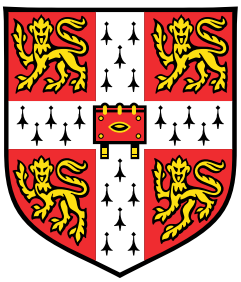
Iterative Monte Carlo Approximations for Bayesian Inference



Samuel Duffield

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

St Edmund's College

September 2021

For my beloved Alexandra

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Samuel Duffield
September 2021

Abstract

Author: Samuel Duffield
Thesis Title: Iterative Monte Carlo Approximations for Bayesian Inference

The common theme of this thesis is the concept of using Monte Carlo techniques to approximate a sequence of probability distributions. Novel methodological contributions are found in Chapter 3 through to Chapter 6.

In Chapter 3 we derive a method for the complete characterisation of online statistical models where Monte Carlo approximations are defined sequentially as new data arrive. We then demonstrate the utility of this method in Chapter 4 for the compelling application of de-noising sequential GPS coordinates to be restricted to a road network.

In Chapter 5 and Chapter 6, the sequence of probability distributions are defined artificially in order to gradually (and more effectively) approach a single offline target probability distribution. Chapter 5 adopts ideas from high-dimensional time series to efficiently tackle the difficult setting where we cannot evaluate the density of the target distribution and instead can only generate synthetic data. Chapter 6 explores the use of a scalable Hessian approximation in the more common scenario where the target density can be evaluated and even differentiated.

Finally, Chapter 7 describes a general purpose software package that can be used to implement and customise all of the discussed algorithms at competitive speeds.

Acknowledgements

To my supervisor Professor Sumeetpal Singh, I would like to express my heartiest thanks for the opportunity, experience and guidance. His wisdom has been influential in the development of my research and indeed my ability as a researcher.

My time in the Signal Processing and Communications Laboratory has been thoroughly enjoyable due to the warm and welcoming environment instilled by its members. Immeasurable thanks goes to Jacob Vorstup Goldman for his countless valuable insights; Hugo Hadfield for tolerating my terrible Spanish and annoying questions about Python; Oliver Bonner and Fergal Cotter for their kindness and friendship; Ehsan Asadi and Parham Boroumand for developing my taste for saffron; Alex Grafton and Ulrika Andersson for their crossword support; Alix Marie d'Avigneau, Yaman Kindap and all of the further distinguished members of SigProC who have each contributed to a hugely enjoyable period - both socially and academically. It is sad that we have had to be so distanced for the last two years. Beyond the laboratory, I owe great thanks to my peers from the mathematics department Sam Power and Torben Sell whose regular conversations proved highly enjoyable and improved my mathematical acumen no end.

St Edmund's College has provided a steady and comforting foundation during my time in Cambridge. To those who had the (mis)fortune to call themselves a flatmate of mine - Florence Cochrane, Michael Stanton, Leanne O'Brien and Max Butler - I thank deeply for making the college a home.

I am forever grateful to my parents and brother, Josh. Even if some of the mathematics goes over their heads, I am exceedingly appreciative of their love and unconditional support in all aspects of my life.

The conclusion of my PhD was spent under somewhat atypical conditions, I am grateful to Natalia Sánchez, Nidhi Jetley and Dusty for their companionship during this time. Most importantly, I offer a whale of appreciation to my partner Alex who has provided me with encouragement and such happiness for the entire journey.

A final thanks goes to James Cryne, for finding a typo in the introduction of this thesis.

Table of contents

Chapter 1

Introduction

The first step in a statistical analysis most commonly comes down to the construction of a statistical model or data generating process. We notate this relationship as

$$p(y \mid x),$$

where y represents the collected data and x is a parameter whose value is unknown. In the statistics paradigm, the relationship between the unknown parameter x and the data y is random or stochastic in nature. We therefore refer to $p(y \mid x)$ as a *probability distribution* in y given x - it describes the probability that the data y was generated by the given value of x. It is often useful to describe $p(y \mid x)$ as a function in x since the value of the data y is known, in this case we use the term *likelihood* or *likelihood function*. Interestingly, the term likelihood was introduced by Fisher (1954) specifically to emphasise the point that $p(y \mid x)$ is not a probability distribution in x.

Generally speaking, classical statistics can be thought of as a two step procedure:

- First, is the formulation of the statistical model and the structure of the unknown parameter x. We refer to this step as statistical *modelling*. The general idea is to leverage prior domain knowledge to build a data generating process that is consistent with both the observed data y as well as any constraints or dynamics that we are aware of.

- The second step is that of statistical *inference*. That is, translate the statistical model and observed data y into logical reasoning about the value of the unknown parameter x. This may involve point estimates, confidence intervals or predictions.

The construction of the statistical model is something of an art form with the key features depending on the nature of available prior knowledge and the field of study. Indeed, most

successful statistical models are developed as part of an iterative procedure, interweaving model refinement with statistical inference Gelman et al. (2020).

This statistical paradigm has underpinned data driven advances in a wide range of fields including but by absolutely no means limited to

- Econometrics, e.g. Creal (2012); Glasserman (2004).
- Natural sciences, e.g. Beaumont et al. (2002); Landau and Binder (2005); van Dyk (2014); Wilkinson (2018); Wood (2010).
- Signal processing, e.g. Doucet et al. (2013); Reich and Cotter (2015).
- Sports, e.g. Biermann (2019); Marchi et al. (2018).
- Catching Youtubers cheating at video games, e.g. Minecraft Speedrunning Team (2020).

Over the past century statistical inference has mostly fallen into one of two paradigms - the **frequentist** approach and the **Bayesian** approach.

Frequentist Inference

The frequentist approach dominated for most of the 20th century and focuses on probabilistic uncertainty surrounding the data y. Recall that $p(y \mid x)$ represents a probability distribution in y - alternatively we might say that y is a *random variable* and the data represents a realisation of the random variable. Frequentist reasoning assumes that there is a single true underlying value of the parameter x - on this basis various different estimators of x can be established that minimise a cost function. These include the maximum likelihood estimator $\hat{x} = \arg\max_x p(y \mid x)$ and method of moments estimators $\hat{x} = \arg\min_x ||g(x,y)||^2$, Hansen (1982) - the utility of the estimator is highly dependent on the form of the data and statistical model. Frequentist uncertainty is reasoned by considering how the parameter behaves under repeated sampling of the data. Indeed a frequentist confidence interval (with confidence δ where $0 < \delta < 1$) represents a function of the data $[L(y), U(y)]$ that states

$$p(L(y) < x < U(y)) \leq 1 - \delta.$$

That is, if we were to randomly generate N different data sets (with N very large) the true value of the parameter x would lie in $[L(y), U(y)]$ for at least δN of those data sets.

Obtaining exact bounds $[L(y), U(y)]$ is typically not easy, often we resort to computationally cheap approximations based on asymptotic theory as the amount of data increases, Young and Smith (2005).

Bayesian Inference

Over the past 30 years, driven somewhat by advances in computational technology, the Bayesian paradigm has seen something of a maturation. The Bayesian philosophy is underpinned by the decision to additionally model the unknown parameter x as a random variable. Subsequently, one can define a probability distribution over values of x we would expect to observe before seeing any data. This is termed the *prior* distribution

$$p(x).$$

On one hand the concept of a prior distribution adds another layer of subjectivity to the statistical model, on the other it adds another layer of flexibility - one can formally encode physical constraints (i.e. a temperature parameter cannot be less than -273.15°C or a vehicle's position lies on a roadway).

Given a well-defined prior and likelihood one can apply **Bayes' Theorem**

$$p(x \mid y) = \frac{p(x)p(y \mid x)}{p(y)},$$

to obtain a probability distribution in x given the value of the data y - the so-called *posterior* distribution.

Bayes' theorem itself represents a fact of probability. The more subjective concept of Bayesian inference is better defined by the use of a prior distribution and indeed the overriding principle of treating x as a random variable.

Assuming we have sufficient knowledge of the posterior distribution to be able to calculate the required integrals, then the concept of *expectation*

$$\mathbb{E}_{p(x|y)}[f(x)] = \int f(x)p(x \mid y)dx,$$

represents an extremely general way to incorporate uncertainty into data driven predictions or metrics of interest. The integral $\mathbb{E}_{p(x|y)}[f(x)]$ corresponds to the expected value of the function $f(x)$ given that x is distributed according to the posterior $p(x \mid y)$. Importantly, expectations taken with respect to the posterior distribution do not assume the parameter x takes a single value rather it is distributed over a range, with regions of high probability depending on the value of the observed data.

The integration required to compute expectations of interest for non-trivial models can be very computationally expensive, particularly when the parameter x is high dimensional and/or the statistical model is complex.

Online Data

In many practical settings, data is arriving in real-time and we are expected to provide statistical reasoning instantaneously as the new data arrives. We refer to this setting as *online*, in contrast to *offline* or *static* settings where the amount of data is fixed.

Posterior expectations make no assumptions over the amount of data gathered, as such the Bayesian paradigm is very powerful in the face of only a few observations. This can be extended to the case where inference is required as data arrives sequentially - i.e. online inference.

One option for handling online data is to apply iterative Bayesian posterior updates

$$p(x \mid y_{0:T}) = \frac{p(x \mid y_{0:T-1})p(y_T \mid x)}{p(y_T \mid y_{0:T-1})},$$

where $y_{0:T-1}$ is an array containing the first $T-1$ observations and $y_{0:T}$ is the same array extended to include a new observation y_T.

For many online settings, it is more useful to model the state of the underlying parameter as variable with time i.e. $x \to x_T$. This increases the complexity of the statistical model as we are no longer interested in the state of a single parameter but rather the state of a parameter as a function of time. Fortunately, there exists a convenient and general *state-space model* framework Chopin and Papaspiliopoulos (2020) that will allow us to execute fully Bayesian online inference for the posterior distributions $p(x_T \mid y_{0:T})$ and even $p(x_{0:T} \mid y_{0:T})$.

Outline

Chapter 2 presents a thorough review of *Monte Carlo* methods, which underpins all subsequent chapters. Monte Carlo methods approximate a probability distribution of interest (i.e. posterior distribution) by a collection of representative points - which we refer to synonymously as samples or particles. Once samples have been generated they provide a very fast and flexible method of numerical integration through expectation.

Chapter 3 develops a new Monte Carlo method that approximates the full historical posterior distribution $p(x_{0:T} \mid y_{0:T})$ for online inference in state-space models. In doing so we provide a solution to the well-known problem of path degeneracy for Monte Carlo methods in state-space models that has previously limited online inference to marginals $p(x_t \mid y_{0:T})$ rather than $p(x_{0:T} \mid y_{0:T})$.

In Chapter 4 a novel statistical model is developed for the problem of map-matching - converting a series of noisy, unconstrained GPS coordinates into a logical trajectory constrained to a road network. The particularly difficult case of dense, urban road networks is examined

where the data may provide significant uncertainty over the underlying trajectory - motivating a Bayesian approach. Urban map-matching also represents a compelling application of the online inference technique introduced in Chapter 3 which is demonstrated on real data. Chapter 4 is accompanied by `bmm`, an open source `python` package that provides easy and flexible map-matching using the discussed Monte Carlo methods for offline and online settings.

Chapter 5 considers offline or static inference in the constrained scenario where the likelihood function $p(y \mid x)$ cannot be evaluated due to computational restraints. In these settings, approximate inference is still possible when the generation of synthetic data from the likelihood is available and relatively cheap. We apply ideas from ensemble Kalman filtering (a collection of numerically efficient algorithms originating from meteorological applications) to this very general setting of *intractable likelihoods* for both maximum likelihood optimisation and uncertainty quantification.

In Chapter 6 we consider offline Bayesian inference problems where the posterior distribution is differentiable - as made possible by advances in modern automatic differentiation software. We borrow established optimisation techniques to leverage the first derivative (gradient) of the posterior distribution into a cheap and scalable approximation of the second derivative (Hessian). In doing so, we characterise the local scaling of the posterior distribution and accelerate Bayesian inference for difficult high-dimensional problems.

Chapter 7 presents a second `python` package, `mocat` that provides a flexible and fast Monte Carlo framework for a wide variety of Bayesian inference tasks. This includes the methods introduced in Chapter 3, Chapter 5, Chapter 6 and much more!

Notation

$\mathbb{I}(condition)$ Identity function, 1 if *condition* otherwise 0

$\mathbb{I}_d$ Identity matrix - diagonal matrix $\in \mathbb{R}^{d \times d}$ with all ones on the diagonal

$\delta(x \mid y)$ Dirac (point) distribution in the random variable x with all mass at the value y

$\mathbf{N}(x \mid \mu, \Sigma)$ Gaussian distribution in the random variable x with mean μ and covariance Σ

$\mathbf{U}(x \mid a, b)$ Uniform distribution in the random variable x over (a, b)

$x \sim \pi$ A sample or simulation with value x from the distribution π

Chapter 2

Background

Monte Carlo methods provide a very general approach to approximating integrals

$$\mathbb{E}_\pi[f(x)] = \int f(x)\pi(x)dx, \tag{2.1}$$

where π is a well defined probability distribution.

In the case we can generate samples from π directly and $\mathbb{E}_\pi[f(x)] < \infty$ then the law of large numbers tells us that

$$\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} f(x^{(i)}) = \mathbb{E}_\pi[f(x)], \qquad x^{(i)} \sim \pi. \tag{2.2}$$

And so by taking a suitably large number of samples we can approximate $\mathbb{E}_\pi[f(x)]$ to arbitrary precision.

Algorithm 1 Pure Monte Carlo

1: Sample $x^{(i)} \sim \pi$ $\qquad i = 1,\ldots,N$
2: **return** $\{x^{(i)}\}_{i=1}^{N}$

But why is it called **Monte Carlo**?

Monte Carlo simulation dates back to the work of physicist Enrico Fermi before being refined by John von Neumann and Stanislaw Ulam. But it was their colleague Nicholas Metropolis that coined the term Monte Carlo, likening the element of chance in the method to that of casino games, abundant in Monte Carlo, Monaco.

2.1 Importance Sampling

In most practical settings we cannot sample directly from π. We may however be able to evaluate its density $\pi(x)$ for any given x and additionally have access to an alternative importance distribution q which we can sample from and also evaluate its density $q(x)$.

We can then write

$$\begin{aligned}\mathbb{E}_\pi[f(x)] &= \int f(x)\pi(x)dx, \\ &= \int \frac{f(x)\pi(x)}{q(x)} q(x)dx.\end{aligned}$$

and use Equation (2.2) to get a consistent importance sampling estimator

$$\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} \frac{f(x^{(i)})\pi(x^{(i)})}{q(x^{(i)})} = \mathbb{E}_\pi[f(x)], \qquad x^{(i)} \sim q.$$

We can write the above estimator as $\frac{1}{N}\sum_{i=1}^{N} f(x^{(i)})w^{(i)}$ with importance weights $w^{(i)} = \frac{\pi(x^{(i)})}{q(x^{(i)})}$.

But why is it called Importance Sampling?

The variance of the Monte Carlo estimator can be improved by instead sampling from $q(x) \propto f(x)\pi(x)$ - the *important* regions of the integrand.

The task of choosing an importance distribution is often a difficult one. It is natural to attempt to minimise the variance of the single-sample importance estimator

$$\mathrm{Cov}_q\left[\frac{f(x)\pi(x)}{q(x)}\right] = \mathbb{E}_\pi\left[\frac{f(x)^2\pi(x)}{q(x)}\right] - \mathbb{E}_\pi[f(x)]^2. \tag{2.3}$$

From which it can be seen that the optimal choice of importance distribution is $q(x) \propto f(x)\pi(x)$ and that it produces a zero variance estimator. Indeed the normalising constant for this choice of proposal distribution is exactly the expectation of interest.

In many statistical tasks we are interested in using samples to take expectations over a variety of different functions. Chatterjee and Diaconis (2018) show that in terms of expected absolute deviation, the most difficult test function is $f(x) = 1\ \forall x$, and that the number of samples required increases proportional to $\exp(D_{\mathrm{KL}}(\pi||q))$, where D_{KL} is the Kullback-Leibler divergence. This motivates a guiding principle of choosing the importance distribution q to be close to π.

2.1.1 Self-Normalised Importance Sampling

In the above we have assumed that we can evaluate the density $\pi(x)$ exactly. It is much more common, particularly in Bayesian inference, to only have access to evaluations of π up to a normalisation constant. That is

$$\pi(x) = \frac{\gamma(x)}{Z},$$

where we can evaluate $\gamma(x)$ exactly but not $Z = \int \gamma(x)dx$.

We can navigate this hurdle by approximating the normalisation constant with importance samples

$$\begin{aligned} Z &= \int \gamma(x)dx, \\ &= \int \frac{\gamma(x)}{q(x)} q(x)dx. \end{aligned}$$

and so

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} \frac{\gamma(x^{(i)})}{q(x^{(i)})} = Z. \tag{2.4}$$

Consequently, most importance samplers self-normalise the weights

$$\begin{aligned} w^{(i)} &= \frac{\frac{\gamma(x^{(i)})}{q(x^{(i)})}}{\sum_{j=1}^{N} \frac{\gamma(x^{(j)})}{q(x^{(j)})}}, \\ \lim_{N\to\infty} \sum_{i=1}^{N} w^{(i)} f(x^{(i)}) &= \mathbb{E}_{\pi}[f(x)], \qquad x^{(i)} \sim q. \end{aligned} \tag{2.5}$$

We often write $w^{(i)} \propto \frac{\gamma(x^{(i)})}{q(x^{(i)})}$ where self-normalisation in i is implied.

Note that we have assumed that the importance density q is itself normalised. In the case that it is not, the mean of the weights Equation (2.4), then approximates the ratio of normalising constants Z/Z_q instead, but the estimator in Equation (2.5) is still consistent.

Algorithm 2 Self-Normalised Importance Sampling

1: Sample $x^{(i)} \sim q$ $\quad i = 1,\dots,N$
2: Normalise $\hat{Z} = \frac{1}{N}\sum_{i=1}^{N} \frac{\gamma(x^{(i)})}{q(x^{(i)})}$
3: Weight $w^{(i)} = \frac{\gamma(x^{(i)})}{q(x^{(i)})N\hat{Z}}$ $\quad i = 1,\dots,N$
4: **return** $\{x^{(i)}, w^{(i)}\}_{i=1}^{N}$

2.2 Rejection Control

In some settings, it may be inefficient to store samples with low importance weights. We can then consider the algorithm introduced in Liu et al. (1998) that only retains a sample $x \sim q$ with probability $\min\left(1, \frac{\gamma(x)}{q(x)C}\right)$ for parameter C that controls the acceptance rate.

Algorithm 3 Rejection Control

for $i = 1, \ldots, N$ **do**
  Sample $x' \sim q$
  Sample $u \sim \mathbf{U}(\cdot \mid 0, 1)$
  if $u < \min\left(1, \frac{\gamma(x')}{q(x')C}\right)$ **then** set $x^{(i)} = x'$
  else go to 2
if $C > \frac{\gamma(x^{(i)})}{q(x^{(i)})}$ $\forall i$ **then**
  return $\{x^{(i)}\}_{i=1}^N$
else
  Normalise $\hat{Z} = \frac{1}{N}\sum_{i=1}^N \max\left(\frac{\gamma(x^{(i)})}{q(x^{(i)})}, C\right)$
  Weight $w^{(i)} = \max\left(\frac{\gamma(x^{(i)})}{q(x^{(i)})}, C\right)/N\hat{Z}$ $i = 1 \ldots, N$
  return $\{x^{(i)}, w^{(i)}\}_{i=1}^N$

But why is it called Rejection Control?

Pure *rejection* sampling was first used by John von Neumann and the term refers to the process of discarding low probability proposals. The introduction of the C parameter allows the user to *control* the acceptance rate.

This algorithm can be viewed as an importance sampler with adjusted importance distribution

$$q^*(x) \propto \min\left(1, \frac{\gamma(x)}{q(x)C}\right) q(x),$$

and so comes with accompanying importance weights

$$\begin{aligned} w &= \frac{\pi(x)}{q^*(x)}, \\ &\propto \frac{\gamma(x)}{\min\left(1, \frac{\gamma(x)}{q(x)C}\right) q(x)}, \\ &\propto \max\left(\frac{\gamma(x)}{q(x)}, C\right). \end{aligned}$$

In this light we can see that the choice of $C = 0$, naturally accepts all proposals and recovers pure importance sampling. On the other hand taking C sufficiently large such that $C > \frac{\gamma(x)}{q(x)}$ $\forall x$ produces weights that are all equal (and as such unweighted), this is known as (pure) *rejection sampling*.

The parameter C trades off acceptance rate vs sample quality. When memory is no issue and the particles are not reused in a recursive manner it is desirable to set $C = 0$ and retain the pure importance sampling setup where all generated particles contribute to expectations - i.e. the algorithm is *waste-free*. In memory constrained or iterative settings it may be desirable to generate a weighted sample where each particle has a significant weight as achieved by using

a non-zero C. A convenient feature of rejection control is that C can be set adaptively after generating an initial N proposals from q. For example, we could set C to the αth quantile of $\{w'^{(i)}\}_{i=1}^{N}$ where α is some desired acceptance rate, $w'^{(i)} \propto \frac{\gamma(x'^{(i)})}{q(x'^{(i)})}$ and $x'^{(i)} \sim q$ are initial proposals that can be subsequently sent to the rejection control sampler after setting C - the adaptation therefore comes at no additional cost.

2.3 Sequential Monte Carlo

The theory of importance sampling can be extended to the case where expectations are taken over a sequence of target distributions on a state-space of increasing dimension

$$\pi_{0:t}(x_{0:t}), \qquad t = 0, \ldots, T. \tag{2.6}$$

This concept of sequential importance sampling was first introduced by Gordon et al. (1993) in the specific context of state-space models 2.6 and then fully generalised to any sequence of targets in Del Moral et al. (2006).

2.3.1 Reweighting

Let us assume we have a weighted sample $(x_{0:t-1}, w_{t-1})$ from $\pi_{0:t-1}$ and desire a sample from $\pi_{0:t}$. We can then extend our sample by proposing from a conditional importance distribution

$$x_t \sim q_{t|0:t-1}(x_t|x_{0:t-1}), \tag{2.7}$$

and weight over the entire space

$$\begin{aligned} w_t &= \frac{\pi_{0:t}(x_{0:t})}{q_{0:t}(x_{0:t})}, \\ &= \frac{\pi_{0:t}(x_{0:t})}{q_{0:t-1}(x_{0:t-1})q_{t|0:t-1}(x_t|x_{0:t-1})}, \\ &= \frac{\pi_{0:t-1}(x_{0:t-1})}{q_{0:t-1}(x_{0:t-1}))} \frac{\pi_{0:t}(x_{0:t})}{\pi_{0:t-1}(x_{0:t-1})q_{t|0:t-1}(x_t|x_{0:t-1})}, \\ &= w_{t-1} \frac{\pi_{0:t}(x_{0:t})}{\pi_{0:t-1}(x_{0:t-1})q_{t|0:t-1}(x_t|x_{0:t-1})}. \end{aligned}$$

In practice, as described in 2.1.1, we may not know the normalising constant of $\pi_{0:t}$ and can instead replace it with unnormalised $\gamma_{0:t}$ where $\pi_{0:t}(x_{0:t}) = \gamma_{0:t}(x_{0:t})/Z_{0:t}$ and the normalising constant $Z_{0:t}$ can be approximated by normalising the sequential importance weights.

It is common to focus on the latest marginal distribution $\pi_t(x_t) = \int \pi_{0:t}(x_{0:t})dx_{0:t-1}$ rather than the full joint distribution. In this case we can rewrite the weights as

$$w_t = w_{t-1} \frac{\pi_t(x_t)\pi_{0:t-1|t}(x_{0:t-1}|x_t)}{\pi_{0:t-1}(x_{0:t-1})q_{t|0:t-1}(x_t|x_{0:t-1})}. \tag{2.8}$$

where $\pi_{0:t-1|t}(x_{0:t-1}|x_t)$ is a normalised auxiliary distribution that we are free to choose. In this setting where we care only about x_t and discard $x_{0:t-1}$, the optimal choice of $\pi_{0:t-1|t}(x_{0:t-1}|x_t)$ (that which minimises the variance of the weights) is shown in Del Moral et al. (2006) to be

$$\pi^{\text{opt}}_{0:t-1|t}(x_{0:t-1}|x_t) = q_{0:t-1|t}(x_{0:t-1}|x_t) = \frac{q_{0:t-1}(x_{0:t-1})q_{t|0:t-1}(x_t|x_{0:t-1})}{q_t(x_t)}, \tag{2.9}$$

where $q_t(x_t) = \int q_{0:t-1}(x_{0:t-1})q_{t|0:t-1}(x_t|x_{0:t-1})dx_{0:t-1}$ ensures the density is normalised. The importance weights then act only in the reduced space

$$w^{\text{opt}}_t = \frac{\pi_t(x_t)}{q_t(x_t)}, \tag{2.10}$$

despite the samples being generated sequentially.

Unfortunately, the integration required in the calculation of q_t is rarely tractable. However, as we will see, suboptimal but tractable backward kernels have been used with great success in a variety of settings and of course still inherit the consistency of estimators from Equation (2.5).

2.3.2 Resampling

Iterating the weight update, Equation (2.8), can lead to an accumulation in the variance of the weights. This can be mitigated using a resampling technique. Resampling converts a weighted sample $\left\{x^{(i)}_{0:t}, w^{(i)}_t\right\}_{i=1}^N$ into an unweighted sample $\left\{x^{(i)}_{0:t}, \frac{1}{N}\right\}_{i=1}^N$ that may contain duplicates. There are a variety of resampling techniques, for a review see Douc and Cappe (2005), here we only describe the most popular multinomial resampling (with replacement)

$$\begin{aligned} &\text{Sample} && a^{(i)}_t \sim \text{Categorical}\left(\left\{w^{(j)}_t\right\}_{j=1}^N\right), && i = 1,\ldots,N, \\ &\text{Set} && x^{(i)}_{0:t} \leftarrow x^{(a^{(i)}_t)}_{0:t} \quad w^{(i)}_t \leftarrow \frac{1}{N}, && i = 1,\ldots,N. \end{aligned} \tag{2.11}$$

Where $\text{Categorical}(\{w^{(i)}\}_{i=1}^N) = \sum_{i=1}^N w^{(i)}\delta(\cdot \mid i)$ simply draws an index i from the index set $\{1,\ldots,N\}$ with probabilities $w^{(i)}$.

Algorithm 4 Sequential Monte Carlo

1: Sample $x_0^{(i)} \sim q_0$ $\quad i = 1\ldots,N$

2: Normalise $\hat{Z}_0 = \frac{1}{N}\sum_{i=1}^{N} \frac{\gamma_0(x_0^{(i)})}{q_0(x_0^{(i)})}$

3: Weight $w_0^{(i)} = \frac{\gamma_0(x_0^{(i)})}{q_0(x_0^{(i)})N\hat{Z}_0}$ $\quad i = 1\ldots,N$

4: **for** $t = 1,\ldots,T$ **do**

5: **if** `resampling criterion` **then**

6: $\left\{\tilde{x}_{0:t-1}^{(i)}, \tilde{w}_{t-1}^{(i)} = \frac{1}{N}\right\}_{i=1}^{N} = \text{Resample}\left(\left\{x_{0:t-1}^{(i)}, w_{t-1}^{(i)}\right\}_{i=1}^{N}\right)$

7: **else**

8: $\left\{\tilde{x}_{0:t-1}^{(i)}, \tilde{w}_{t-1}^{(i)}\right\}_{i=1}^{N} = \left\{x_{0:t-1}^{(i)}, w_{t-1}^{(i)}\right\}_{i=1}^{N}$

9: Propose

$$x_t^{(i)} \sim q_{t|0:t-1}(x_t \mid \tilde{x}_{0:t-1}^{(i)}) \qquad i = 1\ldots,N$$

10: Normalise

$$\hat{Z}_{t|0:t-1} = \sum_{i=1}^{N} \tilde{w}_{t-1}^{(i)} \frac{\gamma_t(x_t^{(i)})\gamma_{0:t-1|t}(\tilde{x}_{0:t-1}^{(i)}|x_t^{(i)})}{\gamma_{0:t-1}(\tilde{x}_{0:t-1}^{(i)})q_{t|0:t-1}(x_t^{(i)}|\tilde{x}_{0:t-1}^{(i)})}$$

11: Reweight

$$w_t^{(i)} = \tilde{w}_{t-1}^{(i)} \frac{\gamma_t(x_t^{(i)})\gamma_{0:t-1|t}(\tilde{x}_{0:t-1}^{(i)}|x_t^{(i)})}{\gamma_{0:t-1}(\tilde{x}_{0:t-1}^{(i)})q_{t|0:t-1}(x_t^{(i)}|\tilde{x}_{0:t-1}^{(i)})\hat{Z}_{t|0:t-1}} \qquad i = 1\ldots,N$$

12: **return** $\left\{\left\{x_t^{(i)}, w_t^{(i)}\right\}_{i=1}^{N}\right\}_{t=0}^{T}$

Resampling is a rejuvenation mechanism that ensures particles with negligible weights are not proposed at the next iteration. However, resampling should only be applied if particles need rejuvenating as resampling can only reduce sample diversity - the number of unique particles. As such it is common to resample when some resampling criterion is met. The most common such criterion resamples when the effective sample size (ESS) Kong et al. (1994) falls below some predefined threshold

$$\begin{aligned} \mathrm{ESS}(\{w^{(i)}\}_{i=1}^N) &= \frac{(\sum_{i=1}^N w^{(i)})^2}{\sum_{i=1}^N w^{(i)\,2}}, && (2.12)\\ &= \frac{1}{\sum_{i=1}^N w^{(i)\,2}}. && \text{(for normalised weights)} \end{aligned}$$

Conveniently, $\mathrm{ESS}(\{w^{(i)}\}_{i=1}^N) \in [1,N]$ and for uniform weights (i.e. unweighted) we have $\mathrm{ESS}(\{N^{-1}\}_{i=1}^N) = N$. The effective sample size can be thought of as an empirical approximation to (a transformation of) the χ^2-divergence between target π and importance distribution q Chopin and Papaspiliopoulos (2020) as

$$\mathrm{ESS}(\{w^{(i)}\}_{i=1}^N) \xrightarrow{N\to\infty} \frac{N}{1+\chi^2(\pi||q)}.$$

Where $\chi^2(\pi||q) = \mathbb{E}_q[(\frac{\pi(x)}{q(x)} - 1)^2]$ is non-negative and $\chi^2(\pi||\pi) = 0$.

The composition of optional resampling (2.11), proposal (2.7) and reweighting (2.8) results in a single iteration of sequential Monte Carlo (SMC) in its most general form, Algorithm 4.

But why is it called effective sample size?

The term *effective sample size* is used to refer to any metric of sample quality that (typicallly) lies in $[1,N]$ such that exact samples from π have an ESS of N. Indeed in MCMC, an alternative ESS is used based on the autocorrelation of the chain.

2.4 Markov Chain Monte Carlo

An alternative approach to the aforementioned importance sampling regimes is that of Markov chain Monte Carlo (MCMC).

In MCMC, a Markov chain is formed based on some kernel $K(y \mid x)$. We say that a kernel is π-invariant (or admits π as a stationary distribution) if

$$\int \pi(x)K(y \mid x)dx = \pi(y). \tag{2.13}$$

This invariance means that if we were to take a sample $x \sim \pi(\cdot)$ and generate $y \sim K(\cdot \mid x)$ then our new sample would also have marginal distribution π. The implication is that under certain conditions (informally[1][2][3]) a law of large numbers applies to the scheme that iterates sampling $x^{(i)} \sim K(\cdot \mid x^{(i-1)})$ with $x^{(0)} \sim K_0$, when $K_0 = \pi$. We can therefore approximate expectations of interest using the estimator in Equation (2.2). For a full theoretical treatment including kernel dependent convergence rates for when $K_0 \neq \pi$ see Meyn and Tweedie (1993) or more recently Douc et al. (2018).

A Markov kernel is said to be π-reversible if

$$\pi(x)K(y \mid x) = \pi(y)K(x \mid y). \tag{2.14}$$

We can show that π-reversibility implies π-invariance (2.13), indeed

$$\begin{aligned}\int \pi(x)K(y \mid x)dx &= \int \pi(y)K(x \mid y)dx,\\ &= \pi(y)\int K(x \mid y)dx,\\ &= \pi(y).\end{aligned}$$

It is typically easier to verify π-reversibility rather than π-invariance directly, as we will see in the next two sections.

Note that although π-reversibility implies π-invariance the reverse is not necessarily true, indeed the composition of π-reversible kernels are typically not π-reversible but π-invariance is preserved under composition.

But why is it called Markov Chain Monte Carlo?

A(n) (Andrey) Markov chain is any discrete-time stochastic process such that $p(x^{(i)} \mid x^{(0:i-1)}) = p(x^{(i)} \mid x^{(i-1)})$. Thus, MCMC forms a Markov chain by iterating the same π-invariant kernel and taking Monte Carlo averages along the final chain.

2.4.1 Gibbs Sampling

Suppose we can partition our space $x = (x_a)_{a\in\mathcal{A}}$ such that we can sample from each full conditional $\pi(x_a|x_{-a})$ where $x_{-a} = (x_b)_{b\in\mathcal{A}\setminus a}$. Then a Gibbs sampler iteratively selects an index $a \in \mathcal{A}$ either randomly or deterministically and then samples

$$y_a \sim \pi(\,\cdot \mid x_{-a}), \tag{2.15}$$

before setting $y = y_a \cup x_{-a}$.

[1] Aperiodicity - the Markov chain does not have a cyclic structure.
[2] Irreducibility - the Markov chain explores all regions of the state-space with positive probability under π.
[3] Ergodicity - π is the only distribution for which Equation (2.13) holds.

We can show for a given index, Gibbs sampling is π-reversible

$$\begin{aligned}
\pi(x)K_a(y \mid x) &= \pi(x)\pi(y_a \mid x_{-a})\delta(y_{-a} \mid x_{-a}), \\
&= \pi(x_{-a})\pi(x_a \mid x_{-a})\pi(y_a \mid x_{-a})\delta(y_{-a} \mid x_{-a}), \\
&= \pi(y_{-a})\pi(x_a \mid y_{-a})\pi(y_a \mid y_{-a})\delta(x_{-a} \mid y_{-a}), \qquad [y_{-a} = x_{-a}] \\
&= \pi(y)\pi(x_a \mid y_{-a})\delta(x_{-a} \mid y_{-a}), \\
&= \pi(y)K_a(x \mid y).
\end{aligned}$$

Gibbs sampling was first introduced in Geman and Geman (1984) and provides a very general approach for the construction of Markovian kernels that leverage the structure of the state space, either directly or through the use of auxiliary variables.

Algorithm 5 Gibbs Sampler

1: Sample $x^{(0)} \sim K_0$
2: **for** $i = 1, \ldots, N$ **do**
3: Select a either deterministically or randomly
4: Sample $x^{(i)} \sim \pi(x_a \mid x_{-a}^{(i-1)})\delta(x_{-a} \mid x_{-a}^{(i-1)})$
5: **return** $\{x^{(i)}\}_{i=1}^{N}$

2.4.2 Accept-Reject

An alternative way to ensure π-reversibility is to adopt an accept-reject approach where a sample is proposed $y \sim q(\cdot \mid x)$ from a proposal distribution q but only accepted with a certain probability $\alpha(x,y)$ and otherwise the previous state in the chain x is duplicated.

As described in Tierney (1998), a Markov kernel of this ilk can be expressed as

$$K(y \mid x) = \alpha(x,y)q(y \mid x) + \delta(y \mid x)\int (1 - \alpha(x,x'))q(x' \mid x)dx'. \tag{2.16}$$

In deriving valid acceptance probabilities, $\alpha(x,y)$, it will be useful to consider *balancing functions* $g : [0,\infty) \to [0,1]$ that satisfy

$$g(r) = rg(1/r), \qquad r > 0, \tag{2.17}$$

with $g(0) = 0$.

Stochastic Proposals

Suppose that q has positive probability everywhere π does, then we can show that setting

$$\alpha(x,y) = g\left(\frac{\pi(y)q(x \mid y)}{\pi(x)q(y \mid x)}\right) \tag{2.18}$$

induces a kernel that is π-reversible. Indeed, Equation (2.14) is satisfied trivially when $y = x$ so for $y \neq x$ we get

$$\begin{aligned}
\pi(x)K(y \mid x) &= \pi(x)q(y \mid x)\; g\left(\frac{\pi(y)q(x \mid y)}{\pi(x)q(y \mid x)}\right), \\
&= \pi(y)q(x \mid y)\; g\left(\frac{\pi(x)q(y \mid x)}{\pi(y)q(x \mid y)}\right), \\
&= \pi(y)K(x \mid y).
\end{aligned}$$

Therefore Equation (2.17) provides a general framework for constructing accept-reject kernels that are π-reversible for any proposal distribution $q(y \mid x)$ that has positive probability everywhere π does.

A significant property of the ratio $\frac{\pi(y)q(x|y)}{\pi(x)q(y|x)}$ is that it does not require the target distribution π to be normalised. Thus, as in importance sampling, Section 2.1.1, we can approximate expectations with respect to unnormalised targets. However, unlike importance sampling type approaches, Markov chain Monte Carlo does not directly provide us with an estimate of the normalising constant.

Deterministic Proposals

In the case that the proposal is deterministic $q(y \mid x) = \delta(y \mid T(x))$ and T is an involution $T = T^{-1}$. We can set

$$\alpha(x,y) = g\left(\frac{\pi(y)}{\pi(x)}|\nabla T(x)|\right), \tag{2.19}$$

with Jacobian matrix $[\nabla T(x)]_{ij} = \frac{\partial}{\partial x_{ij}} T(x)$ and a balancing function g from Equation (2.17). For $y \neq x$ we can then check for π-reversibility

$$\begin{aligned}
\pi(x)K(y|x) &= \pi(x)\delta(y \mid T(x))\; g\left(\frac{\pi(y)}{\pi(x)}|\nabla T(x)|\right), \\
&= \pi(y)\delta(y \mid T(x))|\nabla T(x)|\; g\left(\frac{\pi(x)}{\pi(y)}\frac{1}{|\nabla T(x)|}\right), \\
&= \pi(y)\delta(x \mid T(y))\; g\left(\frac{\pi(x)}{\pi(y)}|\nabla T(y)|\right), \\
&= \pi(y)K(x|y).
\end{aligned}$$

As $|\nabla T(y)| = |\nabla T^{-1}(y)| = |[\nabla T(x)]^{-1}| = |\nabla T(x)|^{-1}$ by the inverse function theorem and $\delta(x \mid T(y)) = \delta(y \mid T^{-1}(x))|\nabla T^{-1}(x)| = \delta(y \mid T(x))|\nabla T(x)|$ by the change of variables formula for random variables and the involution property $T = T^{-1}$.

Note that to take consistent ergodic averages we still need to ensure irreducibility and aperiodicity which often are not characteristics of deterministic proposals. To overcome this we can compose a deterministic accept-reject kernel with a stochastic kernel that is also π-invariant.

Choice of Balancing Function

One example of a valid acceptance probability is $g_{\mathrm{B}}(r) = \frac{r}{1+r}$ Barker (1965) but the most popular is that of Metropolis-Hastings $g_{\mathrm{MH}}(r) = \min(1, r)$ Metropolis et al. (1953), Hastings (1970). The popularity of the Metropolis-Hastings acceptance probability can be somewhat attributed to Peskun (1973) who showed that it is optimal amongst kernels of the form in Equation (2.16). Where said optimality is defined in terms of asymptotic variance in Monte Carlo estimators $\frac{1}{N}\sum_{i=1}^{N} f(x^{(i)})$, where the key observation is that g_{MH} maximises the probability of moving from x to a new y.

Now that we have established some general accept-reject rules in both stochastic and deterministic settings we can describe some of the most frequently used proposals.

Random Walk

Possibly the simplest choice of proposal is

$$q(y \mid x) = \mathbf{N}(y \mid x, \varepsilon \mathrm{D}), \tag{2.20}$$

for stepsize $\varepsilon \in (0, \infty)$ and preconditioning matrix $\mathrm{D} \in \mathbb{R}^{d \times d}$ which is often simply set to the identity $\mathbb{I}_d$. A useful property of the random walk proposal is that $q(y|x) = q(x|y)$ and thus the

Algorithm 6 Accept-Reject MCMC

Sample $x^{(0)} \sim K_0$
for $i = 1, \ldots, N$ **do**
    Propose $x' \sim q(\cdot \mid x^{(i-1)})$
    Set α according to Equation (2.18) or Equation (2.19)
    Sample $u \sim \mathbf{U}(\cdot \mid 0, 1)$
    if $u < \alpha$ **then**
        $x^{(i)} = x'$
    else
        $x^{(i)} = x^{(i-1)}$
return $\{x^{(i)}\}_{i=1}^{N}$

acceptance probability reduces to $\alpha(x,y) = g\left(\frac{\pi(y)}{\pi(x)}\right)$. Much is to be said for the simplicity of the random walk proposal - not least from a coding perspective - however without utilising any π specific information in the proposal it can perform very poorly for difficult or structured problems.

Gradient informed proposals

It is natural to consider a Markov proposal as a discretisation of a particular stochastic differential equation (SDE), indeed the random walk proposal is an exact discretisation of the Brownian motion $dx_t = \sqrt{\mathrm{D}}dW_t$.

In the case that we have an SDE with π as a stationary distribution and we discretised it perfectly all proposals would be accepted by the definition of stationarity. Fundamentally Ma et al. (2015) show that an SDE of the form

$$dx_t = b(x_t)dt + \sqrt{2\mathrm{D}(x_t)}dW_t, \tag{2.21}$$

has stationary distribution π if and only if the drift term $b(x)$ can be written in the form

$$\begin{aligned} b(x) &= -(\mathrm{D}(x) + \mathrm{Q}(x))\nabla U(x) + \Gamma(x), \\ \Gamma_i(x) &= \sum_{j=1}^{d} \frac{\partial}{\partial x_j}(\mathrm{D}_{ij}(x) + \mathrm{Q}_{ij}(x)). \end{aligned} \tag{2.22}$$

where U is the potential function i.e. $\pi(x) = \exp(-U(x))$, D is symmetric, positive semidefinite and Q is skew-symmetric, i.e. $\mathrm{Q} = -\mathrm{Q}^{\mathrm{T}}$.

Note that there still exist stochastic processes that do not take the form in Equation (2.21) (e.g. jump processes) but admit π as a stationary distribution.

Thus a gradient informed proposal with tractable density can be obtained with an Euler-Maruyama discretisation

$$q(y \mid x) = \mathbf{N}(y \mid x + \varepsilon b(x), 2\varepsilon \mathrm{D}(x)), \tag{2.23}$$

which will be nearly π-reversible for small stepsize ε.

The choice of $\mathrm{D}(x) = \mathrm{D}$ and $\mathrm{Q} = 0$ induces an SDE corresponding to (preconditioned) *overdamped Langevin* dynamics. This combined with the Euler-Maruyama discretisation and the Metropolis-Hastings acceptance probability gives rise to a (preconditioned) *Metropolis adjusted Langevin algorithm* (MALA).

A very popular approach in constructing gradient informed proposals is to introduce an auxiliary variable $v \in \mathbb{R}^d$, termed *momenta*. Defining an extended target which is Gaussian $\mathcal{N}(v \mid 0, \mathrm{M})$ in the momenta, gives the extended potential

$$U(x,v) = U(x) + \frac{1}{2} v^{\mathrm{T}} \mathrm{M}^{-1} v, \tag{2.24}$$

here M is a preconditioning matrix that is often set to $\mathbb{I}_d$. We can obtain the more general *Langevin dynamics* (or *underdamped Langevin dynamics*) by defining Equation (2.22) on this extended space and setting

$$\mathrm{D}(x,v) = \begin{pmatrix} 0 & 0 \\ 0 & \gamma \mathbb{I}_d \end{pmatrix}, \qquad \mathrm{Q}(x,v) = \begin{pmatrix} 0 & -\mathbb{I}_d \\ \mathbb{I}_d & 0 \end{pmatrix}, \tag{2.25}$$

where γ is a *friction* parameter.

Taking the limit $\gamma \to \infty$ induces an SDE that can be solved for v. This solution is an SDE only in x that exactly corresponds to that of overdamped Langevin dynamics.

Setting $\gamma = 0$ induces a fully deterministic SDE, i.e. ordinary differential equation (ODE). This ODE (or system of ODEs) is commonly referred to as *Hamiltonian dynamics*.

The strength of this construction, in terms of practical simulation, is that there exists a variety of efficient discretisations for Hamiltonian dynamics that are *volume preserving*, i.e. $|\nabla T(x)| = 1$ and reversible $T(x,v) = (x',v') \implies T(x',-v') = (x,v)$ so we can obtain π-reversibility with an acceptance probability from Equation (2.19) (defined on the extended

target Equation (2.24)). The most commonly used such discretisation is the *leapfrog integrator*

$$\begin{aligned} v_{t+\varepsilon/2} &= v_t - \frac{\varepsilon}{2}\nabla U(x_t), \\ x_{t+\varepsilon} &= x_t + \varepsilon \mathrm{M}^{-1} v_{t+\varepsilon/2}, \\ v_{t+\varepsilon} &= v_{t+\varepsilon/2} - \frac{\varepsilon}{2}\nabla U(x_{t+\varepsilon}). \end{aligned} \tag{2.26}$$

Reversibility and volume preservation are preserved under composition, therefore multiple steps of the leapfrog (or other) integrator are often taken. We then flip the momenta, $v = -v$, to ensure an involution before applying the accept-reject step. For alternative discretisations as well as proofs of reversibility and volume preservation see Leimkuhler and Matthews (2015).

Deterministic discretisations of Hamiltonian dynamics are typically interweaved with *momenta refreshment* to ensure irreducibility and aperiodicity. In the case of *Hamiltonian Monte Carlo* this is done by simply sampling the momenta v afresh from $\mathcal{N}(0, \mathbb{I}_d)$ which is π-invariant for the extended π as it represents a single step of a Gibbs sampler. Although for non-zero friction γ it is possible to partially refresh momenta by splitting the SDE in Equation (2.25) and solving the corresponding Ornstein-Uhlenbeck SDE in v exactly - this is discussed in detail in Section 6.2.

By utilising gradient information the samplers described above can typically explore the state space much faster than a random walk.

Stepsize Tuning

For all of the described accept-reject MCMC samplers a stepsize parameter ε and preconditioning matrix D have to be tuned. For random walk Metropolis-Hastings on a Gaussian target $\pi(x) = \mathcal{N}(x|0,\Sigma)$ it can be shown Rosenthal (2009) that the optimal stepsize and preconditioning matrix combination is

$$\varepsilon \mathrm{D} = \frac{2.38^2}{d}\Sigma. \tag{2.27}$$

It is difficult to reason much further on the optimality of the static preconditioning matrix D for general target distributions, thus choosing D to approximate the covariance of the target π (either adaptively Andrieu and Thoms (2008) or if further information about π is available) is a commonly applied heuristic.

