Bayesian Learning of Continuous Time Dynamical Systems

with

Applications in Functional Magnetic Resonance Imaging

Lawrence Murray

Doctor of Philosophy

Institute for Adaptive and Neural Computation

School of Informatics

University of Edinburgh

2008

Abstract

Temporal phenomena in a range of disciplines are more naturally modelled in continuous-time than coerced into a discrete-time formulation. Differential systems form the mainstay of such modelling, in fields from physics to economics, geoscience to neuroscience. While powerful, these are fundamentally limited by their determinism. For the purposes of probabilistic inference, their extension to stochastic differential equations permits a continuous injection of noise and uncertainty into the system, the model, and its observation.

This thesis considers Bayesian filtering for state and parameter estimation in general non-linear, non-Gaussian systems using these stochastic differential models. It identifies a number of challenges in this setting over and above those of discrete time, most notably the absence of a closed form transition density. These are addressed via a synergy of diverse work in numerical integration, particle filtering and high performance distributed computing, engineering novel solutions for this class of model.

In an area where the default solution is linear discretisation, the first major contribution is the introduction of higher-order numerical schemes, particularly stochastic Runge-Kutta, for more efficient simulation of the system dynamics. Improved runtime performance is demonstrated on a number of problems, and compatibility of these integrators with conventional particle filtering and smoothing schemes discussed.

Finding compatibility for the smoothing problem most lacking, the major theoretical contribution of the work is the introduction of two novel particle methods, the *kernel forward-backward* and *kernel two-filter* smoothers. By harnessing kernel density approximations in an importance sampling framework, these attain cancellation of the intractable transition density, ensuring applicability in continuous time. The use of kernel estimators is particularly amenable to parallelisation, and provides broader support for smooth densities than a sample-based representation alone, helping alleviate the well known issue of degeneracy in particle smoothers.

Implementation of the methods for large-scale problems on high performance computing architectures is provided. Achieving improved temporal and spatial complexity, highly favourable runtime comparisons against conventional techniques are presented.

Finally, attention turns to real world problems in the domain of Functional Magnetic Resonance Imaging (fMRI), first constructing a biologically motivated stochastic differential model of the neural and hemodynamic activity underlying the observed signal in fMRI. This model and the methodological advances of the work culminate in application to the deconvolution and effective connectivity problems in this domain.

Acknowledgements

I would firstly like to thank my principal supervisor, Amos Storkey, for making this work possible, and for extensive guidance and advice throughout. My thanks to David McGonigle for his expertise and motivation on all things fMRI, and Chris Williams for his watchful eye at annual reviews.

For technical assistance, additional thanks to David McGonigle for providing the fMRI Session Effects data set used in later chapters. The work has made use of resources provided by the Edinburgh Compute and Data Facility (ECDF)[1], partially supported by the eDIKT initiative[2].

My thanks to Kian Ming Chai for many informal discussions and brainstorms, and Andrew Dai for knowing all too much on the state of the art in computing. To the wider research group who've come and gone – Nicolas Heess, Edwin Bonilla, Stefan Harmeling, John Quinn, Wolfgang Lehrach, Moray Allan, Felix Agakov, Tom Griffiths and others for planting the seeds of whatever ideas I've lost trace of in time.

[1]http://www.ecdf.ed.ac.uk/
[2]http://www.edikt.org

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Lawrence Murray*)

To my dearest Alison. It's all done now honey.

Table of Contents

Chapter 1

Introduction

In fields as diverse as finance, biology and the physical sciences, dynamical systems are naturally modelled using continuous-time stochastic processes. Such equations are a mainstay of stock market prediction, neural modelling, environmental monitoring and other applications. More recently, they have arisen in the hemodynamics underpinning Functional Magnetic Resonance Imaging (fMRI), which particularly motivates this work.

Continuous time modelling can provide an expressiveness that discrete time cannot. The list of notable behaviours more amicable to continuous-time includes jumps, phase transitions and bistabilities. All of these behaviours are actuated only by the continual injection of stochasticity into a system. The introduction of noise at only preset discrete times rarely suffices to capture the onset of these behaviours effectively.

Stochasticity not only drives these intrinsic behaviours of the phenomena under study, but can also be used to account for uncertainty in the model itself. Stochasticity introduced to poorly understood components reflects inconfidence in the model, while limited stochasticity implies firmer knowledge. This is particularly important in the fMRI application of this work, where an understanding of the coupling between neural and hemodynamic activity in the brain is limited [1]. Almost surely the same can be said of other application areas, particularly young fields where computational models are yet to mature.

Given observations and a parameterised model of a continuous-time dynamical system, we are interested in estimating the underlying state and parameters of the model. To do so, some of our most powerful machinery is that of Bayesian filtering, fundamentally the Kalman filter, and in more recent years Monte Carlo techniques such as the particle filter. Largely developed for discrete-time systems, and at most the special case of linear-Gaussian systems in continuous time, the application of such methods to general non-linear, non-Gaussian continuous-time models is far from straightforward. Their applicability is important, however, as such methods can effectively combine a physical model describing system dynamics with actual observations. In the case of fMRI, purely statistical methods such as Structural Equation Modelling (SEM) [2; 3] oppose deterministic methods such as Dynamic Causal Modelling (DCM) [4]. Bayesian filtering represents an intermediate between the extremes of these two paradigms.

This thesis identifies and addresses substantial challenges to state and parameter estimation over the broad class of continuous-time dynamical systems described by Stochastic Differential Equations (SDEs). It attacks these from all sides by drawing together a novel synergy of diverse material from the largely disparate subjects of numerical integration, Bayesian filtering and high performance computing. The methods presented are general, easy to apply and parallelisable. They do not rely on particular functional forms, model dependent integrations or similar. Indeed, the methods are Monte Carlo based, and for the most part compartmentalise the model as a black box.

The work's major contribution is to the *smoothing problem* – estimating the state of a dynamical system across time conditioned on an entire data sequence. Fast, parallelised, approximate methods are presented in the form of the *kernel forward-backward* and *kernel two-filter* smoothers. Both of these are applicable to the continuous time setting without the serious compromises required of conventional methods, and are particularly good candidates for distributed computing architectures. Their use of kernel density estimators may also provide broader support throughout smoothing calculations, potentially alleviating the degeneracy problem apparent in the reweighting technique of conventional particle smoothers.

Smoothing is important in a number of situations:

- to accurately perform state estimation across an entire time series by ex-

ploiting observations as much as possible,
- to assess model fit and perform model comparison, and
- for iterative parameter estimation schemes such as Maximum Likelihood (ML) estimation using Expectation-Maximisation (EM).

The computational efficiency of these kernel smoothers is particularly important for the last of these, and their parallel implementation allows them to be scaled up to large problems. Use of conventional particle smoothers in such an iterated context is untenable owing to their computational expense, all the worse as dimensionality and data size increase.

While the methods are developed and presented in a general light, they have been motivated by problems in fMRI and we acknowledge a bias in pursuing matters most relevant to this case. The peculiarities of this domain include low signal to noise ratio, sparsity of observations in time, fundamental uncertainty in the models used and significant scale in terms of both dimensionality and data size. Despite this, the methods introduced are generally applicable and should find use in many other domains. While fMRI provides the most compelling demonstration of these methods, a number of artificial simulation problems have been employed to demonstrate other points not well illustrated by these.

1.1 Contributions

This thesis tackles a number of outstanding limitations in the applicability of Bayesian filtering to dynamical systems:

- use of continuous time, and
- scalability with dimensionality.

It attacks these from all sides through the novel synergy of a number of substantial but separate bodies of work:

- numerical integration of SDEs,
- Bayesian filtering methods, and
- parallel, distributed and high performance computing.

The major contributions of this work are:

- A substantial review of methodology leading to a clear identification of the challenges inherent in applying Bayesian filtering methods to continuous-time problems.
- Introduction of higher-order Runge-Kutta schemes for numerical integration into a Bayesian filtering and smoothing framework, with demonstrably improved performance.
- Introduction of the kernel forward-backward and kernel two-filter smoothers, facilitating efficient assessment of model fit and iterative parameter estimation schemes. These are applicable to a broader range of dynamical systems than both conventional methods and related work, in particular the general class of models expressed using stochastic differential equations. By their construction they facilitate use of higher-order Runge-Kutta methods, and are computationally more efficient than conventional methods in terms of both space and runtime resources. In addition, through their use of density estimators, they provide a handle to address the degeneracy problem identifiable in all conventional particle smoothing techniques – a means of generating new samples to support the smooth density rather than merely reweighting samples that were drawn with the intent of supporting the filter density only.
- A concrete implementation of these and other approaches suitable for large-scale, high-performance distributed computing environments. This has culminated in the release and continued development and maintenance of the open source `dysii`[1] C++ library for probabilistic inference and learning in continuous-time dynamical systems.
- Extension into the stochastic setting of a biologically motivated deterministic differential model of the neural and hemodynamic activity underlying the observed Blood Oxygen Level Dependent (BOLD) signal in fMRI, as well as novel application of this model and the methods above to fMRI analysis, in particular the deconvolution and effective connectivity problems.

1.2 Impact

This work has contributed to the following publications:

[1]`http://www.indii.org/software/dysii/`

- Murray, L. and Storkey, A. (2008) Continuous time particle filtering for fMRI. *Advances in Neural Information Processing Systems*, **20**.
- Storkey, A. J., Simonotto, E., Whalley, H., Lawrie, S., Murray, L. and McGonigle, D. (2007) Learning Structural Equation Models for fMRI. *Advances in Neural Information Processing Systems*, **19**.

The implementation of the ideas presented in this work is available as:

- The `dysii` Dynamic Systems Library, online at `http://www.indii.org/software/dysii/` and on SourceForge at `http://www.sourceforge.net/projects/dysii/`. This is a substantial body of code, consisting of approximately 20 000 lines of C++ code.

1.3 Related Work

Very recent work has identified similar issues to those discussed throughout this work. In this section we review these and contrast our own methods to them.

1.3.1 Continuous-time modelling

In the context of financial modelling, where SDEs are prevalent, [5] presents the idea of introducing $m - 1$ latent points between each pair of observations of a partially observed stochastic process. This corresponds precisely to an Euler-Maruyama discretisation, with m tuned to control error appropriately. Batch Markov Chain Monte Carlo (MCMC) may then be performed over the system.

We find such fixed time-step Euler-Maruyama discretisation computationally expensive, even untenable in the worst cases, and discuss this at length in §4-5. Our own methods facilitate an adaptive time-step in the first instance, and higher order discretisations than Euler-Maruyama in the second. This potentially delivers substantially leaner runtimes, which we demonstrate through empirical results. The batch MCMC approach may also suffer from poor mixing, particularly in high dimensional systems where we might expect many modes in the posterior. While the importance sampling techniques of this work also suffer under dimensionality for other reasons, their sequential approach to inference, involving simulation of the dynamical system, allows them to sample from all modes.

Rather than attempting to maintain tractability of the transition density, [6] instead constructs an estimator of the density evaluations. The methodology begins with the *exact algorithm* [7] for simulating *Brownian bridges* – trajectories given fixed start and end points. The exact algorithm is a rejection sampling scheme built around the Poisson estimator, the acceptance rate of which can be related to the transition density. This allows arbitrary proposals to be made at time t_{n+1} and matched to samples at time t_n. The result is an unbiased estimator of the transition density with various nice theoretical properties. This estimator can then replace the transition density in the particle filter weight calculation to obtain the *random weight particle filter* [6]. One could potentially apply the same ideas to the smoothing problem.

The main limitation of this methodology, as acknowledged by the authors, is that the SDE must be transformed into one with gradient drift and additive diffusion in order to use the exact algorithm. That is, of the form:

$$d\mathbf{x} = \mathbf{a}(\mathbf{x}, t)\, dt + B(t)\, d\mathbf{W}\,, \tag{1.1}$$

where there exists an $A : \mathbb{R}^N \to \mathbb{R}$ such that $\mathbf{a}(\mathbf{x}, t) = \nabla A(\mathbf{x}, t)$. Note in particular that the B term cannot depend on the state $\mathbf{x}$, thus prohibiting correlated noise structures.

The authors discuss the conversion of any given equations into equivalent equations of this form, which is always possible in the one-dimensional case. It is not always possible in higher-dimensional cases, however. This may be stifling, although it is difficult to judge how significant a limitation it poses in practice. Even if theoretically tenable, conversion of models into such a form may prove especially difficult pragmatically.

Furthermore, while numerical integration of the SDE is not required *per se*, the authors advocate the use of local linearization [8] or Euler-Maruyama in establishing an appropriate proposal distribution for the filter. Discretisation error may therefore be avoided in relegating these schemes to the proposal only, but the computational expense of these low-order integrators is not.

The methods of this work apply to a more general class of systems, as given by (3.1). In particular they permit multiplicative or other correlated noise, as in the fMRI models that we use experimentally in §7.1, and do not impose constraints on the drift term. We would highlight that the single kernel treatment of the kernel forward-backward smoother (§5.4.1) alludes to the use of a single kernel as an estimator of the transition density also, albeit a much simpler one to that proposed in [6].

Fundamentally different approaches to Monte Carlo may be considered, in particular variational methods [9; 10]. These are promising, particularly as parameters may be estimated at the same time as the state. As with all variational approaches, however, there is the practical problem of choosing an appropriate family of approximate closed-form densities to fit to the true distribution, a decision exponentially more difficult in high dimensional spaces.

1.3.2 Other material

A number of other works are worth noting, although by and large they bear only superficial resemblance to the methods described here. Nevertheless, they have levered similar techniques to address some of the issues fringing this work.

In discussing the regularised particle filter, [11] derives an analagous regularised smoother based on diversifying particle stock using kernel densities. Its derivation is equivalent to that of conventional smoothers, with the introduction of kernels over the filter density only providing a regularisation equivalent to that employed for the forward pass. It is not motivated by the continuous time setting of this work, and in particular makes no attempt to eliminate the transition density or establish a scheme for drawing new samples to support the smooth density.

In [12], *kd* trees are used to improve the performance of a conventional smoother, as we use them in §6 to speed up kernel density evaluations. This is again not motivated by continuous-time models and does not attempt to address issues surrounding the potential intractability of the transition density, or degeneracy.

It is also worth noting a number of other techniques to perform some sort of interpolation over sample points in a particle filtering context. Similar to *kd* trees, [13] makes use of *density trees*, loosely speaking histograms of heterogenous bin size, albeit in a discrete state space. Parametric methods such as Gaussian mixtures are an obvious alternative to these nonparametric methods also, and numerous works on Gaussian mixture based filters exist for the discrete-time case [14; 15; 16]. Variational distribution fits could potentially be coupled in also.

1.3.3 Summary

This section has reviewed very recent developments in the area and contrasted the development of our own approach with them ahead of its detailed exposition. The main advantage of our own methods is their generality, with other approaches reliant on particular assumptions imposed on the underlying form of the dynamical system.

1.4 Structure

The work is separated into two parts. Part I develops the Bayesian filtering aspect of this work in a generalised manner, while Part II specialises this to the particular case of fMRI.

Part I opens with a technical review of essential material – §2 introducing Bayesian filtering, smoothing and parameter estimation, and §3 continuous time stochastic processes. The two are fused with emergent challenges in §4, with a number of working problems introduced for experimental elucidation. §5 represents the major theoretical contribution of the work, deriving the kernel forward-backward and kernel two-filter smoothers and demonstrating their application to the models introduced in the preceding chapter. §6 is the major computational contribution of the work, discussing implementation for distributed computing.

The developed methods are applied to fMRI analysis in Part II. §7 provides a brief introduction to essential material in fMRI, before §8 applies the theory developed in Part I to salient problems in this domain.

§9 provides a final assessment of the work, fruitful avenues for future work and concluding remarks.

Part I

Bayesian Learning of Continuous Time Dynamical Systems

Chapter 2

Bayesian Learning of Dynamical Systems

This chapter provides a general mathematical formulation of dynamical systems (§2.1) and an overview of methods for inference and learning within such systems. It particularly focuses on the class of sequential Monte Carlo methods commonly called *particle filters*, and their application in solving the filtering (§2.2), smoothing (§2.3) and parameter estimation (§2.4) problems.

2.1 Dynamical systems

For an ascending sequence of T time points $t_1, \ldots, t_T$, we are provided with measurements $\mathbf{y}(t_1), \ldots, \mathbf{y}(t_T) \in \mathbb{R}^M$ indicative of the latent state $\mathbf{x}(t) \in \mathbb{R}^N$ of a dynamical system across time t. The state of the system transitions according to a stochastic Markov function $\mathbf{f}(\mathbf{x}, \mathbf{v}, \boldsymbol{\theta}, t)$, where $\mathbf{v}$ is system noise and $\boldsymbol{\theta}$ static parameters across time. The measurement acquired from the system in any state may be predicted by a known function $\mathbf{g}(\mathbf{x}, \mathbf{w}, \boldsymbol{\theta})$, where $\mathbf{w}$ is measurement noise. Figure 2.1 provides the graphical model.

Of interest is the learning of the unknown parameters $\boldsymbol{\theta}$, as well as estimating the state of the latent process $\mathbf{x}(t)$ given the observations of the process $\mathbf{y}(t)$.

For notational convenience and without loss of generality, we let $\mathbf{x}_0$ denote the

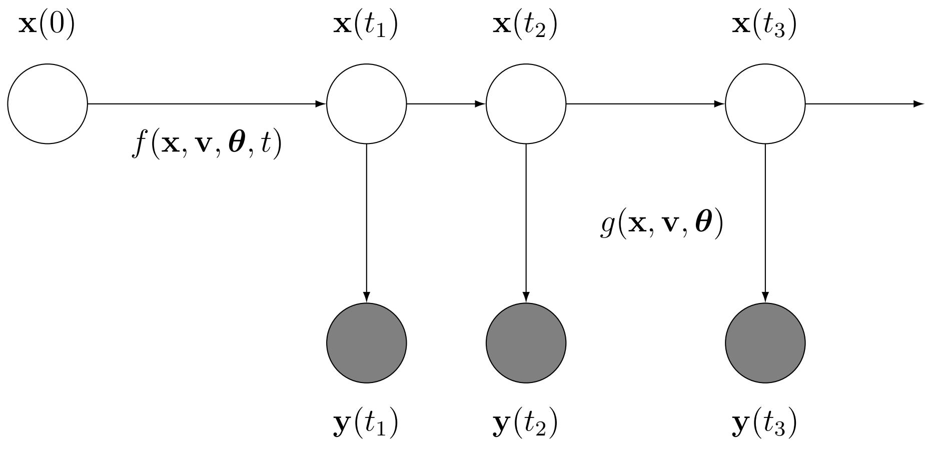


Figure 2.1: Directed graphical model of the dynamical system formulation.

initial state of the system at time $t = 0$, and denote $\mathbf{x}_n = \mathbf{x}(t_n)$ and $\mathbf{y}_n = \mathbf{y}(t_n)$. In referring to sequences, $\mathbf{x}_{i:j}$ with $i < j$ will refer to the set $\{\mathbf{x}_i, \ldots, \mathbf{x}_j\}$ and likewise $\mathbf{y}_{i:j}$ to $\{\mathbf{y}_i, \ldots, \mathbf{y}_j\}$. While this compact notation does suggest discrete time sequences, we stress that it is simply a shorthand for referring to the original processes $\mathbf{x}(t)$ and $\mathbf{y}(t)$ at particular time points. We also particularly note that $t_1, \ldots, t_n$ need not be equispaced.

The conditional independencies implicit in the structure of Figure 2.1 highlight the assumed *Markovian* nature of $\mathbf{f}(\cdot)$. A Markov process is one in which the following property is asserted[1]:

Property 2.1 (Markov conditional property) *For an ascending sequence of times $t_1, \ldots, t_T$, a Markov process exhibits the property:*

$$p(\mathbf{x}_{j:T} \,|\, \mathbf{x}_{1:i}) = p(\mathbf{x}_{j:T} \,|\, \mathbf{x}_i)\,,$$

for $i < j$. That is, conditionals on the process depend only on the most recent state of the process.

By applying the product rule:

$$\begin{aligned} p(\mathbf{x}_{1:T}) &= p(\mathbf{x}_T \,|\, \mathbf{x}_{1:T-1})p(\mathbf{x}_{1:T-1}) && (2.1) \\ &= p(\mathbf{x}_T \,|\, \mathbf{x}_{T-1})p(\mathbf{x}_{1:T-1})\,, && (2.2) \end{aligned}$$

[1]Merely asserted and not derived because it is arguable whether any such process can really exist outside of mathematical abstraction. See [17] for discussion.

and expanding recursively, the following corollary holds:

Property 2.2 (Markov joint property) *For an ascending sequence of times $t_1, \ldots, t_T$, a Markov process exhibits the property:*

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_1) \prod_{i=2}^{T} p(\mathbf{x}_i \,|\, \mathbf{x}_{i-1}) \,.$$

That is, joint densities over states along a time sequence may be factorised as the product of conditional densities between states at neighbouring times in the sequence.

These two properties will be critical in the derivation of the methods described in this chapter.

2.2 Filtering

To solve the *filtering problem*, we wish to estimate $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$ for all $n = 1, \ldots, T$, that is, the distribution over the state at each time point given the measurements acquired up to that time. We will refer to this as the *filter density*.

In the simplest case where the filter densities are Gaussian and the transition and measurement functions linear, the seminal Kalman filter [18] may be used to obtain the optimal solution to the filtering problem. Extensions of this for the nonlinear case including the extended, and more recently unscented, Kalman filters [19; 20]. Extensions of these to the non-Gaussian case include the Gaussian sum Kalman filter [14] and Gaussian mixture sigma-point particle filter [16], respectively. We particularly discriminate such methods from those of discrete-state methods, such as the Hidden Markov Model, which are not applicable in continuous state settings.

The applicability of these methods requires that simplifying assumptions be imposed on the filter densities, and transition and measurement functions. In the general case of nonlinear functions and analytically intractable densities, sample-based Monte Carlo methods trump these analytical tools. In the context of dynamical systems, such methods are commonly grouped under the heading of *particle filters*, in bulk representing the sequential extension of importance sampling. We concentrate our review on this family of techniques.

2.2.1 Particle filter

Rather than assume that the state at time t_n is distributed in some closed form manner, arbitrary filter densities may be admitted by approximating the state with a weighted set of P samples $\{(\mathbf{s}_n^{(i)}, \pi_n^{(i)})\}$, where $i = 1, \ldots, P$ and $\sum_{i=1}^{P} \pi_n^{(i)} = 1$, so that $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) \approx \sum_{i=1}^{P} \pi_n^{(i)} \delta(\mathbf{x}_n - \mathbf{s}_n^{(i)})$. Given the filter density at one time, we recursively approximate that at the next by propagating these samples ("particles") through time using the transition function $\mathbf{f}(\cdot)$ and adjusting their weights using the likelihood of the observation at the next time under the measurement function $\mathbf{g}(\cdot)$.

To motivate the basic idea, we begin with the simplest particle filter, the *bootstrap filter* [21][2], before introducing a more general formulation.

Algorithm 2.1 (Bootstrap particle filter) *For time t_n, and given the weighted sample set $\{(\mathbf{s}_{n-1}^{(i)}, \pi_{n-1}^{(i)})\}$ representing the filter density $p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1})$, propagate each sample $\mathbf{s}_{n-1}^{(i)}$ through the transition function $\mathbf{f}(\cdot)$ to obtain $\mathbf{s}_n^{(i)}$ and weight with:*

$$\pi_n^{(i)} = p(\mathbf{y}_n \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)}) \pi_{n-1}^{(i)} \tag{2.3}$$

Normalise all weights to sum to 1. The weighted sample set $\{(\mathbf{s}_n^{(i)}, \pi_n^{(i)})\}$ then approximates the filter density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$.

This basic algorithm suffers from the problem of *degeneracy*, tending to heap weight on only a single particle after several iterations and normalisation. To mitigate this, an additional *resampling* step is added to eliminate lowly weighted particles and multiply highly weighted particles. This may simply be a multinomial draw from the sample set, with replacement and probabilities commensurate with weights, producing a new set of equally weighted particles. A stratified approach [23], motivated by minimising resampling bias, is most common.

[2] Or *condensation algorithm* [22].

We now introduce a more general importance sampling formulation into which common variants of the algorithm will be fit. This novel formulation is different to that of other work in the way it considers importance sampling of pairs $\{\mathbf{s}_{n-1:n}^{(i)}\}$ rather than single points $\{\mathbf{s}_n^{(i)}\}$. We have done this to integrate resampling into the proposal, which will emphasise emergent issues in the continuous-time setting later in this work.

A recursive factorisation of the filter density proceeds as follows:

$$\begin{aligned} p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) &= \frac{p(\mathbf{y}_n \,|\, \mathbf{x}_n, \mathbf{y}_{1:n-1}) p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n-1})}{p(\mathbf{y}_n \,|\, \mathbf{y}_{1:n-1})} && (2.4) \\ &\propto p(\mathbf{y}_n \,|\, \mathbf{x}_n) p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n-1}) && (2.5) \\ &\propto p(\mathbf{y}_n \,|\, \mathbf{x}_n) \int p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1}) p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1})\, d\mathbf{x}_{n-1}\,. && (2.6) \end{aligned}$$

In the base case:

$$p(\mathbf{x}_1 \,|\, \mathbf{y}_1) \propto p(\mathbf{y}_1 \,|\, \mathbf{x}_1) \int p(\mathbf{x}_1 \,|\, \mathbf{x}_0) p(\mathbf{x}_0)\, d\mathbf{x}_0\,, \tag{2.7}$$

where $p(\mathbf{x}_0)$ is the prior over the initial state of the system. To reduce clutter, we use proportions ($\propto$) to give densities up to a constant factor. This is a common feature of the derivations given here and elsewhere, and is justified shortly. We also particularly highlight the importance of the $p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})$ term for later in this work, and will refer to it as the *transition density*.

Proceeding from (2.6), note the joint density:

$$p(\mathbf{x}_{n-1:n} \,|\, \mathbf{y}_{1:n}) \propto p(\mathbf{y}_n \,|\, \mathbf{x}_n) p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1}) p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1})\,, \tag{2.8}$$

and consider importance sampling from this using a proposal density $q'(\mathbf{x}_{n-1:n})$. For $i = 1, \ldots, P$, draw $\mathbf{s}_{n-1:n}^{(i)} \sim q'(\mathbf{x}_{n-1:n})$ and weight with:

$$\pi_n^{(i)} = \frac{p(\mathbf{y}_n \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)}) p(\mathbf{x}_n = \mathbf{s}_n^{(i)} \,|\, \mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)}) p(\mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)} \,|\, \mathbf{y}_{1:n-1})}{q'(\mathbf{x}_{n-1:n} = \mathbf{s}_{n-1:n}^{(i)})}\,. \tag{2.9}$$

Then normalise all weights to sum to 1 by reassigning:

$$\pi_n^{(i)} \leftarrow \frac{\pi_n^{(i)}}{\sum_{j=1}^{P} \pi_n^{(j)}}\,. \tag{2.10}$$

Such normalisation is hereafter considered implicit after any assignment of weights. Observe that as the $p(\mathbf{y}_n \,|\, \mathbf{y}_{1:n-1})$ term is dependent only on observations, it is the same for all particles, and so eliminated during this normalisation. This justifies the use of proportions introduced above.

In summary, the general algorithm works as follows:

Algorithm 2.2 (Abstract particle filter) *Given the weighted sample set* $\{(\mathbf{s}_{n-1}^{(i)}, \pi_{n-1}^{(i)})\}$*, representing the density* $p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1})$*, and a proposal density* $q'(\mathbf{x}_{n-1:n})$*, sample* $\mathbf{s}_{n-1:n}^{(i)} \sim q'(\mathbf{x}_{n-1:n})$ *and weight with:*

$$\pi_n^{(i)} = \frac{p(\mathbf{y}_n \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)})p(\mathbf{x}_n = \mathbf{s}_n^{(i)} \,|\, \mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)})p(\mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)} \,|\, \mathbf{y}_{1:n-1})}{q'(\mathbf{x}_{n-1:n} = \mathbf{s}_{n-1:n}^{(i)})}. \tag{2.11}$$

The weighted sample set $\{(\mathbf{s}_n^{(i)}, \pi_n^{(i)})\}$ *then approximates the filter density* $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$.

Note that resampling, in this context, is absorbed into the proposal density.

In most cases, the runtime complexity of a particle filter is $\mathcal{O}(TP)$, and memory complexity $\mathcal{O}(NP)$. This complexity could conceivably increase for particularly elaborate proposal distributions.

In the following sections we consider alternatives for the proposal $q'(\mathbf{x}_{n-1:n})$, deriving some common variants of the basic particle filtering algorithm. In addition to a derivation of the bootstrap filter in this framework, we focus on the two which will prove most relevant later in this work, the *auxiliary* [24] and *regularised* [25] particle filters.

2.2.2 Bootstrap particle filter

As $p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1})$ is not available in closed form, consider a factorised importance density of the form $q'(\mathbf{x}_{n-1:n}) = q(\mathbf{x}_n)p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1})$, such that:

$$\pi_n^{(i)} = \frac{p(\mathbf{y}_n \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)})p(\mathbf{x}_n = \mathbf{s}_n^{(i)} \,|\, \mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)})}{q(\mathbf{x}_n = \mathbf{s}_n^{(i)})}, \tag{2.12}$$

$p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1})$ having been cancelled so as to avoid calculating it explicitly. This is sufficient if independently drawing each $\mathbf{s}_{n-1}^{(i)} \sim p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1})$. This is akin to a simple resampling scheme, which should preferably remain decoupled from the main algorithm. In practice the particles $\{\mathbf{s}_{n-1}^{(i)}\}$ from the previous time point are reused deterministically to reduce sampling error. In this case their importance weights $\{\pi_{n-1}^{(i)}\}$ must also be multiplied in:

$$\pi_n^{(i)} = \frac{p(\mathbf{y}_n \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)})p(\mathbf{x}_n = \mathbf{s}_n^{(i)} \,|\, \mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)})}{q(\mathbf{x}_n = \mathbf{s}_n^{(i)})}\pi_{n-1}^{(i)} \,. \tag{2.13}$$

Any resampling strategy can be employed as necessary to combat degeneracy.

The bootstrap filter of Algorithm 2.1 corresponds precisely to the case where $q(\mathbf{x}_n) = p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)})$, cancelling the transition density in the numerator of (2.11). While simple and computationally efficient, this proposal critically neglects consideration of the upcoming observation $\mathbf{y}_n$, and so is suboptimal in cases where observations are known ahead of time[3]. Other proposals attempt to better fit the target density so as to provide a more lucid representation for any fixed number of samples. Combined with appropriate resampling strategies, they may also diversify the particle stock, curtailing myopic loitering in neighbourhoods of high likelihood.

