true families ζ to which each segment of the data belongs (*pink line* for $\zeta = 1$, *turquoise line* for $\zeta = 2$, *purple line* for $\zeta = 3$). **Bottom plot**: Estimated posterior mean μ for the standard `RJMCMC-SMC` (*solid blue line*) and Anytime `RJMCMC-ASMC` (*dot-dashed orange line*) algorithms and estimated posterior segment families ζ (*dashed blue line* for `RJMCMC-SMC`, *dotted orange line* for `RJMCMC-ASMC`). Both algorithms are able to recover accurate posterior estimates.

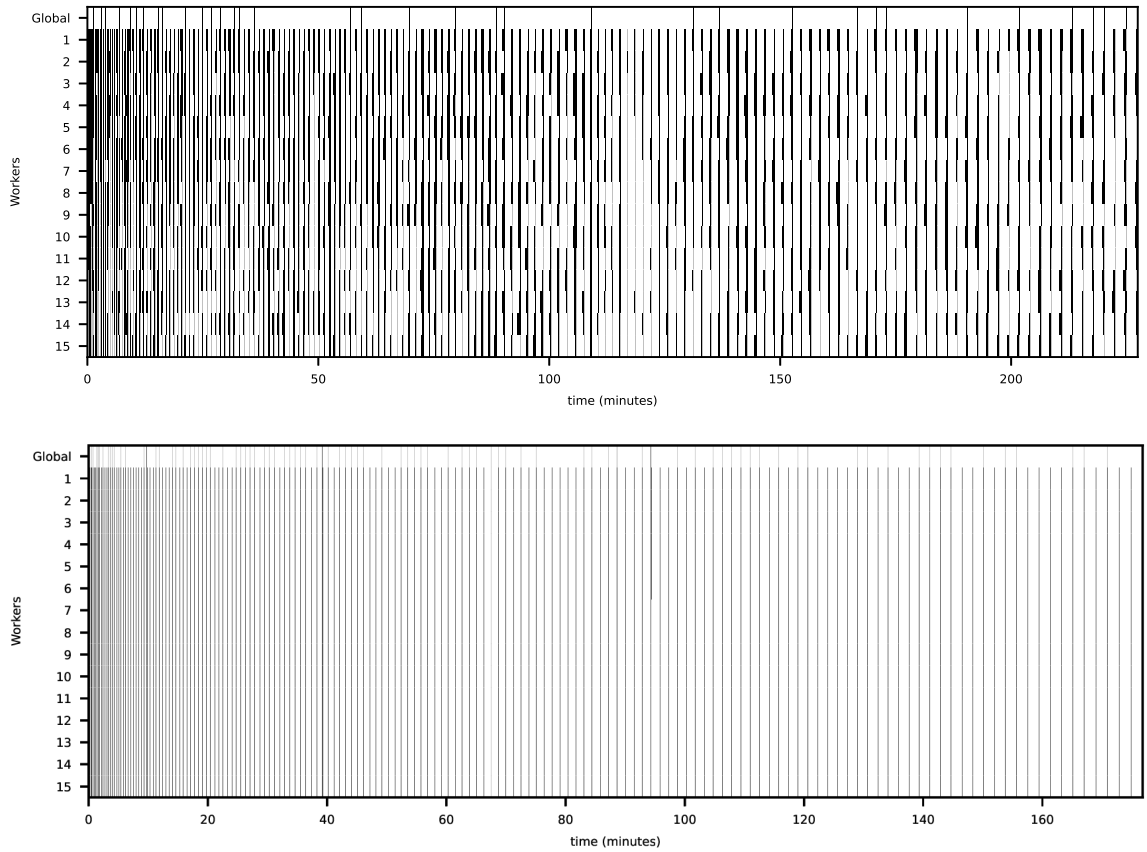


Fig. 5.3 Compute profiles for runs of the standard `RJMCMC-SMC` (top plot) and Anytime `RJMCMC-ASMC` (bottom plot) algorithms showing working time (*white*) and idle time (*black*) on each of 15 workers (corresponding to each row). The `RJMCMC-SMC` (top) displays visible idle times throughout its run, which are greatly reduced once anytime moves are introduced (bottom). As a result, the `RJMCMC-ASMC` algorithm is able to save nearly an hour of computation time.

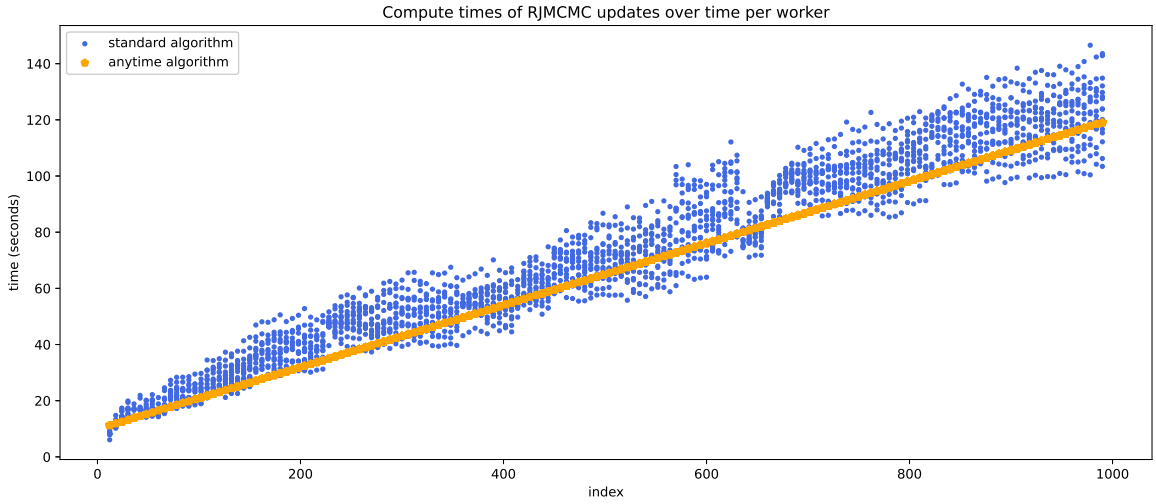


Fig. 5.4 Compute times of RJ-MCMC moves for each worker at each step of the standard `RJMCMC-SMC` (*blue*) and Anytime `RJMCMC-ASMC` (*orange*) algorithms. Each dot corresponds to the time taken by one of 15 workers to complete its set of RJ-MCMC moves. While overall the compute times follow a similar increasing trend for both algorithms, a real-time budget forced all workers to take the same time to complete their RJ-MCMC moves for the `RJMCMC-ASMC` algorithm, while there is significant variation in the compute times of the workers for the `RJMCMC-SMC` algorithm.

5.7 Discussion

In this chapter, we presented a general method for multiple changepoint inference using RJ-MCMC updates within an SMC sampler that makes use of the Anytime framework in a parallel computing environment. We illustrated how it can be implemented on a complex changepoint model with some expensive update moves – for instance, the use of forward-filtering backward simulation for a finite state space model (F-FFBSi) in Section 5.6.3.3 to sample the segment families ($\mathfrak{M}_\zeta$ update) – and demonstrated that the addition of the Anytime framework minimises idling and makes the algorithm faster and more efficient. Previously, Anytime SMC samplers had been introduced in Murray et al. [2016b], as well as SMC samplers which employ RJ-MCMC for changepoint inference in Del Moral et al. [2006], and RJ-MCMC algorithms for similar, complex applications in Boys and Henderson [2004]. In this chapter, all these elements were combined into a single algorithm for the first time. Additionally, while the illustrative model of this chapter was considered in Fearnhead and Vasileiou [2009]; Yıldırım [2012], they performed parameter inference via filtering recursions instead of RJ-MCMC. As a result, the RJ-MCMC updates for this model were derived for the fist time in this chapter.

It is nonetheless important to note that the expensive $\mathfrak{M}_\zeta$ update in the featured model

could have been only performed once the SMC sampler completed its pass through the whole data, in a similar fashion to the latent parameters (μ, σ^2) update. Indeed, additional experiments showed that not updating ζ every time RJ-MCMC updates were performed did not significantly negatively affect the algorithm's ability to correctly locate the changepoints. However, the algorithm construction was kept that way in order to properly demonstrate the benefits of the Anytime framework for a changepoint model where expensive RJ-MCMC updates are unavoidable. For example, in Quan et al. [2019], the data are distributed according to a Student t-distribution (Prince [2012]) with latent parameters (μ, σ^2, v) instead of a Gaussian, which means that it is impossible to collapse out the latent parameters, as the t-distribution does not belong to the exponential family. As a result, letting m_t be the number of changepoints at step t of the SMC sample, the parameters $(\mu_{1:m_t+1}, \sigma^2_{1:m_t+1}, v_{1:m_t+1})$ must all be updated sequentially for all segments via Metropolis-Hastings at each iteration of the RJ-MCMC updates. The case study in Quan et al. [2019] however only considers a single changepoint, so would only moderately benefit from the Anytime framework. Additionally, we did not estimate the hyperparameters of the featured model, as their estimation is not affected by Anytime moves being present or not. They can, however, be straightforwardly estimated by setting hyperpriors and updating them via MCMC (Boys and Henderson [2004]; Wyse and Friel [2010]). If an expectation-maximization (EM) approach is preferred, this has been covered in detail in Yıldırım et al. [2013].

To summarise, this chapter presented new insights and a general method to performing changepoint inference using RJ-MCMC within an Anytime SMC sampler, and can serve as a guide for dealing with any particularly complex changepoint models in the future, such as models in which the compute times of RJ-MCMC moves exhibit a heavy-tail behaviour.

Appendix 5.A Marginal likelihood

To obtain evidence function $R(s,t,k)$ of observations $y_{s:t}$ given in Equation 5.13, conditional on them belonging to the same family $\zeta_i = k$ (i.e. between change points τ_{i-1} and τ_i), it suffices to integrate out the latent parameters μ_i and σ_i^2 as follows:

$$
\begin{aligned}
R(t,s,k) &= \int_{\sigma_i^2}\int_{\mu_i} p(\mu_i|\sigma_i^2,\zeta_i=k)p(\sigma_i^2)\prod_{j=s}^{t} f(y_j|\mu_i,\sigma_i^2)\mathrm{d}\mu_i\mathrm{d}\sigma_i^2 \\
&= \int_{\sigma_i^2}\int_{\mu_i} (2\pi\sigma_i^2)^{-\frac{l}{2}}\exp\left[-\frac{\sum_{j=s}^{t}(y_j-\mu_i)^2}{2\sigma_i^2}\right] \\
&\left(2\pi\frac{\sigma_i^2}{\delta_k}\right)^{-\frac{1}{2}}\exp\left[-\frac{\delta_k(\mu_i-\xi_k)^2}{2\sigma_i^2}\right]\frac{\gamma^\nu}{\Gamma(\nu)}(\sigma_i^2)^{-(\nu+1)}\exp\left[-\frac{\gamma}{\sigma_i^2}\right]\mathrm{d}\mu_i\mathrm{d}\sigma_i^2 \\
&= (2\pi)^{-\frac{l}{2}}\sqrt{\delta_k}\frac{\gamma^\nu}{\Gamma(\nu)}\int_{\sigma_i^2}(\sigma_i^2)^{-(\nu+\frac{l}{2}+1)}\exp\left[-\frac{\gamma}{\sigma_i^2}\right] \\
&\int_{\mu_i}\left(2\pi\sigma_i^2\right)^{-\frac{1}{2}}\exp\left[-\frac{(l+\delta_k)\mu_i^2-2(B_1+\delta_k\xi_k)\mu_i+B_2+\delta_k\xi_k^2}{2\sigma_i^2}\right]\mathrm{d}\mu_i\mathrm{d}\sigma_i^2 \\
&= (2\pi)^{-\frac{l}{2}}\sqrt{\frac{\delta_k}{l+\delta_k}}\frac{\gamma^\nu}{\Gamma(\nu)}\int_{\sigma_i^2}(\sigma_i^2)^{-(\nu+\frac{l}{2}+1)}\exp\left[-\frac{2\gamma+B_2+\delta_k\xi_k^2-\frac{(B_1+\delta_k\xi_k)^2}{l+\delta_k}}{2\sigma_i^2}\right] \\
&\times \underbrace{\int_{\mu_i}\left(2\pi\frac{\sigma_i^2}{l+\delta_k}\right)^{-\frac{1}{2}}\exp\left[-\frac{l+\delta_k}{2\sigma_i^2}\left(\mu_i-\frac{B_1+\delta_k\xi_k}{l+\delta_k}\right)^2\right]\mathrm{d}\mu_i\mathrm{d}\sigma_i^2}_{\text{integrates to 1 w.r.t }\mu_i} \\
&= (2\pi)^{-\frac{l}{2}}\sqrt{\frac{\delta_k}{\delta_k'}}\frac{\gamma^\nu}{\Gamma(\nu)}\frac{\Gamma(\nu')}{\gamma'^{\nu'}}\underbrace{\int_{\sigma_i^2}\frac{\gamma'^{\nu'}}{\Gamma(\nu')}(\sigma_i^2)^{-(\nu'+1)}\exp\left[-\frac{\gamma'}{\sigma_i^2}\right]\mathrm{d}\sigma_i^2}_{\text{integrates to 1 w.r.t }\sigma_i^2} \\
&= \pi^{-\frac{l}{2}}\sqrt{\frac{\delta_k}{\delta_k'}}\frac{(2\gamma)^\nu}{(2\gamma')^{\nu'}}\frac{\Gamma(\nu')}{\Gamma(\nu)},
\end{aligned}
$$

where for simplicity we set $l := s-t+1$, $B_1 := \sum_{j=s}^{t} y_j$, $B_2 := \sum_{j=s}^{t} y_j^2$, and the posterior parameters are given by $\delta_k' = l+\delta_k$, $\nu' = \nu+\frac{l}{2}$ and $\gamma' = \gamma+\frac{1}{2}\left(B_2+\delta_k\xi_k^2-\frac{(B_1+\delta_k\xi_k)^2}{\delta_k'}\right)$.

Chapter 6

Particle Smoothing for Parameter Inference on Dynamic Single Molecules with Stochastic Trajectories

6.1 Chapter overview

Parameter inference is a major aspect of single-molecule microscopy. It allows for the estimation of the parameters that best fit the models describing the behaviours of single molecules in a cellular environment, which emit photons onto a detector. In this chapter, we investigate parameter inference for molecules whose trajectories are described by stochastic differential equations (SDEs), in which the drift and diffusion are parameters of interest. However, the photon detection process makes parameter inference difficult. Indeed, not only are both the time and location at which photons arrive on the detector random but according to optical diffraction theory, the photon detection locations are distributed according to complex profiles such as the Airy profile and the Born and Wolf model. In the past, a Gaussian approximation has been employed to simplify the problem, but this is not always accurate.

Another important aspect of parameter inference is the need to assess the quality of parameter estimates. The variance of the estimates is bounded below by the Cramér-Rao lower bound (CRLB), which is the inverse of the Fisher information matrix (FIM). While it is possible to obtain analytic expressions for the FIM for a static molecule (Ober et al. [2020a]), evaluating the FIM for stochastically moving molecules is difficult. Vahid et al. [2020] were able to obtain an analytical solution, but only for Gaussian measurements and for a specific set of observed photon detection times.

In this chapter, we address the problem of estimating model parameters and characterising

their precision via the CRLB, in order to bring new insights into parameter inference for non-static molecules with general motion and observation models. We also consider the separation distance estimation problem for two moving molecules, and similarly characterise the precision limits for estimating the separation distance between two molecules with stochastic trajectories, thus allowing for the generalisation of results in Ram et al. [2013]. To achieve all this, we discretise the observation interval while still taking into account the random photon detection times through the introduction of missing observations. This allows us to formulate the problem as a straightforward state space model. From this, standard particle filtering and smoothing algorithms can be employed for parameter inference. Most importantly, we are able to estimate the FIM for the Airy profile and Born and Wolf model, which could not be done before. This is achieved by estimating the score and observed information matrix (OIM) via particle smoothing.

The work done in this chapter is the subject of a journal paper Marie d'Avigneau et al. [2021] written in collaboration with co-authors Dr Sumeetpal Singh and Prof Raimund Ober and submitted for publication.

6.2 Introduction

In recent years, *single-molecule microscopy* has become a powerful tool in cell biology (Saxton [1997]; Saxton and Jacobson [1997]). Indeed, it has allowed significant insight to be gained into the behaviour of single molecules in cellular environments using *fluorescence microscopy*, which was not available in previous studies where molecules were observed in bulk (Dange et al. [2008]; Ober et al. [2004a,b]; Ram et al. [2006b]; Yu et al. [2006]). Single-molecule fluorescence microscopy (see Moerner and Fromm [2003]; Shashkova and Leake [2017] for reviews) consists of using a suitable fluorophore to label the molecule of interest, exciting said fluorophore with a specific light source and capturing the fluorescence emitted by the molecule through a microscope system onto a detector during a fixed acquisition time.

One of the main aims of the analysis of single-molecule microscopy data is to infer on the parameters of interest relating to the model describing the movement of a molecule, as well as on its state. In the case of single-molecule tracking, as in Deschout et al. [2014]; Small and Stahlheber [2014], a natural example of states that can be estimated over time are its positional coordinates. This can be done using particle or Kalman filtering methods (Ashley and Andersson [2015]; Calderon [2016]; Vahid et al. [2020]). The motion of an object in a cellular environment is affected by a multitude of deterministic, as well as random factors (Briane et al. [2018]). In many applications, such as Calderon [2016]; Vahid et al. [2020], the trajectory of a molecule is modelled by *stochastic differential equations* (SDEs)

(see Oksendal [2013] for an introduction). Therefore, another example of parameters of interest − also known as *hyperparameters* − are the drift and diffusion of the SDE describing the motion of the molecule. These parameters are generally estimated using *maximum likelihood* (ML) estimation techniques (Calderon [2016]; Relich et al. [2016]; Vahid et al. [2020]). In Ashley and Andersson [2015], the ML estimates of the hyperparameters are obtained using expectation-maximization (EM) methods within a particle filter, or sequential Monte Carlo algorithm in what is known as the SMC-EM algorithm (Cappé et al. [2006]; Kantas et al. [2015], and Del Moral et al. [2010] for an online implementation). The particle filtering and smoothing methods employed in Ashley and Andersson [2015] to approximate the expectation step of the EM algorithm are sequential importance resampling (SIR) (Doucet and Johansen [2009]) coupled with the forward-filtering backward-smoothing (FFBS) algorithm (Lindsten and Schön [2013]). In this chapter, we describe a more general and efficient particle smoothing approach to obtain approximations of expectations of interest, including for parameter estimation purposes.

Throughout this chapter, we consider the ideal, *fundamental data model* (Ober et al. [2004b]; Ram et al. [2006b]), which is crucial in that it provides accessible lower bounds for the limits of accuracy of more realistic practical models, where factors such as pixelisation and readout noise come into play and make inference more challenging. According to this model, a process which affects parameter inference is the detection process of the photons emitted by the molecule. The detection process is intrinsically random both in time and location. Indeed, while many methods such as Calderon [2016]; Calderon and Bloom [2015]; Calderon et al. [2013] have assumed that the arrival times of the photons on the detector were uniformly distributed, Ober et al. [2004b]; Ram et al. [2006b] suggest that the arrival times of photons follow a Poisson process instead. As for the arrival location of these photons on the detector, a wide range of measurement models exist − corresponding to the various types of detector. The typical measurement model used for an in-focus source is the *Airy profile* (Chao et al. [2016]; Vahid et al. [2020]). If the molecule is out of focus, 3D models are generally used instead, such as the *Born and Wolf model* (Born and Wolf [2013]). Often, these models make parameter inference difficult, and researchers have often opted for a Gaussian approximation to these models, such as in Berglund [2010]; Michalet and Berglund [2012]; Relich et al. [2016]. However, Vahid et al. [2020] argue that in practice, assuming Gaussian distributed photon locations on the detector is not an accurate approximation of the underlying model.

In addition to making sure any estimate of the parameters of interest is unbiased, it is vital that they be accurate, as otherwise, the biological process being investigated may be misrepresented (Liu et al. [2013]; Michalet [2010]). In estimation theory, the *Cramér-Rao*

lower bound (CRLB) derived by Cramér [1999]; Darmois [1945]; Fréchet [1943]; Rao [1992] establishes a lower bound on the variance of unbiased estimates, and is therefore often used as a benchmark for the quality of a given estimator. The CRLB is equal to the inverse of the *Fisher information matrix* (FIM) (see Ly et al. [2017] for a tutorial). The mathematical framework for obtaining the Fisher information in the context of single-molecule microscopy was developed in Ober et al. [2004b]; Ram et al. [2006b] and further explored in Chao et al. [2016], in the context of a stationary molecule, where expressions for the FIM were derived for various detector types. In Vahid et al. [2020], a method was developed to obtain the FIM for a moving molecule whose trajectory is described by a linear SDE. For a 2D Gaussian approximation of the photon detection process, the authors take advantage of the Kalman filter formulae to obtain an analytical form for the FIM for a specific set of photon detection times. However, if the Airy profile is used instead, the computational cost of performing numerical integration becomes prohibitive for more than a single photon. Among other things, we build on Vahid et al. [2020] and develop a framework which enables the estimation of the FIM for the hyperparameters of the Airy and Born and Wolf profiles.

In this chapter, we develop a general framework to obtain particle approximations of expectations of interest, including for complex photon detection models such as the Airy profile and Born and Wolf model. The ability to approximate these expectations is important for two things: estimating the *score* and *observed information matrix* (OIM) for the hyperparameters of the latent process describing the molecule trajectory, and obtaining ML estimates of said hyperparameters. Access to the score and/or OIM is vital in order to be able to estimate the FIM. To achieve this, the observation interval is first discretised and the problem reformulated as a discrete-time state space model, which takes into account the random arrival times of photons on the detector. Then, the *auxiliary particle filter* (APF) from Pitt and Shephard [1999] is employed in conjunction with *forward smoothing* methods (Del Moral et al. [2010]; Olsson et al. [2017]) to obtain particle approximations of the expectations of interest. Ashley and Andersson [2015] similarly employed time discretisation of the observation interval but did not attempt to estimate the FIM for hyperparameters.

The numerical experiments in this chapter consist of applying the methodology to estimate the *limit of accuracy*, i.e. the square root of the CRLB, for the 2D Gaussian, Airy profiles and Born and Wolf model by using estimates of the score and OIM obtained by forward smoothing. This is repeated for various expected mean photon counts in order to verify that for molecules with stochastic trajectories, the limit of accuracy exhibits an inverse square root decay with respect to mean photon count, i.e. the uncertainty of the hyperparameter estimates decreases as the expected number of photons increases. This has already been proven for a static molecule (Ober et al. [2020a]). Then, the methodology is applied to the

problem of assessing the limit of accuracy for the mean separation distance between two closely spaced molecules.

This chapter is structured as follows. In Section 6.3, the model is presented, including the molecule trajectory, described by a SDE, and the photon detection time and location processes. Section 6.4 formulates the model as a discrete-time state space model with a discretised observation interval. Then, Section 6.5 establishes the main parameter inference aims and methods, which consist of particle filtering and smoothing of additive functionals in order to estimate the score and OIM for hyperparameters, as well as their ML estimates, and methods to estimate the FIM from the score and observed information. Numerical experiments are run in Section 6.6 to first estimate the limit of accuracy for the drift and diffusion coefficients of the SDE for all photon detection profiles and then estimate the limit of accuracy for the separation distance between two dynamic molecules. Section 6.7 provides concluding remarks and an overview of future work.

6.3 Model specification

For the purpose of this chapter, a basic optical system is considered, also known in Chao et al. [2016]; Vahid et al. [2020] as the *fundamental data model*. See Figure 6.1 for an overview of the optical system. Under the fundamental model, we assume that the photons are observed under ideal conditions, in which the detector $\mathscr{Y} = \mathbb{R}^2$ is non-pixelated. This model does not describe image data obtained from actual microscopy experiments the way more realistic, or *practical* models do. However, the fundamental model is crucial, in that it offers an obtainable lower bound to the CRLB of parameters of the more realistic practical model, which is much more difficult to obtain. In this section, the various aspects of the model are described. These include the true molecule trajectory, occurring in the object space, the photon detection locations in the image space, and the times at which photons arrive on the detector.

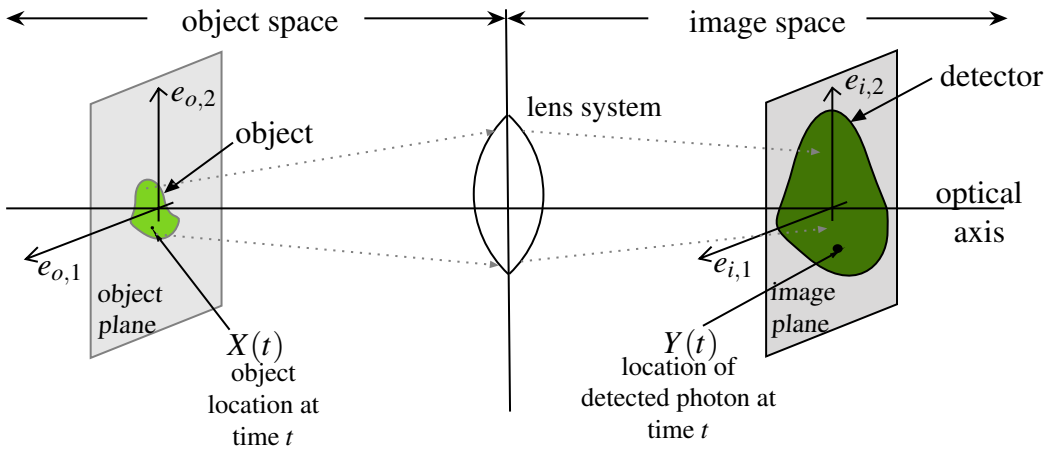


Fig. 6.1 Illustration of an optical microscope. At time $t \geq t_0$, the molecule is located at $X(t)$ in the object space and might be moving along the object plane. If the molecule is out of focus, it will instead move along a plane parallel to the object plane but displaced along the optical axis. The molecule emits photons through the lens system into the image space and its image is acquired on the planar detector $\mathscr{Y}$ located on the image plane. The location of the detected photons at time t is denoted by $Y(t)$.