In the seminal work of Roberts and Rosenthal (2001) pseudo-optimal Metropolis-Hastings acceptance rates of 0.234 for random walk Metropolis-Hastings and 0.574 for MALA are established. These rates are pseudo-optimal in the sense that they are optimal in the limit of extending the target with itself infinitely many times. They are however still very much utilised in practice - by tuning the stepsize ε to achieve the desired acceptance rate. This can be done

adaptively (with diminishing adaptation) via a Robbins-Monro scheme Andrieu and Thoms (2008) or with preliminary runs. The optimal rate for Hamiltonian Monte Carlo-like algorithms depends on the number of integrator steps but is typically set to 0.651 Beskos et al. (2013) or higher.

2.4.3 Markov Chain Monte Carlo within Sequential Monte Carlo

It is sometimes possible to use a Markovian transition kernel $q_{t|0:t-1}(x_t \mid x_{0:t-1}) = q_{t|t-1}(x_t \mid x_{t-1})$ that is π_{t-1}-invariant in the proposal step of a sequential Monte Carlo algorithm, Section 2.3.

In this case the optimal backward kernel, Equation (2.9), remains intractable. However a natural sub-optimal choice Del Moral et al. (2006) is

$$\pi_{0:t-1|t}(x_{0:t-1} \mid x_t) = \pi_{0:t-2|t-1}(x_{0:t-2} \mid x_{t-1}) \frac{\pi_{t-1}(x_{t-1}) q_{t|t-1}(x_t \mid x_{t-1})}{\pi_{t-1}(x_t)}, \tag{2.28}$$

which is normalised by the π_{t-1}-invariance of $q_{t|t-1}$.

In this case, the SMC weights, Equation (2.8), take the very simple form

$$w_t = w_{t-1} \frac{\pi_t(x_t)}{\pi_{t-1}(x_t)}. \tag{2.29}$$

2.5 Approximate Bayesian Computation

Consider a Bayesian inference problem

$$\pi(x) = p(x \mid y) = \frac{p(x)p(y \mid x)}{p(y)},$$

with the additional complication that we cannot evaluate the likelihood density $p(y|x)$ as it is intractable or too costly. Assume, however, that we can relatively cheaply generate a simulated observation $y' \sim p(\cdot \mid x)$ from the likelihood for a given value of x without any trouble.

Approximate Bayesian computation (ABC) proceeds by instead targeting the extended distribution

$$\nu_\kappa(x, y') = \frac{p(x)p(y'|x)\mathbb{I}(d(y', y) < \kappa)}{p_\kappa(y)}, \tag{2.30}$$

where $d : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$ is a *distance function*, κ is a threshold parameter and $p_\kappa(y) = \int p(x)p(y'|x)\mathbb{I}(d(y', y) < \kappa)dy'dx$ is the normalising constant. Note that y is the true given data and y' is an auxiliary random variable representing simulations from the likelihood - in $\nu_\kappa(x, y')$ we suppress the dependence on y for brevity.

By focusing on the marginal distribution of $\nu_\kappa(x,y')$ in x and noting that $\kappa \to 0 \implies \mathbb{I}(d(y',y) < \kappa) \to \delta(y' \mid y)$ we can see that

$$\begin{aligned}
\kappa \to 0 \implies \nu_\kappa(x) &= \int \nu_\kappa(x,y')dy', \\
&= \int \frac{p(x)p(y'|x)\mathbb{I}(d(y',y) < \kappa)}{p_\kappa(y)}dy', \\
&\to \int \frac{p(x)p(y'|x)\delta(y' \mid y)}{p(y)}dy', \\
&= \pi(x),
\end{aligned}$$

where $p_\kappa(y) \to p(y)$ by the same argument. This gives us the ability to control the bias $\nu_\kappa(x) \neq \pi(x)$ for $\kappa > 0$ through the threshold parameter κ, although decreasing κ increases simulation costs leaving a computational trade-off.

2.5.1 ABC Importance Sampling

We can generate weighted samples from $\nu_\kappa(x,y')$ by first generating

$$x \sim q(\cdot), \qquad y' \sim p(\cdot|x),$$

and then applying importance weights with target ν_κ

$$\begin{aligned}
w &\propto \frac{p(x)p(y'|x)\mathbb{I}(d(y',y) < \kappa)}{q(x)p(y'|x)}, \\
&= \frac{p(x)}{q(x)}\mathbb{I}(d(y',y) < \kappa).
\end{aligned}$$

A common choice of importance distribution is the prior $q(x) = p(x)$. In this case, the weights become

$$w \propto \mathbb{I}(d(y',y) < \kappa).$$

But why is it called Approximate Bayesian Computation?

Bayesian computation or *Bayesian inference* refers to the process of taking expectations over the true posterior $p(x \mid y)$, most often with Monte Carlo samples. By instead taking expectations over $\nu_\kappa(x)$ we obtain *approximate Bayesian computation*.

The resulting algorithm, we term *vanilla ABC* Pritchard et al. (1999); Tavaré et al. (1997), is simultaneously an importance sampler and a rejection sampler where a sample (x,y') is accepted if $\mathbb{I}(d(y',y) < \kappa)$. A convenient practical property of the vanilla ABC algorithm is that the threshold parameter κ can be selected after sampling to achieve a desired acceptance rate - making the trade-off between bias and computational tractability particularly transparent.

For complex problems the posterior π or rather the ABC posterior ν_κ (for small κ) is going to look very different from the prior. A consequence is very low acceptance rates for vanilla ABC and therefore an inefficient algorithm. One option is to consider a more sophisticated proposal q, although as described in Section 2.1, general purpose informed proposals are difficult to construct.

Algorithm 7 Importance ABC

1: Sample $x^{(i)} \sim q$ $\quad i = 1,\dots,N$
2: Sample $y^{(i)} \sim p(\cdot \mid x^{(i)})$ $\quad i = 1,\dots,N$
3: Normalise $\hat{p}_\kappa(y) = \frac{1}{N}\sum_{i=1}^{N} \frac{p(x^{(i)})}{q(x^{(i)})}\mathbb{I}(d(y^{(i)},y) < \kappa)$
4: Weight $w^{(i)} = \frac{p(x^{(i)})}{q(x^{(i)})N\hat{p}_\kappa(y)}\mathbb{I}(d(y^{(i)},y) < \kappa)$
5: **return** $\{x^{(i)}, w^{(i)}\}_{i=1}^{N}$

2.5.2 ABC Markov Chain Monte Carlo

An alternative approach is to run a Markov chain Monte Carlo algorithm targeting ν_κ, Marjoram et al. (2003). Consider the following Markovian proposal kernel on the extended space

$$q(x',y' \mid x^{(i-1)},y^{(i-1)}) = q(x' \mid x^{(i-1)},y^{(i-1)})p(y' \mid x'),$$

Then the Metropolis-Hastings acceptance probability becomes

$$\begin{aligned}\alpha(x^{(i-1)},y^{(i-1)},x',y') &= \min\left(1, \frac{\nu_\kappa(x',y')q(x^{(i-1)},y^{(i-1)} \mid x',y')}{\nu_\kappa(x^{(i-1)},y^{(i-1)})q(x',y' \mid x^{(i-1)},y^{(i-1)})}\right), \\ &= \min\left(1, \frac{p(x')q(x^{(i-1)} \mid x',y')}{p(x^{(i-1)})q(x' \mid x^{(i-1)},y^{(i-1)})}\mathbb{I}(d(y',y) < \kappa)\right). \end{aligned} \tag{2.31}$$

As gradients are naturally out of the question, the random walk proposal $q(x' \mid x^{(i-1)},y^{(i-1)}) = \mathcal{N}(x' \mid x^{(i-1)}, \varepsilon \mathrm{D})$ from Equation (2.20) is typically used. By the symmetry of the random walk proposal, the acceptance probability then simplifies to

$$\alpha(x^{(i-1)},y^{(i-1)},x',y') = \min\left(1, \frac{p(x')}{p(x^{(i-1)})}\mathbb{I}(d(y',y) < \kappa)\right).$$

In the case that the posterior distribution is unimodal, it is intuitive to think that this Markovian approach can be much more efficient the vanilla version Marjoram et al. (2003) as the algorithm explores the posterior locally rather than starting afresh at every iteration.

The parameters ε, D and κ can be tuned either adaptively or using trial runs. Vihola and Franks (2020) recommend tuning so that $\varepsilon \mathrm{D} \approx \frac{2.38^2}{d}\Sigma$ as in Equation (2.27) where Σ is the posterior covariance (which can be approximated by the previously generated samples). This leaves κ which can be adapted using a Robbins-Monro schedule to achieve a desired acceptance rate, Vihola and Franks (2020) argue that due to the extension of the state-space the optimal acceptance rate must be lower than 0.234 and make a recommendation of 0.1.

Algorithm 8 ABC-MCMC

Sample $x^{(0)}, y^{(0)} \sim K_0$
for $i = 1, \ldots, N$ **do**
  Propose $x' \sim q(\cdot \mid x^{(i-1)}, y^{(i-1)})$
  Propose $y' \sim p(\cdot \mid x')$
  Set α according to Equation (2.31)
  Sample $u \sim \mathbf{U}(\cdot \mid 0, 1)$
  if $u < \alpha$ **then**
    $x^{(i)}, y^{(i)} = x', y'$
  else
    $x^{(i)}, y^{(i)} = x^{(i-1)}, y^{(i-1)}$
return $\{x^{(i)}\}_{i=1}^N$

2.5.3 ABC Sequential Monte Carlo

We can also utilise a sequential Monte Carlo approach for ABC Sisson et al. (2007), Del Moral et al. (2012), where we define the sequence of intermediate marginal distributions as

$$\nu_{\kappa_t}(x_t, y_t) = \frac{p(x_t)p(y_t|x_t)\mathbb{I}(d(y_t, y) < \kappa_t)}{p_{\kappa_t}(y)}, \qquad t = 0, \ldots, T,$$

for a decreasing sequence of threshold parameters

$$\kappa_0 > \kappa_1 > \cdots > \kappa_T.$$

Assuming we have a Markovian transition $q_{t|0:t-1}(x_t, y_t|x_{0:t-1}, y_{0:t-1}) = q_{t|t-1}(x_t, y_t|x_{t-1}, y_{t-1})$ then the sequential importance weights become

$$w_t \propto w_{t-1} \frac{\nu_{\kappa_t}(x_t, y_t)\nu_{t-1|t}(x_{t-1}, y_{t-1}|x_t, y_t)}{\nu_{\kappa_{t-1}}(x_{t-1}, y_{t-1})q_{t|t-1}(x_t, y_t|x_{t-1}, y_{t-1})}.$$

A choice of transition kernel that induces tractable weights is

$$q_{t|t-1}(x_t, y_t|x_{t-1}, y_{t-1}) = q_{t|t-1}(x_t|x_{t-1}, y_{t-1})p(y_t|x_t).$$

Similarly setting the auxiliary target $\nu_{t-1|t}(x_{t-1}, y_{t-1}|x_t, y_t) = \nu_{t-1|t}(x_{t-1}|x_t, y_t)p(y_{t-1}|x_{t-1})$ then gives

$$w_t \propto w_{t-1} \frac{p(x_t)\nu_{t-1|t}(x_{t-1}|x_t, y_t)}{p(x_{t-1})q_{t|t-1}(x_t|x_{t-1}, y_{t-1})} \mathbb{I}(d(y_t, y) < \kappa_t).$$

Given a transition kernel $q_{t|t-1}(x_t|x_{t-1}, y_{t-1})$ we still need to define a sensible backward kernel $\nu_{t-1|t}(x_{t-1}|x_t, y_t)$ which can be difficult for complex proposals.

Alternatively we could choose $q_{t|t-1}(x_t, y_t|x_{t-1}, y_{t-1})$ to be a $\nu_{\kappa_{t-1}}$-invariant MCMC kernel from 2.5.2. In this case, we can choose the backward kernel as in 2.4.3 and obtain tractable weights

$$w_t \propto w_{t-1} \mathbb{I}(d(y_t, y) < \kappa_t).$$

2.5.4 Distance Functions

The distance function d is an important factor in both the bias induced by extending the space and the efficiency of the aforementioned simulation algorithms.

As the data y can be very high dimensional, most distance functions first summarise the data

$$d(y', y) = d^*(S(y'), S(y)),$$

for a summary function $S : \mathcal{Y} \to \mathcal{Y}^*$ where $\mathcal{Y}^*$ is typically of much lower dimension. The new distance function d^* is most often simply set to the Euclidean distance or a preconditioned version thereof. Choosing summary functions that sufficiently reduce the dimension but retain as much of the information described by the data is a particularly scenario specific task Fearnhead and Prangle (2012).

Algorithm 9 ABC-SMC

1: Sample $x_0^{(i)} \sim q_0$ $\quad i = 1\ldots,N$

2: Sample $y_0^{(i)} \sim p(\cdot \mid x_0^{(i)})$ $\quad i = 1\ldots,N$

3: Normalise $\hat{Z}_0 = \frac{1}{N}\sum_{i=1}^{N} \frac{p(x_0^{(i)})}{q_0(x_0^{(i)})} \mathbb{I}(d(y_0^{(i)}, y) < \kappa_0)$

4: Weight $w_0^{(i)} = \frac{p(x_0^{(i)})}{q_0(x_0^{(i)}) N \hat{Z}_0} \mathbb{I}(d(y_0^{(i)}, y) < \kappa_0)$ $\quad i = 1\ldots,N$

5: **for** $t = 1, \ldots, T$ **do**

6: **if** `resampling criterion` **then**

7: $$\left\{\left(\tilde{x}_{t-1}^{(i)}, \tilde{y}_{t-1}^{(i)}\right), \tilde{w}_{t-1}^{(i)} = \tfrac{1}{N}\right\}_{i=1}^{N} = \text{Resample}\left(\left\{\left(x_{t-1}^{(i)}, y_{t-1}^{(i)}\right), w_{t-1}^{(i)}\right\}_{i=1}^{N}\right)$$

8: **else**

9: $$\left\{\left(\tilde{x}_{t-1}^{(i)}, \tilde{y}_{t-1}^{(i)}\right), \tilde{w}_{t-1}^{(i)}\right\}_{i=1}^{N} = \left\{\left(x_{t-1}^{(i)}, y_{t-1}^{(i)}\right), w_{t-1}^{(i)}\right\}_{i=1}^{N}$$

10: Propose from $\nu_{\kappa_{t-1}}$-invariant $q_{t|t-1}$

$$x_t^{(i)}, y_t^{(i)} \sim q_{t|t-1}(x_t, y_t \mid \tilde{x}_{t-1}^{(i)}, \tilde{y}_{t-1}^{(i)}) \qquad i = 1\ldots,N$$

11: Normalise

$$\hat{Z}_{t|t-1} = \sum_{i=1}^{N} \tilde{w}_{t-1}^{(i)} \mathbb{I}(d(y_t^{(i)}, y) < \kappa_t)$$

12: Reweight

$$w_t^{(i)} = \tilde{w}_{t-1}^{(i)} \mathbb{I}(d(y_t^{(i)}, y) < \kappa_t) / \hat{Z}_{t|t-1} \qquad i = 1\ldots,N$$

13: **return** $\left\{\left\{x_t^{(i)}, w_t^{(i)}\right\}_{i=1}^{N}\right\}_{t=0}^{T}$

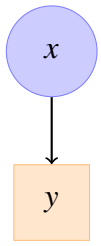


Fig. 2.1 Static Bayesian model.

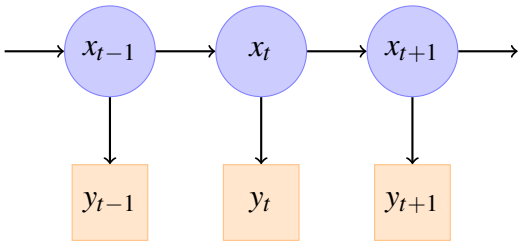


Fig. 2.2 State-space model.

2.6 State-Space Models

But why are they called State-Space Models?

In the language of Rudolf E. Kálmán, the variables $x_{0:T}$ are termed *state variables* and $y_{0:T}$ *observation variables* - thus inferring $x_{0:T}$ given $y_{0:T}$ becomes *state-space modelling*.

The Bayesian paradigm, Figure 2.1, provides a very flexible framework for describing data generating processes. When the data arrives sequentially it can be particularly useful to impose a stricter conditional independence structure where data is generated from an underlying Markov process Figure 2.2. Models with this conditional independence structure are commonly referred to as *state-space models* (or *hidden Markov models*).

The full posterior or *smoothing distribution* of a state-space model takes the form

$$p(x_{0:T}|y_{0:T}) = \frac{p(x_{0:T}, y_{0:T})}{p(y_{0:T})}, \tag{2.32}$$

$$p(x_{0:T}, y_{0:T}) = p(x_0|y_0)\prod_{t=1}^{T} p(x_t|x_{t-1})p(y_t|x_t). \tag{2.33}$$

In online settings, it is often only the *filtering marginal* $p(x_T|y_{0:T})$ that is of interest.

But why is it called filtering and smoothing?

In signal processing, the term *filtering* refers to inferring x_T by denoising the noisy observations $y_{0:T}$. *Smoothing* refers to inference over a full trajectory $x_{0:T}$ where it is common for the transition density $p(x_t|x_{t-1})$ to favour nearby consecutive variables - thus encouraging a smooth trajectory.

2.6.1 Linear Gaussian State-Space Models

A useful subset of state-space models is when all distributions are assumed to be linear and Gaussian, that is

$$p(x_0) = \mathbf{N}(x_0 \mid \mu_{0|0}, \Sigma_{0|0}), \tag{2.34a}$$
$$p(x_t \mid x_{t-1}) = \mathbf{N}(x_t \mid \mathrm{F}_t x_{t-1}, \mathrm{Q}_t), \qquad t = 1, \ldots, T, \tag{2.34b}$$
$$p(y_t \mid x_t) = \mathbf{N}(y_t \mid \mathrm{H}_t x_t, \mathrm{R}_t), \qquad t = 0, \ldots, T. \tag{2.34c}$$

As all intermediate distributions are Gaussian, so is the full joint distribution $p(x_{0:T}, y_{0:T})$. Any conditionals (including $p(x_{0:T}|y_{0:T})$ and $p(x_T|y_{0:T})$) are therefore also Gaussian and can be obtained by applying standard properties of Gaussian distributions to $p(x_{0:T}, y_{0:T})$, however this approach will come at cost $O(T^3)$.

Instead, we can utilise the conditional independence of the state-space model Figure 2.2 to calculate the filtering marginals recursively. Assume we know the previous filtering marginal

$$p(x_{T-1}|y_{0:T-1}) = \mathbf{N}(x_{T-1} \mid \mu_{T-1|T-1}, \Sigma_{T-1|T-1})$$

then the *predictive marginal* is

$$\begin{aligned} p(x_T|y_{0:T-1}) &= \int p(x_{T-1}, x_T \mid y_{0:T-1}) dx_{T-1}, \\ &= \int p(x_T \mid x_{T-1}) p(x_{T-1} \mid y_{0:T-1}) dx_{T-1}, \\ &= \mathbf{N}(x_{T-1} \mid \mu_{T|T-1}, \Sigma_{T|T-1}), \end{aligned}$$

where $\mu_{T|T-1} = \mathrm{F}_T \mu_{T-1|T-1}$ and $\Sigma_{T|T-1} = \mathrm{F}_T \Sigma_{T-1|T-1} \mathrm{F}_T^{\mathrm{T}} + \mathrm{Q}_T$ can be found by applying the laws of total expectation and covariance.

We can then use the predictive marginal and the observation y_T to calculate the updated filtering marginal

$$\begin{aligned} p(x_T|y_{0:T}) &= \frac{p(y_T|x_T) p(x_T|y_{0:T-1})}{p(y_T|y_{0:T-1})}, \\ &= \mathbf{N}(x_T \mid \mu_{T|T}, \Sigma_{T|T}), \end{aligned} \tag{2.35}$$

where

$$\begin{aligned} \mu_{T|T} &= \mu_{T|T-1} + \mathrm{K}_{T|T}(y_T - \mathrm{H}_T \mu_{T|T-1}), \\ \Sigma_{T|T} &= \Sigma_{T|T-1} - \mathrm{K}_{T|T} \mathrm{H}_T \Sigma_{T|T-1}, \\ \mathrm{K}_{T|T} &= \Sigma_{T|T-1} \mathrm{H}_T^{\mathrm{T}} \left(\mathrm{H}_T \Sigma_{T|T-1} \mathrm{H}_T^{\mathrm{T}} + \mathrm{R}_T \right)^{-1}, \end{aligned}$$

can be derived by completing the square in (2.35) or considering the joint distribution of $(x_T, y_T) \mid y_{0:T-1}$ and conditioning on y_T. The resulting recursion is the celebrated *Kalman filter*, Kalman (1960).

Algorithm 10 Kalman Filter

1: Given $\mu_{T-1|T-1}$ and $\Sigma_{T-1|T-1}$

2: Calculate predictive marginal statistics

$$\begin{aligned}\mu_{T|T-1} &= \mathrm{F}_T \mu_{T-1|T-1} \\ \Sigma_{T|T-1} &= \mathrm{F}_T \Sigma_{T-1|T-1} \mathrm{F}_T^{\mathrm{T}} + \mathrm{Q}_T\end{aligned}$$

3: Calculate filtering marginal statistics

$$\begin{aligned}\mathrm{K}_{T|T} &= \Sigma_{T|T-1} \mathrm{H}_T^{\mathrm{T}} \left(\mathrm{H}_T \Sigma_{T|T-1} \mathrm{H}_T^{\mathrm{T}} + \mathrm{R}_T\right)^{-1} \\ \mu_{T|T} &= \mu_{T|T-1} + \mathrm{K}_{T|T}(y_T - \mathrm{H}_T \mu_{T|T-1}) \\ \Sigma_{T|T} &= \Sigma_{T|T-1} - \mathrm{K}_{T|T} \mathrm{H}_T \Sigma_{T|T-1}\end{aligned}$$

4: **return** $\mu_{T|T}$ and $\Sigma_{T|T}$

In offline settings we can store all the filtering marginals from a Kalman filter and run a backward pass to obtain the *smoothing marginals* $p(x_t|y_{0:T})$. Assuming we have $p(x_{t+1}|y_{0:T}) = \mathbf{N}(x_{t+1} \mid \mu_{t+1|T}, \Sigma_{t+1|T})$

$$\begin{aligned}p(x_t|y_{0:T}) &= \int p(x_t, x_{t+1}|y_{0:T}) dx_{t+1}, \\ &= \int p(x_t|x_{t+1}, y_{0:T}) p(x_{t+1}|y_{0:T}) dx_{t+1}, \\ &= \int p(x_t|x_{t+1}, y_{0:t}) p(x_{t+1}|y_{0:T}) dx_{t+1}, \\ &= \int \frac{p(x_{t+1}|x_t) p(x_t|y_{0:t})}{p(x_{t+1}|y_{0:t})} p(x_{t+1}|y_{0:T}) dx_{t+1}. \\ &= \mathbf{N}(x_{t+1} \mid \mu_{t|T}, \Sigma_{t|T}).\end{aligned} \tag{2.36}$$

where

$$\begin{aligned}\mu_{t|T} &= \mu_{t|t} + \mathrm{K}_{t|T}(\mu_{t+1|T} - \mathrm{F}_{t+1} \mu_{t|t}), \\ \Sigma_{t|T} &= \Sigma_{t|t} + \mathrm{K}_{t|T} \left(\Sigma_{t+1|T} - \mathrm{F}_{t+1} \Sigma_{t|t} \mathrm{F}_{t+1}^{\mathrm{T}} - \mathrm{Q}_{t+1}\right) \mathrm{K}_{t|T}^{\mathrm{T}}, \\ \mathrm{K}_{t|T} &= \Sigma_{t|t} \mathrm{F}_t^{\mathrm{T}} \left(\mathrm{F}_t \Sigma_{t|t} \mathrm{F}_t^{\mathrm{T}} + \mathrm{Q}_{t+1}\right)^{-1}.\end{aligned}$$

A full pass of either the Kalman filter or Kalman smoother has cost $O(T)$.

Algorithm 11 Kalman Smoother

1: Given filtering statistics $\left\{\mu_{t|t}, \Sigma_{t|t}\right\}_{t=0}^{T}$ from Kalman filter, Algorithm 10.
2: **for** $t = T-1, \ldots, 0$ **do**
3: Calculate smoothing marginal statistics

$$\begin{aligned}
\mathrm{K}_{t|T} &= \Sigma_{t|t}\mathrm{F}_t^{\mathrm{T}}\left(\mathrm{F}_t\Sigma_{t|t}\mathrm{F}_t^{\mathrm{T}} + \mathrm{Q}_{t+1}\right)^{-1} \\
\mu_{t|T} &= \mu_{t|t} + \mathrm{K}_{t|T}(\mu_{t+1|T} - \mathrm{F}_{t+1}\mu_{t|t}) \\
\Sigma_{t|T} &= \Sigma_{t|t} + \mathrm{K}_{t|T}\left(\Sigma_{t+1|T} - \mathrm{F}_{t+1}\Sigma_{t|t}\mathrm{F}_{t+1}^{\mathrm{T}} - \mathrm{Q}_{t+1}\right)\mathrm{K}_{t|T}^{\mathrm{T}}
\end{aligned}$$

4: **return** $\left\{\mu_{t|T}, \Sigma_{t|T}\right\}_{t=0}^{T}$

2.6.2 Particle Filtering

The sequential Monte Carlo techniques we described in full generality in 2.3 were first introduced in the context of state-space models Gordon et al. (1993).

Sequential Monte Carlo when applied to online inference in state-space models is commonly referred to as *particle filtering*. In this context, we assume we have weighted particles $\{x_{0:T-1}^{(i)}, w_{T-1}^{(i)}\}_{i=1}^{N}$ approximating $p(x_{0:T-1}|y_{0:T-1})$. Upon receiving a new observation y_T, we extend trajectories by sampling from some proposal distribution $q_{T|0:T-1}$

$$x_T^{(i)} \sim q_{T|0:T-1}(x_T \mid x_{0:T-1}^{(i)}, y_T), \qquad i = 1, \ldots, N.$$

Note that we can additionally condition the proposal on the previous observations $y_{0:T-1}$, however the conditional independence of the state-space model, Figure 2.2, implies that $x_{0:T-1}$ are sufficient for $y_{0:T-1}$.

With target distribution $p(x_{0:T}|y_{0:T})$ the importance weights in Equation (2.8) become

$$\begin{aligned}
w_T &= w_{T-1}\frac{p(x_{0:T} \mid y_{0:T})}{p(x_{0:T-1} \mid y_{0:T-1})q_{T|0:T-1}(x_T \mid x_{0:T-1}, y_T)}, \\
&= w_{T-1}\frac{p(x_T \mid x_{T-1})p(y_T|x_T)}{q_{T|0:T-1}(x_T \mid x_{0:T-1}, y_T)}\frac{p(y_{0:T-1})}{p(y_{0:T})}, \\
&\propto w_{T-1}\frac{p(x_T \mid x_{T-1})p(y_T|x_T)}{q_{T|0:T-1}(x_T \mid x_{0:T-1}, y_T)},
\end{aligned} \tag{2.37}$$

where we have substituted in the smoothing distribution Equation (2.32). The weights now take a simple, sequential form and we can also use the self-normalising described in 2.1.1 to approximate $\frac{p(y_{0:T})}{p(y_{0:T-1})} = p(y_T|y_{0:T-1})$.

The most common choice of proposal distribution is

$$q^{\mathrm{BF}}_{T|0:T-1}(x_T \mid x_{0:T-1}, y_T) = p(x_T \mid x_{T-1}), \tag{2.38a}$$

$$w^{\mathrm{BF}}_T \propto w_{T-1} p(y_T|x_T), \tag{2.38b}$$

and the resulting algorithm is termed the *bootstrap filter*. An important property of the bootstrap filter is that the induced weights are independent of $p(x_T|x_{T-1})$ and can therefore provide exact inference in state-space models where the transition density $p(x_T|x_{T-1})$ is intractable (i.e. can be simulated from but not evaluated).

It was shown in Doucet et al. (2000) that the (locally) optimal proposal is

$$q^{\mathrm{opt}}_{T|0:T-1}(x_T \mid x_{0:T-1}, y_T) = \frac{p(x_T \mid x_{T-1}) p(y_T|x_T)}{p(y_T|x_{T-1})}, \tag{2.39a}$$

$$w^{\mathrm{opt}}_T \propto w_{T-1} p(y_T|x_{T-1}). \tag{2.39b}$$

Where the optimality is in the sense of minimising the variance of the weights under the sampling distribution

$$q^{\mathrm{opt}}_{T|0:T-1} = \underset{q_{T|0:T-1}}{\arg\min}\ \mathrm{Cov}_{q_{T|0:T-1}}[w_t].$$

Indeed, we can see that induced optimal weights $w^{\mathrm{opt}}_T \propto w_{T-1} p(y_T|x_{T-1})$ are independent of x_T and therefore $\mathrm{Cov}_{q^{\mathrm{opt}}_{T|0:T-1}}[w_t] = 0$.

However, it is only in a few special cases that both sampling from the optimal proposal and evaluating the induced weights are tractable.

As can be seen from the weights in Equation (2.37), the particle filter is doing importance sampling targeting the smoothing distribution $p(x_{0:T}|y_{0:T})$. Without resampling, the particle filter is entirely non-interacting and is therefore pure importance sampling targeting $p(x_{0:T}|y_{0:T})$ with importance distribution

$$q_{0:T}(x_{0:T}) = q_0(x_0) \prod_{t=1}^{T} q_{t|0:t-1}(x_t|x_{0:t-1}).$$

The dimension of the state-space increases with each iteration and therefore without resampling the divergence between target distribution and importance distribution increases until we are left with only a single particle duplicated N times, Del Moral and Doucet (2003). Thus resampling is essential to ensure a diverse particle approximation to the filtering marginals $p(x_T|y_{0:T})$.

Algorithm 12 Particle Filter

Given $\{x^{(i)}_{0:T-1}, w^{(i)}_{T-1}\}_{i=1}^N$

if `resampling criterion` **then**

$$\left\{\tilde{x}^{(i)}_{0:T-1}, \tilde{w}^{(i)}_{T-1} = \tfrac{1}{N}\right\}_{i=1}^N = \text{Resample}\left(\left\{x^{(i)}_{0:T-1}, w^{(i)}_{T-1}\right\}_{i=1}^N\right)$$

else

$$\left\{\tilde{x}^{(i)}_{0:T-1}, \tilde{w}^{(i)}_{T-1}\right\}_{i=1}^N = \left\{x^{(i)}_{0:T-1}, w^{(i)}_{T-1}\right\}_{i=1}^N$$

Propose

$$x^{(i)}_T \sim q_{T|0:T-1}(x_T \mid \tilde{x}^{(i)}_{0:T-1}, y_T) \qquad i = 1\ldots,N$$

Normalise

$$\hat{Z}_{T|0:T-1} = \sum_{i=1}^N \tilde{w}^{(i)}_{T-1} \frac{p(x^{(i)}_T \mid \tilde{x}^{(i)}_{T-1}) p(y_T \mid x^{(i)}_T)}{q_{T|0:T-1}(x^{(i)}_T \mid \tilde{x}^{(i)}_{0:T-1}, y_T)}$$

Reweight

$$w^{(i)}_T = \tilde{w}^{(i)}_{T-1} \frac{p(x^{(i)}_T \mid \tilde{x}^{(i)}_{T-1}) p(y_T \mid x^{(i)}_T)}{q_{T|0:T-1}(x^{(i)}_T \mid \tilde{x}^{(i)}_{0:T-1}, y_T) \hat{Z}_{T|0:T-1}} \qquad i = 1,\ldots,N$$

Append $x^{(i)}_{0:T} = \tilde{x}^{(i)}_{0:T-1} \cup \tilde{x}^{(i)}_T$

return $\{x^{(i)}_{0:T}, w^{(i)}_T\}_{i=1}^N$

2.6.3 Particle Smoothing

In each iteration of a particle filter only the latest coordinate x_T is generated, and then resampling steps may be performed to ensure only particles with reasonable weights are proposed at the next iteration. Consequently, the diversity (number of unique particles) of the approximations to all previous coordinates $x_{0:T-1}$ can only decrease at each iteration. As such, the resampling operation preserves the diversity of particle approximations to the filtering marginals $p(x_T|y_{0:T})$ but at the cost of degeneracy to the smoothing approximations at early coordinates. Indeed for T large enough the particle filter approximation to the first smoothing marginal $p(x_0|y_{0:T})$ converges to a single particle duplicated N times - this is known as *path degeneracy*. In online settings we may only be interested in the filtering marginals and can accept path degeneracy.

In offline settings where we are interested in expectations over the full smoothing distribution $p(x_{0:T}|y_{0:T})$ we have more work to do. Fortunately, we can recycle the output of a particle filter and, similarly to the Kalman smoother, utilise the conditional independence structure of state-space models to run a backward pass and obtain a diverse particle approximation. Although unlike the Kalman smoother, we can use the particles to approximate the joint smoothing distribution $p(x_{0:T}|y_{0:T})$ rather than only smoothing marginals $\{p(x_t|y_{0:T})\}_{t=0}^{T}$. These particle smoothing techniques can be viewed as recycling the filtering marginals $\{p(x_t|y_{0:t})\}_{t=0}^{T}$ in order to approximate the joint smoothing distribution $p(x_{0:T}|y_{0:T})$.

Assume we have filtering particles which we represent with a tilde

$$\left\{\tilde{x}_{t-1}^{(i)}, \tilde{w}_{t-1}^{(i)}\right\}_{i=1}^{N} \text{ approximating } p(x_{t-1}|y_{0:t-1}),$$

and unweighted smoothing particles

$$\left\{x_t^{(j)}\right\}_{j=1}^{N} \text{ approximating } p(x_t|y_{0:T}).$$

But desire unweighted joint smoothing particles

$$\left\{\left(x_{t-1}^{(j)}, x_t^{(j)}\right)\right\}_{j=1}^{N} \text{ approximating } p(x_{t-1}, x_t|y_{0:T}).$$

We can write the intermediate joint smoothing distribution as

$$p(x_{t-1}, x_t|y_{0:T}) = p(x_t|y_{0:T})p(x_{t-1}|x_t, y_{0:t-1}).$$

Our x_t particles already have the correct marginal distribution so we only need to update x_{t-1} according to $p(x_{t-1}|x_t, y_{0:t-1})$. By Bayes' theorem and the conditional independence structure

of state-space models

$$p(x_{t-1}|x_t, y_{0:t-1}) = \frac{p(x_t|x_{t-1})p(x_{t-1}|y_{0:t-1})}{p(x_t|y_{0:t-1})}, \tag{2.40}$$

and we have empirical approximations

$$\begin{aligned} p(x_{t-1}|y_{0:t-1}) &\approx \sum_{i=1}^{N} \tilde{w}_{t-1}^{(i)} \delta(x_{t-1} \mid \tilde{x}_{t-1}^{(i)}), \\ p(x_t|y_{0:t-1}) &= \int p(x_t|x_{t-1})p(x_{t-1}|y_{0:t-1})dx_{t-1} \\ &\approx \sum_{i=1}^{N} \tilde{w}_{t-1}^{(i)} p(x_t|\tilde{x}_{t-1}^{(i)}). \end{aligned}$$

By plugging these into the decomposition Equation (2.40) we get

$$\begin{aligned} p(x_{t-1}|x_t^{(j)}, y_{0:t-1}) &\approx \frac{\sum_{i=1}^{N} \tilde{w}_{t-1}^{(i)} p(x_t^{(j)}|\tilde{x}_{t-1}^{(i)}) \delta(x_{t-1} \mid \tilde{x}_{t-1}^{(i)})}{\sum_{i=1}^{N} \tilde{w}_{t-1}^{(i)} p(x_t^{(j)}|\tilde{x}_{t-1}^{(i)})}, \\ &= \sum_{i=1}^{N} w_{t-1}^{(i\leftarrow j)} \delta(x_{t-1} \mid \tilde{x}_{t-1}^{(i)}). \end{aligned} \tag{2.41}$$

where $w_{t-1}^{(i\leftarrow j)} \propto \tilde{w}_{t-1}^{(i)} p(x_t^{(j)}|\tilde{x}_{t-1}^{(i)})$ are normalised across i.

By iteratively backwards sampling from this empirical conditional distribution

$$x_{t-1}^{(j)} \sim \sum_{i=1}^{N} w_t^{(i\leftarrow j)} \delta(x_{t-1} \mid \tilde{x}_{t-1}^{(i)}) \qquad t = T, T-1, \ldots 1,$$

with $x_{t-1}^{(j)} = \tilde{x}_{t-1}^{(j)}$, we obtain the *forward filtering-backward simulation* algorithm Godsill et al. (2004). The backward simulation converts the weighted filtering marginal particles into an unweighted particle approximation $\{x_{0:T}^{(j)}\}_{j=1}^{N}$ that is asymptotically unbiased for the joint smoothing distribution $p(x_{0:T}|y_{0:T})$.

The cost of backward simulation is $O(N^2)$ as all the weights $w_t^{(i\leftarrow j)}$ for $i, j = 1, \ldots, N$ need to be computed in Equation (2.41). It was noted in Douc et al. (2011) that by using a rejection sampling technique, we can reduce the cost of backward simulation to $O(N)$ when we can bound the transition density

$$p(x_t|x_{t-1}) < C \qquad \forall x_{t-1}, x_t. \tag{2.42}$$

At a single iteration of backward simulation we have $\left\{\tilde{x}_{t-1}^{(i)}, \tilde{w}_{t-1}^{(i)}\right\}_{i=1}^{N} \sim p(x_{t-1}|y_{0:t-1})$ and $\left\{x_t^{(j)}\right\}_{j=1}^{N} \sim p(x_t|y_{0:T})$. Then the pure rejection sampler proceeds by repeating the following procedure until a sample is accepted for each smoothing trajectory j

1. Sample $x'_{t-1} \sim \sum_{i=1}^{N} \tilde{w}_{t-1}^{(i)} \delta(x_{t-1} \mid \tilde{x}_{t-1}^{(i)})$.
2. With probability $\alpha = p(x_t^{(j)}|x'_{t-1})/C < 1$
   - Set $x_{t-1}^{(j)} = x'_{t-1}$,
   - Otherwise go to 1.

Asymptotically the resulting sample is from the conditional importance distribution

$$q^*_{t-1|t}(x_{t-1}|x_t) = \frac{p(x_t|x_{t-1})}{C} p(x_{t-1}|y_{0:t-1})/Z^*_{t-1|t}(x_t),$$

where the normalisation constant is

$$\begin{aligned} Z^*_{t-1|t}(x_t) &= \int \frac{p(x_t|x_{t-1})}{C} p(x_{t-1}|y_{0:t-1}) dx_{t-1} \\ &= \frac{p(x_t|y_{0:t-1})}{C}. \end{aligned}$$

and therefore we have

$$q^*_{t-1|t}(x_{t-1}|x_t) = p(x_{t-1}|x_t, y_{0:t-1}),$$

as desired.

Therefore, by rejection sampling we no longer need to compute normalising constant of the backwards sampling weights and have reduced the cost to $O(NR_E)$ where R_E is the expected number of attempts proposed to the rejection sampler, which for stays constant for very large N Douc et al. (2011).

Note that if we instead replace the pure rejection sampling with a rejection control sampler 2.2, i.e. we were to use a parameter C that was not a strict bound and augment our smoothing particles with importance weights, the induced weights would include a $p(x_t|y_{0:t-1})$ term which cannot typically be evaluated.

For finite N the rejection sampling technique is not always faster than the N^2 version. A pragmatic approach is described in Taghavi et al. (2013) where up to R proposals are sent to the rejection sampler for each particle. Any particles remaining without an acceptance are sent to the N^2 version. This hybrid backward simulation is described in Algorithm 13.

Algorithm 13 Backward Simulation

Given filtering marginal particles $\left\{ \left\{ \tilde{x}_t^{(i)}, \tilde{w}_t^{(i)} \right\}_{i=1}^N \right\}_{t=0}^T$

$\left\{ x_T^{(i)} \right\}_{i=1}^N = \text{Resample}\left(\left\{ \tilde{x}_T^{(i)}, \tilde{w}_T^{(i)} \right\}_{i=1}^N \right)$

for $t = T-1, \ldots, 0$ **do**

  for $i = 1, \ldots, N$ **do**

    for $r = 1, \ldots, R$ **do** — R = max number of rejections

      Sample $c^* \sim \text{Categorical}\left(\left\{ \tilde{w}_t^{(i)} \right\}_{i=1}^N \right)$

      Sample $u \sim \mathbf{U}(\cdot \mid 0, 1)$

      if $u < p(x_{t+1}^{(i)} \mid \tilde{x}_t^{(c^*)})/C$ **then** — Bound $C > p(x_{t+1} \mid x_t)\ \forall x_t, x_{t+1}$

        Accept c^* and **break** to 10

    if a sample c^* was accepted **then**

      Set $x_t^{(i)} = \tilde{x}_t^{(c^*)}$

    else

      Calculate interacting weights and normalise in k

$$w_t^{(k \leftarrow i)} \propto p(x_{t+1}^{(i)} \mid \tilde{x}_t^{(k)}) \tilde{w}_t^{(k)} \qquad k = 1, \ldots, N$$

      Sample $c_i \sim \text{Categorical}\left(\left\{ w_t^{(k \leftarrow i)} \right\}_{k=1}^N \right)$

      Set $x_t^{(i)} = \tilde{x}_t^{(c_i)}$

return $\{x_{0:T}^{(i)}\}_{i=1}^N$

Chapter 3

Online Particle Smoothing

In this chapter, we introduce a novel approach to online smoothing in state-space models based on a fixed-lag approximation. Unlike classical fixed-lag techniques, our method approximates the joint posterior distribution rather than just the marginals. It does this without suffering path degeneracy as the length of the state-space model increases.

Recall that a state-space model is fully defined by the following distributions for the hidden $\{x_t\}_{t=0}^{\infty}$ and observed process $\{y_t\}_{t=0}^{\infty}$

$$\begin{aligned} &p(x_0), && \\ &p(x_t \mid x_{t-1}), && t = 1,2,3,\dots \\ &p(y_t \mid x_t), && t = 0,1,2,\dots \end{aligned}$$

The statistical inference goal of *smoothing* is the task of approximating the full joint posterior or smoothing distribution

$$p(x_{0:T} \mid y_{0:T}) \propto p(x_0 \mid y_0) \prod_{t=1}^{T} p(x_t \mid x_{t-1}) p(y_t \mid x_t), \tag{3.1}$$

where $x_{0:T} = (x_0, \dots, x_T)$ are latent states to be inferred and $y_{0:T} = (y_0, \dots, y_T)$ are given observations.

For *online smoothing*, we have the additional requirement of being able to quickly and accurately update an approximation of $p(x_{0:T-1} \mid y_{0:T-1})$ to approximate $p(x_{0:T} \mid y_{0:T})$ in light of receiving a new observation y_T.

As described in Chapter 2, particle smoothers approximate $p(x_{0:T} \mid y_{0:T})$ with a collection of weighted particles

$$p(x_{0:T} \mid y_{0:T}) \approx \sum_{i=1}^{N} w_T^{(i)} \delta \left(x_{0:T} \mid x_{0:T}^{(i)} \right),$$

as opposed to particle filters which are (normally) only asked to approximate the filtering marginal $p(x_T \mid y_{0:T})$. Existing online particle smoothing approximations either degenerate as the length of the state-space model, T, increases or only target the smoothing marginals $\{p(x_t \mid y_{0:T})\}_{t=0}^{T}$ rather than the joint smoothing distribution. A particle approximation to the joint smoothing distribution is significantly more useful as mathematical expectations can be calculated over a variety of functions defined over full trajectories, thus providing the user with complete flexibility.

In summary, our motivation is to develop an algorithm that simultaneously satisfies the following requirements

- **Joint Smoothing**: The algorithm efficiently approximates the joint smoothing distribution $p(x_{0:T} \mid y_{0:T})$ rather than only marginals.
- **Online**: Our approximation can be quickly updated on receipt of new observations.
- **Non-degenerate**: The algorithm avoids the path degeneracy of classical particle filters.

The contribution of this chapter is to introduce two general techniques for generating samples on-the-fly that approximate $p(x_{0:T} \mid y_{0:T})$ via a fixed-lag approximation.

The rest of the chapter is structured as follows. In Section 3.1, we recap and discuss related online and offline algorithms for particle smoothing, as summarised in Table 3.1 alongside those introduced in this chapter. Section 3.2 describes how to combine blocked samples in a way that is invariant for a fixed-lag joint smoothing distribution before Section 3.3 introduces two efficient online methods for generating these blocked samples. Section 3.4 investigates numerically the performance of the introduced online particle smoothers and their sensitivity to key parameters. Finally, in Section 3.5 we conclude and discuss some potential extensions.

3.1 Particle Smoothing

Particle smoothing refers to particle approximations of the smoothing distribution $p(x_{0:T} \mid y_{0:T})$. This can be done directly by treating the task as a static Bayesian inference problem Figure 2.1, however by leveraging the conditional independence structure of state-space models Figure 2.2, sequential Monte Carlo techniques can be used to effectively reduce the dimension.

3.1.1 Path Degeneracy

A classical particle filter runs a single forward pass, updating particles at every observation. Each update consists of three steps: an optional *resample* step, a *propagation* step and a *weighting* step, Algorithm 12.

The resampling operation converts a weighted sample into an unweighted sample that likely contains duplicates $\left\{x^{(i)}, w^{(i)}\right\}_{i=1}^{N} \rightarrow \left\{x^{(i)}, \frac{1}{N}\right\}_{i=1}^{N}$. The most common resampling method is multinomial sampling (2.11) which we repeat here

$$\begin{aligned} \text{Sample} \quad & a^{(i)} \sim \text{Categorical}\left(\left\{w^{(i)}\right\}_{i=1}^{N}\right), && i = 1, \ldots, N, \\ \text{Set} \quad & x^{(i)} \leftarrow x^{(a^{(i)})} \quad w^{(i)} \leftarrow \frac{1}{N}, && i = 1, \ldots, N. \end{aligned}$$

Due to the optional nature of the resampling step, adaptive schemes are desirable, with a popular choice to be to only resample if the effective sample size $1/\sum_{i=1}^{N} w_T^{(i)\,2}$ falls below some threshold.

Although most commonly used only to approximate the filtering marginals $p(x_T \mid y_{0:T})$, the particle filter as described in Algorithm 12 does provide an asymptotically unbiased approximation to the full smoothing distribution $p(x_{0:T} \mid y_{0:T})$.

The reason that a classical particle filter is almost never used to approximate the smoothing distribution is due to *path degeneracy*. Path degeneracy occurs in state-space models with large T. For repeated particle filter updates, the early coordinates of particles (e.g. $x_0^{(i)}$) will only be altered in the resampling step. Resampling can only decrease particle diversity (the number of distinct particles) and therefore as T increases the particle approximation of $p(x_0 \mid y_{0:T})$ will eventually collapse to just a single particle repeated N times.