2.2.3 Auxiliary particle filter

The auxiliary particle filter [24] attempts to incorporate knowledge of the next observation into the proposal by setting:

$$\begin{aligned}
q'(\mathbf{x}_{n-1:n}) &= p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n}) && (2.14)\\
&= \frac{p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})p(\mathbf{y}_n \,|\, \mathbf{x}_{n-1}, \mathbf{y}_{1:n-1})p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1})}{p(\mathbf{y}_n \,|\, \mathbf{y}_{1:n-1})} && (2.15)\\
&= \frac{p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})p(\mathbf{y}_n \,|\, \mathbf{x}_{n-1})p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1})}{p(\mathbf{y}_n \,|\, \mathbf{y}_{1:n-1})} && (2.16)\\
&\propto p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})p(\mathbf{y}_n \,|\, \mathbf{x}_{n-1})p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1}) && (2.17)
\end{aligned}$$

Now consider importance sampling from (2.8) using this as proposal. The weight calculation reduces to:

$$\pi_n^{(i)} = \frac{p(\mathbf{y}_n \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)})}{p(\mathbf{y}_n \,|\, \mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)})} \,. \tag{2.18}$$

The denominator may be expanded to:

$$p(\mathbf{y}_n \,|\, \mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)}) = \int p(\mathbf{y}_n \,|\, \mathbf{x}_n)p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)}) \, d\mathbf{x}_n \,, \tag{2.19}$$

[3] Early work on particle filters, including the bootstrap filter, was often motivated by real-time tracking applications where the next measurement is not available ahead of time (see e.g. [22]).

although in practice a single point estimate is usually made, avoiding expensive calculation of this integral. Combining all this results in the following algorithm:

Algorithm 2.3 (Auxiliary particle filter) *For $i = 1, \ldots, P$, draw $\mathbf{s}_*^{(i)} \sim p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)})$ and let:*

$$\beta_*^{(i)} = p(\mathbf{y}_n \,|\, \mathbf{x}_n = \mathbf{s}_*^{(i)}) \pi_{n-1}^{(i)} \tag{2.20}$$

The weighted sample set $\{(\mathbf{s}_{n-1}^{(i)}, \beta_*^{(i)})\}$ now approximates the density $p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n})$. Resample the $\{\mathbf{s}_{n-1}^{(i)}\}$ using the weights $\{\beta_*^{(i)}\}$ in place of the original weights $\{\pi_{n-1}^{(i)}\}$. For each resampled point $\mathbf{s}_{n-1}^{(i)}$, of original index j (repetitions likely), redraw $\mathbf{s}_n^{(i)} \sim p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)})$ and weight with:

$$\pi_n^{(i)} = p(\mathbf{y}_n \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)}) \frac{\pi_{n-1}^{(j)}}{\beta_*^{(j)}} \,. \tag{2.21}$$

The weighted sample set $\{(\mathbf{s}_n^{(i)}, \pi_n^{(i)})\}$ then approximates the filter density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$.

Intuitively, this augments the myopic bootstrap filter with an auxiliary process that provides a one step lookahead, on the assertion that the propagation of particles important at the current time is not as critical as the propagation of particles likely to be important in future. Extension to multiple steps is straightforward, at the risk of degeneration in the auxiliary process as the lookahead increases.

Equivalently, the auxiliary particle filter may be considered a resampling strategy that favours particles likely to be important at the next time point, with an importance reweighting to compensate for this bias.

2.2.4 Regularised particle filter

The regularised particle filter [25] attempts to diversity particle stock by introducing kernel densities. Recall that in the standard particle filter, the filter density is represented as a weighted mixture of point masses:

$$p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) \approx \sum_{i=1}^{P} \pi_n^{(i)} \delta(\mathbf{x}_n - \mathbf{s}_n^{(i)}) \,. \tag{2.22}$$

In order to interpolate this density between sample points, consider replacing the Dirac functions with a broader kernel $\mathcal{K}$ of bandwidth h, obtaining:

$$p_{\mathcal{K}}(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) \approx \frac{1}{h^N} \sum_{i=1}^{P} \pi_n^{(i)} \mathcal{K}(\frac{1}{h} \|\mathbf{x}_n - \mathbf{s}_n^{(i)}\|) \,. \tag{2.23}$$

$\mathcal{K}$ may be any kernel satisfying a set of weak conditions, of which the univariate Gaussian is one such case, and used throughout this work. Other possible selections include uniform, cosine and Epanechnikov kernels, along with a range of other classes of both finite and infinite extent. Whatever the selection of kernel, the result is a *kernel density estimate* [26] of the intractable filter density, from which it is possible to make approximate density calculations at arbitrary points in the space.

Selection of an appropriate bandwidth h can be tricky. For this purpose we may wish to incorporate a standardisation of the sample points:

$$p_{\mathcal{K}}(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) \approx \frac{1}{h^N |L_n|^{-1}} \sum_{i=1}^{P} \pi_n^{(i)} \mathcal{K}(\frac{1}{h} \|L_n^{-1}(\mathbf{x}_n - \mathbf{s}_n^{(i)})\|) \,, \tag{2.24}$$

where L_n is the Cholesky decomposition of the sample covariance matrix at time t_n. This is useful as it brings to bear a wealth of guidance as to the setting of h. In particular, for P samples from a standard N-dimensional Gaussian distribution, and Gaussian kernel $\mathcal{K}$, the optimal bandwidth is given by [26]:

$$h_{\text{opt}}(N, P) = \left[\frac{4}{(N+2)P} \right]^{\frac{1}{N+4}} \,. \tag{2.25}$$

As a rule of thumb, multiples of this are generally an appropriate selection. To firm intuition of such kernel densities, Figure 2.2 depicts kernel density estimates of a simple Gaussian mixture density for various bandwidth settings.

To implement the regularised particle filter, the intractable filter density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$ is replaced by its kernel density estimate $p_{\mathcal{K}}(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$. Particles are resampled from the kernel density before proceeding according to the standard bootstrap filter, so that $q(\mathbf{x}_n) = p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1} = \mathbf{s}_{n-1}^{(i)})$:

Algorithm 2.4 (Regularised particle filter) *For* $i = 1, \ldots, P$, *draw* $\mathbf{s}_{\mathcal{K}}^{(i)} \sim p_{\mathcal{K}}(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1})$ *and* $\mathbf{s}_n^{(i)} \sim p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1} = \mathbf{s}_{\mathcal{K}}^{(i)})$, *and let:*

$$\pi_n^{(i)} = p(\mathbf{y}_n \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)}) \tag{2.26}$$

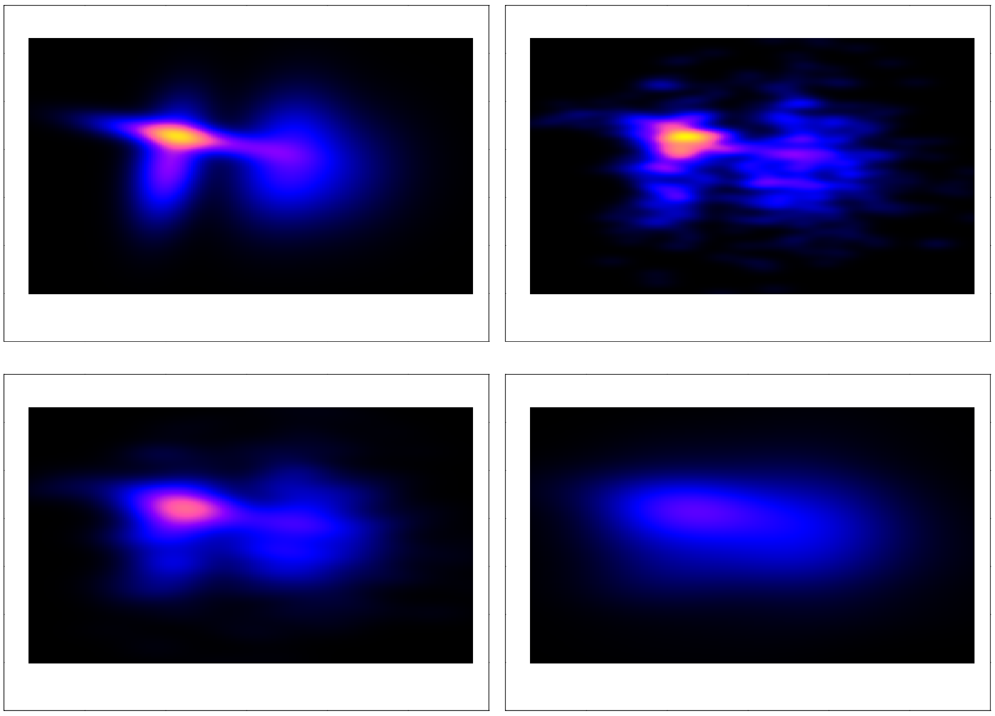


Figure 2.2: Example kernel density estimation; **(top left)** original four component Gaussian mixture in $N = 2$ dimensions; $P = 1000$ Gaussian kernel density estimates of bandwidth **(top right)** $h_{\text{opt}}/2$, **(bottom left)** h_{opt} and **(bottom right)** $2h_{\text{opt}}$. Colour scale is equivalent in all cases.

The weighted sample set $\{(\mathbf{s}_n^{(i)}, \pi_n^{(i)})\}$ *or kernel density* $p_{\mathcal{K}}(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) = \frac{1}{h^N} \sum_{i=1}^{P} \pi_n^{(i)} \mathcal{K}(\frac{1}{h} \|\mathbf{x}_n - \mathbf{s}_n^{(i)}\|)$ *then approximates the filter density* $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$.

To efficiently sample from the kernel density, first draw $j \sim \{i = 1, \ldots, P\}$ with probabilities given by the weights $\{\pi_n^{(i)}\}$, then take the sample $\mathbf{s}_{\mathcal{K}}^{(i)} = \mathbf{s}_n^{(j)} + k^{(i)}\mathbf{e}^{(i)}$, where $k^{(i)} \sim \mathcal{K}(\cdot)$ and $\mathbf{e}^{(i)}$ is a vector drawn uniformly from the sphere of unit vectors in $\mathbb{R}^N$. The latter may be obtained by exploiting the spherical symmetry of a Gaussian distribution, drawing an N-dimensional vector and normalising to unit length.

The main advantage of the regularised particle filter is that its kernels facilitate greater diversity in particle stock. This becomes particularly useful for parameter estimation, discussed in §2.4.

2.2.5 Other considerations

There are several other potentially useful proposals and one can list numerous possibilities. By way of example, these include the output of a concurrently running unscented Kalman filter [27], and use of density trees [13] on a similar vein to kernel densities. While these broad proposals are useful in maintaining particle diversity, they may require an increase in the number of particles to provide adequate support for the true density. Finer control can be obtained via precise application of domain specific transition kernels [28][ch.6], known as the *resample-move* strategy. An exhaustive review of published proposals is beyond the scope of this work. We point the reader to [28] for a handle into the literature.

One additional heuristic is worth mentioning. Because resampling constitutes an additional source of sampling error, it can be worth regulating so as to perform only when necessary. A useful measure is that of *effective sample size* [29] (ESS), defined as:

$$P_{\text{ess}} = \frac{\left(\sum_{i=0}^{P} \pi^{(i)}\right)^2}{\sum_{i=0}^{P} (\pi^{(i)})^2}\,, \tag{2.27}$$

that is, the inverse of the sum of the normalised weights. Intuitively, this provides a measure, not greater than P, approximating the equivalent number of independent samples corresponding to the weighted particle stock. Observe that if all weights are equal, as after resampling, $P_{\text{ess}} = P$. The greater the variance of the weights, the smaller P_{ess}, until one weight dominates and the measure approaches 1.

Effective sample size may be used to trigger resampling. A simple strategy, for example, is to resample only when $P_{\text{ess}} < P/2$.

2.3 Smoothing

To solve the *smoothing problem*, we wish to calculate $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T})$ for all $n = 1, \dots, T$, that is, the distribution over the state at each time point given all the available measurements. We will refer to this as the *smooth density*.

Unlike filtering, which is often motivated by real-time applications, or at least the constraint of observations arriving online, smoothing is usually an offline operation used to improve the estimates of a filter *post hoc*. In most cases a smoother accompanies a compatible filter, and so our focus here will be on techniques complementary to the methods presented in §2.2.

2.3.1 Filter-smoother

Our first consideration is to apply a filter while preserving whole particle trajectories [23]. Rather than marginalising out the particle history to obtain the filter density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$, trajectories are preserved to instead approximate $p(\mathbf{x}_{1:n} \,|\, \mathbf{y}_{1:n})$ by $\{(\mathbf{s}_{1:n}^{(i)}, \pi_n^{(i)})\}$, where $\mathbf{s}_{1:n}^{(i)}$ stores the entire trajectory of the ith particle since the start of the filter. After obtaining $p(\mathbf{x}_{1:T} \,|\, \mathbf{y}_{1:T})$ at conclusion of the filter, the smooth density at any time may be obtained by marginalising out samples from all other times. We refer to this approach as the *filter-smoother*.

The main attraction of this method is negligible additional computational cost over that of a standard filter. Indeed, like a filter, the runtime complexity of the method is $\mathcal{O}(TP)$. Memory requirements increase to $\mathcal{O}(TNP)$ given the storage of entire trajectories.

Clearly, if the filter is to avoid degeneracy, a resampling scheme will be employed. By purpose, resampling deliberately terminates some trajectories while branching others, and unfortunately this stymies the filter-smoother. After some time, it is likely that the entire particle stock will descend from a single common ancestor at some earlier time [30]. Marginalisation to obtain the smooth density at that earlier time will produce a degenerate estimate heaped on a single particle.

For this reason, despite its computational attractiveness, the filter-smoother is rarely of practical use.

2.3.2 Two-filter smoother

Consider the following factorisation of the smooth density:

$$\begin{aligned} p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T}) &= \frac{p(\mathbf{y}_{n:T} \,|\, \mathbf{x}_n, \mathbf{y}_{1:n-1}) p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n-1})}{p(\mathbf{y}_{n:T} \,|\, \mathbf{y}_{1:n-1})} && (2.28) \\ &\propto p(\mathbf{y}_{n:T} \,|\, \mathbf{x}_n) p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n-1}) \,. && (2.29) \end{aligned}$$

In this way the smooth density is proportional to the product of a forward looking likelihood and filter density. Focusing on the likelihood:

$$\begin{aligned} p(\mathbf{y}_{n:T}\,|\,\mathbf{x}_n) &= \frac{p(\mathbf{x}_n\,|\,\mathbf{y}_{n:T})p(\mathbf{y}_{n:T})}{p(\mathbf{x}_n)} && (2.30)\\ &\propto \frac{p(\mathbf{x}_n\,|\,\mathbf{y}_{n:T})}{p(\mathbf{x}_n)}\,, && (2.31)\end{aligned}$$

so that the likelihood may be obtained through the combination of a prior and a filter starting at time t_T and progressing *backward* in time. Care should be taken in the construction of such a filter, in particular noting that the inverse of the transition function $\mathbf{f}(\cdot)$ does not necessarily provide the correct reverse dynamics [12][§3.2.1], as it may not propagate noise as intended. Nevertheless, in many cases such a backward filter can be constructed directly. In its absence:

$$\begin{aligned} p(\mathbf{x}_n\,|\,\mathbf{y}_{n:T}) &= \frac{p(\mathbf{y}_n\,|\,\mathbf{x}_n,\mathbf{y}_{n+1:T})p(\mathbf{x}_n\,|\,\mathbf{y}_{n+1:T})}{p(\mathbf{y}_n\,|\,\mathbf{y}_{n+1:T})} && (2.32)\\ &\propto p(\mathbf{y}_n\,|\,\mathbf{x}_n)p(\mathbf{x}_n\,|\,\mathbf{y}_{n+1:T}) && (2.33)\\ &\propto p(\mathbf{y}_n\,|\,\mathbf{x}_n)\int p(\mathbf{x}_n\,|\,\mathbf{x}_{n+1})p(\mathbf{x}_{n+1}\,|\,\mathbf{y}_{n+1:T})\,d\mathbf{x}_{n+1}\,, && (2.34)\end{aligned}$$

which has so far proceeded as per the forward filter (2.4-2.6). The problem here is still the presence of the reverse dynamics, assumed unknown, which may be removed with a simple application of Bayes' rule:

$$p(\mathbf{x}_n\,|\,\mathbf{y}_{n:T}) \propto p(\mathbf{y}_n\,|\,\mathbf{x}_n)\int \frac{p(\mathbf{x}_{n+1}\,|\,\mathbf{x}_n)p(\mathbf{x}_n)}{p(\mathbf{x}_{n+1})}p(\mathbf{x}_{n+1}\,|\,\mathbf{y}_{n+1:T})\,d\mathbf{x}_{n+1}\,, \qquad (2.35)$$

Clearly the tradeoff here is the requirement for priors $p(\mathbf{x}_n)$ and $p(\mathbf{x}_{n+1})$ in the latter case. Observe:

$$p(\mathbf{x}_n) = \int\cdots\int p(\mathbf{x}_0)\prod_{i=1}^{n} p(\mathbf{x}_i\,|\,\mathbf{x}_{i-1})\,d\mathbf{x}_{1:n-1}\,, \qquad (2.36)$$

so that for a closed form $p(\mathbf{x}_0)$ and linear transition function these priors will be tractable. In this case the problem is solved [23], although in general this will not be the case[4].

To evade this problem we introduce the approximations $\gamma_n(\mathbf{x}_n) \approx p(\mathbf{x}_n)$ [12]:

$$p(\mathbf{x}_n\,|\,\mathbf{y}_{n:T}) \propto p(\mathbf{y}_n\,|\,\mathbf{x}_n)\int \frac{p(\mathbf{x}_{n+1}\,|\,\mathbf{x}_n)\gamma_n(\mathbf{x}_n)}{\gamma_{n+1}(\mathbf{x}_{n+1})}p(\mathbf{x}_{n+1}\,|\,\mathbf{y}_{n+1:T})\,d\mathbf{x}_{n+1}\,. \qquad (2.37)$$

[4]And, indeed, if they were, something like a Kalman filter may be more appropriate than a particle filter.

Each $\gamma_n(\mathbf{x}_n)$ may, for example, be an approximate analytical distribution fit to a sample-based representation of $p(\mathbf{x}_n)$ (see [12] for discussion). By incorporating a proposal distribution $q(\mathbf{x}_n)$ to generate samples, we arrive at the following recursive algorithm for the backward filter:

Algorithm 2.5 (Backward particle filter) *Given the weighted sample set $\{(\tilde{\mathbf{s}}^{(i)}_{n+1}, \tilde{\pi}^{(i)}_{n+1})\}$, representing the backward filter density $p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{n+1:T})$, approximate priors $\gamma_n(\mathbf{x}_n)$ and $\gamma_{n+1}(\mathbf{x}_n)$, and a proposal distribution $q(\mathbf{x}_n)$, sample $\tilde{\mathbf{s}}^{(i)}_n \sim q(\mathbf{x}_n)$ and weight with:*

$$\tilde{\pi}^{(i)}_n = \frac{p(\mathbf{y}_n \,|\, \mathbf{x}_n = \tilde{\mathbf{s}}^{(i)}_n)\gamma_n(\mathbf{x}_n = \tilde{\mathbf{s}}^{(i)}_n)}{q(\mathbf{x}_n = \tilde{\mathbf{s}}^{(i)}_n)} \sum_{j=1}^{P} \frac{p(\mathbf{x}_{n+1} = \tilde{\mathbf{s}}^{(j)}_{n+1} \,|\, \mathbf{x}_n = \tilde{\mathbf{s}}^{(i)}_n)\tilde{\pi}^{(j)}_{n+1}}{\gamma_{n+1}(\mathbf{x}_{n+1} = \tilde{\mathbf{s}}^{(j)}_{n+1})}\,. \tag{2.38}$$

The weighted sample set $\{(\tilde{\mathbf{s}}^{(i)}_n, \tilde{\pi}^{(i)}_n)\}$ then approximates the backward filter density $p(\mathbf{x}_n \,|\, \mathbf{y}_{n:T})$.

Possible selections for the proposal distribution are akin to those for the forward filter.

Whether by direct filtering using reverse dynamics or use of Algorithm 2.5, we have demonstrated that it is possible to perform a filter forward in time, and a filter backward in time. All that remains is to fuse the results to obtain the smooth density. Substituting (2.31) into (2.29), we obtain:

$$\begin{aligned}
p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T}) &\propto p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n-1})\frac{p(\mathbf{x}_n \,|\, \mathbf{y}_{n:T})}{p(\mathbf{x}_n)} && (2.39)\\
&\approx p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n-1})\frac{p(\mathbf{x}_n \,|\, \mathbf{y}_{n:T})}{\gamma_n(\mathbf{x}_n)} && (2.40)\\
&\propto \frac{p(\mathbf{x}_n \,|\, \mathbf{y}_{n:T})}{\gamma_n(\mathbf{x}_n)} \int p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})p(\mathbf{x}_{n-1} \,|\, \mathbf{y}_{1:n-1})\, d\mathbf{x}_{n-1}\,, && (2.41)
\end{aligned}$$

which suggests the following algorithm:

Algorithm 2.6 (Two-filter smoother) *Perform a filter forward in time, and a filter backward in time, to obtain weighted sample sets $\{(\mathbf{s}^{(i)}_n, \pi^{(i)}_n)\}$ and $\{(\tilde{\mathbf{s}}^{(i)}_n, \tilde{\pi}^{(i)}_n)\}$, respectively, for each time t_n. Then, for time t_n, let:*

$$\tilde{\psi}^{(i)}_n = \frac{\tilde{\pi}^{(i)}_n}{\gamma_n(\mathbf{x}_n = \tilde{\mathbf{s}}^{(i)}_n)} \sum_{j=1}^{P} p(\mathbf{x}_n = \tilde{\mathbf{s}}^{(i)}_n \,|\, \mathbf{x}_{n-1} = \mathbf{s}^{(j)}_{n-1})\pi^{(j)}_{n-1}\,. \tag{2.42}$$

The weighted sample set $\{(\tilde{\mathbf{s}}^{(i)}_n, \tilde{\psi}^{(i)}_n)\}$ then approximates the smooth density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T})$.

Runtime complexity of the method is $\mathcal{O}(TP^2)$ given the all-pairs transition density calculations in both Algorithms 2.5 and 2.6. Memory requirements are $\mathcal{O}(TNP)$ for storage of filter results required by the smoother.

The two-filter smoother is substantially more complex than the filter-smoother, and unfortunately has similar degeneracy issues to it. This stems from the smooth density being represented by a reweighting of the particles obtained by the backward filter; these samples may not support it adequately. The problem is exacerbated for $n \ll T$, where the mass of the filter density is likely to be further from that of the smooth density due to conditioning on fewer observations. In the extreme, during the backward pass, all weight may be heaped on a single particle, or even none at all, if there is little overlap in the significant masses of the forward filter densities and backward likelihoods. Ideally, a smoother would richly propose samples from this overlapping region, but this is not the case here.

In the absence of reverse dynamics, the potentially heuristic selection of the $\gamma_n(\mathbf{x}_n)$ densities may also inhibit its practical application.

2.3.3 Forward-backward smoother

Like the two-filter smoother, the forward-backward smoother attempts to reweight particles already obtained in order to represent the smooth densities. This time, rather than perform two filtering passes, only the forward filter is performed and an alternative backward pass provides the reweighting. In contrast to the two-filter smoother, the forward-backward smoother reweights particles from the forward filter, not an additional backward filter.

Consider the following factorisation of the smooth density:

$$
\begin{aligned}
p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T}) &= \int p(\mathbf{x}_{n:n+1} \,|\, \mathbf{y}_{1:T})\, d\mathbf{x}_{n+1} && (2.43)\\
&= \int p(\mathbf{x}_n \,|\, \mathbf{x}_{n+1}, \mathbf{y}_{1:T}) p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:T})\, d\mathbf{x}_{n+1} && (2.44)\\
&= \int p(\mathbf{x}_n \,|\, \mathbf{x}_{n+1}, \mathbf{y}_{1:n}) p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:T})\, d\mathbf{x}_{n+1} && (2.45)\\
&= \int \frac{p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n, \mathbf{y}_{1:n}) p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:T})}{p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:n})}\, d\mathbf{x}_{n+1} && (2.46)\\
&= \int \frac{p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n) p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:T})}{p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:n})}\, d\mathbf{x}_{n+1} && (2.47)\\
&= \int \frac{p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n) p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:T})}{\int p(\mathbf{x}_{n+1} \,|\mathbf{x}_n) p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})\, d\mathbf{x}_n}\, d\mathbf{x}_{n+1}\,. && (2.48)
\end{aligned}
$$

After completion of a forward filter, the filter density at each time t_n is represented by the weighted sample set $\{(\mathbf{s}_n^{(i)}, \pi_n^{(i)})\}$. Reusing these samples and applying (2.48), the following reweighting emerges:

$$
\psi_n^{(i)} = \sum_{j=1}^{P} \frac{p(\mathbf{x}_{n+1} = \mathbf{s}_{n+1}^{(j)} \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)}) \pi_n^{(i)} \psi_{n+1}^{(j)}}{\sum_{k=1}^{P} p(\mathbf{x}_{n+1} = \mathbf{s}_{n+1}^{(j)} \,|\, \mathbf{x}_n = \mathbf{s}_n^{(k)}) \pi_n^{(k)}}\,, \tag{2.49}
$$

so that the weighted sample set $\{(\mathbf{s}_n^{(i)}, \psi_n^{(i)})\}$ approximates the smooth density. Note that for each pairing of particles across times t_n and t_{n+1} the transition density between them is calculated twice. These may optionally be precalculated, although to the detriment of space complexity, arriving at [30]:

Algorithm 2.7 (Forward-backward smoother) *Perform a filter forward in time, at the conclusion of which $p(\mathbf{x}_T \,|\, \mathbf{y}_{1:T})$ is known and approximated by $\{(\mathbf{s}_T^{(i)}, \pi_T^{(i)})\}$. Initialise with $\psi_T^{(i)} = \pi_T^{(i)}$ and proceed recursively as follows:*

$$
\begin{aligned}
\alpha_n^{(i,j)} &= p(\mathbf{x}_{n+1} = \mathbf{s}_{n+1}^{(i)} \,|\, \mathbf{x}_n = \mathbf{s}_n^{(j)}) && (2.50)\\
\gamma_n^{(i)} &= \sum_{j=1}^{P} \pi_n^{(j)} \alpha_n^{(i,j)} && (2.51)\\
\delta_n^{(j)} &= \sum_{i=1}^{P} \psi_{n+1}^{(i)} \frac{\alpha_n^{(i,j)}}{\gamma_n^{(i)}} && (2.52)\\
\psi_n^{(j)} &= \pi_n^{(j)} \delta_n^{(j)}\,, && (2.53)
\end{aligned}
$$

The weighted sample set $\{(\mathbf{s}_n^{(i)}, \psi_n^{(i)})\}$ then approximates the smooth density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T})$.

Runtime complexity is $\mathcal{O}(TP^2)$ given the all-pairs transition density calculations – the $\alpha_n^{(i,j)}$ terms – in each step. Memory requirements are $\mathcal{O}(TNP)$ for storage of filter results required by the smoother. If $\alpha_n^{(i,j)}$ terms are precalculated ahead of both $\gamma_n^{(i)}$ and $\delta_n^{(i)}$ calculations, a constant factor gain to runtime is obtained at the burden of $\mathcal{O}(NP^2)$ storage.

Like the two-filter smoother, the forward-backward smoother reuses and reweights particles already obtained by a filter, and so may suffer from the same degeneracy issues.

Unlike the two-filter smoother, note that the forward-backward smoother only requires the measurements $\mathbf{y}_{1:T}$ for the forward pass, not for the backward pass.

2.3.4 Other considerations

Beyond the above reviewed methods, other strategies tend to forego the memory efficient recursive formulation and sample whole or part trajectories [31], or specialise to a subset of cases under the general state space model introduced in §2.1. We do not elaborate on these methods here, considering the three methods given to be the most established and relevant for our purposes.

2.4 Parameter estimation

Parameter estimation within the particle filter framework has been largely neglected until recently, perhaps due to its initial application to object tracking, where the model is fixed. The Bayesian approach is to simply augment the state with parameters to form $\mathbf{x}^* = \{\mathbf{x}, \boldsymbol{\theta}\}$. The particle filter may then proceed as normal, treating the parameters as any other variable.

The problem with this approach is that as the parameters are static, parameter samples are limited to those from the prior $p(\mathbf{x}_0^*)$ throughout the duration of the filter. State variables, on the other hand, vary throughout by virtue of the system dynamics and noise.

In response, artificial dynamics may be introduced to the parameters for exploratory purposes [32], a simple diffusion being the obvious starting point. Caution is advised, however, as if not chosen carefully, such dynamics may inflate the variance of parameter samples and retard or reverse convergence.

An attractive alternative form is the regularised particle filter (§2.2.4), where the kernel density estimate $p_{\mathcal{K}}(\mathbf{x}^*_n \,|\, \mathbf{y}_{1:t})$ naturally provides new parameter estimates whenever samples are redrawn. If the kernel bandwidth is designed to shrink relative to the sample variance, a refinement of new parameter samples emerges in harmony with convergence [11]. One way of achieving this is to hold h constant while standardising as in (2.24). This implicitly introduce artificial dynamics in the form of a controlled diffusion.

Another drawback to the introduction of artificial dynamics is that, now having dynamics of their own, parameter estimates vary across time and smoothing is required in order to incorporate all data into their estimate at all time points. For truly static parameters, we should expect that estimation is complete at the conclusion of the filter, where we have $p(\boldsymbol{\theta} \,|\, \mathbf{y}_{1:T})$. While the regularised particle filter introduces implicit dynamics on the parameters also, its more rigorous and controlled kernel density approach arguably mitigates this problem.

The Bayesian approach to parameter estimation, providing complete posteriors over estimates, may be forgone in favour of single point estimates. Frameworks such as EM or gradient ascent [33] may be used to obtain a single ML or other estimate. In some cases parameter estimates can be updated recursively during a single filter, but more commonly a smoother is required to provide a complete likelihood between each parameter adjustment. These schemes may be susceptible to becoming stuck in local maxima, or be prohibitively expensive given the iteration of a smoother. They provide little or no assessment of confidence or uncertainty in the parameter estimate, and may also rely on assumptions such as an analytically derived derivative of the log likelihood, which may not be available in practice.

2.5 Summary

This chapter has outlined the basic framework for recursive Bayesian learning of dynamical systems. It has defined the filtering (§2.2), smoothing (§2.3) and parameter estimation (§2.4) problems, and in particular focused on sequential Monte Carlo methods for the solution of these in the most general case. Filtering methods surveyed were the bootstrap (§2.2.2), auxiliary (§2.2.3) and regularised (§2.2.4) particle filters, with accompanying smoothing methods in the form of the filter-smoother (§2.3.1), two-filter smoother (§2.3.2) and forward-backward smoother (§2.3.3). Parameter estimation has been considered by taking either a Bayesian approach or aiming for a single parameter estimate (§2.4). The regularised particle filter was singled out as having particularly elegant properties for Bayesian parameter estimation.

Chapter 3

Continuous Time Diffusions

We now consider continuous time dynamical systems modelled by the *stochastic differential equation* (SDE):

$$d\mathbf{x} = \underbrace{\mathbf{a}(\mathbf{x}, t)\, dt}_{\text{drift}} + \underbrace{B(\mathbf{x}, t)\, d\mathbf{W}}_{\text{diffusion}}, \tag{3.1}$$

where $d\mathbf{W}$ is an increment of the multivariate *Wiener process*, a model of simple Brownian motion. The equation consists of a deterministic *drift* component and stochastic *diffusion* component. Equations such as this arise in a wide range of domains. They have particular application in the modelling of various physical phenomena, accounting for uncertainly in the model, as well as in finance and biology, where stochasticity plays an intrinsic part of the system itself.

We detail the components of such equations in this chapter, along with a practical methodology to wield them.

SDEs are an arcane art. Rather than tackling them head on, we segway down the avenue of the *ordinary differential equation* (ODE) in §3.1. This allows us to introduce several core concepts early on without the additional complications of stochasticity. SDEs are then introduced as an augmentation of ODEs in §3.2. Error control and step size adjustment are considered in §3.3, and finally some useful properties are surveyed in §3.4.

3.1 Ordinary differential equations

An ODE may be expressed in the form:

$$\frac{d\mathbf{x}}{dt} = \mathbf{a}(\mathbf{x}, t). \tag{3.2}$$

From the outset this is a special case of (3.1) where $B(\mathbf{x}, t) = \emptyset$. It represents deterministic drift dynamics only. While lacking stochasticity, it is important to realise that such deterministic models are successfully applied to a range of physical phenomenon, in particular where they naturally model position, velocity and acceleration.

(3.2) describes a *first order* ODE, in that it specifies first order derivatives only. Any differential system of higher order may be converted to a first order system by introducing additional variables, however, so we need only deal with this base case.

3.1.1 Initial value problems

An ODE is commonly used in defining an *initial value problem.* Succinctly put, given the initial state of the system $\mathbf{x}(0)$ at time 0, we wish to determine the state of the system $\mathbf{x}(t)$ at some future time t.

Clearly, we could simply integrate (3.2) with respect to t, using $\mathbf{x}(0)$ as the constant term and obtaining the function $\mathbf{x}(t)$ to solve the problem at any time t. For most real problems, however, this analytical integration is either impossible or impractical, and we do not wish to depart from the most general case.