6.3.1 Molecule trajectory

For notational simplicity, let $X_t := X(t) \in \mathbb{R}^d$ denote the true, d-dimensional location of the molecule at time t. Given hyperparameters θ, let $f^{\theta}_{s,t}(x_t|x_s)$ denote the probability density function of X_t given the previous location X_s. Assume that the molecule trajectory $(X_t)_{t_0 \leq t \leq T}$ follows a linear *stochastic differential equation* (SDE)

$$\mathrm{d}X_t = b(t, X_t)\mathrm{d}t + \sigma(t, X_t)\mathrm{d}B_t, \tag{6.1}$$

where $b(t, X_t) := b_0 + b(t)X_t$ and $\sigma(t, X_t) := \sigma(t)$ represent the *drift* and *diffusion* coefficients, respectively, b_0 is the zero order drift coefficient, and $(\mathrm{d}B_t)_{t_0 \leq t \leq T}$ is a Wiener process with $\mathbb{E}\left[\mathrm{d}B_t \mathrm{d}B_t^{\intercal}\right] = \mathbb{I}_{d \times d}$. According to Evans [2012]; Jazwinski [2007] the solution to the SDE in Equation 6.1 at discrete time points $t_0 < t_1 < \ldots$ is given by

$$X_{t_{i+1}} = \Phi(t_i, t_{i+1})X_{t_i} + a(t_i, t_{i+1}) + W_g(t_i, t_{i+1}), \tag{6.2}$$

where the *fundamental matrix function* $\Phi \in \mathbb{R}^{d \times d}$ satisfies the following for all $s, t, u \geq t_0$

$$\begin{aligned} \frac{\mathrm{d}\Phi(s,t)}{\mathrm{d}t} &= b(t)\Phi(s,t), \\ \Phi(t,t) = \mathbb{I}_{d \times d}, &\qquad \Phi(s,t)\Phi(t,u) = \Phi(s,u), \end{aligned} \tag{6.3}$$

the vector $a(t_i, t_{i+1}) \in \mathbb{R}^d$ is given by

$$a(t_i, t_{i+1}) = \int_{t_i}^{t_{i+1}} b_0 \Phi(t_i, t) \mathrm{d}t,$$

and finally the process $\left(W_g(t_i, t_{i+1}) = \int_{t_i}^{t_{i+1}} \Phi(t_i, t)\sigma(t)\mathrm{d}B_t\right)_{i=1}^{\infty}$ is a white noise sequence with mean zero and covariance

$$R(t_i, t_{i+1}) = \int_{t_i}^{t_{i+1}} \Phi(t_i, t)\sigma(t)\sigma^{\intercal}(t)\Phi^{\intercal}(t_i, t)\mathrm{d}t. \tag{6.4}$$

Therefore, the transition density $f^{\theta}_{t_{i+1},t_i}(x'|x)$ can be expressed as a Gaussian with mean $\mu(x, t_i, t_{i+1}) = \Phi(t_i, t_{i+1})x + a(t_i, t_{i+1})$ and covariance $R(t_i, t_{i+1})$, i.e.

$$X_{t_{i+1}}|(X_{t_i} = x) \sim \mathcal{N}\left(\mu(x, t_i, t_{i+1}), R(t_i, t_{i+1})\right). \tag{6.5}$$

Example 6.1. Let the trajectory of a molecule be given by the following SDE

$$\mathrm{d}X_t = b\mathbb{I}_{d\times d}X_t\mathrm{d}t + \sqrt{2\sigma}\mathrm{d}B_t, \tag{6.6}$$

where in the drift term $b \in \mathbb{R}$, in the diffusion term $\sigma > 0$, and $(\mathrm{d}B_t)_{t_0\le t\le T}$ is a Wiener process and let $\theta = (\sigma, b)$. Assuming the time points $t_0, t_1, \ldots$ are equidistant, i.e. $t_{i+1} - t_i = \Delta$ for all $i = 0, 1, \ldots$, let the fundamental matrix $\Phi_\Delta := \varphi_\Delta^\theta \mathbb{I}_{d\times d}$ where $\varphi_\Delta^\theta \in \mathbb{R}$ and the covariance matrix $R_\Delta := r_\Delta^\theta \mathbb{I}_{d\times d}$ where $r_\Delta^\theta > 0$. Then, by solving Equation 6.3 and plugging the result into Equation 6.4, it is straightforward to obtain

$$\varphi_\Delta^\theta = \begin{cases} e^{b\Delta} & \text{if } b \neq 0, \\ 1 & \text{if } b = 0, \end{cases} \qquad \text{and} \qquad r_\Delta^\theta = \begin{cases} \frac{\sigma}{b}\left(e^{2b\Delta} - 1\right) & \text{if } b \neq 0, \\ 2\sigma\Delta & \text{if } b = 0. \end{cases}$$

The initial distribution $X_{t_0} \sim \mathcal{N}(x_0, P_0)$ has covariance matrix $P_0 = p_0\mathbb{I}_{d\times d}$ where $p_0 \in \mathbb{R}$.

In a 2-dimensional setting (i.e. $d = 2$), let the drift $b = -10$ s^{-1}, the diffusion $\sigma = 1$ μm^2/s and the initial covariance $p_0 = 10^{-2}$ μm^2 and mean $x_0 = (4.4, 4.4)^\intercal$ μm. By simulating the molecule trajectory for the time interval $[0, 0.1]$ seconds, we obtain the trajectory in Figure 6.2.

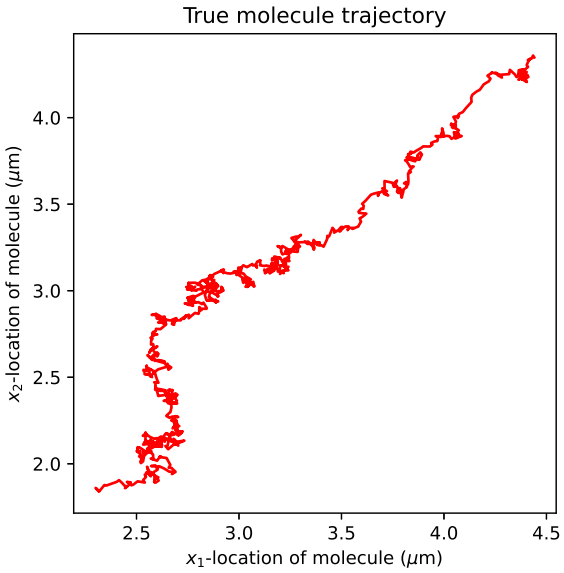


Fig. 6.2 Trajectory of a molecule in the object space with stochastic trajectory described in Equation 6.6 and with diffusion and drift coefficients $\sigma = 1$ μm^2/s and $b = -10$ s^{-1}, respectively. The molecule moves during an interval of $[0, 0.1]$ seconds and its initial location is Gaussian distributed with mean $x_0 = (4.4, 4.4)^\intercal$ μm and covariance $P_0 = 10^{-2}\mathbb{I}_{2\times 2}$ μm^2.

6.3.2 Photon detection locations

The true molecule trajectory cannot be observed directly. Instead, a fluorescence microscope is used: the molecule of interest is labelled using a suitable fluorophore, magnified through a lens system and the photons it emits arrive on a detector $\mathscr{Y} := \mathbb{R}^2$ for a fixed time period (see Figure 6.1). The arrival location of a photon on the detector is random, and using the typical approximation of the optical microscope from Goodman [2005], it can be described as follows. Let $Y \in \mathscr{Y}$ denote the observed location of a detected photon. For an object located at $(x_{0,1}, x_{0,2}, z_0) \in \mathbb{R}^3$ in the object space, its *photon distribution profile* (Ram et al. [2006b]) is given by the density

$$g_\theta(y|x) := \frac{1}{|M|} q_{z_0}\left(M^{-1}y - (x_{0,1}, x_{0,2})^\intercal\right), \quad y \in \mathbb{R}^2, \tag{6.7}$$

where $M \in \mathbb{R}^{2\times 2}$ is an invertible *lateral magnification matrix* and the *image function* $q_{z_0} : \mathbb{R}^2 \to \mathbb{R}$ describes the image of an object in the detector space when that object is located at $(0,0,z_0)$ in the object space. Note that the subscript θ is used in the left-hand side of Equation 6.7 to include dependence on hyperparameters. In this case, the hyperparameter of interest will generally be z_0.

Three types of image functions are considered, and their densities are plotted in Figure 6.3. First of all, according to optical diffraction theory from Born and Wolf [2013], an in-focus point source (i.e. when $z_0 = 0$) will typically generate an image that follows the Airy profile

$$q(x_1, x_2) = \frac{J_1^2\left(\frac{2\pi n_\alpha}{\lambda_e}\sqrt{x_1^2 + x_2^2}\right)}{\pi(x_1^2 + x_2^2)}, \quad (x_1, x_2) \in \mathbb{R}^2, \tag{6.8}$$

where n_α is the numerical aperture of the objective lens, λ_e is the emission wavelength of the molecule and $J_1(\cdot)$ represents the first order Bessel function of the first kind.

Often, to simplify the problem, the 2D Gaussian approximation to the Airy profile has been used instead (see Cheezum et al. [2001]; Stallinga and Rieger [2010]; Thompson et al. [2002]; Zhang et al. [2007])

$$q(x_1, x_2) = \frac{1}{2\pi\sigma_a^2} \exp\left[-\frac{x_1^2 + x_2^2}{2\sigma_a^2}\right], \quad (x_1, x_2) \in \mathbb{R}^2, \tag{6.9}$$

If the point source of interest is out of focus, then a 3D Born and Wolf model, developed by

Born and Wolf [2013], is used instead

$$q_{z_0}(x_1,x_2) = \frac{4\pi n_\alpha^2}{\lambda_e^2}\left|\int_0^1 J_0\left(\frac{2\pi n_\alpha}{\lambda_e}\sqrt{x_1^2+x_2^2}\rho\right)\exp\left(\frac{j\pi n_\alpha^2 z_0}{n_o\lambda_e}\rho^2\right)\rho d\rho\right|^2, \quad (x_1,x_2)\in\mathbb{R}^2, \tag{6.10}$$

where $z_0 \in \mathbb{R}$ is the location of the object on the optical axis, n_o is the refractive index of the objective lens immersion medium and $J_0(\cdot)$ is the zero-th order Bessel function of the first kind. Note that the Airy profile is simply a special case of the Born and Wolf model. Indeed, if the object is in focus, then $z_0 = 0$ on the optical axis and Equations 6.8 and 6.10 coincide.

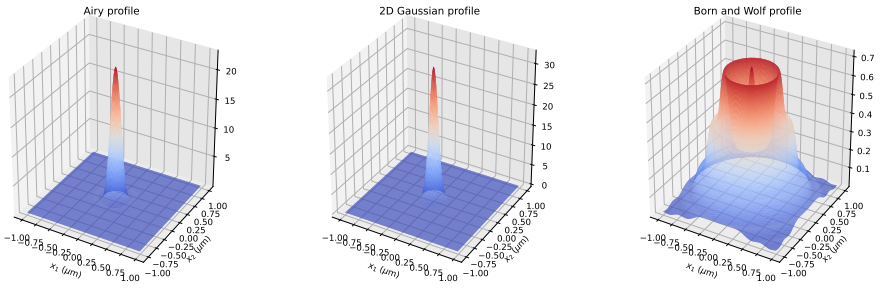


Fig. 6.3 Illustration of the image functions considered. The Airy (*left*) and 2D Gaussian (*middle*) profiles, given in Equations 6.8 and 6.9 respectively, describe an in-focus point source and the Born and Wolf model (*right*) given in Equation 6.10 describes an out of focus point source with optical axis location $z_0 = 1$ μm.

6.3.3 Photon detection times

Just like the photon detection locations, the times at which the photons arrive on the detector $\mathscr{Y}$ are random. More specifically, in Chao et al. [2016]; Vahid et al. [2020], the arrival of the photons on the detector, or *photon detection process*, can be modelled as a Poisson process. Let $N(t)$ be the number of photons detected at time $t \geq t_0$ for initial time $t_0 \in \mathbb{R}$ and let $\lambda(t)$ be the *photon detection rate*, representing the rate at which the photons emitted by the object hit the detector at any given time t. For example, the detection rate of an object that has high photostability will simply be constant, while an exponentially decaying $\lambda(t)$ can indicate that the object image is photobleaching, or fading over time. The arrival times of the photons on the detector $\mathscr{Y}$ are denoted $t_1, t_2, \ldots$ where t_i denotes the arrival time of the i-th photon.

6.3.4 The observed data

Let $n_p = N(T) - N(t_0)$ be the number of photons detected in the interval $[t_0, T]$. We have now established the two aspects of the data that can be observed in a basic optical system during this interval, namely the detection times $t_1, t_2, \ldots, t_{n_p}$ of photons and the location of those detected photons $Y_{t_1}, Y_{t_2}, \ldots, Y_{t_{n_p}}$ on the detector $\mathscr{Y}$. Assume that, conditionally on the current object location X_{t_i}, the location of the i-th detected photon Y_{t_i} at time t_i is independent of the previous locations and time points of the detected photons, i.e. for $x_{t_i} \in \mathscr{X}$,

$$p_\theta(y_{t_i}|x_{t_i}, y_{t_{i-1}}, \ldots, y_{t_0}) = p_\theta(y_{t_i}|x_{t_i}) =: g_\theta(y_{t_i}|x_{t_i}), \quad y_{t_i} \in \mathscr{Y}, \tag{6.11}$$

where the density g_θ is the photon distribution profile from Equation 6.7. This is a reasonable assumption, as at any given time, processes such as photon emission and image formation only depend on the state of the emitting fluorescent molecule at that time, and not on any prior event.

Example 6.2. Let the trajectory of a molecule be given by the SDE in Example 6.1 and simulated using the same parameters and for the same time interval. Let $\mathscr{Y}$ be a non-pixelated detector. Then, let the photon detection rate be constant such that the mean number of photons is 500, and the photon distribution profile be given by Equation 6.7, where the magnification matrix $M = m\mathbb{I}_{2\times 2}$ with $m = 100$. The image functions for the Airy, 2D Gaussian and Born and Wolf profiles are given by Equations 6.8, 6.9 and 6.10 respectively, where $n_\alpha = 1.4$, $\lambda_e = 0.52$ μm, $n_o = 1.515$, $\sigma_a = 7 \times 10^{-2}$ μm and $z_0 = 1$ μm. By simulating the detected photon locations based on the same molecule trajectory and according to these three models, we obtain the observed photon trajectories in Figure 6.4. Note that the parameters of the Airy and 2D Gaussian profiles have been chosen so that the Gaussian profile approximates the Airy profile.

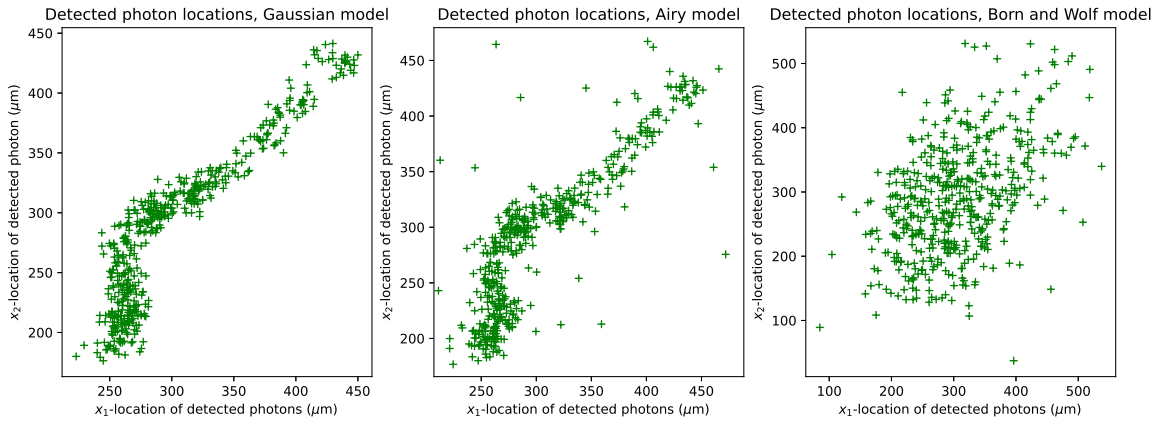


Fig. 6.4 Detected photon locations of a moving molecule with stochastic trajectory for the 2D Gaussian (*left*), Airy (*middle*) profiles and Born and Wolf model (*right*)

6.4 The model as a state space model

It is possible to reformulate this model as a discrete state space model that takes into account the random arrival times of photons. This is achieved by discretising the time interval during which photons are recorded.

6.4.1 Reformulation

For simplicity, assume that the photon detection rate is constant, i.e. $\lambda(t) = \lambda \in [0,1]$ for all $t \geq t_0$. First of all, let $X_t = (x_{t,1}, x_{t,2}) \in \mathscr{X}$ where $\mathscr{X} := \mathbb{R}^2$ denotes the state of the molecule at time $t \geq t_0$, which includes its location $x_{t,1:2}$ on the object plane. The location of the object on the optical axis is assumed to be constant and equal to the initial location parameter, i.e. z_0 for all $t \geq t_0$. The probability of recording an observation, i.e. detecting a photon in the small interval $(t, t+h]$ is

$$\mathbb{P}\left[N(t+h) - N(t) > 0\right] = \lambda h + o(t), \quad \lambda \in [0,1], t \geq t_0.$$

Let t_i denote the arrival time of the i-th photon on a detector $\mathscr{Y}$ for $i = 1, 2, \ldots$ and $Y_{t_i} \in \mathscr{Y}$ be the location of the captured photon on the detector. Assume the location of a detected photon is distributed according to the probability density function

$$Y_{t_i} | \left(X_{t_i} = x\right) \sim g_\theta(\cdot|x), \quad i = 1, 2, \ldots,$$

where g_θ is the photon distribution profile given in Equation 6.7. The recorded data in the time interval $[t_0, T]$, $0 \leq t_0 < T$ comprises of n observations with arrival times $t_0 < t_1 < \ldots <$

$t_{n_p} \leq T$ and photon locations $y_{t_1}, \ldots, y_{t_{n_p}}$. The inference objective is to estimate the trajectory of the molecule $(X_t)_{t_0 \leq t \leq T}$ given data (t_i, y_{t_i}), $i = 1, \ldots, n_p$. As seen in Section 6.3.1, the molecule evolves according to the probability density function

$$X_{t_{i+1}} | (X_{t_i} = x) \sim f^{\theta}_{t_i, t_{i+1}}(\cdot | x), \quad i = 1, 2, \ldots, n_p,$$

where θ denotes the model parameters and $f^{\theta}_{s,t}$ for $t > s \geq t_0$ is the homogeneous continuous time Markov transition density given by the 2D version of the Gaussian distribution in Equation 6.5 for $d = 2$.

6.4.1.1 Non-constant photon detection rate

If the photon detection rate $\lambda(t)$ is not assumed to be constant, then we redefine the state of an object at time $t \geq t_0$ as $X_t = (x_{t,1}, x_{t,2}, \lambda_t) \in \mathscr{X}$ where $\mathscr{X} := \mathbb{R}^2 \times [0,1]$. The state at time t now includes the location of the molecule $(x_{t,1}, x_{t,2})$ as well as the probability λ_t of detecting a photon it emits. The Markov transition density $p_\theta(x'|x)$ can be defined as follows

$$p_\theta(x_{t_{i+1}} | x_{t_i}) = f^{\theta}_{t_i, t_{i+1}}(x_{t_{i+1},1:2} | x_{t_i,1:2}) l_\theta(\lambda_{t_{i+1}} | \lambda_{t_i}), \quad x_{t_{i+1}}, x_{t_i} \in \mathscr{X}$$

where t_i and t_{i+1} denote the arrival times of the i-th and $(i+1)$-th photons, respectively, $f^{\theta}_{t_i, t_{i+1}}$ is the Markov transition density for the object location defined above and l_θ is the Markov transition density for the photon detection rate.

6.4.2 Time discretisation

Let $(t_1, y_{t_1}), \ldots, (t_{n_p}, y_{t_{n_p}})$ be a realisation of the photon arrival times and locations observed in the time interval $[t_0, \ldots T]$. Setting $t_0 := 0$ for convenience, we adopt a discrete-time formulation where the time interval $[0, T]$ is divided into segments of length Δ. Let $x_k \in \mathscr{X}$ denote the state of the molecule at time $t = (k-1)\Delta$ where $k = 1, \ldots, n$ for $n := \lceil T/\Delta \rceil$. We assume the discretisation is fine enough so that an interval $(k\Delta, k\Delta + \Delta]$ contains at most one arrival time t_i. Then, for $k = 1, \ldots, n$, let

$$y_k = \begin{cases} \emptyset, & \text{if} \quad t_i \notin (k\Delta - \Delta, k\Delta], \quad \forall i = 0, 1, \ldots, n_p, \\ y_{t_i}, & \text{if} \quad t_i \in (k\Delta - \Delta, k\Delta], \end{cases}$$

where $y_{t_i} \in \mathscr{Y}$ denotes the location of the i-th detected photon on the detector $\mathscr{Y}$. The vector y_k is assigned $\emptyset$ to indicate the absence of an observation in the corresponding interval. If

$x = (x_1, x_2, \lambda) \in \mathscr{X}$, let

$$G_k^\theta(x) = \begin{cases} 1 - \Delta\lambda, & \text{if } y_k = \emptyset, \\ \lambda g_\theta(y_{t_i}|x_{1:2}), & \text{if } y_k = y_{t_i}, \end{cases}$$

then $G_k^\theta(x)$ is the so called potential function.

It is also straightforward to establish that for $k = 1\ldots, n$ the probability density function of X_{k+1} given the previous state X_k is $f_\Delta^\theta(x_{k+1}|x_k) := f_{k\Delta,k\Delta+\Delta}^\theta(x_{k+1}|x_k)$ from Equation 6.5, thus transforming Equation 6.2 into

$$X_{k+1} = \Phi_\Delta X_k + a_\Delta + W_x, \quad W_x \sim \mathscr{N}(0, R_\Delta),$$

where $\Phi_\Delta = \Phi(k\Delta, k\Delta + \Delta)$ is now constant and similarly for a_Δ and R_Δ.

To summarise, $(X_k)_{k=1}^\infty$ and $(Y_k)_{k=1}^\infty$ are $\mathscr{X}$- and $\mathscr{Y}$-valued stochastic processes where the molecule trajectory in the object space $(X_k)_{k=1}^\infty$ corresponds to the unobserved latent Markov process with Markov transition density $f_\Delta^\theta(x'|x)$ and initial density $\nu_\theta(x)$, and the photon detection locations (or lack of) $(Y_k)_{k=1}^\infty$ represent the observed process with conditional density or potential function $G_k^\theta(x)$, i.e.

$$X_1 \sim \nu_\theta(\cdot), \quad X_{k+1}|(X_k = x) \sim f_\Delta^\theta(\cdot|x), \tag{6.12}$$

$$Y_k|(X_k = x) \sim G_k^\theta(x), \quad k = 1, 2, \ldots. \tag{6.13}$$

Note that if the object is static, so that the drift and diffusion coefficient in Equation 6.1 are zero, the model simplifies from a state space model to a basic inference problem with independent observations. The observed process is still described by Equation 6.13 but the location of the object x_0 becomes part of the hyperparameters.

6.5 Parameter inference

6.5.1 Inference aim

Now that we have formulated the problem in Equations 6.12 and 6.13 as a state space model, the first aim is going to be to estimate the posterior probability density function of $X_{1:n} := \{X_1, \ldots, X_n\}$, $n \in \mathbb{N}$, given the observations $Y_{1:n}$, also known as the *joint smoothing*

distribution, which is given by

$$p_\theta(x_{1:n}|y_{1:n}) = \frac{p_\theta(x_{1:n}, y_{1:n})}{p_\theta(y_{1:n})}, \tag{6.14}$$

where the numerator represents the *joint density*

$$p_\theta(x_{1:n}, y_{1:n}) = \nu_\theta(x_1) \prod_{k=2}^{n} f_\Delta^\theta(x_k|x_{k-1}) \prod_{k=1}^{n} G_k^\theta(x_k), \tag{6.15}$$

where $\nu_\theta(x_1)$ is the initial distribution of X_1, and the denominator represents the *marginal likelihood* of the observed data

$$p_\theta(y_{1:n}) = \int_{\mathscr{X}^n} p_\theta(x_{1:n}, y_{1:n}) \mathrm{d}x_{1:n}. \tag{6.16}$$

Estimating $p_\theta(x_{1:n}|y_{1:n})$ is what allows the molecule to be tracked and is done using a particle filter. The second aim is to obtain particle approximations of smoothed additive functionals, which in turn will allow for ML estimation of the hyperparameters θ via gradient ascent and Expectation-Maximization (EM), as well as the estimation of their score and OIM. Finally, the third aim is to use the estimates of the score and OIM of the hyperparameters to obtain an approximation of their FIM.

6.5.2 Tracking the molecule using a particle filter

6.5.2.1 The auxiliary particle filter

The particle approximation of the marginal posterior of $X_1, \ldots, X_n$ defined in Equation 6.14 is given by

$$\hat{p}(x_{1:n}|y_{1:n}) = \sum_{i=1}^{N} \omega_n^{(i)} \delta_{X_{1:n}^{(i)}}(x_{1:n}),$$

where $X_{1:n}^{(1:N)}$ are the particles and $\omega_n^{(1:N)}$ their corresponding normalised importance weights, i.e. $\sum_{i=1}^{N} \omega_n^{(i)} = 1$. To obtain this particle approximation, we employ sequential Monte Carlo (SMC) methods in the form of the *auxiliary particle filter* (APF), developed by Pitt and Shephard [1999] and extensively used in the literature, e.g. by Carpenter et al. [1999]; Fearnhead et al. [2008]; Papaspiliopoulos [2010]; Poyiadjis et al. [2011]. Define the *auxiliary density*

$$h_\theta(x_k, y_k|x_{k-1}) = h_\theta(x_k|y_k, x_{k-1}) h_\theta(y_k|x_{k-1}) \tag{6.17}$$

as a non-negative function where it is easy to sample from the *proposal distribution* $h_\theta(x_k|y_k,x_{k-1})$ and it is easy to evaluate $h_\theta(y_k|x_{k-1})$ for any $x_{k-1} \in \mathscr{X}$, $y_k \in \mathscr{Y}$. The particle filter then proceeds as in Algorithm 3.9. The APF is chosen here for its generality, but is only one of the possible particle filters that could be applied to track the particles.