In addition to criteria such as the effective sample size that ensure resampling is only applied when necessary, a variety of techniques have been developed to mitigate the effects of path degeneracy. These include adaptively increasing the number of particles Elvira et al. (2017) as well as more sophisticated resampling schemes Douc and Cappe (2005); Li et al. (2015) which aim to only resample particles with negligible weights. These approaches can improve performance over multinomial resampling however still suffer the collapse in the approximation of $p(x_0 \mid y_{0:T})$ for T sufficiently large.

The merits of resampling are well known, in particular resampling is vital in ensuring a diverse particle approximation to the filtering marginals - the particle filter's primary task. Indeed, a particle filter without resampling is simply importance sampling on an a space whose dimension increases with every new observation. As a result, the divergence between importance and target distribution increases and the number of particles must increase exponentially Chatterjee and Diaconis (2018) - the addition of resampling means the number of particles can be kept constant and a stable approximation to the filtering marginals is maintained.

3.1.2 Marginal Fixed-Lag

An alternative approach in mitigating the path degeneracy induced by repeated resampling is to simply stop resampling the early coordinates of particles, as proposed in Kitagawa and Sato (2001). That is replace the resample step (Algorithm 12, line 3) for $T > L$ with the scheme described in Algorithm 14. The justification for freezing $x^{(i)}_{0:T-L-1}$ is that after a certain lag L

Algorithm 14 Marginal Fixed-lag Resampling (for $T > L$)

1: Fix $\{x^{(i)}_{0:T-L-1}\}_{i=1}^N$
2: Resample only recent coordinates

$$\left\{x^{(i)}_{T-L:T}, w^{(i)}_T\right\}_{i=1}^N \rightarrow \left\{x^{(i)}_{T-L:T}, \frac{1}{N}\right\}_{i=1}^N$$

Stitch arbitrarily $x^{(i)}_{0:T} = (x^{(i)}_{0:T-L-1}, x^{(i)}_{T-L:T})$.

the smoothing distribution of early coordinates become (approximately) independent of new observations

$$p(x_{0:t} \mid y_{0:T}) \approx p(x_{0:t} \mid y_{0:\min(t+L,T)}). \tag{3.2}$$

The major issue with the fixed-lag resampling scheme described in Algorithm 14 is that early and recent coordinates of particles are arbitrarily stitched together and therefore only provide a particle approximation to the *fixed-lag marginal* smoothing distribution:

$$p^L_{\text{marg}}(x_{0:T} \mid y_{0:T}) = \left[\prod_{t=0}^{T-L-1} p(x_t \mid y_{0:t+L})\right] p(x_{T-L:T} \mid x_{T-L-1}, y_{T-L:T}). \tag{3.3}$$

As such we lose information about the joint distribution of early coordinates $x_{0:T-L-1}$.

A more useful particle approximation targets the *fixed-lag joint* smoothing distribution

$$p^L(x_{0:T} \mid y_{0:T}) = p(x_0 \mid y_{0:L}) \left[\prod_{t=1}^{T-L-1} p(x_t \mid x_{t-1}, y_{t:t+L})\right] p(x_{T-L:T} \mid x_{T-L-1}, y_{T-L:T}), \tag{3.4}$$

which permits expectations over full trajectories $x_{0:T}$.

3.1.3 Offline Smoothing

The idea of running a forward particle filter pass that stores the marginal filter approximations $\{x^{(i)}_t, w^{(i)}_t\}_{i=1}^N \sim p(x_t \mid y_{0:T})$ before recycling them in a backward pass was first introduced as *forward filtering-backward smoothing* in Hürzeler and Künsch (1998) and then further

developed in Doucet et al. (2000). This method runs a full backward pass that updates the weights based on the backward decomposition

$$p(x_{t-1} \mid x_t, y_{0:t-1}) = \frac{p(x_t \mid x_{t-1})p(x_{t-1} \mid y_{0:t-1})}{p(x_t \mid y_{0:t-1})}. \tag{3.5}$$

In this method, the particles remain arbitrarily stitched and their weights updated according to Equation (3.5) in order to approximate the marginal smoothing distributions $p(x_t \mid y_{0:T})$.

As mentioned in Section 2.6.3, *forward filtering-backward simulation* (FFBSi) Godsill et al. (2004) similarly runs a full backward pass based on the same decomposition, but samples a new ancestor from the particle cloud (according to Equation (3.5)) rather than only updating the weights and thus targets the joint smoothing distribution. Additionally, Douc et al. (2011) showed that rejection sampling can significantly reduce the complexity of this pass in the case that $p(x_t \mid x_{t-1})$ has a tractable upper bound.

Both of these algorithms require a full backward pass in light of every new observation and therefore are not suitable for online smoothing. In addition, the popular PMCMC approach Andrieu et al. (2010) is iterative in nature and thus not extensible to online smoothing.

3.1.4 Online Smoothing

It was noted in Del Moral et al. (2010) that the forward filtering-backward smoothing algorithm can be implemented in a single forward pass in the case of *additive functionals*, i.e. expectations of the form

$$\mathbb{E}_{p(x_{0:T}|y_{0:T})}[h(x_{0:T})] \quad \text{where} \quad h(x_{0:T}) = \sum_{t=0}^{T-1} \tilde{h}_t(x_t, x_{t+1}).$$

The PaRIS algorithm Olsson and Westerborn (2017) combines this technique with rejection sampling to implement an efficient and cheap version of forward filtering-backward simulation in a single forward pass for additive functionals.

Although it permits the implementation of these online algorithms, the requirement of additive functionals is very restrictive. Additionally, the forward only technique is *function specific*, i.e. it directly updates an approximation to the expected value of the additive functional. Our approach induces a controllable bias through the fixed-lag approximation but can approximate any expectation $\mathbb{E}_{p(x_{0:T}|y_{0:T})}[f(x_{0:T})]$ over the joint smoothing distribution and is therefore significantly more general than the marginal or additive functional approaches.

Block sampling Doucet et al. (2006) is an online method that targets the full smoothing distribution (3.1). Every time a new observation is received, the block sampling scheme discards the most recent coordinates (within lag L) and re-proposes from an enlarged proposal distribution based on many more recent observations. Although this scheme does make use of

	Joint Smoothing	Online	Path Degeneracy	Fixed-lag Approx.	Complexity
Particle Filter Gordon et al. (1993)	✓	✓	✓		N
Marginal Fixed-lag Kitagawa and Sato (2001)		✓	For large L	✓	N
Forward Filtering-Backward Smoothing Doucet et al. (2000)					N^2
Forward Filtering-Backward Simulation Godsill et al. (2004)	✓				N^2 N with RS†
Forward Filtering-Backward Smoothing for Additive Functionals Del Moral et al. (2010)		✓			N^2
PaRIS Olsson and Westerborn (2017)		✓			N with RS†
Block Sampling Doucet et al. (2006)	✓	✓	✓		LN
Online Particle Smoother (Algorithm 16)	✓	✓	For large L	✓	N^2 N with RS†
Online Particle Smoother with Backward Simulation (Algorithm 17)	✓	✓		✓	LN^2 LN with RS†

Table 3.1 Comparison of particle smoothing algorithms, for number of particles N and fixed-lag parameter L.
†For these algorithms the rejection sampling technique of Douc et al. (2011) can be applied to obtain linear complexity when a bound for the transition density, Equation (2.42), is available.

a fixed-lag parameter, the weights and resampling still act on the full smoothing distribution Equation (3.1). By moving a larger proportion of the trajectories, block sampling can (when combined with adaptive resampling schemes) mitigate but not avoid path degeneracy as resampling still takes place over full trajectories. Our proposed method builds on the block sampling approach by proposing blocks with a single coordinate overlap and then resampling in a way that targets the fixed-lag joint (3.4).

3.2 Fixed-lag Particle Stitching

In this section, we show how the backward simulation derivation in Section 2.6.3 can be adapted into a fixed-lag forward simulation technique.

For the forward implementation we initially assume we can generate block samples directly from $p(x_{T-L-1:T} \mid y_{0:T})$ even though this is not immediately possible for non-trivial state-space models. In Section 3.3 we then describe two efficient methods for block sampling by recycling previously generated particles.

3.2.1 Fixed-lag Forward Simulation - Intractable

In the setting of fixed-lag forward simulation, we have

$$\left\{x_{t-1}^{(i)}\right\}_{i=1}^{N} \text{ approximating } p(x_{t-1} \mid y_{0:T-1}),$$
$$\left\{\tilde{x}_t^{(j)}, \tilde{w}_t^{(j)}\right\}_{j=1}^{N} \text{ approximating } p(x_t \mid y_{0:T}),$$

where $t = T - L$ and the fixed-lag approximation implies $p(x_{t-1} \mid y_{0:T-1}) \approx p(x_{t-1} \mid y_{0:T})$.

We desire unweighted samples from the joint

$$\left\{\left(x_{t-1}^{(i)}, x_t^{(i)}\right)\right\}_{i=1}^{N} \text{ approximating } p^L(x_{t-1}, x_t \mid y_{0:T}),$$

where

$$p^L(x_{t-1}, x_t \mid y_{0:T}) = p(x_{t-1} \mid y_{0:T-1}) p(x_t \mid x_{t-1}, y_{t:T}).$$

Both $\left\{x_{t-1}^{(i)}\right\}_{i=1}^{N}$ and $\left\{\tilde{x}_t^{(j)}, \tilde{w}_t^{(j)}\right\}_{j=1}^{N}$ have the correct marginals but we propose freezing $\left\{x_{t-1}^{(i)}\right\}_{i=1}^{N}$ in order to avoid path degeneracy as T increases. Thus, obtaining the desired joint samples amounts to sampling

$$x_t^{(i)} \sim p(x_t \mid x_{t-1}^{(i)}, y_{t:T}) \qquad \text{for } i = 1, \ldots, N.$$

We cannot sample this directly, so we try the decomposition

$$\begin{aligned} p(x_t \mid x_{t-1}^{(i)}, y_{t:T}) &= p(x_t \mid x_{t-1}^{(i)}, y_{0:T}) \\ &= \frac{p(y_{t:T} \mid x_t) p(x_t \mid x_{t-1}^{(i)})}{p(y_{t:T} \mid x_{t-1}^{(i)})}, \\ &= \frac{p(y_{t:T} \mid x_t) p(x_t \mid x_{t-1}^{(i)})}{p(y_{t:T} \mid x_{t-1}^{(i)}) p(x_t \mid y_{0:T})} p(x_t \mid y_{0:T}). \end{aligned}$$

where in the last line we have multiplied and divided by $p(x_t \mid y_{0:T})$ - the sampling distribution of $\left\{\tilde{x}_t^{(j)}, \tilde{w}_t^{(j)}\right\}_{j=1}^{N}$.

Application of Bayes' theorem to $p(x_t \mid y_{0:T})$ gives us

$$p(x_t \mid x_{t-1}^{(i)}, y_{t:T}) = \frac{p(y_{t:T} \mid y_{0:t-1}) p(x_t \mid x_{t-1}^{(i)})}{p(y_{t:T} \mid x_{t-1}^{(i)}) p(x_t \mid y_{0:t-1})} p(x_t \mid y_{0:T}),$$

which is where we stop as the density $p(x_t \mid y_{0:t-1})$ is not tractable (note that we could replace $p(x_t \mid y_{0:t-1})$ with an empirical approximation using marginal filtering particles but the resulting algorithm would have a prohibitive $O(N^3)$ complexity).

3.2.2 Fixed-lag Forward Simulation - Tractable

Now consider the setting of fixed-lag forward simulation but with a single coordinate overlap

$$\begin{aligned}\left\{x_{t-1}^{(i)}\right\}_{i=1}^{N} &\text{ approximating } p(x_{t-1} \mid y_{0:T-1}),\\ \left\{(\tilde{x}_{t-1}^{(j)}, \tilde{x}_t^{(j)}), \tilde{w}_t^{(j)}\right\}_{j=1}^{N} &\text{ approximating } p(\breve{x}_{t-1}, x_t \mid y_{0:T}),\end{aligned}$$

where $\breve{x}_{t-1}$ is the coordinate overlap to be discarded. We now desire samples from an extended joint distribution

$$\left\{(x_{t-1}^{(i)}, x_t^{(i)}, \breve{x}_{t-1}^{(i)})\right\}_{i=1}^{N} \text{ approximating } \pi(x_{t-1}, x_t, \breve{x}_{t-1} \mid y_{0:T}),$$

where

$$\begin{aligned}\pi(x_{t-1}, x_t, \breve{x}_{t-1} \mid y_{0:T}) &= p^L(x_{t-1}, x_t \mid y_{0:T})\lambda(\breve{x}_{t-1} \mid x_{t-1}, x_t \mid y_{0:T}),\\ &= p(x_{t-1} \mid y_{0:T-1})p(x_t \mid x_{t-1}, y_{t:T})\lambda(\breve{x}_{t-1} \mid x_{t-1}, x_t, y_{0:T}).\end{aligned}$$

We eventually discard $\breve{x}_{t-1}$ and so are free to choose λ. Again we freeze $\left\{x_{t-1}^{(i)}\right\}_{i=1}^{N}$ and note that they have the correct marginals $\pi(x_{t-1} \mid y_{0:T}) = p(x_{t-1} \mid y_{0:T-1})$. Thus generating samples from the extended joint amounts to sampling

$$\left(x_t^{(i)}, \breve{x}_{t-1}^{(i)}\right) \sim \pi(x_t, \breve{x}_{t-1} \mid x_{t-1}^{(i)}, y_{0:T}) \qquad \text{for } i = 1, \ldots, N.$$

Where

$$\begin{aligned}\pi(x_t, \breve{x}_{t-1} \mid x_{t-1}^{(i)}, y_{0:T}) &= p(x_t \mid x_{t-1}^{(i)}, y_{t:T})\lambda(\breve{x}_{t-1} \mid x_{t-1}^{(i)}, x_t, y_{0:T}),\\ &= \frac{p(y_{t:T} \mid x_t)p(x_t \mid x_{t-1}^{(i)})}{p(y_{t:T} \mid x_{t-1}^{(i)})}\lambda(\breve{x}_{t-1} \mid x_{t-1}^{(i)}, x_t, y_{0:T}).\end{aligned}$$

In order to make an empirical approximation we multiply and divide by $p(\breve{x}_{t-1}, x_t \mid y_{0:T}) = p(\breve{x}_{t-1} \mid y_{0:T}) p(x_t \mid \breve{x}_{t-1}, y_{t:T})$ - the sampling distribution of $\{(\tilde{x}_{t-1}^{(j)}, \tilde{x}_t^{(j)}), \tilde{w}_t^{(j)}\}_{j=1}^N$

$$\pi(x_t, \breve{x}_{t-1} \mid x_{t-1}^{(i)}, y_{0:T}) = \frac{p(y_{t:T} \mid x_t) p(x_t \mid x_{t-1}^{(i)})}{p(y_{t:T} \mid x_{t-1}^{(i)}) p(\breve{x}_{t-1}, x_t \mid y_{0:T})} \lambda(\breve{x}_{t-1} \mid x_{t-1}^{(i)}, x_t, y_{0:T}) p(\breve{x}_{t-1}, x_t \mid y_{0:T}).$$

Bayes' theorem on $p(\breve{x}_{t-1}, x_t \mid y_{0:T})$ gives

$$\begin{aligned}\pi(x_t, \breve{x}_{t-1} \mid x_{t-1}^{(i)}, y_{0:T}) &= \frac{p(x_t \mid x_{t-1}^{(i)}) p(y_{t:T} \mid y_{0:t-1})}{p(x_t \mid \breve{x}_{t-1}) p(y_{t:T} \mid x_{t-1}^{(i)}) p(\breve{x}_{t-1} \mid y_{0:t-1})} \lambda(\breve{x}_{t-1} \mid x_{t-1}^{(i)}, x_t, y_{0:T}) p(\breve{x}_{t-1}, x_t \mid y_{0:T}).\end{aligned}$$

We now observe that the choice of $\lambda(\breve{x}_{t-1} \mid x_{t-1}^{(i)}, x_t, y_{0:T}) = p(\breve{x}_{t-1} \mid y_{0:t-1})$ will make all terms involving $(\breve{x}_{t-1}, x_t)$ tractable.

$$\pi(x_t, \breve{x}_{t-1} \mid x_{t-1}^{(i)}, y_{0:T}) = \frac{p(x_t \mid x_{t-1}^{(i)}) p(y_{t:T} \mid y_{0:t-1})}{p(x_t \mid \breve{x}_{t-1}) p(y_{t:T} \mid x_{t-1}^{(i)})} p(\breve{x}_{t-1}, x_t \mid y_{0:T}).$$

Then using the empirical approximations

$$\begin{aligned}p(\breve{x}_{t-1}, x_t \mid y_{0:T}) &\approx \sum_{j=1}^N \tilde{w}_t^{(j)} \delta\left(\breve{x}_{t-1}, x_t \mid \tilde{x}_{t-1}^{(j)}, \tilde{x}_t^{(j)}\right), \\ \frac{p(y_{t:T} \mid x_{t-1}^{(i)})}{p(y_{t:T} \mid y_{0:t-1})} &= \int \frac{p(x_t \mid x_{t-1}^{(i)})}{p(x_t \mid \breve{x}_{t-1})} p(\breve{x}_{t-1}, x_t \mid y_{0:T}) d\breve{x}_{t-1} dx_t, \\ &\approx \sum_{j=1}^N w_t^{(j)} \frac{p(\tilde{x}_t^{(j)} \mid x_{t-1}^{(i)})}{p(\tilde{x}_t^{(j)} \mid \tilde{x}_{t-1}^{(j)})}.\end{aligned}$$

Finally giving us an empirical approximation to the extended conditional

$$\pi(x_t, \breve{x}_{t-1} \mid x_{t-1}^{(i)}, y_{0:T}) = \frac{\sum_{j=1}^N \tilde{w}_t^{(j)} \frac{p(\tilde{x}_t^{(j)} \mid x_{t-1}^{(i)})}{p(\tilde{x}_t^{(j)} \mid \tilde{x}_{t-1}^{(j)})} \delta\left(x_t, \breve{x}_{t-1} \mid \tilde{x}_t^{(j)}, \tilde{x}_{t-1}^{(j)}\right)}{\sum_{j=1}^N \tilde{w}_t^{(j)} \frac{p(\tilde{x}_t^{(j)} \mid x_{t-1}^{(i)})}{p(\tilde{x}_t^{(j)} \mid \tilde{x}_{t-1}^{(j)})}} \tag{3.6}$$

$$= \sum_{j=1}^N w_t^{(i \to j)} \delta\left(x_t, \breve{x}_{t-1} \mid \tilde{x}_t^{(j)}, \tilde{x}_{t-1}^{(j)}\right), \tag{3.7}$$

where

$$w_t^{(i\to j)} \propto \tilde{w}_t^{(j)} \frac{p(\tilde{x}_t^{(j)} \mid x_{t-1}^{(i)})}{p(\tilde{x}_t^{(j)} \mid \tilde{x}_{t-1}^{(j)})}$$

are normalised across j. Again, we can now sample from this approximation directly, discarding the sampled $\breve{x}_{t-1}$. This leaves samples (x_{t-1}, x_t) from the desired joint $p^L(x_{t-1}, x_t \mid y_{0:T})$. Repeating this for each i results in an $O(N^2)$ algorithm that is asymptotically unbiased for the fixed-lag joint distribution.

We have described above an algorithm that stitches together particles $p(x_{T-L-1} \mid y_{0:T-1})$ with those from $p(x_{T-L-1:T-L} \mid y_{0:T})$. By the conditional independence structure of state-space models this is equivalent to stitching together blocks from $p(x_{0:T-L-1} \mid y_{0:T-1})$ with those from $p(x_{T-L-1:T} \mid y_{0:T})$ - assuming we can sample from $p(x_{T-L-1:T} \mid y_{0:T})$.

3.2.3 Rejection Sampling

Sampling from (3.7) can be done directly at a computational complexity of $O(N^2)$. However, when a bound for the transition density is available Equation (2.42)

$$p(x_t|x_{t-1}) < C \qquad \forall x_{t-1}, x_t,$$

we can utilise the rejection sampling approach of Douc et al. (2011) to avoid calculating all N^2 normalisation constants and bring the computational complexity down to $O(N)$. In practice, we can apply the early-stopping regime of Taghavi et al. (2013) where, for each particle independently, the rejection sampler is halted after $R < N$ proposals and if no acceptance has occured the direct scheme is applied, thus setting $R = 0$ recovers the direct scheme. This hybrid stitching algorithm is detailed in Algorithm 15.

3.3 Sampling from $p(x_{T-L-1:T} \mid y_{0:T})$

We now describe two methods for sampling the coordinates $\tilde{x}_{T-L-1:T}$ in a way that is asymptotically unbiased for $p(x_{T-L-1:T} \mid y_{0:T})$, and can therefore be plugged into the aforementioned fixed-lag particle stitching procedure.

3.3.1 Particle Filter

Recall the online setting where we have unweighted particles $\{x_{0:T-1}^{(i)}\}_{i=1}^N$ approximating $p^L(x_{0:T-1}|y_{0:T-1})$ and receive a new observation y_T.

Algorithm 15 Fixed-lag Particle Stitching

1: Given

$$\left\{x_{0:T-L-1}^{(i)}\right\}_{i=1}^{N} \text{ approximating } p^{L}(x_{0:T-L-1} \mid y_{0:T-1}),$$
$$\left\{\tilde{x}_{T-L-1:T}^{(j)}, \tilde{w}_{T}^{(j)}\right\}_{j=1}^{N} \text{ approximating } p(x_{T-L-1:T} \mid y_{0:T}),$$

2: Calculate the non-interacting stitching weights and normalise in j

$$\hat{w}_{T}^{(j)} \propto \frac{1}{p(\tilde{x}_{T-L}^{(j)} \mid \tilde{x}_{T-L-1}^{(j)})} \tilde{w}_{T}^{(j)} \qquad j = 1, \ldots, N.$$

3: **for** $i = 1, \ldots, N$ **do**
4: **for** $r = 1, \ldots, R$ **do** R = max number of rejections
5: Sample $c^* \sim \text{Categorical}\left(\left\{\hat{w}_{T}^{(j)}\right\}_{j=1}^{N}\right)$
6: Sample $u \sim \mathbf{U}(\cdot \mid 0, 1)$
7: **if** $u < p(\tilde{x}_{T-L}^{(c^*)} \mid x_{T-L-1}^{(i)})/C$ **then** Bound $C > p(x_t \mid x_{t-1})\ \forall x_{t-1}, x_t$
8: Accept c^* and **break** to 9
9: **if** a sample c^* was accepted **then**
10: Set $x_{T-L:T}^{(i)} = \tilde{x}_{T-L:T}^{(c^*)}$
11: **else**
12: Calculate the stitching weights and normalise in j

$$w_{T}^{(i \to j)} \propto \frac{p(\tilde{x}_{T-L}^{(j)} \mid x_{T-L-1}^{(i)})}{p(\tilde{x}_{T-L}^{(j)} \mid \tilde{x}_{T-L-1}^{(j)})} \tilde{w}_{T}^{(j)} \qquad j = 1, \ldots, N.$$

13: Sample $c_i \sim \text{Categorical}\left(\left\{w_{T}^{(i \to j)}\right\}_{j=1}^{N}\right)$
14: Set $x_{T-L:T}^{(i)} = \tilde{x}_{T-L:T}^{(c_i)}$
15: **return** $\left\{x_{0:T}^{(i)}\right\}_{i=1}^{N}$ approximating $p^{L}(x_{0:T} \mid y_{0:T})$.

Our first method is based on the fact that the particle approximation provided by a classical particle filter is asymptotically unbiased for the full joint smoothing distribution, Equation (3.1). Although this approximation deteriorates due to path degeneracy it may still be sufficient for sampling the later coordinates $x_{T-L-1:T}$.

Thus, we propose applying the classical particle filter proposal and weighting steps to $\{x_{0:T-1}^{(i)}\}_{i=1}^N$, generating weighted particles $\{x_{0:T}^{(i)}, \tilde{w}_T^{(i)}\}_{i=1}^N$, before splitting the trajectories

$$\left\{x_{0:T}^{(i)}, \tilde{w}_T^{(i)}\right\}_{i=1}^N \rightarrow \left\{x_{0:T-L-1}^{(i)}\right\}_{i=1}^N \text{ and } \left\{\tilde{x}_{T-L-1:T}^{(i)}, \tilde{w}_T^{(i)}\right\}_{i=1}^N,$$

where $\tilde{x}_{T-L-1:T}^{(i)} = x_{T-L-1:T}^{(i)}$. Under the fixed-lag approximation, $x_{0:T-L-1}$ is conditionally independent of y_T and therefore the new weights need not apply to these earlier coordinates. Whereas the $\{\tilde{x}_{T-L-1:T}^{(i)}, \tilde{w}_T^{(i)}\}_{i=1}^N$ are asymptotically unbiased for the desired sampling distribution $p(x_{T-L-1:T}|y_{0:T})$ and can therefore be plugged into the stitching procedure, Algorithm 15. The coordinates x_{T-L-1} are duplicated to provide the overlap required for stitching.

Algorithm 16 Online Particle Smoother (for $T > L$)

1: Given $\left\{x_{0:T-1}^{(i)}\right\}_{i=1}^N$ approximating $p^L(x_{0:T-1}|y_{0:T-1})$
2: Fix $\left\{x_{0:T-L-1}^{(i)}\right\}_{i=1}^N$
3: Execute particle filter propagate and reweight steps, Algorithm 12: lines 5:9

$$\text{Generate } \left\{\tilde{x}_{T-L-1:T}^{(j)}, \tilde{w}_T^{(j)}\right\}_{j=1}^N \text{ from } \left\{x_{T-L-1:T-1}^{(j)}\right\}_{j=1}^N \text{ and } y_T,$$

forming a weighted sample approximating $p(x_{T-L-1:T}|y_{0:T})$.
4: Stitch together

$$\left\{x_{0:T-L-1}^{(i)}\right\}_{i=1}^N \text{ and } \left\{\tilde{x}_{T-L-1:T}^{(j)}, \tilde{w}_T^{(j)}\right\}_{j=1}^N \rightarrow \left\{x_{0:T}^{(i)}\right\}_{i=1}^N.$$

using Algorithm 15.
5: **return** $\left\{x_{0:T}^{(i)}\right\}_{i=1}^N$ approximating $p^L(x_{0:T}|y_{0:T})$

The algorithm is described in Algorithm 16, and ends up being a relatively simple modification to a classical particle filter where the resampling step is compulsory and altered to include the stitching probabilities in the weights, Equation (3.7).

When the transition bound, Equation (2.42), is available the complexity of the update remains $O(N)$ or $O(N^2)$ when the bound is unavailable.

3.3.2 Partial Backward Simulation

If the lag parameter L is chosen to be too large, the above mechanism will still suffer from path degeneracy in the same way a particle filter does or indeed the online marginal smoother Kitagawa and Sato (2001). To remedy this we propose a partial run of the forward filtering-backward simulation Godsill et al. (2004), Douc et al. (2011) (Algorithm 13) at each time step to rejuvenate the trajectories $x_{T-L-1:T}$. This technique is considered in Clapp and Godsill (1999) for generating samples from $p(x_{T-L:T}|y_{0:T})$ without the subsequent stitching.

The backward simulation has the additional requirement of storing the marginal approximations $\left\{\tilde{x}_t^{(k)}, \tilde{w}_t^{(k)}\right\}_{k=1}^N$ for $t = T-L-1, \ldots, T$ from the particle filter, but permits the use of adaptive resampling and typically avoids path degeneracy even for large L.

The resulting algorithm, Algorithm 17, has a complexity of $O(LN)$ per update if the transition bound Equation (2.42) is available otherwise $O(LN^2)$.

We note that both techniques to sample from $p(x_{T-L-1:T}|y_{0:T})$ utilise the output of a particle filter. Indeed, they are both also applicable to alternative filtering techniques such as auxiliary particle filters Pitt and Shephard (1999) and the backward simulation technique to filters that discard historic trajectories such as the marginal particle filter Klaas et al. (2005).

3.4 Numerical Experiments

In this section we examine the performance of the aforementioned online smoothers in a simple one-dimensional linear Gaussian state-space model

$$p(x_0) = \mathbf{N}(x_0 \mid 0, 1), \tag{3.8a}$$
$$p(x_t \mid x_{t-1}) = \mathbf{N}(x_t \mid x_{t-1}, 1), \qquad t = 1, \ldots, T, \tag{3.8b}$$
$$p(y_t \mid x_t) = \mathbf{N}(y_t \mid x_t, 1), \qquad t = 0, \ldots, T. \tag{3.8c}$$

In this example we can find the full joint distribution $p(x_{0:T}, y_{0:T})$ and smoothing distribution $p(x_{0:T} \mid y_{0:T})$ analytically by first noting that we can rewrite

$$x_t = \sum_{i=1}^{t} \eta_i,$$
$$y_t = \sum_{i=1}^{t} \eta_i + \zeta_t,$$

Algorithm 17 Online Particle Smoother with Backward Simulation (for $T > L$)

1: Given

$$\left\{x_{0:T-1}^{(i)}\right\}_{i=1}^{N} \text{ approximating } p^{L}(x_{0:T-1} \mid y_{0:T-1})$$
$$\left\{\tilde{x}_{t}^{(k)}, \tilde{w}_{t}^{(k)}\right\}_{j=1}^{N} \text{ approximating } p(x_t \mid y_{0:t}) \qquad \text{for } t = T-L-1, \ldots, T-1$$

2: Fix $\left\{x_{0:T-L-1}^{(i)}\right\}_{i=1}^{N}$

3: Execute particle filter update, Algorithm 12, to generate the new marginal filtering sample

$$\text{Generate } \left\{\tilde{x}_{T}^{(k)}, \tilde{w}_{T}^{(k)}\right\}_{k=1}^{N} \text{ from } \left\{\tilde{x}_{T-1}^{(k)}, \tilde{w}_{T-1}^{(k)}\right\}_{k=1}^{N} \text{ and } y_T.$$

4: Run partial backward simulation, Algorithm 13, on the weighted filtering samples

$$\left\{\tilde{x}_{t}^{(k)}, \tilde{w}_{t}^{(k)}\right\}_{k=1}^{N} \text{ for } t = T, \ldots, T-L-1 \rightarrow \left\{\tilde{x}_{T-L-1:T}^{(j)}\right\}_{j=1}^{N}.$$

forming an unweighted sample approximating $p(x_{T-L-1:T} | y_{0:T})$.

5: Stitch together

$$\left\{x_{0:T-L-1}^{(i)}\right\}_{i=1}^{N} \text{ and } \left\{\tilde{x}_{T-L-1:T}^{(j)}, \tfrac{1}{N}\right\}_{j=1}^{N} \rightarrow \left\{x_{0:T}^{(i)}\right\}_{i=1}^{N}.$$

using Algorithm 15.

6: **return**

$$\left\{x_{0:T}^{(i)}\right\}_{i=1}^{N} \text{ approximating } p^{L}(x_{0:T} \mid y_{0:T})$$
$$\left\{\tilde{x}_{t}^{(k)}, \tilde{w}_{t}^{(k)}\right\}_{j=1}^{N} \text{ approximating } p(x_t \mid y_{0:t}) \qquad \text{for } t = T-L, \ldots, T$$

where each η_i and ζ_i are independent draws from $\mathbf{N}(\cdot \mid 0, 1)$. We can therefore write the joint distribution $p(x_{0:T}, y_{0:T})$ as

$$\begin{pmatrix} x_{0:T} \\ y_{0:T} \end{pmatrix} \sim \mathbf{N}\left(\cdot \,\middle|\, \begin{pmatrix} \underline{0}_T \\ \underline{0}_T \end{pmatrix}, \begin{pmatrix} \Sigma^x & \Sigma^{xy} \\ \Sigma^{yx} & \Sigma^y \end{pmatrix} \right)$$

where $\underline{0}_T$ is a length T vector of zeros. The covariance matrices are defined as

$$\begin{aligned}
\Sigma^x_{ij} &= \mathrm{Cov}[x_i, x_j], \\
&= \mathrm{Cov}\left[\sum_{k=1}^{i} \eta_k, \sum_{l=1}^{j} \eta_l \right], \\
&= \min(i, j), \\
\Sigma^{xy}_{ij} &= \mathrm{Cov}[x_i, y_j], \\
&= \mathrm{Cov}\left[\sum_{k=1}^{i} \eta_k, \sum_{l=1}^{j} \eta_l + \zeta_j \right], \\
&= \min(i, j), \\
\Sigma^y_{ij} &= \mathrm{Cov}[y_i, y_j], \\
&= \mathrm{Cov}\left[\sum_{k=1}^{i} \eta_k + \zeta_i, \sum_{l=1}^{j} \eta_l + \zeta_j \right], \\
&= \min(i, j) + \mathbb{I}[i = j],
\end{aligned}$$

and $\Sigma^{yx} = \Sigma^{xy\mathrm{T}}$. Then by applying the standard rule for Gaussian conditionals, we get the smoothing distribution

$$x_{0:T} \mid y_{0:T} \sim \mathbf{N}\left(\cdot \mid \Sigma^{xy}\Sigma^{y\,-1} y_{0:T}, \Sigma^x - \Sigma^{xy}\Sigma^{y\,-1}\Sigma^{xy} \right). \tag{3.9}$$

The smoothing mean and one standard deviation are plotted in Figure 3.1 alongside the true simulated process $x_{0:T}$ and observations $y_{0:T}$ all for $T = 40$.

In this example, the (locally) optimal proposal Equation (2.39) is tractable

$$\begin{aligned}
q_{T|0:T-1}(x_T \mid x_{0:T-1}, y_T) &= p(x_T \mid x_{T-1}, y_T), \\
&= \mathbf{N}\left(x_T \,\middle|\, \frac{1}{2} x_{T-1} + \frac{1}{2} y_T, \frac{1}{2} \right).
\end{aligned}$$

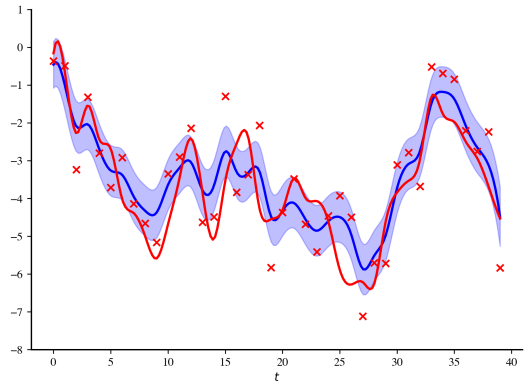


Fig. 3.1 True hidden process and observations in red. Smoothing mean $\pm$ one standard deviation in blue.

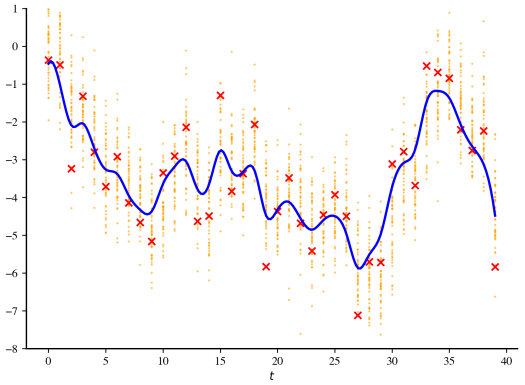


Fig. 3.2 Particle filter marginals.

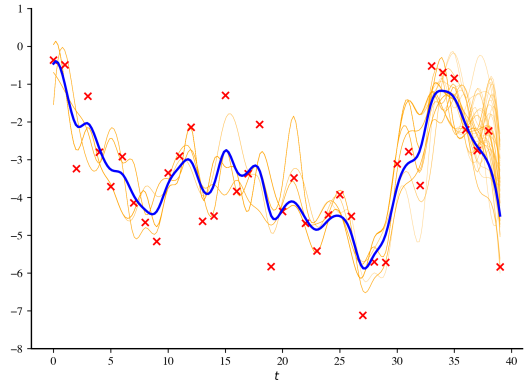


Fig. 3.3 Particle filter for smoothing.

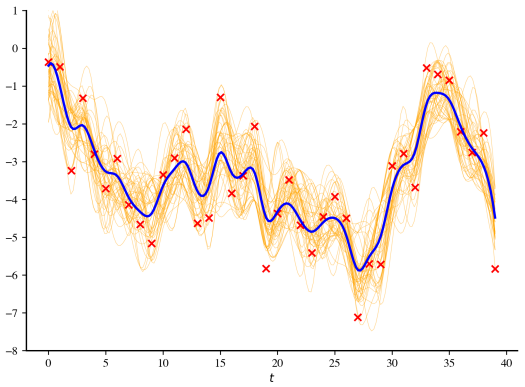


Fig. 3.4 Forward filtering-backward simulation.

In Figure 3.2 we display the particle filter approximations to the filtering marginals $p(x_t \mid y_{0:T}), t = 0, \ldots, T$, we observe that the particles suitably cover the high probability regions around the true smoothing posterior mean in blue.

In Figure 3.3 we attempt to use the full trajectories from the particle filter at time T to approximate the smoothing distribution. We can see that the particle approximation is diverse for the later coordinates but severe path degeneracy has occurred at early coordinates - due to the repeated resampling.

Figure 3.4 represents the marginal approximations from Figure 3.2 recycled with offline backward simulation (FFBSi). The result is a diverse particle approximation to the joint smoothing distribution $p(x_{0:T} \mid y_{0:T})$.

In Figures 3.5-3.10 we visualise online particle smoothing techniques that make use of a fixed-lag approximation, Equation (3.2).

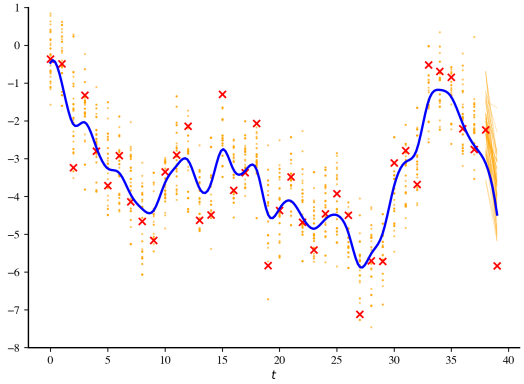


Fig. 3.5 Marginal fixed-lag, $L = 2$.

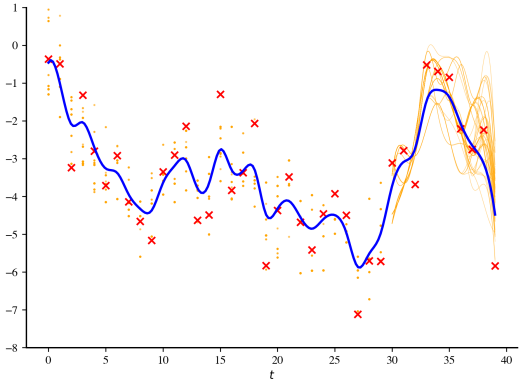


Fig. 3.6 Marginal fixed-lag, $L = 10$.

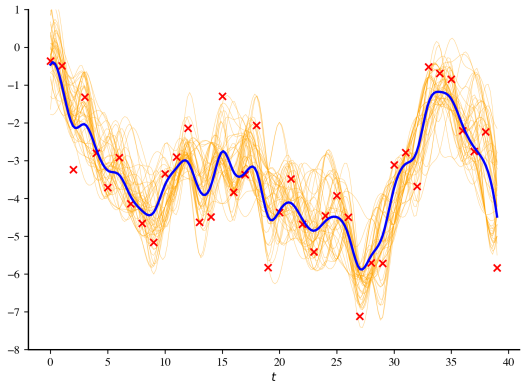


Fig. 3.7 Online particle smoother, $L = 2$.

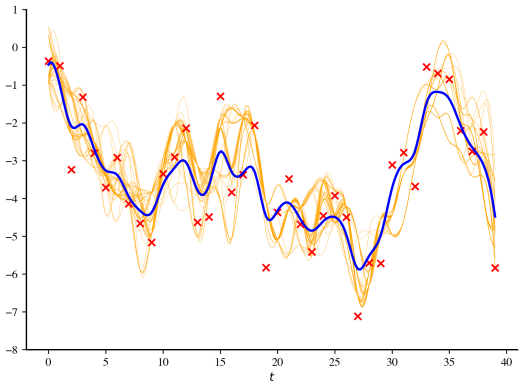


Fig. 3.8 Online particle smoother, $L = 10$.

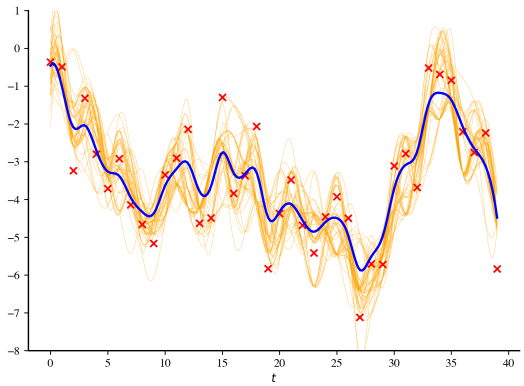


Fig. 3.9 Online particle smoother with backward simulation, $L = 2$.

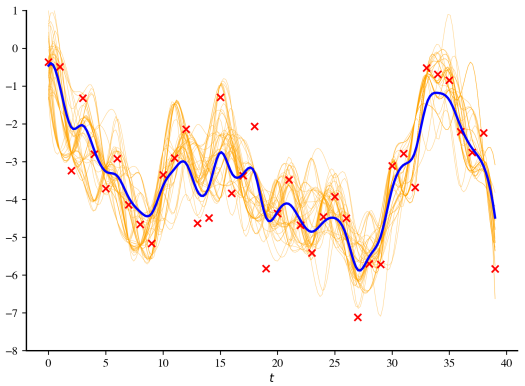


Fig. 3.10 Online particle smoother with backward simulation, $L = 10$.

As seen from Equation (3.3), the marginal fixed-lag approach of Kitagawa and Sato (2001) only approximates the marginals for early coordinates - due to arbitrary stitching. We can see that the marginals are diverse for small lag parameter Figure 3.5 but deteriorate under repeated resampling for the larger lag parameter Figure 3.6.

The same effect is observed for the online particle smoother with blocks generated using the particle filter, Figure 3.7 and Figure 3.8 - with the key difference that the full joint distribution is approximated rather than just marginals.

Adding backward simulation mitigates against effects of path degeneracy for large lags as observed by the diverse particle approximation in Figure 3.10. The fixed-lag joint distribution Equation (3.4) will look different for varying lags, the larger the lag the closer it is to the true smoothing distribution

$$L \to T \implies p^L(x_{0:T} \mid y_{0:T}) \to p(x_{0:T} \mid y_{0:T}).$$

However, the sensitivity of $p^L(x_{0:T} \mid y_{0:T})$ to the choice of L is difficult to observe from the visualised particle approximations in Figure 3.9 and Figure 3.10.

In order to investigate the choice of the lag parameter L we compare the particle approximations from the aforementioned algorithms that approximate the joint smoothing distribution with the true smoothing distribution, Equation (3.9). We do so by first extracting the empirical mean and covariance from the particle approximations

$$\begin{aligned}
\bar{x}_{0:T} &= \frac{1}{N}\sum_{i=1}^{N} x_{0:T}^{(i)}, \\
\bar{\mathrm{C}}_{0:T} &= \frac{1}{N-1}\sum_{i=1}^{N} (x_{0:T}^{(i)} - \bar{x}_{0:T})(x_{0:T}^{(i)} - \bar{x}_{0:T})^{\mathrm{T}}.
\end{aligned}$$

We then evaluate the KL-divergence between the inferred Gaussian distribution $\hat{p}(x_{0:T}|y_{0:T}) = \mathbf{N}(x_{0:T} \mid \bar{x}_{0:T}, \bar{\mathrm{C}}_{0:T})$ and the true Gaussian smoothing distribution $p(x_{0:T} \mid y_{0:T})$ described in Equation (3.9). Here we use the identity for the KL-divergence between two Gaussian distributions

$$\begin{aligned}
&\mathrm{KL}\left(\mathbf{N}(x \mid \mu_0, \Sigma_0) || \mathbf{N}(x \mid \mu_1, \Sigma_1)\right) \\
&\quad = \frac{1}{2}\left(\mathrm{trace}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^{\mathrm{T}}\Sigma_1^{-1}(\mu_1 - \mu_0) - d + \log\det\Sigma_1 - \log\det\Sigma_0\right), \quad (3.10)
\end{aligned}$$

where d is the dimension of x, in our example $d = T = 40$.

All algorithms in Figure 3.11 were ran with $N = 10000$. We observe that the resampled final particle filter trajectories perform very poorly - due to path degeneracy in the early coordinates.

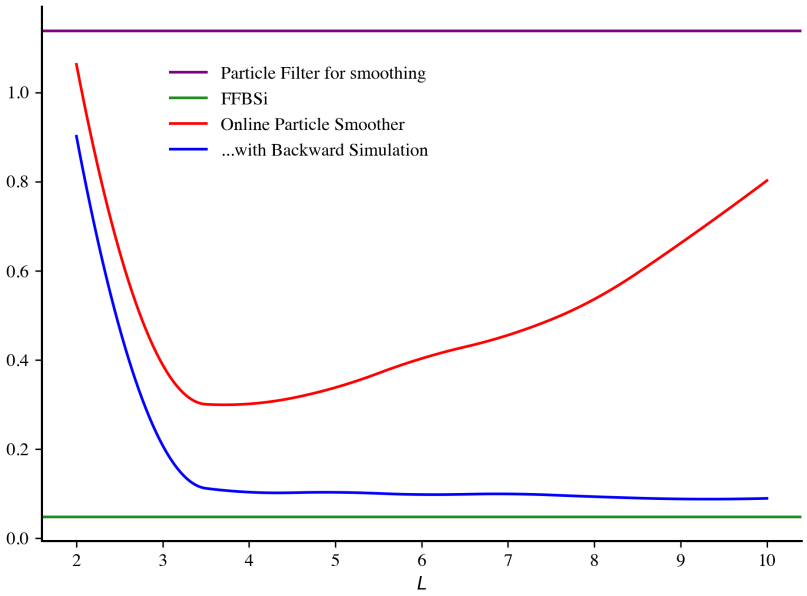


Fig. 3.11 $\mathrm{KL}(\hat{p}(x_{0:T}|y_{0:T})||p(x_{0:T}|y_{0:T}))$
for (joint) particle smoothing techniques with varying lag parameter.

The KL-divergence for the offline forward filtering-backward simulation is very small and therefore it is accurately approximating the smoothing mean $\bar{x}_{0:T}$ and covariance $\bar{\mathrm{C}}_{0:T}$ statistics. Both of these techniques are asymptotically unbiased for the true smoothing distribution and do not utilise a lag parameter. However, the online particle smoothers are asymptotically unbiased for the fixed-lag joint Equation (3.4) and we observe their performance is sensitive to the choice of lag parameter L. In particular, the online particle smoother (with blocks generated from the particle filter) is poor for small lags due to a divergence between $p^L(x_{0:T} \mid y_{0:T})$ and $p(x_{0:T} \mid y_{0:T})$, increasing the lag parameter mitigates this, but for suitably large lag parameters (here $L > 5$) path degeneracy sets in and more particles would be required. In contrast, the addition of backward simulation means the lag parameter can be set arbitrarily high to obtain performance approaching that of the offline FFBSi. Increasing the lag beyond $L = 5$ appears to only gradually increase the quality of the approximation. However, we must note that these takeaways are very much limited to this example and even this particular trajectory and set of observations, Figure 3.1 - in Chapter 4 we also examine the lag parameter for a particular map-matching model. In general, choosing the lag parameter is difficult as it depends on the mixing times of the state-space model Olsson et al. (2008).