Instead, a range of numerical schemes are available for obtaining approximate integrations. By assuming that the system is approximately linear over a short interval of time Δt, we arrive at the most basic of these numerical schemes.

Algorithm 3.1 (Euler scheme) *Let t_n and $\mathbf{x}_n \approx \mathbf{x}(t_n)$ be given. Then, for time step Δt:*

$$t_{n+1} = t_n + \Delta t \tag{3.3}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{a}(\mathbf{x}_n, t_n)\Delta t\,, \tag{3.4}$$

so that $\mathbf{x}_{n+1} \approx \mathbf{x}(t_{n+1})$.

The result of iterating this step is a sequence of time points t_n and approximations $\mathbf{x}_n \approx \mathbf{x}(t_n)$, tracing a trajectory through the system over time.

The Euler scheme is more formally derived from a first order Taylor expansion of $\mathbf{x}(t)$ about t_n:

$$\begin{aligned}\mathbf{x}(t) &= \mathbf{x}(t_n) + \dot{\mathbf{x}}(t_n)(t - t_n) + \mathcal{O}\left((t - t_n)^2\right) \\ &= \mathbf{x}(t_n) + \mathbf{a}\left(\mathbf{x}\left(t_n\right), t_n\right)\left(t - t_n\right) + \mathcal{O}\left((t - t_n)^2\right) ,\end{aligned}$$

and therefore,

$$\mathbf{x}_{n+1} \approx \mathbf{x}_n + \mathbf{a}\left(\mathbf{x}_n, t_n\right)\Delta t + \mathcal{O}\left(\Delta t^2\right) . \tag{3.5}$$

The trailing term $\mathcal{O}\left(\Delta t^2\right)$ denotes the remainder or error resulting from truncation of the Taylor series. In this case, second order terms and above have been truncated, such that the second order term dominates the error and we say it is of order 2. The *local error* of the Euler scheme is therefore $\mathcal{O}\left(\Delta t^2\right)$ and it is of *weak order* 2. The number of steps taken to solve the problem over an interval of time is proportional to $1/\Delta t$, and so the *cumulative error* across all time steps is $\mathcal{O}\left(\Delta t\right)$ and it is of *strong order* 1. Weak and strong orders are an important means of categorising numerical methods for ODEs.

We can consider schemes of higher order.

Algorithm 3.2 (Runge-Kutta scheme) *Let* t_n *and* $\mathbf{x}_n \approx \mathbf{x}(t_n)$ *be given. Then, for time step* Δt*:*

$$\begin{aligned}t_{n+1} &= t_n + \Delta t \qquad (3.6)\\ \mathbf{x}_{n+1} &= \mathbf{x}_n + \frac{\Delta t}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right) , \qquad (3.7)\end{aligned}$$

where

$$\begin{aligned}\mathbf{k}_1 &= \mathbf{a}(\mathbf{x}_n, t_n) \qquad (3.8)\\ \mathbf{k}_2 &= \mathbf{a}(\mathbf{x}_n + \frac{\Delta t}{2}k_1, t_n + \frac{\Delta t}{2}) \qquad (3.9)\\ \mathbf{k}_3 &= \mathbf{a}(\mathbf{x}_n + \frac{\Delta t}{2}k_2, t_n + \frac{\Delta t}{2}) \qquad (3.10)\\ \mathbf{k}_4 &= \mathbf{a}(\mathbf{x}_n + \Delta t k_3, t_n + \Delta t) \qquad (3.11)\end{aligned}$$

This represents the classic Runge-Kutta with weak order 5 and strong order 4. Runge-Kutta schemes may be generalised to any order using the following framework:

Algorithm 3.3 (Abstract Runge-Kutta scheme) *Let t_n and $\mathbf{x}_n \approx \mathbf{x}(t_n)$ be given. Then, for time step Δt:*

$$t_{n+1} = t_n + \Delta t \tag{3.12}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta t \sum_{i=1}^{s} b_i \mathbf{k}_i \,, \tag{3.13}$$

where:

$$\mathbf{k}_1 = \mathbf{a}(\mathbf{x}_n, t_n) \tag{3.14}$$

$$\mathbf{k}_i = \mathbf{a}(\mathbf{x}_n + \Delta t \sum_{j=1}^{i-1} a_{i,j}\mathbf{k}_j, t_n + c_i \Delta t) \tag{3.15}$$

The task is then to select the number of stages s and coefficients $a_{i,j}$, b_i and c_i to satisfy the required order conditions. These are commonly represented as the tableau:

$$\begin{array}{c|cccc}
0 & & & & \\
c_2 & a_{2,1} & & & \\
c_3 & a_{3,1} & a_{3,2} & & \\
\vdots & \vdots & & \ddots & \\
c_s & a_{s,1} & a_{s,2} & \dots & a_{s,s-1} \\
\hline
 & b_1 & b_2 & \dots & b_s
\end{array}$$

Using this notation, the classic Runge-Kutta of Algorithm 3.2 is expressed as:

$$\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}$$

In general, higher order methods have smaller error than lower order methods given the same time step. Alternatively, given a fixed error bound, a higher order method may take fewer, larger time steps to arrive at a solution within that bound. Higher order methods do require additional calculation within each time step, but this is generally less than the calculation saved in taking fewer steps. As a rule of thumb, gains are observed up to an order of about 8 [34]. For some problems, the nature of the system may be such that a higher order method is unable to take larger steps than a lower order method while still maintaining accuracy anyway, so that the increased computational cost at each step is detrimental [34].

3.1.2 Implicit methods

Up to this point, all the methods described are considered *explicit* in the sense that $\mathbf{x}_{n+1}$ is expressed as a function of $\mathbf{x}_n$. Another important family of methods are the *implicit* methods.

To motivate this, consider an alternative derivation of the Euler scheme (Algorithm 3.1), by observing:

$$\dot{\mathbf{x}}(t) \approx \frac{\mathbf{x}(t+\Delta t) - \mathbf{x}(t)}{\Delta t}\,, \tag{3.16}$$

and rearranging to:

$$\mathbf{x}(t+\Delta t) \approx \mathbf{x}(t) + \dot{\mathbf{x}}(t)\Delta t\,, \tag{3.17}$$

which is the explicit Euler scheme, also known as the *forward* Euler scheme. Consider a similar *backward* derivation:

$$\dot{\mathbf{x}}(t) \approx \frac{\mathbf{x}(t) - \mathbf{x}(t-\Delta t)}{\Delta t}\,, \tag{3.18}$$

so that by rearranging:

$$\mathbf{x}(t) \approx \mathbf{x}(t-\Delta t) + \dot{\mathbf{x}}(t)\Delta t\,, \tag{3.19}$$

or equivalently:

$$\mathbf{x}(t+\Delta t) \approx \mathbf{x}(t) + \dot{\mathbf{x}}(t+\Delta t)\Delta t\,. \tag{3.20}$$

This leads directly to the implicit Euler scheme, also known as the backward Euler scheme for reasons that the above derivation makes clear:

Algorithm 3.4 (Implicit Euler scheme) *Let t_n and $\mathbf{x}_n \approx \mathbf{x}(t_n)$ be given. Then, for time step Δt:*

$$t_{n+1} = t_n + \Delta t \tag{3.21}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{a}(\mathbf{x}_{n+1}, t_{n+1})\Delta t\,. \tag{3.22}$$

The scheme is implicit because $\mathbf{x}_{n+1}$ is now expressed in terms of both $\mathbf{x}_n$ and itself, such that solving the equation is necessary to advance the scheme. This may be done numerically using a root finding algorithm such as Newton's method.

While on the surface implicit methods may seem retrograde, in practice they have an important role in the solution of *stiff* systems. Loosely speaking, a stiff system is simply a system that is difficult to solve numerically due to instability. This is usually due to fluctuating components at multiple scales that cause rapid variation in the solution, such that the accumulation of error causes a numerical scheme to diverge from the true solution. [35] provides detailed treatment.

Like its explicit counterpart, the implicit Euler scheme is of weak order 2 and strong order 1. Higher order implicit schemes, including implicit Runge-Kutta schemes, are available.

3.2 Stochastic differential equations

With an appreciation of ODEs as the deterministic drift component of an SDE, we now introduce a stochastic *diffusion* term $B(\mathbf{x},t)\boldsymbol{\xi}(t)$ into (3.2) to obtain the Langevin equation [17]:

$$\frac{d\mathbf{x}}{dt} = \mathbf{a}(\mathbf{x},t) + B(\mathbf{x},t)\boldsymbol{\xi}(t)\,. \tag{3.23}$$

This is naive for reasons given below, but is nonetheless an appealing way to first conceptualise an SDE. Intuitively, the SDE consists of the deterministic drift component $\mathbf{a}(\mathbf{x},t)$ and the stochastic diffusion component $B(\mathbf{x},t)\boldsymbol{\xi}(t)$. The most salient aspect of the equation is the nature of the stochastic process $\boldsymbol{\xi}(t)$. In general, this is a "rapidly and irregularly fluctuating random function of time" [17, p.80], mathematically idealised by defining $\boldsymbol{\xi}(t')$ and $\boldsymbol{\xi}(t)$ as statistically independent for $t' \neq t$. For simplicity, but without loss of generality, we enforce $\mathbb{E}(\boldsymbol{\xi}(t)) = \mathbf{0}$; a nonzero mean may be absorbed into the drift term.

In practice, one of a number of processes with nice properties are usually chosen as a basis to approximate $\boldsymbol{\xi}(t)$. The most common of these is derived from the *Wiener process* [17, §3.8.1] $\mathbf{W}(t)$, a diffusion process where $\mathbf{W}(t+\Delta t) - \mathbf{W}(t) \sim$

$\mathcal{N}(\mathbf{0}, I\sqrt{\Delta t})$. Note that this is a Markov process, with continuous but highly irregular sample paths that are nowhere differentiable, mean approaching zero but variance approaching infinity as $\Delta t \to \infty$. We accept these properties without further discussion, although point the reader to [17] for detailed results.

The Wiener process is introduced into an SDE by letting:

$$\int_0^t \boldsymbol{\xi}(t')dt' = \mathbf{W}(t) \tag{3.24}$$

We now have the paradox that $\mathbf{W}(t)$ is the integral of $\boldsymbol{\xi}(t)$, yet not itself differentiable. This highlights the naivety of (3.23). Noting that by (3.24):

$$d\mathbf{W}(t) \equiv \mathbf{W}(t + dt) - \mathbf{W}(t) \tag{3.25}$$

$$= \boldsymbol{\xi}(t)\, dt\,, \tag{3.26}$$

rewrite (3.23) as:

$$d\mathbf{x} = \mathbf{a}(\mathbf{x}, t)\, dt + B(\mathbf{x}, t)\, d\mathbf{W}(t) \tag{3.27}$$

$d\mathbf{W}(t)$ is then usually abbreviated as $d\mathbf{W}$ for notational convenience, arriving at the original form of the SDE given in (3.1).

3.2.1 Initial value problems

Integration of SDEs is intrinsically dependent on discretisation of the stochasticity. Two schemes exist, the *Itô* [17, §4.2.1] and *Stratonovich* [17, §4.2.3] integrals, derived from the forward Euler and trapezoidal discretisations, respectively [36]. These give slightly different results based on their discretised interpretation of the Wiener process. Fokker-Planck equations may also provide an avenue for integration [17][ch.5].

As for ODEs, integration of SDEs is attractive in that it provides efficient analytical solutions to initial value problems. Again, however, this integration is often intractable, and numerical methods are perhaps more fruitful for general application.

The stochastic equivalent of the forward Euler scheme exploits its linear discretisation to preserve the Gaussianity of the Wiener noise through the time step:

Algorithm 3.5 (Euler-Maruyama scheme) *Let t_n and $\mathbf{x}_n \approx \mathbf{x}(t_n)$ be given. Then, for time step Δt and Wiener increment $\Delta\mathbf{W} \sim \mathcal{N}(\mathbf{0}, I\sqrt{\Delta t})$:*

$$t_{n+1} = t_n + \Delta t \tag{3.28}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{a}(\mathbf{x}_n, t_n)\Delta t + B(\mathbf{x}_n, t_n)\Delta\mathbf{W}\,. \tag{3.29}$$

Higher order schemes pose a greater challenge, as each step now squeezes the noise increment through a nonlinear function, such that Gaussianity is not maintained. It is therefore not sufficient to simply apply a single step of a Runge-Kutta, say, then add a noise sample [37].

The general Runge-Kutta framework established for ODEs (Algorithm 3.3) still stands, but establishing appropriate coefficients for a given order condition is more challenging due to the diffusion component. A particularly attractive and practical option is based on the observation that any scheme for ordinary differential equations (ODEs) can be adapted to SDEs with approximately half the order [38]. The crux of this relies on converting the Itô equation (3.1) into its equivalent form under the Stratonovich interpretation of SDEs [35, p157]:

$$d\mathbf{x} = \left[\mathbf{a}(\mathbf{x}, t) - \frac{1}{2}\sum_i \frac{\partial B(\mathbf{x}, t)}{\partial x_i} B_{i,*}(\mathbf{x}, t)^T\right] dt + B(\mathbf{x}, t)\, d\mathbf{W}\,, \tag{3.30}$$

where $B_{i,*}(\mathbf{x}, t)$ is the ith row of $B(\mathbf{x}, t)$. The extra term arises as a result of calculating the derivative at the midpoint of the increment under Stratonovich, rather than at the beginning as under Itô. Once using the Stratonovich interpretation, the standard chain rule of calculus applies, unlike under Itô, so that existing solvers for ODEs may be applied to the system. All of this leads to the following extension of the general Runge-Kutta scheme for Itô SDEs:

Algorithm 3.6 (Abstract stochastic Runge-Kutta scheme) *Let t_n and $\mathbf{x}_n \approx \mathbf{x}(t_n)$ be given. Then, for time step Δt and Wiener increment $\Delta\mathbf{W} \sim \mathcal{N}(\mathbf{0}, I\sqrt{\Delta t})$, let*

$$\mathbf{h}(\mathbf{x}, t, \Delta t, \Delta\mathbf{W}) = \mathbf{a}(\mathbf{x}, t) - \frac{1}{2}\sum_i \frac{\partial B(\mathbf{x}, t)}{\partial x_i} B_{i,*}(\mathbf{x}, t)^T + B(\mathbf{x}, t)\frac{\Delta\mathbf{W}}{\Delta t}$$

where $B_{i,*}(\mathbf{x}, t)$ is the ith row of $B(\mathbf{x}, t)$. Then:

$$t_{n+1} = t_n + \Delta t \tag{3.31}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta t \sum_{i=1}^{s} b_i \mathbf{k}_i\,, \tag{3.32}$$

where:

$$\mathbf{k}_1 = \mathbf{h}(\mathbf{x}_n, t_n, \Delta t, \Delta\mathbf{W}) \tag{3.33}$$

$$\mathbf{k}_i = \mathbf{h}(\mathbf{x}_n + \Delta t \sum_{j=1}^{i-1} a_{i,j}\mathbf{k}_j, t_n + c_i\Delta t, \Delta t, \Delta\mathbf{W}) \tag{3.34}$$

Using this conversion, it is possible to reapply any method designed for ODEs to SDEs.

3.3 Error control

To this point, we have neglected the matter of choosing an appropriate step size Δt. In the simplest case, this is fixed for all time steps to a length that is small enough to maintain some error threshold, while large enough for integration over the interval of interest to be computationally tenable. The problem with this is that the step size will likely be reduced universally by a handful of erratic regions where the integrator must tiptoe or risk inaccuracy. An *adaptive* scheme is preferable – reducing the step size for difficult regions while increasing it where possible to quickly stride over tame behaviour.

In order to establish such a scheme, we require:

- a means of estimating the error,
- an error bound,
- a means of adjusting the step size, decreasing it and repeating a step if the error bound is exceeded, and increasing it for the next step otherwise.

We will address each of these in turn in this section.

3.3.1 Error estimation

A simple means of error estimation is to compare the result of a single step of size Δt to two steps of size $\Delta t/2$. The difference between the two provides an approximation of the error in the single Δt step. While simple, this has been observed to give unsatisfactory results by at least one author in the particular case of SDEs [39].

Another option is to compare the results of two numerical schemes of different order, using the higher order method to provide error estimates validating the results of the lower order method. The drawback here, of course, is the need to perform the integration twice. The family of *embedded* Runge-Kutta schemes are attractive here (see e.g. [34; 40; 41]). By sharing coefficients and intermediate evaluations, such methods efficiently produce two approximations at each time step, one from a qth order, and one from a lower pth[1] order Runge-Kutta.

At time t_n, if $\mathbf{x}_n$ is the estimate of the qth order Runge-Kutta and $\hat{\mathbf{x}}_n$ that of the pth order, an approximation of the error is given by:

$$\boldsymbol{\epsilon}_n = \Delta t|\mathbf{x}_n - \hat{\mathbf{x}}_n|\,, \tag{3.35}$$

where $|\cdot|$ is simply an element-wise absolute value.

It is worth noting that this is an estimate of *local* error only, not cumulative error. Other methods exist for the estimate of cumulative error (see e.g. [39]), although they are perhaps motivated more by validation than the pursuit of adaptive step size control. Other recent methods attempt to split error into separate contributions by the drift and diffusion components [42].

3.3.2 Error bound

The setting of an appropriate error bound is essentially arbitrary, but we outline a fairly general idea here that incorporates most important considerations. It includes both an absolute error term, and errors relative to the magnitude of both $\mathbf{x}_n$ and its derivative $\dot{\mathbf{x}}_n$. It is, in fact, the scheme used by the ODE components of the GNU Scientific Library[2].

Let $\delta_{\mathrm{abs}} \geq 0$ be the maximum permissable absolute error in any component of $\mathbf{x}$ and $\delta_{\mathrm{rel}} \geq 0$ the maximum permissable relative error. Let $\beta_x \geq 0$ and $\beta_{dx} \geq 0$ be scaling factors. Then, the error bound at time t_n is given by:

$$\boldsymbol{\delta}_n = \boldsymbol{\delta}_{\mathrm{abs}} + \delta_{\mathrm{rel}}(\beta_x|\mathbf{x}| + \beta_{dx}|\dot{\mathbf{x}}|)\,, \tag{3.36}$$

where $|\cdot|$ is again simply an element-wise absolute value.

[1]Usually $q = p + 1$

[2]`http://www.gnu.org/software/gsl/`

3.3.3 Step size adjustment

At time t_n, given the error estimate $\boldsymbol{\epsilon}_n$ and error bound $\boldsymbol{\delta}_n$, let:

$$\gamma = \|\boldsymbol{\epsilon}_n \oslash \boldsymbol{\delta}_n\|_\infty \tag{3.37}$$

where $\oslash$ is element-wise division and $\|\cdot\|_\infty$ the ∞-norm, that is, the component of the vector with greatest magnitude. If $\gamma > 1$ then the error in at least one component has exceeded the error bound, so the step is rejected and undone. If $\gamma \leq 1$ the step is accepted and the integration proceeds as normal. In either case, the step size for the next step is adjusted to:

$$\Delta t \leftarrow \Delta t \cdot \frac{9}{10}\gamma^{-\frac{1}{p+1}} \tag{3.38}$$

The $\frac{9}{10}$ here is a "safety factor". In practice, it can be useful to put a lower and upper bound on the maximum change in step size also.

The only additional point worth making for the stochastic setting is that a Wiener increment sampled for a rejected time step should be preserved for conditioning the increment samples for proceeding proposals [39]. This is important to ensure that the integration proceeds along the same Wiener trajectory, and that unlikely Wiener increments are not inadvertently treated as unnacceptable error and discarded.

3.4 Properties of diffusions

In this final section we pluck out a number of salient properties of diffusion processes. The list is not meant to be exhaustive, and is biased in favour of those that will play a significant part later in this work.

3.4.1 Autocorrelation

The autocorrelation $R(s,t)$ of a process is the correlation between its states at two different times s and t, with $t > s$. For a Markov process, its properties are only dependent upon time differences, not absolute times, and so $R(s,t) = R(t-s)$. For such processes the autocorrelation is calculated across *lag times*. For a single lag time Δt and a sequence of samples $\mathbf{x}_1, \ldots, \mathbf{x}_T$ taken from the process at times equispaced by Δt, the autocorrelation may be estimated as:

$$R(\Delta t) = \left[\frac{1}{T}\sum_{i=1}^{T}\left(\mathbf{x}_i\mathbf{x}_{i-1}^T\right) - \boldsymbol{\mu}\boldsymbol{\mu}^T\right]\Sigma^{-1}\,. \tag{3.39}$$

Here, $\boldsymbol{\mu}$ is the mean of the process and Σ its covariance. If the calculation is not normalised by Σ it provides the *autocovariance*. Usually, $\boldsymbol{\mu}$ and Σ are not available analytically, and are calculated using the standard sample formulas, which provide a biased estimate of the autocorrelation.

In introducing autocorrelation, it is worth mentioning as an aside the alternative numerical integration scheme of *local linearization* [8]. Unlike the Euler-Maruyama scheme, which performs a linear step based on the gradient at time t, the local linearization scheme performs a linear step based on the autocorrelation function for a lag time equal to the proposed step size.

3.4.2 Stationarity

A process is considered *stationary* if its statistics are invariant under time translation. That is, for any Δt:

$$p(\mathbf{x}(t_1) = \mathbf{s}_1, \ldots, \mathbf{x}(t_T) = \mathbf{s}_T) = p(\mathbf{x}(t_1 + \Delta t) = \mathbf{s}_1, \ldots, \mathbf{x}(t_T + \Delta t) = \mathbf{s}_T)\,. \tag{3.40}$$

In such a case the one-time probability is independent of time:

$$p(\mathbf{x}(t)) = p_s(\mathbf{x})\,, \tag{3.41}$$

where $p_s(\mathbf{x})$ is the *equilibrium distribution* of $\mathbf{x}$. Further, the two time joint probability is dependent only on time differences:

$$p(\mathbf{x}(t_1) = \mathbf{s}_1, \mathbf{x}(t_2) = \mathbf{s}_2) = p_s(\mathbf{x}(t_1 - t_2) = \mathbf{s}_1, \mathbf{x}(0) = \mathbf{s}_2)\,, \tag{3.42}$$

as is the conditional:

$$p(\mathbf{x}(t_1) = \mathbf{s}_1 \,|\, \mathbf{x}(t_2) = \mathbf{s}_2) = p_s(\mathbf{x}(t_1 - t_2) = \mathbf{s}_1 \,|\, \mathbf{x}(0) = \mathbf{s}_2)\,. \tag{3.43}$$

Now, given a stationary process and starting state $\mathbf{s}_0$, define the Markov chain:

$$p(\mathbf{x}(t)) = p_s(\mathbf{x}(t) \,|\, \mathbf{x}(t_0) = \mathbf{s}_0) \tag{3.44}$$

$$p(\mathbf{x}(t) \,|\, \mathbf{x}(t') = \mathbf{s}') = p_s(\mathbf{x}(t) \,|\, \mathbf{x}(t') = \mathbf{s}') \tag{3.45}$$

Then, as $t \to \infty$ or $t_0 \to -\infty$:

$$p(\mathbf{x}(t)) \to p_s(\mathbf{x}) \tag{3.46}$$

3.5 Summary

This chapter has introduced continuous-time diffusion processes through the formalism of stochastic differential equations (SDEs). It eased into the topic in §3.1 with ordinary differential equations (ODEs), framed as deterministic drift processes, and described various numerical schemes for their solution. §3.2 then introduced a stochastic diffusion component to extend these to SDEs. For the numerical solution of such problems, it advocated the very practical approach of converting an Itô SDE to its equivalent Stratonovich form, in order to apply any of the myriad of numerical methods developed for ODEs. §3.3 discussed the practical matters of error estimation, error bounds and adaptive step size control, critical for both the accuracy and computational efficiency of these numerical schemes. Finally, §3.4 presented a number of salient properties that will prove useful later in this work.

Chapter 4

A Fusion of Sorts

Having introduced Bayesian methods for learning in dynamical systems (§2), and continuous-time stochastic diffusions (§3), we now attempt to fuse the two, considering Bayesian learning of continuous time dynamical systems. We find that the results are unsatisfactory, and in this chapter discuss the outstanding challenges for a harmonious union of the two. In doing so, this chapter firmly lays out the context of this work and the problems that it seeks to address.

§4.1 combines the theory of dynamical systems with that of continuous-time stochastic diffusions to develop continuous-time dynamical systems. Problems in applying conventional methods to such systems are outlined in §4.2, with a number of methods known to work, with various shortcomings, provided in §4.3. Finally, experimental results for these relevant methods applied to a number of artificial problems are given in §4.4.

4.1 Continuous-time dynamical systems

Consider the general dynamical system formulation of §2.1 in the specific case where time t is continuous and the transition function $\mathbf{f}(\mathbf{x}, \mathbf{v}, \boldsymbol{\theta}, t)$ is known only by its first order derivatives, given by an SDE:

$$d\mathbf{f} = \mathbf{a}(\mathbf{x}, \boldsymbol{\theta}, t)\, dt + B(\mathbf{x}, \boldsymbol{\theta}, t)\, d\mathbf{W}\,. \tag{4.1}$$

In contrast to the general SDE definition of (3.1), note the introduction of the dynamical system's static parameters $\boldsymbol{\theta}$, and that system noise $\mathbf{v}$ is now sourced from the Wiener process $\mathbf{W}$. The remainder of the dynamical system formulation is consistent with the use of continuous time and does not change.

This formulation is sometimes referred to as the *continuous-discrete* model, acknowledging that while the state now transitions continuously, measurements remain restricted to discrete time points. This is in contrast to the *continuous-continuous* model, where the measurement too is a continuous process. In a practical setting this is arguably irrelevant – a continuous measurement can only derive from an analog signal, as digitisation alone would constitute discretisation, making such a model unimplementable on modern computers. The continuous signal could presumably only arise from some form of interpolation or curve fitting to discrete time samples. We are yet to see a real world application of the continuous-continuous model, and so do not believe our formulation here is significantly limited by the assumption of discrete time measurements.

4.2 Challenges

We now consider some of the conflicts that emerge in the combination of these two ideas.

Firstly and most importantly, consider the transition density $p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})$. This will only be available in an exact or approximate analytical form in one of the following two cases:

1. The SDE is integrable, such as by using Itô or Stratonovich calculus, or Fokker-Planck equations. In this case $\mathbf{f}(\cdot)$ can be determined exactly.
2. A linear discretisation scheme is used, such as Euler-Maruyama or local linearisation. In this case, $p(\mathbf{x}(t + \Delta t) \,|\, \mathbf{x}(t))$ can be calculated if a single step of size Δt is taken by the integrator.

The first of these is too limited in scope to include many real world problems of interest. The second requires that filtering and smoothing steps are the same size as steps of the numerical integration scheme. This is because the transition density is only analytical for a single step of the discretisation, so that multiple steps may not be made between each filtering or smoothing step. T, the number of times at which the filter or smoother is evaluated, increases, and cannot vary independently of the numerical scheme. This can be extremely expensive, especially so if measurements are sparse, and many additional filtering steps must be inserted between measurement times.

The effects of this on a filter are generally not too great, as particles must be propagated throughout the entire length of time regardless of the number of steps that the filter takes to achieve this. The same cannot be said for a smoother, however, where it may not be necessary to perform particle propagations at all, and the number of its more complex $\mathcal{O}(P^2)$ steps must fit within computational limits.

When one considers that the step size is also being fixed to that of a low-order numerical scheme with necessarily small step size compared to higher-order schemes, the problem is numbingly apparent. Furthermore, an adaptive step size is hard to fit into this framework – all particles must progress through a single step of equal size in order to end at the same time for the calculations at each filtering step. Either a step size appropriate for all particles is adaptively selected – a “lowest common denominator” situation where all particles progress as slowly as the most difficult – or a constant step size is used.

Consequently, we must, except in the most simple of cases, assume that $\mathbf{f}(\cdot)$ is unknown and that the transition density $p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})$ is analytically intractable. In fact, even for simple cases, if we wish to make use of our knowledge of faster higher-order integration schemes we must assume the same.

This is not to say that it is impossible to simulate *realisations* of $p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})$. Indeed, the numerical integration schemes of §3 facilitate precisely this. To do so, simply construct the initial value problem beginning at some particular $\mathbf{x}_{n-1}$ and simulate for time $t_n - t_{n-1}$. However, for arbitrary $\mathbf{x}_n$, calculation of the transition density given a particular starting point is allusive.

The implications of this are formidable – the general formulation of the particle filter (Algorithm 2.2), backward particle filter (Algorithm 2.5), two-filter smoother (Algorithm 2.6) and forward-backward smoother (Algorithm 2.7) all assume analytical knowledge of this density. At face value, at least, this makes them inapplicable to modelling with continuous-time dynamical systems except when a potentially poor linear discretisation is used.

Another obvious observation to make is the computational cost of simulating realisations from the transition. Compared to discrete-time models which will generally require only a single function evaluation, propagation of a particle through the SDEs requires multiple steps of an appropriate numerical scheme. While efficient methods exist, as surveyed in §3, this is still a relatively expensive operation.

Finally, consider the backward particle filter and the use of reverse dynamics. Naively, reversing the dynamics of an SDE system is a simple negation of derivatives. Unfortunately, this usually results in a divergent system for which the cumulative error of any numerical scheme causes instability. This precludes use of the reverse dynamics in, for example, a two-filter smoother.

4.3 Rescuing methods

Having proclaimed what will not work in the continuous time setting, it is of course worth considering what will. Recall that the simplicity of the bootstrap particle filter (Algorithm 2.1) is its use of the transition density as proposal. By letting $q(\mathbf{x}_n) = p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})$, it cancels the intractable transition density in the weight calculation of Algorithm 2.2. The simple bootstrap therefore remains applicable in the continuous time setting.

We can generalise this to any proposals of the form $q'(\mathbf{x}_{n-1:n}) = p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})q(\mathbf{x}_{n-1})$. All such proposals cancel the intractable transition density, and observe that $q(\mathbf{x}_{n-1})$ essentially represents a resampling of the filter density at time t_{n-1}. Any such proposals, which we will refer to as *resampling proposals*, are applicable in continuous time. In addition to the bootstrap, these include the auxiliary (Algorithm 2.3) and regularised (Algorithm 2.4) particle filters previously reviewed. Proposals which do not take this form, such as the Gaussian of the unscented particle filter [27] (acquired via the unscented Kalman filter equations [19] rather than propagation of particles) leave the transition density lingering and so unfortunately remain inapplicable.

The filter-smoother (§2.3.1) remains relevant, requiring no further density calculations over its coupling with a workable resampling proposal. Unfortunately, the two-filter (Algorithm 2.6) and forward-backward (Algorithm 2.7) smoothers cannot be easily manipulated to relieve their dependence on the transition density[1]. They remain applicable only in the two cases identified in §4.2.

4.4 Experiments

We demonstrate these applicable conventional methods using a number of SDE model examples which will be reused in later chapters as a baseline for comparision. The models are introduced in detail here, including discussion of the particular nuances which make them interesting subjects of study.

4.4.1 Toy

Our first example is a very simple linear model simulating a body moving in 2d space. The interest in this model is that it may be solved exactly using analytical methods, and so provides a basis to assess sampling and numerical error in particle methods. The state is given by $(a, b, v)^T$, where (a, b) is the body's x and y coordinate, and v its translational velocity in some fixed direction ϑ. The system dynamics are given by the equations:

$$
\begin{aligned}
da &= v \cos\vartheta \, dt + \sigma_a \, dW && (4.2)\\
db &= v \sin\vartheta \, dt + \sigma_b \, dW && (4.3)\\
dv &= -\gamma v \, dt + \sigma_v \, dW \, . && (4.4)
\end{aligned}
$$

Exact integration of the equations yields the autoregressive for a lag time of 1:

$$
\mathbf{x}_{n+1} = A\mathbf{x}_n + \Sigma_x \mathbf{w} \, , \tag{4.5}
$$

[1]Although doing so is one of the main contributions of this work.

where:

$$A = \begin{pmatrix} 1 & 0 & \cos\vartheta \\ 0 & 1 & \sin\vartheta \\ 0 & 0 & 1-\gamma \end{pmatrix} \tag{4.6}$$

$$\Sigma_x = \begin{pmatrix} \sigma_a & 0 & 0 \\ 0 & \sigma_b & 0 \\ 0 & 0 & \sigma_v \end{pmatrix} \tag{4.7}$$

and $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, I)$.