6.5.2.2 Choices of auxiliary density

When the exact form of the posterior density $p_\theta(x_k|y_k,x_{k-1})$ of the state X_k given the observation Y_k and previous state X_{k-1} is available, the suggested optimal choice of auxiliary density in Pitt and Shephard [1999] is

$$h_\theta(x_k|y_k,x_{k-1}) = p_\theta(x_k|y_k,x_{k-1}), \quad \text{and} \quad h_\theta(y_k|x_{k-1}) = p_\theta(y_k|x_{k-1}),$$

where the proposal distribution $p_\theta(x_k|y_k,x_{k-1})$, being the target density, is optimal. This is the case when the 2D Gaussian approximation to the Airy profile (Equation 6.9) is used to describe the photon distribution. The auxiliary density can then be derived exactly and the components of Equation 6.17 are

$$h^\theta_\Delta(x_k|y_k,x_{k-1}) := \begin{cases} \mathcal{N}(\Phi_\Delta x_{k-1} + a_\Delta, R_\Delta), & \text{if } y_k = \emptyset, \\ \mathcal{N}(\mu_\Delta, \Sigma_\Delta), & \text{otherwise,} \end{cases}$$

where for $\Sigma_a = \sigma_a \mathbb{I}_{2\times 2}$, we have

$$\mu_\Delta = (R_\Delta^{-1} + \Sigma_a^{-1})^{-1}(R_\Delta^{-1}(\Phi_\Delta x_{k-1} + a_\Delta) + \Sigma_a^{-1} M^{-1} y_k), \quad \text{and} \quad \Sigma_\Delta = (R_\Delta^{-1} + \Sigma_a^{-1})^{-1},$$

and

$$h^\theta_\Delta(y_k|x_{k-1}) := \begin{cases} 1 - \Delta\lambda, & \text{if } y_k = \emptyset, \\ \lambda \mathcal{N}(\xi_\Delta, \Xi_\Delta), & \text{otherwise,} \end{cases}$$

where

$$\xi_\Delta = M(\Phi_\Delta x_{k-1} + a_\Delta), \quad \text{and} \quad \Xi_\Delta = M(\Sigma_a + R_\Delta)M^\intercal.$$

See Appendix 6.A for the full derivation. Conversely, when it is impossible to directly sample from the target density – which is the case when using the Airy or Born and Wolf profiles (Equations 6.8 and 6.10) directly – one can approximate the quantities in Equation 6.17. A common choice is

$$h^\theta_\Delta(x_k|y_k,x_{k-1}) := f^\theta_\Delta(x_k|x_{k-1}), \quad \text{and } h^\theta_\Delta(y_k|x_{k-1}) := h_\theta(y_k),$$

where f_Δ^θ is the Markov transition density and $h_\theta(y_k)$ is a strictly positive arbitrary function. In this case, by choosing $h_\theta(y_k) \propto 1$, the auxiliary particle filter becomes the well-known *bootstrap filter*, introduced in Gordon et al. [1993] and summarised in Algorithm 3.8. Given weighted particle sample $(X_{k-1}^{(1:N)}, \omega_{k-1}^{(1:N)})$ at step k, we denote an instance of running the particle filter as

$$\left(X_k^{(1:N)}, \omega_k^{(1:N)}\right) := \mathsf{PF}_\Delta\left(X_{k-1}^{(1:N)}, \omega_{k-1}^{(1:N)}\right).$$

For this particular problem, we must also take into account the missing observations introduced by the time discretisation. Since a lack of observation does not bring any new information, it suffices to only run the particle filter at segments which contain an observation. A typical iteration of this approach is summarised in Algorithm 6.1. The interval counter is initialised at $c_0 := 1$ and counts the number of discrete intervals since (and including) the last observation.

Algorithm 6.1 Particle filter for SDE with missing observations

Input: weighted particle sample $\left(X_{k-1}^{(1:N)}, \omega_{k-1}^{(1:N)}\right)$ and interval counter c_{k-1} at step $k-1$.

1: **if** $y_k = \emptyset$ **then**
2: $\quad c_k := c_{k-1} + 1$
3: $\quad$ Do not run the particle filter

$$\left(X_k^{(1:N)}, \omega_k^{(1:N)}\right) := \left(X_{k-1}^{(1:N)}, \omega_{k-1}^{(1:N)}\right).$$

4: **else**
5: $\quad$ Run the particle filter with updated interval length, i.e.

$$\left(X_k^{(1:N)}, \omega_k^{(1:N)}\right) := \mathsf{PF}_{c_k\Delta}\left(X_{k-1}^{(1:N)}, \omega_{k-1}^{(1:N)}\right).$$

6: $\quad c_k := 1$
7: **end if**

Output: updated particle sample $\left(X_t^{(1:N)}, \omega_t^{(1:N)}\right)$.

Example 6.3. Let the trajectory of a molecule be given by the SDE in Example 6.1 and simulated three times (one for each measurement model) using the same parameters and for the same time interval. Observations are generated as per in Example 6.2 for the 2D Gaussian, Airy profiles and Born and Wolf model and the molecules are tracked using the particle filter. The resulting estimated trajectories for each measurement model are given in Figure 6.5.

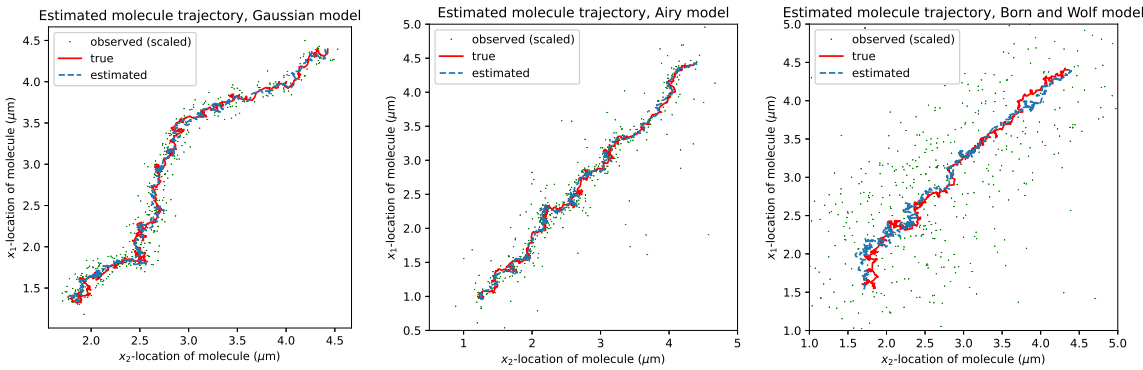


Fig. 6.5 Estimated molecule trajectories for the 2D Gaussian (*left*), Airy (*middle*) profiles and Born and Wolf model (*right*). The scaled observations (i.e. divided by m) for each measurement model are displayed for information.

6.5.3 Particle approximations of expectations of additive functionals

Recall that the second inference aim is to obtain ML estimates of the hyperparameters θ, as well as estimates of their score and OIM. To achieve these aims, we make use of smoothed additive functionals.

Assume that there exists a real-valued function S_k^θ, $k \geq 0$ such that it is an *additive functional* given by

$$S_k^\theta(x_{1:k}) = \sum_{j=1}^{k} s_j^\theta(x_{j-1}, x_j), \tag{6.18}$$

where $s_1^\theta(x_0, x_1) := s_1^\theta(x_1)$ and $\{s_k^\theta\}_{k\geq 0}$ is a sequence of *sufficient statistics* which may depend on the value of the observations $y_{0:k}$. The main aim is to compute the posterior or *smoothing expectation*, given by

$$\mathscr{S}_k(\theta) := \mathbb{E}_\theta\left[S_k^\theta(X_{1:k})\,|y_{1:k}\right] = \int_{\mathscr{X}} S_k^\theta(x_{1:k})\, p_\theta(x_{1:k}|y_{1:k})\mathrm{d}x_{1:k}. \tag{6.19}$$

If the model in question is linear and Gaussian or the state space $\mathscr{X}$ is finite, then the expectation $\mathscr{S}_k(\theta)$ can be computed exactly by recursion. However, this is not the case if the Airy or Born and Wolf profiles are used to describe photon distribution. In this case, SMC methods can again be employed to approximate the expectation as follows

$$\hat{\mathscr{S}}_k(\theta) := \sum_{i=1}^{N} \omega_k^{(i)} S_k^\theta\left(X_{1:k}^{(i)}\right),$$

where the weighted sample $(X_{1:k}^{(1:N)}, \omega_k^{(1:N)})$ is a particle approximation of the joint smoothing distribution $p_\theta(x_{1:k}|y_{1:k})$ obtained using a particle filter.

A simple way of estimating the smoothing expectation $\mathscr{S}_n(\theta)$ for a set of n observations $y_{1:n}$ is to run the desired particle filter in a 'forward pass' through the whole data to obtain the particle approximation $(X_n^{(1:N)}, \omega_n^{(1:N)})$ at step n, followed then by a 'backward smoothing' pass through the data, starting from the latest sample y_n. This is the case of algorithms such as the fixed-lag smoother by Kitagawa and Sato [2001]; Olsson et al. [2008, 2011], forward-filtering backward smoothing (FFBSm) by Doucet et al. [2000]; Hürzeler and Künsch [1998]; Kitagawa [1996] and forward-filtering backward simulation (FFBSi) by Godsill et al. [2004]. See Section 3.4 for more details on these algorithms.

However, if one wishes to avoid multiple passes through the data, it is also possible to take advantage of the form of the additive functional in Equation 6.18 to estimate $\mathscr{S}_k(\theta)$ in an online or 'forward-only' fashion, as proposed in Del Moral et al. [2010]. Introducing the *auxiliary function*

$$T_k^\theta(x_k) := \int_{\mathscr{X}^{k-1}} S_k^\theta(x_{1:k}) p_\theta(x_{1:k-1}|y_{1:k-1}, x_k) \mathrm{d}x_{1:k-1},$$

the following recursion is then created

$$T_k^\theta(x_k) = \int_{\mathscr{X}} \left[T_{k-1}^\theta(x_{k-1}) + s_k^\theta(x_{k-1}, x_k) \right] p_\theta(x_{k-1}|y_{1:k-1}, x_k) \mathrm{d}x_{k-1}, \tag{6.20}$$

where $T_0^\theta := 0$ and its particle approximation given the weighted sample $(X_{1:k}^{(1:N)}, w_k^{(1:N)})$ and previous state particle approximation $\hat{T}_{k-1}^\theta(X_{k-1}^{(1:N)})$ is given by

$$\hat{T}_k^\theta\left(X_k^{(i)}\right) = \sum_{j=1}^N \Psi_k^\theta(i,j) \left[\hat{T}_{k-1}^\theta\left(X_{k-1}^{(j)}\right) + s_k^\theta\left(X_{k-1}^{(j)}, X_k^{(i)}\right) \right] \tag{6.21}$$

for all $i \in [\![1, N]\!]$, and where

$$\Psi_k^\theta(i,j) := \frac{\omega_{k-1}^{(j)} f_\Delta^\theta\left(X_k^{(i)}|X_{k-1}^{(j)}\right)}{\sum_{j=1}^N \omega_{k-1}^{(j)} f_\Delta^\theta\left(X_k^{(i)}|X_{k-1}^{(j)}\right)}. \tag{6.22}$$

Finally, using the recursion on the auxiliary function T_k^θ, the smoothing expectation in Equation 6.19 can be rewritten as

$$\mathscr{S}_k(\theta) = \int_{\mathscr{X}} T_k^\theta(x_k) p_\theta(x_k|y_{1:k}) \mathrm{d}x_k, \tag{6.23}$$

and its particle approximation is

$$\hat{\mathscr{S}}_k(\theta) = \sum_{i=1}^{N} \omega_k^{(i)} \hat{T}_k^{\theta}\left(X_k^{(i)}\right).$$

This algorithm is known as *Forward smoothing SMC* (SMC-FS) and is summarised in Algorithm 3.14. The complexity of this algorithm is $\mathscr{O}\left(N^2\right)$ as it involves, among other things, evaluating f_{Δ}^{θ} and s_k^{θ} over all combinations $\{(X_{k-1}^{(j)}, X_k^{(i)})\}_{(i,j)\in[\![1,N]\!]^2}$, and computing the particle approximation of the recursion in Equation 6.21 is a particularly expensive task. To reduce computation time, the *particle-based, rapid incremental smoother* (PaRIS) by Olsson et al. [2017] employs a user-defined sample size $\tilde{N} \in \mathbb{N}$, also known as the *precision parameter*, and introduces an additional resampling step as follows: sample $\tilde{N}$ indices, i.e.

$$I_k^{(i)} \sim \mathbb{P}\left(\left\{\Psi_k^{\theta}(i,j)\right\}_{j=1}^{N}\right), \quad \text{for all } i \in [\![1,\tilde{N}]\!], \tag{6.24}$$

where $\Psi_k^{\theta}(i,j)$ is defined in Equation 6.22, and update the auxiliary statistic by replacing Equation 6.21 with the following:

$$\hat{T}_k^{\theta}\left(X_k^{(i)}\right) = \frac{1}{\tilde{N}} \sum_{j=1}^{\tilde{N}} \hat{T}_{k-1}^{\theta}\left(X_{k-1}^{(I_k^{(j)})}\right) + s_k^{\theta}\left(X_{k-1}^{(I_k^{(j)})}, X_k^{(i)}\right), \quad \text{for all } i \in [\![1,\tilde{N}]\!]. \tag{6.25}$$

This already speeds up the algorithm, as the sufficient statistic s_k^{θ} is now only evaluated $N \times \tilde{N}$ times and the precision parameter can be as low as $\tilde{N} = 2$ and still return accurate estimates. It is also possible to apply an accept-reject technique suggested in Douc et al. [2011] and described in Algorithm 3.13 to further reduce the complexity of the algorithm to linear, or $\mathscr{O}(N)$. Note that if considering the 2D Gaussian photon detection profile, while for the purpose of this chapter we apply the SMC-FS algorithm to obtain estimates of additive functionals, exact Kalman filtering and smoothing techniques can also be employed in practice, as per Section 3.2.2

Next, we establish the specific additive functionals T_k^{θ} and sufficient statistics s_k^{θ} required to estimate the hyperparameters θ and their score and OIM.

6.5.4 Estimation of the score and observed information matrix (OIM)

The score and OIM have important applications to ML estimation, e.g. see Le Gland and Mevel [1997]; Poyiadjis et al. [2011]. They can also be instrumental in assessing the performance of such an estimator, either directly, as argued by Efron and Hinkley [1978], or

as tools to estimate the FIM when the latter cannot be computed exactly, as we will see in this section. We aim to compute, recursively in time, the score vector $\mathscr{G}_k(\theta) := \nabla \log p_\theta(y_{1:k})$ and OIM $\mathscr{H}_k(\theta) := -\nabla^2 \log p_\theta(y_{1:k})$ where $p_\theta(y_{1:k})$ denotes the marginal likelihood at time step $1 \leq k \leq n$ defined in Equation 6.16, ∇ denotes the gradient and ∇^2 the Hessian.

6.5.4.1 Establishing the sufficient statistics

First of all, assume that the regularity conditions allowing for differentiation and integration to be switched around in expressions are satisfied. Let us establish the Fisher and Louis identities for the score and observed information matrix, respectively, from Cappé et al. [2006]; Douc et al. [2014]:

$$\begin{aligned} \mathscr{G}_k(\theta) &= \int_{\mathscr{X}} \nabla \log p_\theta(x_k, y_{1:k}) p_\theta(x_k|y_{1:k}) \mathrm{d}x_k, \\ \mathscr{H}_k(\theta) &= \nabla \log p_\theta(y_{1:k}) \nabla \log p_\theta(y_{1:k})^\intercal - \frac{\nabla^2 p_\theta(y_{1:k})}{p_\theta(y_{1:k})}, \end{aligned} \tag{6.26}$$

where

$$\begin{aligned} \frac{\nabla^2 p_\theta(y_{1:k})}{p_\theta(y_{1:k})} &= \int_{\mathscr{X}} \nabla \log p_\theta(x_k, y_{1:k}) \nabla \log p_\theta(x_k, y_{1:k})^\intercal p_\theta(x_k|y_{1:k}) \mathrm{d}x_k \\ &\quad + \int_{\mathscr{X}} \nabla^2 \log p_\theta(x_k, y_{1:k}) p_\theta(x_k|y_{1:k}) \mathrm{d}x_k, \end{aligned} \tag{6.27}$$

and note that Equations 6.26 and 6.27 can be rewritten as

$$\begin{aligned} \nabla \log p_\theta(y_{1:k}) &= \mathbb{E}\left[\alpha_k^\theta(X_k)|y_{1:k}\right], \\ \frac{\nabla^2 p_\theta(y_{1:k})}{p_\theta(y_{1:k})} &= \mathbb{E}\left[\alpha_k^\theta(X_k)\alpha_k^\theta(X_k)^\intercal|y_{1:k}\right] + \mathbb{E}\left[\beta_k^\theta(X_k)|y_{1:k}\right], \end{aligned}$$

where the expectations are with respect to the density $p(x_k|y_{1:k})$, and the functions $\alpha_k^\theta(x_k) := \nabla \log p_\theta(x_k, y_{1:k})$ and $\beta_k^\theta := \nabla^2 \log p_\theta(x_k, y_{1:k})$ are the additive functionals of interest. A recursion for α_k^θ and β_k^θ is straightforward to obtain, more details in Poyiadjis et al. [2011]. For α_k^θ and β_k^θ, Equation 6.20 becomes

$$\begin{aligned} \alpha_k^\theta(x_k) &= \int_{\mathscr{X}} \left[\alpha_{k-1}^\theta(x_{k-1}) + s_k^\alpha(x_{k-1}, x_k)\right] p_\theta(x_{k-1}|y_{1:k-1}, x_k) \mathrm{d}x_{k-1}, \\ \beta_k^\theta(x_k) &= \int_{\mathscr{X}} \left[\beta_{k-1}^\theta(x_{k-1}) + s_k^\beta(x_{k-1}, x_k)\right] p_\theta(x_{k-1}|y_{1:k-1}, x_k) \mathrm{d}x_{k-1} - \alpha_k^\theta(x_k)\alpha_k^\theta(x_k)^\intercal, \end{aligned}$$

where the sufficient statistics are given by

$$s_k^{\alpha}(x_{k-1},x_k) := \nabla \log G_k^{\theta}(x_k) + \nabla \log f_{\Delta}^{\theta}(x_k|x_{k-1}), \tag{6.28}$$

$$s_k^{\beta}(x_{k-1},x_k) := \left[\alpha_{k-1}^{\theta}(x_{k-1}) + s_k^{\alpha}(x_{k-1},x_k)\right]\left[\alpha_{k-1}^{\theta}(x_{k-1}) + s_k^{\alpha}(x_{k-1},x_k)\right]^{\intercal} + \nabla^2 \log G_k^{\theta}(x_k) + \nabla^2 \log f_{\Delta}^{\theta}(x_k|x_{k-1}). \tag{6.29}$$

Finally, to approximate the score and OIM, adapt the particle approximation in Equation 6.21 or 6.25 to the recursions in Equations 6.28 and 6.29 to obtain the score estimate

$$\hat{\mathscr{G}}_k(\theta) = \sum_{i=1}^{N} \omega_k^{(i)} \hat{\alpha}_k^{\theta}\left(X_k^{(i)}\right),$$

and OIM estimate

$$\hat{\mathscr{H}}_k(\theta) = \hat{\mathscr{G}}_k(\theta)\hat{\mathscr{G}}_k(\theta)^{\intercal} - \sum_{i=1}^{N} \omega_k^{(i)} \left[\hat{\alpha}_k^{\theta}\left(X_k^{(i)}\right)\hat{\alpha}_k^{\theta}\left(X_k^{(i)}\right)^{\intercal} + \hat{\beta}_k^{\theta}\left(X_k^{(i)}\right)\right].$$

Next, we apply this framework to a possible application of the single-molecule tracking model. We focus on the case where the photon distribution is described by the Airy or 2D Gaussian profile.

Example 6.4. Let the trajectory of a molecule be given by the following SDE

$$\mathrm{d}X_t = b\mathbb{I}_{2\times 2}X_t\mathrm{d}t + \sqrt{2\sigma}\mathrm{d}B_t,$$

where in the drift term, $b \neq 0$, in the diffusion term, $\sigma > 0$, and $(\mathrm{d}B_t)_{t_0\leq t\leq T}$ is a Wiener process. Let the photon detection process be described by the Airy or 2D Gaussian profile. Then, the parameters of interest are $\theta = (\sigma, b)$. Recall from Section 6.4.2 and Example 6.1 that the solution to the SDE can be written as

$$X_k = e^{b\Delta}X_{k-1} + W_x, \quad W_x \sim \mathscr{N}\left(0, \frac{\sigma}{b}\left(e^{2\Delta b} - 1\right)\mathbb{I}_{2\times 2}\right), \tag{6.30}$$

and since the potential function G_k does not depend on θ in this case, it can be dropped from Equations 6.28 and 6.29 and the components of the sufficient statistic $s_k^{\alpha}(x_{k-1},x_k)$ for the

additive functional α_k^θ are

$$\frac{\partial}{\partial\sigma}\log f_\Delta^\theta(x_k|x_{k-1}) = -\frac{1}{\sigma} + \frac{b\left\|x_k - e^{b\Delta}x_{k-1}\right\|^2}{2\sigma^2\left(e^{2b\Delta}-1\right)},$$

$$\begin{aligned}\frac{\partial}{\partial b}\log f_\Delta^\theta(x_k|x_{k-1}) &= \frac{1}{b} - \frac{2\Delta e^{2\Delta b}}{\left(e^{2\Delta b}-1\right)} - \frac{\left\|x_k - e^{\Delta b}x_{k-1}\right\|^2}{2\sigma(e^{2\Delta b}-1)} \\ &\quad + \frac{\Delta b e^{\Delta b}(x_k - e^{\Delta b}x_{k-1})^\intercal x_{k-1}}{\sigma(e^{2\Delta b}-1)} + \frac{\left\|x_k - e^{\Delta b}x_{k-1}\right\|^2 \Delta b e^{2\Delta b}}{\sigma(e^{2\Delta b}-1)^2}.\end{aligned}$$

The components of the sufficient statistic $s_k^\beta(x_{k-1},x_k)$ for β_k^θ are given in Appendix 6.B. Note that these derivatives can be evaluated for any value of Δ, and it is therefore possible to adapt them in order to only compute sufficient statistics when an observation is recorded, as in Algorithm 6.1.

6.5.5 Estimating the Fisher information matrix (FIM)

The *Fisher information matrix* (FIM) is widely used in estimation problems as an indicator of the performance of a given estimator. Indeed, it is a key element of the Cramér-Rao inequality, or *Cramér-Rao Lower Bound* (CRLB) derived by Cramér [1999]; Darmois [1945]; Fréchet [1943]; Rao [1992], which states that for an unbiased estimate $\hat{\theta}$ of the parameter θ, its covariance has lower bound

$$\mathrm{Cov}(\hat{\theta}) \succeq \mathscr{I}_n(\theta)^{-1},$$

where given matrices A and B, the inequality $A \succeq B$ indicates that $A - B$ is a positive semi-definite matrix, and $\mathscr{I}_n(\theta)$ denotes the FIM in a random sample $Y_1,\ldots,Y_n$ of size n (DeGroot and Schervish [2012]) as

$$\mathscr{I}_n(\theta) = \mathbb{E}_\theta\left[\nabla \log p_\theta(Y_{1:n})\nabla \log p_\theta(Y_{1:n})^\intercal\right] \tag{6.31}$$

$$= \mathbb{E}_\theta\left[-\nabla^2 \log p_\theta(Y_{1:n})\right], \tag{6.32}$$

where the second equality is proven in Duchi [2016]. When the expectations in Equations 6.31 and 6.32 are intractable − which is the case when the Airy profile is used to describe the photon detection locations in the single-molecule tracking model − there are two ways one can go about estimating the FIM.

6.5.5.1 Estimating the FIM for a large sample

Firstly, note that from Equation 6.32, the relationship between the FIM and OIM is simply

$$\mathscr{I}_n(\theta) = \mathbb{E}_\theta\left[\mathscr{H}_n(\theta)\right],$$

where $\mathscr{H}_n(\theta) = -\nabla^2 \log p_\theta(y_{1:n})$ denotes the OIM. Then, for a general state space model, Bickel et al. [1998] proved that under mild assumptions,

$$\frac{1}{n}\mathscr{H}_n(\theta) \to \mathscr{I}(\theta) \quad \text{as } n \to \infty,$$

where $\mathscr{I}(\theta)$ is the asymptotic FIM. So for a large enough sample size n, the OIM and FIM can be used interchangeably, i.e. for $n \gg 1$,

$$\mathscr{H}_n(\theta) \approx \mathscr{I}_n(\theta). \tag{6.33}$$

See Figures 6.6 and 6.15 for an illustration. Therefore, the first way of estimating the *asymptotic* FIM in the single-molecule tracking model is simply to obtain the OIM for a large sample size.

6.5.5.2 Estimating the Fisher information matrix for short samples

If the size n of the sample of interest is not large enough to estimate the FIM using the OIM, it is also possible to instead obtain a particle approximation of the expectation in Equation 6.31 using the score as follows: generate D datasets $y_{1:n}^{(1:D)}$ of (smaller) size n where $y_{1:n}^{(d)} := \{y_1^{(d)}, \ldots, y_n^{(d)}\}$, and according to the same parameters θ. The outer product of the score can then be used in the estimate of the Fisher information matrix as follows

$$\hat{\mathscr{I}}_n(\theta) = \frac{1}{D}\sum_{j=1}^{D} \mathscr{G}_n^{(d)}(\theta)\mathscr{G}_n^{(d)}(\theta)^\intercal, \tag{6.34}$$

where for $d = 1, \ldots, D$, the vector $\mathscr{G}_n^{(d)}(\theta) := \nabla \log p_\theta(y_{1:n}^{(d)})$ is the score for the d-th dataset of size n. The OIM can similarly be averaged over D datasets to estimate the FIM.

Now that the method for estimating the FIM has been established, it can be used in an experimental design setting to plan experiments with the aim of returning the most accurate parameter estimates. Next, we focus on how these estimates can be obtained.

Example 6.5. To verify these approaches to estimate the FIM, consider the straightforward special case of estimating the FIM for the location $x_0 = (x_{0,1}, x_{0,2})$ parameters of a static

molecule emitting photons at a constant rate. In Chao et al. [2016]; Ober et al. [2004b], the analytical expression for the FIM is derived for both the Gaussian and Airy profiles, and its diagonal components given observations $y_{1:n}$ they are given by

$$\mathscr{I}_n^{\text{Gauss}}(x_{0,1}) = \mathscr{I}_n^{\text{Gauss}}(x_{0,2}) = \frac{N_{phot}}{\sigma_a^2},$$
$$\mathscr{I}_n^{\text{Airy}}(x_{0,1}) = \mathscr{I}_n^{\text{Airy}}(x_{0,2}) = N_{phot}\alpha^2,$$

where $\alpha = \frac{2\pi n_a}{\lambda_e}$, N_{phot} denotes the expected photon count, $\mathscr{I}^{\text{Gauss}}(x_{0,i})$ denotes the (i,i)-th element of the FIM, corresponding to parameter component x_i, for the Gaussian profile, and $\mathscr{I}^{\text{Airy}}(x_{0,i})$ denotes the same for the Airy profile. As mentioned in Section 6.4.2, having a static molecule simplifies the model. Since we have independent data, the true values of score $\mathscr{G}$ and OIM $\mathscr{H}$ can be derived as follows. Given a set of observations $y_{1:n}$ distributed according to the 2D Gaussian profile,

$$\mathscr{G}_n^{\text{Gauss}}(x_0) = \sum_{k=1}^{n} \Sigma_a^{-1}(M^{-1}y_k - x_0)\mathbb{1}_{y_k\neq\emptyset}, \qquad \mathscr{H}_n^{\text{Gauss}}(x_0) = \Sigma_a^{-1}\sum_{k=1}^{n}\mathbb{1}_{y_k\neq\emptyset},$$

where $\Sigma_a = \sigma_a^2\mathbb{I}_{2\times 2}$ and $\mathbb{1}_{y_k\neq\emptyset}$ indicates there isn't a missing observation at interval k. Similarly, the score and OIM for the Airy profile can be derived exactly

$$\mathscr{G}_n^{\text{Airy}}(x_0) = \sum_{k=1}^{n}\gamma_k(M^{-1}y_k - x_0)\mathbb{1}_{y_k\neq\emptyset},$$
$$\mathscr{H}_n^{\text{Airy}}(x_0) = \sum_{k=1}^{n}\left(\chi_k(M^{-1}y_k - x_0)(M^{-1}y_k - x_0)^\intercal + \gamma_k\mathbb{I}_{2\times 2}\right)\mathbb{1}_{y_k\neq\emptyset},$$

where

$$\gamma_k = \frac{2\alpha}{r}\frac{J_2(\alpha r_k)}{J_1(\alpha r_k)}, \qquad \chi_k = -\frac{2\alpha^2}{r_k^2}\left[\frac{J_3(\alpha r_k)}{J_1(\alpha r_k)} - \frac{J_2^2(\alpha r_k)}{J_1^2(\alpha r_k)}\right],$$

and $r_k = \sqrt{(M^{-1}y_k - x_0)^\intercal(M^{-1}y_k - x_0)}$. See Appendix 6.C for the full derivation.