In Figure 3.12 and Figure 3.13 we investigate the effect of choosing the maximum number of rejections sent to the rejection sampler. Recall for each choice of R the algorithm samples from the same distribution and therefore it is a purely computational parameter. Further that setting $R = 0$ regains the full N^2 stitching (3.7) and backward simulation (2.41) techniques. We

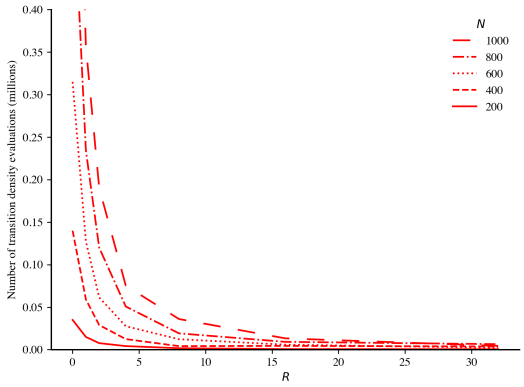


Fig. 3.12 Online Particle Smoother

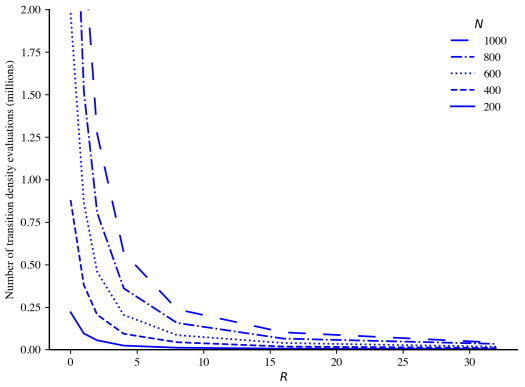


Fig. 3.13 with Backward Simulation

Number of transition density $p(x_t \mid x_{t-1})$ evaluations per observation for varying maximum number of rejections R for fixed $L = 5$.

observe that larger values of R result in far fewer evaluations of the transition density $p(x_t \mid x_{t-1})$, which represents the $O(N^2)$ bottleneck in the full stitching and backward simulation techniques. The effect of this parameter is further examined in Taghavi et al. (2013) and in the context of map-matching in Chapter 4.

3.5 Discussion

This chapter has introduced a solution to the problem of path degeneracy in classical particle filtering. We do so by first invoking a fixed-lag approximation and secondly adapting the technique that underpins backward simulation in order to execute an online forward simulation. As with backward simulation, this comes at a complexity of $O(N^2)$, we can however adopt the rejection sampling strategy of Douc et al. (2011) and reduce the complexity to $O(N)$ when a bound for the transition density is available. An important feature of the presented online smoothers is that they target an approximation to the joint smoothing distribution $p(x_{0:T} \mid y_{0:T})$ and therefore each particle represents a plausible, complete trajectory permitting mathematical expectations with complete generality, this contrasts with alternative marginal or additive functional based online smoothing techniques.

The choice of L determines the distance between the distributions $p^L(x_{0:T}|y_{0:T})$ and $p(x_{0:T}|y_{0:T})$, naturally we desire this to be small and therefore L large. The question of how large is a difficult one as it is dependent on the mixing of the state-space model, as discussed in Olsson et al. (2008) - for now we recommend choosing the lag parameter based on preliminary runs with synthetic data simulated from a given state-space model. An interesting extension would be to investigate the possibility of using a variable lag parameter which is

chosen dynamically, as achieved for a function specific version of the marginal fixed-lag particle filter in Alenlöv and Olsson (2019).

In some state-space models, the transition bound is not easily available - particularly when it varies between observation times. A pragmatic approach is to find the bound adaptively. One can set an initial psuedo-bound as the maximum of the first N transition density evaluations before proceeding as usual. On the event a new evaluation exceeds the pseudo-bound, the pseudo-bound can be reset and the algorithm (at that observation time) restarted. Indeed this implementation is available in `mocat`, Chapter 7, although caution is required as the transition density could well be unbounded - in which case the $O(N^2)$ variant is recommended.

The realisation of a low-probability transition or observation in the true underlying process can cause degeneracy either at stitching time or in the filtering weights. It is possible to reintroduce particle diversity by applying an MCMC kernel after stitching as in resample-move particle filters Gilks and Berzuini (2001) or particle rejuvenation Lindsten et al. (2015). The MCMC kernel is applied independently to each particle and must be invariant for the full smoothing distribution $p(x_{0:T}|y_{0:T})$ but as we are only concerned with increasing particle diversity rather than taking ergodic averages we need only propose moving a subset of the trajectories, whether that be at stitching time or the latest observation time.

Chapter 4

Map-Matching

In this chapter, we examine the task of map-matching - inferring a vehicle's trajectory given a road network and noisy GPS observations. We specifically focus on the difficult scenario of dense urban road networks. We formulate the problem as a state-space model and demonstrate the applicability of offline particle smoothing and the online particle smoothing from Chapter 3.

A road network is defined as a graph in $\mathbb{R}^2$, where intersections are represented by nodes (vertices). Roads, which are assumed to be single lane and one-way, are represented by edges which preserve the possibly complex geometry of the road network (a two-way road is represented by two edges). Some collections of nodes and edges are depicted in Figure 4.1 alongside some real GPS data.

We start by describing map-matching as a state-space model in Section 4.1 with a transition density that is specifically designed for difficult urban road networks. We then develop and demonstrate how to implement offline particle smoothing for given static parameter values of the state-space model's transition and observation densities in Section 4.2, before discussing how to tune said static parameters using a maximum likelihood approach. In Section 4.4 we examine the application of the online particle smoothers from Chapter 3 to map-matching and review in Section 4.5. Finally, this chapter is accompanied with a `python` package called `bmm` (Bayesian map-matching) that provides easy offline and online map-matching as well as parameter tuning - a description of `bmm`'s the core functionality is found in Section 4.6.

4.1 Model

There are two key points to consider in constructing a statistical model for map-matching, the first is the prior knowledge that the vehicle takes a continuous trajectory, which in this work we assume is pinned to the road network. The second is that at observation times the vehicle is (or

Fig. 4.1 Snapshot of Porto road network and some real GPS observations from the Porto taxi dataset Moreira-Matias et al. (2013).

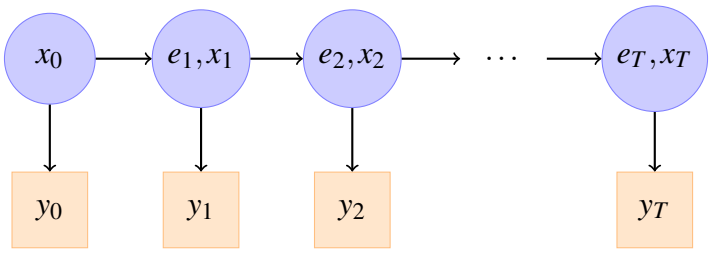


Fig. 4.2 Conditional independence structure of the state-space model for map-matching.

was) somewhat close to the received GPS observation coordinate - how close is defined by the level of the GPS noise which may or may not be known a priori.

4.1.1 Model Variables

We now formally define the variables that constitute a single vehicle's trajectory

- $e_t \subset \mathbb{N}$, a finite ordered set of connected edge labels, each edge label corresponds to a unique one-way section of road. The variable e_t details the edges traversed (and in which order) between observation times $t-1$ and t, including the choices made at encountered intersections (nodes).
- $x_t \in \mathbb{R}^2$ the position of the vehicle at observation time. The variable x_t defines a Cartesian coordinate restricted to lie on the road network, specifically x_t lies on the final edge of the finite ordered set of edge labels e_t, for $t > 0$.
- $y_t \in \mathbb{R}^2$ noisy observation of the vehicle's position x_t, not restricted to the road network.

Note here the change of notation from Chapter 3, now x_t refers only to vehicle position and the full latent states are $x_0, (e_1, x_1), (e_2, x_2), \ldots, (e_T, x_T)$ depicted as a state-space model in Figure 4.2. We will also use the notation $e(x_t)$ to denote the edge label of the edge on which x_t lies, whilst we denote e_t^o for the first edge in the series e_t and e_t^* for the last edge of e_t.

4.1.2 Model Distributions

The most popular map-matching approach is that of Newson and Krumm (2009) and variations thereof - such as the online adaptation in Goh et al. (2012). Their data generating process is summarised as

- $p(x_0)$ uniform on the road network.

- $p(e_t, x_t \mid x_{t-1}, y_{t-1}, y_t) \propto \exp(-\beta | \|x_t - x_{t-1}\|_{e_t} - \|y_t - y_{t-1}\| |)$ where $\|x_t - x_{t-1}\|_{e_t}$ is the distance travelled between x_{t-1} and x_t along the series e_t (restricted to the road network) whereas $\|y_t - y_{t-1}\|$ is the *great circle* distance between observations (not restricted to the road network).
- $p(y_t|x_t) = \mathcal{N}(y_t \mid x_t, \sigma^2_{\text{GPS}} I_2)$ isotropic Gaussian observation noise.

The motivation behind the transition density $p(e_t, x_t \mid x_{t-1}, y_{t-1}, y_t)$ is to penalise routes that are particularly windy or non-direct such as multiple loops of a roundabout - which if not taken into consideration could be given high probability in dense urban road networks.

The inference procedure of Newson and Krumm (2009) proceeds, at each observation time, by selecting a candidate point (typically the closest point to the observation) on each edge within a truncation distance (200m) of the observed GPS coordinate, y_t. This forms a discrete state-space and the Viterbi algorithm can be applied to produce a single route of high probability.

It was noted in Raymond et al. (2012) that in this formulation the transition density depends on the observations and therefore violates the conditional independence structure of a state-space model. They propose the logical modification $p(e_t, x_t \mid x_{t-1}) \propto \exp(-\beta | \|x_t - x_{t-1}\|_{e_t} - \|x_t - x_{t-1}\| |)$.

This Viterbi approach produces only a single route without uncertainty quantification which contrasts with Monte Carlo based techniques. Particle filters were applied to map-matching in Davidson et al. (2011); Kempinska et al. (2016) without tackling the problem of path degeneracy. In an offline setting, Roth et al. (2012) introduced the use of FFBSi for map-matching. In this work, we use a formulation that takes the particle smoothing approach of Roth et al. (2012) but includes a term in the transition density adapted from Newson and Krumm (2009) that penalises non-direct routes (which is vital in dense urban road networks) and an additional term that puts probability mass on a stationary vehicle between observation times (due to traffic lights or congestion). We also use the optimal proposal density (that takes into account the new observation) rather than simply the bootstrap proposal which will perform poorly for small GPS noise or dense road networks with frequent intersections.

Our transition density can be written as

$$p(e_t, x_t | x_{t-1}) = \frac{\gamma(\|x_t - x_{t-1}\|_{e_t}) \exp(-\beta | \|x_t - x_{t-1}\|_{e_t} - \|x_t - x_{t-1}\| |)}{Z(x_{t-1})}, \tag{4.1}$$

with normalising constant

$$Z(x_{t-1}) = \sum_{e_t} \int_{x_t} \gamma(\|x_t - x_{t-1}\|_{e_t}) \exp(-\beta | \|x_t - x_{t-1}\|_{e_t} - \|x_t - x_{t-1}\| |) \mathrm{d}x_t, \tag{4.2}$$

where we always have the constraints $e(x_{t-1}) = e_t^o$ (the first edge of e_t) and $e(x_t) = e_t^*$ (the last edge of e_t). Recall that e_t is an finite ordered set of edge labels, starting with e_t^o and ending with e_t^*. The summation in (4.2) is taken over all possible series of edges starting at $e(x_{t-1})$.

Thus, the following distributions fully define our state-space model:

- $\gamma(\|x_t - x_{t-1}\|_{e_t})$. Prior on distance travelled between observations - some simple analytical distribution on $\mathbb{R}^+$, penalising lengthy routes. We assume an exponential distribution with probability mass at 0 to represent the possibility of the vehicle remaining stationary (at traffic lights, in heavy traffic etc)

$$\begin{aligned}&\gamma(\|x_t - x_{t-1}\|_{e_t}) \\ &\quad = p^0 \mathbb{I}(\|x_t - x_{t-1}\|_{e_t} = 0) + (1-p^0)\mathbb{I}(\|x_t - x_{t-1}\|_{e_t} > 0)\lambda \exp(-\lambda \|x_t - x_{t-1}\|_{e_t}).\end{aligned} \tag{4.3}$$

- $\exp(-\beta |\|x_t - x_{t-1}\|_{e_t} - \|x_t - x_{t-1}\||)$ adapted from Newson and Krumm (2009), penalising non-direct (or winding) routes. Non-direct routes with lots of curvature will have a high discrepancy between the road distance travelled and great circle distance and thus will have a low probability under this term, reflecting a driver's preference to take short, direct routes where possible.

- $p(y_t|x_t) = \mathcal{N}(y_t|x_t, \sigma^2_{\text{GPS}} I_2)$. Isotropic Gaussian observation noise.

- We set $p(x_0)$ to be uniform on the road network. I.e. no prior information on the start of the trajectory other than constricting it to the road network (as with all inferred positions).

To make $p(x_0|y_0)$ tractable we define the initial observation density to be a truncated Gaussian:

$$p(y_0|x_0) \propto \mathcal{N}(y_0|x_0, \sigma^2_{\text{GPS}} I_2)\, \mathbb{I}(||y_0 - x_0|| < r_{\text{GPS}}),$$

giving

$$p(x_0|y_0) \propto \mathcal{N}(x_0|y_0, \sigma^2_{\text{GPS}} I_2)\, \mathbb{I}(||y_0 - x_0|| < r_{\text{GPS}}),$$

where x_0 is restricted to the road network but y_0 is not. We set $r_{\text{GPS}} = 5\sigma_{\text{GPS}}$.

4.1.3 Optimal Proposal

The (locally) optimal proposal Doucet et al. (2000) for particle filtering combines the transition density Equation (4.1) and the newly received observation y_T:

$$q^{\text{opt}}(x_T, e_T \mid x_{T-1}, e_{T-1}, y_T) \propto p(e_T, x_T|x_{T-1}) p(y_T|x_T). \tag{4.4}$$

The standard reweighting step of the particle filter update (Algorithm 12) then becomes

$$w_T^{\text{opt}\,(i)} \propto p(y_T|x_{T-1}^{(i)}),$$

where

$$p(y_T|x_{T-1}) = \sum_{e_T} \int_{x_T} p(e_T, x_T|x_{T-1}) p(y_T|x_T)\, \mathrm{d}x_T, \tag{4.5}$$

is the normalisation constant of the proposal distribution, Equation (4.4).

Neither sampling from the optimal proposal (4.4), nor evaluating the subsequent weights (4.5), nor evaluating the normalising constant of the transition density (4.2) are immediately tractable as we do not have closed form expressions for the edge geometries.

Instead, we opt to approximate the required integrals numerically by discretising the edges up to some maximal possible distance travelled $d_{\max}$. This numerical integration can be implemented efficiently across particles by caching route searches and likelihood evaluations - as many particles will typically lie on the same or adjacent edges.

4.2 Offline Smoothing

We now have all of the tools we need to develop an offline map-matching technique that provides uncertainty quantification through a collection of particles - each one representing a plausible continuous trajectory for the vehicle. Indeed, this mirrors the approach of Roth et al. (2012) but with the aforementioned model modifications and particle filter proposals generated from the optimal proposal density.

4.2.1 Backward Simulation

In the context of backward simulation, we propose combining

$$\left(x_{t:T}^{(j)}, e_{t:T}^{(j)}\right) \text{ with a sample from } \left\{\left(\tilde{x}_{t-1}^{(i)}, \tilde{e}_{t-1}^{(i)}\right)\right\}_{i=1}^{N}.$$

The backward weights take the form

$$\begin{aligned} w_{t-1}^{(i\leftarrow j)} &\propto \tilde{w}_{t-1}^{(i)} p\left(e_t^{(j)}, x_t^{(j)} \mid \tilde{e}_{t-1}^{(i)}, \tilde{x}_{t-1}^{(i)}\right), \\ &= \tilde{w}_{t-1}^{(i)} p\left(e_t^{(j)}, x_t^{(j)} \mid \tilde{x}_{t-1}^{(i)}\right) \mathbb{I}\left[e\left(\tilde{x}_{t-1}^{(i)}\right) = e_t^{o\,(j)}\right], \end{aligned}$$

which we can calculate in closed form under the aforementioned discretisation to implement the full $O(N^2)$ backward simulation. For the $O(N)$ rejection sampling variant in the backward

direction we require a bound

$$p\left(e_t, x_t \mid x_{t-1}\right) < C, \quad \forall x_{t-1}, e_t, x_t.$$

On inspection of Equation (4.1) we note that this equivalent to bounding $\gamma(\|x_t - x_{t-1}\|_{e_t})$ - which we can do so with $C = \max(p^0, (1-p^0)\lambda)$.

In practice, the acceptance rate of the rejection sampling will depend greatly on the quality of this bound - which depends on the value of the parameters p^0 and λ. In dense urban road networks, we will typically have $p^0 >> (1-p^0)\lambda$. However, at a given observation time, we may often have that none of the N^2 particles overlap such that $\|x_t^{(j)} - \tilde{x}_{t-1}^{(i)}\|_{e_t^{(j)}} = 0$ and therefore using the larger bound $C = p^0$ would be wasteful. We can speed up the rejection sampling by adopting the adaptive approach described in Section 3.5 where we initiate the bound at each observation time with the smaller $C = (1-p^0)\lambda$ and if any particles are observed to overlap such that $\|x_t^{(j)} - \tilde{x}_{t-1}^{(i)}\|_{e_t^{(j)}} = 0$, we restart the rejection sampling at the observation time with $C = p^0$.

4.2.2 Synthetic Data

In Figures 4.3-4.5 we demonstrate the benefits of uncertainty quantification for offline map-matching by comparing FFBSi (Figure 4.4) against the popular optimisation approach of Newson and Krumm (2009) (Figure 4.3) on challenging synthetic observations for a trip between the Cambridge Engineering department and the Fort St George pub. For FFBSi, we use parameter values discussed in the next section, Section 4.3.

Although the optimisation based approach finds a plausible route (point estimate) it misses out on others that are equally plausible and thus valuable information is lost, this is particularly prevalent when inferring the distances the vehicle travelled, Figure 4.5. There is significant uncertainty in both the edges traversed and the distances travelled that is captured by FFBSi but not by the Viterbi algorithm.

All particles generated by FFBSi appear plausible - more direct routes are preferred but not overly so - this provides evidence to suggest the model is well suited to difficult dense urban road networks and that the optimal proposal is efficiently generating high probability particles.

Fig. 4.3 Offline FFBSi with $N = 1000$ for synthetic trip across Cambridge.

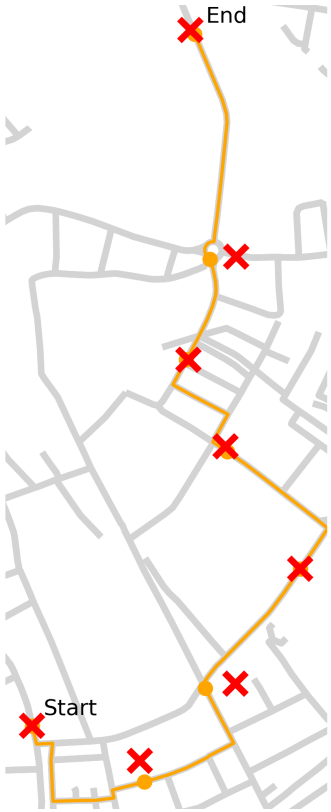


Fig. 4.4 Viterbi map-matching Newson and Krumm (2009) for synthetic trip across Cambridge.

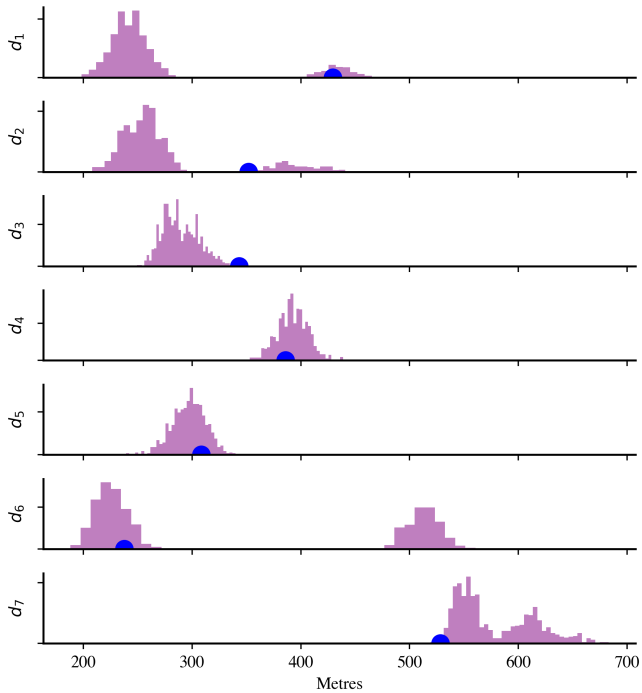


Fig. 4.5 Histograms represent $p(\|x_t - x_{t-1}\|_{e_t} | y_{0:T})$ from FFBSi. Spots represent distances inferred using Viterbi map-matching.

4.3 Parameter Inference

We have described our state-space model for map-matching in Section 4.1, however we are still left to determine suitable values for the parameters $\theta = (p^0, \lambda, \beta, \sigma_{\text{GPS}})$. We first describe the intuition behind each of the parameters

- p^0 - defines the a priori probability of the vehicle remaining stationary between observations.
- λ - large values of λ penalise long distance routes, in particular vehicles are not expected to travel at speeds above the speed limit.
- β - large values of β penalise routes with significant curvature, such as multiple loops of a roundabout.
- σ_{GPS} - controls the level of noise in the GPS observations, which is assumed to be the same in both x and y directions and constant across observation times.

4.3.1 Expectation Maximisation

A very general approach to parameter inference in Bayesian models is to explicitly rewrite the joint distribution to be conditioned on the parameters

$$p_\theta(x,y) = p(x,y \mid \theta). \tag{4.6}$$

The goal is then to find the value of the parameter θ that maximises the marginal likelihood

$$\theta_{\text{MLE}} = \arg\max_\theta p(y \mid \theta), \tag{4.7}$$

where

$$p(y \mid \theta) = \int p(x,y \mid \theta) dx.$$

Note that we could additionally put a prior on the parameter θ and then optimise to find the maximum a posteriori parameter $\theta_{\text{MAP}} = \arg\max_\theta p(y \mid \theta)p(\theta)$ or even use MCMC methods to explore the posterior $p(\theta \mid y)$ or $p(\theta, x \mid y)$ as in the popular particle MCMC of Andrieu et al. (2010).

Expectation maximisation is an iterative algorithm that searches for a local maximum of $p(y \mid \theta)$ (or $p(\theta \mid y)$). Given a current value $\theta^{(\ell)}$, a single iteration of expectation maximisation is composed of two steps

1. **E-step**: Compute $\mathbf{Q}(\theta \mid \theta^{(\ell)}) = \mathbb{E}_{p(x|y,\theta^{(\ell)})}[\log p(x,y \mid \theta)]$.
2. **M-step**: $\theta^{(\ell+1)} = \arg\max_{\theta} \mathbf{Q}(\theta \mid \theta^{(\ell)})$.

We can show that each iteration can only increase the value of $p(y \mid \theta)$. First rewrite

$$\log p(x,y \mid \theta) = \log p(y \mid \theta) + \log p(x \mid y, \theta),$$

and take expectations with respect to $p(x \mid y, \theta^{(\ell)})$

$$\mathbf{Q}(\theta \mid \theta^{(\ell)}) = \log p(y \mid \theta) + \mathbb{E}_{p(x|y,\theta^{(\ell)})}[\log p(x \mid y, \theta)].$$

We then consider $\mathbf{Q}(\theta^{(\ell+1)} \mid \theta^{(\ell)}) - \mathbf{Q}(\theta^{(\ell)} \mid \theta^{(\ell)})$ which is non-negative by the M-step

$$\begin{aligned}
&\mathbf{Q}(\theta^{(\ell+1)} \mid \theta^{(\ell)}) - \mathbf{Q}(\theta^{(\ell)} \mid \theta^{(\ell)}) \\
&\qquad = \log p(y \mid \theta^{(\ell+1)}) - \log p(y \mid \theta^{(\ell)}) + \mathbb{E}_{p(x|y,\theta^{(\ell)})}\left[\log \frac{p(x \mid y, \theta^{(\ell+1)})}{p(x \mid y, \theta^{(\ell)})}\right] \\
&\qquad \geq 0.
\end{aligned}$$

By Jensen's equality we have

$$\mathbb{E}_{p(x|y,\theta^{(\ell)})}\left[\log \frac{p(x \mid y, \theta^{(\ell+1)})}{p(x \mid y, \theta^{(\ell)})}\right] \leq \log \mathbb{E}_{p(x|y,\theta^{(\ell)})}\left[\frac{p(x \mid y, \theta^{(\ell+1)})}{p(x \mid y, \theta^{(\ell)})}\right] = 0.$$

Therefore

$$\begin{aligned}
& \log p(y \mid \theta^{(\ell+1)}) \geq \log p(y \mid \theta^{(\ell)}) \\
\implies \qquad & p(y \mid \theta^{(\ell+1)}) \geq p(y \mid \theta^{(\ell)}).
\end{aligned}$$

When we cannot evaluate the integral $\mathbf{Q}(\theta \mid \theta^{(\ell)}) = \mathbb{E}_{p(x|y,\theta^{(\ell)})}[\log p(x,y \mid \theta)]$ analytically, we typically resort to approximate methods such as Monte Carlo (or variational inference). In the case that $p(x,y \mid \theta)$ is part of the *exponential family*, the M-step can be solved exactly. However, in general the M-step can be difficult or intractable but the gradient $\nabla_\theta \mathbf{Q}(\theta \mid \theta^{(\ell)})$ may be available and thus we can replace the M-step with a gradient step

$$\theta^{(\ell+1)} = \theta^{(\ell)} + \varepsilon_\ell \nabla_\theta \mathbf{Q}(\theta \mid \theta^{(\ell)})|_{\theta=\theta^{(\ell)}},$$

which gives *gradient EM*.

Algorithm 18 Expectation Maximisation

1: Given initial $\theta^{(0)}$
2: **for** $\ell = 0, \ldots, L-1$ **do**
3: Approximate $\mathbf{Q}(\theta \mid \theta^{(\ell)}) = \mathbb{E}_{p(x|y,\theta^{(\ell)})}[\log p(x, y \mid \theta)]$
e.g. generate samples $\{x^{(i)}\}_{i=1}^{N} \sim p(x \mid y, \theta^{(\ell)})$
4: Set $\theta^{(\ell+1)} = \arg\max_\theta \mathbf{Q}(\theta \mid \theta^{(\ell)})$
or if this is intractable set $\theta^{(\ell+1)} = \theta^{(\ell)} + \varepsilon_\ell \nabla_\theta \mathbf{Q}(\theta \mid \theta^{(\ell)})|_{\theta=\theta^{(\ell)}}$
5: **return** $\theta^{(L)}$

In the context of offline parameter inference in state-space models, the E-step becomes

$$\mathbf{Q}(\theta \mid \theta^{(\ell)}) = \mathbb{E}_{p(x_{0:T}|y_{0:T},\theta^{(\ell)})}\left[\log p_\theta(x_0) + \sum_{t=1}^{T} \log p_\theta(x_t \mid x_{t-1}) + \sum_{t=0}^{T} \log p_\theta(y_t \mid x_t)\right],$$

which can be approximated using particle smoothing (FFBSi)

$$\mathbf{Q}(\theta \mid \theta^{(\ell)}) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\log p_\theta\left(x_0^{(i)}\right) + \sum_{t=1}^{T} \log p_\theta\left(x_t^{(i)} \mid x_{t-1}^{(i)}\right) + \sum_{t=0}^{T} \log p_\theta\left(y_t \mid x_t^{(i)}\right)\right].$$

4.3.2 Offline Parameter Inference for Map-matching

As discussed above, we can use the expectation maximisation algorithm with FFBSi to tune the map-matching parameters $\theta = (p^0, \lambda, \beta, \sigma_{\text{GPS}})$. In the M-step we can maximise analytically with respect to the parameter σ_{GPS}, however the normalisation constant (4.2) makes maximisation of the parameters p^0, λ and β intractable and thus we take gradient steps in these parameters with decreasing stepsize $\varepsilon_\ell = 10^{-5} \times \ell^{-0.5}$.

In Figure 4.6, we verify the expectation maximisation algorithm, running FFBSi at each iteration over 20 trips simulated from the map-matching model with known parameters on the Cambridge road network. The algorithm correctly converges to the vicinity of the true parameters.

We then tune the parameters on the map-matching model over 20 real trips from the Porto taxi dataset Moreira-Matias et al. (2013) in Figure 4.7. The result is $p^0 \approx 0.14$, $\lambda \approx 0.07$, $\beta \approx 0.05$ and $\sigma_{\text{GPS}} \approx 5.2$ which are the parameters we used in Section 4.2.2 and in Section 4.4.

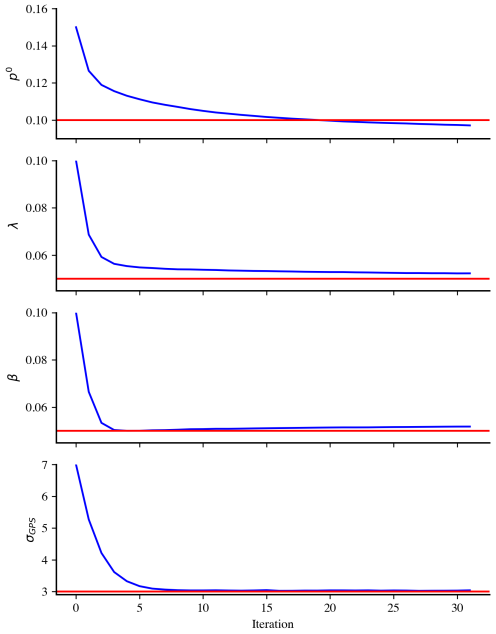


Fig. 4.6 Convergence of expectation maximisation over 20 synthetic trips in Cambridge. True parameters in red.

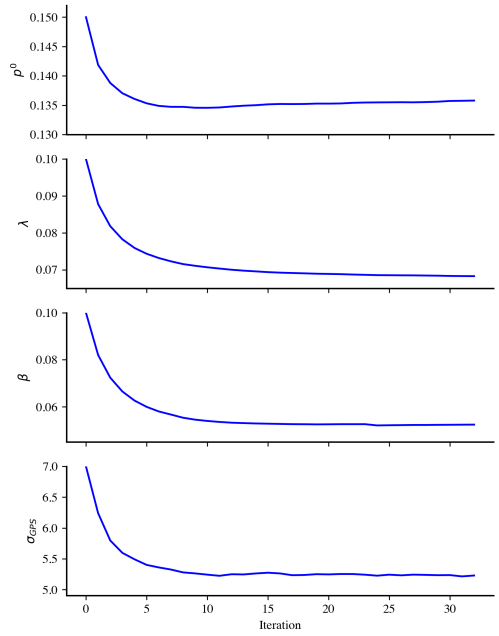


Fig. 4.7 Convergence of expectation maximisation over 20 real trips in Porto.

4.4 Online Smoothing

We now apply the online particle smoothers introduced in Chapter 3 to the problem of map-matching. Recall that the goal of the online smoothing is to obtain an approximation to $p(x_{0:T-1} \mid y_{0:T-1})$ that can be quickly and accurately updated to approximate $p(x_{0:T} \mid y_{0:T})$ on receipt of a new observation y_T. More specifically the techniques developed in Chapter 3 utilise a fixed-lag approximation to update the particles with a complexity that is independent of T.

The advantages of targeting the full joint smoothing distribution $p(x_{0:T} \mid y_{0:T})$ rather than smoothing marginals $\{p(x_t \mid y_{0:T})\}_{t=0}^{T}$, Kitagawa and Sato (2001) are particularly prevalent in map-matching. By only targeting smoothing marginals, arbitrary stitching cannot produce a continuous trajectory, Figure 4.8, and therefore does not permit expectations over multiple observation times. In contrast, fixed-lag particle stitching, Figure 4.9, produces particles that each represent a continuous trajectory and therefore can be used to approximate mathematical expectations with complete generality.

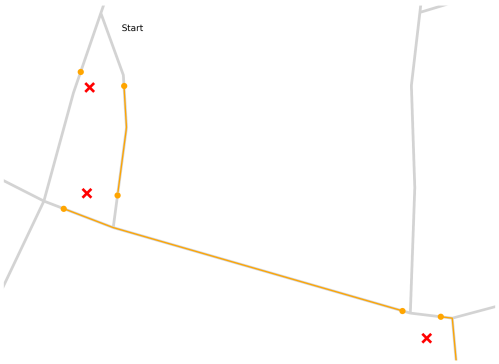


Fig. 4.8 Single particle with arbitrary stitching targeting $\{p(x_t \mid y_{0:T})\}_{t=0}^{T}$.

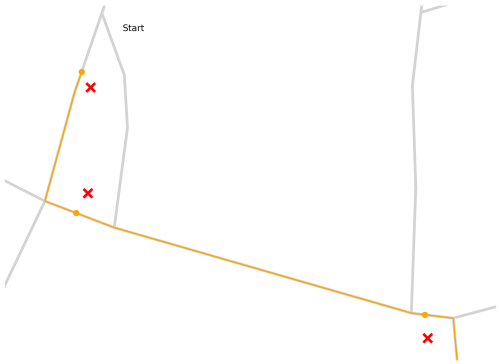


Fig. 4.9 Single particle with fixed-lag particle stitching targeting $p(x_{0:T} \mid y_{0:T})$.

4.4.1 Fixed-lag Particle Stitching

In the context of the particle stitching described in Section 3.2, we propose stitching together each

$$\left(x_{0:T-L-1}^{(i)}, e_{1:T-L-1}^{(i)}\right) \text{ with a sample from } \left\{\left(\tilde{x}_{T-L-1:T}^{(j)}, \tilde{e}_{T-L:T}^{(j)}\right)\right\}_{j=1}^{N}.$$

where $\tilde{x}_{T-L-1:T}$ is an overlapping coordinate required for fixed-lag particle stitching, that is ultimately discarded. Thus the fully adjusted weights become

$$w_T^{(i\to j)} \propto \frac{p(\tilde{e}_{T-L}^{(j)}, \tilde{x}_{T-L}^{(j)} | x_{T-L-1}^{(i)})}{p(\tilde{e}_{T-L}^{(j)}, \tilde{x}_{T-L}^{(j)} | \tilde{x}_{T-L-1}^{(j)})} w_T^{(j)} \mathbb{I}\left(e(x_{T-L-1}^{(i)}) = \tilde{e}_{T-L}^{o\,(j)}\right).$$

Similarly, for the rejection sampling we get non-interacting weights

$$\hat{w}_T^{(j)} \propto \frac{w_T^{(j)}}{p(\tilde{e}_{T-L}^{(j)}, \tilde{x}_{T-L}^{(j)} | \tilde{x}_{T-L-1}^{(j)})},$$

and we accept a sample if $e(x_{T-L-1}^{(i)}) = \tilde{e}_{T-L}^{o\,(j)}$ and

$$u < \frac{\gamma\left(\|\tilde{x}_{T-L}^{(j)} - x_{T-L-1}^{(i)}\|_{\tilde{e}_{T-L}^{(j)}}\right) \exp\left(-\beta \left|\|\tilde{x}_{T-L}^{(j)} - x_{T-L-1}^{(i)}\|_{\tilde{e}_{T-L}^{(j)}} - \|\tilde{x}_{T-L}^{(j)} - x_{T-L-1}^{(i)}\|\right|\right)}{C},$$

where $u \sim \mathbf{U}(\cdot \mid 0,1)$. We can adopt the same adaptive procedure for the bound parameter $C > \gamma\left(\|\tilde{x}_{T-L} - x_{T-L-1}\|_{e_{T-L}}\right)$ as described in Section 4.2.1.

4.4.2 Real Data

We now explore the effects of the algorithmic parameters in the online particle smoothers (sample size, lag parameter and number of rejections) for map-matching a route from the Porto taxi dataset Moreira-Matias et al. (2013), Figure 4.18.

In Figure 4.10 and Figure 4.11, we analyse the posterior distribution for the cumulative (road) distance travelled by the taxi across each minute. As observations are received every 15 seconds this amounts to expectations averaged over blocks of the full smoothing distribution and thus marginal approximations, Figure 4.8, would be insufficient. For each minute of the trip, we compare particle approximations by calculating a total variation metric over the empirical distributions. This total variation metric is calculated by binning the distance travelled variable, then the empirical distributions are defined over a discrete space and the total variation metric is tractable

$$\mathrm{TV}\left(\left\{d_1^{(i)}\right\}_{i=1}^{N_1}, \left\{d_2^{(j)}\right\}_{j=1}^{N_2}\right) = \frac{1}{2}\sum_{\mathfrak{d}\in\mathfrak{D}}\left|\frac{\sum_{i=1}^{N_1}\mathbb{I}[d_1^{(i)}\in\mathfrak{d}]}{N_1} - \frac{\sum_{j=1}^{N_2}\mathbb{I}[d_2^{(j)}\in\mathfrak{d}]}{N_2}\right|,$$

where $\mathfrak{D}$ is the space $[0, d_{\max}]$ discretised to 5m incremental bins.

Figures 4.12-4.15 depicts the varying algorithmic performance on the start of the route.

We initially observe that setting $L = 0$ performs very poorly for all sample sizes. The overly small lag parameter results in a large deviation between the true smoothing distribution $p(x_{0:T} \mid y_{0:T})$ and the joint fixed-lag smoothing distribution $p^L(x_{0:T} \mid y_{0:T})$ (3.4) - Figure 4.13.

The online particle smoother suffers from path degeneracy for the large lag $L = 10$ as observed by a loss of particle diversity (compared to the backward simulation techniques) in Fig. 4.14. It does however perform well for $L = 3$.

The addition of backward simulation avoids the issue of path degeneracy in this setting. We observe that increasing the lag parameter from $L = 3$ to $L = 10$ does little to improve performance and as such can posit that the distributions $p(x_{0:T} \mid y_{0:T})$ and $p^L(x_{0:T} \mid y_{0:T})$ are suitably close for $L = 3$, at least for the observations on this route, Figure 4.18.

Finally, in Figure 4.16 and Figure 4.17 we compare the effect on algorithmic runtime from increasing the maximum number of rejections, R, attempted in the hybrid stitching scheme Algorithm 15, as well as backward simulation if applicable. Recall that setting $R = 0$ recovers the full $O(N^2)$ scheme. For a suitably large number of rejections the runtimes of both algorithms can be seen to increase linearly in N.

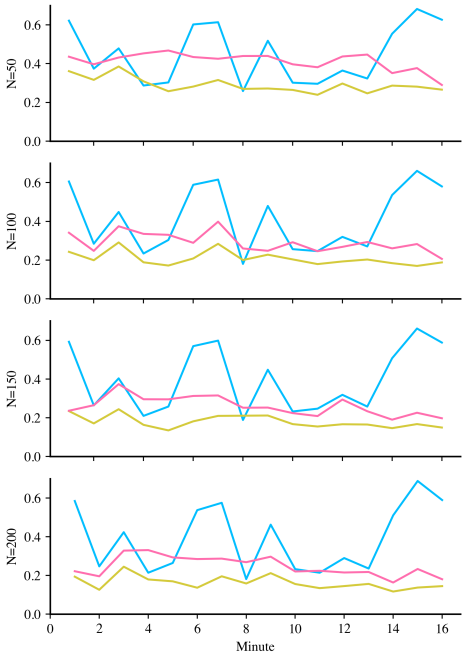


Fig. 4.10 Online Particle Smoother

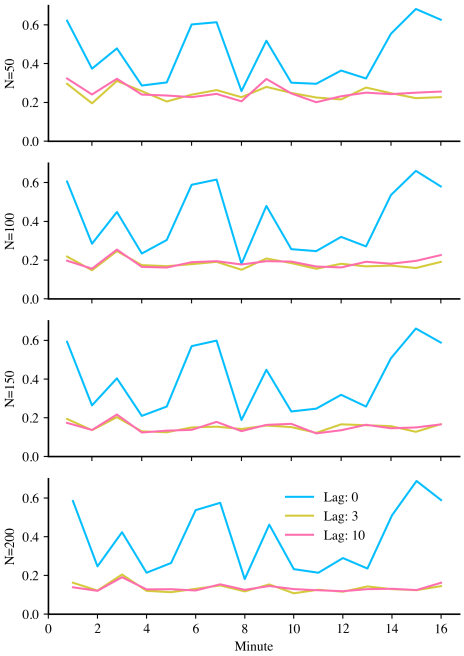


Fig. 4.11 with Backward Simulation.

Total variation distance from gold standard (offline FFBSi) for posterior over cumulative distance travelled in each minute of a 16 minute taxi route (observation every 15 seconds) Moreira-Matias et al. (2013). Simulations averaged over 20 random seeds.

Fig. 4.12 Gold standard, offline FFBSi with $N = 1000$.

Fig. 4.13 Online particle smoother with $L = 0$ and $N = 200$. Poor approximation due to small lag.

Fig. 4.14 Online particle smoother, $L = 10$ and $N = 200$. Evidence of path degeneracy due to large lag.

Fig. 4.15 Online particle smoother with backward simulation, $L = 10$ and $N = 200$.

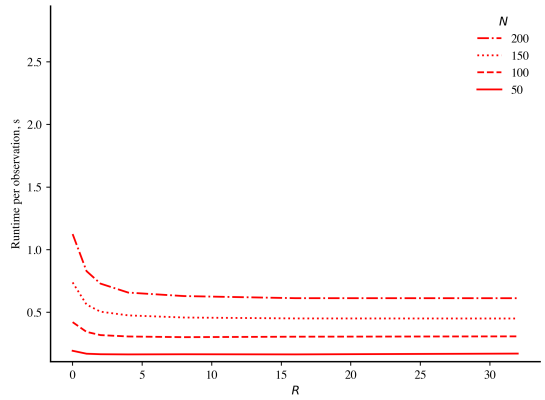


Fig. 4.16 Online Particle Smoother

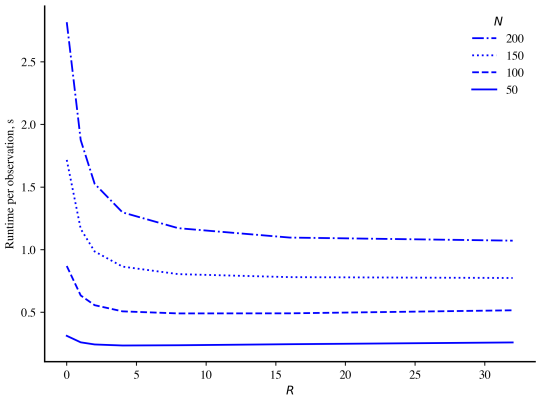


Fig. 4.17 with Backward Simulation.

Algorithmic runtime vs maximum number of rejections tolerated for $L = 3$. Runtimes averaged over 65 observations (Fig. 4.18) and 20 random seeds.

4.5 Discussion

This chapter has focused on the specific application of Monte Carlo techniques to map-matching. In dense, urban networks inferring the true trajectory of a vehicle given infrequent and/or noisy GPS observations can be a challenging task. By adopting a particle based approach targeting the smoothing distribution we provide a collection of plausible continuous trajectories for the vehicle, from which generic quantities can be estimated with uncertainty quantification - such as the speed or distance travelled of the vehicle as well as their decisions at intersections.

In Section 4.1, we have developed a map-matching state-space model that explicitly restricts vehicle movement to the road network and is specifically designed to handle urban networks with frequent intersections and roads with complex geometry. We discussed how to apply efficient particle smoothing given parameter values and GPS observations in Section 4.2 and demonstrated the validity of the model as well as the benefits of uncertainty quantification over the state-of-the-art optimisation based approach of Newson and Krumm (2009). We then described how to tune the parameters given offline GPS observations in Section 4.3 before applying the novel online smoothing techniques from Chapter 3 to obtain real time particle approximations to a vehicle's continuous route and location.

Looking forward, it would be interesting to develop more a sophisticated form for the γ term in the transition density that penalises lengthy routes. In particular, this should be developed to incorporate further information such as irregular timestamps (observation times) and speed limits. Further, the model could also be enhanced to encode information on road systems and traffic lights. However these enhancements assume extra information that may or may not be available.

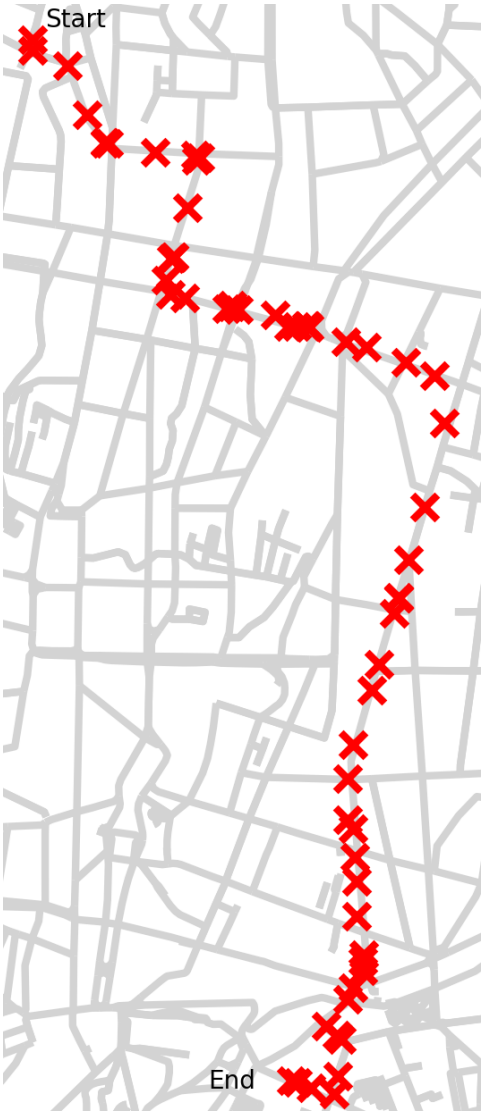


Fig. 4.18 GPS observations for single taxi route Moreira-Matias et al. (2013).

It would also be useful to extend map-matching to multiple lane road networks. This extension is nontrivial as vehicles have the opportunity to switch lanes without reaching a formal intersection - therefore they would have to be treated differently (rather than simply adding additional road labels).