Measurements are simply a noisy observation of the position, given by:

$$\mathbf{y}_n = C\mathbf{x}_n + \Sigma_y \mathbf{v} \tag{4.8}$$

where:

$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{4.9}$$

$$\Sigma_y = \begin{pmatrix} \sigma_y & 0 \\ 0 & \sigma_y \end{pmatrix} \tag{4.10}$$

and $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, I)$.

A Gaussian prior over the state is given by Table 4.1.

An exact solution to the problem, up to accumulated floating point error, may be obtained using the Kalman filter and Rauch-Tung-Striebel (RTS) smoother [43]. As the system is linear, the transition density is tractable and a solution, up to sampling and numerical integration error, may be obtained using a particle filter and forward-backward or two-filter smoother.

We fix $\vartheta = .8$, $\gamma = .1$, $\sigma_a = \sigma_b = \sigma_v = .1$ and $\sigma_y = 1$. Figure 4.1 then plots the results of the filters and Figure 4.2 the smoothers.

4.4.2 Double well

The double well is a simple one dimensional stochastic differential model of the form:

$$dx = 4x(1 - x^2)\,dt + \sigma_x\,dW\,, \tag{4.11}$$

	Mean	Variance
a	1	2
b	1	2
v	1	.1

Table 4.1: Prior over the variables of the toy model.

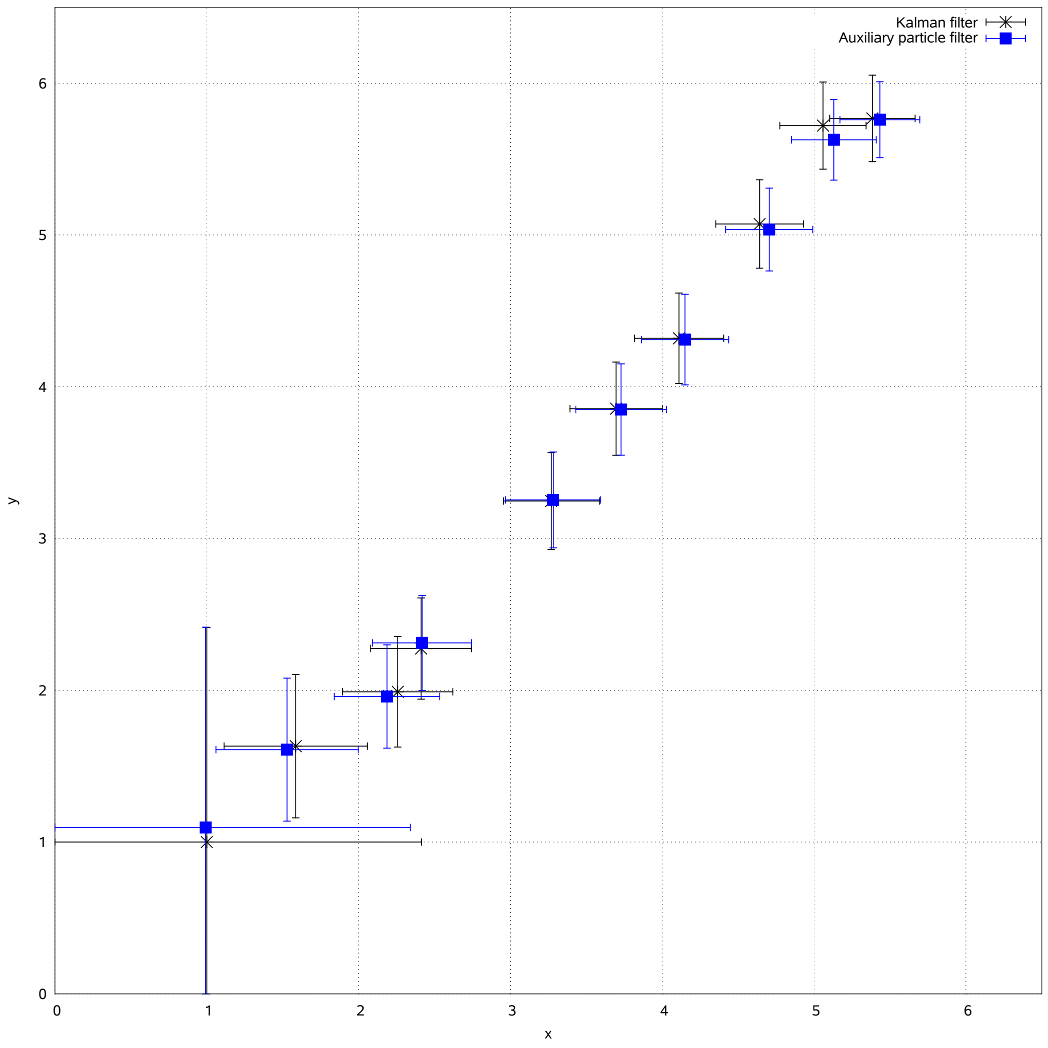


Figure 4.1: Filtered (x, y) position estimates for the toy model. Each point represents the mean for a particular method at a particular time, with bars indicating two standard deviations in either direction. The Kalman filter results, being analytical, may be considered ground truth.

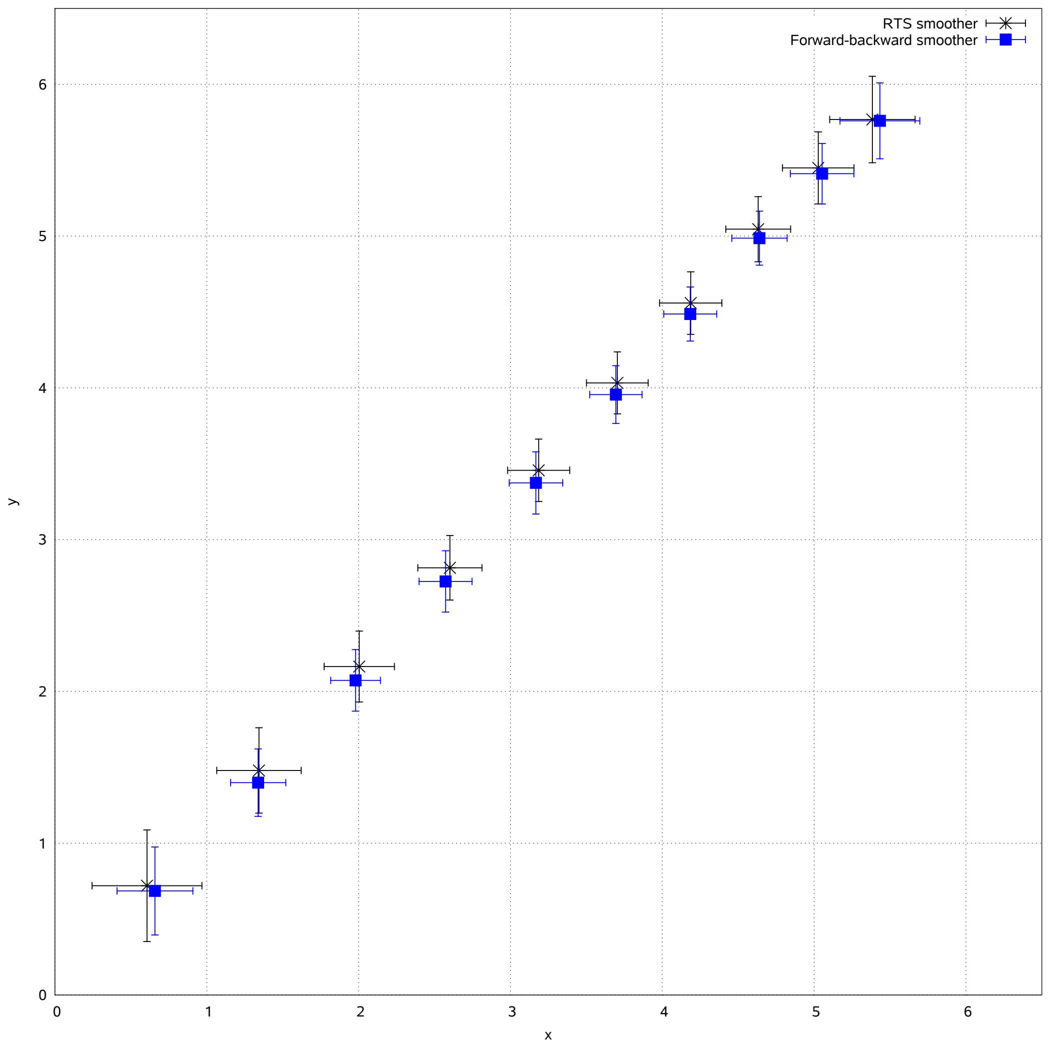


Figure 4.2: Filtered (x, y) position estimates for the toy model. Each point represents the mean for a particular method at a particular time, with bars indicating two standard deviations in either direction. The Rauch-Tung-Striebel (RTS) smoother results, being analytical, may be considered ground truth.

	1st component		2nd component	
	Mean	Variance	Mean	Variance
x	.893	.107	-.893	.107
$\ln \sigma_x$	$\ln 1$	.1	$\ln 1$	.1
$\ln \sigma_y$	$\ln .2$	.4	$\ln .2$	.4

Table 4.2: Prior over the variables and parameters of the double well model.

which, for the purposes of the filtering framework presented here, we will assume can be observed with some additional error $\xi \sim \mathcal{N}(0, 1)$, diffused by a parameter σ_y, so that:

$$y = x + \sigma_y \xi \,. \tag{4.12}$$

The system has stable states at $x = \pm 1$, shifting between these at a regularity comensurate with the magnitude of the diffusion parameter σ_x. The pair of stable states is the saliency in this otherwise simple system. Stochasticity provides the sole means of transitioning from one state to the other, without which the system would settle into one or other of the states, depending on its initial value. In this way, the double well represents a simple system in which stochasticity is the essential driver of the dynamics of interest.

To establish intuition, Figure 4.3 plots single trajectories simulated from the system for various values of the diffusion parameter σ_x. Note the clear distinction between the two stable states. Increasing σ_x produces systems that shift more regularly between the two stable states. This manifests as a reduction in autocorrelation (Figure 4.4) and a broadening of the modes of the equilibrium distribution (Figure 4.5).

A prior is established over x, $\ln \sigma_x$ and $\ln \sigma_y$ to scope the system into the range of interesting dynamics. This takes the form of a two component mixture of Gaussians, with means and covariances as in Table 4.2. The means and variances for x are obtained by fitting the Gaussian mixture to a sample based representation of the equilibrium distribution for $\sigma_x = .8$ using Expectation-Maximisation.

With the model now defined, we proceed with methodological aspects, beginning with a comparison of various numerical integration schemes for handling the system. After this comparison, we apply applicable methods to the filter-

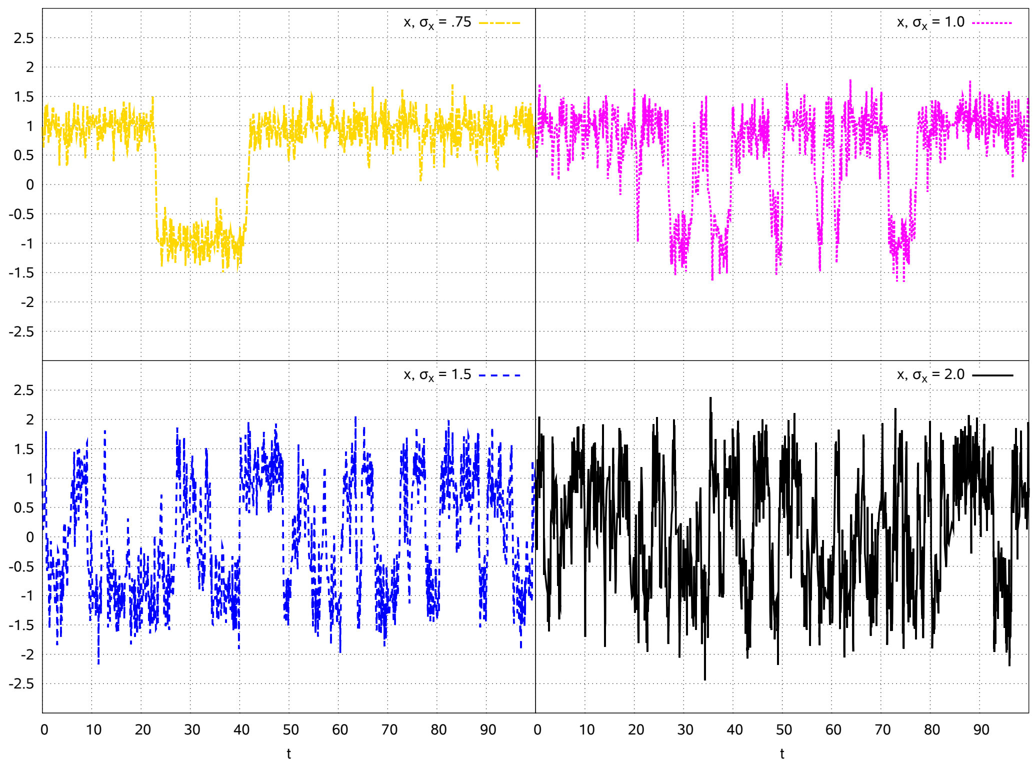
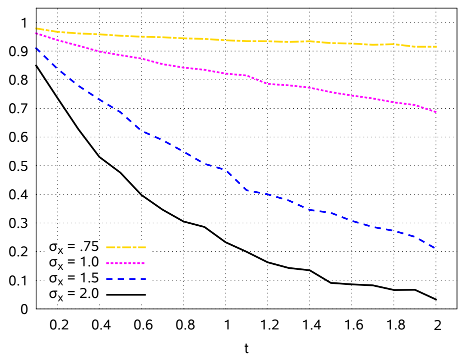


Figure 4.3: Sample trajectories simulated from the double well system with varying values of the diffusion parameter σ_x.



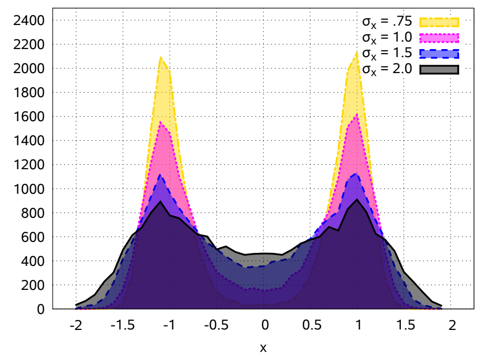


Figure 4.4: Autocorrelation of the double well system with varying values of the diffusion parameter σ_x.

Figure 4.5: Equilibrium distribution of the double well system with varying values of the diffusion parameter σ_x.

ing, smoothing and parameter estimation problems over this model, presenting detailed results for later discussion.

4.4.2.1 Numerical integration

Trajectories may be simulated using any one of a number of numerical schemes adapted to SDEs, as reviewed in Chapter 3. Ultimately we wish to select a scheme which provides a discretisation capturing the interesting behaviour of the system, and which is computationally efficient. We investigate the application of seven different schemes to the double well system:

EM(1) Explicit Euler-Maruyama (order 1),
RK(2) Explicit embedded Runge-Kutta (order 2/3),
RK(4) Explicit embedded Runge-Kutta-Fehlberg (order 4/5) [40],
RK(8) Explicit embedded Runge-Kutta-Prince-Dormand (order 8/9) [41],
EM(1)IMP Implicit Euler-Maruyama (order 1),
RK(2)IMP Implicit Runge-Kutta (order 2),
RK(4)IMP Implicit Runge-Kutta (order 4), and

All methods are implemented as in `dysii`, which builds these SDE schemes around the ODE schemes of the GSL. The EM(1) method approximates error by comparing each step with two half steps, halving or doubling the step size appropriately in response. All other methods use the error control procedure used in the GSL, outlined in §3.3.

For confidence, it is worthwhile establishing the validity and consistency of these methods before proceeding. For the remainder of this section we fix $\sigma_x = 1$ and error bounds $\delta_{\text{abs}} = 10^{-3}$ and $\delta_{\text{rel}} = 10^{-2}$.

Figure 4.6 depicts single trajectories drawn using each method. In their own right these are not particularly useful for comparison, although do serve to demonstrate at a glance that sensible results are being achieved in all cases. It is not expected that these trajectories are identical, as each method relies on a pseudorandom number sequence of different length, despite the use of a common seed.

More telling is a comparison of the properties of the trajectories produced. Figure 4.7 plots the autocorrelation function approximated for each method using a large

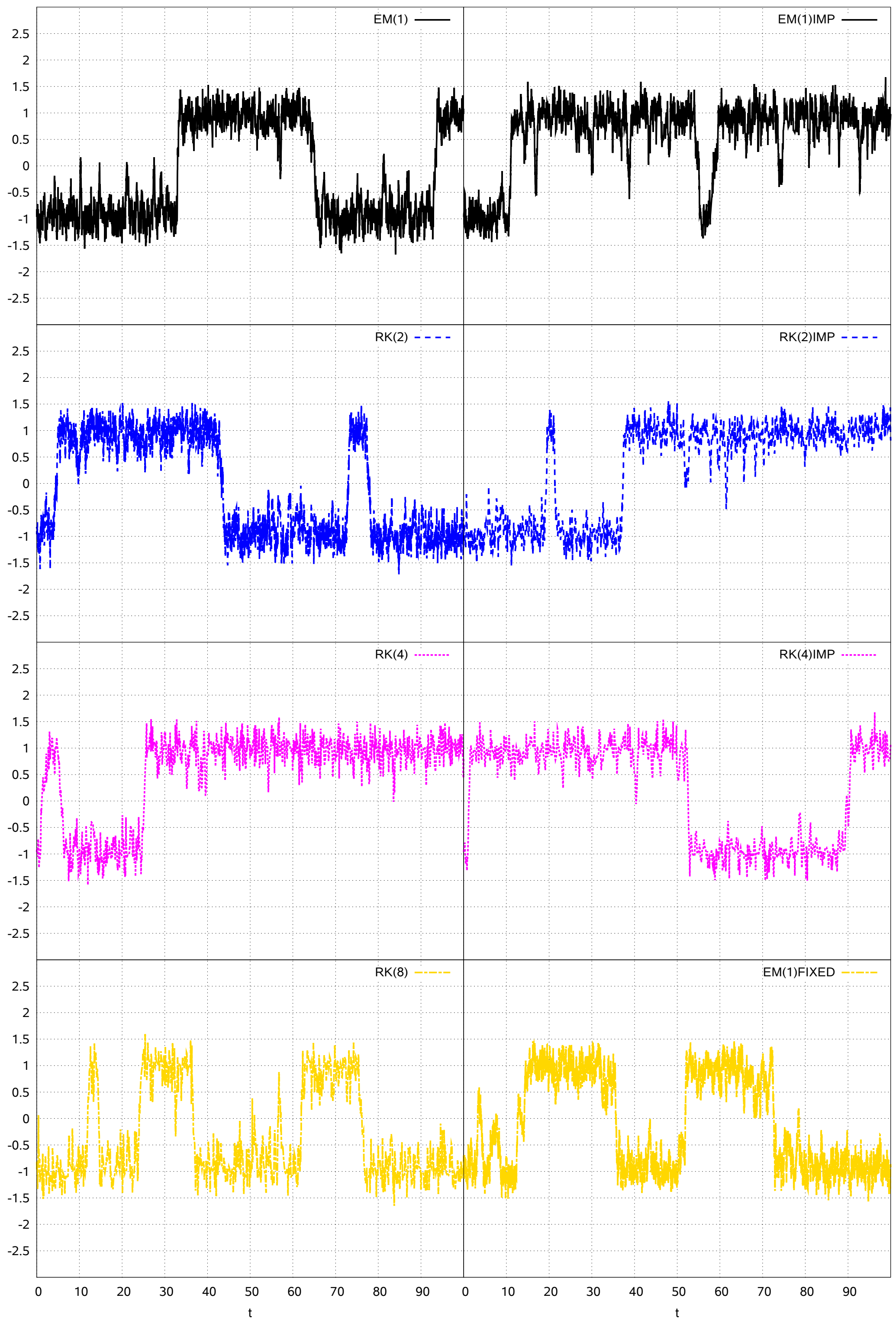


Figure 4.6: Single sample drawn from the double well model using each numerical scheme. Each case uses $\sigma_x = 1$.

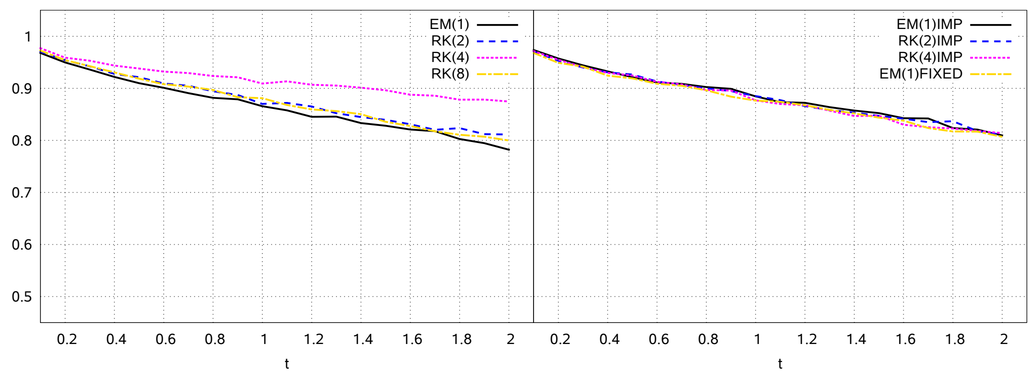


Figure 4.7: Autocorrelation of the double well model for each numerical scheme, approximated at discrete time lags along the x-axis. Each case uses $\sigma_x = 1$ with 10^4 samples.

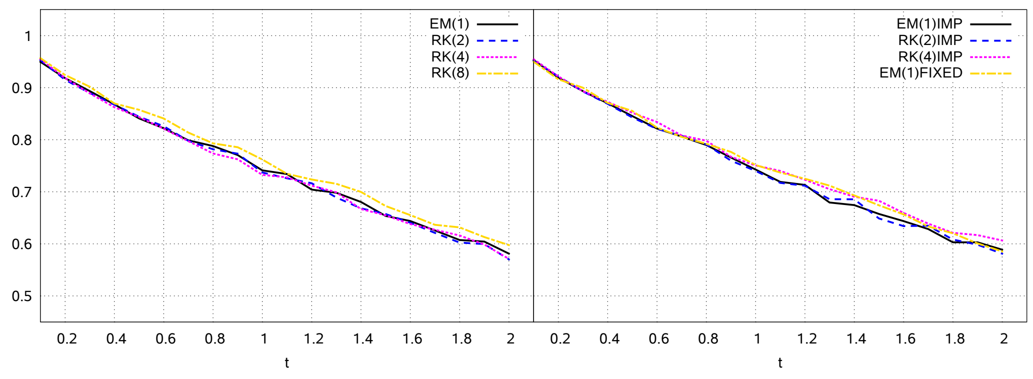


Figure 4.8: Adjusted autocorrelation of the double well model for each numerical scheme, approximated at discrete time lags along the x-axis. Each scheme uses σ_x fixed to the appropriate value in Table 4.3 with 10^4 samples.

number of samples. The explicit and implicit methods are separated for clarity in presentation only. We might expect that the autocorrelation function is the same for all methods, and higher for shorter lag times where the system is more likely constrained to a single well. We can see that the former is not the case. The RK(4) method produces greater autocorrelation than the other methods for all lag times. We may interpret this as the diffusion parameter σ_x more weakly permeating the system in this method. This is confirmed in Figure 4.9, where RK(4), and indeed RK(8), output more peaked equilibrium distributions, more evidence of a comparitively weaker σ_x.

The slight difference in behaviours induced by these methods is not necessarily a problem in its own right, and besides, there is no ground truth upon which to base disqualification. We discuss this further at the end of the section. At any rate, arguably the chosen numerical scheme merely provides a discretisation of the continuous-time dynamics, and is absorbed amongst the various other assumptions and simplifications intrinsic to the model. We argue that the behaviour of the model, as implemented, is more important than its particular formulation. For a runtime comparison of numerical schemes, however, the difference introduces a bias that favours some methods over others. Schemes where σ_x has less influence benefit from less erratic trajectories, in essence performing easier integrations than other schemes.

We correct for this by adjusting σ_x independently to match autocorrelation across methods. The approach is simple but effective – for each scheme we target a lag 1 autocorrelation of .75 and perform a binary search over σ_x values to achieve this within some error bound.

The second column of Table 4.3 provides the results of this search. As expected from the preceding discussion, a larger σ_x is required for RK(4) to produce a similar autocorrelation to the other methods. Figure 4.8 provides the autocorrelation for each method when using their adjusted σ_x values. Clearly the situation has improved, with RK(4) drawn into the other methods. Equilibrium distribution plots in Figure 4.10 are less satisfying, although do show an improvement over the universally set σ_x case. At any rate, we are content enough to proceed with a performance comparison.

For each scheme, we simulate 5000 trajectories, starting with a conservative step size of 10^{-4}, allowing each 500 steps burn-in to reach a more regular size, then taking 50 steps and recording the step size accepted by the error control algorithm. Figure 4.11 plots a histogram over the step sizes accepted by each scheme. The fourth column of Table 4.3 provides the total time progressed by each method during this run[2].

The higher-order RK(4) and RK(8) schemes clearly trump the others in terms of step size and consequently total time progression. This is not an end to the story, however, as the higher order schemes have computationally more expensive steps.

[2]The are under the histogram in Figure 4.11.

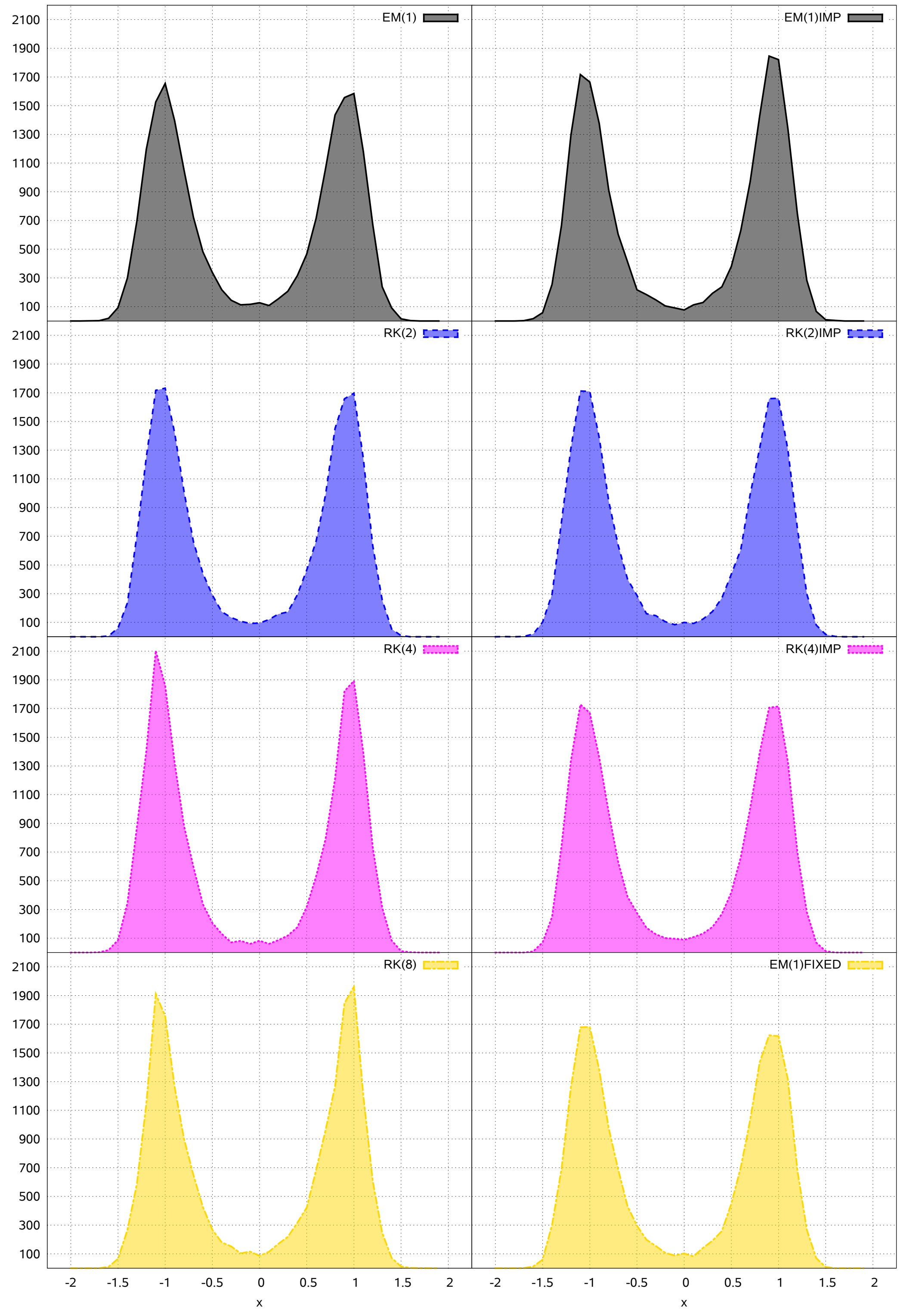


Figure 4.9: Equilibrium distribution histogram of the double well system for each numerical scheme. Each case uses $\sigma_x = 1$ with 2×10^4 samples, and burn in time of 100 units with samples taken every 5 units thereafter.

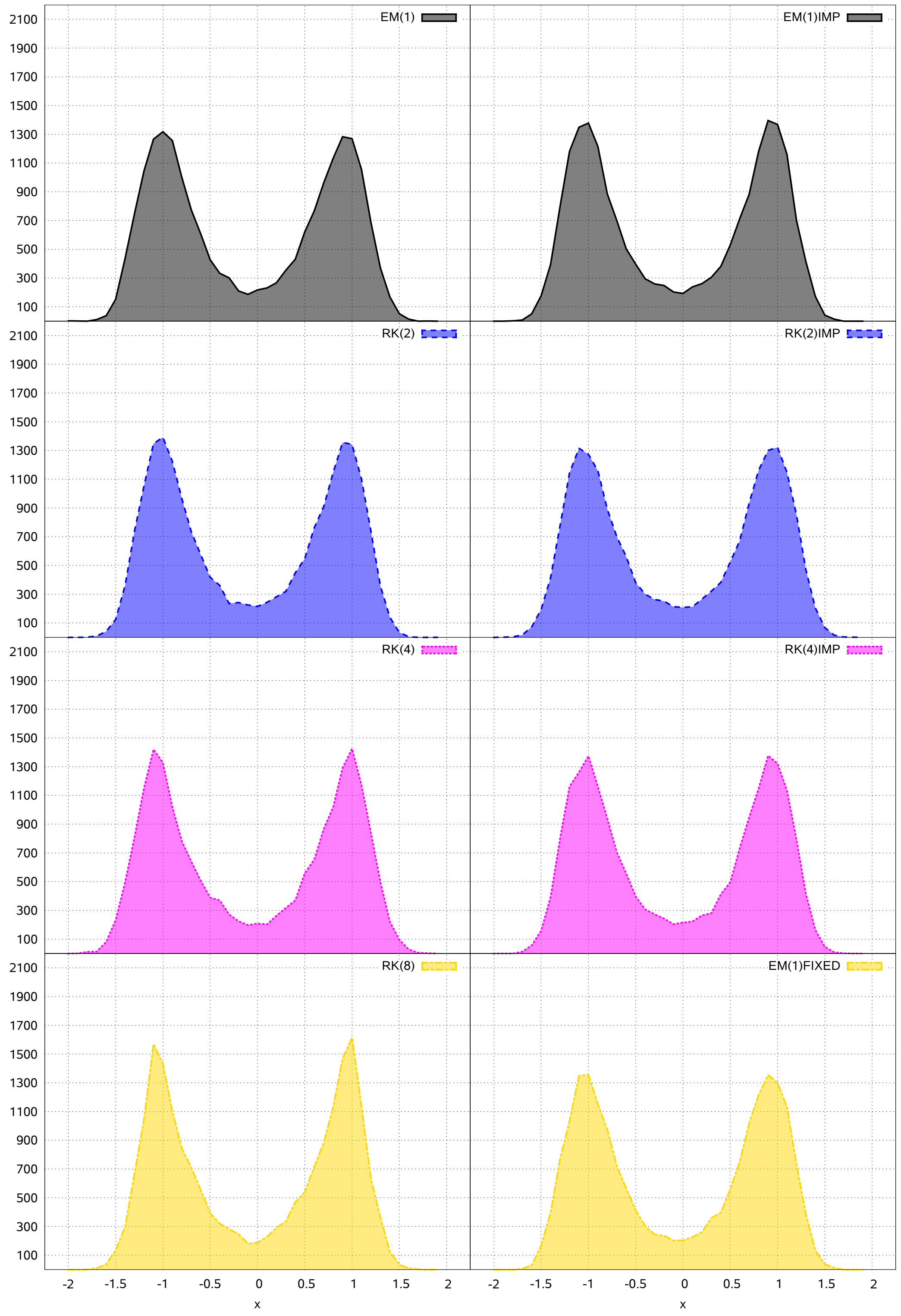


Figure 4.10: Adjusted equilibrium distribution histogram of the double well system for each numerical scheme. Each scheme uses σ_x fixed to the appropriate value in Table 4.3 with 2×10^4 samples, and burn in time of 100 units with samples taken every 5 units thereafter.