Using the same settings as in Example 6.2, we simulate $D_l = 40$ 'large' datasets according to the Airy and Gaussian profiles consisting of observations obtained during the interval $[0, 0.2]$ seconds. We also simulate $D_s = 400$ 'short' datasets consisting of observations obtained during the shorter interval $[0, 0.02]$ seconds. The score and OIM are obtained for all datasets and the FIM for the large and short datasets is estimated in three ways: *(i)* using the OIM returned from a single dataset selected at random (Equation 6.33), *(ii)* using the mean outer product of the score (Equation 6.34) over all datasets and *(iii)* using the mean OIM across all datasets. Finally, the square root of the CRLB, also known as the *limit of accuracy*

and denoted

$$\delta_\vartheta = \sqrt{CRLB_\vartheta}$$

for parameter ϑ is obtained. This is repeated for various expected photon counts in order to compare the evolution of the estimated limit of accuracy as the expected number of photons increases to the true limit of accuracy obtained using the true FIM. In Figures 6.6 (and 6.15 in Appendix 6.C), it is evident that, apart from very low photon counts, all approaches are able to return accurate estimates of the limit of accuracy. Comparing Figures 6.6a and 6.6b (and similarly comparing Figures 6.15a, 6.15b in the appendix), it also becomes apparent that for long datasets, approach *(i)* is slightly more accurate than *(ii)*, and the opposite is true for short datasets. Similar results can be obtained for the Born and Wolf model, as an analytical expression for the FIM is also available for a static object.

6.5.6 Parameter estimation

Being able to estimate the biophysical parameters of the molecular interactions is very important in single-molecule tracking. In this section, we present two *maximum likelihood* (ML) estimation methods that make use of smoothed additive functionals.

6.5.6.1 By gradient ascent

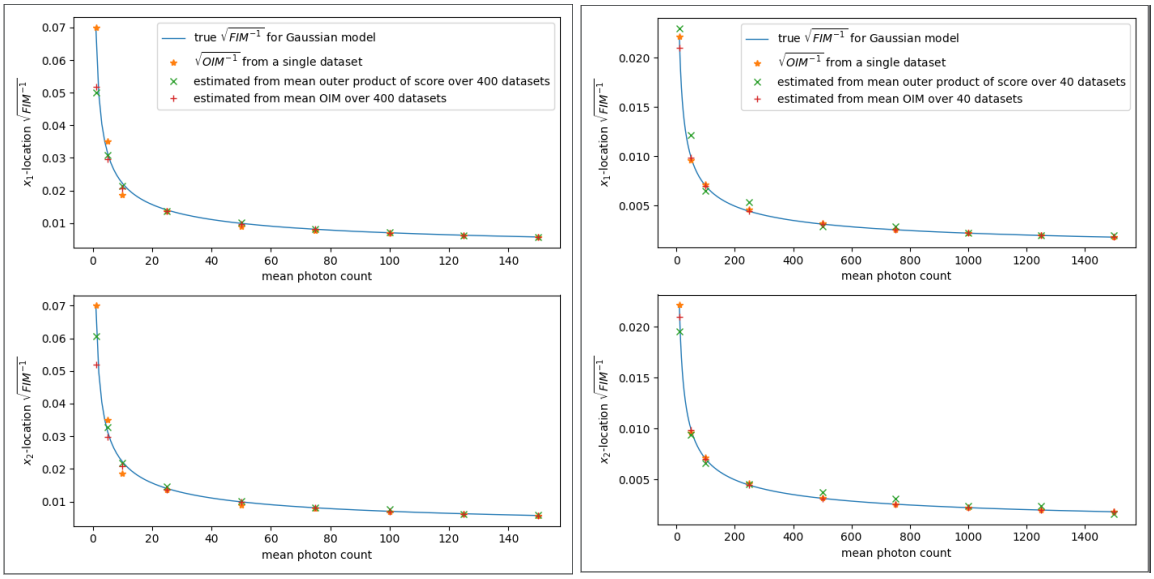
Given observations $y_{1:n}$ of size $n \in \mathbb{N}$, the marginal log-likelihood of the observations $p_\theta(y_{1:n})$ may be maximised via the *steepest ascent algorithm* Cauchy [1847]; Lemaréchal [2012]:

$$\theta_{i+1} = \theta_i + \gamma_{i+1}\mathscr{G}_n(\theta_i), \tag{6.35}$$

where $\mathscr{G}_n(\theta_i) = \nabla \log p_\theta(y_{1:n})|_{\theta=\theta_i}$ is the score vector evaluated at the current estimate θ_i, and the step-size sequence $\{\gamma_i\}_{i=1}^\infty$ consists of small positive numbers and satisfies $\sum_i \gamma_i = \infty$ and $\sum_i \gamma_i^2 < \infty$; for example, take $\gamma_i = i^{-a}$ where $0.5 < a < 1$. One can also include the observed information matrix in order to follow the *Newton-Raphson algorithm* described in Nocedal and Wright [2006]. In this case, Equation 6.35 becomes

$$\theta_{i+1} = \theta_i - \gamma_{i+1}\mathscr{H}_n(\theta_i)^{-1}\mathscr{G}_n(\theta_i),$$

where $\mathscr{H}_n(\theta_i) = \nabla^2 \log p_\theta(y_{1:n})|_{\theta=\theta_i}$ is the observed information matrix evaluated at the current estimate θ_i.



(a) Many short datasets, 2D Gaussian profile (b) Few long datasets, 2D Gaussian profile

Fig. 6.6 True and estimated limit of accuracy for mean photon counts ranging from (a) 1 to 150 (b) 10 to 1500. The limit of accuracy is estimated for the location parameters (x_1, x_2) of a static in-focus molecule. The estimates are obtained by taking the square root of the inverse of the FIM, obtained for (a) 400 'short' and (b) 40 'long' simulated datasets using approaches *(i)* $\star$, *(ii)* $\times$ and *(iii)* $+$ for comparison purposes. To generate each dataset, the photon detection times are simulated according to a Poisson process with constant rate corresponding to the expected mean photon count for (a) $[0, 0.02]$ and (b) $[0, 0.2]$ seconds and the intervals are discretised. The photon detection locations are generated according to the 2D Gaussian profile, with parameters as in Example 6.2. The true limit of accuracy (*blue solid line*) is also computed as it is available analytically (Ober et al. [2020c]). Estimates of the limit of accuracy based on a single dataset (approach *(i)*) are more accurate when the dataset is long, while taking the mean outer product of the score over all datasets (approach *(ii)*) yields more accurate estimates for a large number of short datasets. Approach *(iii)* provides a good balance between the two. More generally, estimates of the limit of accuracy are relatively poor for very low mean photon counts but quickly improve as it increases.

6.5.6.2 By expectation-maximization (EM)

Another approach to obtaining ML estimates of the hyperparameters θ is to use the *expectation-maximization* (EM) algorithm by Dempster et al. [1977]; Wu [1983] defined as follows:

- *Expectation* step: given the current parameter estimate θ_i and observations $y_{1:n}$,

$$\mathcal{Q}(\theta, \theta_i) = \mathbb{E}_{\theta_i}\left[\log p_\theta(X_{1:n}, y_{1:n}) | y_{1:n}\right],$$

  where the joint density $p_\theta(x_{1:n}, y_{1:n})$ is defined in Equation 6.15 and the expectation is with respect to the posterior $p_{\theta_i}(x_{1:n}|y_{1:n})$.

- *Maximisation* step:

$$\theta_{i+1} = \underset{\theta \in \Theta}{\operatorname{argmax}}\, \mathcal{Q}(\theta, \theta_i).$$

Recall that the *Expectation* step cannot be done exactly when using the Airy or Born and Wolf profile. In this case, the posterior expectation can be estimated using particle approximations of smoothed additive functionals. First of all, let $S_k^\theta(x_{1:k}) := \log p_\theta(x_{1:k}, y_{1:k})$ denote the additive functionals of interest at step k. Their corresponding sufficient statistics such that $S_k^\theta(x_{1:k}) = \sum_{k=1}^n s_k^\theta(x_{k-1}, x_k)$ are given by

$$s_k^\theta(x_{k-1}, x_k) := \log f_\Delta^\theta(x_k|x_{k-1}) + \log G_k^\theta(x_k),$$

where for notational simplicity, $f_\Delta^\theta(x_1|x_0) := \nu_\theta(x_1)$. In the *Maximisation* step, define the function Λ to obtain the maximising argument of $\mathcal{Q}(\theta, \theta_i)$ as

$$\theta_{i+1} = \Lambda\left(n^{-1}\mathbb{E}\left[S_n^{\theta_i}(X_{1:n})|y_{1:n}\right]\right).$$

Example 6.6. Building on Example 6.4, note that given the model specification in Equation 6.30, it is impossible to compute the maximum of $\mathcal{Q}(\theta, \theta_i)$ for the parameter $b \neq 0$ directly. However, as seen previously, the equation can also be written such that we simply have

$$X_k = \varphi_\theta X_{k-1} + W_x, \quad W_x \sim \mathcal{N}\left(0, r_\theta \mathbb{I}_{2\times 2}\right),$$

where the *auxiliary parameters* are given by

$$\varphi_\theta := e^{b\Delta} \qquad \text{and} \qquad r_\theta := \frac{\sigma}{b}\left(e^{2b\Delta} - 1\right).$$

It is straightforward to maximise $\mathcal{Q}(\theta, \theta_i)$ for the auxiliary parameters φ_θ and r_θ as follows:

let $\{S_{l,k}(x_{1:k})\}_{l=1}^{3}$ denote the additive functionals of interest at time k and $\{s_{l,k}(x_{k-1},x_k)\}_{l=1}^{3}$ their corresponding sufficient statistics. Luckily, the sufficient statistics are easily obtained, since for the Gaussian and Airy profiles, the likelihood G_k does not depend on θ:

$$s_{1,k}(x_{k-1},x_k) = x_k^\intercal x_{k-1}, \quad s_{2,k}(x_{k-1},x_k) = x_{k-1}^\intercal x_{k-1}, \quad s_{3,k}(x_{k-1},x_k) = x_k^\intercal x_k.$$

The maximisation function is given by

$$\Lambda(c_1,c_2,c_3) = \left(\frac{c_3}{2} - \frac{c_1^2}{2c_2}, \frac{c_1}{c_2}\right).$$

Finally, to obtain ML estimates for b and σ, simply use the following transformation

$$b = \Delta^{-1}\log\varphi_\theta \quad \text{and} \quad \sigma = \frac{r_\theta \log\varphi_\theta}{\Delta(\varphi_\theta^2 - 1)}.$$

Note that when dealing with measurements distributed according to the Born and Wolf model, we must also estimate the optical axis location parameter z_0, which is done via gradient ascent.

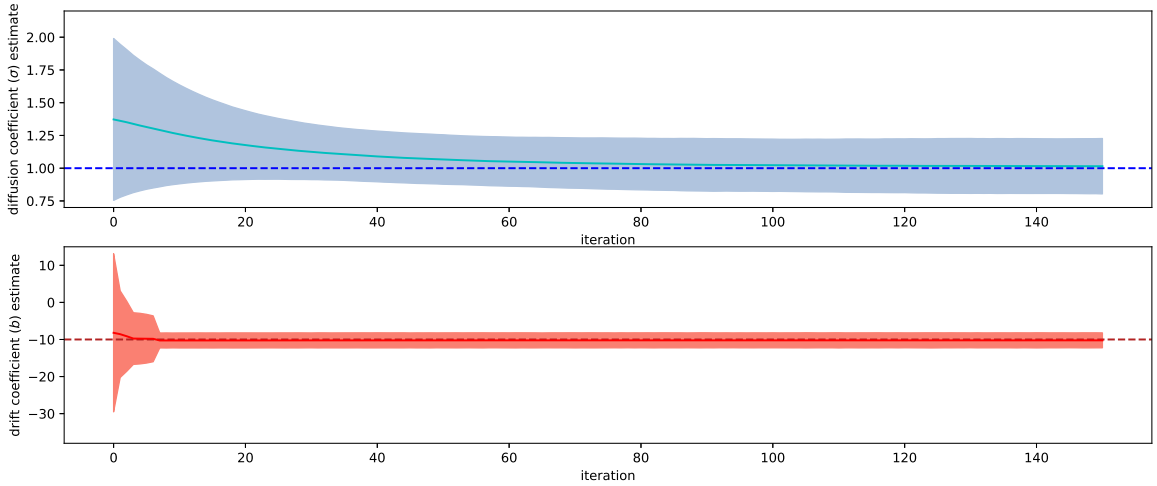


Fig. 6.7 Estimates of the diffusion (σ) and drift (b) coefficient over 150 EM iterations or passes through the data. The blue and red *dashed lines* represent the true parameter values $\sigma = 1\ \mu\text{m}^2/\text{s}$ and $b = -10\ \text{s}^{-1}$, respectively. The red and blue *solid lines* and *bands* correspond to the mean estimates and their corresponding 95% confidence intervals over 50 datasets generated during the time interval $[0,0.2]$ seconds, with initial location $x_0 = (5.5,5.5)^\intercal\ \mu\text{m}$ and a mean photon count of 1000. The observations were generated according to the Airy profile with parameters as in Example 6.2 and the sufficient statistics were estimated using the PaRIS algorithm.

6.6 Numerical experiments

In this section, we apply the particle smoother known as SMC-FS to estimate the FIM, and thus the limit of accuracy, of the diffusion and drift coefficients in the context of a moving molecule with a stochastic trajectory. Experiments are first run with photon detection locations described by the Gaussian and Airy profiles, and then the Born and Wolf model, where an additional hyperparameter, namely the optical axis location, must be considered as well. The methodology is then applied to the problem of assessing the limit of accuracy for the mean separation distance between two closely spaced molecules.

6.6.1 Limit of accuracy of drift and diffusion coefficients for the Gaussian and Airy profiles

Consider a molecule with trajectory described by the SDE in Example 6.1. In Vahid et al. [2020], the authors took advantage of the Kalman filter formulae to evaluate the FIM for the diffusion (σ) and drift (b) coefficients. However, it was only possible to obtain an analytic solution for a particular set of detection times $t_1, t_2, \ldots$ and for the 2D Gaussian photon distribution profile. Otherwise, the computational cost of performing numerical integration was too high for more than one photon.

In our particle filtering framework, it is also possible to take advantage of the Kalman filter formulae in order to obtain an accurate approximation of the true score and OIM by numerical differentiation, and for any detection times schedule. An estimate of the FIM is therefore obtained by evaluating the true OIM for 3000 datasets and taking their mean, as described in Section 6.5.5. The molecule trajectories are simulated for $[0, 0.2]$ seconds, with diffusion coefficient $\sigma = 1\ \mu\text{m}^2/\text{s}$, drift coefficient $b = -10\ \text{s}^{-1}$, and initial location Gaussian distributed with mean $x_0 = (5.5, 5.5)^\intercal\ \mu\text{m}$ and covariance $P_0 = 10^{-2}\mathbb{I}_{2\times 2}\ \mu\text{m}^2$. The observations are generated according to the Gaussian profile (Equation 6.9) with parameters as in Example 6.2. Of course, it is not possible to employ the Kalman filter formulae for the Airy and Born and Wolf profiles, and we must resort to using the SMC-FS algorithm instead. First of all, to evaluate the performance of the SMC-FS algorithm, the algorithm is employed using 500 particles to estimate the score and OIM for the same 3000 Gaussian datasets, and we similarly take the mean OIM over all datasets to estimate the FIM.

Next, we move on to the Airy profile, for which it was too computationally costly in Vahid et al. [2020] to obtain the FIM for more than a single photon. We estimate the OIM for the diffusion and drift coefficients using the SMC-FS algorithm with 500 particles for 2040 datasets, where the molecule trajectories are simulated using the same parameters as for the Gaussian profile, and the observations are generated according to the Airy profile (Equation

6.8) with parameters as in Example 6.2. This is repeated for various mean photon counts ranging from 10 to 1250. Then, the limit of accuracy, denoted δ_ϑ for hyperparameter ϑ, is computed, and the results are displayed in Figure 6.8.

Both Figures 6.8a and 6.8b display an inverse square root decay of the limit of accuracy with respect to the mean photon count. This is consistent with the results for a static molecule from Example 6.5, and means that the quality of diffusion and drift estimates improves as the mean photon count increases. In addition to that, comparing the limit of accuracy obtained from the estimated and true OIM for the Gaussian profile in Figure 6.8a indicates that the SMC-FS algorithm is able to return accurate estimates of the score and FIM for a stochastically moving molecule. Indeed, apart from a very slight discrepancy for very low photon counts for the drift coefficient, the estimates of the limit of accuracy are almost indistinguishable.

6.6.2 Limit of accuracy of drift, diffusion and optical axis location for the Born and Wolf model

When the molecule is out of focus, which means the photon detection locations are distributed according to the Born and Wolf model (Equation 6.10), the FIM components for the diffusion and drift coefficients can be obtained as for the Airy and Gaussian profiles. However, a new hyperparameter must be considered, namely the optical axis location, denoted z_0. While previously, differentiating the log potential function was not needed, the vector of hyperparameters is now $\theta = (\sigma, b, z_0)$, and $G_k^\theta(x_k)$ depends on z_0 for $k = 1, \ldots, n$.

While it requires quite a bit of numerical integration, differentiating $\log q_{z_0}(x_1, x_2)$ for a given $x = (x_1, x_2) \in \mathbb{R}^2$ with respect to z_0 is not impossible. For notational simplicity, let $\alpha = \frac{2\pi n_\alpha}{\lambda_e}$, $r = \sqrt{x_1^2 + x_2^2}$ and $W = \frac{\pi n_\alpha^2}{n_o \lambda_e}$ and note that we can rewrite Equation 6.10 as

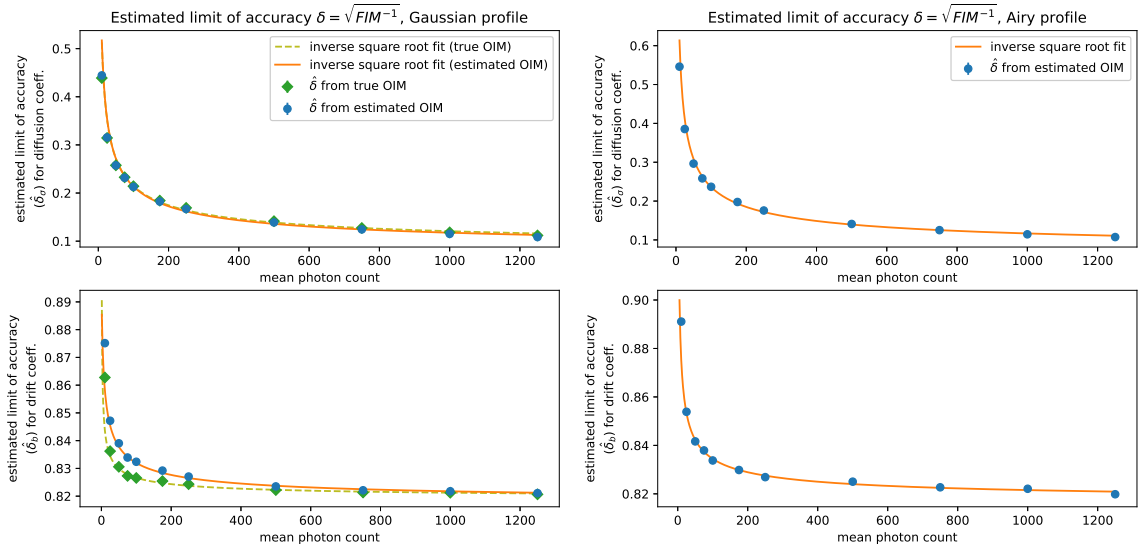
$$q_{z_0}(x_1, x_2) = \frac{\alpha^2}{\pi}\left(U_{z_0}^2 + V_{z_0}^2\right),$$

where

$$U_{z_0} := \int_0^1 J_0(\alpha r \rho)\cos\left(W z_0 \rho^2\right)\rho d\rho, \qquad V_{z_0} := \int_0^1 J_0(\alpha r \rho)\sin\left(W z_0 \rho^2\right)\rho d\rho.$$

The first derivative was derived in Ober et al. [2020b] and is given by

$$\frac{\partial \log q_{z_0}(x_1, x_2)}{\partial z_0} = 2\frac{U_{z_0}\dot{U}_{z_0} + V_{z_0}\dot{V}_{z_0}}{U_{z_0}^2 + V_{z_0}^2},$$



(a) 2D Gaussian profile (b) Airy profile

Fig. 6.8 Evolution of the estimated limit accuracy for mean photon counts ranging from 10 to 1250. The limit of accuracy is estimated for the diffusion (σ) and drift (b) coefficients for an in-focus molecule with stochastic trajectory. The estimates are obtained by taking the square root of the inverse of the FIM, obtained by estimating the OIM using the SMC-FS algorithm with 500 particles for (a) 3000 and (b) 2040 simulated datasets. To generate each dataset, the molecule trajectories are simulated according to the SDE in Example 6.1 for $[0, 0.2]$ seconds, with $\sigma = 1\ \mu\text{m}^2/\text{s}$, $b = -10\ \text{s}^{-1}$, and initial location Gaussian distributed with mean $x_0 = (5.5, 5.5)^\intercal\ \mu\text{m}$ and covariance $P_0 = 10^{-2}\mathbb{I}_{2\times 2}\ \mu\text{m}^2$. The observations are generated according to the (a) 2D Gaussian and (b) Airy profiles, with parameters as in Example 6.2. For the (a) 2D Gaussian profile, the limit of accuracy is also estimated by using the true OIM obtained using numerical differentiation applied to the Kalman filter. An inverse square root curve (*orange* and *green dashed*) is fitted to the resulting estimated limits of accuracy for comparison.

where

$$\dot{U}_{z_0} := \frac{\partial U_{z_0}}{\partial z_0} = \int_0^1 J_0(\alpha r\rho)\cos\left(Wz_0\rho^2\right)W\rho^3 d\rho,$$
$$\dot{V}_{z_0} := \frac{\partial V_{z_0}}{\partial z_0} = -\int_0^1 J_0(\alpha r\rho)\sin\left(Wz_0\rho^2\right)W\rho^3 d\rho.$$

The second derivative with respect to z_0 is given by

$$\frac{\partial^2 \log q_{z_0}(x_1,x_2)}{\partial z_0^2} = 2\frac{U_{z_0}\ddot{U}_{z_0} + \dot{U}_{z_0}^2 + V_{z_0}\ddot{V}_{z_0} + \dot{V}_{z_0}^2}{U_{z_0}^2 + V_{z_0}^2} - \left(\frac{\partial \log q_{z_0}(x_1,x_2)}{\partial z_0}\right)^2,$$

where

$$\ddot{U}_{z_0} := \frac{\partial^2 U_{z_0}}{\partial z_0^2} = -\int_0^1 J_0(\alpha r\rho)\cos\left(Wz_0\rho^2\right)W^2\rho^5 d\rho,$$
$$\ddot{V}_{z_0} := \frac{\partial^2 V_{z_0}}{\partial z_0^2} = -\int_0^1 J_0(\alpha r\rho)\sin\left(Wz_0\rho^2\right)W^2\rho^5 d\rho.$$

Fortunately, the potential function only depends on z_0, so any cross terms in the FIM and OIM between z_0 and either σ or b will be zero.

The OIM is estimated for the diffusion, drift coefficients and optical axis location using the SMC-FS algorithm with 500 particles for 2040 datasets, where the molecule trajectories are simulated using the same parameters as for the Gaussian and Airy profiles, and the observations are generated according to the Born and Wolf model with parameters as in Example 6.2. Then, the limit of accuracy for mean photon counts ranging from 10 to 1250 is computed, and the results are displayed in Figure 6.9. Once again, there is an inverse square root decay of the limit of accuracy with respect to the mean photon count for all hyperparameters considered.