Map-matching is often used as an internal tool within a larger inference goal. Therefore, an exciting extension is to use particle map-matching to address problems such as analysing driver behaviour or congestion prediction, particularly in online settings.

4.6 bmm

This chapter is accompanied by `bmm`, an open source `python` Van Rossum and Drake (2009) package that implements both offline and online particle smoothing for map-matching. The documentation for the `bmm` package can be found at bmm.readthedocs.io as well as source code at github.com/SamDuffield/bmm, including code to reproduce all of the simulations in this chapter. In this section we review the key functionality of the `bmm` package, which is hosted on PyPI (2021) and can be installed easily from the command line with `pip`.

```
pip install bmm
```

4.6.1 Downloading a graph

`bmm` is built on top of `OSMnx` Boeing (2017), which is a useful tool for downloading and processing data from OpenStreetMap contributors (2017).

```
import osmnx as ox
import bmm
```

Using `OSMnx` you can easily download the graph for a given region into `python`

```
graph = ox.graph_from_place('Cambridge, UK')
```

or more flexibly by defining a longitude-latitude bounding box

```
graph = ox.graph_from_bbox(north=52.22, south=52.19, east=0.136,
 ↪  west=0.11)
```

Either of these commands will download the OpenStreetMap data encoding the road network for the region and store it in a NetworkX object Hagberg et al. (2008). By default, `OSMnx` will simplify the road network, removing redundant edges and nodes whilst storing the non-linear edge geometry. In fact, `bmm` is applicable to any NetworkX graph as long as edges are labelled with `u, v, k, geometry` variables, where `u` and `v` are origin and destination nodes, `k` is a key differentiating the possibility of multiple edges with the same `u` and `v` and `geometry` encodes the edge curvature. Note that edges are assumed to be single lane and one-way - a two-way road is represented by two edges. By default, `OSMnx` downloads all streets and paths stored in the region, however this can be adjusted to filter for only `drive` / `bike` / `walk` streets with the `network_type` argument.

Longitude-latitude represents a spherical coordinate system with unit degrees, this can be cumbersome for inference purposes. It is common practice to project into the UTM (Universal Transverse Mercator) *x*-*y* coordinate system which divides the world into sixty square zones with unit metres. The `graph` object can be projected into UTM automatically

```
graph = ox.project_graph(graph)
```

4.6.2 Offline Map-matching

Assuming we have GPS observations of a route loaded in longitude-latitude polyline form, i.e. a list or numpy array Harris et al. (2020) of 2 dimensional longitude-latitude coordinates

```
import numpy as np
polyline_longlat = np.array([[-8.613234, 41.1669],
                             [-8.613243, 41.166666],
                             [-8.612253, 41.166342],
                             [-8.611695, 41.16528],
                             [-8.611182, 41.164668],
                             [-8.611173, 41.164677], ...
```

we can use bmm to convert to UTM *x*-*y* coordinates

```
polyline_utm = bmm.long_lat_to_utm(polyline_longlat)
polyline_utm
    array([[532445.46, 4557356.90],
           [532444.82, 4557330.92],
           [532528.03, 4557295.32],
           [532575.36, 4557177.63],
           [532618.70, 4557109.89],
           [532619.45, 4557110.89], ...
```

We are now ready to map-match the polyline to the graph using the forward filtering-backward simulation algorithm

```
matched_particles = bmm.offline_map_match(graph,
 ↪  polyline=polyline_utm, n_samps=100, timestamps=15)
```

The `n_samps` parameter represents the number of particles to generate whereas the `timestamps` parameters is either the constant time interval between observations (in seconds) or an array of UNIX timestamps (in seconds) representing each observation time.

The output is a `bmm.MMParticles` object that stores the inferred route for each particle. In particular, it contains a `particles` attribute - a list of arrays, each array has a row every time the route reaches an intersection or if an observation occurs. Each array has columns

- `t` - time (seconds) if row represents an observation time otherwise 0,
- `u` - index of edge origin node,
- `v` - index of edge destination node,
- `k` - edge key (0 unless there are multiple `u`-`v` edges in the graph),
- `alpha` - proportional position in [0,1] along edge,
- `x` - *x* coordinate if row represents an observation time otherwise 0,
- `y` - *y* coordinate if row represents an observation time otherwise 0,
- `d` - distance travelled since previous observation time.

```
matched_particles.particles[0]
 array([[0.,  4.53e+09, 4.74e+08, 0., 0.1, 5.32e+05, 4.56e+06, 0.],
        [0.,  4.53e+09, 4.75e+08, 0., 1.,  0.,       0.,       39.],
        [15., 4.75e+08, 4.75e+08, 0., 0.9, 5.32e+05, 4.56e+06, 52.12],
        [0.,  4.75e+08, 4.75e+08, 0., 1.,  0.,       0.,       2.],
        ↪  ...
```

The map-matched particles can be plotted alongside the observed polyline

```
bmm.plot(graph, particles=matched_particles, polyline=polyline_utm)
```

Fig. 4.19 Taxi polyline (red) from Moreira-Matias et al. (2013) map-matched (orange) and visualised using bmm.

4.6.3 Online Map-matching

In the case that we receive observations in an online fashion, we can update our `MMParticles` object as GPS observations arrive.

First we have to initiate the `MMParticles` object with the first observation in the polyline (which assume is already in UTM coordinates)

```
matched_particles = bmm.initiate_particles(graph,
 ↪  first_observation=polyline_utm[0], n_samps=100)
```

Then we can extend the particle approximation as new observations are received

```
matched_particles = bmm.update_particles(graph, matched_particles,
 ↪  new_observation=polyline_utm[1], time_interval=15)
```

The `bmm.update_particles` implements the fixed-lag online smoothing algorithms introduced in Chapter 3 and described for map-matching Section 4.4. It has a number of additional, optional parameters. An `update` parameter which can be set to either `'PF'` or `'BSi'` to choose between online smoothing with particle filter block sampling Algorithm 16 or with backward simulation block sampling Algorithm 17 - it defaults to backward simulation `'BSi'` although must be consistent across repeated calls of `bmm.update_particles`. Additionally, there is a `lag` parameter which defaults to 3 and controls the length of the particle history regenerated as each observation comes in as well as a `max_rejections` parameter which defaults to 20 and determines how many attempts are applied to the rejection sampler Section 3.2.3, setting `max_rejections=0` obtains the full $O(N^2)$ scheme for stitching (and backward simulation if `update='BSi'`).

4.6.4 Parameter Tuning

An additional optional argument to `offline_map_match`, `initiate_particles` and `update_particles` is `mm_model` which is a `bmm.MapMatchingModel` object and describes the form of the state-space model Section 4.1.2 via its methods and attributes.

A `MapMatchingModel` object contains methods to evaluate the transition density components from Equation (4.1). In particular, the `deviation_prior_evaluate` method evaluates the

$$\exp(-\beta|\|x_t - x_{t-1}\|_{e_t} - \|x_t - x_{t-1}\||)$$

component that penalises non-direct routes dependent on the parameter attribute `deviation_beta`. Whilst, the `distance_prior_evaluate` method is a placeholder for the $\gamma(\|x_t - x_{t-1}\|_{e_t})$ component which is not implemented in the raw `MapMatchingModel` object - similarly for its gradient `distance_prior_gradient`, truncation `d_max` and upper bound `distance_prior_bound`. Additionally there is a method for the observation density `likelihood_evaluate` which depends on the parameter `gps_sd`.

The default value for the `bmm.MapMatchingModel` object is `bmm.ExponentialMapMatchingModel`, i.e. the distance prior described in Equation (4.3)

$$\gamma(d_t) = p^0 \mathbb{I}(d_t = 0) + (1-p^0)\mathbb{I}(d_t > 0)\lambda \exp(-\lambda d_t),$$

where $d_t = \|x_t - x_{t-1}\|_{e_t}$, the distance along edges e_t between observation times.

To deal with variable time intervals we parameterise the stationary ($d_t = 0$) probability as

$$p^0 = \exp(-r^0 \Delta t),$$

for time interval (between observation times) Δt.

This leaves in total 4 model parameters

- `ExponentialMapMatchingModel.deviation_beta`
- `ExponentialMapMatchingModel.gps_sd`
- `ExponentialMapMatchingModel.zero_dist_prob_neg_exponent` (or r^0)
- `ExponentialMapMatchingModel.lambda_speed`

as well as `ExponentialMapMatchingModel.max_speed` which determines the (time interval dependent) truncation distance `ExponentialMapMatchingModel.d_max`.

These parameters can be tuned using expectation-maximisation Algorithm 18 over multiple polylines

```
mm_model = bmm.ExponentialMapMatchingModel()
optimised_params = bmm.offline_em(graph, mm_model, timestamps,
 ↪  polylines, save_path, n_ffbsi=100, n_iter=50)
```

where `polylines` is a list of (UTM) polylines accompanied by their corresponding `timestamps`. Indeed, the default values in `bmm.ExponentialMapMatchingModel` are taken from expectation-maximisation on the Porto taxi dataset Moreira-Matias et al. (2013) as in Figure 4.7.

Chapter 5

Ensemble Kalman Inversion for Generic Likelihoods

Ensemble Kalman inversion represents a powerful technique for inference in statistical models with likelihoods of the form $y \mid x \sim \mathbf{N}(y \mid \mathcal{H}(x), \mathrm{R})$ where the covariance R is known and $\mathcal{H}(x)$ is a potentially non-linear function that maps the unknown $x \in \mathbb{R}^{d_x}$ into observation space $\mathbb{R}^{d_y}$. In this chapter, we generalise the ensemble Kalman approach to generic likelihoods, $y \mid x \sim p(y \mid x)$ where the likelihood can be sampled from, but its density not necessarily evaluated. The generalisation is obtained by empirically estimating the noise covariance R in a manner that remains asymptotically unbiased in the fully linear Gaussian case. We examine the performance of generalised ensemble Kalman inversion for both optimisation and uncertainty quantification against state-of-the-art approximate Bayesian computation techniques from Section 2.5.

This chapter marks a departure from the sequential Bayesian inference setting of state-space models into the offline setting of static Bayesian inference, Figure 2.1, which we will build upon in Chapter 6. However, ensemble Kalman methods have originated from the field of high-dimensional state-space models - as will be described in the start of this chapter before proceeding to demonstrate their applicability in static Bayesian inference problems.

In Section 5.1 we briefly recount the origin of ensemble Kalman methods and their implementation within state-space models. Section 5.2 details the more recent application of ensemble Kalman techniques to static Bayesian inference problems where the likelihood has additive Gaussian noise before introducing our generalisation to generic likelihoods in Section 5.3. Section 5.4 investigates numerically the ensemble Kalman and approximate Bayesian computation techniques in two difficult static Bayesian inference problems with intractable likelihoods, before concluding in Section 5.5.

5.1 Ensemble Kalman Filter

Ensemble Kalman techniques were first introduced Evensen (1994) for online inference in state-space models, i.e. filtering. In the so-called update step of an *ensemble Kalman filter*, it is assumed that we have particles approximately distributed according to the predictive marginal or prior $p(x_T \mid y_{0:T-1})$ and seek particles distributed according to the filtering marginal or posterior $p(x_T \mid y_{0:T}) \propto p(y_T \mid x_T)p(x_T \mid y_{0:T-1})$. For online inference in state-space models, this update step is combined with a predictive step and iterated as more data is received. Ensemble Kalman techniques are asymptotically biased but have been shown to be numerically stable in a variety of settings.

5.1.1 Linear Gaussian State-space Models

Consider the special case of linear Gaussian state-space models, Section 2.6.1, where we have

$$p(x_T \mid y_{0:T-1}) = \mathbf{N}(x_T \mid \mu_{T|T-1}, \Sigma_{T|T-1}), \tag{5.1a}$$
$$p(y_T \mid x_T) = \mathbf{N}(y_T \mid \mathrm{H}_T x_T, \mathrm{R}_T), \tag{5.1b}$$

with each $x_t \in \mathbb{R}^{d_x}$ and $y_t \in \mathbb{R}^{d_y}$. Then the filtering marginal is analytically tractable

$$p(x_T \mid y_{0:T}) = \mathbf{N}(m_{T|T}, \Sigma_{T|T}),$$
$$m_{T|T} = \mu_{T|T-1} + \Sigma_{T|T-1}\mathrm{H}_T^{\mathrm{T}}\left(\mathrm{H}_T\Sigma_{T|T-1}\mathrm{H}_T^{\mathrm{T}} + \mathrm{R}_T\right)^{-1}(y_T - \mathrm{H}_T\mu_{T|T-1}), \tag{5.2a}$$
$$\Sigma_{T|T} = \Sigma_{T|T-1} - \Sigma_{T|T-1}\mathrm{H}_T^{\mathrm{T}}\left(\mathrm{H}_T\Sigma_{T|T-1}\mathrm{H}_T^{\mathrm{T}} + \mathrm{R}_T\right)^{-1}\mathrm{H}_T\Sigma_{T|T-1}. \tag{5.2b}$$

The above posterior comprises the update step of a classical Kalman Filter, Algorithm 10.

The original ensemble Kalman filter Evensen (1994) was derived to deal with the fact that in generic state-space models the predictive marginal $p(x_T \mid y_{0:T-1})$ is non-Gaussian and it's density cannot be evaluated. In this setting, we assume we have particles approximately distributed according to the prior $\{\hat{x}_T^{(i)}\}_{i=1}^N \sim p(x_T \mid y_{0:T-1})$ and can use the empirical covariance

$$\hat{\mathrm{C}}_T^{xx} = \frac{1}{N-1}\sum_{i=1}^{N}(\hat{x}_T^{(i)} - \bar{\hat{x}}_T)(\hat{x}_T^{(i)} - \bar{\hat{x}}_T)^{\mathrm{T}}$$

in place of $\Sigma_{T|T-1}$. Assuming the likelihood is still linear and Gaussian $p(y_T \mid x_T) = \mathbf{N}(y_T \mid \mathrm{H}_T x_T, \mathrm{R}_T)$ with H_T and R_T known, Evensen (1994) proposed applying the Kalman mean update, Equation (5.2a), to each particle

$$x_T^{(i)} = \hat{x}_T^{(i)} + \hat{\mathrm{C}}_T^{xx}\mathrm{H}_T^{\mathrm{T}}\left(\mathrm{H}_T\hat{\mathrm{C}}_T^{xx}\mathrm{H}_T^{\mathrm{T}} + \mathrm{R}_T\right)^{-1}(y - \mathrm{H}_T\hat{x}_T^{(i)}), \tag{5.3}$$

for $i = 1, \ldots, N$ introducing the original ensemble Kalman update.

It was then noted simultaneously by Burgers et al. (1998) and Houtekamer and Mitchell (1998) that the update in Equation (5.3) is asymptotically biased even in the Gaussian case and further that this can be corrected by the addition of a randomly generated perturbation

$$\begin{aligned} x_T^{(i)} &= \hat{x}_T^{(i)} + \hat{\mathrm{C}}_T^{xx}\mathrm{H}_T^{\mathrm{T}}\left(\mathrm{H}_T\hat{\mathrm{C}}_T^{xx}\mathrm{H}_T^{\mathrm{T}} + \mathrm{R}_T\right)^{-1}(y - \mathrm{H}_T\hat{x}_T^{(i)} - \eta_T^{(i)}), \\ \eta_T^{(i)} &\sim \mathbf{N}(\eta_T \mid 0, \mathrm{R}_T), \end{aligned} \tag{5.4}$$

for $i = 1, \ldots, N$. Therefore, if the prior is Gaussian $p(x_T \mid y_{0:T-1}) = \mathbf{N}(x_T \mid \mu_{T|T-1}, \Sigma_{T|T-1})$ we have $\hat{\mathrm{C}}_T^{xx} \to \Sigma_{T|T-1}$ as $N \to \infty$ and that under Equation (5.4) it can be shown, Le Gland et al. (2009), that

$$\begin{aligned} \mathbb{E}[x_T^{(i)}] &\to \mu_{T|T-1} + \Sigma_{T|T-1}\mathrm{H}_T^{\mathrm{T}}\left(\mathrm{H}_T\Sigma_{T|T-1}\mathrm{H}_T^{\mathrm{T}} + \mathrm{R}_T\right)^{-1}(y_T - \mathrm{H}_T\mu_{T|T-1}), \\ \mathrm{Cov}[x_T^{(i)}] &\to \Sigma_{T|T-1} + \Sigma_{T|T-1}\mathrm{H}_T^{\mathrm{T}}\left(\mathrm{H}_T\Sigma_{T|T-1}\mathrm{H}_T^{\mathrm{T}} + \mathrm{R}_T\right)^{-1}\mathrm{H}_T\Sigma_{T|T-1}, \end{aligned}$$

coinciding with Equation (5.2). Thus the ensemble Kalman filter update is asymptotically unbiased in the linear Gaussian case. We can ensure the above asymptotic statistics without adding any additional noise by instead deterministically shifting the particles, this is achieved by the popular *ensemble transform Kalman filter* Bishop et al. (2001) and *ensemble adjustment Kalman filter* Anderson (2001). Although in this work we will only consider stochastic ensemble Kalman updates.

These techniques are known to be asymptotically biased for general state-space models, i.e. non-Gaussian priors. There is however, a vast amount of empirical evidence demonstrating stability in a variety of complex non-linear state-space models with a very small number of particles, e.g. Houtekamer et al. (2005), Roth et al. (2017). In our view, this summarises the ensemble Kalman paradigm:

Asymptotically unbiased in the linear Gaussian case, otherwise biased but empirically stable. (5.5)

We also note that an important motivation for the ensemble Kalman filter is that in cases where $N << d_y$ (a requirement for many meteorological applications), the matrix inversion lemma can be applied to the update Equation (5.4) so that the complexity is $O(N^3)$ rather than $O(d_y^3)$.

Algorithm 19 Ensemble Kalman Filter

1: Given $\{x_{T-1}^{(i)}\}_{i=1}^{N}$

2: Predict

$$\hat{x}_T^{(i)} \sim p(x_T \mid x_{T-1}^{(i)}) \qquad i = 1\ldots,N$$

3: Generate perturbed observations

$$y_T^{(i)} \sim \mathbf{N}(y_T \mid \mathcal{H}_T(\hat{x}_T^{(i)}), \mathrm{R}_T) \qquad i = 1\ldots,N$$

4: Calculate covariances $\hat{\mathrm{C}}_T^{x\mathrm{H}}$ and $\hat{\mathrm{C}}_T^{\mathrm{HH}}$ - Equation (5.6)

5: Update

$$x_T^{(i)} = \hat{x}_T^{(i)} + \hat{\mathrm{C}}_T^{x\mathrm{H}} \left(\hat{\mathrm{C}}_T^{\mathrm{HH}} + \mathrm{R}_T\right)^{-1} (y_T - y_T^{(i)}) \qquad i = 1\ldots,N$$

6: **return** $\{x_T^{(i)}\}_{i=1}^{N}$

5.1.2 Non-Linear Gaussian Likelihoods

It was first noted in Houtekamer and Mitchell (2001) that an ensemble Kalman move can still be applied to likelihoods of the form

$$y_T \mid x_T \sim \mathbf{N}(y_T \mid \mathcal{H}_T(x_T), \mathrm{R}_T),$$

where $\mathcal{H}_T$ is now some non-linear function and R_T is still known. This can be done by noting that if the likelihood was linear and Gaussian, the matrices $\Sigma_{T|T-1}\mathrm{H}_T^{\mathrm{T}}$ and $\mathrm{H}_T\Sigma_{T|T-1}\mathrm{H}_T^{\mathrm{T}}$ could also be approximated empirically with

$$\hat{\mathrm{C}}_T^{x\mathrm{H}} = \frac{1}{N-1}\sum_{i=1}^{N}(\hat{x}_T^{(i)} - \hat{\bar{x}}_T)(\mathcal{H}_T(\hat{x}_T^{(i)}) - \mathcal{H}_T(\hat{\bar{x}}_T))^{\mathrm{T}}, \tag{5.6a}$$

$$\hat{\mathrm{C}}_T^{\mathrm{HH}} = \frac{1}{N-1}\sum_{i=1}^{N}(\mathcal{H}_T(\hat{x}_T^{(i)}) - \mathcal{H}_T(\hat{\bar{x}}_T))(\mathcal{H}_T(\hat{x}_T^{(i)}) - \mathcal{H}_T(\hat{\bar{x}}_T))^{\mathrm{T}}. \tag{5.6b}$$

As before we have that in the fully linear Gaussian case (5.1) that $\hat{\mathrm{C}}_T^{x\mathrm{H}} \to \Sigma_{T|T-1}\mathrm{H}_T^{\mathrm{T}}$ and $\hat{\mathrm{C}}_T^{\mathrm{HH}} \to \mathrm{H}_T\Sigma_{T|T-1}\mathrm{H}_T^{\mathrm{T}}$. Therefore the non-linear ensemble Kalman update

$$\begin{aligned} x_T^{(i)} &= \hat{x}_T^{(i)} + \hat{\mathrm{C}}_T^{x\mathrm{H}} \left(\hat{\mathrm{C}}_T^{\mathrm{HH}} + \mathrm{R}_T\right)^{-1} (y_T - \mathcal{H}_T(x_T^{(i)}) - \eta_T^{(i)}), \\ \eta_T^{(i)} &\sim \mathbf{N}(\eta_T \mid 0, \mathrm{R}_T), \end{aligned} \tag{5.7}$$

leads to particles $\{x_T^{(i)}\}_{i=1}^N$ that are asymptotically unbiased for the true posterior $p(x_T \mid y_{0:T})$ in the linear Gaussian case (5.1) but can be applied to state-space models with the only requirement being that the likelihood takes the form $p(y_T \mid x_T) = \mathbf{N}(y_T \mid \mathcal{H}_T(x_T), \mathrm{R}_T)$.

5.2 Ensemble Kalman Inversion

It was recognised in Iglesias et al. (2013) that the utility of the ensemble Kalman update can be extended outside of state-space models, in particular to static Bayesian inference problems of the form

$$x \sim p(x), \tag{5.8a}$$
$$y \sim p(y \mid x) = \mathbf{N}(y \mid \mathcal{H}(x), \mathrm{R}). \tag{5.8b}$$

Where we are asymptotically unbiased in the special case of Gaussian prior and linear Gaussian likelihood

$$p(x) = \mathbf{N}(x \mid m, \mathrm{Q}), \tag{5.9a}$$
$$p(y \mid x) = \mathbf{N}(y \mid \mathrm{H}x, \mathrm{R}), \tag{5.9b}$$
$$\implies p(x,y) = \mathbf{N}\left(\begin{pmatrix} m \\ \mathrm{H}m \end{pmatrix}, \begin{pmatrix} \mathrm{Q} & \mathrm{QH}^{\mathrm{T}} \\ \mathrm{HQ} & \mathrm{HQH}^{\mathrm{T}} + \mathrm{R} \end{pmatrix}\right), \tag{5.9c}$$

where the posterior is

$$p(x \mid y) = \mathcal{N}(m^y, \mathrm{Q}^y),$$
$$m^y = m + \mathrm{QH}^{\mathrm{T}} \left(\mathrm{HQH}^{\mathrm{T}} + \mathrm{R}\right)^{-1} (y - \mathrm{H}m), \tag{5.10a}$$
$$\mathrm{Q}^y = \mathrm{Q} - \mathrm{QH}^{\mathrm{T}} \left(\mathrm{HQH}^{\mathrm{T}} + \mathrm{R}\right)^{-1} \mathrm{HQ}. \tag{5.10b}$$

with $x \in \mathbb{R}^{d_x}$ and $y \in \mathbb{R}^{d_y}$. For the rest of the chapter we will focus on static Bayesian inference problems (rather than state-space models).

For non-Gaussian priors and non-linear Gaussian likelihoods (5.8), the limiting distribution of the particles from a single step ensemble Kalman update (5.7) may be quite different from the true posterior. However, as introduced in Iglesias et al. (2013), by iterating the ensemble Kalman update Equation (5.7) one can still generate a collection of informative particles. This idea was refined in Iglesias (2015) to be more in line with the tempered likelihood approach of SMC samplers Del Moral et al. (2006); Jasra et al. (2011), Chapter 6. This iterative ensemble

Kalman approach is termed *ensemble Kalman inversion* and a single iteration takes the form

$$\begin{aligned} x_\ell^{(i)} &= x_{\ell-1}^{(i)} + \mathrm{C}_{\ell-1}^{x\mathrm{H}} \left(\mathrm{C}_{\ell-1}^{\mathrm{HH}} + h_\ell^{-1}\mathrm{R}\right)^{-1} \left(y - \mathcal{H}(x_{\ell-1}^{(i)}) - \eta_\ell^{(i)}\right), \\ \eta_\ell^{(i)} &\sim \mathbf{N}(\eta_\ell \mid 0, h_\ell^{-1}\mathrm{R}). \end{aligned} \tag{5.11}$$

where $\mathrm{C}_{\ell-1}^{x\mathrm{H}}$ and $\mathrm{C}_{\ell-1}^{\mathrm{HH}}$ are as in Equation (5.6) applied to the particles $\{x_{\ell-1}^{(i)}\}_{i=1}^N$. The parameter h_ℓ^{-1} can be considered a stepsize parameter or equivalently h_ℓ as an incremental inverse temperature parameter. In the fully linear Gaussian case (5.9), the particles $\{x_\ell^{(i)}\}_{i=1}^N$ at each step are asymptotically unbiased for a tempered version of the posterior

$$\begin{aligned} p_\ell(x) &\propto p(x)p(y \mid x)^{\lambda_\ell} \\ &= \mathcal{N}(m_\ell, \mathrm{Q}_\ell), \\ m_\ell &= m + \mathrm{Q}\mathrm{H}^{\mathrm{T}} \left(\mathrm{H}\mathrm{Q}\mathrm{H}^{\mathrm{T}} + \lambda_\ell^{-1}\mathrm{R}\right)^{-1} (y - \mathrm{H}m), && \text{(5.12a)} \\ \mathrm{Q}_\ell &= \mathrm{Q} - \mathrm{Q}\mathrm{H}^{\mathrm{T}} \left(\mathrm{H}\mathrm{Q}\mathrm{H}^{\mathrm{T}} + \lambda_\ell^{-1}\mathrm{R}\right)^{-1} \mathrm{H}\mathrm{Q}, && \text{(5.12b)} \end{aligned}$$

where $\lambda_\ell = \sum_{r=1}^{\ell} h_r$ is the inverse temperature. The tempered posterior can also be defined iteratively

$$\begin{aligned} m_\ell &= m_{\ell-1} + \mathrm{Q}_{\ell-1}\mathrm{H}^{\mathrm{T}} \left(\mathrm{H}\mathrm{Q}_{\ell-1}\mathrm{H}^{\mathrm{T}} + h_\ell^{-1}\mathrm{R}\right)^{-1} (y - \mathrm{H}m_{\ell-1}), && \text{(5.13a)} \\ \mathrm{Q}_\ell &= \mathrm{Q}_{\ell-1} - \mathrm{Q}_{\ell-1}\mathrm{H}^{\mathrm{T}} \left(\mathrm{H}\mathrm{Q}_{\ell-1}\mathrm{H}^{\mathrm{T}} + h_\ell^{-1}\mathrm{R}\right)^{-1} \mathrm{H}\mathrm{Q}_{\ell-1}, && \text{(5.13b)} \end{aligned}$$

Thus, we can iterate L steps with stepsizes such that $\lambda_L = \sum_{r=1}^{L} h_r = 1$ and obtain particles $\{x_L^{(i)}\}_{i=1}^N$ that are asymptotically unbiased for the true posterior in the linear Gaussian case.

Iterating until $\lambda_L = 1$ is only one possible stopping criterion, another common approach is to adopt an optimisation style stopping criterion, for example Iglesias (2015) suggest stopping at iteration L when $(y - \bar{y}_L)^{\mathrm{T}}\mathrm{R}^{-1}(y - \bar{y}_L) < \tau$ for some stopping parameter τ.

5.3 Ensemble Kalman Inversion for Generic Likelihoods

In this section, we generalise ensemble Kalman inversion to the case where the data is generated from any likelihood $p(y \mid x)$ rather than the setting described in Sections 5.1 and 5.2 which is restricted to likelihoods of the form $\mathbf{N}(y \mid \mathcal{H}(x), \mathrm{R})$ where R is known. The resulting algorithm only requires samples from the prior $p(x)$ and the ability to simulate from the likelihood $p(y \mid x)$ for a given x, as in approximate Bayesian computation, Section 2.5. The final particle approximation is asymptotically biased for the true posterior with the exception of the fully linear Gaussian case (5.9).

5.3.1 Generic Likelihoods

In the more general case where we can only simulate $y \sim p(\cdot \mid x)$, we can no longer form the empirical covariance matrices in Equation (5.6) as we do not have access to the deterministic $\mathcal{H}$. Instead we can form

$$\mathrm{C}_\ell^{xx} = \frac{1}{N-1}\sum_{i=1}^{N}(x_\ell^{(i)} - \bar{x}_\ell)(x_\ell^{(i)} - \bar{x}_\ell)^{\mathrm{T}}, \tag{5.14a}$$

$$\mathrm{C}_\ell^{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_\ell^{(i)} - \bar{x}_\ell)(y_\ell^{(i)} - \bar{y}_\ell)^{\mathrm{T}}, \tag{5.14b}$$

$$\mathrm{C}_\ell^{yy} = \frac{1}{N-1}\sum_{i=1}^{N}(y_\ell^{(i)} - \bar{y}_\ell)(y_\ell^{(i)} - \bar{y}_\ell)^{\mathrm{T}}, \tag{5.14c}$$

and $\mathrm{C}_\ell^{yx} = \mathrm{C}_\ell^{xy\mathrm{T}}$ where each $y_\ell^{(i)} \sim p(\cdot \mid x_\ell^{(i)})$.

In the fully linear Gaussian case (5.9) or rather the tempered version (5.12, 5.13) we have $\mathrm{C}_\ell^{xy} \to \mathrm{Q}_\ell \mathrm{H}^T$ and $\mathrm{C}_\ell^{yy} \to \mathrm{H}\mathrm{Q}_\ell \mathrm{H}^{\mathrm{T}} + \mathrm{R}$ as $N \to \infty$. We now note that we can also calculate

$$\mathrm{C}_\ell^{y|x} = \mathrm{C}_\ell^{yy} - \mathrm{C}_\ell^{yx}\mathrm{C}_\ell^{xx\,-1}\mathrm{C}_\ell^{xy}, \tag{5.15}$$

and that in the linear Gaussian case

$$\begin{aligned}\mathrm{C}_\ell^{y|x} &\to (\mathrm{H}\mathrm{Q}_{\ell-1}\mathrm{H}^{\mathrm{T}} + \mathrm{R}) - (\mathrm{H}\mathrm{Q}_{\ell-1})\mathrm{Q}_{\ell-1}^{-1}(\mathrm{Q}_{\ell-1}\mathrm{H}^{\mathrm{T}}),\\ &= \mathrm{R}.\end{aligned}$$

Thus we can use the particles to approximate the covariance of the likelihood empirically.

We now consider the tempered ensemble Kalman iteration (5.11) but with the following adjustment

$$\begin{aligned} x_\ell^{(i)} &= x_{\ell-1}^{(i)} + \mathrm{C}_{\ell-1}^{xy}\left(\mathrm{C}_{\ell-1}^{yy} + (h_\ell^{-1}-1)\mathrm{C}_{\ell-1}^{y|x}\right)^{-1}\left(y - y_{\ell-1}^{(i)} - \eta_\ell^{(i)}\right),\\ y_{\ell-1}^{(i)} &\sim p(y_{\ell-1} \mid x_{\ell-1}^{(i)}),\\ \eta_\ell^{(i)} &\sim \mathbf{N}(\eta \mid 0, (h_\ell^{-1}-1)\mathrm{C}_{\ell-1}^{y|x}).\end{aligned} \tag{5.16}$$

In the linear Gaussian case and large sample limit the above ensemble Kalman move becomes

$$\begin{aligned} x_\ell^{(i)} &= x_{\ell-1}^{(i)} + \mathrm{Q}_{\ell-1}\mathrm{H}^{\mathrm{T}}\left((\mathrm{H}\mathrm{Q}_{\ell-1}\mathrm{H}^{\mathrm{T}} + \mathrm{R}) + (h_\ell^{-1}-1)\mathrm{R}\right)^{-1}(y - y_{\ell-1}^{(i)} - \eta_\ell^{(i)}), \\ y_{\ell-1}^{(i)} &\sim \mathbf{N}(y_{\ell-1} \mid \mathrm{H}x_{\ell-1}^{(i)}, \mathrm{R}), \\ \eta_\ell^{(i)} &\sim \mathbf{N}(\eta_\ell \mid 0, (h_\ell^{-1}-1)\mathrm{R}). \end{aligned}$$

Where the $(h_\ell^{-1}-1)$ scaling has been chosen to ensure that

$$\begin{aligned} \mathbb{E}[x_\ell^{(i)}] &= m_{\ell-1} + \mathrm{Q}_{\ell-1}\mathrm{H}^{\mathrm{T}}\left(\mathrm{H}\mathrm{Q}_{\ell-1}\mathrm{H}^{\mathrm{T}} + h_\ell^{-1}\mathrm{R}\right)^{-1}(y - \mathrm{H}m_{\ell-1}), \\ \mathrm{Cov}[x_\ell^{(i)}] &= \mathrm{Q}_{\ell-1} - \mathrm{Q}_{\ell-1}\mathrm{H}^{\mathrm{T}}\left(\mathrm{H}\mathrm{Q}_{\ell-1}\mathrm{H}^{\mathrm{T}} + h_\ell^{-1}\mathrm{R}\right)^{-1}\mathrm{H}\mathrm{Q}_{\ell-1}. \end{aligned}$$

These limiting statistics coincide with Equation (5.13) and therefore the ensemble Kalman update in Equation (5.16) is asymptotically unbiased in the linear Gaussian case, and remains faithful to the ensemble Kalman paradigm 5.5.

5.3.2 Stepsize Selection

The motivation of iterative ensemble Kalman inversion is that for difficult non-Gaussian problems, moving directly from prior to posterior in one step is an extremely difficult task, by taking many smaller steps the particles can explore the state-space and have a better chance of settling in regions of high posterior probability. There is, therefore, a trade-off to be made - more steps means greater exploration but also more likelihood simulations and a longer runtime.

The stepsizes equivalently define an inverse temperature schedule

$$0 = \lambda_0 < \lambda_1 < \cdots < \lambda_L$$

where $h_\ell = \lambda_\ell - \lambda_{\ell-1} > 0$ for $\ell = 1, \ldots, L$.

In SMC samplers Del Moral et al. (2006) it is common for the next inverse temperature (and therefore stepsize) to be selected adaptively Jasra et al. (2011) such that the effective sample size at each iteration decreases by a fixed amount, i.e. select λ_ℓ such that $\mathrm{ESS}(\{w_\ell^{(i)}\}_{i=1}^N) \approx \rho N$, where the normalised weights are a function of λ_ℓ and $\rho \in (0,1)$ is a tuning parameter that controls the size of the steps. The root in λ_ℓ can be found using a bisection algorithm in $(\lambda_{\ell-1}, \lambda_L]$ and requires no additional likelihood evaluations. This idea was ported to ensemble Kalman inversion in Iglesias et al. (2018), as ensemble Kalman inversion does not compute

Algorithm 20 Ensemble Kalman Inversion for Generic Likelihoods

Given (possibly adaptive) methods to set inverse temperatures λ_ℓ and stopping criterion L
Sample from prior

$$x_0^{(i)} \sim p(x) \qquad i = 1, \dots, N$$

for $\ell = 1, \dots, L$ **do**
    Simulate observations

$$y_{\ell-1}^{(i)} \sim p(y \mid x_{\ell-1}^{(i)}) \qquad i = 1, \dots, N$$

    Form sample covariances

$$\begin{aligned}
\mathrm{C}_{\ell-1}^{xx} &= \frac{1}{N-1} \sum_{i=1}^{N} (x_{\ell-1}^{(i)} - \bar{x}_{\ell-1})(x_{\ell-1}^{(i)} - \bar{x}_{\ell-1})^{\mathrm{T}} \\
\mathrm{C}_{\ell-1}^{xy} &= \frac{1}{N-1} \sum_{i=1}^{N} (x_{\ell-1}^{(i)} - \bar{x}_{\ell-1})(y_{\ell-1}^{(i)} - \bar{y}_{\ell-1})^{\mathrm{T}} \\
\mathrm{C}_{\ell-1}^{yy} &= \frac{1}{N-1} \sum_{i=1}^{N} (y_{\ell-1}^{(i)} - \bar{y}_{\ell-1})(y_{\ell-1}^{(i)} - \bar{y}_{\ell-1})^{\mathrm{T}} \\
\mathrm{C}_{\ell-1}^{y|x} &= \mathrm{C}_{\ell-1}^{yy} - \mathrm{C}_{\ell-1}^{yx} \mathrm{C}_{\ell-1}^{xx\,-1} \mathrm{C}_{\ell-1}^{xy}
\end{aligned}$$

    where $\mathrm{C}_{\ell-1}^{yx} = \mathrm{C}_{\ell-1}^{xy\mathrm{T}}$
    Set stepsize $h_\ell = \lambda_\ell - \lambda_{\ell-1}$
    for $i = 1, \dots, N$ **do**
        Generate observation perturbations

$$\eta_\ell^{(i)} \sim \mathcal{N}\left(0, \left(h_\ell^{-1} - 1\right) \mathrm{C}_{\ell-1}^{y|x}\right)$$

        Move particles

$$x_\ell^{(i)} = x_{\ell-1}^{(i)} + \mathrm{C}_{\ell-1}^{xy} \left(\mathrm{C}_{\ell-1}^{yy} + (h_\ell^{-1} - 1)\mathrm{C}_{\ell-1}^{y|x}\right)^{-1} \left(y - y_{\ell-1}^{(i)} - \eta_\ell^{(i)}\right)$$

return $\{x_L^{(i)}\}_{i=1}^N$

importance weights, the following psuedo-weights are used

$$\hat{w}_\ell^{(i)} \propto \exp\left(-\frac{1}{2}(\lambda_\ell - \lambda_{\ell-1})\left(y - \mathcal{H}(x_{\ell-1}^{(i)})\right)^{\mathrm{T}} \mathrm{R}^{-1}\left(y - \mathcal{H}(x_{\ell-1}^{(i)})\right)\right).$$

The effective sample size (over true importance weights) can be viewed as a measure of the χ^2-divergence between the tempered posterior at $\lambda_{\ell-1}$ and λ_ℓ as described in Chopin and Papaspiliopoulos (2020), Section 2.3.2. In Iglesias and Yang (2020) this idea was extended to instead use an effective sample size based on the symmetric KL-divergence.

In this work, we adopt the more conventional χ^2 approach with adapted pseudo-weights

$$\hat{w}_\ell^{(i)} \propto \exp\left(-\frac{1}{2}(\lambda_\ell - \lambda_{\ell-1})\left(y - y_{\ell-1}^{(i)}\right)^{\mathrm{T}} \mathrm{C}_{\ell-1}^{y|x-1}\left(y - y_{\ell-1}^{(i)}\right)\right). \tag{5.17}$$

As noted in Iglesias et al. (2018), the lack of resampling means the user can be more aggressive with the choice of ρ, in our experiments we set $\rho = \frac{1}{2}$.

5.3.3 Stopping Criteria

We consider two stopping criteria:

- **Sampling** - stop when $\lambda_L = 1$. This stopping criterion mimics that of tempered likelihood approaches for tractable likelihoods and is applied to ensemble Kalman inversion in Iglesias et al. (2018). This approach is asymptotically unbiased for the true posterior in the linear Gaussian case and aims to quantify uncertainty around parameters.

- **Optimisation** - stop when all marginal standard deviations of the particles fall below 0.1 of their initial or prior standard deviation. Under this approach the algorithm is iterated until the particles form a consensus approaching a single point estimate, which in the linear Gaussian case will (asymptotically) be the maximum likelihood estimator.

5.4 Numerical Experiments

We now examine the performance of both ensemble Kalman inversion for sampling and optimisation versus two approximate Bayesian computation methods for two static Bayesian inference problems with intractable likelihoods.

The first approximate Bayesian computation technique is that of ABC-SMC, Section 2.5.3. We follow Del Moral et al. (2012) in designing the proposal distribution, the threshold schedule

$\{\kappa_\ell\}_{\ell=0}^L$, resampling criterion and stopping criterion. The proposal distribution is the random-walk Metropolis-Hastings kernel from Section 2.5.3 with proposal covariance $\varepsilon \mathrm{D} = \frac{2.38^2}{d} \mathrm{C}_{\ell-1}^{xx}$ a scaled version of the empirical covariance of the previous particles. Each threshold is chosen adaptively such that ESS$\approx 0.9N$. Particles are resampled when the ESS drops below $0.5N$. Finally, the algorithm terminates when the acceptance rate of the Metropolis-Hastings kernel first falls below 1.5%.

We also run the random-walk Metropolis-Hastings kernel directly as an ABC-MCMC algorithm. As described in Section 2.5.2, Vihola and Franks (2020) we use a Robbins-Monro schedule to adaptively tune the stepsize, preconditioner combination to $\frac{2.38^2}{d}\hat{\Sigma}$ where $\hat{\Sigma}$ is the empirical covariance of the historical chain and adapt the threshold parameter such that 10% of samples are accepted. We note that for ABC-MCMC there is additional work (not considered in these simulations) based on post-hoc regression adjustments Beaumont et al. (2002) that resemble the ensemble Kalman update Nott et al. (2012). These methods however utilise no tempering and are applied on top of an ABC-MCMC sample - they are therefore, in our view, somewhat removed from the ensemble Kalman paradigm.

As mentioned, for EKI we determine the stepsize h_ℓ (equivalently the next inverse temperature λ_ℓ) adaptively such that the ESS of the pseudo-weights, Equation (5.17), is approximately $0.5N$. We examine both the sampling and optimisation stopping criteria discussed in Section 5.3.3.

5.4.1 g-and-k Distribution

Popular as a benchmark for ABC algorithms the g-and-k distribution family is defined by the quantile function

$$F^{-1}(u) = A + B\left(1 + c\frac{1-\exp(-gz(u))}{1+\exp(-gz(u))}\right)(1+z(u)^2)^k z(u), \tag{5.18}$$

where $z(\cdot)$ is the quantile function of a standard normal distribution. The g-and-k family represent a class of distributions with flexible skewness and kurtosis features useful for modelling financial returns Drovandi and Pettitt (2011) and life insurance Peters et al. (2016). The constant c is typically set to 0.8 and considered known. We set the remaining parameters $x = (A, B, g, k)$ with *true* values $(3, 1, 2, 1/2)$ but consider them unknown (to be inferred) with prior $\mathbf{U}(\cdot \mid 0, 10)^4$. The g-and-k likelihood is defined implicitly by the quantile function (5.18). This likelihood function cannot be evaluated easily and thus we cannot readily utilise traditional posterior inference methods such as MCMC or importance sampling - although their ABC

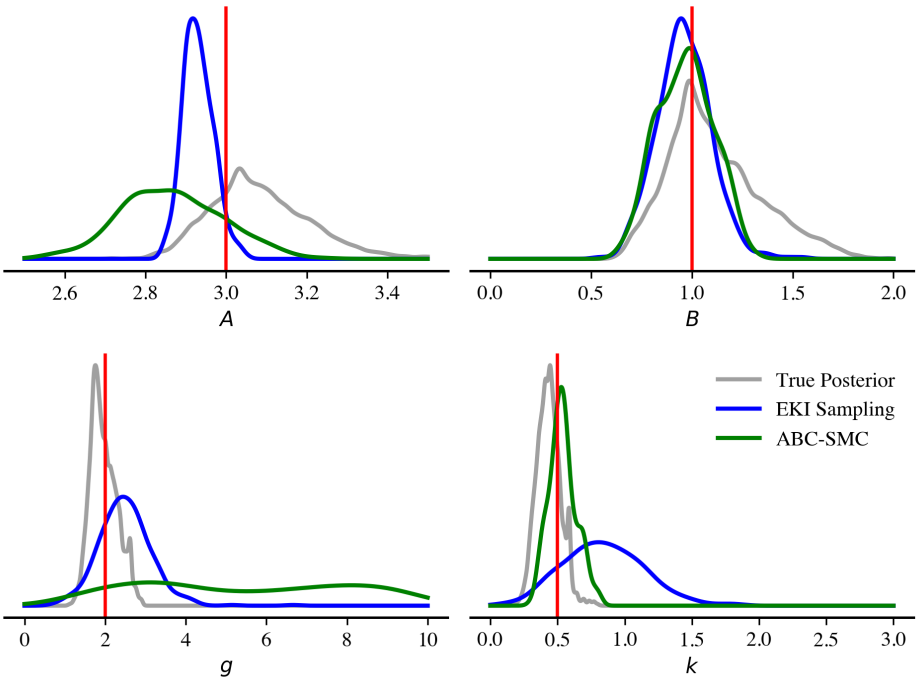


Fig. 5.1 Marginal densities for $N = 500$ on g-and-k distribution. Underlying simulation values in red.

counterparts remain possible as the likelihood can be easily simulated for given parameters by simply evaluating the quantile function (5.18) at a sampled standard uniform random variable.

We assume we have data of 1000 i.i.d. samples from Equation (5.18) which we summarise into 100 evenly spaced order statistics for both EKI and ABC. For ABC we use the Euclidean distance function. For both ABC and EKI we unconstrain the parameters using the transformation $z(\cdot/10)$.

In Figure 5.1, we compare the marginal distributions produced by EKI for sampling and those from ABC-SMC. We observe that EKI has centred on the vicinity of the truth for all 4 parameters whereas ABC-SMC has failed to concentrate in the g variable in particular. We note that the two posteriors differ quite significantly - an indication of the high levels of non-linearity in the g-and-k likelihood - yet the EKI posterior remains informative. We also display the *true* posterior obtained from a long run (10^5 samples) of classical random-walk Metropolis-Hastings where the density function is evaluated using numerical inversion and numerical differentiation Rayner and MacGillivray (2002). We observe relative agreement between all three approaches in the B parameter but intriguing discrepancies in the remaining dimensions.