Scheme	Adjusted σ_x	Runtime	Progression	$\frac{\text{Progression}}{\text{Runtime}}$	$\frac{\text{Progression}}{\text{Steps}}$
EM(1)	.986	.67	8721	13017	.0174
RK(2)	1.011	.53	2563	4836	.0051
RK(4)	1.117	.78	**33229**	**42601**	**.0665**
RK(8)	.986	1.51	31837	21085	.0637
EM(1)IMP	1.030	.55	14303	26005	.0286
RK(2)IMP	1.027	.55	14519	26398	.0290
RK(4)IMP	1.008	.77	14161	18391	.0283
EM(1)FIXED	1.008	**.25**	5000	20000	.01

Table 4.3: Comparison of numerical schemes, all run for 5×10^5 steps. The second column gives σ_x values fixed to give an approximate 1s lag autocorrelation of .75, as detailed in the text. The third column gives real time taken for the method to complete its 5×10^5 steps, and the fourth column the number of seconds in model time progressed by the integrator in this period. The remaining columns are calculations based on these figures.

Total runtime must be considered, and this is given in the third column of Table 4.3. Given that the total number of steps has been fixed, the lower order methods are expected to perform more favourably in terms of raw runtime here. The final selection is based on the total model time progression per runtime second, given in the fifth column of Table 4.3.

On the basis of this criterion, the RK(4) method is a clear winner, and we choose this for our remaining experiments. Without presenting results, we note that the same conclusion is drawn without the σ_x adjustment, or indeed, by drawing a random σ_x from the prior for each trajectory rather than fixing this parameter. We can consider the average step size of this method, as in the sixth column of Table 4.3, as a guide for the suggested step size with which to initialise the method at the start of each particle propagation.

We acknowledge that the selection of RK(4) on runtime performance grounds is not without caveats, particularly in light of it being the most maverick in terms of both autocorrelation and equilibrium, as in Figures 4.7 and 4.9, and precisely that which prompted the simple σ_x adjustments for comparison in the first place. Potentially, the method is arrogantly underestimating its own error and boldly

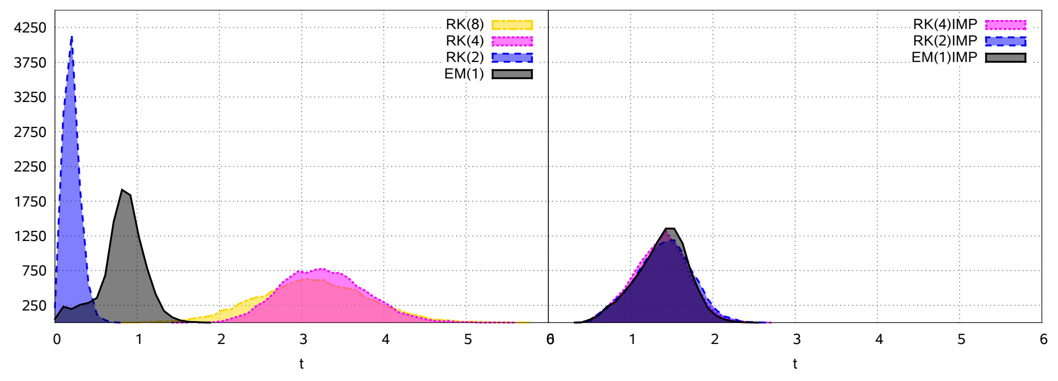


Figure 4.11: Time progression histogram for each numerical scheme across 5000 runs, with 50 steps per run, and 500 steps burn-in to eliminate the effects of an initial time step suggestion of 10^{-4}.

taking step sizes larger than justified. We do note that in the implementation of the step size adjustment used here, as in (3.38), RK(4) sets p to its higher order embedded scheme (5th order) whereas all other methods set this to their lower order. Such subtle differences in heuristics may improve its performance significantly, at least in this particular case. On the other hand, we would expect such a higher-order scheme to obtain more accurate estimates of error, which may instead call into question the lower-order methods. Ultimately, the lack of an analytical ground truth makes this difficult to assess, and we leave the matter as an interesting open question. At the very least it is worth reiterating that the scheme successfully captures the qualitative behaviours of most interest in this model, and for that reason we can be satisfied with its results.

4.4.2.2 Data

Fixing $\sigma_x = .8$, the system is simulated to provide artificial data with known ground truth for testing various filtering and smoothing methods. The sequence of measurement times begins at $t_0 = 0$, followed by $t_n = t_{n-1} + \Delta t_n$ with $\ln \Delta t_n \sim \mathcal{N}(.5, .5)$, so that measurements arrive at an irregular rate, up to $n = T = 500$. A trajectory is then simulated from the system, stopping at each of these times, with a draw from the measurement function (4.12) used to obtain a data point from the state. Stopping at specified times is straightforward – simply shorten any proposed step size that would exceed the specified time.

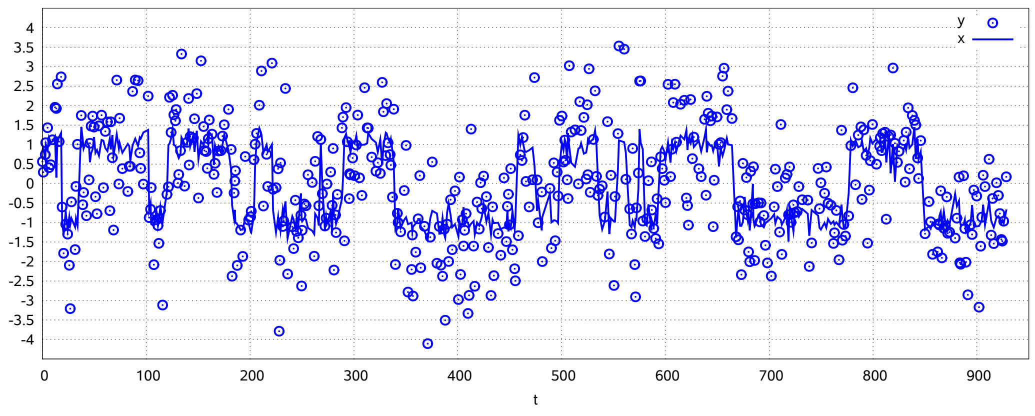


Figure 4.12: Simulated data set for the double well model, y, and true underlying trajectory, x.

Figure 4.12 plots the data set along with the true underlying trajectory of x.

4.4.2.3 Filtering

We now consider the filtering problem applied to the artificial data set. We investigate four different filtering techniques for this problem:

Bootstrap the bootstrap particle filter using deterministic stratified resampling.

Auxiliary the auxiliary particle filter using deterministic stratified resampling.

Bootstrap + regularised the bootstrap method above followed by the addition of kernel noise.

Auxiliary + regularised the auxiliary method above followed by the addition of kernel noise.

In the latter two cases, a Gaussian kernel with bandwidth $h = \frac{1}{2}h_{\text{opt}}$ is used, so that at time t_n, after stratified resampling, each particle is perturbed with:

$$\mathbf{s}_n^{(i)} \leftarrow \mathbf{s}_n^{(i)} + L_n(h\xi_n^{(i)}\mathbf{e}_n^{(i)}) \,, \tag{4.13}$$

where $\xi_n^{(i)} \sim \mathcal{N}(0,1)$, $\mathbf{e}^{(i)}$ is a unit vector drawn uniformly from the unit sphere in $\mathbb{R}^N$ and L_n is the Cholesky decomposition of the sample covariance matrix at time t_n. This is equivalent to a draw from a kernel density. It is efficient

Method	RK(4)	EM(1)FIXED
Bootstrap filter	7.05	268.55
Auxiliary filter	6.29	457.46
Bootstrap + regularised filter	6.13	439.98
Auxiliary + regularised filter	6.44	400.85
Forward-backward smoother	na	5891.77

Table 4.4: Runtime performance results for smoother and numerical scheme combinations on the double well model. All runs are performed in parallel across 4 processes on a single quad-core processor. Times are given in wallclock seconds. The forward-backward smoother is performed over results of the auxiliary filter, with runtime of the auxiliary filter excluded.

computationally, but also in terms of sample variance by exploiting stratified resampling.

Given the results of §4.4.2.1, the RK(4) scheme is chosen for numerical integration in all cases, with an initial step size of .067 suggested from Table 4.3, to be subsequently adapted by error control. For the purposes of the filtering problem, we fix $\sigma_x = .8$, its true value, and $P = 500$. Resampling is performed whenever $P_{\text{ess}} < P/2$. The task is performed across 4 processes on a single quad-core processor. Details of such parallel implementation are given in §6.

Figure 4.13 plots the filtered state estimates for each filter along with the effective sample size (ESS) at each time point.

All methods appear to capture the underlying ground truth of Figure 4.12 effectively, without significant degeneracy evident from ESS traces. There is no significant qualitative or quantitative reason to prefer one method over the others in these results.

Table 4.4 gives runtime results for each method over a single run. There is little to differentiate methods here, although we opt for the auxiliary particle filter on the basis of its slightly faster runtime than the bootstrap, and having no need to consider kernel bandwidth configuration.

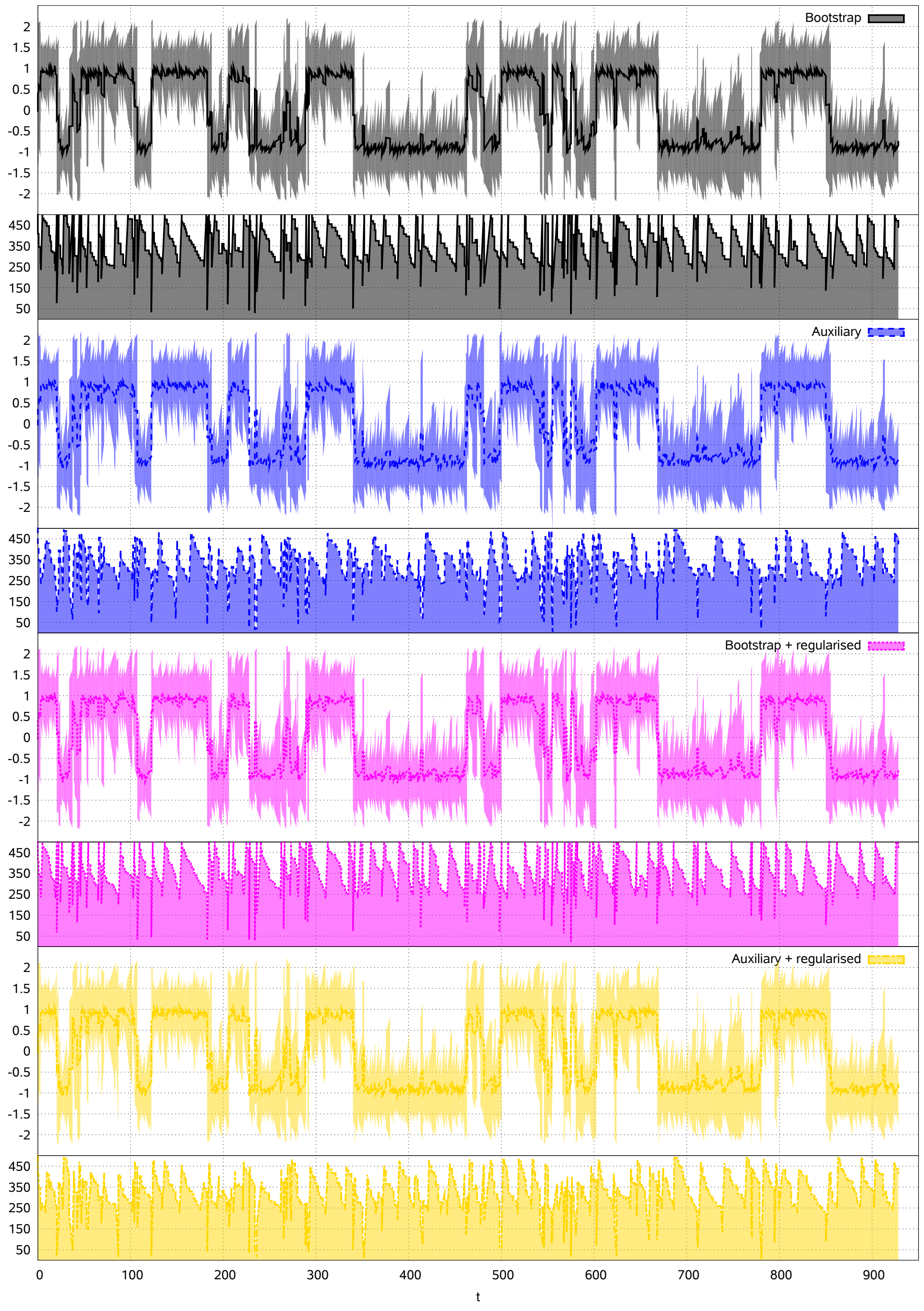


Figure 4.13: Filter results for the double well model using the RK(4) numerical integration scheme and various filtering methods, as indicated. Line indicates mean and shaded region two standard deviations. Corresponding ESS across time is given for each method as an indication of degeneracy avoidance and resampling regularity.

4.4.2.4 Smoothing

We now consider the smoothing problem. As already discussed, both the forward-backward and two-filter smoothers require a tractable transition density. This can only be achieved using a numerical scheme for the SDE with a linear step. Furthermore, the time discretisation for the filter and smoother must match that of the numerical scheme.

We therefore introduce the EM(1)FIXED scheme, an Euler-Maruyama scheme, as EM(1), with constant time step. No error control can be used in such a scheme, but in order to restrict error to something reasonable, we target the lower tail of the distribution over progressions for EM(1) in Figure 4.11 and mandate a step size of .01. A smaller step will need to be taken immediately preceding each of the times $t_1, \dots, t_T$ so that the filter may stop to incorporate measurements.

Figure 4.14 provides the results of the filter in the same manner as Figure 4.13 for the RK(4) scheme. Note that while $T = 500$, and only 500 steps of the filter are required using the RK(4) scheme, $t_T/.01 \approx 10^5$ are required using EM(1)FIXED, precisely the same number as used to integrate each of the particle trajectories. Clearly, this is significantly slower than using RK(4).

The results may now be smoothed. Results for the forward-backward smoother over the results of the auxiliary filter are given in Figure 4.15. Performance results are given in Table 4.4. The smoothing is completed successfully and represents a significant improvement over the filter result in terms of smoothness and variance.

4.4.2.5 Parameter estimation

We may also consider estimation of the parameter σ_x. Each of the filter methods may be applied in turn with this parameter added into the state in a Bayesian fashion.

The Root Mean Square Error (RMSE) of the parameter estimation as the filters progress through time is given in Figure 4.16. Given the finite length of the observation sequence and relatively few number of transitions between stable states in this time, it is not surprising that the estimation is imperfect and the RMSE does not quite approach zero. The fluctuations in the regularised methods are

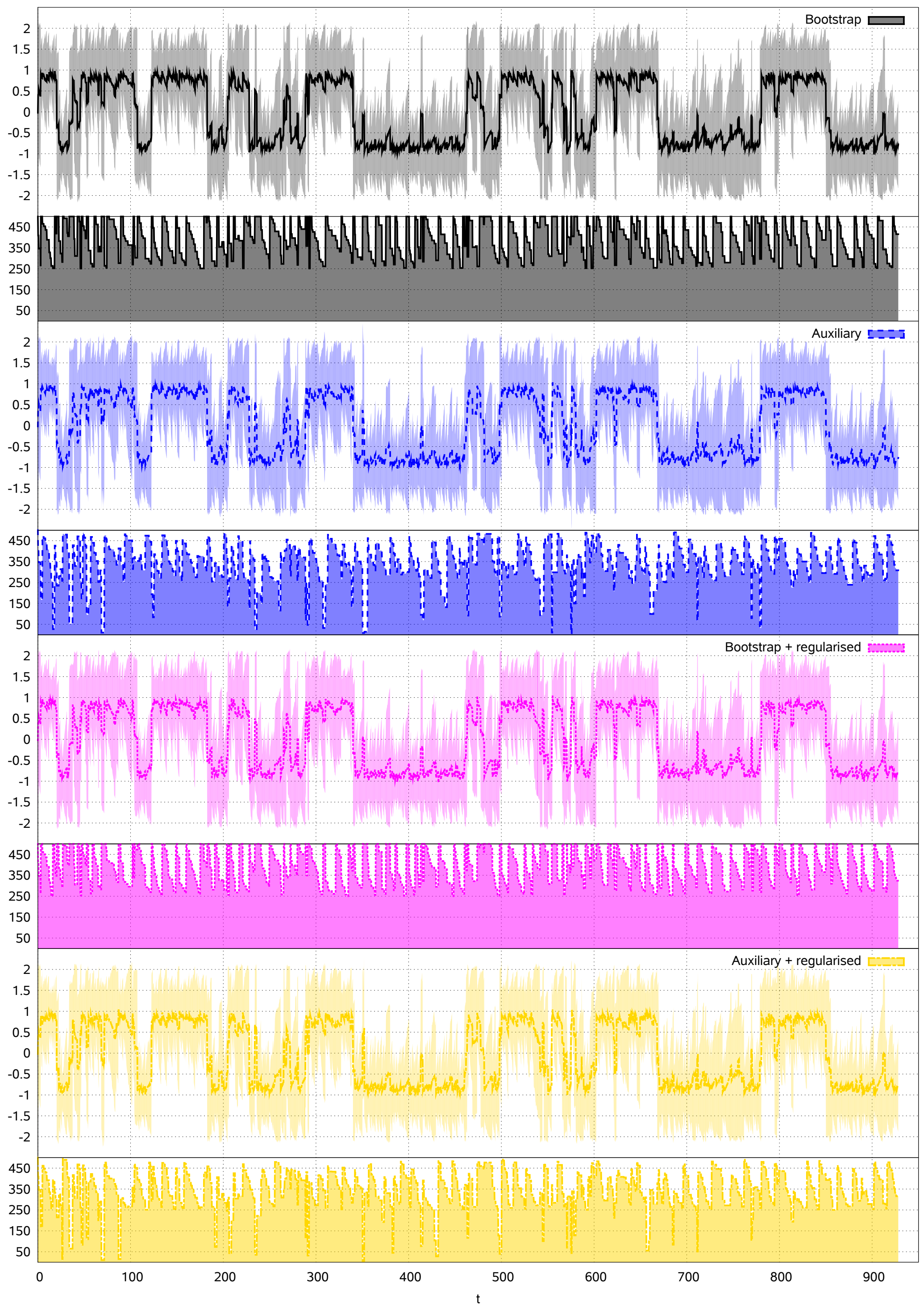


Figure 4.14: Filter results for the double well model for various methods, as indicated, using a fixed time step Euler-Maruyama numerical integrator of step size $.01$. Line indicates mean and shaded region two standard deviations. Corresponding ESS across time is given for each method as an indication of degeneracy avoidance and resampling regularity.

caused by the generation of new parameter estimations via the implied artificial dynamic of kernel density approximations. The flatlining of the bootstrap and auxiliary filter result may be due to a lack of sufficient parameter sample diversity to progress the estimate any closer to its true value.

4.5 Summary

This chapter has considered the application of conventional Bayesian filtering and smoothing methods, developed for discrete time, to continuous-time dynamical systems (§4.1). In attempting to fuse the two it has identified several difficulties, most particularly (§4.2):

- the unavailability of a closed form transition density $p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})$ in the general case,
- the expensive computational cost of particle propagations, and
- the divergence of reverse dynamics.

Given the first of these difficulties, only those particle filters using *resampling proposals* are applicable to the continuous time setting (§4.3). These include the bootstrap, auxiliary and regularised particle filters. Conventional smoothers are only applicable when a linear discretisation is used, and in this case their rate is tied to that of this discretisation.

Applicable conventional methods have been demonstrated experimentally on two artificial problems (§4.4), with satisfactory filtering and parameter estimation results, but satisfactory smoothing results only with significant computational burden. These experimental results also demonstrate the significant performance benefits to be gained from our use of higher-order Runge-Kutta methods over the common use of low-order methods such as Euler-Maruyama (§4.4).

To these issues we may add the additional problem identified in §2.3:

- degeneracy in the filter-smoother, two-filter smoother and forward-backward smoother due to reweighting of particles obtained by a filter.

Together, these challenges constitute the major motivation for this work and firmly establish its context.

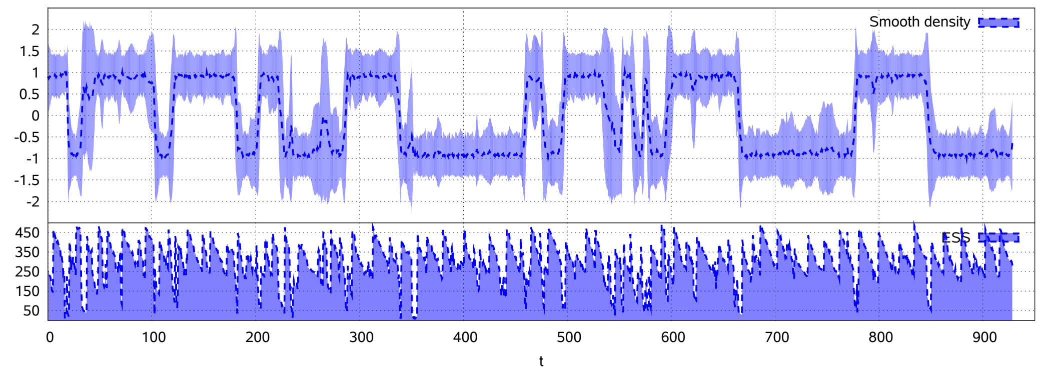


Figure 4.15: Smoothed results and ESS across time for the double well model, using a forward-backward smoother over the results of an auxiliary particle filter.

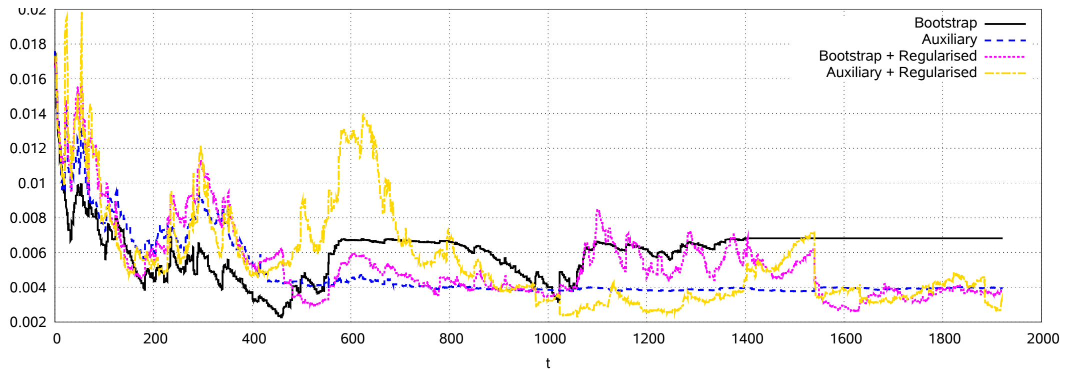


Figure 4.16: RMSE of σ_x Bayesian parameter estimation on the double well model for various filtering methods.

Chapter 5

Kernel Smoothers

Having highlighted the problems in combining conventional Bayesian methods for time-series with continuous-time dynamical systems, we now alleviate them with two novel solutions to the smoothing problem, the *kernel forward-backward* and *kernel two-filter* smoothers. These methods exploit kernel density estimates and resampling proposals to establish importance sampling schemes that cancel the transition density. Consequently, unlike the standard forward-backward and two-filter smoothers, they are applicable to the continuous time setting and permit use of higher-order numerical integration schemes for the SDEs of the dynamical system. In doing so they deliver substantial runtime performance gains over conventional techniques. Additionally, they facilitate the drawing of new samples to support the smooth density, rather than being limited to those drawn during filtering. In this way, the degeneracy problem, even for the discrete time case, begins to be addressed.

This chapter introduces the general derivation and formulation of these two methods in §5.1-5.2, as well as discussing some issues related to parameter estimation under them in §5.3. Some theoretical results are given in §5.4 to relate the methods to existing work, while experimental results in §5.5 demonstrate their benefits over conventional methods, particularly with regard to significant runtime performance gains.

5.1 Kernel forward-backward smoother

The kernel forward-backward smoother follows a similar derivation to that of the forward-backward smoother (§2.3.3), with the introduction of kernel densities to permit arbitrary proposal distributions for importance sampling.

Factorise the smooth density as follows:

$$\begin{aligned} p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T}) &= \frac{p(\mathbf{y}_{n+1:T} \,|\, \mathbf{x}_n, \mathbf{y}_{1:n}) p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})}{p(\mathbf{y}_{n+1:T} \,|\, \mathbf{y}_{1:n})} && (5.1) \\ &\propto p(\mathbf{y}_{n+1:T} \,|\, \mathbf{x}_n) p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) && (5.2) \\ &\propto p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) \int p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n) p(\mathbf{y}_{n+1:T} \,|\, \mathbf{x}_{n+1}) \, d\mathbf{x}_{n+1} && (5.3) \\ &\propto p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) \int p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n) \frac{p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:T})}{p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:n})} \, d\mathbf{x}_{n+1} \,. && (5.4) \end{aligned}$$

To eliminate the integral and simplify the derivation, consider the joint:

$$p(\mathbf{x}_{n:n+1} \,|\, \mathbf{y}_{1:T}) = \frac{p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}) p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n) p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:T})}{p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:n})} \,. \tag{5.5}$$

Now consider importance sampling from this with a proposal distribution $q'(\mathbf{x}_{n:n+1})$ of the form:

$$q'(\mathbf{x}_{n:n+1}) = p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n) q(\mathbf{x}_n) \,, \tag{5.6}$$

so as to cancel the intractable transition density in (5.5)[1]. Drawing $(\mathbf{s}_n^{(i)}, \mathbf{s}'^{(i)}_{n+1}) \sim q'(\mathbf{x}_{n:n+1})$, the weight calculation is reduced to:

$$\psi_n^{(i)} = \frac{p(\mathbf{x}_n = \mathbf{s}_n^{(i)} \,|\, \mathbf{y}_{1:n}) p(\mathbf{x}_{n+1} = \mathbf{s}'^{(i)}_{n+1} \,|\, \mathbf{y}_{1:T})}{p(\mathbf{x}_{n+1} = \mathbf{s}'^{(i)}_{n+1} \,|\, \mathbf{y}_{1:n}) q(\mathbf{x}_n = \mathbf{s}_n^{(i)})} \,, \tag{5.7}$$

and the weighted sample set $\{(\mathbf{s}_n^{(i)}, \psi_n^{(i)})\}$ represents the smooth density at time t_n.

The filter densities $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$ and $p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:n})$ may be obtained through a preceding filter, and the smooth density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T})$ is known recursively. Depending on the selection of $q(\mathbf{x}_n)$, some of these will need to be approximated. Several techniques could be used for this, such as Gaussian mixtures or variational fits. We choose to use kernel density estimates for two reasons. Firstly, being non-parametric, they are generally applicable to all models without knowledge of internals. Secondly, they provide some nice opportunities for parallelism and optimisation, which we discuss in the implementation of these methods in §6.

[1]Note parallels with the resampling proposals for particle filters described in §4.3.

Note that as all of the approximated densities are conditioned only on measurements, they are the same for all samples $\{\mathbf{s}_{n:n+1}^{(i)}\}$. Conversely, the transition density, a nuisance because of its conditioning on $\mathbf{s}_n^{(i)}$, has been eliminated.

The algorithm is summarised as:

Algorithm 5.1 (Kernel forward-backward smoother) *Perform a filter forward in time, at the conclusion of which $p(\mathbf{x}_T \,|\, \mathbf{y}_{1:T})$ is known and approximated by $\{(\mathbf{s}_T^{(i)}, \psi_T^{(i)} = \pi_T^{(i)})\}$. Then, for time t_n, draw $\mathbf{s}_n^{(i)}$ from some importance distribution $q(\mathbf{x}_n)$, draw $\mathbf{s}'^{(i)}_{n+1} \sim p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)})$ via numerical integration, and let:*

$$\psi_n^{(i)} = \frac{p(\mathbf{x}_n = \mathbf{s}_n^{(i)} \,|\, \mathbf{y}_{1:n}) p(\mathbf{x}_{n+1} = \mathbf{s}'^{(i)}_{n+1} \,|\, \mathbf{y}_{1:T})}{p(\mathbf{x}_{n+1} = \mathbf{s}'^{(i)}_{n+1} \,|\, \mathbf{y}_{1:n}) q(\mathbf{x}_n = \mathbf{s}_n^{(i)})}, \tag{5.8}$$

The weighted sample set $\{(\mathbf{s}_n^{(i)}, \psi_n^{(i)})\}$ then approximates the smooth density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T})$.

The interest now is in selecting an appropriate proposal distribution $q(\mathbf{x}_n)$. While theoretically any density where $q(\mathbf{x}_n) > 0$ whenever $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T}) > 0$, for all $\mathbf{x}_n$, is tenable, in practice some facilitate more efficient sampling than others. We discuss several options in the following sections.

5.1.1 Filter as proposal

One option is to set $q(\mathbf{x}_n) = p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$. Recall that $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$ is approximated by the weighted sample set $\{(\mathbf{s}_n^{(i)}, \pi_n^{(i)})\}$. By preserving these samples and propagating each through the SDEs of the system to obtain $\{\mathbf{s}'^{(i)}_{n+1}\}$, the smoothed weight reduces to:

$$\psi_n^{(i)} = \frac{p_{\mathcal{K}}(\mathbf{x}_{n+1} = \mathbf{s}'^{(i)}_{n+1} \,|\, \mathbf{y}_{1:T})}{p_{\mathcal{K}}(\mathbf{x}_{n+1} = \mathbf{s}'^{(i)}_{n+1} \,|\, \mathbf{y}_{1:n})} \pi_n^{(i)} \,. \tag{5.9}$$

The weighted set $\{(\mathbf{s}_n^{(i)}, \psi_n^{(i)})\}$ then approximates the smoothed density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$.

Kernel densities provide approximations at arbitrary points of the two densities of (5.9). $p_{\mathcal{K}}(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:n})$ is the kernel estimate of the filter density calculated during the forward pass. $p_{\mathcal{K}}(\mathbf{x}_n \,|\, \mathbf{y}_{1:T})$ is known when the forward pass terminates at time $n = T$, and recursively calculated as the smooth density as the backward pass proceeds.