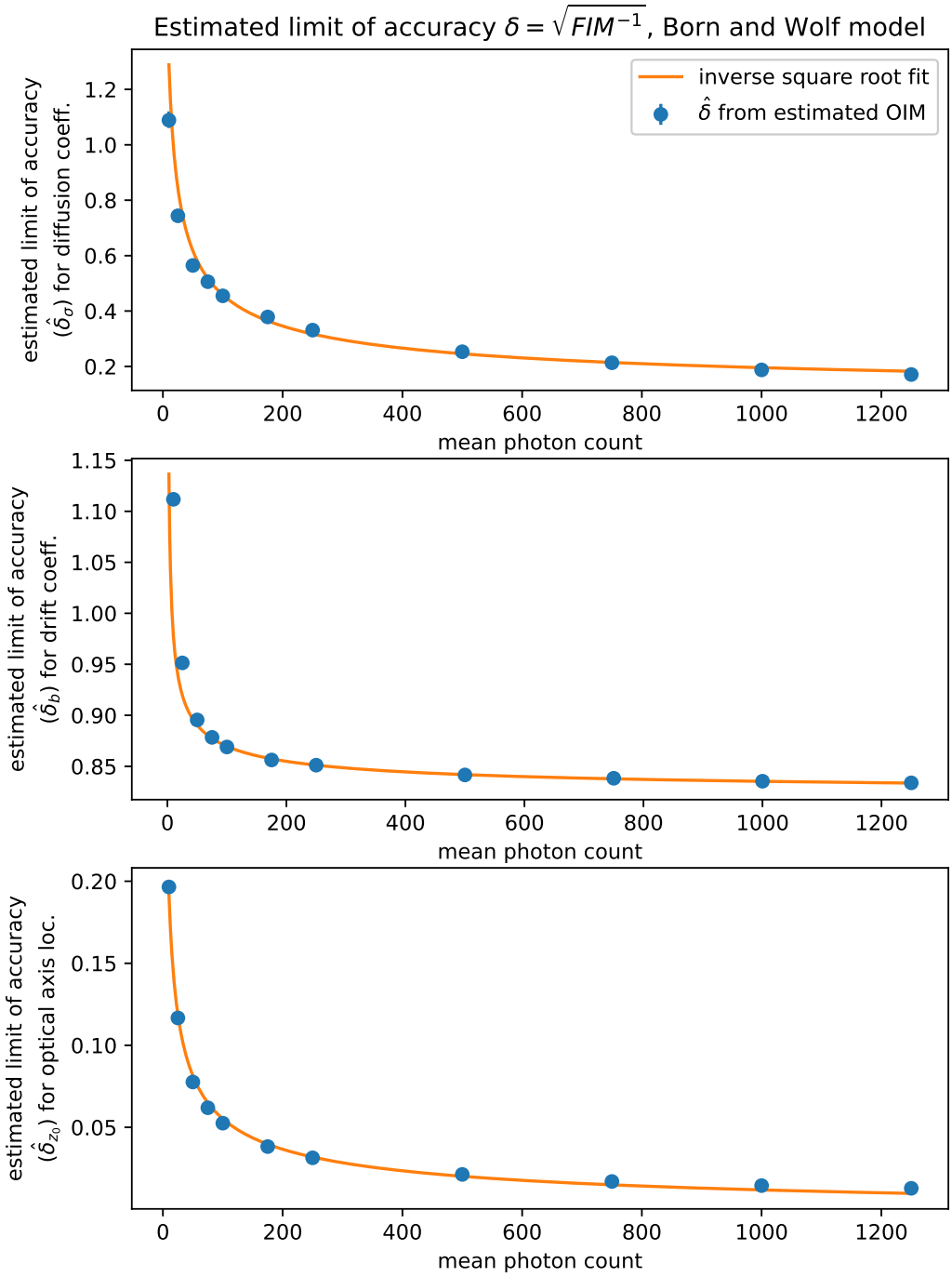


Fig. 6.9 Evolution of the estimated limit accuracy for mean photon counts ranging from 10 to 1250. The limit of accuracy is estimated for the diffusion (σ), drift (b) coefficients and optical axis location (z_0) for an out of focus molecule with stochastic trajectory. The estimates are obtained by taking the square root of the inverse of the FIM, obtained by estimating the OIM using the SMC-FS algorithm with 500 particles for 2040 simulated datasets. To generate each dataset, the molecule trajectories are simulated according to the SDE in Example 6.1 for $[0, 0.2]$ seconds, with $\sigma = 1\ \mu\text{m}^2/\text{s}$, $b = -10\ \text{s}^{-1}$, and initial location Gaussian distributed with mean $x_0 = (5.5, 5.5)^\intercal\ \mu\text{m}$ and covariance $P_0 = 10^{-2}\mathbb{I}_{2\times 2}\ \mu\text{m}^2$. The observations are generated according to the Born and Wolf model with parameters as in Example 6.2, where $z_0 = 1\ \mu\text{m}$. An inverse square root curve (*orange*) is fitted to the resulting estimated limits of accuracy for comparison.

6.6.3 Limit of accuracy of the separation distance between two molecules for the Airy profile

Being able to estimate the distance of separation between two closely spaced molecules is an important aspect of single-molecule microscopy. In the past, Rayleigh's criterion (Born and Wolf [2013]) has been used to define the minimum distance between two point sources such that they can be distinguished in the image. However, Ram et al. [2006a] developed a resolution measure which predicts that increasing the photon count makes it possible to estimate a separation distance between two molecules that is shorter than Rayleigh's criterion. This resolution measure is simply defined as the limit of accuracy for the separation distance. In Ram et al. [2013], further research was carried out in order to develop analytical formulations of the FIM – and hence limit of accuracy – for the separation distance under various detection scenarios. So far, the limit of accuracy has only been derived for static molecules. In this experiment, we apply our methodology to estimate the limit of accuracy for the locations and separation distance between two molecules that are not static, but diffusing independently at their respective stationary distributions, as illustrated in Figure 6.10.

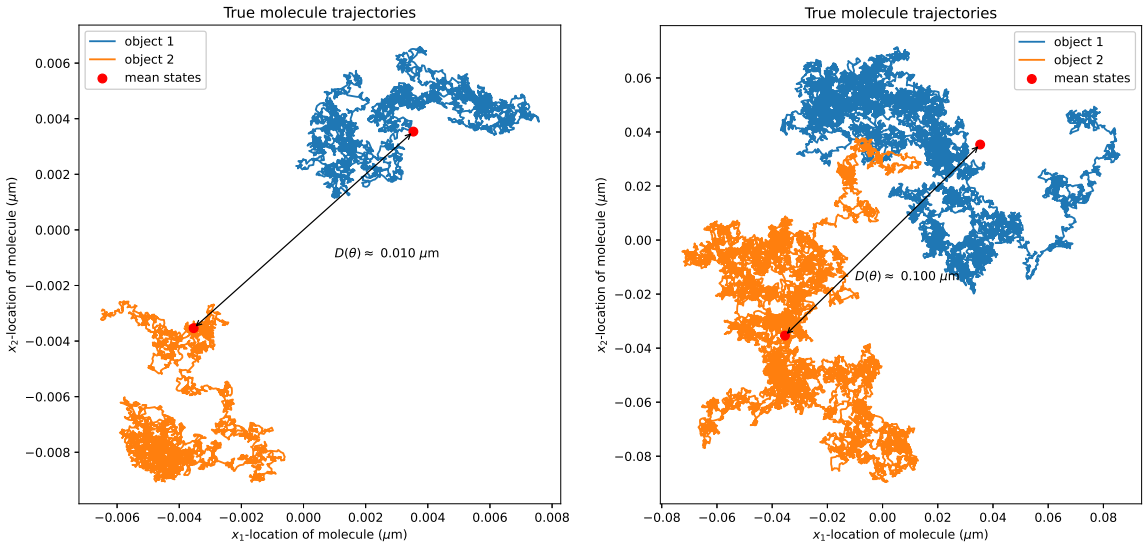


Fig. 6.10 Examples of two molecules diffusing independently at a mean separation distance of 0.01 μm with diffusion coefficient $\sigma = 10^{-4}$ μm^2/s (**left**) and of 0.1 μm with $\sigma = 10^{-3}$ μm^2/s (**right**). For an Airy distributed photon detection profile with $n_\alpha = 1.4$ and $\lambda_e = 0.52$ μm, Rayleigh's resolution limit is ≈ 0.227 μm. Note that increasing value of the diffusion coefficient σ leads to the molecule trajectories overlapping even if their stationary distributions are further apart.

Let $X_t = (X_{t,1}, X_{t,2})^\intercal$ be the cartesian coordinates of a moving molecule with stationary

distribution $\mathcal{N}(x_0, \sigma\mathbb{I}_{2\times 2})$ for all t. The continuous time dynamics are given by

$$\mathrm{d}X_t = (x_0 - X_t)\mathrm{d}t + \sqrt{2\sigma}\mathrm{d}B_t. \tag{6.36}$$

From Section 6.3.1, it is straightforward to establish the solution to this SDE, which yields the conditional pdf $f_\Delta^{x_0}$ of X_{k+1} at the $(k+1)$-th discrete segment, given $X_k = x$ at the k-th segment, as

$$X_{k+1}|(X_k = x) = \Phi_\Delta x + a_\Delta + W_x, \quad W_x \sim \mathcal{N}(0, R_\Delta),$$

where $\Phi_\Delta = e^{-\Delta}$, $a_\Delta = -x_0(e^{-\Delta} - 1)$ and $R_\Delta = -\sigma(e^{-2\Delta} - 1)\mathbb{I}_{2\times 2}$.

In this experiment, consider two independently diffusing molecules whose states are (X_t, Z_t), where X_t is the state of the first molecule and Z_t is the state of the second. The molecules are initialised at their respective stationary distributions, $\mathcal{N}(x_0, \sigma\mathbb{I}_{2\times 2})$ and $\mathcal{N}(z_0, \sigma\mathbb{I}_{2\times 2})$, and the conditional pdf of (X_{k+1}, Z_{k+1}) given $(X_k, Z_k) = (x_k, z_k)$ is $f_\Delta^{x_0}(x_{k+1}|x_k) f_\Delta^{z_0}(z_{k+1}|z_k)$ owing to their independent motions.

Assume that the mean location of the molecules $(x_0, z_0) =: \theta = (\theta_1, \theta_2, \theta_3, \theta_4)^\intercal$ is non-random but unknown and to be estimated. Let $\hat{\theta} = \left(\hat{\theta}_1(Y_{1:n}), \hat{\theta}_2(Y_{1:n}), \hat{\theta}_3(Y_{1:n}), \hat{\theta}_4(Y_{1:n})\right)^\intercal$ denote an estimate of θ given observations $Y_{1:n}$. Recall that the FIM, denoted $\mathcal{I}_n(\theta)$, is given by

$$\mathcal{I}_n(\theta) = \mathbb{E}\left[\nabla \log p_\theta(Y_{1:n})\, \nabla \log p_\theta(Y_{1:n})^T\right],$$

where

$$\nabla \log p_\theta(Y_{1:n}) = p_\theta(Y_{1:n})^{-1} \begin{pmatrix} \frac{\partial}{\partial \theta_1} \log p_\theta(Y_{1:n}) \\ \vdots \\ \frac{\partial}{\partial \theta_4} \log p_\theta(Y_{1:n}) \end{pmatrix}.$$

For any scalar-valued function $D(\theta) \in \mathbb{R}$, we can estimate $D(\theta)$ using $D(\hat{\theta})$ where $\hat{\theta}$ is the estimate of θ. Assuming the estimate is unbiased, we have the following CRLB for the function D,

$$\mathbb{E}\left[\left(D(\hat{\theta}) - D(\theta)\right)^2\right] \geq \nabla D(\theta)^\intercal \mathcal{I}_n(\theta)^{-1} \nabla D(\theta), \tag{6.37}$$

where $\nabla D(\theta) := (\partial D/\partial\theta_1, \ldots, \partial D/\partial\theta_4)^\intercal$. For example, to estimate the separation between the two molecules we have $D(\theta) = \sqrt{(\theta_1 - \theta_3)^2 + (\theta_2 - \theta_4)^2}$, and as a result

$$\nabla D(\theta) = \frac{1}{D(\theta)} \begin{pmatrix} \theta_1 - \theta_3 \\ \theta_2 - \theta_4 \\ -(\theta_1 - \theta_3) \\ -(\theta_2 - \theta_4) \end{pmatrix}.$$

This experiment is essentially the dynamic version of the experiments on estimating the separation of two static molecules by Ram et al. [2013]. The key difference here is that the molecules are diffusing. The observations $Y_{1:n}$ are generated as in Ram et al. [2013], i.e. according to the following mixture

$$\begin{aligned} G_k(x_k, z_k) &= \begin{cases} 1 - \Delta\lambda_\theta, & \text{if } y_k = \emptyset, \\ \lambda_\theta \left[\varepsilon_x g(y_k|x_k) + \varepsilon_z g(y_k|z_k)\right], & \text{otherwise,} \end{cases} \\ &= \begin{cases} 1 - \Delta\lambda_\theta, & \text{if } y_k = \emptyset, \\ \lambda_x g(y_k|x_k) + \lambda_z g(y_k|z_k), & \text{otherwise,} \end{cases} \end{aligned} \tag{6.38}$$

where g is the photon distribution profile given in Equation 6.7, $\varepsilon_x := \frac{\lambda_x}{\lambda_\theta}$ for $\lambda_\theta = \lambda_x + \lambda_z$ and similarly for ε_z. The measurement model considered in this experiment is the Airy profile (Equation 6.8), but it is straightforward to also apply the methodology to the 2D Gaussian profile and Born and Wolf model.

In the first part of the experiment, we replicate results similar to those in Ram et al. [2013] for two static molecules, and observe how introducing diffusion affects the progression of the limit of accuracy for the mean location θ, denoted δ_θ, as well as the limit of accuracy $\delta_{D(\theta)}$ for the separation distance (obtained using Equation 6.37), as this separation distance between the two molecules increases. We set $\lambda_x = \lambda_z = \lambda$ for simplicity. Estimating δ_θ for the static case is performed as in Example 6.5. The molecules are observed during an interval of $[0, 1]$ seconds with a mean photon count, denoted N_{phot}, of 3000, and the parameters of the Airy profile are unchanged (i.e. $n_\alpha = 1.4$, $\lambda_e = 0.52$ μm), as is the lateral magnification matrix ($M = 100\mathbb{I}_{2\times 2}$). For the dynamic case, the molecules are observed for the same interval and mean photon count, and for diffusion coefficients σ varying from 10^{-3} to 10^{-4} $\mu\text{m}^2/\text{s}$. The estimate of the limit of accuracy is obtained by estimating the OIM via the SMC-FS algorithm for approximately 100 datasets. The resulting estimated limits of accuracy $\hat{\delta}_\theta$ and $\hat{\delta}_{D(\theta)}$ are given in Figure 6.11.

The second part of the experiment involves similarly estimating the limits of accuracy δ_θ and $\delta_{D(\theta)}$ for various separation distances, but this time the diffusion coefficient remains fixed, i.e. $\sigma = 10^{-4}$ $\mu\text{m}^2/\text{s}$, and the mean photon count N_{phot} is set to vary between 100 and 4500. The resulting estimated limits of accuracy are given in Figure 6.12.

As the separation distance $D(\theta)$ gets closer to zero, the limit of accuracy increases, indicating that estimates would become less accurate. Additionally, an inverse square root curve was fit to each set of estimated limits of accuracy in Figures 6.11 and 6.12. This is consistent with results in Ram et al. [2006a] that showed an inverse square root relationship

between separation distance and $\delta^{static}_{D(\theta)}$ for two static molecules, and indicates that these results can be generalised to dynamic molecules.

In Ober et al. [2004b], it is suggested that the limit of accuracy for the location of a static molecule, known as *localisation accuracy* and denoted δ^{loc}, is of the form $\frac{\sigma_a}{\sqrt{N_{phot}}}$ where N_{phot} is the mean photon count and σ_a the standard deviation of the photon detection profile, though the authors point out the expression may vary for a non-Gaussian measurement model. The interpretation for this is that the quality of location estimates of a single static molecule deteriorates as the measurement uncertainty σ_a increases. Now in Ram et al. [2013], it is shown that the limit of accuracy for the separation distance between two molecules $\delta^{static}_{D(\theta)}$ and the localisation accuracy for each of these molecules are related as follows

$$\lim_{D(\theta)\to\infty} \delta^{static}_{D(\theta)} = \sqrt{\left(\delta^{loc}_1\right)^2 + \left(\delta^{loc}_2\right)^2},$$

where δ^{loc}_1 and δ^{loc}_2 denote the localisation accuracy for the first and second molecule, respectively. This means that $\delta^{static}_{D(\theta)}$ is similarly affected by measurement uncertainty σ_a as are the localisation accuracies for the two molecules.

In this experiment, the introduction of diffusion negatively affects the improvement in estimation accuracy as the mean distance of separation between the two molecules increases. This is evidenced in Figure 6.11 by the more and more slowly decaying limits of accuracy as the value of the diffusion coefficient σ increases, and in Figure 6.13 by the increasing trend in $\hat{\delta}_{D(\theta)}$ for all values of $D(\theta)$ as σ increases. As a result, the diffusion coefficient in the dynamic model can be translated into additional observation uncertainty which affects $\delta_{D(\theta)}$ in a way reminiscent of how σ_a affects $\delta^{static}_{D(\theta)}$ in the static case. Future work will include analytical investigations with the aim of obtaining further insights and developing explicit expressions for the links between $\delta_{D(\theta)}$ and the diffusion coefficient σ which support the results of this experiment and formally generalise results from Ober et al. [2004b]; Ram et al. [2013] for dynamic molecules. This will also include analytical research into the link between $\delta_{D(\theta)}$ and the limit of accuracy for the mean location x_0 of a single dynamic molecule, which can be translated as *mean localisation accuracy* for dynamic molecules.

While the introduction of diffusion leads to less accurate estimates, Figure 6.12 displays a stronger decay in the limit of accuracy as the mean photon count N_{phot} increases, thus indicating that increasing the mean photon count N_{phot} improves those estimates. This is reinforced in Figure 6.14, which also suggests that the relationship between $\delta_{D(theta)}$ and N_{phot} is an inverse square root. This is a generalisation to the dynamic case of results in Ram et al. [2006a] which showed an inverse square root relationship between $\delta^{static}_{D(\theta)}$ and N_{phot} for two static molecules.

In summary, this experiment employs the framework developed in this chapter for estimating the FIM of parameters of dynamic molecules using SMC in order to gain insight into generalising results from Ram et al. [2006b, 2013] about the effects of separation distance, measurement uncertainty and mean photon count to a context in which the two molecules considered follow a SDE rather than being static.

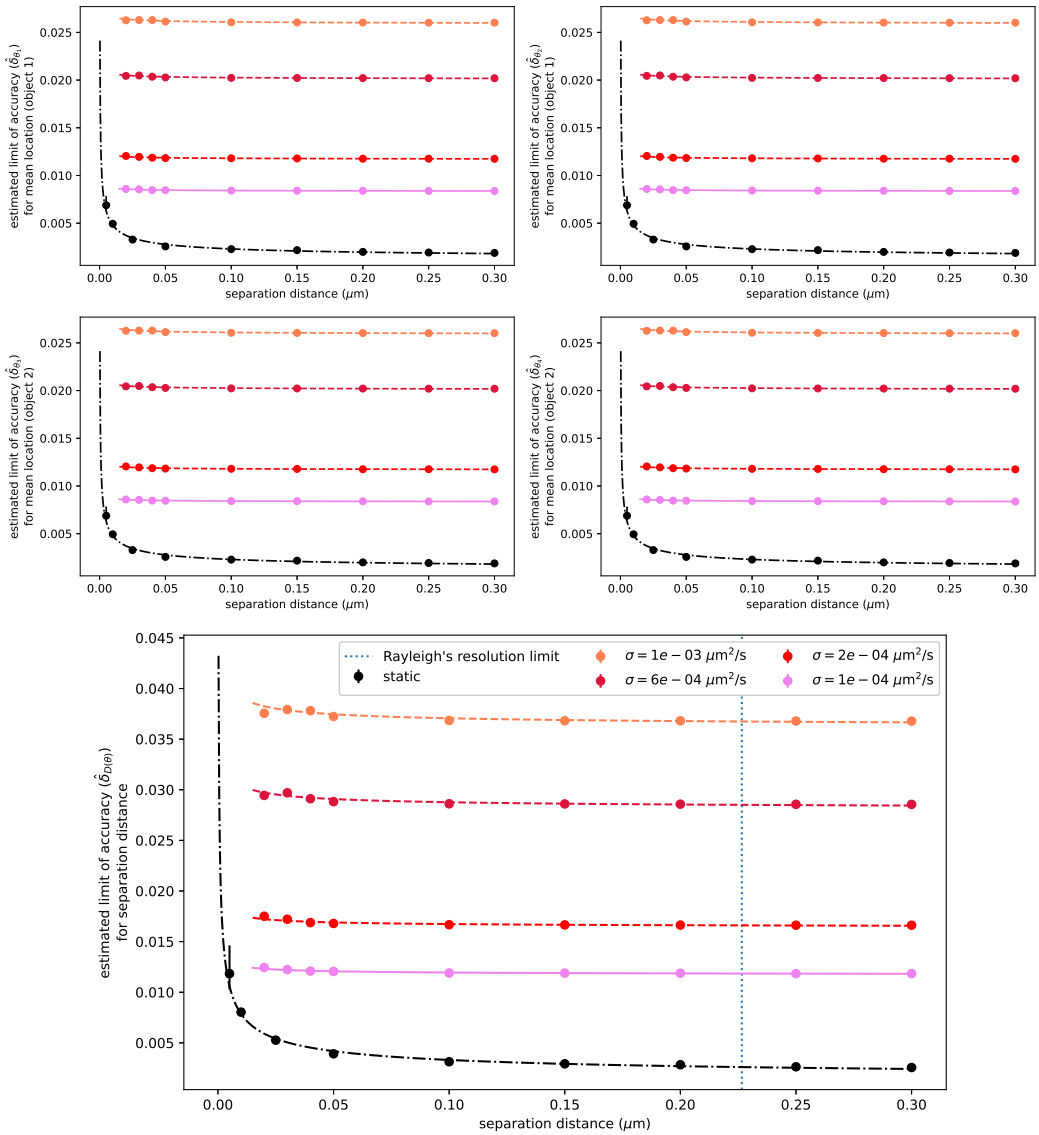


Fig. 6.11 Comparison of the evolution of the estimated limit accuracy for separation distances ranging from 20×10^{-3} to 300×10^{-3} μm for various diffusion coefficient σ values. The limit of accuracy δ_θ is estimated for the location (in the static case) or mean location (in the dynamic case) $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$ of two molecules. The estimates are obtained by taking the square root of the inverse of the FIM, obtained by estimating the OIM directly for 400 simulated datasets (in the static case) or by running the SMC-FS algorithm with 500 particles for approximately 100 simulated datasets (in the dynamic case). For the static case, the photon detection times are simulated according to a Poisson process with constant rate corresponding to the expected mean photon count $N_{phot} = 3000$ during an interval of $[0, 1]$ seconds. For the dynamic case, the molecule trajectories are initialised at their stationary distributions $\mathcal{N}(\theta_{1:2}, \sigma \mathbb{I}_{2\times 2})$ and $\mathcal{N}(\theta_{3:4}, \sigma \mathbb{I}_{2\times 2})$ and each is propagated according to its corresponding SDE (Equation 6.36) for the same time interval and mean photon count. The observations are generated according to a mixture of Airy profiles (Equation 6.38) with parameters as in Example 6.2. In the dynamic case, this is repeated for σ varying from 10^{-3} to 10^{-4} μm^2/s. The limit of accuracy for the separation distance estimate $\hat{\delta}_{D(\theta)}$ is obtained using the square root of the CRLB obtained using Equation 6.37. Finally, an inverse square root curve is fitted to each of the resulting sets of estimated limits of accuracy for comparison purposes. Note that the *pink* set of estimates and their corresponding *solid* fitted curve coincide with those in Figure 6.12.

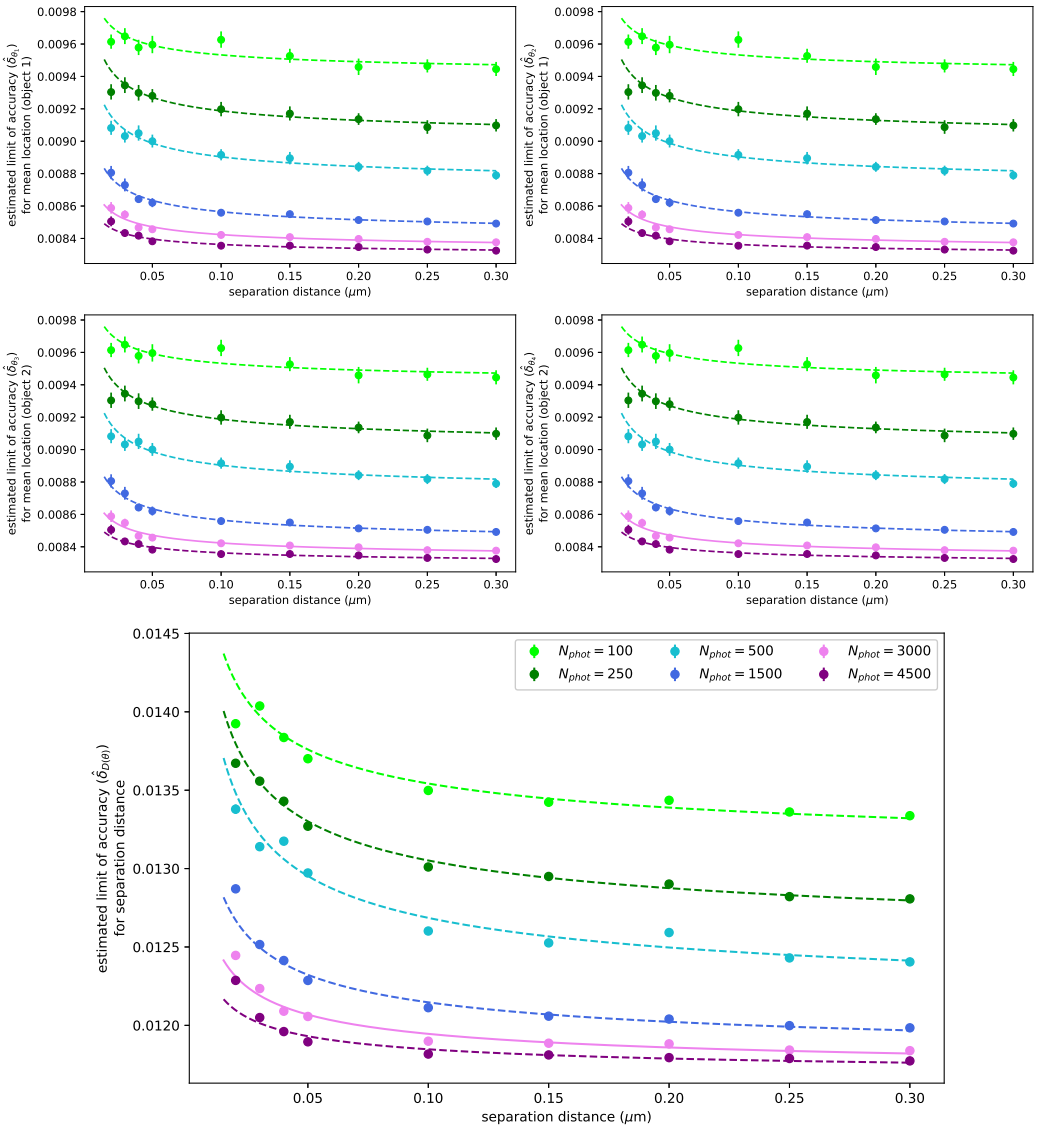


Fig. 6.12 Comparison of the evolution of the estimated limit accuracy for separation distances between two dynamic molecules ranging from 20×10^{-3} to 300×10^{-3} μm for various mean photon counts N_{phot}. The limit of accuracy δ_θ is estimated for the mean location $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$ of two molecules. The estimates are obtained by taking the square root of the inverse of the FIM, obtained by estimating the OIM using the SMC-FS algorithm with 500 particles for approximately 100 simulated datasets. The molecule trajectories are initialised at their stationary distributions $\mathcal{N}(\theta_{1:2}, \sigma\mathbb{I}_{2\times 2})$ and $\mathcal{N}(\theta_{3:4}, \sigma\mathbb{I}_{2\times 2})$ and each is propagated according to its corresponding SDE (Equation 6.36) with $\sigma = 10^{-4}$ μm^2/s during an interval of $[0, 1]$ seconds. The observations are generated according to a mixture of Airy profiles (Equation 6.38) with parameters as in Example 6.2. This is repeated for N_{phot} varying from 100 to 4500. The limit of accuracy for the separation distance estimate $\hat{\delta}_{D(\theta)}$ is obtained using the square root of the CRLB obtained using Equation 6.37. Finally, an inverse square root curve is fitted to each of the resulting sets of estimated limits of accuracy for comparison purposes. Note that the *pink* set of estimates and their corresponding *solid* fitted curve are identical to those in Figure 6.11. Any variation in estimates for low separation distances is due to Monte Carlo error, and can be reduced by increasing the number of simulated datasets.