We then examine the second EKI stopping criterion, that of optimisation. The EKI optimisation procedure, Section 5.3.3, iterates beyond $\lambda_\ell = 1$ until a consensus is reached on a single point for an approximate maximum likelihood estimator. Figure 5.2 displays seven equally

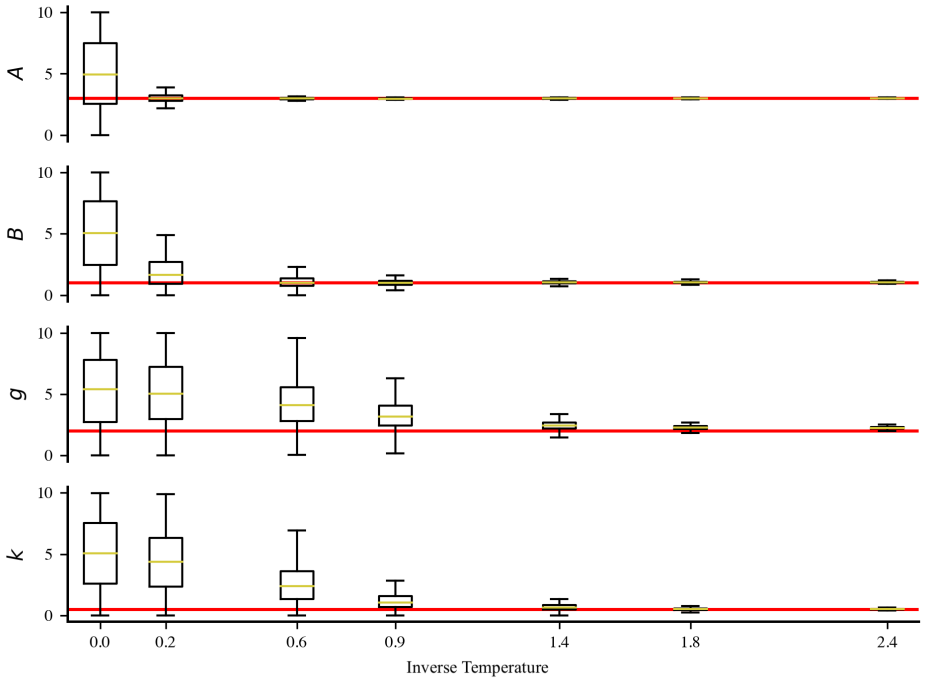


Fig. 5.2 Convergence of EKI for optimisation, $N = 500$, on g-and-k distribution. Underlying simulation values in red.

spaced iterations as the adaptive inverse temperature schedule increases above $\lambda_\ell = 1$ before terminating when all standard deviations are suitably small. The particles converge quickly and very closely to the true underlying parameter values.

Finally, we vary the sample size N between 200 and 5000 then repeat the experiment 10 times. The root mean squared error is defined as

$$\text{RMSE}(\{x^{(i)}\}_{i=1}^N = \sqrt{\frac{1}{Nd_x}\sum_{i=1}^{N}\sum_{m=1}^{d_x}(x^{(i)}[m] - x^\dagger[m])^2},$$

where $x[m]$ is the mth dimension of x and $x^\dagger$ is the true underlying parameter. We plot the root mean squared error against the number of likelihood simulations utilised by each algorithm as N changes in Figure 5.3. We observe that both EKI stopping criteria are consistently outperforming both ABC-SMC and ABC-MCMC for the same computational cost. Curiously, we observe that the performance of EKI for sampling deteriorates as N increases although this is not the case for the optimisation approach. We posit that this might be due to the smaller sample sizes over estimating the noise covariance and therefore moving the particles further - this suggests that stopping when the inverse temperature $\lambda_L = 1$ is suboptimal for this example

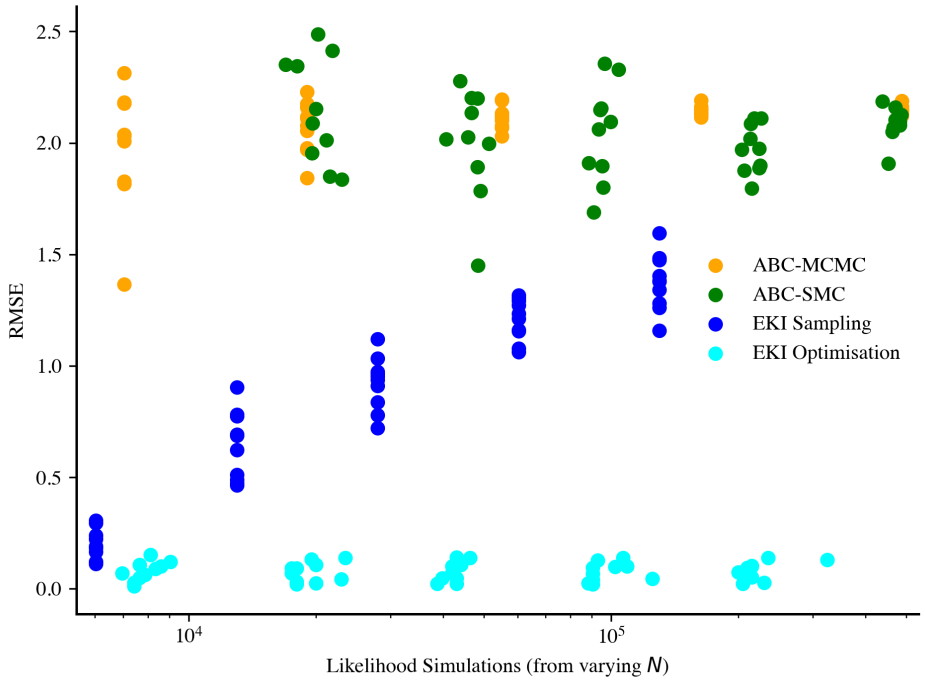


Fig. 5.3 Root mean squared error for number of likelihood simulation induced by varying N, on g-and-k distribution. Repeated over 10 randomly generated sets of observations.

and that the induced EKI posterior for inverse temperatures larger than $\lambda_\ell = 1$ are closer to the true posterior.

5.4.2 Stochastic Lorenz 96

We now consider the task of inferring the initial conditions of a noisy version of the Lorenz 96 model Lorenz (1995). The Lorenz 96 (from 1995) is a simplified model of oceanic flows that is commonly used as a testbed for high-dimensional data assimilation techniques such as the ensemble Kalman filter.

The Lorenz 96 dynamics (with added stochasticity) are defined by the SDE

$$dx_t[m] = -x_t[m-2]x_t[m-1]dt + x_t[m-1]x_t[m+1]dt - x_t[m]dt + Fdt + dW_t,$$

for $m = 1,\dots,d_x$ with cyclic coordinates $x_t[0] = x_t[d_x]$, $x_t[-1] = x_t[d_x - 1]$ and $x_t[d_x + 1] = x_t[1]$. We adopt the common high-dimensional setup of $d_x = 40$ and $F = 8$, which is known to produce challenging, chaotic dynamics.

We define our inference goal as obtaining the initial conditions $x_0 \in \mathbb{R}^{d_x}$ given observations of every other dimension, perturbed by Gaussian noise with variance 0.1, at times $t = 1,2,3,4,5$

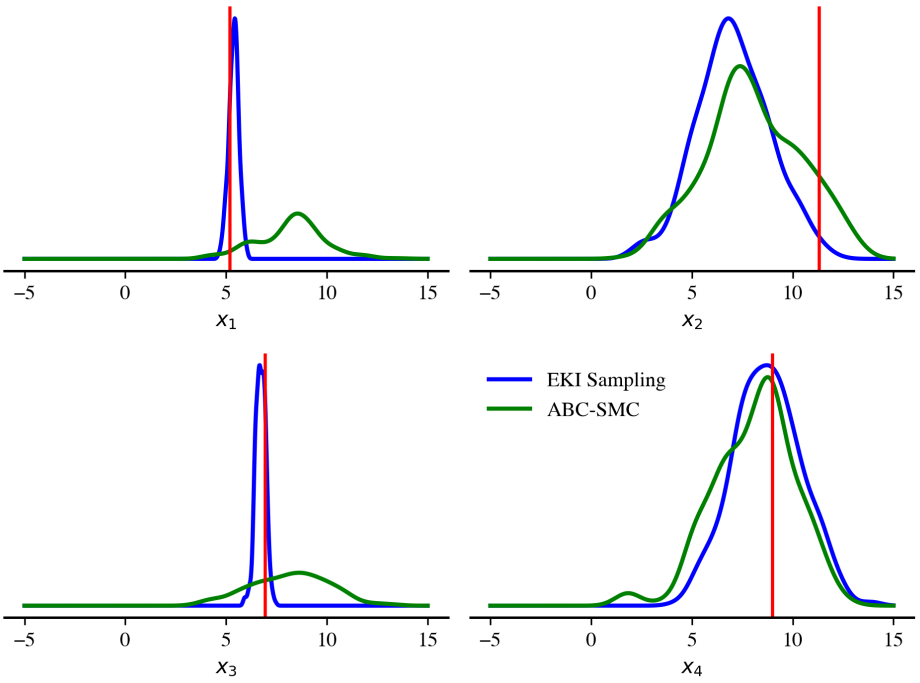


Fig. 5.4 Marginal densities for $N = 500$ on Lorenz 96 example. Underlying simulation values in red.

- resulting in $d_y = 100$ observations. We set the prior to $p(x_0) = \mathbf{N}(x_0 \mid F, 5\mathbb{I}_{40})$ and simulate from the likelihood with an Euler-Maruyama scheme (stepsize 0.001). The stochasticity in the Lorenz 96 dynamics provides a more realistic influence of noise but the stochastic Euler-Maruyama scheme makes evaluating the likelihood density intractable.

In our experiments, underlying true values for the initial conditions are sampled from the prior and then observations generated using the same Euler-Maruyama scheme as above.

Recall that odd dimensions are directly observed whereas even dimensions are unobserved. We can see in Figure 5.4 that EKI for sampling converges very closely around the truth in observed dimensions and is understandably less certain about unobserved dimensions. In contrast ABC-SMC, performs similarly for all dimensions and is likely struggling with the dimensionality of the problem.

When we push the EKI into high inverse temperatures in Figure 5.5 for optimisation, we first see that the particles struggle to collapse in the dimensions without observations. This is an indication that the given data is insufficient to provide a confident point estimate in those unobserved coordinates. We also notice that the particles converge in the second dimension away from the true underlying value of the parameter - this may imply that the true maximum likelihood is not necessarily guaranteed to be close to the truth (in unobserved dimensions) under this model setup.

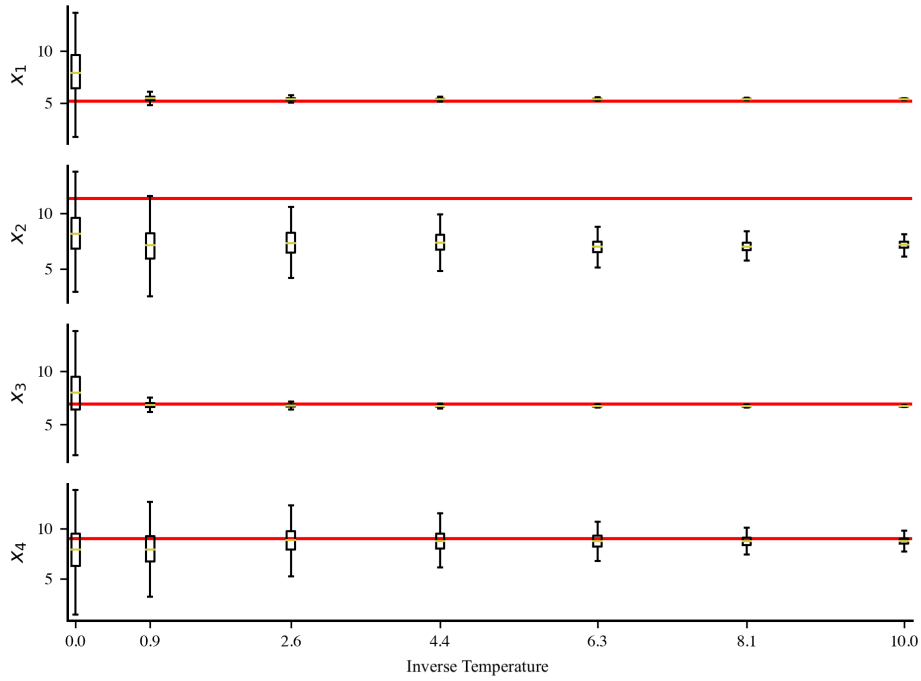


Fig. 5.5 Convergence of EKI for optimisation, $N = 500$, on Lorenz 96 example. Underlying simulation values in red.

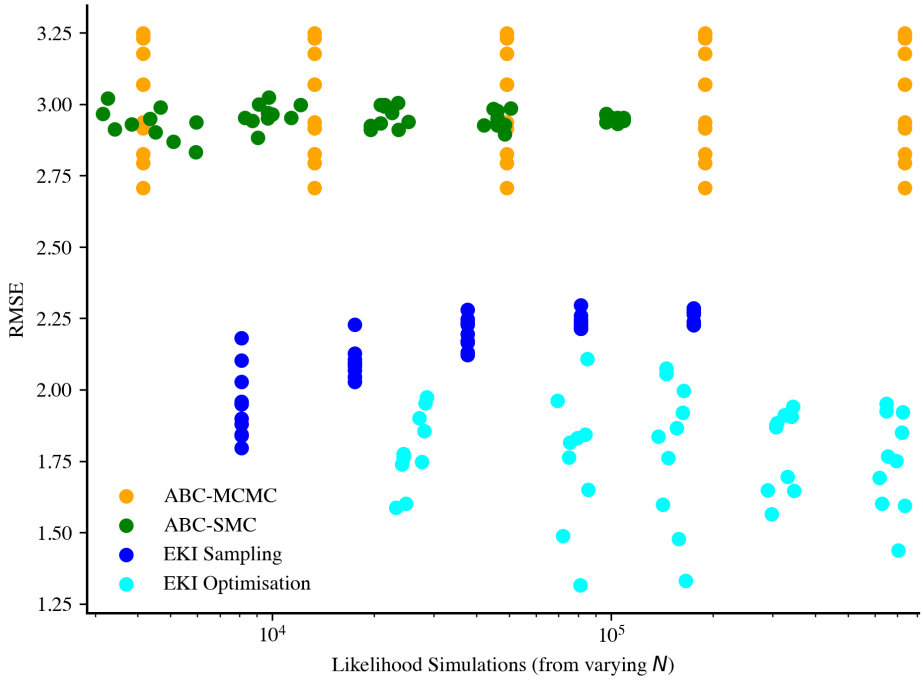


Fig. 5.6 Root mean squared error for number of likelihood simulation induced by varying N, on Lorenz 96 example. Repeated over 10 randomly generated sets of observations.

In Figure 5.6, we also observe the phenomenon of decreasing performance in EKI for sampling as N increases for the L96 example - although less so than the g-and-k example. However, again we see that EKI consistently outperforms ABC, although this is perhaps not surprising in a high-dimensional example given the regular use of ensemble Kalman techniques in this type of problem. In this case, the EKI for optimisation utilised significantly more likelihood simulations as it requires more iterations to converge - it also suffered high variance results whereas the performance of the EKI for sampling was more steady.

5.5 Discussion

In this chapter, we have extended the work of ensemble Kalman inversion Iglesias et al. (2018, 2013) to problems with generic likelihoods as opposed to the common restriction of additive Gaussian noise. We remain faithful to the ensemble Kalman paradigm, i.e. our generalisation is asymptotically unbiased in the linear Gaussian case. We described how to apply the technique for both optimisation and for sampling (uncertainty quantification). We have demonstrated both speed and accuracy of the novel ensemble Kalman inversion algorithm in a difficult benchmark problem as well as a high dimensional spatial example. The computational cost of the ensemble Kalman inversion is $O(Ld_y^3 + LNd_y^2)$ and only requires LN likelihood simulations which is the typical computational bottleneck for problems within approximate Bayesian computation.

We observe the curious phenomenon that increasing the number of particles fails to increase accuracy when applying ensemble Kalman inversion for sampling - at least in terms of root mean square error. An outstanding question is how to correct for this. This phenomenon is not entirely new but is not well understood, it would be intriguing to investigate whether it could potentially be mitigated by covariance regularisation Houtekamer and Mitchell (2001) or moment-matching ideas Lei and Bickel (2011).

A natural extension of the stochastic ensemble Kalman inversion algorithm presented in this chapter would be the conversion to the square-root ensemble Kalman variants Anderson (2001); Bishop et al. (2001) that instead deterministically move particles in a way that remains asymptotically unbiased for fully linear Gaussian problems. By removing a layer of stochasticity we hope to further increase the numerical stability of the inversion algorithm.

Outside of linear Gaussian problems, ensemble Kalman inversion is asymptotically biased. It would interesting to investigate embedding an ensemble Kalman kernel within an ABC-SMC sampler. That is, with importance weights from Section 2.5.3

$$w_\ell \propto w_{\ell-1} \frac{p(x_\ell)\nu_{\ell-1|\ell}(x_{\ell-1}|x_\ell, y_\ell)}{p(x_{\ell-1})q_{\ell|\ell-1}(x_\ell|x_{\ell-1}, y_{\ell-1})} \mathbb{I}(d(y_\ell, y) < \kappa_\ell).$$

Where now $q_{\ell|\ell-1}$ is an ensemble Kalman move and $\nu_{\ell-1|\ell}$ is some backward kernel. This way, the ensemble Kalman inversion would inherit the theory of approximate Bayesian computation and become asymptotically unbiased for ν_κ. However, it is not clear how to choose the backward kernel to induce stable importance weights.

A further application of the presented ensemble Kalman inversion would be its use within state-space models. It is possible that the use of an iterative tempered approach within a single update step of an ensemble Kalman filter may improve sample quality. Additionally, we could adapt ensemble Kalman inversion to be used for state-space models with intractable observation densities Jasra et al. (2012).

Chapter 6

Quasi-Newton Sequential Monte Carlo

In this chapter, we investigate the use of a Hessian approximation to accelerate Sequential Monte Carlo for static Bayesian inference tasks. The Hessian matrix represents a natural pre-conditioner, capturing the local scaling and correlations of the target distribution, however calculating the Hessian matrix exactly is computationally expensive. We instead adapt a quasi-Newton method, ubiquitous in optimisation literature, to convert gradient evaluations from a particle's historical trajectory into a computationally cheap, positive definite projection of the Hessian matrix.

Sequential Monte Carlo for static Bayesian inference problems with tempered intermediate distributions is reviewed in Section 6.1. In Section 6.2 we detail a very general approach to incorporating gradient information via a SMC transition kernel. We then describe the L-BFGS Hessian approximation and its implementation within the transition kernel in Section 6.3. Section 6.4 presents two challenging numerical experiments, one high dimensional toy example with difficult scaling and one multi-modal with real data. Section 6.5 provides discussion and extensions.

6.1 Likelihood Tempering

Sequential Monte Carlo requires a sequence of target distributions $\pi_{0:t}(x_{0:t})$, $t = 0, \ldots, T$. For static Bayesian inference problems, the sequence is typically defined artificially such that the marginal distribution of particles at the final iteration is the posterior distribution $\pi_T(x) = p(x \mid y) \propto p(x)p(y \mid x)$.

In this work we will consider the case of *likelihood tempering* Gelman and Meng (1998); Neal (2001) where we fix the intermediate marginal distributions

$$\begin{aligned}\pi_t(x) &\propto p(x)p(y \mid x)^{\lambda_t}, \\ &\propto \exp(-U_t(x)),\end{aligned}$$

where $U_t(x) = -\log p(x) - \lambda_t \log p(x \mid y)$ is the *tempered potential* and λ_t is an increasing *inverse temperature* parameter

$$0 \leq \lambda_0 < \lambda_1 < \cdots < \lambda_T = 1. \tag{6.1}$$

Under this construction the sequence 'smoothly' transitions from the prior (which we assume we can sample from) to the posterior.

Alternative intermediate targets are possible. For example, in the case where we receive T observations $(y_1, \ldots, y_T)$ we can set batched intermediate targets $\pi_t(x_t) = p(x_t \mid y_1, \ldots, y_t)$ as in Chopin (2002). However, this strategy does not necessarily induce smooth transitions between the intermediate target distributions.

The likelihood tempering approach inspired much of Chapter 5 and although not investigated here it would be possible to consider similar alternative stopping criteria that extend to $\lambda_T > 1$ with the goal of optimisation rather than integration.

6.1.1 Sequential Importance Weights

Having fixed the marginal targets $\pi_t(x_t)$, we can assume we have weighted particles at time $t-1$ approximating $\pi_{t-1}(x_{t-1})$. Suppose we use a Markovian transition kernel

$$q_{t|0:t-1}(x_t \mid x_{0:t-1}) = q_{t|t-1}(x_t \mid x_{t-1}),$$

then we can write the sequential weights as

$$\begin{aligned}w_t &= \frac{\pi_t(x_t)\pi_{t-1|t}(x_{t-1} \mid x_t)}{q_{t-1}(x_{t-1})q_{t|t-1}(x_t \mid x_{t-1})}, \\ &= \frac{\pi_{t-1}(x_{t-1})}{q_{t-1}(x_{t-1})} \frac{\pi_t(x_t)\pi_{t-1|t}(x_{t-1} \mid x_t)}{\pi_{t-1}(x_{t-1})q_{t|t-1}(x_t \mid x_{t-1})}, \\ &= w_{t-1} \frac{\pi_t(x_t)\pi_{t-1|t}(x_{t-1} \mid x_t)}{\pi_{t-1}(x_{t-1})q_{t|t-1}(x_t \mid x_{t-1})}.\end{aligned}$$

where $\pi_{t-1|t}(x_{t-1} \mid x_t)$ is a normalised backward kernel which we are free to choose.

Further suppose that the Markovian transition kernel is π_{t-1}-invariant

$$\int q_{t|t-1}(x_t \mid x_{t-1})\pi_{t-1}(x_{t-1})dx_{t-1} = \pi_{t-1}(x_t).$$

Then the natural (but suboptimal) choice of $\pi_{t-1|t}(x_{t-1} \mid x_t)$, Section 2.4.3, Del Moral et al. (2006), is

$$\pi_{t-1|t}(x_{0:t-1}|x_t) = \frac{\pi_{t-1}(x_{t-1})q_{t|t-1}(x_t|x_{t-1})}{\pi_{t-1}(x_t)},$$

which induces sequential importance weights of the form

$$w_t = w_{t-1}\frac{\pi_t(x_t)}{\pi_{t-1}(x_t)}. \tag{6.2}$$

6.1.2 Adaptive Tempering

Combining the sequential importance weights induced from a π_{t-1}-invariant, Markovian transition kernel with likelihood tempered intermediate distributions gives weights

$$w_t = w_{t-1}p(y \mid x_t)^{\lambda_t - \lambda_{t-1}}. \tag{6.3}$$

A major advantage of this formulation is that the weights w_t can be evaluated as a function of λ_t without any further likelihood evaluations, Jasra et al. (2011), Section 5.3.2. Thus, we can define the inverse temperature schedule adaptively by using a numerical root finder (i.e. bisection) at each iteration to solve

$$\mathrm{ESS}(\lambda_t) \approx \rho N.$$

For $\lambda_t \in (\lambda_{t-1}, 1]$ where ρ is a design parameter controlling the χ^2-distance between π_{t-1} and π_t and therefore the number of iterations, T, required to reach the posterior $\pi_T(x) = p(x \mid y)$. As in Equation (2.12), we define the effective sample size as

$$1 \leq \mathrm{ESS}(\lambda_t) = \frac{\left(\sum_{i=1}^N w_t^{(i)}(\lambda_t)\right)^2}{\sum_{i=1}^N w_t^{(i)}(\lambda_t)^2} \leq N.$$

6.2 Langevin Kernel

We now turn to the choice of transition kernel $q_{t|t-1}(x_t \mid x_{t-1})$. Assuming we have access to gradients $\nabla U_t(x)$ we can adopt the general gradient informed proposal of Horowitz (1991) before applying an accept-reject step to ensure π_{t-1}-reversibility.

The proposal of Horowitz (1991), sometimes referred to as Hamiltonian Monte Carlo with persistent momenta, adopts the common approach of augmenting the state-space with an auxiliary momenta variable, v. As described in Section 2.4.2, we then extend a generic target distribution $\pi(x) \propto \exp(-U(x))$ to be Gaussian in v

$$U(x,v) = U(x) + \frac{1}{2} v^{\mathrm{T}} \mathrm{M}^{-1} v.$$

In continuous time, Langevin dynamics (synonymously underdamped Langevin dynamics) can be written

$$\begin{aligned} dx_t &= \mathrm{M}^{-1} v_t dt, \\ dv_t &= -\nabla U(x_t) dt - \gamma \mathrm{M}^{-1} v_t dt + \sqrt{2\gamma} dW_t, \end{aligned}$$

where γ is a friction parameter controlling the diffusive nature of the momenta. Continuous time invariance for $\pi(x,v) \propto \exp(-U(x,v))$ can be checked with Equation (2.22), Ma et al. (2015).

Langevin dynamics can be discretised by *splitting* the dynamics Leimkuhler and Matthews (2015). First we can consider

$$dv_t = -\gamma \mathrm{M}^{-1} v dt + \sqrt{2\gamma} dW_t,$$

which is an Ornstein-Uhlenbeck process and can be discretised exactly for stepsize ε by sampling from

$$\mathbf{N}(v' \mid e^{-\gamma\varepsilon} v, (1 - e^{-2\gamma\varepsilon})\mathrm{M}).$$

This steps represents a (partial) momenta refreshment. As this step is exact there is no need to apply an accept-reject step in this half of the split dynamics (the acceptance probability is always 1).

The other half of the split is Hamiltonian dynamics

$$\begin{aligned} dx_t &= \mathrm{M}^{-1} v_t dt, \\ dv_t &= -\nabla U(x_t) dt, \end{aligned}$$

which we can discretise using one or more steps of the leapfrog integrator Equation (2.26). We then flip the momenta to induce an involutive deterministic transition before applying a Metropolis accept-reject step.

We then additionally add a final momenta flip after the accept-reject step Horowitz (1991) so that trajectories with high acceptance rates can continue in the same direction whereas a rejection reverses the momenta. This momenta flip is always accepted as the extended target is symmetric in v.

The tuning parameters of this kernel are the stepsize ε, friction parameter γ and the number of leapfrog steps. Taking $\gamma \to \infty$ changes the partial momenta refreshment into full momenta refreshment, i.e. sampling the momenta from $\mathbf{N}(v' \mid 0, \mathrm{M})$. Subsequently taking one leapfrog step results in the Metropolis adjusted Langevin algorithm (MALA) and taking more than one leapfrog step results in Hamiltonian Monte Carlo (HMC) - as such the Horowitz (1991) scheme represents a convenient unifying framework. Sophisticated adaptive schemes for tuning the stepsize within sequential Monte Carlo have been developed in Buchholz et al. (2021), in this work we use a Robbins-Monro algorithm (with constant adaptation stepsize δ) to ensure the average Metropolis-Hastings acceptance probability $\bar{\alpha}_t = \frac{1}{N}\sum \alpha_t^{(i)}$ is pushed towards a target α^*. The full sequential Monte Carlo regime in the case of a single leapfrog step and stepsize adaptation is detailed in Algorithm 21.

6.3 Quasi-Newton Langevin Kernel

The idea behind Girolami and Calderhead (2011) is to extend gradient based Markov Chain Monte Carlo methods Section 2.4 to the case where the the preconditioning matrix M is position dependent $\mathrm{M}(x)$. The *simplified pre-conditioned Langevin dynamics* become

$$\begin{aligned} dx_t &= \mathrm{M}(x_t)^{-1} v_t dt, \\ dv_t &= -\nabla U(x_t)dt - \gamma \mathrm{M}(x_t)^{-1} v_t dt + \sqrt{2\gamma} dW_t. \end{aligned}$$

for target

$$\pi(x,v) \propto \exp(-U(x,v)), \qquad U(x,v) = U(x) + \frac{1}{2} v^{\mathrm{T}} \mathrm{M}(x)^{-1} v.$$

Note that the continuous time dynamics are no longer π-invariant as we have omitted the matrix derivative terms from Equation (2.22), hence **simplified** pre-conditioned Langevin dynamics. The matrix derivatives are normally prohibitively expensive to compute and we can still retain π-invariance via an accept-reject step Leimkuhler et al. (2018).

Algorithm 21 Metropolised SMC with Langevin Proposal

Sample from prior $x_0^{(i)} \sim p(x_0)$ $\quad i = 1\ldots,N$

Sample momenta $v_0^{(i)} \sim \mathbf{N}(v_0 \mid 0, \mathrm{M}_0)$ $\quad i = 1\ldots,N$

Solve for $\lambda_0 \in (0,1]$ such that $\mathrm{ESS}(\lambda_0) \approx \rho N$

Normalise $\hat{Z}_0 = \frac{1}{N}\sum_{i=1}^N p(y \mid x_0^{(i)})^{\lambda_0}$

Weight $w_0^{(i)} = \frac{p(y|x_0^{(i)})^{\lambda_0}}{N\hat{Z}_0}$ $\quad i = 1\ldots,N$

Set $t = 0$

while $\lambda_t \leq 1$ **do**

  Set $t = t+1$

  if $\mathrm{ESS}(\lambda_{t-1}) < \kappa N$ **then**

    $$\left\{\tilde{x}_{0:t-1}^{(i)}, \tilde{w}_{t-1}^{(i)} = \tfrac{1}{N}\right\}_{i=1}^N = \mathrm{Resample}\left(\left\{x_{0:t-1}^{(i)}, w_{t-1}^{(i)}\right\}_{i=1}^N\right)$$

  else

    $$\left\{\tilde{x}_{0:t-1}^{(i)}, \tilde{w}_{t-1}^{(i)}\right\}_{i=1}^N = \left\{x_{0:t-1}^{(i)}, w_{t-1}^{(i)}\right\}_{i=1}^N$$

  for $i = 1,\ldots,N$ **do**

    Set $x'_{[0]} = \tilde{x}_{t-1}^{(i)}$ and refresh momenta

    $$v'_{[0]} \sim \mathbf{N}(v_t \mid e^{-\gamma\varepsilon_t}\tilde{v}_{t-1}^{(i)}, (1-e^{-2\gamma\varepsilon_t})\mathrm{M}_{t-1})$$

    Leapfrog

    $$\begin{aligned} v'_{[\varepsilon_t/2]} &= v'_{[0]} - \frac{\varepsilon_t}{2}\nabla U_{t-1}(x'_{[0]}) \\ x'_{[\varepsilon_t]} &= x'_{[0]} + \varepsilon_t \mathrm{M}_{t-1}^{-1} v'_{[\varepsilon_t/2]} \\ v'_{[\varepsilon_t]} &= v'_{[\varepsilon_t/2]} - \frac{\varepsilon_t}{2}\nabla U_{t-1}(x'_{[\varepsilon_t]}) \end{aligned}$$

    Flip momenta $v'_{[\varepsilon_t]} = -v'_{[\varepsilon_t]}$

    With probability $\min\left(1, \frac{\pi_{t-1}(x'_{[\varepsilon_t]}, v'_{[\varepsilon_t]})}{\pi_{t-1}(x'_{[0]}, v'_{[0]})}\right)$ set $\left(x_t^{(i)}, v_t^{(i)}\right) = \left(x'_{[\varepsilon_t]}, v'_{[\varepsilon_t]}\right)$

    otherwise $\left(x_t^{(i)}, v_t^{(i)}\right) = \left(x'_{[0]}, v'_{[0]}\right)$

    Flip momenta $v_t^{(i)} = -v_t^{(i)}$

  Solve for $\lambda_t \in (\lambda_{t-1}, 1]$ such that $\mathrm{ESS}(\lambda_t) \approx \rho N$

  Normalise $\hat{Z}_{t|0:t-1} = \sum_{i=1}^N \tilde{w}_{t-1}^{(i)} p(y \mid x_t^{(i)})^{\lambda_t}$

  Reweight

  $$w_t^{(i)} = \tilde{w}_{t-1}^{(i)} \frac{p(y \mid x_t^{(i)})^{\lambda_t}}{\hat{Z}_{t|0:t-1}} \qquad i = 1\ldots,N$$

  Adapt stepsize $\log\varepsilon_{t+1} = \log\varepsilon_t + \delta(\bar{\alpha}_t - \alpha^*)$

return $\left\{\left\{x_t^{(i)}, w_t^{(i)}\right\}_{i=1}^N\right\}_{t=0}^T$

Approximating a convex neighbourhood of the target with a Gaussian distribution suggests a logical choice for the position dependent preconditioner $\mathrm{M}(x)$ is the Hessian matrix $\mathrm{B}(x) = \nabla^2 U(x)$.

Unfortunately, we cannot easily use the Hessian matrix directly for general target distributions. Firstly, the matrix inversion and square root required comes at a prohibitive cost of $O(d^3)$ and secondly the Hessian matrix is not necessarily positive definite, a requirement for a valid covariance matrix.

6.3.1 BFGS

Zhang and Sutton (2011) suggest overcoming these issues by invoking a Hessian approximation based on the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm Nocedal and Wright (2006). The BFGS algorithm and its limited-memory variant (L-BFGS) represent the state-of-art in non-stochastic optimisation where a sequence of iterates $x_t = x_{t-1} - \varepsilon_t \hat{\mathrm{B}}_{t-1}^{-1} \nabla U(x_{t-1})$ is generated to minimise the function $U(x)$ where $\hat{\mathrm{B}}_{t-1}^{-1}$ is an approximation to the inverse Hessian at x_{t-1} - thus representing a so called *quasi-Newton* method. The traditional BFGS and L-BFGS algorithms do not provide access to a factorised form of the Hessian matrix that we require for sampling algorithms. Zhang and Sutton (2011) instead utilise the following recursion which directly approximates a square-root of the Hessian and its inverse

$$\mathrm{B}_{t+1} = \mathrm{C}_{t+1}\mathrm{C}_{t+1}^{\mathrm{T}}, \qquad \mathrm{B}_{t+1}^{-1} = \mathrm{S}_{t+1}\mathrm{S}_{t+1}^{\mathrm{T}}, \tag{6.4a}$$

$$\mathrm{C}_{t+1} = (\mathbb{I}_d - \mathrm{u}_t \mathrm{t}_t^{\mathrm{T}})\mathrm{C}_t, \qquad \mathrm{S}_{t+1} = (\mathbb{I}_d - \mathrm{p}_t \mathrm{q}_t^{\mathrm{T}})\mathrm{S}_t, \tag{6.4b}$$

$$\mathrm{t}_t = \frac{s_t}{s_t^{\mathrm{T}} \mathrm{B}_t s_t}, \qquad \mathrm{p}_t = \frac{s_t}{s_t^{\mathrm{T}} y_t}, \tag{6.4c}$$

$$\mathrm{u}_t = \sqrt{\frac{s_t^{\mathrm{T}} \mathrm{B}_t s_t}{s_t^{\mathrm{T}} y_t}}\, y_t + \mathrm{B}_t s_t \qquad \mathrm{q}_t = \sqrt{\frac{s_t^{\mathrm{T}} y_t}{s_t^{\mathrm{T}} \mathrm{B}_t s_t}}\, \mathrm{B}_t s_t + y_t, \tag{6.4d}$$

where $s_t = x_{t+1} - x_t$ and $y_t = \nabla U(x_{t+1}) - \nabla U(x_t)$. The limited-memory variant initiates the recursion with diagonal matrices $\mathrm{C}_{t-m} = \mathrm{S}_{t-m}^{-1}$ that represent an initial guess for the square root of the Hessian and inverse.

In practice, the vectors $\{\mathrm{p}_r, \mathrm{q}_r, \mathrm{u}_r, \mathrm{t}_r\}_{r=t-m}^{t}$ are pre-computed at a cost of $O(m^2 d)$ and subsequent matrix vector products can be computed at a cost of $O(md)$ using the sequence of

inner products

$$\begin{aligned}
\mathrm{C}_{t+1}z &= \prod_{r=t-m+1}^{t} (\mathbb{I}_d - \mathrm{u}_r \mathrm{t}_r^{\mathrm{T}})\mathrm{C}_{t-m}z, \\
&= (\mathbb{I}_d - \mathrm{u}_t \mathrm{t}_t^{\mathrm{T}}) \ldots (\mathbb{I}_d - \mathrm{u}_{t-m+1} \mathrm{t}_{t-m+1}^{\mathrm{T}})\mathrm{C}_{t-m}z, \\
\mathrm{S}_{t+1}z &= \prod_{r=t-m+1}^{t} (\mathbb{I}_d - \mathrm{p}_r \mathrm{q}_r^{\mathrm{T}})\mathrm{S}_{t-m}z, \\
&= (\mathbb{I}_d - \mathrm{p}_t \mathrm{q}_t^{\mathrm{T}}) \ldots (\mathbb{I}_d - \mathrm{p}_{t-m+1} \mathrm{q}_{t-m+1}^{\mathrm{T}})\mathrm{S}_{t-m}z,
\end{aligned}$$

thus the L-BFGS variant as described above provides access to a factorised approximation of the Hessian and inverse Hessian matrices all at a cost that is linear in dimension.

We are yet to ensure the Hessian approximation is positive definite. We can do this by checking that $s_r^{\mathrm{T}} y_r > 0$ for each $r = t-m, \ldots, t$ Nocedal and Wright (2006). This is obtained in Zhang and Sutton (2011) by simply removing points from the recursion when $s_r^{\mathrm{T}} y_r \leq 0$. In this work, we adopt a strategy similar to Schraudolph et al. (2007) where we notice that we can instead approximate $\nabla^2 U(x) + \beta \mathrm{B}_{t-m}$ by adjusting each $y_r \leftarrow y_r + \beta \mathrm{B}_{t-m} s_r$. We can guarantee a positive definite approximation given β suitably large and a positive definite (diagonal) initial guess B_{t-m}. In practice, we adaptively set

$$\beta = \max\left(0, \max\left(\left\{\frac{-s_r^{\mathrm{T}} y_r}{s_r^{\mathrm{T}} \mathrm{B}_{t-m} s_r}\right\}_{r=t-m}^{t}\right) + \omega\right), \tag{6.5}$$

so that each $s_r^{\mathrm{T}} y_r > \omega$ for some bounding parameter $\omega > 0$.

We now have the tools we need to apply a Langevin proposal on the extended target with adjusted tempered potential

$$U_t(x_t, v_t) = U_t(x_t) + \frac{1}{2} v_t^{\mathrm{T}} \mathrm{B}_t(x_{t-m-1:t-1})^{-1} v_t,$$

where $\mathrm{B}_t(x_{t-m-1:t-1})$ represents the L-BFGS Hessian approximation at inverse temperature λ_t using trajectory values $x_{t-m-1:t-1}$.

The use of the previous trajectory means the proposal is no longer Markovian. Zhang and Sutton (2011) correct for this to obtain a valid Markov Chain Monte Carlo sampler by extending the state to include all m previous values. In this work, we do not correct for this bias and instead note as in Wang et al. (2021) that the bias is controllable as increasing m and decreasing the stepsize ε_t increases the accuracy of the Hessian approximation (or rather its positive definite projection). A modified version of the full sequential Monte Carlo procedure Algorithm 21 that uses an L-BFGS subroutine is described in Algorithm 22.

Algorithm 22 Metropolised SMC with Quasi-Newton Langevin Proposal

Sample from prior $x_0^{(i)} \sim p(x_0)$ $\quad i = 1\ldots,N$

Sample momenta $v_0^{(i)} \sim \mathbf{N}(v_0 \mid 0, \mathrm{M}_0)$ $\quad i = 1\ldots,N$

Solve for $\lambda_0 \in (0,1]$ such that $\mathrm{ESS}(\lambda_0) \approx \rho N$

Normalise $\hat{Z}_0 = \frac{1}{N}\sum_{i=1}^N p(y \mid x_0^{(i)})^{\lambda_0}$

Weight $w_0^{(i)} = \frac{p(y|x_0^{(i)})^{\lambda_0}}{N\hat{Z}_0}$ $\quad i = 1\ldots,N$

Set $t = 0$

while $\lambda_t \leq 1$ **do**

  Set $t = t+1$

  if $\mathrm{ESS}(\lambda_{t-1}) < \kappa N$ **then**

    $$\left\{\tilde{x}_{0:t-1}^{(i)}, \tilde{w}_{t-1}^{(i)} = \tfrac{1}{N}\right\}_{i=1}^N = \mathrm{Resample}\left(\left\{x_{0:t-1}^{(i)}, w_{t-1}^{(i)}\right\}_{i=1}^N\right)$$

  else

    $$\left\{\tilde{x}_{0:t-1}^{(i)}, \tilde{w}_{t-1}^{(i)}\right\}_{i=1}^N = \left\{x_{0:t-1}^{(i)}, w_{t-1}^{(i)}\right\}_{i=1}^N$$

  for $i = 1,\ldots,N$ **do**

    Compute $\left\{\mathrm{p}_r^{(i)}, \mathrm{q}_r^{(i)}, \mathrm{u}_r^{(i)}, \mathrm{t}_r^{(i)}\right\}_{r=t-m-1}^{t-1}$ using $\left\{\tilde{x}_r^{(i)}, \nabla U_{t-1}(\tilde{x}_r^{(i)})\right\}_{r=t-m-1}^{t-1}$

    Set $x'_{[0]} = \tilde{x}_{t-1}^{(i)}$ and refresh momenta

$$v'_{[0]} \sim \mathbf{N}(v_t \mid e^{-\gamma\varepsilon_t}\tilde{v}_{t-1}^{(i)}, (1-e^{-2\gamma\varepsilon_t})\mathrm{B}_{t-1}(\tilde{x}_{t-m-1:t-1}^{(i)}))$$

    Leapfrog

$$\begin{aligned} v'_{[\varepsilon_t/2]} &= v'_{[0]} - \frac{\varepsilon_t}{2}\nabla U_{t-1}(x'_{[0]}) \\ x'_{[\varepsilon_t]} &= x'_{[0]} + \varepsilon_t \mathrm{B}_{t-1}(\tilde{x}_{t-m-1:t-1}^{(i)})^{-1} v'_{[\varepsilon_t/2]} \\ v'_{[\varepsilon_t]} &= v'_{[\varepsilon_t/2]} - \frac{\varepsilon_t}{2}\nabla U_{t-1}(x'_{[\varepsilon_t]}) \end{aligned}$$

    Flip momenta $v'_{[\varepsilon_t]} = -v'_{[\varepsilon_t]}$

    With probability $\min\left(1, \frac{\pi_{t-1}(x'_{[\varepsilon_t]}, v'_{[\varepsilon_t]})}{\pi_{t-1}(x'_{[0]}, v'_{[0]})}\right)$ set $\left(x_t^{(i)}, v_t^{(i)}\right) = \left(x'_{[\varepsilon_t]}, v'_{[\varepsilon_t]}\right)$

    otherwise $\left(x_t^{(i)}, v_t^{(i)}\right) = \left(x'_{[0]}, v'_{[0]}\right)$

    Flip momenta $v_t^{(i)} = -v_t^{(i)}$

  Solve for $\lambda_t \in (\lambda_{t-1}, 1]$ such that $\mathrm{ESS}(\lambda_t) \approx \rho N$

  Normalise $\hat{Z}_{t|0:t-1} = \sum_{i=1}^N \tilde{w}_{t-1}^{(i)} p(y \mid x_t^{(i)})^{\lambda_t}$

  Reweight

$$w_t^{(i)} = \tilde{w}_{t-1}^{(i)} \frac{p(y \mid x_t^{(i)})^{\lambda_t}}{\hat{Z}_{t|0:t-1}} \qquad i = 1\ldots,N$$

  Adapt stepsize $\log\varepsilon_{t+1} = \log\varepsilon_t + \delta(\bar{\alpha}_t - \alpha^*)$

return $\left\{\left\{x_t^{(i)}, w_t^{(i)}\right\}_{i=1}^N\right\}_{t=0}^T$

6.4 Numerical Experiments

We now investigate the numerical performance of the sequential Monte Carlo regimes with both classical Langevin proposal and quasi-Newton Langevin proposal.

It is common for sequential Monte Carlo on static problems to take multiple leapfrog steps and/or accept-reject steps at each iteration alongside an aggressive choice of the ESS threshold parameter ρ (i.e. $\rho = 0.5$) that controls the size of temperature jumps. With an aggressive choice of ρ many likelihood evaluations are taken at each iteration but the χ^2-distance between tempered distributions is large. In this work, we take an alternative approach with only one leapfrog step accompanied with accept-reject step alongside modest $\rho = 0.95$, this way only one likelihood evaluation is executed per particle per iteration. Although not utilised here, this approach has the advantage that every likelihood evaluation can contribute to expectation approximations via the waste-recycling technique used in Gramacy et al. (2010); Nguyen et al. (2016). This technique uses the fact that the output from each iteration $\{x_t^{(i)}, w_t^{(i)}\}_{i=1}^N$ is asymptotically unbiased for the tempered intermediate distribution $\pi_t(x)$, and therefore we can adjust the weights to target the posterior $\pi_T(x)$, this way all NT particles contribute to posterior expectations.

In all experiments we fix $N = 1000$ and resample when the effective sample size falls below $0.5N$. As mentioned we use a Robbins-Monro schedule with constant adaptation stepsize $\delta = 1$ to keep the Metropolis acceptance probability close to 80%. We investigate a range of friction parameters $\gamma \in \{0.1, 1, 10, 100, \infty\}$. For the classical Langevin proposals we adopt the common practice of setting the preconditioning matrix $\mathrm{M}_t = \mathbb{I}_d$. In the L-BFGS subroutine, we fix the positive-definite parameter $\omega = 1$. All experiments are repeated 20 times.

6.4.1 High Dimensional Gaussian

The first inference task we consider exhibits high dimensionality and inhomogeneous scaling Neal (2010) where the target distribution $\pi(x)$ is a 100-dimensional, zero mean Gaussian with covariance $\Sigma = \mathrm{diag}(1, \ldots, 100)$.

As the problem does not have a prior-likelihood structure we use the following artificial likelihood tempering

$$\pi_t(x_t) = \pi_0(x_t) \left(\frac{\pi(x_t)}{\pi_0(x)} \right)^{\lambda_t},$$

where $\pi_0(x) = \mathbf{N}(x \mid 0, 50.5\mathbb{I}_{100})$.

In our L-BFGS implementation, we use a memory $m = 20$ and initiate the Hessian approximations with the inverse of the diagonal sample covariance of the previous particles.

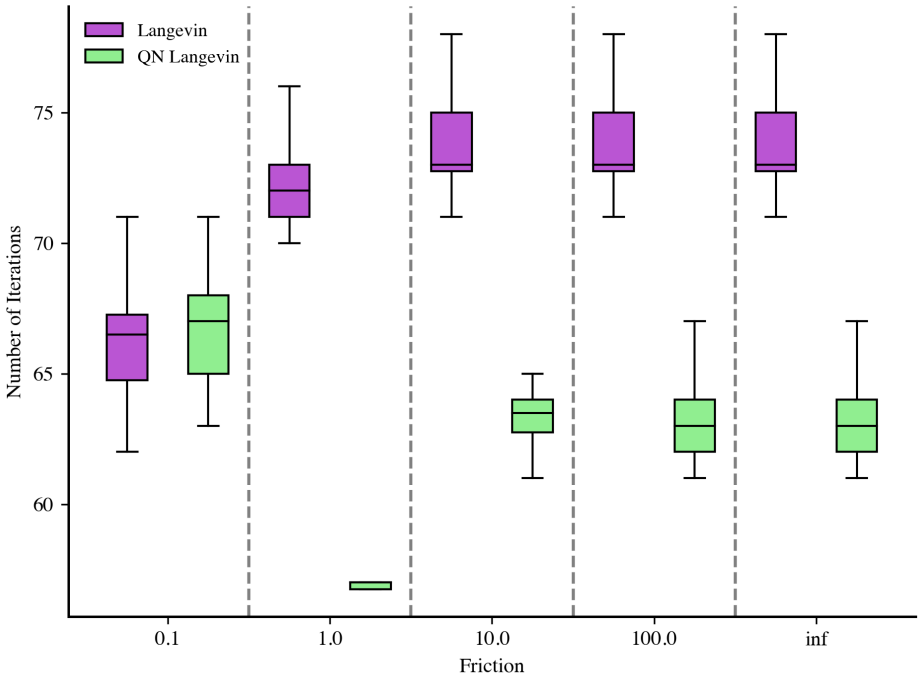


Fig. 6.1 Number of temperatures required to reach posterior for varying friction parameter in high dimensional Gaussian example.