Clearly, this will suffer from the same degeneracy issue of the forward-backward smoother if the filter density fails to adequately support the smooth density. Nevertheless, it is suitable in many situations and worth considering for its potential efficiency, given that one density evaluation is cancelled.

The algorithm requires two kernel density evaluations at each time step, and these dominate the runtime complexity. If these are performed using the dual-tree algorithm [44], runtime complexity is $\mathcal{O}(T \lg P \lg P)$ in the best case, and $\mathcal{O}(TP^2)$ in the worst. Details of such an implementation are given in §6.

5.1.2 Equilibrium as proposal

For a stationary process, we can consider using the equilibrium distribution as a static proposal, setting $q(\mathbf{x}_n) = p_s(\mathbf{x})$ for all times. This is a particularly attractive option, as it allows sensible generation of new particles during the backwards pass, potentially avoiding the degeneracy problem inherent in the reweighting of filter particles.

At the beginning of the pass, $\{\mathbf{s}_s^{(i)}\}$ are independently sampled from $p_s(\mathbf{x})$, with corresponding densities $\beta_s^{(i)} = p_s(\mathbf{s}_s^{(i)})$. At each time t_n, propagate $\mathbf{s}_s^{(i)}$ through the SDEs of the system to obtain $\mathbf{s'}_{n+1}^{(i)}$. The smoothed weight becomes:

$$\psi_n^{(i)} = \frac{p_{\mathcal{K}}(\mathbf{x}_n = \mathbf{s}_s^{(i)} \,|\, \mathbf{y}_{1:n}) p_{\mathcal{K}}(\mathbf{x}_{n+1} = \mathbf{s'}_{n+1}^{(i)} \,|\, \mathbf{y}_{1:T})}{p_{\mathcal{K}}(\mathbf{x}_{n+1} = \mathbf{s'}_{n+1}^{(i)} \,|\, \mathbf{y}_{1:n}) \beta_s^{(i)}}, \tag{5.10}$$

so that the weighted set $\{(\mathbf{s}_s^{(i)}, \psi_n^{(i)})\}$ approximates the smoothed density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$. Kernel density estimates are again used to circumvent intractability.

The basic algorithm now requires three kernel density evaluations, potentially a fourth if the equilibrium distribution itself is approximated in this way. The runtime complexity is therefore $\mathcal{O}(T \lg P \lg P)$ in the best case, and $\mathcal{O}(TP^2)$ in the worst. A number of optimisations can significantly reduce this, detailed in the next chapter (§6).

5.1.3 Other proposals

As with proposal distributions for particle filters, an endless variety of possibilities can spring to mind, many exploiting peculiar functional properties, or specialising to particular problem types.

A few speculative ideas are worth mentioning without elaborating too much on the details. One option would be to approximate the next smooth density using the current smooth density. This may be effective if the time difference is sufficiently small that they are likely to be similar. If too dissimilar, a linearisation of the reverse dynamics applied to the current smooth density may provide a reasonable approximation. Perhaps more rigorous would be use of the system's autocorrelation function over the time interval[2]. For some models, an easy, but possibly inefficient option, is to simply use the prior $p(\mathbf{x}_0)$.

Selection of a proposal distribution is also precisely where domain knowledge and model-specific structures may be best exploited for the greatest computational or statistical gain. *Resample-move* [28][ch.6] transition kernels could be used to support a mutually exclusive set of different behaviours. The Jacobian could be used to predict expansion or contraction, perhaps even combined with dimensionality reduction for large-scale problems with many dimensions (e.g. [45]). There is no reason why adaptive proposal strategies could not be employed either.

5.2 Kernel two-filter smoother

Like its counterpart above, the kernel two-filter smoother may be compared to the two-filter smoother (2.3.2), having a similar derivation, and exploiting kernel densities for applicability to continuous-time dynamical systems. In contrast to the standard two-filter smoother, however, the approach never involves an explicit calculation of the backward filter density $p(\mathbf{x}_n \,|\, \mathbf{y}_{n:T})$, and does not require the prior $p(\mathbf{x}_n)$ or its substitute $\gamma_n(\mathbf{x}_n)$.

[2]This would be similar to the discretisation scheme for SDEs proposed by [8].

Factorise the smooth density as follows:

$$p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T}) = \frac{p(\mathbf{y}_{n:T} \,|\, \mathbf{x}_n, \mathbf{y}_{1:n-1}) p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n-1})}{p(\mathbf{y}_{n:T} \,|\, \mathbf{y}_{1:n-1})} \tag{5.11}$$

$$\propto p(\mathbf{y}_{n:T} \,|\, \mathbf{x}_n) p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n-1}) \,, \tag{5.12}$$

expanding the likelihood term:

$$p(\mathbf{y}_{n:T} \,|\, \mathbf{x}_n) = p(\mathbf{y}_n \,|\, \mathbf{x}_n) p(\mathbf{y}_{n+1:T} \,|\, \mathbf{y}_n, \mathbf{x}_n) \tag{5.13}$$

$$= p(\mathbf{y}_n \,|\, \mathbf{x}_n) p(\mathbf{y}_{n+1:T} \,|\, \mathbf{x}_n) \tag{5.14}$$

$$= p(\mathbf{y}_n \,|\, \mathbf{x}_n) \int p(\mathbf{y}_{n+1:T} \,|\, \mathbf{x}_{n+1}) p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n) \, d\mathbf{x}_{n+1} \,. \tag{5.15}$$

Now observe:

$$p(\mathbf{x}_n \,|\, \mathbf{y}_{n:T}) \propto p(\mathbf{x}_n) p(\mathbf{y}_{n:T} \,|\, \mathbf{x}_n) \tag{5.16}$$

$$\propto p(\mathbf{x}_n) p(\mathbf{y}_n \,|\, \mathbf{x}_n) \int p(\mathbf{y}_{n+1:T} \,|\, \mathbf{x}_{n+1}) p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n) \, d\mathbf{x}_{n+1} \,, \tag{5.17}$$

and consider the joint:

$$p(\mathbf{x}_{n:n+1} \,|\, \mathbf{y}_{n:T}) \propto p(\mathbf{x}_n) p(\mathbf{y}_n \,|\, \mathbf{x}_n) p(\mathbf{y}_{n+1:T} \,|\, \mathbf{x}_{n+1}) p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n) \,. \tag{5.18}$$

Now consider importance sampling from this using a proposal distribution $q'(\mathbf{x}_{n:n+1})$ of the form:

$$q'(\mathbf{x}_{n:n+1}) = p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n) q(\mathbf{x}_n) \,, \tag{5.19}$$

so as to cancel the intractable transition density in (5.18). Drawing $(\mathbf{s}_n^{(i)}, \mathbf{s}'^{(i)}_{n+1}) \sim q'(\mathbf{x}_{n:n+1})$, the weight calculation for the backward filter is:

$$\tilde{\pi}_n^{(i)} = \frac{p(\mathbf{x}_n) p(\mathbf{y}_n \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)}) p_{\mathcal{K}}(\mathbf{y}_{n+1:T} \,|\, \mathbf{x}_{n+1} = \mathbf{s}'^{(i)}_{n+1})}{q(\mathbf{x}_n = \mathbf{s}_n^{(i)})} \,, \tag{5.20}$$

so that the weighted sample set $\{(\mathbf{s}_n^{(i)}, \tilde{\pi}_n^{(i)})\}$ would represent the backward filter density, if not for the expected unavailability of the prior $p(\mathbf{x}_n)$. Consider the weight calculation for the backward likelihood:

$$\beta_n^{(i)} = \frac{\tilde{\pi}_n^{(i)}}{p(\mathbf{x}_n)} \tag{5.21}$$

$$= \frac{p(\mathbf{y}_n \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)}) p_{\mathcal{K}}(\mathbf{y}_{n+1:T} \,|\, \mathbf{x}_{n+1} = \mathbf{s}'^{(i)}_{n+1})}{q(\mathbf{x}_n = \mathbf{s}_n^{(i)})} \,, \tag{5.22}$$

noting that the prior $p(\mathbf{x}_n)$ has now been cancelled. The weighted sample set $\{(\mathbf{s}_n^{(i)}, \beta_n^{(i)})\}$ now represents the backward likelihood $p(\mathbf{y}_{n:T} \,|\, \mathbf{x}_n)$. This alone is sufficient for basing the recursion of the method; the backward filter density is not required.

Note that $p_{\mathcal{K}}(\mathbf{y}_{n+1:T} \,|\, \mathbf{x}_{n+1})$ gives a kernel *likelihood* estimate. In terms of implementation, this is, for all intents and purposes, identical to a kernel *density* estimate, although need not integrate to one over $\mathbf{x}_n$, and indeed may be infinite.

Recalling (2.29), the smooth density weights may be calculated by:

$$\psi_n^{(i)} = \beta_n^{(i)} p_{\mathcal{K}}(\mathbf{x}_n = \mathbf{s}_n^{(i)} \,|\, \mathbf{y}_{1:n-1}) \,, \tag{5.23}$$

and the weighted sample set $\{(\mathbf{s}_n^{(i)}, \psi_n^{(i)})\}$ represents the smooth density at time t_n. The algorithm is summarised below:

Algorithm 5.2 (Kernel two-filter smoother) *Perform a filter forward in time, at the conclusion of which $p(\mathbf{x}_T \,|\, \mathbf{y}_{1:T})$ is known and approximated by $\{(\mathbf{s}_T^{(i)}, \psi_T^{(i)} = \pi_T^{(i)})\}$. Let:*

$$\beta_T^{(i)} = \frac{p(\mathbf{y}_T \,|\, \mathbf{x}_T = \mathbf{s}_T^{(i)})}{p_{\mathcal{K}}(\mathbf{x}_T = \mathbf{s}_T^{(i)} \,|\, \mathbf{y}_{1:T})} \psi_T^{(i)} \,, \tag{5.24}$$

so that $p(\mathbf{y}_T \,|\, \mathbf{x}_T)$ is approximated by a kernel likelihood over $\{(\mathbf{s}_T^{(i)}, \beta_T^{(i)})\}$.

Then, for time t_n, draw $\mathbf{s}_n^{(i)}$ from some importance distribution $q(\mathbf{x}_n)$, draw $\mathbf{s'}_{n+1}^{(i)} \sim p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)})$, and let:

$$\beta_n^{(i)} = \frac{p(\mathbf{y}_n \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)}) p_{\mathcal{K}}(\mathbf{y}_{n+1:T} \,|\, \mathbf{x}_{n+1} = \mathbf{s'}_{n+1}^{(i)})}{q(\mathbf{x}_n = \mathbf{s}_n^{(i)})} \,. \tag{5.25}$$

The weighted sample set $\{(\mathbf{s}_n^{(i)}, \beta_n^{(i)})\}$ then approximates the backward likelihood $p(\mathbf{y}_{n:T} \,|\, \mathbf{x}_n)$. Now, let:

$$\psi_n^{(i)} = \beta_n^{(i)} p_{\mathcal{K}}(\mathbf{x}_n = \mathbf{s}_n^{(i)} \,|\, \mathbf{y}_{1:n-1}) \,. \tag{5.26}$$

The weighted sample set $\{(\mathbf{s}_n^{(i)}, \psi_n^{(i)})\}$ then approximates the smooth density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T})$.

Similar options for the proposal $\mathbf{s}_{n+1}^{(i)}$ may be used as for the kernel forward-backward smoother. Note, however, that use of the filter density as proposal provides no additional cancelling. No other obvious cancelling proposals are available, partly because unlike the forward-backward smoother, the smooth density is calculated as an aside – it is the likelihood calculation that is essential for the recursion. Even the uncorrected filter density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n-1})$, providing cancellation for smoothed weights, needs to be evaluated to recover likelihood weights for the next step of the recursion.

The algorithm requires two kernel density (likelihood) evaluations. This is fewer than for the kernel forward-backward smoother, although fewer optimisations are available also (see §6).

5.3 Parameter estimation

For a Bayesian approach to parameter estimation, ideally a filter should be sufficient to conclude parameter estimates, as the posterior over static parameters will have absorbed all data at the conclusion of a forward pass. Unfortunately, in practice an artificial dynamic must be applied to the parameters, either explicitly or implicitly[3], such that smoothing may be required to assess parameter estimates. One may also wish to smooth to assess the fit or likelihood of the data under the model, or to utilise an iterative parameter estimation scheme such as ML using Expectation Maximisation.

If the full posterior over parameters is to be preserved, smoothing may simply be performed as normal, continuing to treat the parameters as regular state variables. If parameters are to be fixed and the state estimate smoothed under the selected values, a number of options are available. The first is to simply fix parameter values, then filter and smooth over the remaining variables. This is particularly attractive if working with a stationary system, where fixing the parameters is akin to fixing the equilibrium distribution, which could then be used as an effective proposal.

Obviously this approach requires two filter passes, one for parameter estimation and one to complement the smoother. If for whatever reason this is untenable, one might consider reconditioning kernels for the backward pass. Take, for example, the filter density $p(\mathbf{x}_n, \boldsymbol{\theta} \,|\, \mathbf{y}_{1:n})$ over the augmented state $\mathbf{x}^*_n = [\mathbf{x}^T_n, \boldsymbol{\theta}^T_n]^T$,

[3]As in the case of the regularised particle filter.

represented by the kernel density:

$$p(\mathbf{x}^*_n \,|\, \mathbf{y}_{1:n}) \approx \frac{1}{h^N |L^*_n|^{-1}} \sum_{i=1}^{P} \pi^{(i)}_n \mathcal{K}(\frac{1}{h} \| L^{*-1}_n (\mathbf{x}^*_n - \mathbf{s}^{*(i)}_n) \|) \,, \tag{5.27}$$

where $\mathcal{K}$ is the Gaussian kernel. Now consider the density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}, \boldsymbol{\theta})$. This is obtained by conditioning each of the kernels individually on the parameters $\boldsymbol{\theta}$. Kernel i has mean $\mathbf{s}^{*(i)}_n$ and standard deviation hL^*_n, partitioned as:

$$\mathbf{s}^{*(i)}_n = \begin{pmatrix} \mathbf{s}_n \\ \boldsymbol{\theta}_n \end{pmatrix} \tag{5.28}$$

$$L^*_n = h \begin{pmatrix} L_{\mathbf{x}} & L_{\mathbf{x},\boldsymbol{\theta}} \\ L_{\boldsymbol{\theta},\mathbf{x}} & L_{\boldsymbol{\theta}} \end{pmatrix} . \tag{5.29}$$

By applying the standard conditional rules for Gaussians, a new Gaussian kernel is obtained with mean and covariance:

$$\hat{\mathbf{s}}^{(i)}_n = \mathbf{s}^{(i)}_n + \Sigma_{\mathbf{x},\boldsymbol{\theta}} \Sigma^{-1}_{\boldsymbol{\theta}} (\boldsymbol{\theta} - \boldsymbol{\theta}_n) \tag{5.30}$$

$$\hat{\Sigma}_n = \Sigma_{\mathbf{x}} - \Sigma_{\mathbf{x},\boldsymbol{\theta}} \Sigma^{-1}_{\boldsymbol{\theta}} \Sigma_{\boldsymbol{\theta},\mathbf{x}} \,, \tag{5.31}$$

and standard deviation $\hat{L}_n$ given by the Cholesky decomposition of $\hat{\Sigma}_n$. Note that this standard deviation is constant for all kernels. The conditioned kernel density is now:

$$p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n}, \boldsymbol{\theta}) \approx \frac{1}{h^N |\hat{L}_n|^{-1}} \sum_{i=1}^{P} \pi^{(i)}_n \mathcal{K}(\frac{1}{h} \| \hat{L}^{-1}_n (\mathbf{x}_n - \hat{\mathbf{s}}^{(i)}_n) \|) \,, \tag{5.32}$$

which allows the adjustment of filter densities already obtained during parameter estimation, without the execution of a second filter with fixed parameters.

While the above is specific to the Gaussian kernel, there is no reason that the same cannot be applied to other varieties of kernel function. For some, it may produce an interesting scenario where the conditioning produces kernels of a different class for the smoothing pass.

5.4 Theoretical results

We now derive a number of results regarding the kernel filters presented to tie them in with existing work. The main result of this section is that *the use of* *these kernel smoothers is essentially equivalent to approximating the transition density using a single kernel.*

We first introduce some terminology for clarity of discussion.

Definition 5.1 (Support equivalence) *Take two kernel density estimates $p_{\mathcal{K}1}(\mathbf{x})$ and $p_{\mathcal{K}2}(\mathbf{x})$, defined over weighted sample sets $\{(\mathbf{s}_1^{(i)}, \pi_1^{(i)})\}$ and $\{(\mathbf{s}_2^{(i)}, \pi_2^{(i)})\}$, respectively. Then, $p_{\mathcal{K}1}(\mathbf{x})$ and $p_{\mathcal{K}2}(\mathbf{x})$ are said to have equivalent support if $\{\mathbf{s}_1^{(i)}\} = \{\mathbf{s}_2^{(i)}\}$.*

Note that an order may be induced over the weighted sample set by ordering components according to their index i. In such cases the ordered weights may be denoted as the vector $\boldsymbol{\pi}$.

The results of a kernel density evaluation for multiple query points will be represented in matrix form. Specifically:

Definition 5.2 *For a kernel density $p_{\mathcal{K}}(\mathbf{x})$ defined over a weighted sample set $\{(\mathbf{s}^{(i)}, \pi^{(i)})\}$, and set of query points $\{\mathbf{s}'^{(j)}\}$, $A\boldsymbol{\pi}$ denotes the result of evaluating $p_{\mathcal{K}}(\mathbf{x})$ at each of the points $\{\mathbf{s}'^{(j)}\}$, where:*

$$A^{(j,i)} = \frac{1}{h^N |L_n|^{-1}} \mathcal{K}(\frac{1}{h} \| L_n^{-1} (\mathbf{s}'^{(j)} - \mathbf{s}^{(i)}) \|) \,. \tag{5.33}$$

5.4.1 Forward-backward smoothing

We now claim:

Proposition 5.1 *The kernel forward-backward smoother is equivalent to the forward-backward smoother when:*

- *the filter density is used as the proposal distribution,*
- *the particles supporting this are reused,*
- *the same standardisation matrix L_n is used for both $p_{\mathcal{K}}(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:T})$ and $p_{\mathcal{K}}(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:n})$, or no standardisation is performed, and*
- *the transition density is approximated by the single point kernel estimator:*

$$p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n = \mathbf{s}_n^{(j)}) \approx \frac{1}{h^N |L_n|^{-1}} \mathcal{K}(\frac{1}{h} \| L_n^{-1} (\mathbf{x}_{n+1} - \mathbf{s}_{n+1}^{(j)}) \|) \,. \tag{5.34}$$

We begin by reframing Algorithm 2.7 in the matrix form introduced above:

Algorithm 5.3 (Forward-backward smoother, matrix form) *Perform a filter forward in time, at the conclusion of which $p(\mathbf{x}_T \,|\, \mathbf{y}_{1:T})$ is known and approximated by $\{(\mathbf{s}_T^{(i)}, \psi_T^{(i)} = \pi_T^{(i)})\}$. Initialise with $\psi_T^{(i)} = \pi_T^{(i)}$ and proceed recursively as follows:*

$$A_n^{(i,j)} \;=\; p(\mathbf{x}_{n+1} = \mathbf{s}_{n+1}^{(i)} \,|\, \mathbf{x}_n = \mathbf{s}_n^{(j)})\,, \tag{5.35}$$

then:

$$\boldsymbol{\gamma}_n \;=\; A_n \boldsymbol{\pi}_n \tag{5.36}$$

$$\boldsymbol{\delta}_n \;=\; A_n^T(\boldsymbol{\psi}_{n+1} \oslash \boldsymbol{\gamma}_n) \tag{5.37}$$

$$\boldsymbol{\psi}_n \;=\; \boldsymbol{\pi}_n \otimes \boldsymbol{\delta}_n\,, \tag{5.38}$$

or equivalently:

$$\boldsymbol{\psi}_n \;=\; \boldsymbol{\pi}_n \otimes A_n^T(\boldsymbol{\psi}_{n+1} \oslash A_n \boldsymbol{\pi}_n)\,, \tag{5.39}$$

where $\otimes$ is element-wise multiplication and $\oslash$ element-wise division.

The weighted sample set $\{(\mathbf{s}_n^{(i)}, \psi_n^{(i)})\}$ then approximates the smooth density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T})$.

Likewise, the kernel forward-backward smoother with filter density as proposal may be rephrased. Here, $\boldsymbol{\beta}_{n+1}$ indicates the uncorrected filter weights at time t_{n+1}. Consequently, $\{(\mathbf{s}_n^{(i)}, \pi_n^{(i)})\}$ represents the filter density $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:n})$ at time t_n, and $\{(\mathbf{s}_{n+1}^{(i)}, \beta_n^{(i)})\}$ the uncorrected filter density $p(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:n})$ at time t_{n+1}.

Algorithm 5.4 (Kernel forward-backward smoother, matrix form) *Perform a filter forward in time, at the conclusion of which $p(\mathbf{x}_T \,|\, \mathbf{y}_{1:T})$ is known and approximated by $\{(\mathbf{s}_T^{(i)}, \psi_T^{(i)} = \pi_T^{(i)})\}$. Then, for time t_n, draw $\mathbf{s'}_{n+1}^{(i)} \sim p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_n = \mathbf{s}_n^{(i)})$ by propagating through the SDEs of the system, and let:*

$$A^{(j,i)} = \frac{1}{h^N |L_n|^{-1}} \mathcal{K}\Big(\frac{1}{h} \|L_n^{-1}(\mathbf{s'}_{n+1}^{(i)} - \mathbf{s}_{n+1}^{(j)})\|\Big)\,. \tag{5.40}$$

Then:

$$\boldsymbol{\gamma}_n \;=\; A_n^T \boldsymbol{\beta}_{n+1} \tag{5.41}$$

$$\boldsymbol{\delta}_n \;=\; A_n^T(\boldsymbol{\psi}_{n+1} \oslash \boldsymbol{\gamma}_n) \tag{5.42}$$

$$\boldsymbol{\psi}_n \;=\; \boldsymbol{\pi}_n \otimes \boldsymbol{\delta}_n\,, \tag{5.43}$$

or equivalently:

$$\boldsymbol{\psi}_n = \boldsymbol{\pi}_n \otimes (A_n^T \boldsymbol{\psi}_{n+1} \oslash A_n^T \boldsymbol{\beta}_n) , \tag{5.44}$$

The weighted sample set $\{(\mathbf{s}_n^{(i)}, \psi_n^{(i)})\}$ *then approximates the smooth density* $p(\mathbf{x}_n \,|\, \mathbf{y}_{1:T})$*.*

Note that this assumes support equivalence between $p_{\mathcal{K}}(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:T})$ and $p_{\mathcal{K}}(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:n})$, implying that the same matrix L_n is used to standardise both. These assumptions are as listed in Proposition 5.1.

We prove Proposition 5.1 first using a simple proof for a special case, and then a more elaborate proof for the general case.

Proof 5.1 (Of Proposition 5.1 in the absence of resampling) *Observe that if resampling has not been performed between times* t_n *and* t_{n+1} *then* $\boldsymbol{\beta}_{n+1} = \boldsymbol{\pi}_n$*. If propagations are additionally reused from the filter,* $\mathbf{s'}_{n+1}^{(i)} = \mathbf{s}_{n+1}^{(i)}$*, and the matrix* A_n *is symmetric by the symmetry property of the kernel* $\mathcal{K}(\|\cdot\|)$*. It follows that:*

$$\begin{aligned} \boldsymbol{\psi}_n &= \boldsymbol{\pi}_n \otimes (A_n^T \boldsymbol{\psi}_{n+1} \oslash A_n^T \boldsymbol{\beta}_n) & (5.45)\\ &= \boldsymbol{\pi}_n \otimes A_n^T (\boldsymbol{\psi}_{n+1} \oslash A_n \boldsymbol{\pi}_n) , & (5.46) \end{aligned}$$

deriving the forward-backward smoother from the kernel forward-backward smoother.
□

In the general case, resampling may have been performed between times t_n and t_{n+1}, such that $\boldsymbol{\beta}_{n+1} \neq \boldsymbol{\pi}_n$ and propagations may not be reused. The proof in this case relies on alternative forms for the correction factors $\boldsymbol{\gamma}_n$.

Proof 5.2 (Of Proposition 5.1 in the general case) *Consider the calculation of the correction weight for the kernel forward-backward smoother:*

$$\boldsymbol{\gamma}_n = A_n^T \boldsymbol{\beta}_{n+1} \,. \tag{5.47}$$

This represents a kernel density estimate for the query points $\{\mathbf{s'}_{n+1}^{(i)}\}$ *in* $p_{\mathcal{K}}(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:n})$ *over the weighted sample set* $\{(\mathbf{s}_{n+1}^{(j)}, \beta_{n+1}^{(j)}\}$*.*

Consider the alternative representation for $p_{\mathcal{K}}(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:n})$ *over the weighted sample set* $\{(\mathbf{s'}_{n+1}^{(j)}, \pi_n^{(j)}\}$*, given by the proposal distribution propagations, and perform* *a kernel density estimate using* $\{\mathbf{s}_{n+1}^{(i)}\}$ *as the query points. Notice that this exchanges the query and target points, providing an alternative but equally valid set of correction factors* $\boldsymbol{\gamma}_n$.

We now have:

$$\boldsymbol{\gamma}_n = A_n \boldsymbol{\pi}_n \,, \tag{5.48}$$

implying:

$$\begin{aligned} \boldsymbol{\psi}_n &= \boldsymbol{\pi}_n \otimes (A_n^T \boldsymbol{\psi}_{n+1} \oslash A_n^T \boldsymbol{\beta}_n) && (5.49) \\ &= \boldsymbol{\pi}_n \otimes A_n^T (\boldsymbol{\psi}_{n+1} \oslash A_n \boldsymbol{\pi}_n) \,, && (5.50) \end{aligned}$$

deriving the forward-backward smoother from the kernel forward-backward smoother.
□

5.4.2 Discussion

These results are useful to tie the kernel-based smoothers into existing work. In particular, they hint at the possibility of estimating the intractable transition density using single point estimators for other methods as well, such as the wider family of particle filters and smoothers that assume a closed form transition. We also highlight the similarity between such an approach and the random-weight particle filter of [6], discussed in §1.3.1.

Note also the significant resource advantage of the kernel forward-backward over the forward-backward smoother. As the transpose of A_n is never needed, there is no need to precalculate A_n at $\mathcal{O}(P^2)$ spatial complexity, or alternatively calculate its values twice. The space versus runtime tradeoff of the forward-backward smoother is therefore eliminated with the best result in both time and space.

5.5 Experiments

Now armed with methods capable of efficiently solving the smoothing problem for continuous-time dynamical systems, we extend the experimental results of the previous section and provide comparision.

5.5.1 Toy

The kernel forward-backward and kernel two-filter smoothers may be applied to the toy model introduced in §4.4.1. In review, as the system is linear, the RTS smoother provides an exact solution to the smoothing problem up to accumulated floating point error, and the forward-backward or two-filter smoother up to sampling and numerical integration error.

The proposed kernel forward-backward and kernel two-filter smoothers have an additional source of error in the form of the kernel density approximations. Figure 5.1 augments Figure 4.2 with the results of these two smoothers. Note that the forward-backward, kernel forward-backward and two-filter smoothers all use the same filter results for smoothing calculations.

5.5.2 Double well

For the double well model, both the kernel forward-backward and kernel two-filter smoothers may be immediately applied to the filter results already obtained for the double well model. We need only consider selection of an appropriate proposal distribution.

Use of the filter density is relevant, and as this is a stationary process, the equilibrium distibution may prove useful too. Because the autocorrelation for this model must be calculated numerically for each lag time, and because measurements arrive at irregular intervals, use of the autocorrelation function will prove too inefficient. Likewise, repeating the last smooth density as importance does not seem attractive, as the system exhibits rapid fluctuations.

We perform the smoothing using $P = 500$ particles, as for the filter, with the kernel bandwidth set to h_{opt}. The task is distributed across 4 processes on a quad-core processor.

Figures 5.2 & 5.3 present results for the kernel forward-backward smoother using equilibrium and filter proposal distributions, respectively. Figures 5.2 & 5.4 present results for the kernel two-filter smoother using equilibrium and uncorrected filter proposal distributions. Recall that the prior is precisely the equilib-

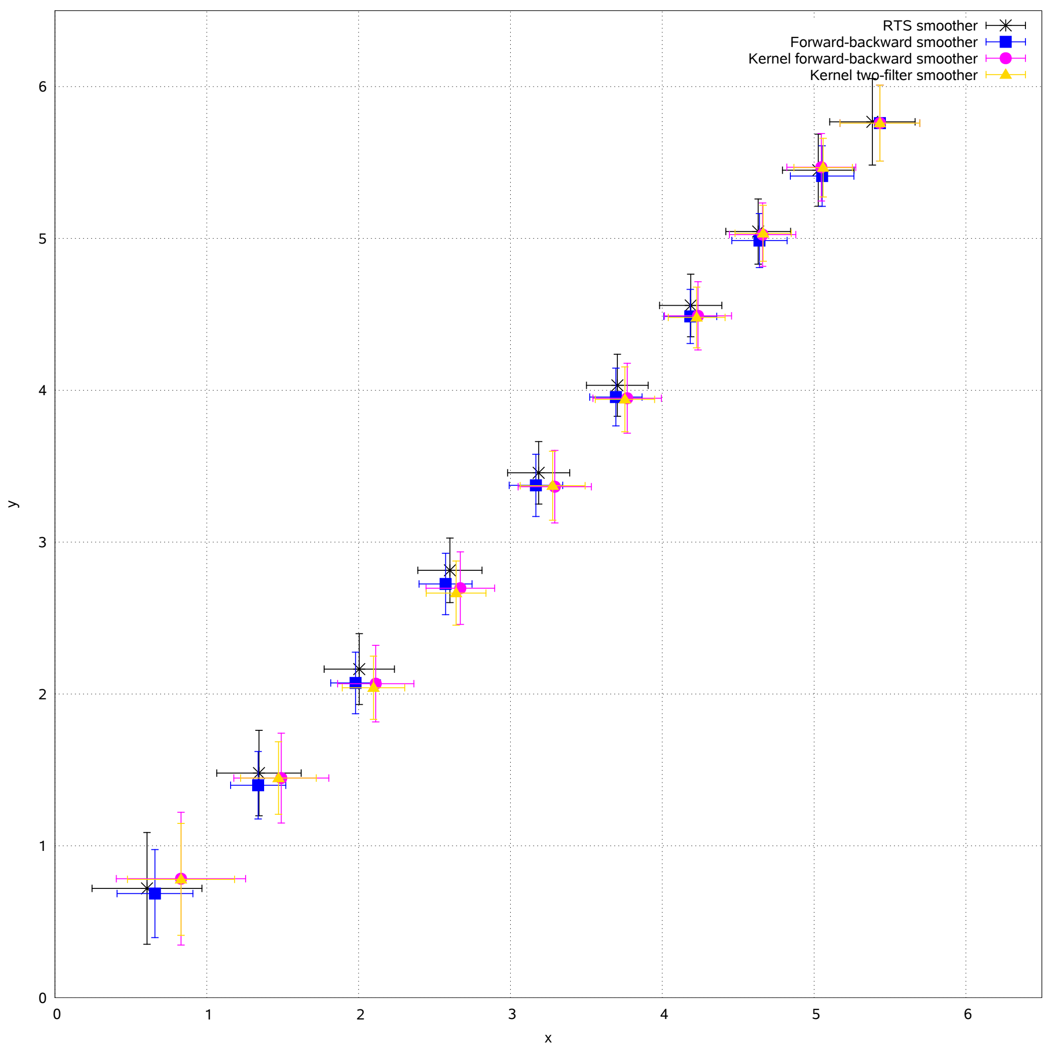


Figure 5.1: Filtered (x, y) position estimates for the toy model. Each point represents the mean for a particular method at a particular time, with bars indicating two standard deviations in either direction.