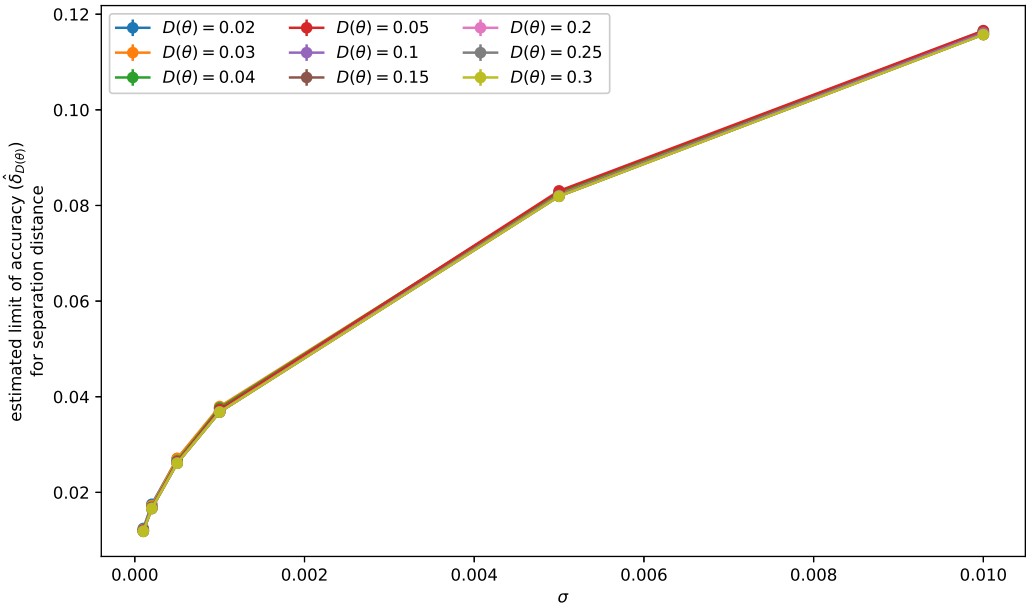


Fig. 6.13 Evolution of the estimated limit of accuracy for the separation distance $\delta_{D(\theta)}$ between two molecules for diffusion coefficient ranging from 10^{-4} to 10^{-2} μm^2/s. Estimates are obtained through the same algorithm and parameters as in Figure 6.11, with separation distances ranging from 20×10^{-3} to 300×10^{-3} μm.

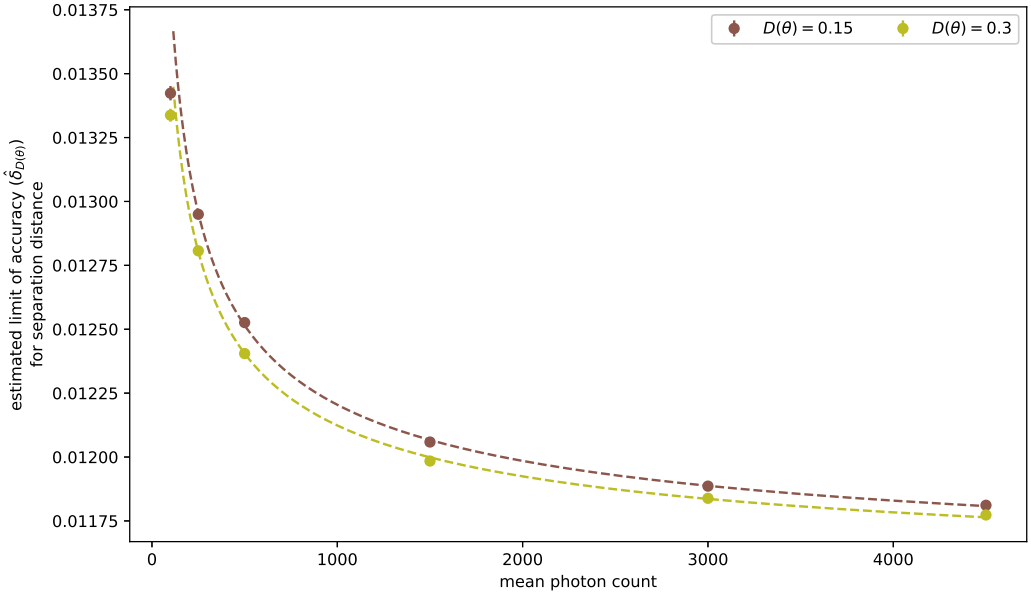


Fig. 6.14 Evolution of the estimated limit of accuracy for the separation distance $\delta_{D(\theta)}$ (obtained using Equation 6.37) between two molecules for mean photon counts ranging from 100 to 4500. Estimates are obtained through the same algorithm and parameters as in Figure 6.12, with separation distances 150×10^{-3} (*brown*) and 300×10^{-3} μm (*olive*). Inverse square root curves are fitted to the resulting estimates $\hat{\delta}_{D(\theta)}$ for comparison.

6.7 Conclusion

In this chapter, we introduced an SMC approach to performing parameter inference when tracking a molecule with stochastic trajectory for a fixed time interval. Recall the three main aspects of this type of single-molecule microscopy, namely the true location of the molecule in the object space, which follows a linear SDE, the Poisson distributed arrival process of the photons it emits on the detector in the image space, and the arrival location of those photons on the detector, which follows either a 2D Gaussian, Airy profile, or Born and Wolf model.

First of all, we discretised the time interval in order to formulate the problem as a simple state space model, in which all states are equally spaced in time, but a number of observations are marked as missing. From this, it was possible to employ particle filters to track the molecule, and particle smoothers for parameter inference, such as parameter estimation via gradient descent or EM. Most importantly, a forward smoothing algorithm was employed in order to estimate the score and OIM of the data regardless of the distribution of the photon locations. For the first time, this allowed for the estimation of the FIM and hence the limit of accuracy (square root of the CRLB), which could not be done before for the Airy profile and Born and Wolf model, and could only be achieved analytically for a specific set of photon detection times for the 2D Gaussian profile. The methodology was subsequently applied to characterise the precision limits for estimating the separation distance between two moving molecules, thus providing new insights into results for the static case from Ram et al. [2013].

Although for the first time a method has been described to estimate the limit of accuracy for the hyperparameters of dynamic single molecules with non-uniform observation times and complex measurement models, such as the Airy profile or Born and Wolf model, there is scope to use the techniques developed here to provide a wider range of more computationally efficient approaches. Indeed, an advantage of the straightforward state space model formulation of the problem is access to the vast range of filtering and smoothing algorithms available. While we employed forward smoothing, any kind of particle smoothing algorithm would be suitable, and indeed, the SMC-FS algorithm of Del Moral et al. [2010] employed for forward smoothing, even though it mitigates issues related to path degeneracy, is of $\mathcal{O}(N^2)$ complexity. It was mentioned that the PaRIS algorithm of Olsson et al. [2017] can reduce the complexity of the algorithm to linear. Further work into making the particle smoother more efficient will include investigating using the accept-reject approach (see Algorithm 3.13) in order to speed up the PaRIS algorithm, as well as employing the fixed-lag smoother. As for the model itself, the next steps include moving away from the ideal fundamental model into more complex but realistic models which take into account pixelisation and the size of the detector, as well as treating the arrival rate of photons as non-constant (see Section 6.4.1.1), thus making the photon arrival process an inhomogeneous Poisson process.

Appendix 6.A Derivation of the auxiliary density for the 2D Gaussian profile

To derive the optimal auxiliary density $p(x_k, y_k|x_{k-1})$ for the APF (Section 6.5.2.2) in the d-dimensional Gaussian case, we first state the state space model at hand.

$$X_k = \Phi_\Delta X_{k-1} + a_\Delta + W_x, \qquad W_x \sim \mathcal{N}(0, R_\Delta),$$
$$Y_k = \begin{cases} \emptyset, & \text{if no observation was recorded,} \\ MX_k + W_y, \quad W_y \sim \mathcal{N}(0,\Sigma), & \text{otherwise,} \end{cases}$$

where $\Sigma = M^{\mathrm{T}}\Sigma_a M$ and $\Sigma_a = \sigma_a \mathbb{I}_{d\times dd}$. Recall that the optimal auxiliary density is given by

$$p_\theta(x_k, y_k|x_{k-1}) = p_\theta(x_k|y_k, x_{k-1}) p_\theta(y_k|x_{k-1}).$$

Their respective probability density functions are, for dimension n,

$$f_\Delta^\theta(x_k|x_{k-1}) = (2\pi)^{-\frac{d}{2}} |R_\Delta|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(x_k - \Phi_\Delta x_{k-1} - a_\Delta)^{\mathrm{T}} R_\Delta^{-1}(x_k - \Phi_\Delta x_{k-1} - a_\Delta)\right],$$
$$G_k^\theta(x_k) = \begin{cases} 1 - \Delta\lambda, & \text{if } y_k = \emptyset, \\ \Delta\lambda (2\pi)^{-\frac{d}{2}} |\Sigma|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(y_k - Mx_k)^{\mathrm{T}}\Sigma^{-1}(y_k - Mx_k)\right], & \text{otherwise.} \end{cases}$$

We first consider the situation in which no observation was recorded in the k-th discrete interval. It is straightforward to see that

$$p_\theta(x_k, y_k|x_{k-1}) = f_\Delta^\theta(x_k|x_{k-1})(1 - \lambda\Delta).$$

Now, if instead an observation was recorded, we have

$$p(x_k, y_k|x_{k-1}) \propto \exp\left[-\frac{1}{2}(x_k - \Phi_\Delta x_{k-1} - a_\Delta)^{\mathrm{T}} R_\Delta^{-1}(x_k - \Phi_\Delta x_{k-1} - a_\Delta)\right] \exp\left[-\frac{1}{2}(y_k - Mx_k)^{\mathrm{T}}\Sigma^{-1}(y_k - Mx_k)\right].$$

Taking the log and multiplying by -2 for simplicity, if $\doteq$ denotes equality up to an additive constant, we obtain

$$
\begin{aligned}
&-2\log p(x_k, y_k|x_{k-1}) \\
&\doteq x_k^{\mathrm{T}} \underbrace{(R_\Delta^{-1} + M^{\mathrm{T}}\Sigma^{-1}M)}_{\Sigma_\Delta^{-1}} x_k - 2x_k^{\mathrm{T}}(R_\Delta^{-1}(\Phi_\Delta x_{k-1} + a_\Delta) + M^{\mathrm{T}}\Sigma^{-1}y_k) \\
&+ (\Phi_\Delta x_{k-1} + a_\Delta)^{\mathrm{T}} R_\Delta^{-1}(\Phi_\Delta x_{k-1} + a_\Delta) + y_k^{\mathrm{T}}\Sigma^{-1}y_k \\
&= x_k^{\mathrm{T}}\Sigma_\Delta^{-1}x_k - 2x_k^{\mathrm{T}}\Sigma_\Delta^{-1}\underbrace{\Sigma_\Delta(R_\Delta^{-1}(\Phi_\Delta x_{k-1} + a_\Delta) + M^{\mathrm{T}}\Sigma^{-1}y_k)}_{\mu_\Delta} + \mu_\Delta^{\mathrm{T}}\Sigma_\Delta^{-1}\mu_\Delta \\
&- (R_\Delta^{-1}(\Phi_\Delta x_{k-1} + a_\Delta) + M^{\mathrm{T}}\Sigma^{-1}y_k)^{\mathrm{T}}\Sigma_\Delta(R_\Delta^{-1}(\Phi_\Delta x_{k-1} + a_\Delta) + M^{\mathrm{T}}\Sigma^{-1}y_k) \\
&+ (\Phi_\Delta x_{k-1} + a_\Delta)^{\mathrm{T}} R_\Delta^{-1}(\Phi_\Delta x_{k-1} + a_\Delta) + y_k^{\mathrm{T}}\Sigma^{-1}y_k \\
&= (x_k - \mu_\Delta)^{\mathrm{T}}\Sigma_\Delta^{-1}(x_k - \mu_\Delta) + y_x^{\mathrm{T}}\underbrace{(\Sigma^{-1} - \Sigma^{-1}M\Sigma_\Delta M^{\mathrm{T}}\Sigma^{-1})}_{\Xi_\Delta^{-1}} y_x \\
&- 2(\Phi_\Delta x_{k-1} + a_\Delta)^{\mathrm{T}}(R_\Delta^{-1}\Sigma_\Delta M^{\mathrm{T}}\Sigma^{-1}) + (\Phi_\Delta x_{k-1} + a_\Delta)^{\mathrm{T}}(R_\Delta^{-1} - R_\Delta^{-1}\Sigma_\Delta R_\Delta^{-1})(\Phi_\Delta x_{k-1} + a_\Delta).
\end{aligned}
$$

Therefore, the auxiliary density is given by

$$
p(x_k, y_k|x_{k-1}) \propto \exp\left[-\frac{1}{2}(x_k - \mu_\Delta)^{\mathrm{T}}\Sigma_\Delta^{-1}(x_k - \mu_\Delta)\right] \exp\left[-\frac{1}{2}(y_k - \xi_\Delta)^{\mathrm{T}}\Xi_\Delta^{-1}(y_k - \xi_\Delta)\right],
$$

where plugging in $\Sigma = M\Sigma_a M^{\mathrm{T}}$, the parameters of its components are

$$
\begin{aligned}
\mu_\Delta &= (R_\Delta^{-1} + \Sigma_a^{-1})^{-1}(R_\Delta^{-1}(\Phi_\Delta x_{k-1} + a_\Delta) + \Sigma_a^{-1}M^{-1}y_k), \\
\Sigma_\Delta &= (R_\Delta^{-1} + \Sigma_a^{-1})^{-1}, \\
\xi_\Delta &= M(\Phi_\Delta x_{k-1} + a_\Delta), \\
\Xi_\Delta &= M(\Sigma_a + R_\Delta)M^{\mathrm{T}}.
\end{aligned}
$$

Appendix 6.B Sufficient statistic for estimating the OIM by forward smoothing

In Example 6.4, recall that the molecule trajectory is described by the following SDE in d-dimensional space

$$\mathrm{d}X_t = b\mathbb{I}_{d\times d}X_t\mathrm{d}t + \sqrt{2\sigma}\mathrm{d}B_t,$$

where in the drift term, $b \neq 0$, in the diffusion term, $\sigma > 0$, and $(\mathrm{d}B_t)_{t_0\leq t\leq T}$ is a Wiener process. The log transition density can be written as

$$\log f_\Delta^\theta(x_k|x_{k-1}) = -\frac{d}{2}\log(2\pi\sigma) + \frac{d}{2}\log(b) - \frac{d}{2}\log\left(e^{2\Delta b}-1\right) - \frac{b\left\|x_k - e^{\Delta b}x_{k-1}\right\|^2}{2\sigma(e^{2\Delta b}-1)}. \tag{6.39}$$

To obtain the sufficient statistics in Equations 6.28 and 6.29, if the photon location process is distributed according to the Airy or 2D Gaussian profiles, it suffices to take the gradient and hessian of the log transition density in Equation 6.39 with respect to the diffusion σ and drift b coefficients, i.e.

$$\nabla\log f_\Delta^\theta(x_k|x_{k-1}) = \begin{pmatrix}\mathrm{g}_1\\ \mathrm{g}_2\end{pmatrix}, \qquad \nabla^2\log f_\Delta^\theta(x_k|x_{k-1}) = \begin{pmatrix}\mathrm{H}_{11} & \mathrm{H}_{12}\\ \mathrm{H}_{21} & \mathrm{H}_{22}\end{pmatrix},$$

where

- Gradient w.r.t σ

$$\mathrm{g}_1 := -\frac{d}{2\sigma} + \frac{b\left\|x_k - e^{b\Delta}x_{k-1}\right\|^2}{2\sigma^2\left(e^{2b\Delta}-1\right)}.$$

- Gradient w.r.t b

$$\begin{aligned}\mathrm{g}_2 := \frac{d}{2b} - \frac{d\Delta e^{2\Delta b}}{\left(e^{2\Delta b}-1\right)} - \frac{\left\|x_k - e^{\Delta b}x_{k-1}\right\|^2}{2\sigma(e^{2\Delta b}-1)}\\ + \frac{\Delta b e^{\Delta b}(x_k - e^{\Delta b}x_{k-1})^{\mathrm{T}}x_{k-1}}{\sigma(e^{2\Delta b}-1)} + \frac{\left\|x_k - e^{\Delta b}x_{k-1}\right\|^2\Delta b e^{2\Delta b}}{\sigma(e^{2\Delta b}-1)^2}.\end{aligned}$$

- Hessian w.r.t σ then σ

$$\mathsf{H}_{11} := \frac{d}{2\sigma^2} - \frac{b\left\|x_k - e^{b\Delta}x_{k-1}\right\|^2}{\sigma^3\left(e^{2b\Delta}-1\right)}.$$

- Hessian w.r.t b then σ and vice versa

$$\begin{aligned}\mathsf{H}_{12} = \mathsf{H}_{21} := &\frac{\left\|x_k - e^{\Delta b}x_{k-1}\right\|^2}{2\sigma^2(e^{2\Delta b}-1)} \\ &- \frac{\Delta b e^{\Delta b}(x_k - e^{\Delta b}x_{k-1})^{\mathrm{T}}x_{k-1}}{\sigma^2(e^{2\Delta b}-1)} - \frac{\left\|x_k - e^{\Delta b}x_{k-1}\right\|^2 \Delta b e^{2\Delta b}}{\sigma^2(e^{2\Delta b}-1)^2}.\end{aligned}$$

- Hessian w.r.t b then b

$$\begin{aligned}\mathsf{H}_{22} := &-\frac{d}{2b^2} - \frac{2d\Delta^2 e^{2\Delta b}}{e^{2\Delta b}-1} + \frac{2d\Delta^2 e^{4\Delta b}}{\left(e^{2\Delta b}-1\right)^2} \\ &+ \frac{x_k^{\mathrm{T}}x_k}{\sigma}\left[\frac{2\Delta e^{2\Delta b} + 2\Delta^2 b e^{2\Delta b}}{(e^{2\Delta b}-1)^2} - \frac{4\Delta^2 b e^{4\Delta b}}{(e^{2\Delta b}-1)^3}\right] \\ &+ \frac{x_k^{\mathrm{T}}x_{k-1}}{\sigma}\left[\frac{2\Delta e^{\Delta b} + \Delta^2 b e^{\Delta b}}{e^{2\Delta b}-1} - \frac{4\Delta e^{3\Delta b} + 8\Delta^2 b e^{3\Delta b}}{(e^{2\Delta b}-1)^2} + \frac{8\Delta^2 b e^{5\Delta b}}{(e^{2\Delta b}-1)^3}\right] \\ &- \frac{x_{k-1}^{\mathrm{T}}x_{k-1}}{\sigma}\left[\frac{2\Delta e^{2\Delta b} + 2\Delta^2 b e^{2\Delta b}}{e^{2\Delta b}-1} - \frac{2\Delta e^{4\Delta b} + 6\Delta^2 b e^{4\Delta b}}{(e^{2\Delta b}-1)^2} + \frac{4\Delta^2 b e^{6\Delta b}}{(e^{2\Delta b}-1)^3}\right].\end{aligned}$$

Appendix 6.C Score and OIM for static molecule observed via Airy profile

In Example 6.5, we consider the problem of estimating the FIM for the location parameters (x_1, x_2) of an in-focus static molecule. This is achieved by computing the score and OIM for the observed data. If the photon detection locations are described by the 2D Gaussian profile, the differentiation is straightforward, but in the case of the Airy profile (Equation 6.8), the computations are more involved.

Given observation $y \in \mathbb{R}^2$ and invertible lateral magnification matrix $M \in \mathbb{R}^{2\times 2}$, for notational simplicity let $v = M^{-1}y$, $r = \sqrt{(v_1 - x_1)^2 + (v_2 - x_2)^2}$ and $\alpha = \frac{2\pi n_\alpha}{\lambda_e}$. The log photon distribution profile (Equation 6.7) is given by

$$\log g(y) = -\log(|M|) + \log q(y),$$

where the image function is

$$q(y) = \frac{J_1^2(\alpha r)}{\pi r^2}.$$

First of all, use the relation $\frac{\partial}{\partial x} x^{-n} J_n(x) = -x^{-n} J_{n+1}(x)$ for $n \in \mathbb{N}$ in order to obtain the gradient and hessian of $q(y)$. Where the subscript i appears, the result is valid for $i = 1, 2$.

$$\begin{aligned}
\frac{\partial q(y)}{\partial x_i} &= \frac{2\alpha}{\pi}(v_i - x_i)\frac{J_1(\alpha r)}{r}\frac{J_2(\alpha r)}{r^2}, \\
\frac{\partial^2 q(y)}{\partial x_i^2} &= \frac{2\alpha^2}{\pi r^4}(v_i - x_i)^2\left[J_1(\alpha r)J_3(\alpha r) + J_2^2(\alpha r)\right] - \frac{2\alpha}{\pi}\frac{J_1(\alpha r)}{r}\frac{J_2(\alpha r)}{r^2}, \\
\frac{\partial^2 q(y)}{\partial x_1 \partial x_2} &= \frac{2\alpha^2}{\pi r^4}(v_1 - x_1)(v_2 - x_2)\left[J_1(\alpha r)J_3(\alpha r) + J_2^2(\alpha r)\right].
\end{aligned}$$

To derive the components of the gradient and hessian of $\log q(y)$, we make use of the following identities

$$\nabla \log q(y) = \frac{\nabla q(y)}{q(y)}, \qquad \nabla^2 \log q(y) = \frac{\nabla^2 q(y)}{q(y)} - [\nabla \log q(y)]^2.$$

Therefore, for $i = 1, 2$, the components of the log gradient are given by

$$\frac{\partial \log q(y)}{\partial x_i} = \frac{2\alpha}{r}\frac{J_2(\alpha r)}{J_1(\alpha r)}(v_i - x_i),$$

and the diagonal components of the log hessian are

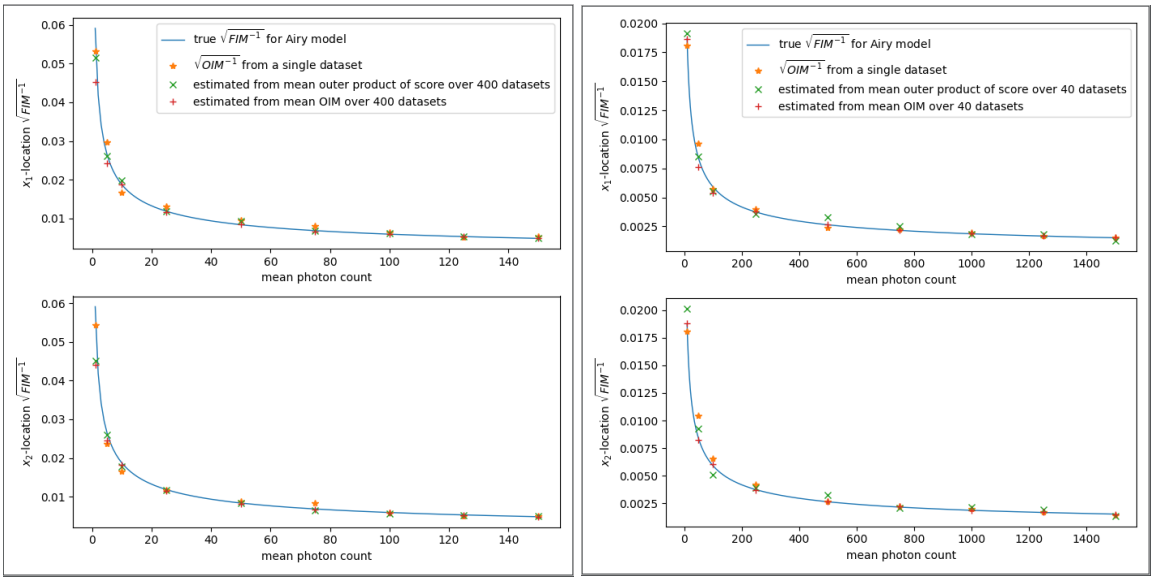
$$\frac{\partial[\log q(y)]^2}{\partial x_i^2} = \frac{2\alpha^2}{r^2}(v_i - x_i)^2\left[\frac{J_3(\alpha r)}{J_1(\alpha r)} - \frac{J_2^2(\alpha r)}{J_1^2(\alpha r)}\right] - \frac{2\alpha}{r}\frac{J_2(\alpha r)}{J_1(\alpha r)}.$$

And finally, the cross terms are given by

$$\frac{\partial[\log q(y)]^2}{\partial x_1 x_2} = \frac{2\alpha^2}{r^2}(v_1 - x_1)(v_2 - x_2)\left[\frac{J_3(\alpha r)}{J_1(\alpha r)} - \frac{J_2^2(\alpha r)}{J_1^2(\alpha r)}\right].$$

To summarise, the log gradient and negative log hessian for the Airy profile are

$$\nabla \log g(y) = \gamma(M^{-1}y - x), \qquad \gamma = \frac{2\alpha}{r}\frac{J_2(\alpha r)}{J_1(\alpha r)},$$
$$-\nabla^2 \log g(y) = \chi(M^{-1}y - x)(M^{-1}y - x)^\intercal + \gamma\mathbb{I}_{2\times 2}, \qquad \chi = -\frac{2\alpha^2}{r^2}\left[\frac{J_3(\alpha r)}{J_1(\alpha r)} - \frac{J_2^2(\alpha r)}{J_1^2(\alpha r)}\right].$$



(a) Many short datasets, Airy profile

(b) Few long datasets, Airy profile

Fig. 6.15 True and estimated limit of accuracy for mean photon counts ranging from (a) 1 to 150 (b) 10 to 1500. The limit of accuracy is estimated for the location parameters (x_1, x_2) of a static in-focus molecule. The estimates are obtained by taking the square root of the inverse of the FIM, obtained for (a) 400 'short' and (b) 40 'long' simulated datasets using approaches (i) $\star$, (ii) $\times$ and (iii) $+$ for comparison purposes. To generate each dataset, the photon detection times are simulated according to a Poisson process with rate corresponding to the expected mean photon count for (a) $[0, 0.02]$ and (b) $[0, 0.2]$ seconds and the intervals are discretised. The photon detection locations are generated according to the Airy profile, with parameters as in Example 6.2. The true limit of accuracy (*blue solid line*) is also computed as it is available analytically. Similarly as for the 2D Gaussian profile, estimates of the limit of accuracy based on a single dataset (approach *(i)*) are more accurate for long datasets while taking the mean outer product of the score over all datasets (approach *(ii)*) yields more accurate estimates for a higher number of short datasets. Approach *(iii)* provides a good balance between the two.