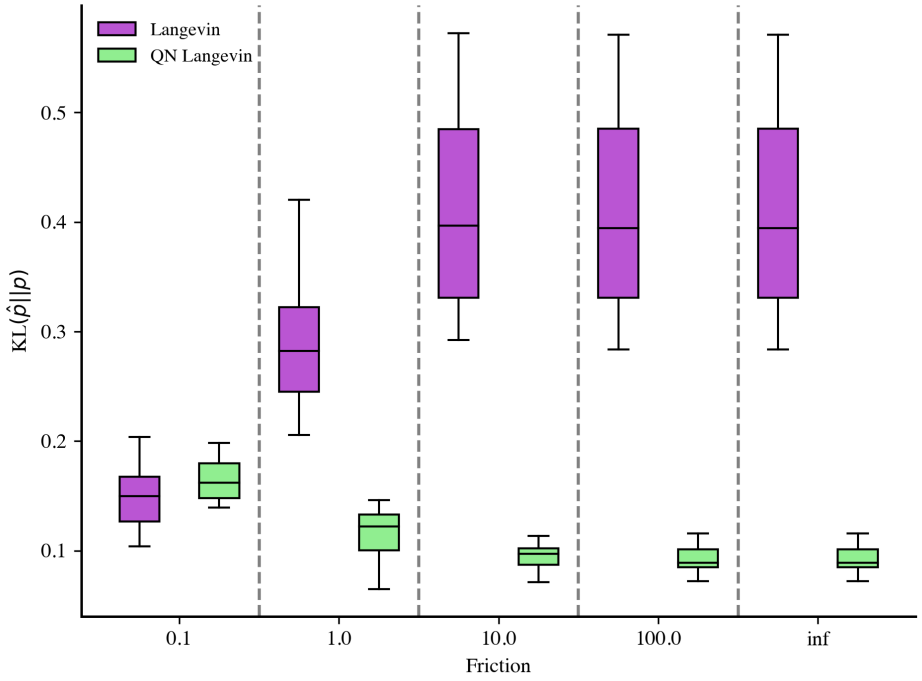


Fig. 6.2 KL-divergence between final particle approximation and target distribution for varying friction parameter in high dimensional Gaussian example.

In Figure 6.1, we display the number of iterations, T, the adaptive sequential Monte Carlo schemes required to reach the target distribution at inverse temperature $\lambda_T = 1$. The number of iterations required represents the difference in computational cost between the Metropolised classical proposal and the Metropolised quasi-Newton Langevin proposal - as both require the same number of likelihood and gradient evaluations per iteration. In Figure 6.2, we analyse the accuracy of the particle approximations by comparing the KL-divergence from the Gaussian distribution induced by the weighted sample mean and covariance of the final particles to the true target distribution $\mathbf{N}(x \mid 0, \Sigma)$ - calculated using Equation (3.10).

Recall that a small friction coefficient, γ, represents only a partial momenta refreshment, whereas $\gamma = \infty$ represents a full momenta refreshment - sampling directly from $\mathbf{N}(v' \mid 0, \mathrm{M}(x))$. We observe in Figure 6.1 that for the smallest friction parameter the momenta is slower to adjust to the scaling of the Hessian approximation, resulting in similar performance between classical and quasi-Newton regimes. For larger friction parameters, the performance of the classical Langevin SMC significantly deteriorates in both speed Figure 6.1 and accuracy Figure 6.2. This is in contrast to the quasi-Newton scheme which is faster and more accurate for larger friction parameters, finding a sweet-spot at $\gamma = 1$.

6.4.2 Gaussian Mixture Model

Our second example is considered in Chopin et al. (2012) and represents fitting a univariate dataset $y = (y_1, \ldots, y_K)$ with a weighted sum of three Gaussian distributions

$$p(y_k \mid z, \mu, v) = \sum_{i=1}^{3} z_i \mathbf{N}(y_k \mid \mu_i, v_k^{-1}),$$

where $\sum_{i=1}^{3} z_i = 1$. We define the prior distribution hierarchically

$$\begin{aligned}
(z_1, z_2, z_3) &\sim \text{Dirichlet}(1,1,1), \\
\mu_i &\sim \mathbf{N}(\cdot \mid m, \kappa^{-1}), && i = 1,2,3, \\
v_i &\sim \text{Gamma}(\cdot \mid \alpha, \beta), && i = 1,2,3, \\
\beta &\sim \text{Gamma}(\cdot \mid g, h),
\end{aligned}$$

where m, κ, α, g and h are constants. Thus our parameter to be inferred is the nine dimensional $x = (\mu, v, z_1, z_2, \beta)$ with $z_3 = 1 - z_1 - z_2$.

Note that we have the constraints $v_i > 0$, $\beta > 0$ and a 3-simplex constraint on the weights z. To facilitate the gradient based sampling algorithms we take log transforms on the positive

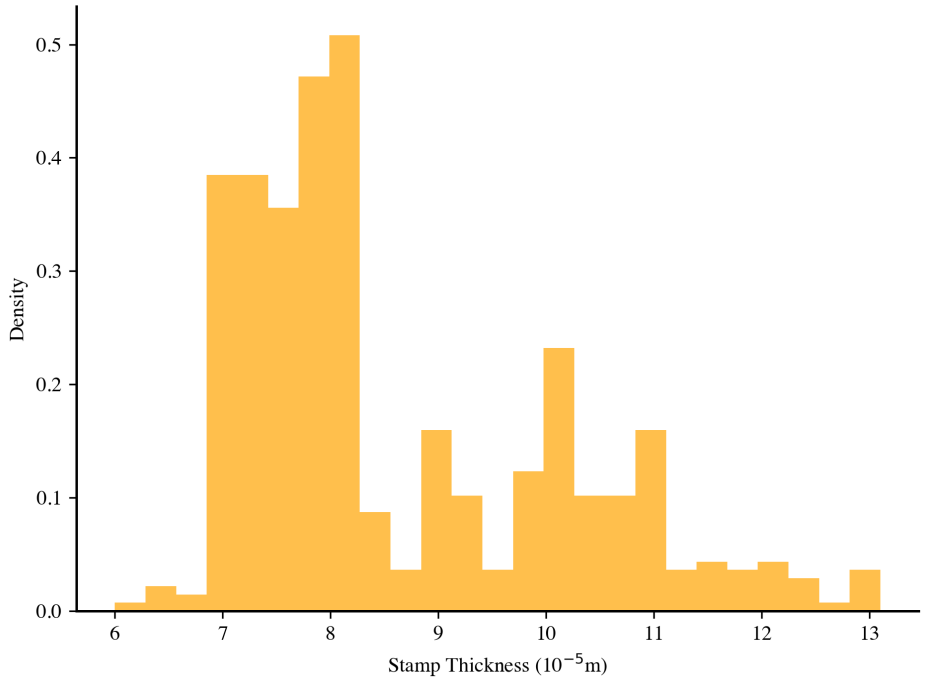


Fig. 6.3 Hidalgo stamp data.

parameters ν, β and use the simplex transformation detailed in Betancourt (2012) to unconstrain the weights z. We adjust the prior density with the transformation Jacobians accordingly.

We consider fitting the Hidalgo stamp dataset Izenman and Sommer (1988) which consists of 485 datapoints representing the thickness of individual stamps, depicted in Figure 6.3. This model exhibits a *label switching problem* where a-priori each of the three mixture components are identical and are therefore invariant to re-labelling. Thus the model admits $3! = 6$ local modes which will each have their own local scaling.

We again tested the quasi-Newton Langevin sequential Monte Carlo scheme (Algorithm 22) against sequential Monte Carlo with classical Langevin proposals (Algorithm 21) over a range of friction parameters. When calculating the L-BFGS Hessian approximations Equation (6.4), we set $m = 40$ and initiated the Hessian with a constant diagonal matrix based on the prior covariance.

Figure 6.4 displays the number of temperatures or iterations required to reach the posterior distribution - which is determined adaptively. We again notice as in the Gaussian example that the more gentle momenta refreshment from the smallest friction results in similar speeds induced by the classical and quasi-Newton proposals. For larger friction parameters, the quasi-Newton accelerates more quickly to the posterior distribution and is significantly faster than its classical counterpart.

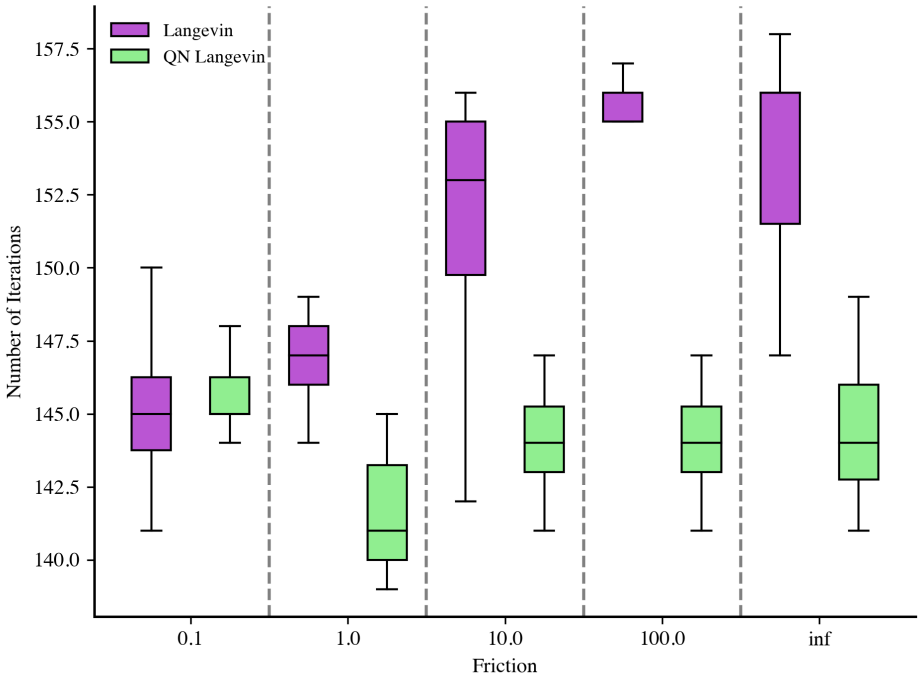


Fig. 6.4 Number of temperatures required to reach posterior for varying friction parameter in Gaussian mixture model example.

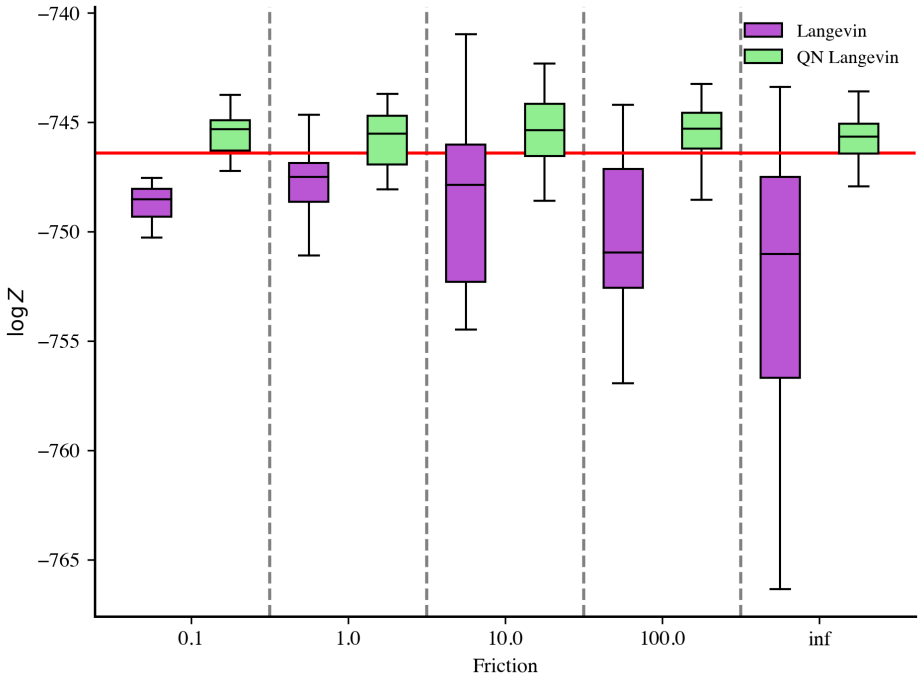


Fig. 6.5 Estimate of log normalising constant for varying friction parameter in Gaussian mixture model example. Truth represented by horizontal red line.

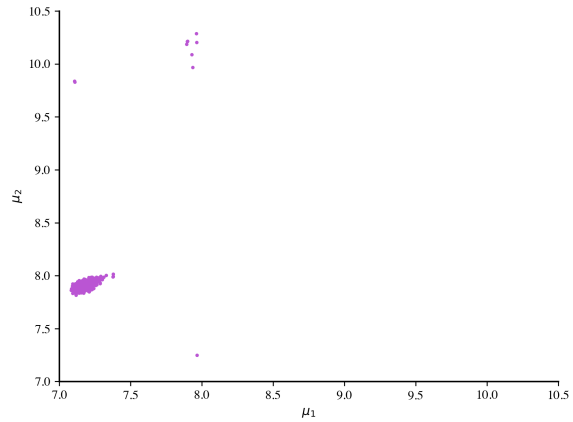


Fig. 6.6 Langevin.

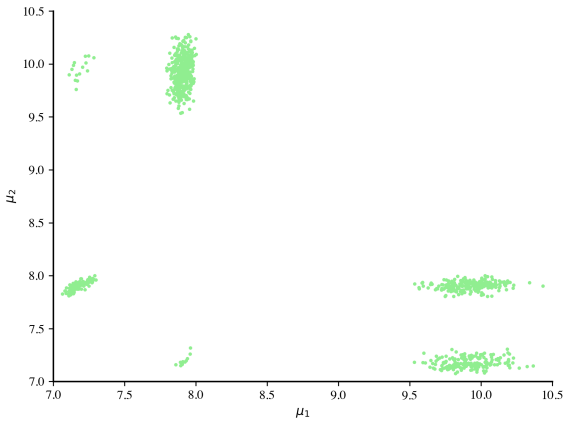


Fig. 6.7 Quasi-Newton Langevin.

SMC posterior samples for Gaussian mixture model.

In this example, the true posterior is not available analytically. Thus to compare the accuracy of particle approximations we compare the estimate of the (log) normalising constant $\hat{Z}_T \approx Z_T = \int p(x)p(y \mid x)dx$ which is calculated sequentially $\hat{Z}_T = \hat{Z}_0 \prod_{t=1}^{T} \hat{Z}_{t|0:t-1}$. The final estimate of $\log \hat{Z}_T$ for the two sequential Monte Carlo algorithms over a range of friction parameters is displayed in Figure 6.5 alongside the "true" log normalising constant calculated from an extended run of SMC with classical Langevin proposals and a very large 10000 particles. We observe that the quasi-Newton approach is consistently more precise and accurate for each friction parameter. In addition, we display the posterior samples generated in the μ_1 and μ_2 dimensions for Langevin proposals in Figure 6.6 and quasi-Newton Langevin proposals in Figure 6.7. We clearly see that the use of the local Hessian approximation has allowed the quasi-Newton algorithm to explore all 6 modes whereas the classical Langevin proposals have all but collapsed to a single mode.

6.5 Discussion

In this chapter, we derived a sequential Monte Carlo technique for sampling from static Bayesian inference problems where gradient evaluations are used to form a Hessian approximation, efficiently preconditioning the dynamics Zhang and Sutton (2011). The Hessian approximation uses a variant of the L-BFGS algorithm that provides access to both the Hessian and inverse Hessian in a factorised square-root form Equation (6.4). The approximation is cheap to compute at a cost of $O(m^2 d)$, where m is a tunable memory parameter that is typically set in the range 10-50. Importantly, the Hessian approximation requires no additional gradient evaluations.

To our knowledge, this work provides the first application of Hessian approximations within sequential Monte Carlo methods for static Bayesian inference problems.

The Hessian approximation utilises the previous m states of the particles trajectory and therefore breaks the validity of the sequential importance weights. We accept this bias in the belief that for practical problems this bias is likely to be dominated by standard Monte Carlo variance and that for suitably large m and small stepsize the Hessian approximation (or rather its positive definite approximation) will be increasingly accurate. In the case that approximation is exact, we become asymptotically unbiased again.

We have demonstrated the benefits of the local preconditioner in both a high dimensional example with difficult scaling as well as a hierarchical, multi-modal example with real data.

With regards to future work, alternative Hessian approximations could be investigated. It is possible to derive a preconditioned version of Langevin dynamics that does not require the square root of the preconditioner Fu et al. (2016); Leimkuhler et al. (2018). This can be achieved by targeting a vanilla extended potential $U(x,v) = U(x) + \frac{1}{2}v^{\mathrm{T}}v$ but instead preconditioning with the modified skew symmetric matrix

$$\mathrm{Q}(x,v) = \begin{pmatrix} 0 & -\nabla^2 U(x)^{-1} \\ \nabla^2 U(x)^{-1} & 0 \end{pmatrix}$$

from Equation (2.22). We now no longer require a factorised version of the Hessian however we still require its inverse. We note that modern automatic differentiation offers Hessian-vector products at an equivalent computational cost to a gradient evaluation, subsequently we can take approximate inverse Hessian-vector products using a (possibly stochastic) truncation of the series $\mathrm{B}^{-1} = \sum_{i=1}^{\infty}(\mathbb{I} - \mathrm{B})^i$ - assuming the spectrum of B allows convergence Agarwal et al. (2017). It may even be possible to extend this further to calculate unbiased estimates of the matrix divergence terms in Equation (2.22) using the techniques described in Leimkuhler et al. (2018).

Another extension would be to investigate the potential of using a Hessian approximation within sequential Monte Carlo schemes that do not make use of an accept-reject step and therefore use a transition kernel that is not π_{t-1}-invariant. That is sequential importance weights of the form

$$w_t = w_{t-1}\frac{\pi_t(x_t)\pi_{t-1|t}(x_{t-1} \mid x_t)}{\pi_{t-1}(x_{t-1})q_{t|t-1}(x_t \mid x_{t-1})},$$

where the forward kernel $q_{t|t-1}$ no longer utilises an accept-reject step and $\pi_{t-1|t}$ is an alternative backward kernel. An optimal forward kernel $q_{t|t-1}$ would directly convert a sample from π_{t-1}

into one from π_t, i.e.

$$\int \pi_{t-1}(x_{t-1}) q_{t|t-1}(x_t \mid x_{t-1}) dx_{t-1} = \pi_t(x_t).$$

It would be exciting to investigate whether one could combine Taylor expansions on the tempered potential Titsias and Papaspiliopoulos (2016) with a Hessian approximation to derive an approximation to an optimal forward kernel - similar to the ideas presented in Chapter 5.

As mentioned, another choice of intermediate distributions for sequential Monte Carlo is the so called batch intermediates Chopin et al. (2012)

$$\pi_t(x_t) = p(x_t \mid y_1, \ldots, y_t).$$

We note that it is indeed possible to combine likelihood tempering and data batching in the choice of intermediate distributions

$$\pi_t(x_t) \propto p(x_t \mid y_1, \ldots, y_r) p(y_{r+1} \mid x_t)^{\lambda_t}.$$

We can now enforce smooth transitions between targets using the same effective sample size based adaptive tempering from Section 6.1.2, although we still require a routine describing how to batch the data. An interesting extension of this approach is that the aforementioned sequential Monte Carlo algorithms can be applied in online settings, where the data is received sequentially but the unknown parameters are still assumed to be static. This represents an advantage over Markov Chain Monte Carlo approaches which cannot be updated in light of further observations.

Chapter 7

mocat

This thesis is accompanied by an open source python Van Rossum and Drake (2009) package mocat that represents a general purpose toolbox for fully customisable Monte Carlo sampling including plug and play implementations for all of the sampling algorithms presented in this thesis (Chapter 2, Chapter 3, Chapter 5, Chapter 6).

Just like bmm, Section 4.6, mocat is hosted on PyPI (2021) and can be installed easily from the command line with pip.

```
pip install mocat
```

Source code can be found at github.com/SamDuffield/mocat.

7.1 JAX

mocat is written in python but is entirely dependent on the package JAX, Bradbury et al. (2018). JAX code is compiled using XLA Sabne (2020), a computational backend in C++, as such JAX provides lightning fast scientific calculations via a convenient python frontend. JAX documentation can be found at jax.readthedocs.io, but note the caveat that JAX is still being regularly updated and some numpy function are yet to be implemented or may not be fully optimised.

JAX adopts the familiar interface of numpy Harris et al. (2020) but goes much further. Indeed many numpy manipulations can be replaced with JAX counterparts

```
from jax import numpy as jnp
```

for an easy speedup.

Carefully written JAX code is **differentiable**

```
from jax import grad

grad(jnp.sin)(2 * jnp.pi)
    DeviceArray(1., dtype=float32)
```

or perhaps more usefully

```
from jax import value_and_grad

value_and_grad(jnp.sin)(2 * jnp.pi)
    (DeviceArray(0., dtype=float32), DeviceArray(1., dtype=float32))
```

Additionally, JAX is equipped with a `jax.jit` function for accelerated just-in-time compilation in a similar fashion to `numba` Lam et al. (2015) as well as efficient parallel dispatch on modern computer architectures (GPUs and TPUs).

In order to retain differentiability and allow XLA acceleration, `if` statements as well as `for` and `while` loops have to be written with care.

`if` statements

For very simple `if` statements that require no additional function calls dependent on the value of the boolean `condition` we can replace

```
if condition:
    x = 5
else:
    x = 10
```

with

```
x = jnp.where(condition, 5, 10)
```

For more complex `if` statements where we only want to execute one of two functions we can use `jax.lax.cond`

```
from jax.lax import cond

x = cond(condition,
         lambda y: y,
         lambda y: y * 2,
         operand=5)
```

`for` loops

Basic `for` loops that can be written in `python` using *list comprehension*

```
out_list = [func(x) for x in range(10)]
```

can be vectorised (for a suitable `func`) using `jax.vmap`

```
out_array = vmap(func)(jnp.arange(10))
```

Typically the vectorised `vmap` call is significantly faster than the native `python` list comprehension.

For Markovian `for` loops where the function iterates depending on its previous value, we can use the extremely flexible `jax.lax.scan`. From the `JAX` documentation we have

```
def scan(f, init, xs):
  carry = init
  ys = []
  for x in xs:
    carry, y = f(carry, x)
    ys.append(y)
  return carry, np.stack(ys)
```

which becomes succinctly

```
from jax.lax import scan

carry, stacked_ys = scan(f, init, xs)
```

Note the flexibility in the output of `scan` - the first element `carry` represents the first element of the output of the final call to `f` whereas the second element `stacked_ys` represents the second element of the output from every call to `f` stacked into a single array. This flexibility as well as the speed of `XLA` compilation makes `jax.lax.scan` extremely useful for iterative algorithms such as Markov chain Monte Carlo Section 2.4.

`JAX` has a similar implementation for `while` loops in `jax.lax.while_loop`, however only the output from the final iteration call is returned. Since `jax.lax.scan` has the added flexibility of being able to return stacked output, it is generally favoured for `while` loops that can be reformulated as `for` loops. For `while` loops where we desire stacked output, `mocat` provides a solution with `mocat.utils.while_loop_stacked`.

Bonus comments

In order to retain differentiability (`jax.grad` and `jax.value_and_grad`), all arrays (intermediate or otherwise) within calls of `vmap`, `scan` etc must be of constant size - this can be an issue for algorithms such as rejection sampling Section 2.2.

`JAX` favours pure functions and as such does not immediately support the in-place updates that are common in `numpy` (e.g. `x[0] = 4`). This can be circumvented using `jax.ops.index_update` however it is typically preferred to use pure functions defined over full arrays or calls to `jnp.where`.

Finally, `JAX` behaves somewhat atypically in its treatment of pseudo-random seeds. In particular, every call to one of `JAX`'s functions that calls a pseudo-random number generator must be accompanied with a two element array representing a random key.

```
from jax import random

random_key = random.PRNGKey(0)
random_key
    DeviceArray([0, 0], dtype=uint32)
```

Using the same key will generate the same random numbers

```
random.normal(random_key)
    DeviceArray(-0.20584235, dtype=float32)

random.normal(random_key)
    DeviceArray(-0.20584235, dtype=float32)
```

for this reason we have to split the random key before calling a pseudo-random number generator

```
random_key, sub_key_1, sub_key_2 = random.split(random_key, 3)
random.normal(sub_key_1)
    DeviceArray(0.5781487, dtype=float32)

random.normal(sub_key_2)
    DeviceArray(0.85355157, dtype=float32)
```

Of course `JAX` has many more useful features and many more *gotchas* - all thoroughly described in the documentation jax.readthedocs.io.

7.2 Monte Carlo Sampling

`mocat` makes it easy to run the most common Monte Carlo algorithms for offline Bayesian inference as well as providing a framework to implement exciting, new sampling algorithms.

`mocat.cdict`

A fundamental object within `mocat` is that of a `cdict`. A `cdict` conveniently stores multiple named attributes including `DeviceArrays`.

```
import mocat

random_key = random.PRNGKey(0)
random_key, sub_key_1, sub_key_2 = random.split(random_key, 3)

sample = mocat.cdict(x=-5+0.1*random.normal(sub_key_1, shape=(4, 2)),
                     momenta=random.normal(sub_key_2, shape=(4, 2)),
                     name='Gaussian sample')

sample.name
    'Gaussian sample'

sample.x
    DeviceArray([[-5.1619368, -4.871614 ],
                 [-5.1136875, -5.048856 ],
                 [-5.0195227, -4.8458843],
                 [-5.037027 , -5.017855 ]], dtype=float32)

sample.momenta
    DeviceArray([[-0.38537452, -1.4707391 ],
                 [ 0.5467919 ,  2.095505  ],
                 [ 1.1165614 ,  0.16117463],
                 [-0.5371375 , -0.89213836]], dtype=float32)
```

We can display the elements of the cdict

```
sample.keys()
    dict_keys(['x', 'momenta', 'name'])
```

and even index all of the `DeviceArrays` stored within the `cdict`

```
first_sample = sample[0]

first_sample.name
    'Gaussian sample'

first_sample.x
    DeviceArray([-5.1619368, -4.871614 ], dtype=float32)

sample.momenta
    DeviceArray([-0.38537452, -1.4707391 ], dtype=float32)
```

We can also save and load `cdicts` easily

```
sample.save(path)
same_sample = mocat.load_cdict(path)
```

`mocat.Scenario`

In order to apply Monte Carlo methods, we need a distribution to sample from! For classical Monte Carlo methods such as Markov chain Monte Carlo and importance sampling we require access to evaluations of the target distribution's density function. In `mocat`, this is achieved via the so called *potential* function $U(x)$

$$U(x) = -\log \pi(x) \iff \pi(x) = \exp(-U(x)),$$

where $\pi(x)$ is the probability density function of the target distribution. A strength of Monte Carlo methods is that $\pi(x)$ (and subsequently $U(x)$) only need be defined up to normalisation constant.

In `mocat`, a target distribution's potential is stored in a `class` that inherits `mocat.Scenario`. This can be done either directly

```
class Funnel(mocat.Scenario):
    dim = 5
    def potential(self,
                  x: jnp.ndarray,
                  random_key: jnp.ndarray = None) -> float:
        return 0.5 * (x[-1] ** 2 / 9 + (x[:-1] ** 2 / jnp.exp(x[-1]) +
        ↪  x[-1]).sum())
```

or by defining a Bayesian prior and likelihood function

```
class BayesFunnel(mocat.Scenario):
    dim = 5
    prior_sd = 10.

    def prior_potential(self,
                        x: jnp.ndarray,
                        random_key: jnp.ndarray = None) -> float:
        return 0.5 * jnp.square(x / self.prior_sd).sum()

    def likelihood_potential(self,
                             x: jnp.ndarray,
                             random_key: jnp.ndarray = None) -> float:
        return 0.5 * (x[-1] ** 2 / 9 + (x[:-1] ** 2 / jnp.exp(x[-1]) +
        ↪  x[-1]).sum())

    def prior_sample(self,
                     random_key: jnp.ndarray) -> jnp.ndarray:
        return self.prior_sd * random.normal(random_key,
        ↪  shape=(self.dim,))
```

Note that the `Scenario` class has a compulsory `dim` attribute that is used by subsequent sampling algorithms. Additionally note that the `BayesFunnel` has a `prior_sd` parameter that is accessed within `prior_potential` via `self.prior_sd` - this technique could also be used alongside a `data` attribute accessed within `likelihood_potential`.

The `prior_sample` method is required for initiating some sampling algorithms, such as the approaches that use tempering in Chapter 5 and Chapter 6.

The additional `random_key` argument in the potential methods is there to permit the option of stochastic mini-batching in the likelihood calls - in the majority of cases it can be left as an unused argument.

Initiating an instance of the scenario

```
funnel_scen = BayesFunnel()
```

will initiate a `potential` method in the case of Bayesian prior and likelihood scenario. Additionally it will automatically initiate the first derivative of the potential `grad_potential` and `potential_and_grad` as well as the equivalent methods for `prior_potential` and `likelihood_potential`.

Markov Chain Monte Carlo

We can now develop our sampling algorithms. Recall that Markov chain Monte Carlo (MCMC), Section 2.4 collects samples by iterating a π-invariant kernel

$$x^{(i)} \sim K(\cdot \mid x^{(i-1)}), \qquad i = 1, \ldots, N,$$

where K most commonly consists of a proposal followed by an accept-reject step. `mocat` has a built-in `MCMCSampler` class that can be inherited to build bespoke MCMC sampling algorithms.

`mocat` has built-in the very general gradient based sampler `mocat.Underdamped` of Horowitz (1991) which as described in Section 6.2 incorporates the popular MALA and HMC sampling algorithms. `mocat` also has the basic random-walk Metropolis Hastings algorithm `mocat.RandomWalk` whose code we describe here

```
class RandomWalk(MCMCSampler):
    name = 'Random Walk'
    correction = Metropolis

    def __init__(self,
                 stepsize: float = None):
        super().__init__()
        self.parameters.stepsize = stepsize
        self.tuning.target = 0.234

    def startup(self,
                scenario: Scenario,
                n: int,
                initial_state: cdict,
                initial_extra: cdict,
                **kwargs) -> Tuple[cdict, cdict]:
        initial_state, initial_extra = super().startup(scenario, n,
                                                       initial_state, initial_extra, **kwargs)
        initial_extra.random_key, scen_key = random.split(initial_extra.random_key)
        initial_state.potential = scenario.potential(initial_state.value, scen_key)
        return initial_state, initial_extra

    def proposal(self,
                 scenario: Scenario,
                 reject_state: cdict,
                 reject_extra: cdict) -> Tuple[cdict, cdict]:
        proposed_state = reject_state.copy()

        d = scenario.dim
        x = reject_state.value

        stepsize = reject_extra.parameters.stepsize

        reject_extra.random_key, subkey, scen_key = random.split(reject_extra.random_key, 3)

        proposed_state.value = x + jnp.sqrt(stepsize) * random.normal(subkey, (d,))
        proposed_state.potential = scenario.potential(proposed_state.value, scen_key)

        return proposed_state, reject_extra

    def acceptance_probability(self,
                               scenario: Scenario,
                               reject_state: cdict, reject_extra: cdict,
                               proposed_state: cdict, proposed_extra: cdict) -> float:
        return jnp.minimum(1., jnp.exp(- proposed_state.potential + reject_state.potential))
```

Firstly, note that the `MCMCSampler` has a `correction` attribute. This represents a `mocat.Correction` object and determines the nature of the accept reject step. The three built-in corrections are

- `Uncorrected` - always accept proposal.

- `Metropolis` - accept proposal with probability determined by the `MCMCSampler`'s `acceptance_probability` method, otherwise duplicate previous sample.
- `RMMetropolis` - as above but additionally adapt the `stepsize` parameter with a Robbins-Monro schedule Andrieu and Thoms (2008) according to the `MCMCSampler`'s `tuning` attribute.

Of course `mocat` permits fully customisable inheritance from the `mocat.Correction` class in order to create alternative MCMC algorithms based on the same proposal.

There are therefore four key methods to define when inheriting `MCMCSampler`.

- `__init__` is the method that is called when an instance of the sampler is initiated. The `MCMCSampler` already creates a `parameters cdict`, the sampler's `__init__` is where the sampling algorithm parameters and their defaults are defined - in the case of `RandomWalk`, only the `stepsize` parameter. It is also the opportunity to set sampler defaults in the `tuning cdict`

```
rw_sampler = mocat.RandomWalk(stepsize=0.1)
rw_sampler.tuning
    mocat.cdict({'parameter': 'stepsize', 'target': 0.234,
     ↪  'metric': 'alpha', 'monotonicity': 'decreasing'})
```

- `startup` is called when the sampler is first exposed to the `Scenario`. The purpose of `startup` is to setup the `cdicts` - `initial_state` and `initial_extra`. When `jax.lax.scan` is called the sampler will adjust iterated `state` and `extra cdicts` were anything in `extra` will be discarded at the end (e.g. random keys) and anything in `state` will be stacked and returned. The `startup` ensures all attributes are initiated correctly to be consistent through `jax.lax.scan` - this includes, for example, initiating gradient evaluations. By default `initial_state` will only be initiated with an initial `value` and `initial_extra` with a `random_key` and iteration counter `iter`.
- `proposal` represents the function that modifies the `state` and `extra` at each iteration. If in addition it is desired to make iterative modifications that are always accepted, these can be applied in the `always` method that is applied before `proposal` (not required for `RandomWalk`).
- `acceptance_probability` determines the probability of accepting a proposal for when the `correction` is set to `Metropolis` (or an inheritance thereof).

We are then ready to sample!

```
mcmc_sample = mocat.run(funnel_scen, rw_sampler, n=100000,
↪  random_key=random.PRNGKey(0))
mcmc_sample.keys()
    dict_keys(['value', 'alpha', 'potential', 'time', 'summary'])
```

where the output is a `cdict` where the samples are stored in `value`, runtime in `time`, a summary of the run parameters in `summary` and any other attributes initiated by the `MCMCSampler` or its `Correction` in `startup`.

Alternatively we could have sampled with stepsize adaptation

```
rw_sampler = RandomWalk(stepsize=0.1)

mcmc_sample = mocat.run(funnel_scen, rw_sampler, n=100000,
↪  random_key=random.PRNGKey(0), correction=mocat.RMMetropolis)
mcmc_sample.keys()
    dict_keys(['value', 'stepsize', 'alpha', 'potential', 'time',
    ↪  'summary'])
```

Transport Sampling

`mocat` also provides a framework for Monte Carlo sampling under an alternative paradigm - that of iteratively updating a full particle approximation of fixed size, as opposed to gradually building a particle approximation of increasing size as in MCMC. In `mocat` samplers following this paradigm (such as sequential Monte Carlo with likelihood tempering) inherit the `TransportSampler` class.

A vanilla `TransportSampler` has four key methods to be implemented

- `__init__` as in `MCMCSampler`, initiate any sampler `parameters` and their default values however `TransportSampler` does not pre-initiate a `tuning` attribute.
- `startup` as in `MCMCSampler` except that `initial_state` now contains a full particle approximation and thus attributes (including `value` which by default is initiated with `n` calls to `scenario.prior_sample`) having a leading dimension of length `n`. `initial_extra` attributes remain singular in length.

- `update` modifies the particle approximation at each iteration.
- `termination_criterion` tells `mocat.run` when to stop.

`mocat` has built-in implementations of Stein Variational Gradient Descent (SVGD) Liu and Wang (2016) (`mocat.SVGD` with some basic kernels defined in `mocat.kernels`) as well as a customisable framework for sequential Monte Carlo with likelihood tempering Section 6.1 including

- `TemperedSMCSampler` a general class with `forward_proposal` and `log_weight` methods to be defined. Either requires a `temperature_schedule` at initiation or the method `next_temperature_adaptive` to be defined.
- `MetropolisedSMCSampler` - specifically for π_t-invariant transition kernels as in Chapter 6. Takes on initiation an `mcmc_sampler` argument which is an `MCMCSampler` object defining the nature the transition kernel. Supports a `temperature_schedule` or effective sample size based adaptive tempering.
- `RMMetropolisedSMCSampler` - as above but additionally adapts the stepsize of the `mcmc_sampler` using a Robbins-Monro schedule according to `mcmc_sampler.tuning` as in Algorithm 21.

We can again sample using `mocat.run`. For SVGD

```
svgd_sampler = mocat.SVGD(stepsize=0.1,
↪  kernel=mocat.kernels.Gaussian(), max_iter=1000)

svgd_sample = mocat.run(funnel_scen, svgd_sampler, n=100,
↪  random_key=random.PRNGKey(0))

svgd_sample.keys()
    dict_keys(['value', 'potential', 'grad_potential', 'time',
    ↪  'summary'])

svgd_sample.value.shape
    (1001, 100, 5)
```

or for sequential Monte Carlo with likelihood tempering

```
mcmc_sampler=mocat.Underdamped(leapfrog_steps=10, stepsize=0.1,
 ↪  friction=jnp.inf)

smc_sampler =
 ↪  mocat.RMMetropolisedSMCSampler(mcmc_sampler=mcmc_sampler)

smc_sample = mocat.run(funnel_scen, smc_sampler, n=10000,
 ↪  random_key=random.PRNGKey(0))

smc_sample.keys()
    dict_keys(['value', 'log_weight', 'ess', 'prior_potential',
     ↪  'grad_prior_potential', 'likelihood_potential',
     ↪  'grad_likelihood_potential', 'potential', 'grad_potential',
     ↪  'temperature', 'log_norm_constant', 'alpha', 'momenta',
     ↪  'stepsize', 'time', 'summary'])
```

Sample Metrics

`mocat` contains a collection of functions to analyse sample quality.

Let us analyse the `mcmc_sample` we generated from the `BayesFunnel` distribution using `RandomWalk`. The first thing we might check is some univariate marginals

```
mocat.hist_1d_samples(mcmc_sample, dim=0)
mocat.hist_1d_samples(mcmc_sample, dim=-1)
```

or perhaps the bivariate marginals without and with burn-in

```
mocat.plot_2d_samples(mcmc_sample, dim1=0, dim2=-1)
mocat.plot_2d_samples(mcmc_sample[1000:], dim1=0, dim2=-1)
```

Given that we modified the stepsize adaptively we should check our acceptance rates were appropriate

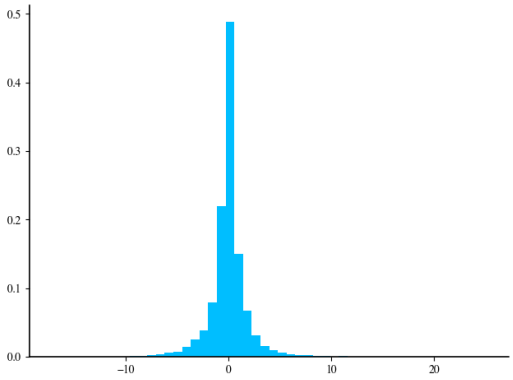


Fig. 7.1 `dim=0`

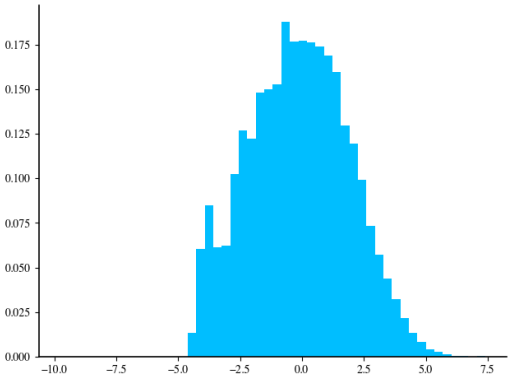


Fig. 7.2 `dim=-1`

Univariate marginals for MCMC samples from `BayesFunnel`.

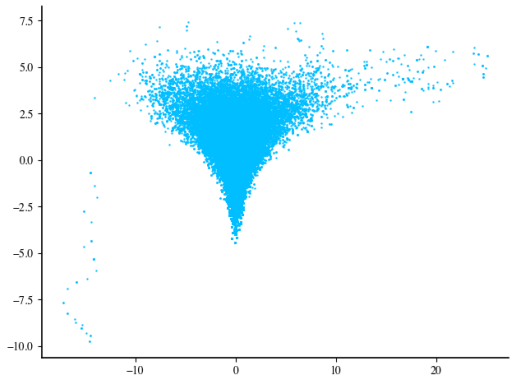


Fig. 7.3 No burn-in

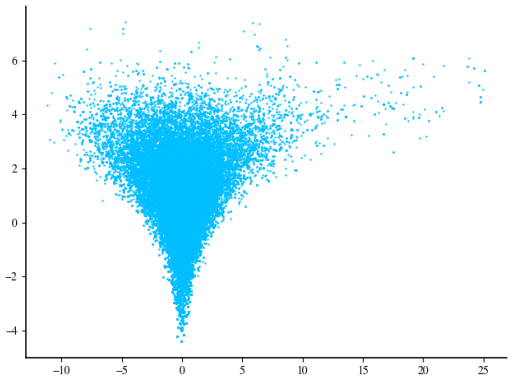


Fig. 7.4 Burn-in of 1000

Bivariate marginals for MCMC samples from `BayesFunnel`.

```
rw_sampler.tuning.target
    0.234

mcmc_sample.alpha.mean()
    DeviceArray(0.23056579, dtype=float32)
```

Lovely stuff. We can even visualise the stepsize adaptation (simply using `matplotlib` Hunter (2007))

```
import matplotlib.pyplot as plt

plt.plot(mcmc_sample.stepsize)
```

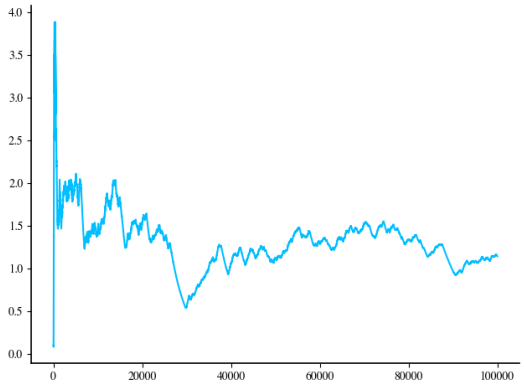


Fig. 7.5 `RandomWalk` stepsize adaptation on `BayesFunnel`

It wouldn't be Markov chain Monte Carlo without some trace plots

```
plt.plot(mcmc_sample.potential[1000:])
plt.plot(mcmc_sample.value[1000:, 0])
```

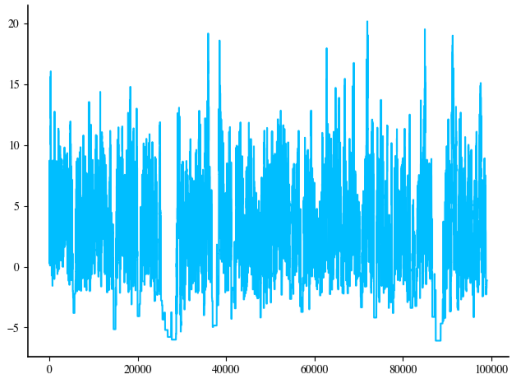


Fig. 7.6 `potential`

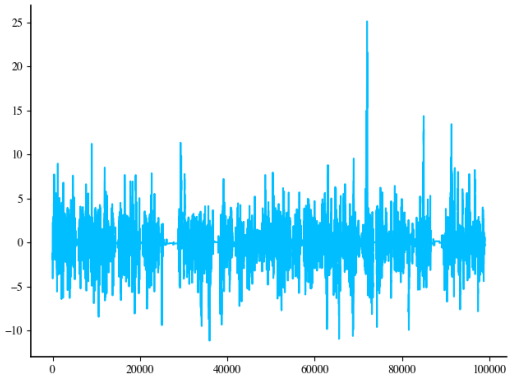


Fig. 7.7 `dim=0`

Trace plots for MCMC samples (with burn-in of 1000) from `BayesFunnel`.

and autocorrelations

```
mocat.autocorrelation_plot(mcmc_sample.potential[1000:])
mocat.autocorrelation_plot(mcmc_sample.value[1000:, 0])
```

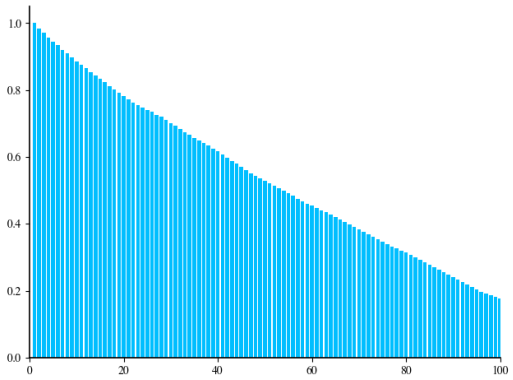


Fig. 7.8 `potential`

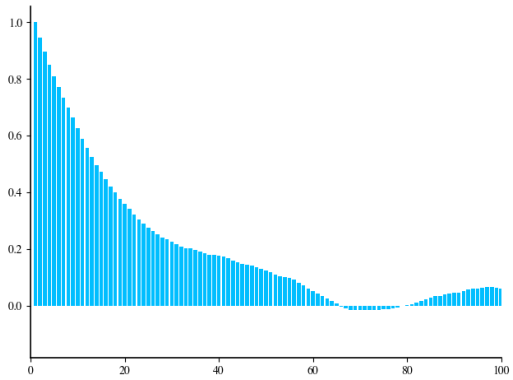


Fig. 7.9 `dim=0`

Autocorrelation plots for MCMC samples (with burn-in of 1000) from `BayesFunnel`.

Indeed we can also calculate the `ess_autocorrelation` (or `integrated_autocorrelation_time`)

```
mocat.ess_autocorrelation(mcmc_sample.potential)
    DeviceArray(8026.1, dtype=float32)

vmap(mocat.ess_autocorrelation)(mcmc_sample.value.T)
    DeviceArray([2079.1, 1665.7, 767.7, 647.3, 33951.8],
     ↪  dtype=float32)
```

In addition, we can analyse the quality of any sample (irrespective of its generating mechanism) using a kernelised Stein discrepancy (KSD) Liu et al. (2016) (which we mini-batch to avoid having to calculate a 100000x100000 gram matrix).

```
mcmc_sample.grad_potential =
 ↪  vmap(funnel_scen.grad_potential)(mcmc_sample.value)

mocat.ksd(mcmc_sample, kernel=mocat.kernels.Gaussian(),
 ↪  ensemble_batchsize=100, random_key=random.PRNGKey(0))
    DeviceArray(0.9774283, dtype=float32)
```

where we had to compute the gradient of the potential (required for KSDs) as it wasn't computed during sampling - this of course wouldn't have been the case if we had used `mocat.Underdamped`.