Method	RK(4)	EM(1)FIXED
Auxiliary filter	6.29	457.46
Forward-backward smoother	na	5891.77
Kernel forward-backward smoother, filter proposal	10.14	1156.02
Kernel two-filter smoother, filter proposal	15.69	1844.72
Kernel forward-backward smoother, equilibrium	27.20	2976.61
Kernel two-filter smoother, equilibrium	39.47	4710.00

Table 5.1: Runtime performance results for smoother and numerical scheme combinations on the double well model. All runs are performed in parallel across 4 processes on a single quad-core processor. Times are given in wallclock seconds. All smoothers are performed over results of the auxiliary filter, with runtime of the auxiliary filter excluded.

rium distribution, and so separate results for use of the prior as proposal distribution are not shown.

Table 5.1 provides runtime performance comparisons for these new methods versus the forward-backward smoother applied in §4.4.2.4. Substantial gains are apparent via their use in combination with the RK(4) numerical integration scheme.

5.6 Discussion

Results of the kernel forward-backward and kernel two-filter smoothers on the toy model (Figure 5.1) are close to that of the exact solution provided by the RTS smoother. The accuracy of the methods for these particular runs is noticeably less than that for the conventional forward-backward smoother, although recall the additional source of error in the form of kernel density approximations. This may not apply in general, particularly for more complex models which may be undersampled by P, where the kernel density could even improve accuracy.

In any case, performance results for the double well model (Table 5.1) very bluntly demonstrate runtime gains of over a hundred fold if such an additional source of error can be tolerated. A conventional forward-backward smoother, forced to use an Euler-Maruyama discretisation with fixed time step to ensure availability of

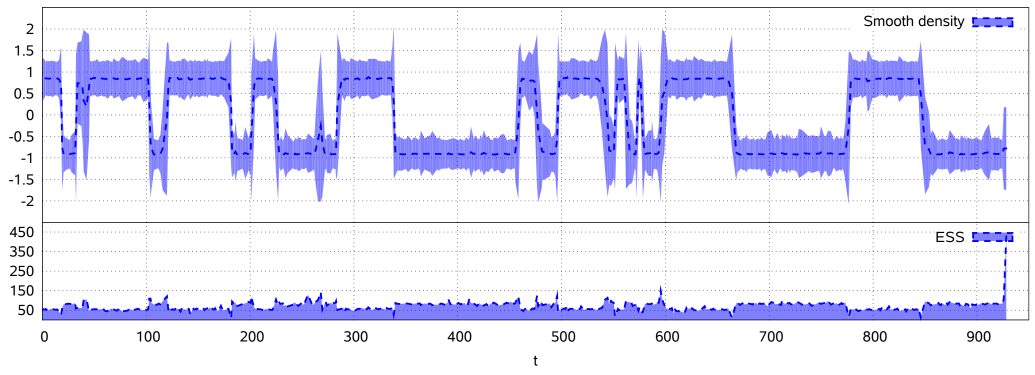


Figure 5.2: Smoothed results for the double well model using the kernel forward-backward smoother with equilibrium as proposal distribution.

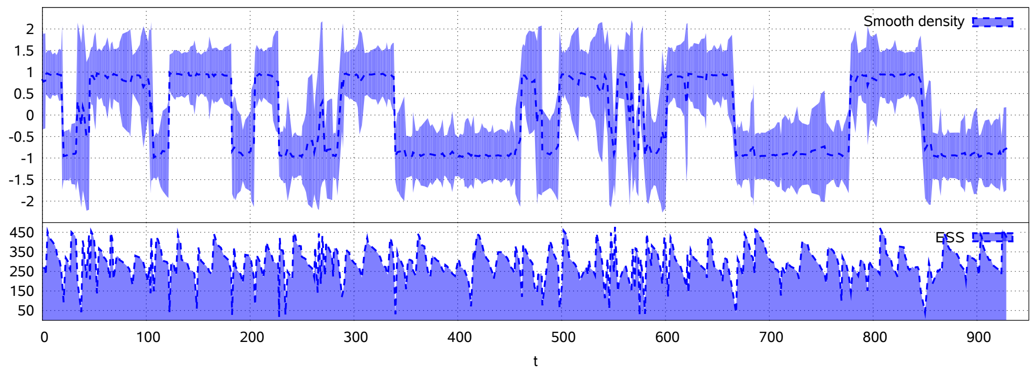


Figure 5.3: Smoothed results for the double well model using the kernel forward-backward smoother with filter density as proposal distribution.

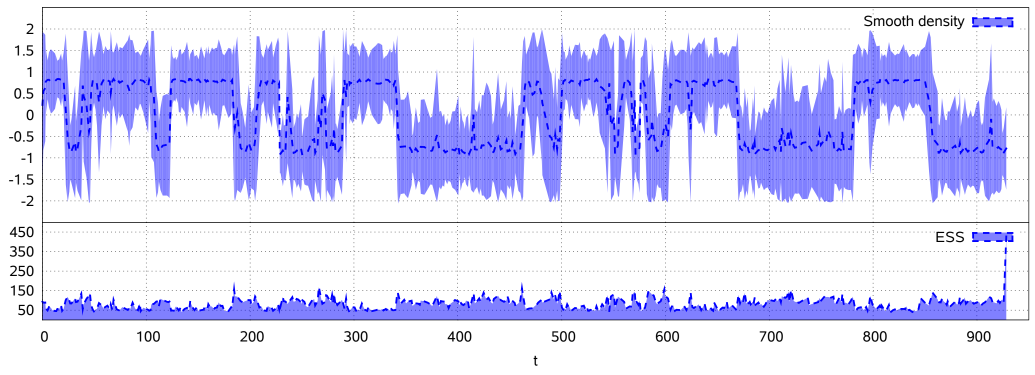


Figure 5.4: Smoothed results for the double well model using the kernel two-filter smoother with equilibrium as proposal distribution.

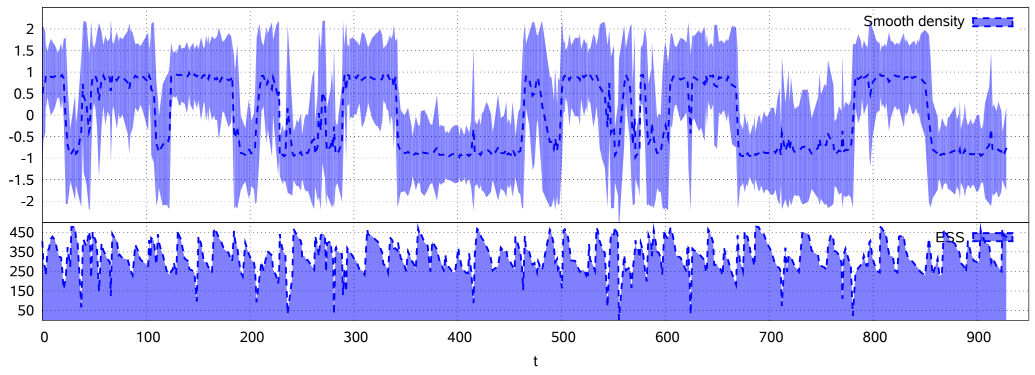


Figure 5.5: Smoothed results for the double well model using the kernel two-filter smoother with uncorrected filter density as proposal distribution.

a closed form transition density (see §4.2), performs substantially slower than our kernel methods tailored for the continuous-time case. Of the two, the kernel forward-backward smoother performs faster than the kernel two-filter. This is something we observe consistently. The difference is accounted for by more aggressive optimisations for the kernel forward-backward smoother, which are given in detail in the next chapter (§6).

Results of the kernel smoothers for the double well model (Figures 5.2-5.5), are on inspection comparable to that for the forward-backward smoother in the previous chapter (Figure 4.15). Similar effective sample size (ESS) is evident across time when comparing Figures 5.3 and 5.5 with Figure 4.15. This is to say that the kernel forward-backward and kernel two-filter smoothers are sampling the posterior as richly as the forward-backward smoother, at least when the filter and uncorrected filter densities, respectively, are used as proposal.

The equilibrium distribution provides poorer ESS, as seen in Figures 5.3 and 5.4. This is to be expected given that the equilibrium proposal draws samples equally from both stable states at any time, rather than biasing in favour of one or the other in the case of the filter density. Half of all samples are effectively wasted. Of couse, increasing P, or even h, can improve these results, we do not do so as this is a point worth demonstrating.

Interestingly, use of the equilibrium distribution seems to better capture transitions between the steady states when using the kernel forward-backward smoother, as seen by comparing Figures 5.2 and 5.3. By drawing new samples like this to support the smooth density, rather than being restricted to reweighting filter particles as in conventional forward-backward and two-filter smoothers, cases of degeneracy at sensitive points like this can be mitigated. This is one example of this.

By employing kernel densities to eliminate the transition density, we have decoupled the rate of the filter and smoother time scale from the SDE discretisation step size. This is more general and flexible than methods which rely on a fine linear discretisation and tie the filter and smoother to this resolution. Of course, this does not prevent the use of such a fine discretisation, it simply lifts the mandate of it. These kernel density methods may therefore prove particularly attractive in situations where measurements are sparse but where state estimates are not needed at a resolution as fine as that of the numerical integration scheme. Smoothing is performed much faster in such situations through the use of these methods.

As a final point, an interesting property of the methods proposed here is the ability to draw new samples to support the smooth density during the backward pass. This is in contrast to the filter-smoother, two-filter smoother and forward-backward smoother described in §2.3, which merely reweight particles originally drawn to support filter densities. While these methods may degenerate when the smooth and filter densities do not overlap significantly, the kernel forward-backward and kernel two-filter smoothers introduce a proposal distribution which may be specifically engineered to target this sweet spot. Combined with kernel estimators, permitting density calculations at arbitrary points, this may help to alleviate the smoother degeneracy problem in some situations.

5.7 Summary

This chapter has introduced the kernel forward-backward (§5.1) and kernel two-filter (§5.2) smoothers as methods suitable for solving the smoothing problem in the continuous time setting. The methods address the issues identified in §4 by:

- cancelling the intractable transition density in smooth weight calculations, and in doing so
- decoupling the rate of the filter and smoother from the numerical scheme used to integrate the SDE,
- permitting use of a higher-order numerical scheme for the integration, using an adaptive time step to permit error control, and
- providing an importance sampling framework for generating new samples to support the smooth density, providing a handle to address issues of degeneracy.

The chapter also included a brief discussion of how these methods may be coupled with a parameter estimation scheme (§5.3), in particular making mention of kernel conditioning to reduce dimensionality for the smoother when parameter values are fixed.

Manipulations in §5.4 tie these methods in with existing theory, while experimental results in §5.5 demonstrate the major speed advantage of these methods over conventional techniques.

Chapter 6

Implementation

Preceding chapters have isolated theoretical contributions from their concrete implementation, but actuation is a running theme of this work, not an afterthought. This chapter addresses the need for such implementation, and the additional challenges posed by it. It has the flavour of parallel and distributed computing, where computational power can be as liberating as clever mathematical formulation, and certainly complementary to it.

Parallelism demands attention throughout method development in the same way that dimensionality and noise do. Simple but parallelisable methods can – often do – trump sophisticated but inherently sequential approaches. Intuition can prove misleading under the intracacies of computer and network architectures. Implementation cannot be the assumed effortless aftermath of the abstract idea.

The approaches described here are realised in the open source `dysii` C++ library[1]. This chapter presents the design-level ideas of most general saliency, avoiding code level details that are better left to API documentation. The chapter separates distributed data structures (§6.2) from the algorithms applied over them (§6.3), first specifying the former to facilitate a clearer treatment of the latter. These are distilled into method-specific optimisations in §6.4, with some experimental results in §6.5.

[1]`http://www.indii.org/software/dysii/`

6.1 Introduction

Several code bases are available for particle filtering, and at least one for the integration of SDEs. On the Bayesian filtering side, the Bayesian Filtering Library[2] (C++) and `ReBeL`[3] (MATLAB) provide examples. On SDEs, Gardiner [17] promotes the `xmds`[4] package. Using the method of converting Itô to Stratonovich SDEs advocated here (§3.2.1), a myriad of implementations of methods for ODEs become available, including those of the GNU Scientific Library[5] (C). All of these provide serial implementations only.

Any implementation must be targeted to some computing platform. For relevancy, even at risk of fashion, targeting the current state of the art in computer systems seems the obvious choice. Parallelism seems the way forward in this regard. At the single processor level, multi-core chips capable of concurrent thread execution have become commonplace. Large scale networks of commodity hardware have eclipsed specialised supercomputers. A distributed memory, coarse-grain parallel implementation encompasses the trends across this complete range of scales. We target such an implementation.

Of all the methods of machine learning, particle filters are perhaps some of the most amicable to parallelisation. Immediately, one can consider propagating particles in parallel at each time step, synchronising only for the purposes of resampling. While a cursory search reveals much literature on various parallel particle filtering strategies, to our knowledge, there is no widely available code with parallel support available at time of writing. The one exception is the `dysii` project spun off from this work.

6.1.1 Assumptions

The implementations described here adopt the Single-Instruction Multiple-Data (SIMD) paradigm. They are suitable for a single processor up to several hundred, and indeed have been successfully applied across up to 200 [46]. We assume dis-

[2]`http://www.orocos.org/bfl/`
[3]`http://choosh.csee.ogi.edu/rebel/`
[4]`http://www.xmds.org/`
[5]`http://www.gnu.org/software/gsl/`

tributed memory, an assumption that encompasses shared memory as a special case, along with homogeneity of hardware, a more limiting assumption to simplify the algorithms, but one that is usually the case for purpose-built networks. We otherwise make no assumptions regarding architecture, and do not exploit particular hardware or network features.

The assumptions are lax enough that the algorithms are broadly applicable, although not necessarily optimised to particular parallel set-ups. In order of increasing scale, they have been tested on solo multi-core machines, networks of up to seven loosely connected compute servers, and the Eddie system of the Edinburgh Compute and Data Facility[6]. We also note that the methods do not rely on parallelism, and will work in a serial context also.

6.1.2 Terminology

Throughout our discussion we will make use of some fairly high-level pseudocode to more clearly articulate ideas. Recalling that the algorithms presented here are SIMD, in all cases the pseudocode is to be executed by all processes concurrently, unless otherwise specified.

We will refer to each process as a *node*, or *process node* where the interpretation may be ambiguous. The number of nodes in the parallel environment is referred to as its *size*. Nodes are numbered sequentially from $0, \dots, \mathit{size} - 1$, a number referred to as node *rank*. Both *size* and *rank* are considered global variables on each node.

For interprocess communication, we adopt the nomenclature of the Message Passing Interface (MPI)[7], in particular using *send* and *receive* to describe point-to-point communication, and *broadcast*, *gather*, *reduce* and so forth for collective communication. For the purposes of pseudo-code, these methods are defined as follows:

- SEND sends data to one node.
- RECV receives data from one node.
- BROADCAST sends data from one node to all nodes.

[6]`http://www.ecdf.ed.ac.uk/`
[7]`http://www.mpi-forum.org/`

- GATHER sends data from all nodes to one node, returning a vector of these data items indexed by rank.
- REDUCE sends an aggregation of data from all nodes to one node, such as a sum of a variable across all nodes.
- ALL-GATHER sends data from all nodes to all nodes, equivalent to calling GATHER once to each node.
- ALL-REDUCE sends an aggregation of data from all nodes to all nodes, equivalent to calling REDUCE once to each node.
- SCAN sends an aggregation of data from nodes $0, \ldots, rank$ to each node.

In context, the purpose of these operations should be clear enough without intricate knowledge of MPI.

Our discussion will not reach sufficient depth for synchronous versus asynchronous messaging to warrant much consideration. As always, however, asynchronous messaging is preferred where possible to better avoid deadlock, busy wait and other such undesirable states in a parallel environment.

6.2 Data structures

We first describe essential data structures and the operations defined on them, from which more complex algorithms will flow easily.

6.2.1 Distributed mixtures

Take any weighted mixture of probability densities:

$$p(\mathbf{x}) = \frac{1}{\sum_{i=1}^{P} \pi_i} \sum_{i=1}^{P} \pi_i p_i(\mathbf{x}) \,. \tag{6.1}$$

Note that the weights themselves are unnormalised, so need not sum to 1. In the context of particle filters, the $p_i(\cdot)$ will usually be Dirac δ functions, although in general may be Gaussians, kernel densities, or any other parametric or nonparametric densities. Let the full set of weighted densities, collectively referred to hereafter as *components*, be denoted by $\mathbb{P}$. Each node then adopts an exclusive subset $\mathbb{P}_{rank} \in \mathbb{P}$ such that $\bigcup_{rank=0}^{size-1} \mathbb{P}_{rank} = \mathbb{P}$, and $\bigcap_{rank=0}^{size-1} \mathbb{P}_{rank} = \emptyset$.

Code 6.1 Redistribution of mixture components across nodes.

```
REDISTRIBUTE-BY-SIZE
 1  $P \leftarrow$ ALL-REDUCE sum of $|\mathbb{P}_{rank}|$ across nodes
 2  $targetSize \leftarrow P \,/\, size$ ▷ integer division
 3  if $rank < P \bmod size$
 4      then $targetSize \leftarrow targetSize + 1$ ▷ take a leftover

 5  $excess \leftarrow |\mathbb{P}_{rank}| - targetSize$
 6  $excesses \leftarrow$ ALL-GATHER $excess$ across nodes

 7  while any element of $excesses$ is nonzero
 8        do $from \leftarrow rank$ of node with greatest excess
 9           $to \leftarrow rank$ of node with least (greatest negative) excess
10           $transfer \leftarrow \min(|excesses[from]|, |excesses[to]|)$
11           if $rank = from$
12              then SEND $transfer$ no. components to rank $to$
13           elseif $rank = to$
14              then RECV $transfer$ no. components from rank $from$

15           $excesses[from] \leftarrow excesses[from] - transfer$
16           $excesses[to] \leftarrow excesses[to] + transfer$
```

For most tasks, ideally $|\mathbb{P}_0| \approx |\mathbb{P}_1| \approx \ldots \approx |\mathbb{P}_{size-1}|$, as this will provide a roughly even workload over nodes to minimise waiting at points of synchrony. The REDISTRIBUTE-BY-SIZE method (Code 6.1) evens up the number of components on each node; nodes of rank $rank < P \bmod size$ adopting an extra component if necessary. Note that the procedure assumes efficient point-to-point communication between all pairs of nodes. On architectures where not all pairs of nodes can efficiently communicate directly, a more sophisticated strategy for pairing than the greedy strategy given here may be required.

Some tasks require that each node has access to the complete set of components $\mathbb{P}$. In general, it cannot be assumed that any one node has sufficient memory to hold the entire set simultaneously, however. The ROTATE method (Code 6.2) is

Code 6.2 Rotation of mixture components between nodes.

```
ROTATE
1  SEND $\mathbb{P}_{rank}$ to node $rank + 1 \pmod{size}$
2  RECV new $\mathbb{P}_{rank}$ from node $rank - 1 \pmod{size}$
```

defined to rotate the subsets around the nodes by rank, allowing each to operate on a subset of the components in turn. After *size* calls to ROTATE, each node will be returned its original subset $\mathbb{P}_{rank}$.

ROTATE calls may be nested in loops of arbitrary depth for problems requiring all-pair matching between two sets, or all-combination matchings between more. One such example is the calculation of the α terms in Algorithm 2.7.

6.2.2 Partition trees

The flat representation of distributed mixtures can be limiting for some operations, in particular for kernel density evaluations, upon which the methods of this work rely heavily. For these tasks, a hierarchical tree representation can be more useful.

The motivation is straightforward. Consider a kernel density built over the set of samples $\{\mathbf{s}^{(i)}\}$ for $i = 1, \ldots, P$. Now consider the query point $\mathbf{x}$, at which to evaluate the density. Naively, the solution is obtained simply by evaluating $\mathcal{K}(\|\mathbf{x} - \mathbf{s}^{(i)}\|)$ for $i = 1, \ldots, P$, then summing and normalising to obtain the density. This requires P norm and kernel evaluations.

Now consider two samples in the density $\mathbf{s}^{(i)}$ and $\mathbf{s}^{(j)}$ such that $\|\mathbf{x} - \mathbf{s}^{(i)}\| < \|\mathbf{x} - \mathbf{s}^{(j)}\|$. We expect, therefore, that $\mathcal{K}(\|\mathbf{x} - \mathbf{s}^{(i)}\|) > \mathcal{K}(\|\mathbf{x} - \mathbf{s}^{(j)}\|)$. Such bounds can be exploited for more efficient approximate density evaluations. In the first instance, under the rounding approximation of floating point arithmetic, if $\mathcal{K}(\|\mathbf{x}-\mathbf{s}^{(i)}\|) = 0$, then immediately $\mathcal{K}(\|\mathbf{x}-\mathbf{s}^{(j)}\|) = 0$ without further evaluation. Potentially, this means significantly fewer than P norm and kernel evaluations.

These properties can be harnessed by erecting a *partition tree* over the components of the mixtures introduced in §6.2.1 to provide information on spatial

Code 6.3 Building a *kd* tree.

```
BUILD-KD-TREE($\mathbb{P}$)
 1  if $|\mathbb{P}| = 1$
 2      then $result \leftarrow$ a leaf node containing this one component
 3      else  select a dimension $index$
 4            select a value along this dimension $value$
 5            $\mathbb{L} \leftarrow \{\mathbf{x} \in \mathbb{P} : x_{index} \leq value\}$
 6            $\mathbb{R} \leftarrow \{\mathbf{x} \in \mathbb{P} : x_{index} > value\}$
 7            $left \leftarrow$ BUILD-KD-TREE($\mathbb{L}$)
 8            $right \leftarrow$ BUILD-KD-TREE($\mathbb{R}$)
 9            $result \leftarrow$ an internal node with children $left$ and $right$
10  return $result$
```

relationships. We employ *kd* trees [44] for this purpose.

Starting with the complete set of components $\mathbb{P}$, a *kd* tree is constructed using the BUILD-KD-TREE method of Code 6.3. After construction, the leaf nodes of the tree each contain a single sample, while the internal nodes envelope a hyper-rectangle of space enclosing all the samples of their descendent nodes. These hyper-rectangles may be represented simply by a lower and upper bound. Figure 6.1 visualises this for a particular set of sample points in two-dimensional space.

For the purposes of later discussion, we add to the *leaf* and *internal* node classes the *prune* node class. A prune node is a leaf node containing more than one component. It represents a collapsing of all the descendants of an internal node into a terminating node. This may be needed if it is not possible to split a subset of points, such as if all points are the same.

Selection of the dimension upon which to split in line 3 of Code 6.3 can follow any one of a number of strategies, including:

- the dimension of largest variance,
- the dimension of largest range, or
- a random dimension.

This list is certainly not exhaustive, but it does suggest reasonable alternatives.

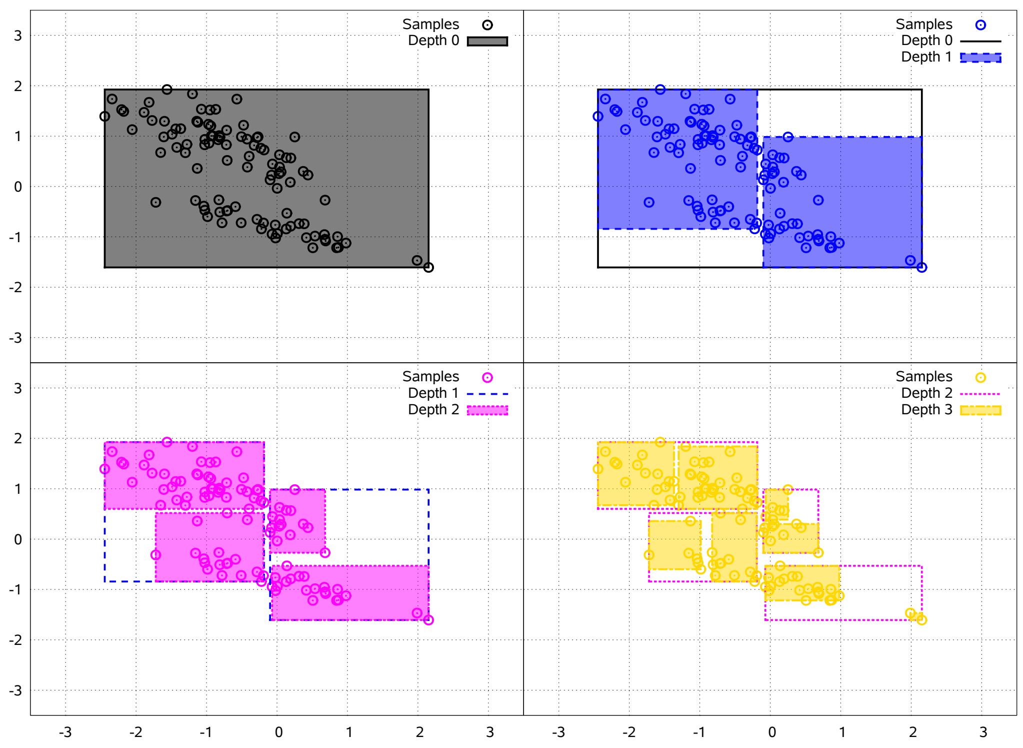


Figure 6.1: Example kd tree partitioning in two dimensions; **(top left)** the root node, depth 0, with hyper-rectangle enclosing all samples, **(top right)** the two children of the root node, depth 1, enclosing a subset of the samples each, **(bottom left)** nodes at depth 2, **(bottom right)** nodes at depth 3. In the latter three cases the space enveloped by parent node hyper-rectangles has been traced for clarity.

Note the time complexity of these is $\mathcal{O}(P)$, $\mathcal{O}(P)$ and $\mathcal{O}(1)$, respectively, in the number of sample points.

Similarly, the value on which to split in line 4 can be selected according to:

- the mean,
- the median, or
- the midpoint.

Again, this is not exhaustive. Time complexities in all cases are $\mathcal{O}(P)$, although some savings are made by coupling with a complementary dimension selection strategy (e.g. the mean is already calculated if using the dimension of largest variance, the midpoint calculation is $\mathcal{O}(1)$ if dimension ranges have already been calculated).

A number of other partition trees are available, including ball trees. These may or may not improve performance, depending on the particular task. Our implementation is restricted to *kd* trees, largely due to their simplicity and amicability to parallelisation.

6.2.3 Distributed partition trees

Naively, it is possible to simply construct a *kd* tree over $\mathbb{P}_{rank}$ independently on each node to accelerate evaluations on that node. This is not equivalent to building a single *kd* tree over all of $\mathbb{P}$, however, and as it does not encode spatial information between samples on different nodes, is suboptimal. We instead propose a method for distributing the entire *kd* tree across nodes in such a way that spatial characteristics are consistent, and the stripping of norm and kernel evaluations is as aggressive as for a single *kd* tree. This delivers a linear improvement in runtime order in the *size* of the system, rather than a sublinear improvement in the naive case.

The idea is to construct a *kd* tree distributed across all process nodes, that has no leaf nodes, but *size* number of prune nodes (see above) of a roughly equal number of components. Components are then redistributed across process nodes so that each stores all the components of a single prune node, and no other. From this point, the remainder of the *kd* tree is rolled out independently on each node, by the usual construction over its new set of components.

Using any dimension selection strategy, but selecting the nth ordered element along that dimension upon which to split, an equal (up to leftovers) number of components can be placed on each node. The nth element along a dimension may be efficiently selected in $\mathcal{O}(P)$ by using a distributed extension to Hoare's algorithm [47], in Code 6.4. During the execution of this algorithm, it is possible that some nodes have exhausted their entire subset onto one side of the pivot *guess* while other nodes have barely depleted any of their elements and are essentially still working with a full set. In practice this has been found to have no real detriment, however [47].

Redistribute-By-Space (Code 6.6) and its auxiliary function Distributed-Build-Kd-Tree (Code 6.5) construct the pruned *kd* tree and redistribute com-

Code 6.4 Distributed nth element select across nodes.

```
NTH-ELEMENT($n, \mathbb{P}$)
    ▷ select guessing node
    $guesser \leftarrow 0$
    repeat  if $rank = guesser$
               then if $|\mathbb{P}| > 0$
                        then $okay \leftarrow$ TRUE
                        else $okay \leftarrow$ FALSE
            BROADCAST $okay$ from rank $guesser$ to all nodes
            if $\neg\, okay$
               then $guesser \leftarrow guesser + 1$ ▷ pass on responsibility
        until $okay$

    ▷ guess $n$th element
    if $rank = guesser$
        then $guess \leftarrow$ an element of $\mathbb{P}_{rank}$
    BROADCAST $guess$ from rank $guesser$ to all nodes

    ▷ partition
    partition $\mathbb{P}_{rank}$ into $\mathbb{L}_{rank}$ and $\mathbb{R}_{rank}$ on $guess$, excluding $guess$ from both

    ▷ continue search?
    $L =$ ALL-REDUCE sum of $|\mathbb{L}_{rank}|$ across nodes
    if $L = n - 1$
        then $result \leftarrow guess$
    elseif $n \leq L$
        then $result \leftarrow$ NTH-ELEMENT($n, \mathbb{L}_{rank}$) ▷ $n$th is in left partition
    else $result \leftarrow$ NTH-ELEMENT($n - |\mathbb{L}_{rank}| - 1, \mathbb{R}_{rank}$) ▷ is in right partition

    return $result$
```

Code 6.5 Build pruned *kd* tree across nodes.

```
DISTRIBUTED-BUILD-KD-TREE($\mathbb{P}$, $numNodes$)
if $numNodes = 1$
    then $result \leftarrow$ a prune node containing all of $\mathbb{P}$
    else $P \leftarrow$ ALL-REDUCE sum of $|\mathbb{P}|$ across nodes
         select a dimension $index$

         $numLeftNodes = numNodes\ /2$ ▷ integer divison
         $numRightNodes = numNodes - numLeftNodes$

         $n \leftarrow P \times leftNumNodes\ /\ numNodes$ ▷ integer division
         $value \leftarrow$ NTH-ELEMENT($n, \mathbb{P}$)

         $\mathbb{L} \leftarrow \{\mathbf{x} \in \mathbb{P} : x_{index} \leq value\}$
         $\mathbb{R} \leftarrow \{\mathbf{x} \in \mathbb{P} : x_{index} > value\}$
         $left \leftarrow$ DISTRIBUTED-BUILD-KD-TREE($\mathbb{L}$, $numNodesLeft$)
         $right \leftarrow$ DISTRIBUTED-BUILD-KD-TREE($\mathbb{R}$, $numNodesRight$)

         $result \leftarrow$ an internal node with children $left$ and $right$
return $result$
```

ponents across nodes based on this. Afterwards, BUILD-KD-TREE($\mathbb{P}_{rank}$) may be called independently on each node to complete the construction of the tree. As the number of components is balanced by this point, any partitioning strategy may be used, and no further communication is required between nodes to complete the task.

6.3 Algorithms

A parallel bootstrap, auxiliary or regularised particle filter can be run almost entirely independently across nodes. Nodes need only synchronise for the purpose of normalising weights before resampling. In addition, however, it is desirable

Code 6.6 Redistribution of components across nodes to facilitate distributed *kd* tree.

```
Redistribute-By-Space
1  Distributed-Build-Kd-Tree($\mathbb{P}_{rank}$, $size$)
2  enumerate the prune nodes of the tree $0, \ldots, size - 1$
3  for $i \leftarrow 0$ to $size - 1$
4        do Gather the components of the $i$th prune node to rank $i$
5  reconstruct $\mathbb{P}_{rank}$ from the gathered components
```

that they also synchronise *after* resampling to balance their work load as much as possible.

The Redistribute-By-Size method of Code 6.1 solves the latter of these two problems. The former requires a distributed resampling strategy, which we discuss in §6.3.1.

The smoothing problem, as addressed by the kernel and conventional forward-backward and two-filter smoothers, is most inhibited by all-pairs type problems. In all cases these may be address using partition trees (see [12] for the standard methods, this work for others). We therefore concentrate on efficient *kd* tree operations for kernel density evaluations, although they remain applicable in general. These are given in §6.3.2.

6.3.1 Resampling

We begin with the stratified resampling algorithm of [23], commonly cited throughout the literature. The serial version of this algorithm is given in Code 6.7. This divides the sequence of weights into P *strata* of equal width by weight, selecting one component from each strata to resample.