Chapter 7

Conclusions

The research in this thesis presented new and efficient Monte Carlo methods for Bayesian inference with the aim of tackling particularly complex models, and a significant portion of the thesis has been geared towards increasing the computational efficiency of these algorithms for use in parallel computing architectures. Additionally, there is scope to further explore the novel methods and algorithms presented in this thesis, and to employ them in a wide range of applications and computing environments. We summarise the contributions of this thesis and provide some insights into future directions.

7.1 Thesis contributions

In Chapter 4, we developed a novel algorithm known as Anytime Parallel Tempering Monte Carlo (APTMC) which combines the features of parallel tempering and Murray et al. [2016b]'s Anytime Monte Carlo framework. This framework is particularly useful for running algorithms in which local moves take a random and varying time to complete in a distributed computing setting, as it significantly reduces idling and improves performance. As part of this chapter, we applied the framework in an approximate Bayesian computation (ABC) setting, making use of Lee [2012]'s efficient 1-hit MCMC kernel, for which we developed a new kind of exchange moves in order to adapt the algorithm to parallel tempering.

Our contributions in Chapter 5 included developing a new and general Anytime SMC sampler which makes use of RJ-MCMC to perform changepoint inference, where the Anytime framework ensures any idling between processors remains minimal. A general framework for performing changepoint inference with this algorithm was described and will be helpful in the future for particularly complex changepoint problems. This was illustrated with a complex model by Fearnhead and Vasileiou [2009] in which the model or family of each

segment must also be estimated as part of the RJ-MCMC updates. We also derived the various RJ-MCMC updates for this particular model as they did not exist yet. Previously, Anytime SMC samplers had been presented in Murray et al. [2016b], as well as SMC samplers which employ RJ-MCMC for changepoint inference in Del Moral et al. [2006], and RJ-MCMC algorithms for similar, complex applications in Boys and Henderson [2004]. In Chapter 5, all these elements were combined into a single algorithm for the first time.

Chapter 6 first formally introduced a new approach to parameter inference in single-molecule microscopy, which had only been touched upon before in Ashley and Andersson [2015]. This approach discretises the observation interval of photons emitted by a stochastically moving molecule arriving on a detector. This meant that the model could be formulated as a straightforward state space model, so that any particle filtering and smoothing algorithm could be applied to it for tracking and parameter inference purposes. As a result, the second major contribution of this chapter was the framework developed for estimating the FIM, and hence its inverse the CRLB, for complex photon detection location profiles such as the Airy profile and Born and Wolf model, and for a molecule whose trajectory is described by an SDE, which could not be done before. A consequence of this was that it enabled us to observe the inverse square root decay of the limit of accuracy (square root of the CRLB) of various hyperparameters such as the drift and diffusion coefficients of the SDE as the expected photon count increases. Finally, the methodology was applied to estimate the limit of accuracy for estimating the separation distance between two moving molecules, thus providing new insights into results for the static case from Ram et al. [2013].

7.2 Future directions

In Chapter 4, we saw that the fact that there is no upper limit on the time the 1-hit ABC kernel race takes to complete can lead to standard ABC algorithms being stuck for extended periods. The Lotka-Volterra predator-prey model considered was particularly affected by this issue. As a result, the improvements in effective sample size brought by adding exchange moves were only modest. This was less the case for the moving average example of Section 4.5.3, but it remains a model in which the likelihood is available anyway, if expensive for large datasets. In this case, scalable approaches such as pseudo-marginal MCMC or subsampling (see Section 2.4.2) are able to improve performance without needing to resort to ABC. Future work on Anytime parallel tempering involving ABC could include finding applications which require ABC and will strongly benefit from exchange moves, such as a problem with an intractable but multimodal posterior

In Chapter 5, the RJ-MCMC Anytime SMC sampler for changepoint inference was

presented in a general fashion, and the illustrated problem could have been implemented in a way that didn't benefit as much from Anytime. However, the framework presented should be of use in the future for anyone encountering a particularly challenging real-life changepoint problem, where for example it is impossible to collapse out the latent parameters from the likelihood and/or completion times of RJ-MCMC moves differ widely between the various numbers of changepoints.

More generally, the Anytime framework can be employed in several other applications, whether it be Anytime parallel tempering, an Anytime SMC sampler or a new algorithm construction. Other algorithms whose completion times depend on the current state of the chain include the No-U-Turn sampler (NUTS) (Hoffman and Gelman [2014]), elliptical slice sampling (Murray et al. [2010]), and the accept-reject sampling scheme of Algorithm 3.13.

Future work that will be carried out, building on Chapter 6, will involve moving away from the convenient fundamental model into more realistic models which take into account pixelation and finite-sized detectors (Vahid et al. [2019]). Another important aspect will be to deal with potentially inhomogeneous photon detection rates, as outlined in Section 6.4.1.1. Future work will also include investigating and comparing the efficiency of different particle filters and smoothers for estimating the FIM. Indeed, in the scenarios considered in the experiments, particle path degeneracy must be addressed, but in order to obtain an accurate estimate of the FIM, the particle smoother must be run several times, so it would be beneficial for its complexity to be only linear. New particle smoothers to consider are the fixed-lag smoother (Section 3.5.1) and the PaRIS algorithm with accept-reject sampling.

References

Aminikhanghahi, S. and Cook, D. J. (2017). A survey of methods for time series change point detection. *Knowledge and information systems*, 51(2):339–367. 5.2

Andrieu, C., Roberts, G. O., et al. (2009). The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37(2):697–725. 2.4.2

Ansley, C. F. and Kohn, R. (1982). A geometrical derivation of the fixed interval smoothing algorithm. *Biometrika*, 69(2):486–487. 3.2.2

Ashley, T. T. and Andersson, S. B. (2015). Method for simultaneous localization and parameter estimation in particle tracking experiments. *Physical Review E*, 92(5):052707. 6.2, 7.1

Atchadé, Y. F., Roberts, G. O., and Rosenthal, J. S. (2011). Towards optimal scaling of Metropolis-coupled Markov chain Monte Carlo. *Statistics and Computing*, 21(4):555–568. 4.3.4

Baragatti, M., Grimaud, A., and Pommeret, D. (2013). Likelihood-free parallel tempering. *Statistics and Computing*, 23(4):535–549. 4.2

Bardenet, R., Doucet, A., and Holmes, C. (2014). An adaptive subsampling approach for MCMC inference in large datasets. In *Proceedings of The 31st International Conference on Machine Learning*, pages 405–413. 2.4.2

Bardenet, R., Doucet, A., and Holmes, C. (2015). On Markov chain Monte Carlo methods for tall data. *arXiv preprint arXiv:1505.02827*. 2.2, 2.4.2

Bardenet, R., Doucet, A., and Holmes, C. (2017). On markov chain monte carlo methods for tall data. *The Journal of Machine Learning Research*, 18(1):1515–1557. 2.4

Barry, D. and Hartigan, J. A. (1992). Product partition models for change point problems. *The Annals of Statistics*, pages 260–279. 5.4.1

Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171. 3.2.1, 5.6.3.3

Beaumont, M. A., Cornuet, J.-M., Marin, J.-M., and Robert, C. P. (2009). Adaptive approximate Bayesian computation. *Biometrika*, 96(4):983–990. 2.7

Beaumont, M. A., Zhang, W., and Balding, D. J. (2002). Approximate Bayesian Computation in Population Genetics. *Genetics*, 162(4):2025–2035. 4.5.3.2

Benson, A., Friel, N., et al. (2018). Adaptive MCMC for multiple changepoint analysis with applications to large datasets. *Electronic Journal of Statistics*, 12(2):3365–3396. 5.2, 5.3.1, 5.3.2, 5.4.1, 5.4.3, 5.6.2

Berglund, A. J. (2010). Statistics of camera-based single-particle tracking. *Physical Review E*, 82(1):011917. 6.2

Beskos, A., Roberts, G., Stuart, A., et al. (2009). Optimal scalings for local metropolis–hastings chains on nonproduct targets in high dimensions. *The Annals of Applied Probability*, 19(3):863–898. 4.2

Betancourt, M. (2017). A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*. 2.4.2

Bickel, P. J., Ritov, Y., Ryden, T., et al. (1998). Asymptotic normality of the maximum-likelihood estimator for general hidden Markov models. *The Annals of Statistics*, 26(4):1614–1635. 6.5.5.1

Blum, M. G. and Tran, V. C. (2010). HIV with contact tracing: a case study in approximate Bayesian computation. *Biostatistics*, 11(4):644–660. 2.7

Born, M. and Wolf, E. (2013). *Principles of optics: electromagnetic theory of propagation, interference and diffraction of light*. Elsevier. 1.3, 6.2, 6.3.2, 6.3.2, 6.6.3

Botev, Z. I., Grotowski, J. F., Kroese, D. P., et al. (2010). Kernel density estimation via diffusion. *The Annals of Statistics*, 38(5):2916–2957. 4.5.1.3

Boys, R. J. and Henderson, D. (2002). On determining the order of Markov dependence of an observed process governed by a hidden Markov model. *Scientific Programming*, 10(3):241–251. 5.4.3

Boys, R. J. and Henderson, D. A. (2004). A Bayesian approach to DNA sequence segmentation. *Biometrics*, 60(3):573–581. 5.2, 5.4.3, 5.7, 7.1

Boys, R. J., Henderson, D. A., and Wilkinson, D. J. (2000). Detecting homogeneous segments in DNA sequences by using hidden Markov models. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 49(2):269–285. 3.2

Boys, R. J., Wilkinson, D. J., and Kirkwood, T. B. (2008). Bayesian inference for a discretely observed stochastic kinetic model. *Statistics and Computing*, 18(2):125–135. 4.5.4

Briane, V., Kervrann, C., and Vimond, M. (2018). Statistical analysis of particle trajectories in living cells. *Physical Review E*, 97(6):062121. 6.2

Briers, M., Doucet, A., and Maskell, S. (2010). Smoothing algorithms for state–space models. *Annals of the Institute of Statistical Mathematics*, 62(1):61. 3.2

Brooks, S. P., Giudici, P., and Roberts, G. O. (2003). Efficient construction of reversible jump markov chain Monte Carlo proposal distributions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(1):3–39. 2.8, 5.2, 5.5

Byron, M. Y., Shenoy, K. V., and Sahani, M. (2004). Derivation of kalman filtering and smoothing equations. In *Technical report*. Stanford University. 3.2.2, 3.2.2

Calderhead, B. and Girolami, M. (2009). Estimating Bayes factors via thermodynamic integration and population MCMC. *Computational Statistics & Data Analysis*, 53(12):4028–4045. 4.2, 4.6

Calderon, C. P. (2016). Motion blur filtering: a statistical approach for extracting confinement forces and diffusivity from a single blurred trajectory. *Physical Review E*, 93(5):053303. 6.2

Calderon, C. P. and Bloom, K. (2015). Inferring latent states and refining force estimates via hierarchical dirichlet process modeling in single particle tracking experiments. *PloS one*, 10(9). 6.2

Calderon, C. P., Thompson, M. A., Casolari, J. M., Paffenroth, R. C., and Moerner, W. (2013). Quantifying transient 3d dynamical phenomena of single mrna particles in live yeast cell measurements. *The Journal of Physical Chemistry B*, 117(49):15701–15713. 6.2

Cappé, O. (2001). Ten years of HMMs. *None*. 3.2

Cappé, O., Godsill, S. J., and Moulines, E. (2007). An overview of existing methods and recent advances in sequential Monte Carlo. *Proceedings of the IEEE*, 95(5):899–924. 3.3.2

Cappé, O., Moulines, E., and Rydén, T. (2006). *Inference in hidden Markov models*. Springer Science & Business Media. 3.3, 6.2, 6.5.4.1

Carmi, A., Septier, F., and Godsill, S. J. (2012). The gaussian mixture MCMC particle algorithm for dynamic cluster tracking. *Automatica*, 48(10):2454–2467. 3.2

Carpenter, J., Clifford, P., and Fearnhead, P. (1999). Improved particle filter for nonlinear problems. *IEE Proceedings-Radar, Sonar and Navigation*, 146(1):2–7. 3.3.5, 6.5.2.1

Cauchy, A. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538. 6.5.6.1

Chao, J., Ward, E. S., and Ober, R. J. (2016). Fisher information theory for parameter estimation in single molecule microscopy: tutorial. *JOSA A*, 33(7):B36–B57. 6.2, 6.3, 6.3.3, 6.5

Cheezum, M. K., Walker, W. F., and Guilford, W. H. (2001). Quantitative comparison of algorithms for tracking single fluorescent particles. *Biophysical journal*, 81(4):2378–2388. 6.3.2

Chib, S. (1998). Estimation and comparison of multiple change-point models. *Journal of econometrics*, 86(2):221–241. 5.2

Chopin, N. (2002). A sequential particle filter method for static models. *Biometrika*, 89(3):539–552. 2.5

Chopin, N. and Papaspiliopoulos, O. (2020). *An introduction to sequential Monte Carlo*. Springer. 3.3

Chowdhury, M. F. R., Selouani, S.-A., and O'Shaughnessy, D. (2012). Bayesian on-line spectral change point detection: a soft computing approach for on-line asr. *International Journal of Speech Technology*, 15(1):5–23. 5.2

Cramér, H. (1999). *Mathematical methods of statistics*, volume 43. Princeton university press. 6.2, 6.5.5

Dange, T., Grunwald, D., Grunwald, A., Peters, R., and Kubitscheck, U. (2008). Autonomy and robustness of translocation through the nuclear pore complex: a single-molecule study. *The Journal of cell biology*, 183(1):77–86. 6.2

Darmois, G. (1945). Sur les limites de la dispersion de certaines estimations. *Revue de l'Institut International de Statistique*, pages 9–15. 6.2, 6.5.5

DeGroot, M. H. and Schervish, M. J. (2012). *Probability and statistics*. Pearson Education. 6.5.5

Del Moral, P. and Doucet, A. (2003). On a class of genealogical and interacting Metropolis models. In *Séminaire de Probabilités XXXVII*, pages 415–446. Springer. 3.3.3

Del Moral, P., Doucet, A., and Jasra, A. (2006). Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436. 2.5, 2.8, 4.2, 4.6, 5.2, 5.3.2, 5.4.3, 5.5, 5.5.1, 5.6.5, 5.7, 7.1

Del Moral, P., Doucet, A., and Jasra, A. (2012a). An adaptive sequential Monte Carlo method for approximate Bayesian computation. *Statistics and Computing*, 22(5):1009–1020. 2.7

Del Moral, P., Doucet, A., Jasra, A., et al. (2012b). On adaptive resampling strategies for sequential Monte Carlo methods. *Bernoulli*, 18(1):252–278. 3.3.3

Del Moral, P., Doucet, A., and Singh, S. (2010). Forward smoothing using sequential Monte Carlo. *arXiv preprint arXiv:1012.5390*. 3.5.2, 5.6.3.3, 6.2, 6.5.3, 6.7

Deligiannidis, G., Doucet, A., and Pitt, M. K. (2018). The correlated pseudomarginal method. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 80(5):839–870. 2.4.2

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22. 6.5.6.2

Denison, D., Mallick, B., and Smith, A. (1998). Automatic Bayesian curve fitting. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 60(2):333–350. 2.8

Deschout, H., Zanacchi, F. C., Mlodzianoski, M., Diaspro, A., Bewersdorf, J., Hess, S. T., and Braeckmans, K. (2014). Precisely and accurately localizing single emitters in fluorescence microscopy. *Nature methods*, 11(3):253. 6.2

Dias, A. and Embrechts, P. (2002). Change-point analysis for dependence structures in nance and insurance. *Novos Rumos em Estatistica (Ed. C. Carvalho, F. Brilhante and F. Rosado), Sociedade Portuguesa de Estatstica, Lisbon*, pages 69–86. 5.2

Douc, R., Garivier, A., Moulines, E., Olsson, J., et al. (2011). Sequential Monte Carlo smoothing for general state space hidden Markov models. *The Annals of Applied Probability*, 21(6):2109–2145. 6.5.3

Douc, R., Moulines, E., and Stoffer, D. (2014). *Nonlinear time series: Theory, methods and applications with R examples*. CRC press. 1.1.2, 3.2, 3.2.1, 3.2.2, 3.3, 3.4.1, 6.5.4.1

Doucet, A., de Freitas, N., and Gordon, N. (2001). *Sequential Monte Carlo Methods in Practice*. Springer Science & Business Media. 3.2, 3.3, 3.3.4

Doucet, A., Godsill, S., and Andrieu, C. (2000). On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and computing*, 10(3):197–208. 3.3.3, 3.4.2, 6.5.3

Doucet, A. and Johansen, A. M. (2009). A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3. 2.5, 3.3.3, 6.2

Drovandi, C. C. and Pettitt, A. N. (2011). Likelihood-free Bayesian estimation of multivariate quantile distributions. *Computational Statistics & Data Analysis*, 55(9):2541–2556. 2.7

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222. 2.4.2

Duchi, J. (2016). Lecture notes for statistics 311/electrical engineering 377. *URL: https://stanford. edu/class/stats311/Lectures/full_notes. pdf*, 2:23. 6.5.5

Earl, D. J. and Deem, M. W. (2004). Optimal allocation of replicas to processors in parallel tempering simulations. *The Journal of Physical Chemistry B*, 108(21):6844–6849. 4.2, 4.6

Earl, D. J. and Deem, M. W. (2005). Parallel tempering: Theory, applications, and new perspectives. *Physical Chemistry Chemical Physics*, 7(23):3910–3916. 1.2, 2.4.1

Eckhardt, R. (1987). Stan Ulam, John von Neumann, and the Monte Carlo method. *Los Alamos Science*, (15):131. 1.1, 1.1.1

Eckley, I. A., Fearnhead, P., and Killick, R. (2011). Analysis of changepoint models. *Bayesian Time Series Models*, pages 205–224. 5.4.3

Efron, B. and Hinkley, D. V. (1978). Assessing the accuracy of the maximum likelihood estimator: Observed versus expected Fisher information. *Biometrika*, 65(3):457–483. 6.5.4

Evans, L. C. (2012). *An introduction to stochastic differential equations*, volume 82. American Mathematical Soc. 6.3.1

Fan, Y. and Brooks, S. P. (2000). Bayesian modelling of prehistoric corbelled domes. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 49(3):339–354. 2.8

Fan, Y. and Sisson, S. A. (2011). *Handbook of Markov Chain Monte Carlo*, chapter Reversible Jump MCMC, pages 67–92. Chapman & Hall/CRC Boca Raton, FL. 2.8

Fearnhead, P. (2006). Exact and efficient Bayesian inference for multiple changepoint problems. *Statistics and computing*, 16(2):203–213. 5.2, 5.3.1, 5.3.2, 5.3.2, 5.4, 5.4.1, 5.4.1, 5.4.3

Fearnhead, P. and Liu, Z. (2007). On-line inference for multiple changepoint problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):589–605. 5.2, 5.3.2, 5.4.1

Fearnhead, P. and Liu, Z. (2011). Efficient bayesian analysis of multiple changepoint models with dependence across segments. *Statistics and Computing*, 21(2):217–229. 5.4.1

Fearnhead, P., Papaspiliopoulos, O., and Roberts, G. O. (2008). Particle filters for partially observed diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(4):755–777. 6.5.2.1

Fearnhead, P. and Prangle, D. (2012). Constructing summary statistics for approximate Bayesian computation: Semi-automatic approximate Bayesian computation. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(3):419–474. 4.2, 4.5.4

Fearnhead, P. and Vasileiou, D. (2009). Bayesian analysis of isochores. *Journal of the American Statistical Association*, 104(485):132–141. (document), 3.2, 5.2, 5.3.1, 5.1, 5.1, 5.3.2, 5.3.2, 5.4, 5.4.3, 5.6, 5.2, 5.7, 7.1

Foreman-Mackey, D., Hogg, D. W., Lang, D., and Goodman, J. (2013). emcee: The MCMC Hammer. *Publications of the Astronomical Society of the Pacific*, 125(925):306. 4.5.1.3

Fréchet, M. (1943). Sur l'extension de certaines évaluations statistiques au cas de petits échantillons. *Revue de l'Institut International de Statistique*, pages 182–205. 6.2, 6.5.5

Friel, N. and Pettitt, A. N. (2008). Marginal likelihood estimation via power posteriors. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(3):589–607. 4.2, 4.6

Gelfand, A. E. and Smith, A. F. (1990). Sampling-based approaches to calculating marginal densities. *Journal of the American statistical association*, 85(410):398–409. 1.1.1, 2.3.2

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE Transactions on Pattern Analysis and Machine Intelligence(6):721–741. 1.1, 1.1.1, 2.3.2

Gerber, M., Chopin, N., Whiteley, N., et al. (2019). Negative association, ordering and convergence of resampling methods. *Annals of Statistics*, 47(4):2236–2260. 3.3.3

Geyer, C. (2011). Importance sampling, simulated tempering and umbrella sampling. *Handbook of Markov Chain Monte Carlo*, pages 295–311. 2.4.1, 4.2

Geyer, C. J. (1991). Markov chain Monte Carlo maximum likelihood. *Interface Foundation of North America*. 1.2, 2.4.1, 4.2

Geyer, C. J. and Thompson, E. A. (1995). Annealing Markov chain Monte Carlo with applications to ancestral inference. *Journal of the American Statistical Association*, 90(431):909–920. 2.4.1

Gilks, W., Richardson, S., and Spiegelhalter, D. (1996). *Markov Chain Monte Carlo in practice*. Chapman and Hall/CRC. 1.1.1, 2.2

Gilks, W. R., Best, N. G., and Tan, K. (1995). Adaptive rejection Metropolis sampling within Gibbs sampling. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 44(4):455–472. 5.4.2

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361. 4.5.4

Godsill, S. J., Doucet, A., and West, M. (2004). Monte Carlo smoothing for nonlinear time series. *Journal of the american statistical association*, 99(465):156–168. 3.4.3, 6.5.3

Goodman, J. and Weare, J. (2010). Ensemble samplers with affine invariance. *Communications in applied mathematics and computational science*, 5(1):65–80. 4.5.1.3

Goodman, J. W. (2005). *Introduction to Fourier optics*. Roberts and Company Publishers. 6.3.2

Gordon, N. J., Salmond, D. J., and Smith, A. F. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE proceedings F (radar and signal processing)*, volume 140, pages 107–113. IET. 3.3.3, 3.3.4, 6.5.2.2

Gray, A. G. and Moore, A. W. (2001). '*N*-body' problems in statistical learning. In *Advances in neural information processing systems*, pages 521–527. 3.5.2

Green, P. J. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732. 1.3, 2.8, 4.2, 4.6, 5.2, 5.3.2, 5.4, 5.4.2, 5.4.2.2, 5.4.3

Green, P. J. (2003). *Highly Structured Stochastic Systems*, volume 27, chapter Trans-dimensional Markov chain Monte Carlo, pages 179–198. Oxford University Press. 2.8, 2.8

Gustafsson, F. (2013). On marginal particle filters with linear complexity. In *2013 5th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 356–359. IEEE. 3.3.6

Haario, H., Saksman, E., Tamminen, J., et al. (2001). An adaptive Metropolis algorithm. *Bernoulli*, 7(2):223–242. 4.3.4

Hamilton, G., Currat, M., Ray, N., Heckel, G., Beaumont, M., and Excoffier, L. (2005). Bayesian estimation of recent migration rates after a spatial expansion. *Genetics*, 170(1):409–417. 2.7

Hamilton, J. D. (1994). *Time Series Analysis*, volume 2. Princeton university press Princeton. 4.5.3

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*. 1.1.1, 2.3.1

Hoffman, M. D. and Gelman, A. (2014). The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623. 2.4.2, 4.2, 7.2

Hritz, J. and Oostenbrink, C. (2007). Optimization of replica exchange molecular dynamics by fast mimicking. *The Journal of chemical physics*, 127(20):204104. 4.2, 4.6

Hughes, J. P., Guttorp, P., and Charles, S. P. (1999). A non-homogeneous hidden Markov model for precipitation occurrence. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 48(1):15–30. 3.2

Hürzeler, M. and Künsch, H. R. (1998). Monte Carlo approximations for general state-space models. *Journal of Computational and graphical Statistics*, 7(2):175–193. 3.4.2, 6.5.3

Itoh, N. and Kurths, J. (2010). Change-point detection of climate time series by nonparametric method. In *Proceedings of the world congress on engineering and computer science*, volume 1, pages 445–448. Citeseer. 5.2

Jabot, F. and Chave, J. (2009). Inferring the parameters of the neutral theory of biodiversity using phylogenetic information and implications for tropical forests. *Ecology letters*, 12(3):239–248. 2.7

Jarrett, R. (1979). A note on the intervals between coal-mining disasters. *Biometrika*, 66(1):191–193. 5.2, 5.4.3

Jasra, A., Stephens, D. A., and Holmes, C. C. (2007a). On population-based simulation for static inference. *Statistics and Computing*, 17(3):263–279. 5.5.2

Jasra, A., Stephens, D. A., and Holmes, C. C. (2007b). Population-based reversible jump Markov chain Monte Carlo. *Biometrika*, 94(4):787–807. 4.2, 4.6

Jazwinski, A. H. (2007). *Stochastic processes and filtering theory*. Courier Corporation. 6.3.1

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45. 3.2.2

Kalman, R. E. and Bucy, R. S. (1961). New Results in Linear Filtering and Prediction Theory. *Journal of Basic Engineering*, 83(1):95–108. 3.2.2

Kantas, N., Doucet, A., Singh, S. S., Maciejowski, J., Chopin, N., et al. (2015). On particle methods for parameter estimation in state-space models. *Statistical science*, 30(3):328–351. 6.2

Karimi, K., Dickson, N., and Hamze, F. (2011). High-performance physics simulations using multi-core CPUs and GPGPUs in a volunteer computing context. *The International Journal of High Performance Computing Applications*, 25(1):61–69. 4.2, 4.6

Kitagawa, G. (1987). Non-gaussian state—space modeling of nonstationary time series. *Journal of the American statistical association*, 82(400):1032–1041. 3.2, 3.4.2

Kitagawa, G. (1996). Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of computational and graphical statistics*, 5(1):1–25. 6.5.3