The kernelised Stein discrepancy is applicable to any sample including `TemperedSMCSamplers` as it also takes a *log_weight* argument. Remember that the output of a `TransportSampler` is a particle approximation at each iteration and so metrics should only be called on the final particle approximation

```
mocat.ksd(smc_sample[-1], kernel=mocat.kernels.Gaussian(),
 ↪  ensemble_batchsize=100, random_key=random.PRNGKey(0),
 ↪  log_weight=smc_sample.log_weight[-1])
```

Naturally the output of a `MetropolisedSMCSampler` also contains the attributes generated during sampling `ess`, `temperature`, `alpha`, `stepsize` and `log_norm_constant` that can be analysed to assess sample quality in addition to the marginal plots and KSD.

7.3 ABC

But what about problems where we cannot compute `likelihood_potential` but can can compute `likelihood_sample`? We can do exactly that using `mocat`'s submodule `abc`

```
from mocat import abc
```

We can define a target distribution with an intractable likelihood density by inheriting the `abc.ABCScenario` class

```
class GaussianABC(abc.ABCScenario):
    dim = 3
    prior_std = 5.
    likelihood_matrix = jnp.array([[1., -0.5, 2.],
                                   [-0.4, -0.1, 0.]])
    likelihood_std = 1.
    data = jnp.array([3., -1.])

    def prior_sample(self,
                     random_key: jnp.ndarray) -> jnp.ndarray:
        return self.prior_std * random.normal(random_key, (self.dim,))

    def prior_potential(self,
                        x: jnp.ndarray,
                        random_key: jnp.ndarray = None) -> float:
        return 0.5 * jnp.square(x / self.prior_std).sum()

    def likelihood_sample(self,
                          x: jnp.ndarray,
                          random_key: jnp.ndarray) -> jnp.ndarray:
        return self.likelihood_matrix @ x
               + self.likelihood_std * random.normal(random_key,
                ↪  shape=(self.likelihood_matrix.shape[0],))
```

If data is summarised, as is common in approximate Bayesian computation (ABC), this is defined implicitly in the `data` attribute and `likelihood_sample` method, i.e. `data` represents the summarised data and `likelihood_sample` directly simulates summarised synthetic data.

`abc.ABCScenario` additionally hosts a `distance_function` method that defaults to the Euclidean distance

```
def distance_function(self,
                      simulated_data: jnp.ndarray) -> float:
    return jnp.sqrt(jnp.square(simulated_data - self.data).sum())
```

`mocat.abc` has built-in implementations of the ABC algorithms described Section 2.5. Indeed `abc.VanillaABC` inherits `abc.ImportanceABC` and is jointly an importance and rejection sampler. The threshold parameter of `abc.VanillaABC` can be defined explicitly

```
vanilla_abc = abc.VanillaABC(threshold=3.)
```

or post-hoc via an acceptance rate

```
vanilla_abc = abc.VanillaABC(acceptance_rate=0.1)
```

and as usual we can run using `mocat.run`

```
gaussian_abc_scen = GaussianABC()

vanilla_abc_sample = mocat.run(gaussian_abc_scen, vanilla_abc,
↪  n=10000, random_key=random.PRNGKey(0))

(vanilla_abc_sample.log_weight == 0).mean()
    DeviceArray(0.1, dtype=float32)
```

where `vanilla_abc_sample.log_weight` represents an array of accept (0.) and reject (-inf) values.

`mocat.abc` also contains a fully customisable `ABCMCMCSampler` class and an implementation of `RandomWalkABC` which by default runs with `mocat.Metropolis` correction but can also adaptively determine both the stepsize and threshold parameters using the `RMMetropolisDiagStepsize` correction as described in Vihola and Franks (2020)

```
abc_mcmc_sampler = abc.RandomWalkABC()

abc_mcmc_sample = mocat.run(gaussian_abc_scen, abc_mcmc_sampler,
↪  n=10000, random_key=random.PRNGKey(0),
↪  correction=abc.RMMetropolisDiagStepsize())

abc_mcmc_sampler.tuning
    mocat.cdict({'parameter': 'threshold', 'target': 0.1, 'metric':
    ↪  'alpha', 'monotonicity': 1})

abc_mcmc_sample.alpha.mean()
    DeviceArray(0.08237284, dtype=float32)

plt.plot(abc_mcmc_sample.stepsize)
plt.plot(abc_mcmc_sample.threshold)
```

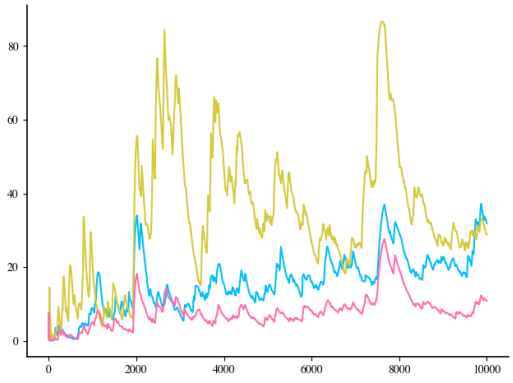


Fig. 7.10 Stepsize adaptation to match sample covariance in each dimension.

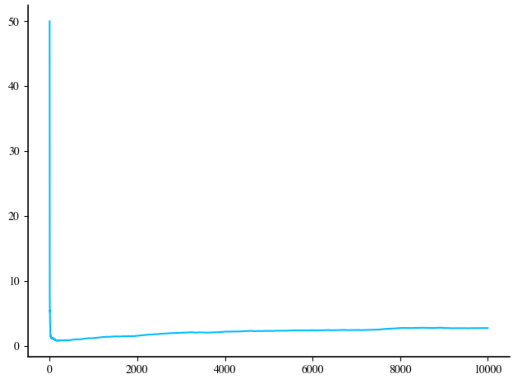


Fig. 7.11 Threshold adaptation

Adaptive ABC-MCMC on `GaussianABC`.

Similarly, `mocat.abc` implements ABC-SMC via a customisable `ABCSMCSampler` and `MetropolisedABCSMCSampler`. By default, `MetropolisedABCSMCSampler` uses an `abc.RandomWalkABC` proposal, an adaptive effective sample size based threshold schedule and sets the `MetropolisedABCSMCSampler.adapt_mcmc_params` to modify the stepsize (diagonal pre-conditioner) to the diagonal sample covariance scaled by $d/2.38^2$ as in Del Moral et al. (2012).

```
abc_smc_sampler = abc.MetropolisedABCSMCSampler()
abc_smc_sample = mocat.run(gaussian_abc_scen, abc_smc_sampler, n=1000,
 ↪  random_key=random.PRNGKey(0))

abc_smc_sample.keys()
    dict_keys(['value', 'log_weight', 'ess', 'prior_potential',
     ↪  'simulated_data', 'distance', 'threshold', 'alpha', 'time',
     ↪  'summary'])

plt.plot(abc_smc_sample.ess)
plt.plot(abc_smc_sample.alpha.mean(1))
plt.plot(abc_smc_sample.threshold)
plt.plot(abc_smc_sample.distance.mean(1))
```

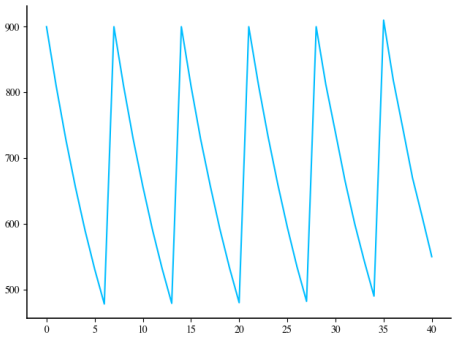


Fig. 7.12 Effective sample size

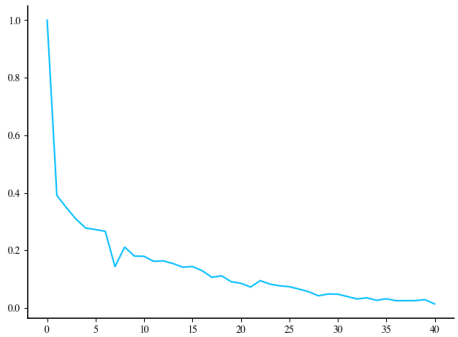


Fig. 7.13 Average Metropolis acceptance rate

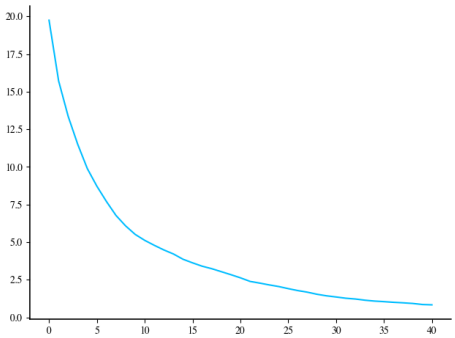


Fig. 7.14 Adaptive threshold parameter

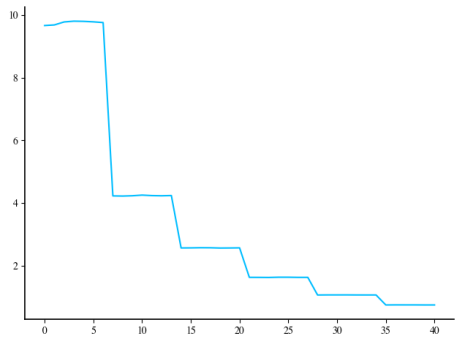


Fig. 7.15 Average distance between simulated and true data

Adaptive ABC-SMC on `GaussianABC`.

`mocat` also includes implementation of the tempered ensemble Kalman inversion (EKI) introduced in Chapter 5. Ensemble Kalman inversion represents a `TransportSampler` and only requires the `Scenario` to have `prior_sample` and `likelihood_sample` implemented. A general implementation is stored in `mocat.TemperedEKI` and EKI with adaptive temperature schedule using the pseudo-weights Equation (5.17) is found in `mocat.AdaptiveTemperedEKI`. The stopping criterion can be adjusted by modifying the `max_temperature` attribute and/or the `termination_criterion` method.

```
eki_sample = mocat.run(gaussian_abc_scen, mocat.AdaptiveTemperedEKI(),
↪  n=1000, random_key=random.PRNGKey(0))
```

7.4 State-space Models

Sequential Bayesian inference in state-space models Section 2.6 is also supported via `mocat`'s `ssm` submodule.

```
from mocat import ssm
```

A fully general state-space model can be stored by inheriting the `ssm.StateSpaceModel`. For example consider the univariate non-linear benchmark model Gordon et al. (1993)

```
class NonLinear1DBenchmark(ssm.StateSpaceModel):
    name = '1D Non-linear Benchmark'
    dim = 1
    dim_obs = 1

    def __init__(self,
                 initial_sd: float = jnp.sqrt(2.),
                 transition_sd: float = jnp.sqrt(10.),
                 likelihood_sd: float = 1.,
                 name: str = None):
        self.initial_sd = initial_sd
        self.transition_sd = transition_sd
        self.likelihood_sd = likelihood_sd
        super().__init__(name=name)

    def initial_potential(self,
                          x: jnp.ndarray,
                          t: float) -> float:
        return 0.5 * jnp.square(x / self.initial_sd).sum()

    def initial_sample(self,
                       t: float,
                       random_key: jnp.ndarray) -> jnp.ndarray:
        return random.normal(random_key, (1,)) * self.initial_sd

    def transition_potential(self,
                             x_previous: jnp.ndarray,
                             t_previous: float,
                             x_new: jnp.ndarray,
                             t_new: float) -> float:
        transition_mean = 0.5 * x_previous+ 25 * x_previous / (1 + x_previous ** 2) + 8 *
        ↪  jnp.cos(1.2 * t_previous)
        return 0.5 * jnp.square((x_new - transition_mean) / self.transition_sd).sum()

    def transition_sample(self,
                          x_previous: jnp.ndarray,
                          t_previous: float,
                          t_new: float,
                          random_key: jnp.ndarray) -> jnp.ndarray:
        transition_mean = 0.5 * x_previous+ 25 * x_previous / (1 + x_previous ** 2) + 8 *
        ↪  jnp.cos(1.2 * t_previous)
        return transition_mean + random.normal(random_key, (1,)) * self.transition_sd

    def likelihood_potential(self,
                             x: jnp.ndarray,
                             y: jnp.ndarray,
                             t: float) -> float:
        lik_mean = x ** 2 / 20
        return 0.5 * jnp.square((y - lik_mean)/self.likelihood_sd).sum()

    def likelihood_sample(self,
                          x: jnp.ndarray,
                          t: float,
                          random_key: jnp.ndarray) -> jnp.ndarray:
        lik_mean = x ** 2 / 20
        return lik_mean + random.normal(random_key, (1,)) * self.likelihood_sd
```

Observe the additional compulsory attribute `dim_obs` describing the dimension of a single observation. The state-space model comes down to the implementation of the following six methods

- `initial_potential` evaluates the potential corresponding to the first latent variable $p(x_0)$.
- `initial_sample` generates a single random sample from $p(x_0)$.
- `transition_potential` evaluates the potential corresponding to the transition density $p_t(x_t \mid x_{t-1})$ which may vary with t.
- `transition_sample` generates a single random sample from $p_t(x_t \mid x_{t-1})$.
- `likelihood_potential` evaluates the potential corresponding to the transition density $p_t(y_t \mid x_t)$ which may vary with t.
- `likelihood_sample` generates a synthetic observation from $p_t(y_t \mid x_t)$. Not necessary for the most basic inference in state-space models (i.e. `ssm.BootstrapFilter`).

Synthetic values of both the underlying trajectory and observations can then be generated

```
benchmark_ssm = NonLinear1DBenchmark()

simulated_values = benchmark_ssm.simulate(t_all=jnp.arange(10),
↪  random_key=random.PRNGKey(0))

simulated_values.keys()
    dict_keys(['x', 'y', 't', 'name'])
```

Linear Gaussian

As discussed in Equation (2.34), a convenient class of state-space models occurs when all distributions $p(x_0)$, $p_t(x_t \mid x_{t-1})$ and $p_t(y_t \mid x_t)$ are linear and Gaussian. `mocat` provides a customisable `ssm.LinearGaussian` class and `ssm.TimeHomogenousLinearGaussian` which reduces computational cost by assuming all of the matrices in the transition and likelihood are time homogeneous, i.e.

$$p_t(x_t \mid x_{t-1}) = p(x_t \mid x_{t-1}), \qquad p_t(y_t \mid x_t) = p(y_t \mid x_t).$$

```
lg_ssm = ssm.TimeHomogenousLinearGaussian(
              initial_mean=jnp.zeros(2),
              initial_covariance=jnp.eye(2),
              transition_matrix=jnp.eye(2),
              transition_covariance=0.2 * jnp.eye(2),
              likelihood_matrix=jnp.array([[0.5, 0.5]]),
              likelihood_covariance=0.3*jnp.eye(1))
```

Specifically for `ssm.LinearGaussian` models, we can run exact marginal filtering - Algorithm 10

```
t = jnp.arange(10)
y = random.normal(random.PRNGKey(0), shape=(10, 1))

filter_means, filter_covs =
 ↪  ssm.run_kalman_filter_for_marginals(lg_ssm, y, t)
```

and exact marginal smoothing Algorithm 11

```
smoother_means, smoother_covs =
 ↪  ssm.run_kalman_smoother_for_marginals(lg_ssm, y, t,
 ↪  filter_output=(filter_means, filter_covs))
```

Particle Methods

For more general state-space models such as the `NonLinear1DBenchmark` model defined above, we cannot do exact inference. Instead we can adopt the Monte Carlo approaches described in Section 2.6 and Chapter 3.

Underlying all of these particle methods is the concept of a particle filter Algorithm 12. A particle filter is defined by its a sequential proposal distribution $q(x_t \mid x_{t-1}, y_t)$ that is permitted to incorporate the new observation y_t. `mocat.ssm` provides a customisable `ParticleFilter` class to be inherited as well as a built-in implementation of the most basic `BootstrapFilter` described here

```
class BootstrapFilter(ParticleFilter):
    name = 'Bootstrap Filter'

    def proposal_potential(self,
                           ssm_scenario: StateSpaceModel,
                           x_previous: jnp.ndarray,
                           t_previous: float,
                           x_new: jnp.ndarray,
                           y_new: jnp.ndarray,
                           t_new: float) -> Union[float, jnp.ndarray]:
        return ssm_scenario.transition_potential(x_previous,
         ↪  t_previous, x_new, t_new)

    def proposal_sample(self,
                        ssm_scenario: StateSpaceModel,
                        x_previous: jnp.ndarray,
                        t_previous: float,
                        y_new: jnp.ndarray,
                        t_new: float,
                        random_key: jnp.ndarray) -> jnp.ndarray:
        return ssm_scenario.transition_sample(x_previous, t_previous,
         ↪  t_new, random_key)

    def intermediate_log_weight(self,
                                ssm_scenario: StateSpaceModel,
                                x_previous: jnp.ndarray,
                                t_previous: float,
                                x_new: jnp.ndarray,
                                y_new: jnp.ndarray,
                                t_new: float) -> Union[float,
                                 ↪  jnp.ndarray]:
        return -ssm_scenario.likelihood_potential(x_new, y_new, t_new)
```

where `intermediate_log_weight` refers to the function $h(x_{t-1}, x_t, y_t)$ that updates the logarithm of the (unnormalised) importance weights

$$\log w_t = \log w_{t-1} + h(x_{t-1}, x_t, y_t) + k \iff w_t \propto w_{t-1} e^{h(x_{t-1}, x_t, y_t)}.$$

`mocat.ssm` has built-in more efficient, informed particle filtering for the specific `NonLinearGaussian` class of state-space models Section 5.1, Godsill et al. (2004) where transitions and likelihoods are time-homogenous and take the form

$$p(x_t \mid x_{t-1}) = \mathbf{N}(x_t \mid f(x_{t-1}), \mathrm{R}),$$
$$p(y_t \mid x_t) = \mathbf{N}(y_t \mid \mathrm{H}x_t, \mathrm{Q}).$$

In this case the (locally) optimal proposal Equation (2.39) is tractable and an implementation is provided in `ssm.OptimalNonLinearGaussianParticleFilter`. Additionally these models are amenable to ensemble Kalman filtering Algorithm 19 and an implementation is stored in `ssm.EnsembleKalmanFilter`.

Now for a given `ParticleFilter` we can tackle inference.

Online

For online particle filtering (for marginals $p(x_T \mid y_{0:T})$ we can initiate a particle approximation at the first observation

```
y, t = simulated_values.y, simulated_values.t

pf = ssm.BootstrapFilter()

random_key, key0 = random.split(random.PRNGKey(0))
filter_marginal = ssm.initiate_particles(benchmark_ssm, pf, n=1000,
↪  random_key=key0, y=y[0], t=t[0])

filter_marginal.keys()
    dict_keys(['value', 'log_weight', 't', 'y', 'ess'])
```

and then update for new observations

```
random_key, key1 = random.split(random_key)

filter_marginal = ssm.propagate_particle_filter(benchmark_ssm, pf,
↪  filter_marginal, y_new=y[1], t_new=t[1], random_key=key1)
```

which by default will append the proposal to the trajectory - the most recent values can be extracted with `filter_marginal[-1]`.

Additionally, we can generate an online particle approximation to the full joint smoothing distribution $p(x_{0:T} \mid y_{0:T})$ using the techniques from Chapter 3

```
smoothing_joint = ssm.initiate_particles(benchmark_ssm, pf, n=1000,
↪  random_key=random.PRNGKey(0), y=y[0], t=t[0])

smoothing_joint = ssm.propagate_particle_smoother(benchmark_ssm, pf,
↪  filter_marginal, y_new=y[1], t_new=t[1], random_key=key1, lag=5)
```

which will execute the online smoother with backward simulation Algorithm 17, setting `backward_sim=False` in `propagate_particle_smoother` will run the online smoother with particle filter block proposal Algorithm 16.

Offline

In the case we have all of the observations at once, we can execute forward filtering-backward simulation Algorithm 13 to generate a particle approximation to the smoothing distribution $p(x_{0:T} \mid y_{0:T})$. We can do this either via an explicit two-step procedure

```
random_key, key0, key1 = random.split(random.PRNGKey(0), 3)

filtering_marginals =
↪  ssm.run_particle_filter_for_marginals(benchmark_ssm, pf, y, t,
↪  n=1000, random_key=key0)

smoothing_joint = ssm.backward_simulation(benchmark_ssm,
↪  filtering_marginals, key1)
```

or all at once

```
smoothing_joint =
 ↪  ssm.forward_filtering_backward_simulation(benchmark_ssm, pf, y, t,
 ↪  n_samps=100, random_key=random.PRNGKey(0))

smoothing_joint.keys()
    dict_keys(['value', 't', 'y', 'ess', 'num_transition_evals',
     ↪  'time', 'pf_time', 'bsi_time'])

plt.plot(smoothing_joint.t, smoothing_joint.value[:, :, 0],
 ↪  c='orange')
plt.plot(simulated_values.t, simulated_values.x, c='red')
```

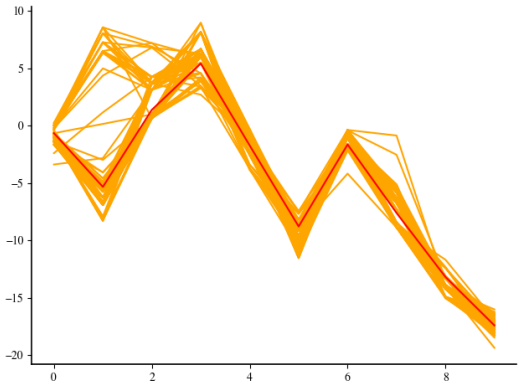


Fig. 7.16 FFBSi applied to `NonLinear1DBenchmark`

Chapter 8

Conclusions

This thesis has presented new Monte Carlo methods for a range of Bayesian inference problems. In particular, we have focused on a sequential approach where particle approximations are iteratively updated. In online settings an iterative procedure is desirable in order to update approximations as new observations arrive. In offline settings iterative procedures are applied, often with adaptation, to ensure the difference between successive target distributions is small and therefore increasing numerical stability

We conclude by summarising these contributions and providing insights into future directions.

8.1 Contributions

In Chapter 3, we introduced a novel online particle smoothing framework that provides a solution to the well-known problem of path degeneracy in the field of particle filtering. That is, the online smoothers approximate the full joint smoothing distribution $p(x_{0:T} \mid y_{0:T})$ in an online fashion with particles that do not degenerate as the length of the state-space model increases. The framework combines a fixed-lag approximation Kitagawa and Sato (2001), ideas from block sampling Doucet et al. (2006) and a newly introduced *particle stitching* algorithm (15) that represents a forward implementation of the technique underlying backward simulation Godsill et al. (2004).

Chapter 4 derives a new state-space model for the problem of map-matching specifically in dense, urban road networks. Map-matching is a particularly compelling application of the online smoothing algorithms from Chapter 3, as the continuity of trajectories makes particle approximations to the smoothing distribution $p(x_{0:T} \mid y_{0:T})$ significantly more useful than filtering $p(x_T \mid y_{0:T})$ or smoothing $p(x_t \mid y_{0:T})$ marginals that do not enforce continuity. We describe how to tune the parameters of the state-space model using data-driven expectation-

maximisation and demonstrate the benefits of uncertainty quantification for map-matching over point estimate based approaches. In order to make map-matching with uncertainty quantification in both online and offline settings more accessible we developed the open source `python` package `bmm` and described its functionality.

In Chapter 5, we shift focus to the difficult task of inference in static Bayesian inference problems where we cannot evaluate the likelihood function, only generate synthetic data. We generalise ideas from numerically efficient ensemble Kalman methods and describe a fully adaptive implementation for both sampling and optimisation. We demonstrate a significant speedup in comparison to state-of-the-art approximate Bayesian computation techniques.

Chapter 6 considers the more standard setup of static Bayesian inference where the likelihood is tractable and furthermore we have access to the gradient of the target potential. We introduce a fully adaptive sequential Monte Carlo sampler that utilises a Hessian approximation to efficiently pre-condition proposals and better grasp the local scaling of the tempered posterior. The Hessian approximation adapts the L-BFGS algorithm which represents the state-of-the-art in non-stochastic optimisation. The L-BFGS Hessian approximation utilises previous states of a particle's trajectory and is extremely cheap in the dimension of the inference problem - we demonstrate the benefits of the Hessian pre-conditioning in difficult and high-dimensional Bayesian inference problems.

Finally we developed `mocat`, a general purpose `python` package using `JAX` for extremely fast and accessible implementations of all of the discussed and introduced techniques, with complete flexibility to adapt existing algorithms and create new ones.

8.2 Future Directions

It would be desirable to obtain theoretical guarantees bounding the error induced by thhe online smoothing algorithms in Chapter 3. This is left as future work as we anticipate this analysis to be somewhat intricate. It would involve combining the work on central limit theorems for particle smoothing such as Del Moral et al. (2010); Douc et al. (2011), the bias introduced by a fixed-lag approximation Olsson et al. (2008) as well as quantifying the impact of using the tractable weights with overlapping coordinate from Section 3.2.2 as opposed to the optimal but intractable weights in Section 3.2.1.

Additionally, the development of an adaptive procedure to determine a suitable lag parameter, similar to that in Alenlöv and Olsson (2019) (for marginal online particle smoothing), would result in a fully adaptive algorithm where we are only left to choose the number of particles N.

The nature of the introduced online smoothing algorithms is extremely general - they provide online approximations to the full posterior distribution of generic state-space models. For this reason, we expect future applications in a wide range of fields such as alternative tracking settings Gustafsson et al. (2002) or financial inference Creal (2012).

An outstanding question regarding the ensemble Kalman inversion introduced in Chapter 5, is understanding its asymptotic behaviour numerically as the sample size N increases. The first step would be to analyse this behaviour in further examples and under alternative metrics. However, the lack of asymptotic results quantifying the bias for generic target distributions (i.e. non-Gaussian) is apparent for all ensemble Kalman methods.

Similarly to the online smoothers, a key contribution of our novel ensemble Kalman inversion algorithm is that it is extremely general. The only requirement is the ability to generate samples from the prior and synthetic data from the likelihood for a given value of the parameter, as such a wide range of applications are available. The benefits of the ensemble Kalman approach are expected to be greatest in cases where the state dimension is high and the posterior distribution is somewhat smooth such as Bayesian inverse problems Dashti and Stuart (2017) and the training of neural networks Kovachki and Stuart (2019) where under the introduced generalisation we can now consider more general models with non-Gaussian noise.

Both the ensemble Kalman inversion in Chapter 5 and the quasi-Newton SMC sampler in Chapter 6 have potential to be embedded within a sequential Monte Carlo sampler (or ABC equivalent) with intermediate weights of the form

$$w_t = w_{t-1} \frac{\pi_t(x_t)\pi_{t-1|t}(x_{t-1} \mid x_t)}{\pi_{t-1}(x_{t-1})q_{t|t-1}(x_t \mid x_{t-1})},$$

where $q_{t|t-1}$ represents the EKI or quasi-Newton kernel and $\pi_{t-1|t}$ is a backward balancing kernel to be defined. In the case of EKI the backward kernel is difficult to define as ideas of likelihood tempering and adaptive ABC thresholding become somewhat convoluted. The quasi-Newton setup is somewhat simpler and we can use Taylor expansions to derive logical forward and backward kernels (e.g. that would be exact for Gaussian targets) in the spirit of Titsias and Papaspiliopoulos (2016). However, the accept-reject step adds an additional layer of numerical stability (as outlandish proposals are simply rejected) which is particularly prevalent when an additional approximation is used, in this case the L-BFGS Hessian approximation.

The last comment is on parallel programming. The fastest modern computer architures such as GPUs and TPUs - on which `mocat` (Chapter 7) and `JAX` are naturally amenable to - obtain their speed by dividing tasks amongst many processors, i.e. in parallel. It is a great strength of sequential Monte Carlo methods (and indeed ensemble Kalman methods) over Markov chain Monte Carlo techniques that they are inherently parallelisable. Investigating the

extent of the computational gains over Markov chain Monte Carlo techniques with the use of parallel architectures would be extremely valuable. Leveraging this notion further, it would be interesting to experiment with modifications of the sequential Monte Carlo and ensemble Kalman techniques that are *embarrassingly parallel* (i.e. without any communication amongst particles) or close to. This would prevent the use of traditional resampling and in the case of likelihood tempering would amount to investigating stochastic numerical solutions to the Fokker-Planck equation that describes the probabilistic evolution of a particle from prior to posterior.

References

Agarwal, N., Bullins, B., and Hazan, E. (2017). Second-Order Stochastic Optimization for Machine Learning in Linear Time. *J. Mach. Learn. Res.*, 18(1):4148–4187.

Alenlöv, J. and Olsson, J. (2019). Particle-Based Adaptive-Lag Online Marginal Smoothing in General State-Space Models. *IEEE Transactions on Signal Processing*, 67(21):5571–5582.

Anderson, J. L. (01 Dec. 2001). An Ensemble Adjustment Kalman Filter for Data Assimilation. *Monthly Weather Review*, 129(12):2884 – 2903.

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

Andrieu, C. and Thoms, J. (2008). A Tutorial on Adaptive MCMC. *Statistics and Computing*, 18(4):343–373.

Barker, A. A. (1965). Monte Carlo Calculations of the Radial Distribution Functions for a Dense Proton-Electron Plasma. *Australian Journal of Physics*.

Beaumont, M. A., Zhang, W., and Balding, D. J. (2002). Approximate Bayesian Computation in Population Genetics. *Genetics*, 162(4):2025–2035.

Beskos, A., Pillai, N., Roberts, G., Sanz-Serna, J.-M., and Stuart, A. (2013). Optimal Tuning of the Hybrid Monte Carlo Algorithm. *Bernoulli*, 19(5A):1501–1534.

Betancourt, M. (2012). Cruising the Simplex: Hamiltonian Monte Carlo and the Dirichlet Distribution.

Biermann, C. (2019). *Football Hackers: The Science and Art of a Data Revolution*. Blink Publishing.

Bishop, C. H., Etherton, B. J., and Majumdar, S. J. (01 Mar. 2001). Adaptive Sampling with the Ensemble Transform Kalman Filter. Part I: Theoretical Aspects. *Monthly Weather Review*, 129(3):420 – 436.

Boeing, G. (2017). OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: Composable transformations of Python+NumPy programs.

Buchholz, A., Chopin, N., and Jacob, P. E. (2021). Adaptive Tuning of Hamiltonian Monte Carlo Within Sequential Monte Carlo. *Bayesian Analysis*, pages 1 – 27.

Burgers, G., van Leeuwen, P. J., and Evensen, G. (01 Jun. 1998). Analysis Scheme in the Ensemble Kalman Filter. *Monthly Weather Review*, 126(6):1719 – 1724.

Chatterjee, S. and Diaconis, P. (2018). The Sample Size Required in Importance Sampling. *Ann. Appl. Probab.*, 28(2):1099–1135.

Chopin, N. (2002). A Sequential Particle Filter Method for Static Models. *Biometrika*, 89(3):539–552.

Chopin, N., Lelièvre, T., and Stoltz, G. (2012). Free Energy Methods for Bayesian Inference: Efficient Exploration of Univariate Gaussian Mixture Posteriors. *Statistics and Computing*, 22(4):897–916.

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

Clapp, T. and Godsill, S. (1999). Fixed-lag Smoothing using Sequential Importance Sampling.

Creal, D. (2012). A Survey of Sequential Monte Carlo Methods for Economics and Finance. *Econometric Reviews*, 31(3):245–296.

Dashti, M. and Stuart, A. M. (2017). *The Bayesian Approach to Inverse Problems*, pages 311–428. Springer International Publishing, Cham.

Davidson, P., Collin, J., and Takala, J. (2011). Application of Particle Filters to a Map-matching Algorithm. *Gyroscopy and Navigation*, 2(4):285.

Del Moral, P. and Doucet, A. (2003). *On a Class of Genealogical and Interacting Metropolis Models*, pages 415–446. Springer New York.

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.

Del Moral, P., Doucet, A., and Jasra, A. (2012). An Adaptive Sequential Monte Carlo method for Approximate Bayesian Computation. *Statistics and Computing*, 22(5):1009–1020.

Del Moral, P., Doucet, A., and Singh, S. S. (2010). A Backward Particle Interpretation of Feynman-Kac Formulae. *ESAIM: M2AN*, 44(5):947–975.

Douc, R. and Cappe, O. (2005). Comparison of Resampling Schemes for Particle Filtering. In *ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005.*, pages 64–69.

Douc, R., Garivier, A., Moulines, E., and Olsson, J. (2011). Sequential Monte Carlo Smoothing for General State Space Hidden Markov Models. *The Annals of Applied Probability*, 21(6):2109–2145.

Douc, R., Moulines, E., Priouret, P., and Soulier, P. (2018). *Markov chains*. Operation research and financial engineering. Springer.

Doucet, A., Briers, M., and Sénécal, S. (2006). Efficient Block Sampling Strategies for Sequential Monte Carlo Methods. *Journal of Computational and Graphical Statistics*, 15(3):693–711.

Doucet, A., Freitas, N., Murphy, K., and Russell, S. (2013). Sequential Monte Carlo Methods in Practice.

Doucet, A., Godsill, S., and Andrieu, C. (2000). On Sequential Monte Carlo Sampling Methods for Bayesian Filtering. *Statistics and Computing*, 10(3):197–208.

Drovandi, C. C. and Pettitt, A. N. (2011). Likelihood-free Bayesian Estimation of Multivariate Quantile Distributions. *Computational Statistics and Data Analysis*, 55(9):2541–2556.

Elvira, V., Míguez, J., and Djurić, P. M. (2017). Adapting the Number of Particles in Sequential Monte Carlo Methods Through an Online Scheme for Convergence Assessment. *IEEE Transactions on Signal Processing*, 65(7):1781–1794.

Evensen, G. (1994). Sequential Data Assimilation with a Nonlinear Quasi-geostrophic Model using Monte Carlo Methods to Forecast Error Statistics. *Journal of Geophysical Research: Oceans*, 99(C5):10143–10162.

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.

Fisher, R. A. (1954). *Statistical Methods for Research Workers*. Oliver and Boyd, Edinburgh.

Fu, T., Luo, L., and Zhang, Z. (2016). Quasi-Newton Hamiltonian Monte Carlo. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, UAI'16, page 212–221, Arlington, Virginia, USA. AUAI Press.

Gelman, A. and Meng, X.-L. (1998). Simulating Normalizing Constants: From Importance Sampling to Bridge Sampling to Path Sampling. *Statistical Science*, 13(2):163 – 185.

Gelman, A., Vehtari, A., Simpson, D., Margossian, C. C., Carpenter, B., Yao, Y., Kennedy, L., Gabry, J., Bürkner, P.-C., and Modrák, M. (2020). Bayesian Workflow.

Geman, S. and Geman, D. (1984). Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741.

Gilks, W. R. and Berzuini, C. (2001). Following a Moving Target — Monte Carlo Inference for Dynamic Bayesian Models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(1):127–146.

Girolami, M. and Calderhead, B. (2011). Riemann Manifold Langevin and Hamiltonian Monte Carlo Methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214.

Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering*. Applications of mathematics : stochastic modelling and applied probability. Springer.

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.

Goh, C., Dauwels, J., Mitrovic, N., Asif, M. T., Oran, A., and Jaillet, P. (2012). Online Map-matching based on Hidden Markov Model for Real-time Traffic Sensing Applications. pages 776–781.

Gordon, N., Salmond, D., and Smith, A. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEEE Proceedings F, Radar and Signal Processing*, 140(2):107–113.

Gramacy, R., Samworth, R., and King, R. (2010). Importance Tempering. *Statistics and Computing*, 20(1):1–7.

Gustafsson, F., Gunnarsson, F., Bergman, N., Forssell, U., Jansson, J., Karlsson, R., and Nordlund, P. . (2002). Particle Filters for Positioning, Navigation, and Tracking. *IEEE Transactions on Signal Processing*, 50(2):425–437.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring Network Structure, Dynamics, and Function using NetworkX. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA.

Hansen, L. P. (1982). Large Sample Properties of Generalized Method of Moments Estimators. *Econometrica*, 50(4):1029–1054.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Hastings, W. K. (1970). Monte Carlo Sampling Methods using Markov Chains and their Applications. *Biometrika*, 57(1):97–109.

Horowitz, A. M. (1991). A Generalized Guided Monte Carlo Algorithm. *Phys. Lett. B*, 268:247–252.

Houtekamer, P. L. and Mitchell, H. L. (01 Jan. 2001). A Sequential Ensemble Kalman Filter for Atmospheric Data Assimilation. *Monthly Weather Review*, 129(1):123 – 137.

Houtekamer, P. L. and Mitchell, H. L. (01 Mar. 1998). Data Assimilation Using an Ensemble Kalman Filter Technique. *Monthly Weather Review*, 126(3):796 – 811.

Houtekamer, P. L., Mitchell, H. L., Pellerin, G., Buehner, M., Charron, M., Spacek, L., and Hansen, B. (01 Mar. 2005). Atmospheric Data Assimilation with an Ensemble Kalman Filter: Results with Real Observations. *Monthly Weather Review*, 133(3):604 – 620.

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3):90–95.

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.

Iglesias, M., Park, M., and Tretyakov, M. V. (2018). Bayesian Inversion in Resin Transfer Molding. *Inverse Problems*, 34(10):105002.

Iglesias, M. and Yang, Y. (2020). Adaptive Regularisation for Ensemble Kalman Inversion.

Iglesias, M. A. (2015). Iterative Regularization for Ensemble Data Assimilation in Reservoir Models. *Computational Geosciences*, 19(1):177–212.

Iglesias, M. A., Law, K. J. H., and Stuart, A. M. (2013). Ensemble Kalman Methods for Inverse Problems. *Inverse Problems*, 29(4):045001.

Izenman, A. J. and Sommer, C. J. (1988). Philatelic Mixtures and Multimodal Densities. *Journal of the American Statistical Association*, 83(404):941–953.

Jasra, A., Singh, S. S., Martin, J. S., and McCoy, E. (2012). Filtering via Approximate Bayesian Computation. *Statistics and Computing*, 22(6):1223–1237.

Jasra, A., Stephens, D. A., Doucet, A., and Tsagaris, T. (2011). Inference for Lévy-Driven Stochastic Volatility Models via Adaptive Sequential Monte Carlo. *Scandinavian Journal of Statistics*, 38(1):1–22.

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45.

Kempinska, K., Davies, T., and Shawe-Taylor, J. (2016). Probabilistic Map-matching using Particle Filters. *arXiv e-prints*, page arXiv:1611.09706.

Kitagawa, G. and Sato, S. (2001). *Monte Carlo Smoothing and Self-Organising State-Space Model*, pages 177–195. Springer New York, New York, NY.

Klaas, M., Freitas, N. d., and Doucet, A. (2005). Toward Practical N^2 Monte Carlo: The Marginal Particle Filter. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 308–315, Arlington, Virginia, USA. AUAI Press.

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.

Kovachki, N. B. and Stuart, A. M. (2019). Ensemble Kalman Inversion: A Derivative-free Technique for Machine Learning Tasks. *Inverse Problems*, 35(9):095005.

Lam, S. K., Pitrou, A., and Seibert, S. (2015). Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6.

Landau, D. and Binder, K. (2005). *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, USA.

Le Gland, F., Monbet, V., and Tran, V.-D. (2009). Large Sample Asymptotics for the Ensemble Kalman Filter. Research Report RR-7014, INRIA.

Lei, J. and Bickel, P. (2011). A Moment Matching Ensemble Filter for Nonlinear Non-Gaussian Data Assimilation. *Monthly Weather Review*, 139:3964–3973.

Leimkuhler, B. and Matthews, C. (2015). *Molecular Dynamics: With Deterministic and Stochastic Numerical Methods*. Interdisciplinary Applied Mathematics. Springer.

Leimkuhler, B., Matthews, C., and Weare, J. (2018). Ensemble Preconditioning for Markov Chain Monte Carlo Simulation. *Statistics and Computing*, 28(2):277–290.

Li, T., Bolic, M., and Djuric, P. M. (2015). Resampling Methods for Particle Filtering: Classification, Implementation, and Strategies. *IEEE Signal Processing Magazine*, 32(3):70–86.

Lindsten, F., Bunch, P., Singh, S. S., and Schön, T. B. (2015). Particle Ancestor Sampling for Near-degenerate or Intractable State Transition Models.

Liu, J. S., Chen, R., and Wong, W. H. (1998). Rejection Control and Sequential Importance Sampling. *Journal of the American Statistical Association*, 93(443):1022–1031.

Liu, Q., Lee, J., and Jordan, M. (2016). A Kernelized Stein Discrepancy for Goodness-of-fit Tests. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 276–284, New York, New York, USA. PMLR.

Liu, Q. and Wang, D. (2016). Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Lorenz, E. (1995). Predictability: A Problem Partly Solved. In *Seminar on Predictability, 4-8 September 1995*, volume 1, pages 1–18, Shinfield Park, Reading. ECMWF, ECMWF.

Ma, Y.-A., Chen, T., and Fox, E. (2015). A Complete Recipe for Stochastic Gradient MCMC. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Marchi, M., Albert, J., and Baumer, B. S. (2018). *Analyzing Baseball Data with R*. Chapman and Hall/CRC.

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.

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.

Meyn, S. and Tweedie, R. (1993). *Markov Chains and Stochastic Stability*. Springer-Verlag.

Minecraft Speedrunning Team (2020). Response to Critique of Dream Investigation Results.

Moreira-Matias, L., Gama, J., Ferreira, M., Moreira, J., and Damas, L. (2013). Predicting Taxi-Passenger Demand Using Streaming Data. *IEEE Transactions on Intelligent Transportation Systems*, 14:1393–1402.

Neal, R. M. (2001). Annealed Importance Sampling. *Statistics and Computing*, 11(2):125–139.

Neal, R. M. (2010). MCMC Using Hamiltonian Dynamics. *Handbook of Markov Chain Monte Carlo*, 54:113–162.

Newson, P. and Krumm, J. (2009). Hidden Markov Map Matching through Noise and Sparseness. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '09, page 336–343, New York, NY, USA. Association for Computing Machinery.

Nguyen, T., Septier, F., Peters, G., and Delignon, Y. (2016). Efficient Sequential Monte-Carlo Samplers for Bayesian Inference. *IEEE Transactions on Signal Processing*, 64(5):1305–1319.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York.

Nott, D. J., Marshall, L., and Ngoc, T. M. (2012). The Ensemble Kalman Filter is an ABC Algorithm. *Statistics and Computing*, 22(6):1273–1276.

Olsson, J., Cappé, O., Douc, R., and Éric Moulines (2008). Sequential Monte Carlo Smoothing with Application to Parameter Estimation in Nonlinear State Space Models. *Bernoulli*, 14(1):155 – 179.

Olsson, J. and Westerborn, J. (2017). Efficient Particle-based Online Smoothing in General Hidden Markov models: The PaRIS Algorithm. *Bernoulli*, 23(3):1951–1996.

OpenStreetMap contributors (2017). Planet Dump Retrieved from https://planet.osm.org.

Peskun, P. H. (1973). Optimum Monte-Carlo Sampling using Markov Chains. *Biometrika*, 60(3):607–612.

Peters, G. W., Chen, W. Y., and Gerlach, R. H. (2016). Estimating Quantile Families of Loss Distributions for Non-Life Insurance Modelling via L-Moments. *Risks*, 4(2).

Pitt, M. K. and Shephard, N. (1999). Filtering via Simulation: Auxiliary Particle Filters. *Journal of the American Statistical Association*, 94(446):590–599.

Pritchard, J., Seielstad, M., Perez-Lezaun, A., and Feldman, M. (1999). Population Growth of Human Y Chromosomes: A Study of Y Chromosome Microsatellites. *Molecular biology and evolution*, 16(12):1791—1798.

PyPI (2021). Python Package Index - PyPI.

Raymond, R., Morimura, T., Osogami, T., and Hirosue, N. (2012). Map Matching with Hidden Markov Model on Sampled Road Network. pages 2242–2245.

Rayner, G. D. and MacGillivray, H. L. (2002). Numerical Maximum Likelihood Estimation for the g-and-k and Generalized g-and-h Distributions. *Statistics and Computing*, 12(1):57–75.

Reich, S. and Cotter, C. (2015). *Probabilistic Forecasting and Bayesian Data Assimilation*. Cambridge University Press.

Roberts, G. O. and Rosenthal, J. S. (2001). Optimal scaling for various Metropolis-Hastings algorithms. *Statistical Science*, 16(4):351 – 367.

Rosenthal, J. S. (2009). Optimal Proposal Distributions and Adaptive MCMC. *Handbook of Markov Chain Monte Carlo*.

Roth, M., Gustafsson, F., and Orguner, U. (2012). On-road Trajectory Generation from GPS data: A Particle Filtering/Smoothing Application. In *2012 15th International Conference on Information Fusion*, pages 779–786.

Roth, M., Hendeby, G., Fritsche, C., and Gustafsson, F. (2017). The Ensemble Kalman Filter: A Signal Processing Perspective. *EURASIP Journal on Advances in Signal Processing*, 2017(1):56.

Sabne, A. (2020). XLA : Compiling Machine Learning for Peak Performance.

Schraudolph, N. N., Yu, J., and Günter, S. (2007). A Stochastic Quasi-Newton Method for Online Convex Optimization. In Meila, M. and Shen, X., editors, *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pages 436–443, San Juan, Puerto Rico. PMLR.

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.

Taghavi, E., Lindsten, F., Svensson, L., and Schön, T. B. (2013). Adaptive Stopping for Fast Particle Smoothing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6293–6297.

Tavaré, S., Balding, D. J., Griffiths, R. C., and Donnelly, P. (1997). Inferring Coalescence Times From DNA Sequence Data. *Genetics*, 145(2):505–518.

Tierney, L. (1998). A note on Metropolis-Hastings Kernels for General State Spaces. *The Annals of Applied Probability*, 8(1):1 – 9.

Titsias, M. and Papaspiliopoulos, O. (2016). Auxiliary Gradient-based Sampling Algorithms. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 80.

van Dyk, D. A. (2014). The Role of Statistics in the Discovery of a Higgs Boson. *Annual Review of Statistics and Its Application*, 1(1):41–59.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Vihola, M. and Franks, J. (2020). On the Use of Approximate Bayesian Computation Markov Chain Monte Carlo with Inflated Tolerance and Post-correction. *Biometrika*, 107(2):381–395.

Wang, Y., Deng, W., and Lin, G. (2021). An Adaptive Hessian Approximated Stochastic Gradient MCMC Method. *Journal of Computational Physics*, 432:110150.

Wilkinson, D. J. (2018). *Stochastic Modelling for Systems Biology*. CRC Press.

Wood, S. N. (2010). Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104.

Young, G. A. and Smith, R. L. (2005). *Essentials of Statistical Inference*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press.

Zhang, Y. and Sutton, C. (2011). Quasi-Newton Methods for Markov Chain Monte Carlo. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.