Kitagawa, G. and Sato, S. (2001). Monte Carlo smoothing and self-organising state-space model. In *Sequential Monte Carlo methods in practice*, pages 177–195. Springer. 3.5.1, 6.5.3

Klaas, M., De Freitas, N., and Doucet, A. (2012). Toward practical N^2 Monte Carlo: The marginal particle filter. *arXiv preprint arXiv:1207.1396*. 3.3.6, 3.5.2

Kohn, R., Quiroz, M., Tran, M.-N., and Villani, M. (2016). Speeding up MCMC by Efficient Data Subsampling. Working Papers 2123/16205, University of Sydney Business School, Discipline of Business Analytics. 2.4.2

Kone, A. and Kofke, D. A. (2005). Selection of temperature intervals for parallel-tempering simulations. *The Journal of chemical physics*, 122(20):206101. 4.3.4

Kong, A., Liu, J. S., and Wong, W. H. (1994). Sequential imputations and Bayesian missing data problems. *Journal of the American statistical association*, 89(425):278–288. 2.5, 3.3.3, 3.3.4

Korattikara, A., Chen, Y., and Welling, M. (2014). Austerity in MCMC land: Cutting the Metropolis-Hastings budget. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 181–189. 2.4.2

Laskey, K. B. and Myers, J. W. (2003). Population Markov chain Monte Carlo. *Machine Learning*, 50(1-2):175–196. 1.2

Lavielle, M. and Lebarbier, E. (2001). An application of mcmc methods for the multiple change-points problem. *Signal processing*, 81(1):39–53. 5.2

Le Gland, F. and Mevel, L. (1997). Recursive identification in hidden Markov models. In *Proceedings of the 36th Conference on Decision and Control, San Diego 1997*, volume 4, pages 3468–3473. 6.5.4

Lee, A. (2012). On the choice of MCMC kernels for approximate Bayesian computation with SMC samplers. In *Simulation Conference (WSC), Proceedings of the 2012 Winter*, pages 1–12. IEEE. 2.7, 2.7, 4.2, 4.4.1, 4.5.2, 7.1

Lee, A. and Łatuszyński, K. (2014). Variance bounding and geometric ergodicity of markov chain monte carlo kernels for approximate bayesian computation. *Biometrika*, 101(3):655–671. 2.7, 4.2, 4.5.4, 4.5.4.1

Lee, A., Yau, C., Giles, M. B., Doucet, A., and Holmes, C. C. (2010). 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. 1.2, 5.2, 5.5.1

Lemaréchal, C. (2012). Cauchy and the gradient method. *Doc Math Extra*, 251:254. 6.5.6.1

Li, J., Najmi, A., and Gray, R. M. (2000). Image classification by a two-dimensional hidden markov model. *IEEE Transactions on Signal Processing*, 48(2):517–533. 3.2

Liang, F., Kim, J., and Song, Q. (2016). A bootstrap Metropolis–Hastings algorithm for Bayesian analysis of big data. *Technometrics*, 58(3):304–318. 2.4.2

Liang, F., Liu, C., and Carroll, R. (2011). *Advanced Markov chain Monte Carlo methods: learning from past samples*, volume 714, chapter Population-based MCMC methods, pages 123–163. John Wiley & Sons. 1.2

Lindsten, F. and Schön, T. B. (2013). Backward simulation methods for Monte Carlo statistical inference. *Foundations and Trends® in Machine Learning*, 6(1):1–143. 6.2

Lingenheil, M., Denschlag, R., Mathias, G., and Tavan, P. (2009). Efficiency of exchange schemes in replica exchange. *Chemical Physics Letters*, 478(1-3):80–84. 2.4.1, 4.3.4

Liu, J. S. and Chen, R. (1995). Blind deconvolution via sequential imputations. *Journal of the american statistical association*, 90(430):567–576. 3.3.4

Liu, J. S. and Chen, R. (1998). Sequential Monte Carlo methods for dynamic systems. *Journal of the American statistical association*, 93(443):1032–1044. 2.5

Liu, J. S. and Lawrence, C. E. (1999). Bayesian inference on biopolymer models. *Bioinformatics (Oxford, England)*, 15(1):38–52. 5.4.1

Liu, S., Kromann, E. B., Krueger, W. D., Bewersdorf, J., and Lidke, K. A. (2013). Three dimensional single molecule localization using a phase retrieved pupil function. *Optics express*, 21(24):29462–29487. 6.2

Lotka, A. J. (1926). Elements of Physical Biology. *Science Progress in the Twentieth Century (1919-1933)*, 21(82):341–343. 4.5.4

Ly, A., Marsman, M., Verhagen, J., Grasman, R. P., and Wagenmakers, E.-J. (2017). A tutorial on fisher information. *Journal of Mathematical Psychology*, 80:40–55. 6.2

Maclaurin, D. and Adams, R. P. (2014). Firefly Monte Carlo: Exact MCMC with Subsets of Data. In *UAI*, pages 543–552. 2.4.2

Marie d'Avigneau, A., Singh, S. S., and Murray, L. M. (2020). Anytime Parallel Tempering. *arXiv preprint arXiv:2006.14875 [stat]*. arXiv: 2006.14875. 1.4, 4.1

Marie d'Avigneau, A., Singh, S. S., and Ober, R. J. (2021). Limits of accuracy for parameter estimation and localisation in Single-Molecule Microscopy via sequential Monte Carlo methods. 1.4, 6.1

Marin, J.-M., Pillai, N. S., Robert, C. P., and Rousseau, J. (2014). Relevant statistics for Bayesian model choice. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(5):833–859. 2.7, 4.5.3

Marin, J.-M., Pudlo, P., Robert, C. P., and Ryder, R. J. (2012). Approximate Bayesian computational methods. *Statistics and Computing*, pages 1–14. 4.5.3, 4.5.3.2

Marin, J.-M. and Robert, C. P. (2007). *Bayesian core : a practical approach to computational Bayesian statistics* . Springer texts in statistics. Springer, New York. 4.5.3

Marinari, E. and Parisi, G. (1992). Simulated tempering: a new Monte Carlo scheme. *EPL (Europhysics Letters)*, 19(6):451. 2.4.1

Marjoram, P., Molitor, J., Plagnol, V., and Tavaré, S. (2003). Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 100(26):15324–15328. 2.7, 2.7

MATLAB (2019). *version 9.7.0.1190202 (R2019b)*. The MathWorks Inc., Natick, Massachusetts. 4.5.1.3

Meinhold, R. J. and Singpurwalla, N. D. (1983). Understanding the Kalman filter. *The American Statistician*, 37(2):123–127. 3.2.2

Metropolis, N. (1987). The beginning of the Monte Carlo method. *Los Alamos Science*, 15:125–130. 1.1

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092. 1.1.1, 2.3.1, 2.3.1

Miasojedow, B., Moulines, E., and Vihola, M. (2013). An adaptive parallel tempering algorithm. *Journal of Computational and Graphical Statistics*, 22(3):649–664. 4.3.4

Michalet, X. (2010). Mean square displacement analysis of single-particle trajectories with localization error: Brownian motion in an isotropic medium. *Physical Review E*, 82(4):041914. 6.2

Michalet, X. and Berglund, A. J. (2012). Optimal diffusion coefficient estimation in single-particle tracking. *Physical Review E*, 85(6):061916. 6.2

Mikusheva, A. (2007). Course materials for 14.384 Time Series Analysis. MIT OpenCourseWare, Massachusetts Institute of Technology. [Downloaded on 22 June 2020]. 2.3.2.2

Minsker, S., Srivastava, S., Lin, L., and Dunson, D. (2014). Scalable and robust Bayesian inference via the median posterior. In *International Conference on Machine Learning*, pages 1656–1664. 2.4.2

Moerner, W. and Fromm, D. P. (2003). Methods of single-molecule fluorescence spectroscopy and microscopy. *Review of Scientific instruments*, 74(8):3597–3619. 1.3, 6.2

Murray, I., Adams, R., and MacKay, D. (2010). Elliptical slice sampling. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 541–548. 4.2, 7.2

Murray, L. (2010). Distributed Markov chain Monte Carlo. In *Proceedings of neural information processing systems workshop on learning on cores, clusters and clouds*, volume 11. 1.2

Murray, L. M. (2013). Bayesian state-space modelling on high-performance hardware using LibBi. *arXiv preprint arXiv:1306.3277*. 5.5.1

Murray, L. M., Lee, A., and Jacob, P. E. (2016a). Parallel resampling in the particle filter. *Journal of Computational and Graphical Statistics*, 25(3):789–805. 3.3.3

Murray, L. M., Singh, S., Jacob, P. E., and Lee, A. (2016b). Anytime Monte Carlo. *arXiv preprint arXiv:1612.03319*. (document), 1.2, 1.4, 2.6, 2.4, 2.6, 4.1, 4.2, 4.3.2, 5.2, 5.5.1, 5.5.2, 5.6.5, 5.7, 7.1

Neal, R. M. (1996). Sampling from multimodal distributions using tempered transitions. *Statistics and computing*, 6(4):353–366. 2.4.1

Neal, R. M. (2001). Annealed importance sampling. *Statistics and computing*, 11(2):125–139. 2.4.1

Neal, R. M. (2011). *Handbook of Markov chain Monte Carlo*, chapter MCMC using Hamiltonian dynamics, pages 113–162. Chapman & Hall/CRC. 2.4.2

Neiswanger, W., Wang, C., and Xing, E. (2013). Asymptotically exact, embarrassingly parallel MCMC. *arXiv preprint arXiv:1311.4780*. 1.2, 2.4.2

Nemeth, C. (2014). *Parameter estimation for state space models using sequential Monte Carlo algorithms*. PhD thesis, Lancaster University. 3.3.4

Nevat, I., Peters, G., and Yuan, J. (2009). Coherent detection for cooperative networks with arbitrary relay functions using likelihood-free inference. In *NEWCOM-ACorn Workshop*. 2.7

Nishihara, R., Murray, I., and Adams, R. P. (2014). Parallel MCMC with generalized elliptical slice sampling. *The Journal of Machine Learning Research*, 15(1):2087–2112. 4.2, 4.6

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media. 6.5.6.1

Nott, D. J. and Leonte, D. (2004). Sampling schemes for Bayesian variable selection in generalized linear models. *Journal of Computational and Graphical Statistics*, 13(2):362–382. 2.8

Ó Ruanaidh, J. J. and Fitzgerald, W. J. (1996). Retrospective Changepoint Detection. In *Numerical Bayesian Methods Applied to Signal Processing*, pages 96–121. Springer. 5.2, 5.4.3

Ober, R. J., Martinez, C., Lai, X., Zhou, J., and Ward, E. S. (2004a). Exocytosis of IgG as mediated by the receptor, FcRn: an analysis at the single-molecule level. *Proceedings of the National Academy of Sciences*, 101(30):11076–11081. 6.2

Ober, R. J., Ram, S., and Ward, E. S. (2004b). Localization accuracy in single-molecule microscopy. *Biophysical journal*, 86(2):1185–1200. 6.2, 6.5, 6.6.3

Ober, R. J., Ward, E. S., and Chao, J. (2020a). *Quantitative Bioimaging: An Introduction to Biology, Instrumentation, Experiments and Data Analysis for Scientists and Engineers*. CRC Press. 6.1, 6.2

Ober, R. J., Ward, E. S., and Chao, J. (2020b). *Quantitative Bioimaging: An Introduction to Biology, Instrumentation, Experiments and Data Analysis for Scientists and Engineers*, chapter Localizing Objects and Single Molecules in Three Dimensions, pages 377–401. CRC Press. 6.6.2

Ober, R. J., Ward, E. S., and Chao, J. (2020c). *Quantitative Bioimaging: An Introduction to Biology, Instrumentation, Experiments[and Data Analysis for Scientists and Engineers*, chapter Localizing Objects and Single Molecules in Two Dimensions, pages 337–375. CRC Press. 6.6

Oksendal, B. (2013). *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media. 6.2

Olsson, J., Cappé, O., Douc, R., Moulines, E., et al. (2008). Sequential Monte Carlo smoothing with application to parameter estimation in nonlinear state space models. *Bernoulli*, 14(1):155–179. 3.5.1, 6.5.3

Olsson, J., Ströjby, J., et al. (2011). Particle-based likelihood inference in partially observed diffusion processes using generalised poisson estimators. *Electronic Journal of Statistics*, 5:1090–1122. 3.5.1, 6.5.3

Olsson, J., Westerborn, J., et al. (2017). Efficient particle-based online smoothing in general hidden Markov models: the PaRIS algorithm. *Bernoulli*, 23(3):1951–1996. 3.5.2, 6.2, 6.5.3, 6.7

Papaspiliopoulos, O. (2010). A methodological framework for Monte Carlo probabilistic inference for diffusion processes. In *In Inference and Learning in Dynamic Models*. Citeseer. 3.3.5, 6.5.2.1

Peskun, P. (1981). Guidelines for choosing the transition matrix in Monte Carlo methods using Markov chains. *Journal of Computational Physics*, 40(2):327–344. 1.1.1, 2.3.1

Peskun, P. H. (1973). Optimum Monte-Carlo sampling using Markov chains. *Biometrika*, 60(3):607–612. 1.1.1, 2.3.1

Peters, G. W., Nevat, I., Sisson, S. A., Fan, Y., and Yuan, J. (2010). Bayesian symbol detection in wireless relay networks via likelihood-free inference. *IEEE Transactions on Signal Processing*, 58(10):5206–5218. 2.7

Peters, G. W., Sisson, S. A., and Fan, Y. (2012). Likelihood-free Bayesian inference for α-stable models. *Computational Statistics & Data Analysis*, 56(11):3743–3756. 2.7

Pitt, M. K. and Shephard, N. (1999). Filtering via simulation: Auxiliary particle filters. *Journal of the American statistical association*, 94(446):590–599. 3.3.5, 3.3.5, 6.2, 6.5.2.1, 6.5.2.2

Poyiadjis, G., Doucet, A., and Singh, S. S. (2011). Particle approximations of the score and observed information matrix in state space models with application to parameter estimation. *Biometrika*, 98(1):65–80. 3.5, 3.5.2, 6.5.2.1, 6.5.4, 6.5.4.1

Prangle, D. et al. (2017). Adapting the ABC distance function. *Bayesian Analysis*, 12(1):289–309. 4.2, 4.5.4

Prince, S. J. (2012). *Computer vision: models, learning, and inference*, chapter Modeling complex data densities, pages 101–142. Cambridge University Press. 5.7

Pritchard, J. K., Seielstad, M. T., Perez-Lezaun, A., and Feldman, M. W. (1999). Population growth of human Y chromosomes: a study of Y chromosome microsatellites. *Molecular biology and evolution*, 16(12):1791–1798. 2.7, 2.7, 4.2, 4.4.1

Punskaya, E., Andrieu, C., Doucet, A., and Fitzgerald, W. J. (2002). Bayesian curve fitting using MCMC with applications to signal segmentation. *IEEE Transactions on Signal Processing*, 50(3):747–758. 5.2

Quan, K., Tanno, R., Duong, M., Nair, A., Shipley, R., Jones, M., Brereton, C., Hurst, J., Hawkes, D., and Jacob, J. (2019). Modelling Airway Geometry as Stock Market Data Using Bayesian Changepoint Detection. In *International Workshop on Machine Learning in Medical Imaging*, pages 345–354. Springer. 5.4.1, 5.4.3, 5.7

Quiroz, M., Tran, M.-N., Villani, M., and Kohn, R. (2017). Speeding up MCMC by delayed acceptance and data subsampling. *Journal of Computational and Graphical Statistics*. 2.4.2

Quiroz, M., Villani, M., and Kohn, R. (2016). Exact subsampling MCMC. *arXiv preprint arXiv:1603.08232*. 2.4.2

Raftery, A. E. and Akman, V. (1986). Bayesian analysis of a poisson process with a change-point. *Biometrika*, pages 85–89. 5.2

Ram, S., Ward, E. S., and Ober, R. J. (2006a). Beyond Rayleigh's criterion: a resolution measure with application to single-molecule microscopy. *Proceedings of the National Academy of Sciences*, 103(12):4457–4462. 6.6.3, 6.6.3

Ram, S., Ward, E. S., and Ober, R. J. (2006b). A stochastic analysis of performance limits for optical microscopes. *Multidimensional Systems and Signal Processing*, 17(1):27–57. 6.2, 6.3.2, 6.6.3

Ram, S., Ward, E. S., and Ober, R. J. (2013). A stochastic analysis of distance estimation approaches in single molecule microscopy: quantifying the resolution limits of photon-limited imaging systems. *Multidimensional systems and signal processing*, 24(3):503–542. 6.1, 6.6.3, 6.6.3, 6.6.3, 6.7, 7.1

Rao, C. R. (1992). Information and the accuracy attainable in the estimation of statistical parameters. In *Breakthroughs in statistics*, pages 235–247. Springer. 6.2, 6.5.5

Rathore, N., Chopra, M., and de Pablo, J. J. (2005). Optimal allocation of replicas in parallel tempering simulations. *The Journal of chemical physics*, 122(2):024111. 4.3.4

Ratmann, O., Jørgensen, O., Hinkley, T., Stumpf, M., Richardson, S., and Wiuf, C. (2007). Using likelihood-free inference to compare evolutionary dynamics of the protein networks of H. pylori and P. falciparum. *PLoS Comput Biol*, 3(11):e230. 2.7

Relich, P. K., Olah, M. J., Cutler, P. J., and Lidke, K. A. (2016). Estimation of the diffusion constant from intermittent trajectories with variable position uncertainties. *Physical Review E*, 93(4):042401. 6.2

Richardson, S. and Green, P. J. (1997). On Bayesian analysis of mixtures with an unknown number of components (with discussion). *Journal of the Royal Statistical Society: series B (statistical methodology)*, 59(4):731–792. 2.8, 5.4.2.2

Robert, C. and Casella, G. (2004a). *Monte Carlo statistical methods*. Springer Science & Business Media. 2.2, 2.3.2.2

Robert, C. and Casella, G. (2004b). *Monte Carlo Statistical Methods*, chapter The Metropolis-Hastings Algorithm. Springer Texts in Statistics, Springer, New York. 2.3.1, 4.2

Robert, C. and Casella, G. (2011). A short history of Markov chain Monte Carlo: Subjective recollections from incomplete data. *Statistical Science*, pages 102–115. 2.3

Robert, C. P., Elvira, V., Tawn, N., and Wu, C. (2018). Accelerating MCMC algorithms. *Wiley Interdisciplinary Reviews: Computational Statistics*, 10(5):e1435. 2.2, 2.4

Roberts, G. O., Rosenthal, J. S., et al. (2001). Optimal scaling for various Metropolis-Hastings algorithms. *Statistical science*, 16(4):351–367. 4.3.4

Roberts, G. O., Tweedie, R. L., et al. (1996). Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*, 2(4):341–363. 2.4.2

Rodinger, T., Howell, P. L., and Pomès, R. (2006). Distributed replica sampling. *Journal of chemical theory and computation*, 2(3):725–731. 4.2, 4.6

Rydén, T., Teräsvirta, T., and Åsbrink, S. (1998). Stylized facts of daily return series and the hidden Markov model. *Journal of applied econometrics*, 13(3):217–244. 3.2

Saxton, M. J. (1997). Single-particle tracking: the distribution of diffusion coefficients. *Biophysical journal*, 72(4):1744. 6.2

Saxton, M. J. and Jacobson, K. (1997). Single-particle tracking: applications to membrane dynamics. *Annual review of biophysics and biomolecular structure*, 26(1):373–399. 6.2

Scott, S. L. (2002). Bayesian methods for hidden Markov models: Recursive computing in the 21st century. *Journal of the American statistical Association*, 97(457):337–351. 3.2.1, 5.6.3.3

Scott, S. L., Blocker, A. W., Bonassi, F. V., Chipman, H. A., George, E. I., and McCulloch, R. E. (2016). Bayes and big data: The consensus Monte Carlo algorithm. *International Journal of Management Science and Engineering Management*, 11(2):78–88. 1.2, 2.4.2

Shashkova, S. and Leake, M. C. (2017). Single-molecule fluorescence microscopy review: shedding new light on old problems. *Bioscience reports*, 37(4):BSR20170031. 1.3, 6.2

Sisson, S. A. (2005). Transdimensional Markov chains: A decade of progress and future perspectives. *Journal of the American Statistical Association*, 100(471):1077–1089. 2.8

Sisson, S. A. and Fan, Y. (2011). *Handbook of Markov Chain Monte Carlo*, chapter Likelihood-free MCMC, pages 313–335. Chapman & Hall/CRC, New York.[839]. 1.3, 2.7

Sisson, S. A., Fan, Y., and Tanaka, M. M. (2007). Sequential Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 104(6):1760–1765. 2.7, 2.7

Small, A. and Stahlheber, S. (2014). Fluorophore localization algorithms for super-resolution microscopy. *Nature methods*, 11(3):267. 6.2

Sokal, A. (1997). Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms. In *Functional integration*, pages 131–192. Springer. 4.5.1.3

Stallinga, S. and Rieger, B. (2010). Accuracy of the Gaussian point spread function model in 2D localization microscopy. *Optics express*, 18(24):24461–24476. 6.3.2

Swendsen, R. H. and Wang, J.-S. (1986). Replica Monte Carlo simulation of spin-glasses. *Physical Review Letters*, 57(21):2607. 1.2, 2.4.1, 4.2

Syed, S., Bouchard-Côté, A., Deligiannidis, G., and Doucet, A. (2019). Non-reversible parallel tempering: A scalable highly parallel MCMC scheme. *arXiv preprint arXiv:1905.02939*. 2.4.1, 4.3.4

Tadesse, M. G., Sha, N., and Vannucci, M. (2005). Bayesian variable selection in clustering high-dimensional data. *Journal of the American Statistical Association*, 100(470):602–617. 2.8

Tavaré, S., Balding, D. J., Griffiths, R. C., and Donnelly, P. (1997). Inferring coalescence times from DNA sequence data. *Genetics*, 145(2):505–518. 2.7, 2.7, 4.2, 4.2, 4.4.1

Thompson, R. E., Larson, D. R., and Webb, W. W. (2002). Precise nanometer localization analysis for individual fluorescent probes. *Biophysical journal*, 82(5):2775–2783. 6.3.2

Tierney, L. (1994). Markov chains for exploring posterior distributions. *The Annals of Statistics*, pages 1701–1728. 2.3.2.3

Toni, T., Welch, D., Strelkowa, N., Ipsen, A., and Stumpf, M. P. (2009). Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society Interface*, 6(31):187–202. 4.2, 4.5.4

Ureten, O. and Serinken, N. (2005). Bayesian detection of Wi-Fi transmitter RF fingerprints. *Electronics Letters*, 41(6):373–374. 5.2

Vahid, M. R., Hanzon, B., and Ober, R. J. (2019). Effect of Pixelation on the Parameter Estimation of Single Molecule Trajectories. *arXiv preprint arXiv:1912.07182*. 7.2

Vahid, M. R., Hanzon, B., and Ober, R. J. (2020). Fisher information matrix for single molecules with stochastic trajectories. *SIAM Journal on Imaging Sciences*, 13(1):234–264. 1.3, 3.2, 6.1, 6.2, 6.3, 6.3.3, 6.6.1

Vermaak, J., Andrieu, C., Doucet, A., and Godsill, S. (2004). Reversible jump Markov chain Monte Carlo strategies for Bayesian model selection in autoregressive processes. *Journal of Time Series Analysis*, 25(6):785–809. 2.8

Volterra, V. (1927). *Variazioni e fluttuazioni del numero d'individui in specie animali conviventi*. C. Ferrari. 4.5.4

Wang, F. and Jordan, K. (2003). Parallel-tempering Monte Carlo simulations of the finite temperature behavior of $(H_2O)_6^-$. *The Journal of chemical physics*, 119(22):11645–11653. 4.2, 4.6

Wang, X. and Dunson, D. B. (2013). Parallelizing MCMC via Weierstrass sampler. *arXiv preprint arXiv:1312.4605*. 2.4.2

West, M. and Harrison, J. (2006). *Bayesian forecasting and dynamic models*. Springer Science & Business Media. 3.2.2

Whiteley, N. and Johansen, A. M. (2010). Recent developments in auxiliary particle filtering. *Inference and learning in dynamic models*, 38:39–47. 3.3.5

Wilkinson, D. (2013). Parallel tempering and Metropolis coupled MCMC. 2.1

Wilkinson, D. J. (2006). Parallel Bayesian computation. *Statistics Textbooks and Monographs*, 184:477. 1.2

Wilkinson, D. J. (2011). *Stochastic Modelling for Systems Biology*. CRC press. 4.5.4

Wilkinson, R. D. and Tavaré, S. (2009). Estimating primate divergence times by using conditioned birth-and-death processes. *Theoretical population biology*, 75(4):278–285. 2.7

Wu, C. J. (1983). On the convergence properties of the EM algorithm. *The Annals of statistics*, pages 95–103. 6.5.6.2

Wyse, J. and Friel, N. (2010). Simulation-based Bayesian analysis for multiple changepoints. *arXiv preprint arXiv:1011.2932*. 2.8, 5.2, 5.4.1, 5.4.3, 5.7

Xie, X. and Evans, R. J. (1991). Multiple target tracking and multiple frequency line tracking using hidden Markov models. *IEEE Transactions on Signal Processing*, 39(12):2659–2676. 3.2

Xing, H., Mo, Y., Liao, W., and Zhang, M. Q. (2012). Genome-wide localization of protein-DNA binding and histone modification by a Bayesian change-point method with ChIP-seq data. *PLoS computational biology*, 8(7). 5.2

Xu, M., Lakshminarayanan, B., Teh, Y. W., Zhu, J., and Zhang, B. (2014). Distributed Bayesian posterior sampling via moment sharing. In *Advances in Neural Information Processing Systems*, pages 3356–3364. 2.4.2

Yao, Y.-C. (1984). Estimation of a noisy discrete-time step function: Bayes and empirical Bayes approaches. *The Annals of Statistics*, pages 1434–1447. 5.3.2

Yu, J., Xiao, J., Ren, X., Lao, K., and Xie, X. S. (2006). Probing gene expression in live cells, one protein molecule at a time. *Science*, 311(5767):1600–1603. 6.2

Yıldırım, S. (2012). *Maximum Likelihood Parameter Estimation in Time Series Models Using Sequential Monte Carlo*. phdthesis, University of Cambridge. 5.7

Yıldırım, S., Singh, S. S., and Doucet, A. (2013). An online expectation–maximization algorithm for changepoint models. *Journal of Computational and Graphical Statistics*, 22(4):906–926. 5.3.1, 5.4, 5.4.3, 5.6, 5.7

Zhang, B., Zerubia, J., and Olivo-Marin, J.-C. (2007). Gaussian approximations of fluorescence microscope point-spread function models. *Applied optics*, 46(10):1819–1829. 6.3.2

Zucchini, W. and Guttorp, P. (1991). A hidden Markov model for space-time precipitation. *Water Resources Research*, 27(8):1917–1923. 3.2