[23] suggests three alternative strategies for the selection of u in this algorithm:

random strategy where $u \sim \mathcal{U}[0, W]$,
stratified strategy where $u \sim \mathcal{U}[\frac{i-1}{P}, \frac{i}{P})$, and
deterministic strategy where $u \leftarrow \frac{i+\alpha}{P}$ for some $\alpha \sim \mathcal{U}[0, 1)$ fixed for all i.

Code 6.7 Stratified resampling [23].

```
Resample
1  enumerate all $(\mathbf{s}^{(i)}, \pi^{(i)}) \in \mathbb{P}$
2  $W \leftarrow \sum_{i=0}^{P} \pi^{(i)}$
3  for $i \leftarrow 0$ to $P$
4        do $u \leftarrow$ some number in $(0, W]$
5           find $k$ such that $\sum_{j=0}^{k-1} \pi^{(j)} < u \leq \sum_{j=0}^{k} \pi^{(j)}$
6           add $(\mathbf{s}^{(k)}, 1)$ to $\mathbb{Q}$
7  return $\mathbb{Q}$
```

Pre-sorting of the components by weight is also considered. The results indicate that the deterministic strategy with sorting is most effective, while also concluding that, "...considering the significant computational cost in sorting, the deterministic algorithnm without sorting might be a reasonable choice," [23].

This computational cost in sorting is additionally confounded in the parallel setting, where complete sorting of the data set may require extensive communication and thus be inefficient. The deterministic algorithm, without sorting, may be extended to the parallel setting, as in Code 6.8. The sequence of weights and strata is set up across all nodes. The main issue is determining which node is to sample from a strata that overlaps two or more nodes. The deterministic scheme, with constant α for all strata, allows this to be determined easily.

After resampling, Redistribute-By-Size should be called to rebalance across nodes. This raises the issue of memory consumption in the imbalanced state prior to the redistribution, in particular whether one node will have substantive enough weight that its memory resources will be consumed by an overburden of resampled particles. One can imagine a number of strategies to combat this. Redistributing by weight prior to resampling is one, although this reeks of the NP-hard bin packing problem, and may simply shift the memory issue from post-resampling to prior. A greedy approach to this has been implemented, but not found to be particularly effective. A redistribution by both size and weight would be more effective, although an order of magnitude more difficult also. We leave these ideas to future work, taking the pragmatic approach of increasing the

Code 6.8 Distributed deterministic stratified resampling.

```
DISTRIBUTED-RESAMPLE
1  enumerate all $(\mathbf{s}^{(i)}, \pi^{(i)}) \in \mathbb{P}_{rank}$
2  $W_{rank} \leftarrow \sum_{i=0}^{P} \pi^{(i)}$
3  $W_{scan} \leftarrow$ SCAN sum of $W_{rank}$ across nodes
4  $W \leftarrow$ BROADCAST $W_{scan}$ from rank $size - 1$ to all nodes
5  $P \leftarrow$ ALL-REDUCE sum of $|\mathbb{P}_{rank}|$ across nodes

6  if $rank = 0$
7     then $\alpha \sim \mathcal{U}[0,1)$
8  BROADCAST $\alpha$ from rank 0 to all nodes ▷ total weight

9  $w \leftarrow \frac{W}{P}$
10 $rem \leftarrow (W_{scan} - W_{rank}) \bmod w$
11 if $rem \geq \alpha w$
12    then ▷ another node will sample from this strata
13          $u \leftarrow (1 + \alpha)w - rem$
14    else ▷ this node samples from this strata
15          $u \leftarrow \alpha w - rem$

16 while $u < W_{rank}$
17       do
18          find $k$ such that $\sum_{j=0}^{k-1} \pi^{(j)} < u \leq \sum_{j=0}^{k} \pi^{(j)}$
19          add $(\mathbf{s}^{(k)}, 1)$ to $\mathbb{Q}$
20          $u \leftarrow u + \frac{W}{P}$

21 $\mathbb{P}_{rank} \leftarrow \mathbb{Q}$
```

Code 6.9 Single tree kernel density evaluation using *kd* tree.

```
SINGLE-TREE-DENSITY($\mathbf{x}$, densityNode)
 1  if densityNode is a leaf node
 2     then result ← $\mathcal{K}(\frac{1}{h}\|\mathbf{x} - \mathbf{s}^{(i)}\|)$
 3     else ▷ densityNode is an internal node
 4           $\mathbf{s}$ ← nearest point to $\mathbf{x}$ in space encompassed by densityNode
 5           if $\mathcal{K}(\|\mathbf{x} - \mathbf{s}\|) > 0$
 6              then ▷ recurse
 7                    left ← left child of densityNode
 8                    right ← right child of densityNode
 9                    result ← SINGLE-TREE-DENSITY($\mathbf{x}$, left)
10                    result ← result +SINGLE-TREE-DENSITY($\mathbf{x}$, right)
11  return result
```

number of nodes if memory issues become apparent.

Auxiliary and regularised resampling are straightforward extensions.

6.3.2 Kernel density evaluation

Kernel density evaluations are performed by exploiting the spatial relationships encoded in a partition tree. SINGLE-TREE-DENSITY (Code 6.9) calculates the density at a single point $\mathbf{x}$. Beginning at the root node of the tree, it determines the nearest point to $\mathbf{x}$ in the space enveloped by the node's bounding box. It then evaluates the kernel over the distance between this point and $\mathbf{x}$ to determine the maximum density contribution by any point in this subtree. If this is deemed significant, the algorithm recurses through the subtree. If it is not deemed significant, the algorithm ceases recursion through this branch of the tree. By ceasing its recursion in this way, the algorithm is able to perform the density calculation without exhaustively comparing $\mathbf{x}$ to all P components under the tree.

Note that the algorithm assumes floating point precision, such that the zero comparison in line 5 is meaningful, even for kernels of infinite extent, due to rounding error. The evaluation represented here is therefore exact up to floating point limitations. Note also that as a comparison against zero is used, the weight of components encompassed by the node is irrelevant.

Multiple density evaluations may be carried out by calling SINGLE-TREE-DENSITY for each query point, but further gains are made by also constructing a partition tree from the query points. This leads to the dual-tree algorithm, DUAL-TREE-DENSITY of Code 6.10.

We add to these methods a significant optimisation in the case that the query and target trees are identical. In this case the symmetry of the kernel may be exploited to curtail the recursion, combining lines 17 & 18 in Code 6.10 into one evaluation that adds the calculated density contributions to *result* for both the query and target nodes, observing that $\textit{left}_1 = \textit{left}_2$ and $\textit{right}_1 = \textit{right}_2$. We refer to this modified procedure as SELF-TREE-DENSITY.

A number of heuristics for approximate density evaluation are available, trading off accuracy to speed. These are based on evaluating both the maximum and minimum density contribution of each node, and pruning the evaluation to the average of these if they are sufficiently close, rather than recursing down the tree. We do not discuss such optimisations here, largely because they complicate presentation of the algorithms, and indeed our implementation does not yet exploit them. We instead simply point the reader to [44].

In some situations there is cause to evaluate densities for multiple kernel bandwidths over the same query and density trees. In such situations the multi-bandwidth dual-tree algorithm [48] may be used, allowing multiple density evaluations in one pass of the *kd* tree, in a runtime approaching that of a single density evaluation. This exploits the fact that for two bandwidths h_{low} and h_{high}, $h_{\text{low}} < h_{\text{high}}$, if $\mathcal{K}(\frac{1}{h_{\text{high}}}\|\cdot\|)$ is not significant, $\mathcal{K}(\frac{1}{h_{\text{low}}}\|\cdot\|)$ will not be either, as it is a tighter distribution.

Finally, we can simultaneously perform queries over the same target sample points with different weight combinations, which we refer to as *multi-weight* calculations. Recalling the matrix representation of kernel density evaluations given in Definition 5.2, this is conceptually equivalent to precalculating the matrix A and multiplying it with various weight vectors $\boldsymbol{\pi}$.

Code 6.10 Dual tree kernel density evaluation using *kd* trees.

```
DUAL-TREE-DENSITY(queryNode, densityNode)
if queryNode is a leaf node
   then s ← single component of queryNode
        result ← SINGLE-TREE-DENSITY(s, densityNode)
elseif densityNode is a leaf node
   then s ← single component of densityNode
        result ← SINGLE-TREE-DENSITY(s, queryNode)
else
        ▷ both are internal nodes
        $(\mathbf{s}_1, \mathbf{s}_2)$ ← pair of points in queryNode and densityNode with
                   minimal distance between
        if $\mathcal{K}(\|\mathbf{s}_1 - \mathbf{s}_2\|) > 0$
          then ▷ recurse
               $left_1$ ← left child of queryNode
               $right_1$ ← right child of queryNode
               $left_2$ ← left child of densityNode
               $right_2$ ← right child of densityNode
               result ← DUAL-TREE-DENSITY($left_1$, $left_2$)
               result ← result +DUAL-TREE-DENSITY($left_1$, $right_2$)
               result ← result +DUAL-TREE-DENSITY($right_1$, $left_2$)
               result ← result +DUAL-TREE-DENSITY($right_1$, $right_2$)
return result
```

6.4 Method optimisations

We now discuss optimisations specific to each of the smoothing methods introduced in this work. The applicability of these optimisations is model dependent, and may require a tradeoff between computational cost and estimation accuracy.

6.4.1 Kernel forward-backward smoother

A number of optimisations can be used to improve performance of the kernel forward-backward smoother when the filter density is used as proposal distribution:

Preserve propagations from filter If resampling has not been performed between times t_n and t_{n+1}, each sample $\mathbf{s}_n^{(i)}$ has already been propagated through the SDE dynamics during the preceding filter. These propagations may be preserved for use in smoothing, obtaining $\mathbf{s}'^{(i)}_{n+1}$ immediately. In light of the expensive propagation of particles discussed in §4.2, this can deliver a significant runtime improvement. In this case, observe also that the samples supporting $p_{\mathcal{K}}(\mathbf{x}_n \,|\, \mathbf{y}_{1:T})$ and $p_{\mathcal{K}}(\mathbf{x}_{n+1} \,|\, \mathbf{y}_{1:n})$ are identical, with only weights differing. This facilitates the remaining optimisations.

Share *kd* tree between densities In the case that the partition function used in building *kd* trees is not dependent on their weights, only one *kd* tree need be constructed. Partitioning on the midpoint of the longest dimension satisfies this criterion, for example.

Share kernel between densities If the kernel used for the two unknown densities is also not dependent on weights, and varies only in bandwidth, the two kernel density calculations may be performed simultaneously in one pass of the *kd* tree using the multi-bandwidth algorithm with multiple weights. If the kernel bandwidth is the same, only the multiple weights need be considered. We note that this is inapplicable if standardisation is used in conjunction with the kernel density.

Self dual-tree query If the above two conditions hold, significant gains may be made by observing that the query points from the propagations are identical to the target points in the kernel density. The evaluation is therefore symmetrical, and may be performed using SELF-TREE-DENSITY.

Use of a static importance distribution for all time steps, such as the equilibrium distribution, opens another potential avenue for optimisation:

Sharing samples In the case that $q(\mathbf{x}_n)$ is to be the same for all time points (e.g. the equilibrium distribution), the samples $\mathbf{s}^{(i)} \sim q(\mathbf{x}_n)$ need only be drawn once and reused for all time points.

Sharing propagations In the case that the time points $t_1, \ldots, t_T$ are also equidistant and the SDE dynamics independent of time (e.g. it is not input driven), these shared samples may be propagated through $f(\cdot)$ only once, and these propagations $\{\mathbf{s}'^{(i)}\}$ may be reused for all time points also.

In the ideal case, where all of these optimisations may be applied, each step of the kernel forward-backward smoother reduces to a single self-tree density evaluation.

6.4.2 Kernel two-filter smoother

Largely the same optimisations may be applied to the kernel two-filter smoother as to the kernel forward-backward smoother, so we do not discuss this further.

6.5 Experiments

Experimental results for the proposed methods are difficult to obtain due to model and distributional dependencies. Nonetheless, we present some simple results here.

Figure 6.2 provides a comparison of exhaustive versus dual-tree evaluation of a kernel density. Query and target samples are drawn from randomly generated Gaussian mixtures. Figure 6.3 similarly compares the dual-tree and self-tree algorithms for the case that the query and target trees are identical.

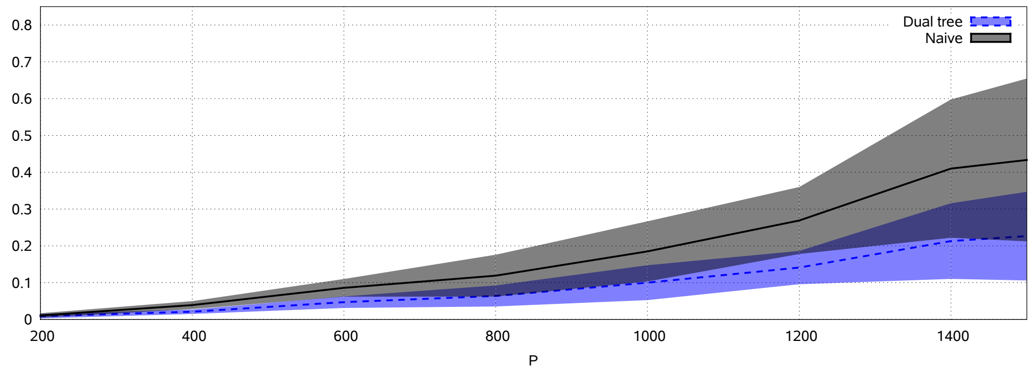


Figure 6.2: Runtime improvement of dual-tree over exhaustive density evaluations for randomly generated 5-component Gaussian mixture query and target distributions in 5 dimensions. Areas represent standard deviation across 10 runs for each value of P. Run is on a single node only.

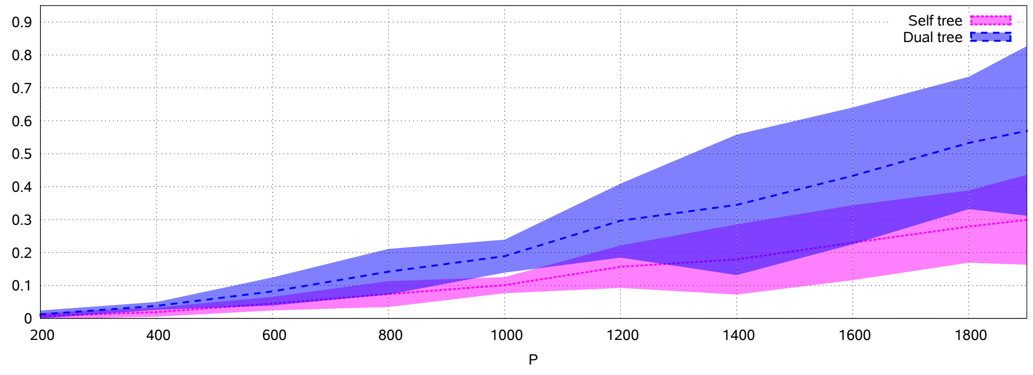


Figure 6.3: Runtime improvement of self-tree over dual-tree density evaluations for randomly generated 5-component Gaussian mixture in 5 dimensions. Areas represent standard deviation across 10 runs for each value of P. Run is on a single node only.

6.6 Summary

This chapter has provided an optimised implementation of the methods presented in preceding chapters in a parallel and distributed context (§6.4). The crux of this is the use of distributed data structures (§6.2.1) and partition trees (§6.2.2), on which efficient dual-tree algorithms, and more so the self-tree algorithm (§6.3.2), may be used for fast kernel density evaluations.

Part II

Applications in Functional Magnetic Resonance Imaging

Chapter 7

A Brief Introduction

Functional Magnetic Resonance Imaging (fMRI) poses large-scale and challenging problems for machine learning research. Before applying the Bayesian filtering and smoothing methods developed in Part I to problems in this domain, we first present a brief introduction to essential material in this chapter.

The chapter first introduces fMRI and the imaging data it produces in §7.1. Brain hemodynamics, essential for the development of plausible biophysical models for describing the phenomena underlying fMRI imaging, are reviewed in §7.2. We then consider the typical use of fMRI, including experimental design (§7.3), data handling (§7.4) and data analysis (§7.5). Our review focuses on effective connectivity studies and the methods used for them, in particular Structural Equation Modelling (SEM) in §7.5.1 and Dynamic Causal Modelling (DCM) in §7.5.2.

7.1 Functional Magnetic Resonance Imaging

Magnetic Resonance Imaging (MRI) exploits the Nuclear Magnetic Resonance (NMR) phenomenon in Hydrogen to produce three dimensional images of the human body, and of particular interest here, the brain. *Functional Magnetic Resonance Imaging* (fMRI) is an MRI technique for measuring neuronal activity in the brain. Two fundamental advances have made this possible. The first is the ability to use NMR to contrast between oxygenated and deoxygenated hemoglobin, known as the Blood Oxygen Level Dependent (BOLD) contrast, and the second is fast image acquisition using Echo Planar Imaging (EPI).

Neural activity elicits a hemodynamic response consisting of a localised increase in blood flow combined with the metabolism of oxygen. Oxyhemoglobin is diamagnetic, whereas deoxyhemoglobin is paramagnetic [49]. In addition, the signal decay rate of deoxyhemoglobin after an RF pulse is faster than that of oxyhemoglobin [50]. As a consequence of these different magnetic properties, oxygenated and deoxygenated hemoglobin exhibit characteristically different NMR signals, and this allows an MRI scanner to contrast between them. This is known as the Blood Oxygen Level Dependent (BOLD) contrast, and allows neural activity to be spatially localised.

In order for such a measurement of functional activity to be meaningful, images must be acquired at a reasonable temporal resolution. While a high spatial resolution allows more precise localisation of neuronal activity, this becomes meaningless if the image is overexposured by the several minutes required to acquire this. EPI is an acquisition method which provides a suitable tradeoff between spatial and temporal resolution to make BOLD images meaningful representations of functional activity.

EPI is a *single shot* technique in that it acquires a complete slice of the brain with a single RF pulse. In contrast, many methods acquire only a single voxel per pulse. A slice is therefore acquired very quickly, typically in 30–100 ms. Resolution does suffer as a result, however, with slices typically having a resolution of only 64×64 pixels. This corresponds to individual voxel sizes of the order of a few millimetres cubed. Whole volumes are acquired slice by slice, with the time taken to acquire the whole volume directly proportional to the number of slices, typically 2–4s in total.

Some peculiar properties of EPI are worth mentioning. Firstly, as single slices are acquired so quickly, each slice has negligible artifacts resulting from subject movement. This is a reasonable assumption to make during data analysis [51, ch.5]. Secondly, inaccuracies in slice selection by the gradient coils of the scanner result in some stimulation of tissue adjacent to the slice being scanned. To avoid this interfering with subsequent scans, one of two strategies is used. The first is to leave a gap between slices, typically of 1mm. The second is to interleave the slice acquisition, so that, for example, all odd numbered slices are acquired first, followed by all even numbered slices. The slice acquisition order has important implications for properly treating the temporal extent of the acquired data.

A primer on the numerous intracacies of image acquisition is beyond the scope of this review, but there are some general issues worth mentioning. Once a region of tissue has been scanned, it takes a certain amount of time for the magnetic field in that region to relax back to baseline. If the region is scanned again before this relaxation is complete, the resulting NMR signal will be weaker than the first. It is therefore necessary to wait some time between consecutive scans of the same location, referred to as the *repetition time* and abbreviated TR. This limits the temporal resolution of the data, typically to 2–4s between the start of each volume acquisition.

See [50] for a great introduction to some of the earlier work in fMRI and its basic principles. For a more modern and thorough work see [51].

7.2 Hemodynamics

While the BOLD signal is generally accepted as being proportional to some measure of neural activity, the precise nature of this connection is still the subject of significant debate. [1] provides a critical review of the debate. The basic understanding is that neural activity has metabolic demands, such that an increase in neural activity causes an increase in the Cerebral Metabolic Rate of Oxygen ($CMRO_2$) in the surrounding capillary bed. The vascular system responds with a delayed surge of fresh arterial blood, increasing Cerebral Blood Flow (CBF) through the affected area and consequently Cerebral Blood Volume (CBV). The response overcompensates for demand, such that the concentration of oxy- compared to deoxy-hemoglobin in the area increases rather than decreases as one might expect. The BOLD signal, being a contrast between oxy- and deoxy-hemoglobin, varies accordingly.

While this high level understanding of the system is generally accepted, the precise form of the coupling is still a significant point of enquiry. Possibilities include gamma oscillations [52] and local field potentials [53; 54], among others. In addition are potentially numerous confounding factors not related to neural activity, such as transient blood pressure changes [55] and even caffeine intake [56]. One of the biggest points of contention is over the presence of an "initial dip" – a brief increase in deoxyhemoglobin immediately proceeding neural activity but preceding the arrival of the arterial flow.

The basic hemodynamic response is described by the Hemodynamic Response Function (HRF) [1]. This is a simple fixed function approximating the rate of change of oxyhemoglobin in the blood vessels neighbouring a burst of neuronal activity. Immediately following the burst of activity, a brief dip in oxyhemoglobin is expected as oxygen is consumed from the blood. The hemodynamic system responds with a surge of bloodflow providing a level of oxyhemoglobin which overcompensates for the amount of oxygen consumed. This peaks at about 5 seconds after the neuronal activity, returning to baseline after roughly 10 seconds. In a typical 1.5 T scanner, the BOLD signal increases only 2–5% above baseline in the event of neuronal activity [50]. The HRF may be convolved with expected neural activity to generate an approximate BOLD signal.

A more complicated model is the *balloon model* [57; 58]. This models a venous compartment as a balloon using Windkessel dynamics [59]. The state of the venous compartment is represented by its blood volume normalised to the volume at rest, $v = V/V_0$ (blood volume V, rest volume V_0), and deoxyhemoglobin (dHb) content normalised to the content at rest, $q = Q/Q_0$ (dHb content Q, rest content Q_0). The compartment receives inflow of fully oxygenated arterial blood $f_{\text{in}}(t)$, extracts oxygen from the blood, and expels partially deoxygenated blood $f_{\text{out}}(t)$. The full dynamics may be represented by the differential system:

$$\frac{dq}{dt} = \frac{1}{\tau_0}\left(f_{\text{in}}(t)\frac{E(t)}{E_0} - f_{\text{out}}(v)\frac{q}{v}\right) \tag{7.1}$$

$$\frac{dv}{dt} = \frac{1}{\tau_0}\left(f_{\text{in}}(t) - f_{\text{out}}(v)\right) \tag{7.2}$$

$$E(t) \approx 1 - (1 - E_0)^{\frac{1}{f_{\text{in}}(t)}} \tag{7.3}$$

$$f_{\text{out}}(v) \approx v^{\frac{1}{\alpha}} \tag{7.4}$$

where τ_0 and α are constants, and E_0 the oxygen extraction fraction at rest.

This base model is driven by the independent input $f_{\text{in}}(t)$. It may be further extended to couple in neural activity $z(t)$ via an abstract vasodilatory signal

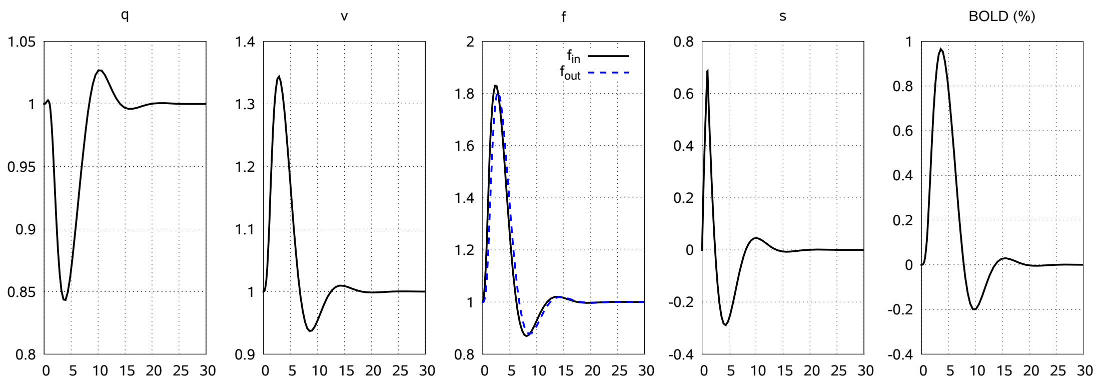


Figure 7.1: Example response of the balloon model to a 1s burst of neural activity at magnitude 1 (time on x axis, response level on y axis).

s [60]:

$$\frac{df}{dt} = s \tag{7.5}$$
$$\frac{ds}{dt} = \epsilon z(t) - \frac{s}{\tau_s} - \frac{(f-1)}{\tau_f} \tag{7.6}$$

The complete system, defined by (7.1-7.6), with $f_{\text{in}}(t) = f$, is now driven by the independent input $z(t)$. From the balloon model, the relative BOLD signal change over the baseline S at any time may be predicted using [57]:

$$\frac{\Delta S}{S} = V_0 \left[k_1(1-q) + k_2\left(1 - \frac{q}{v}\right) + k_3(1-v) \right]. \tag{7.7}$$

Figure 7.1 illustrates the system dynamics. Nominal values for constants are given in Table 7.1.

Vascular measurements in [61] may refute some of the fundamental oxygen limitation assumptions on which the balloon model is based [62]. Some additional extensions to the model are also provided by [63], adding O_2 concentration into the surrounding tissue to better correspond to optical imaging data, as well as better reproduce the controversial initial dip. Some of these issues, as well as updated parameter estimates, are incorporated into a revised balloon model [64].

Constant	Value
τ_0	.98
τ_f	1/.65
τ_s	1/.41
α	.32
ϵ	.8
V_0	.018
E_0	.4
k_1	$7E_0$
k_2	2
k_3	$2E_0 - 0.2$

Table 7.1: Nominal values for constants of the balloon model [57; 60].

7.3 Experimentation

During a basic fMRI experiment, a subject is placed in the scanner and asked to perform some task, such as the Hayling task, Stroop task or a simple finger tapping exercise. The pattern of brain activity evident while performing this task is compared to the pattern of activity when the subject is not performing the task. The objective is to find significant differences between these two brain patterns, and in doing so establish a correlation between the experimental stimulus and a particular pattern of brain activity.

Broadly, the objectives of fMRI studies encompass *activation*, *functional connectivity* and *effective connectivity*, in increasing order of complexity. Activation studies seek to identify the brain regions activated by the experimental stimulus. Functional connectivity studies seek to identify correlations between the activation of brain regions. Effective connectivity studies seek to identify causal relationships between the activation of brain regions, assessing the influence that each region exerts over others. Due to the way in which they relate remote regions of the brain together, both functional and effective connectivity studies seek to establish some sort of functional structure to the brain. The latter is of particular interest in this work.

Effective connectivity is defined as “the influence one neuronal system exerts over another” [65, p.999]. Effective connectivity studies seek to establish causal relationships in neural activity in the brain, from the scale of individual neurons to whole brain regions. Plasticity in the brain gives rise to a constant change in the strength of effective connectivity between local and remote regions, even during the course of a single experiment. This is referred to as *modulation* of connectivity.

A number of recent studies have successfully performed effective connectivity analysis on fMRI data. In [66], fMRI experiments using working memory tasks are conducted to establish differences in connectivity between a group of schizophrenic patients and a control group. In [67], similar experiments affirm Diffusion Tensor Imaging evidence of white matter damage in patients in the early stages of multiple sclerosis. Interestingly, this latter study suggests weaker effective connectivity between particular regions compared to healthy controls, as may be expected given the DTI data. Other effective connectivity studies using fMRI include [68], [69] and [70].

fMRI data is suited to effective connectivity analysis due to its spatial and temporal resolution. Electroencephalography (EEG) is commonly used for the task also. EEG does have the advantage of much higher temporal resolution, which is of particular benefit in determining causation. Its spatial resolution is significantly lower, however, so that interacting regions cannot be as well defined. fMRI provides high spatial resolution for more accurately identifying regions of interest, and adequate temporal resolution to infer some causation.

7.3.1 Experimental design

fMRI data analysis is heavily dependent on the design of the experiment under which the data is collected. Preprocessing and analysis of data depends both on the objectives of the experiment and the way in which it is performed.

Experimental designs are typically classified as either *block* or *event–related* designs. Block designs use several iterations of a rest period followed by sustained activity. Event–related designs use iterations of a brief rest followed by a short burst of activity, which is measured until the brain returns to its resting state.

A typical fMRI experiment involves multiple subjects, and possibly multiple groups. Comparisons may be made between individuals and between groups.

See [71] and [72] for more about experimental design in fMRI.

7.4 Data processing

The experimental design feeds directly into the interpretation of the data acquired during the experimental stage. For block designs, data will consist of a series of brain volumes acquired at regular intervals over the entire course of the experiment. The first few volumes are usually discarded to account for saturation effects. For event designs, volumes will be acquired at different lag times over shorter periods following the onset of each event stimulus.

The spatial and temporal resolution of the images acquired depends on the image acquisition technique and the scanner hardware. Typically EPI is used for acquiring images. Note that the individual slices within a volume will have been acquired at different time offsets, and this may be an important consideration.

Noise is a significant issue with fMRI data. Sources of noise include scanner drift resulting from gradual changes in ambient and scanner temperature, cardiac and respiratory activity of the subject, head and eye movement, and swallowing.

Processing of the data after acquisition typically proceeds through the stages of realignment, normalisation and smoothing before undergoing analysis. These are explained in great detail in [73], and summarised here.

7.4.1 Realignment

Subject movement while in the scanner is an inevitable part of MR imaging. With EPI, a single slice is acquired very quickly (30–100 ms), such that movement artifacts within individual slices can be considered negligible [51, ch.5]. As the slices constituting a whole volume are acquired sequentially, however, movement may contribute significantly to inter–slice, and more so to inter–volume variation.

Correction for this problem is referred to as *realignment*, a specific task in the more general area of *image registration*. Realignment is usually performed by estimating a rigid body transformation to align each volume in the time series with a reference volume [74]. As the volumes are three dimensional, six transformation parameters are estimated in total – three translations (in x, y and z dimensions) and three rotations (pitch, roll and yaw). Estimates are made iteratively using a least squares, mutual information or similar metric, and any appropriate optimisation algorithm.

Parameters are usually estimated for each volume by starting with an initial pass where the first volume in the time series is used as the reference. The estimated transformation is then applied to each volume and a mean volume calculated. Transformation parameters are then estimated for each original volume again in a second pass that uses the mean volume as the reference. The estimated transformation from this second pass is then applied to each original volume to produce the final, realigned time series. This two pass technique tends to give slightly better results than a single pass [74].

7.4.2 Normalisation

Normalisation of volumes to a standard template may be performed if there is particular motivation to do so. This is common in multiple subject studies to allow comparison between subjects. In such cases volumes can be warped to an arbitrary template [75], such as the average of all volumes or a de facto standard EPI template such as that provided by the popular software package SPM (Statistical Parametric Mapping) [76]. An alternative motivation for normalisation is to locate activation areas relative to well defined anatomical regions, in which case a warp to a standard template such as Talairach coordinate space or the Montreal Neurological Institute (MNI) template may be applied.

7.4.3 Smoothing

The data may be smoothed in an effort to increase its Signal to Noise Ratio (SNR). Both high– and low–pass filtering may be applied to achieve this, in both spatial and temporal dimensions. The motivation for low–pass filtering is to reduce noise in the data. The motivation for high–pass filtering is to reduce trends and remove low frequency bands with a high proportion of noise, often from physiological sources such as pulse and respiration, but also from scanner drift caused by, for example, changes in ambient temperature [77].

Spatial smoothing is usually performed with a Gaussian kernel with a Full Width at Half Maximum (FWHM) of 4–8 mm, depending on the voxel size. The FWHM measure corresponds to $2\sqrt{2 \ln 2} \approx 2.355$ standard deviations.

Detrending may also be a necessary preprocessing step (see e.g. [78]).

7.5 Data analysis

The methods employed for analysis of the processed data depend on the objectives of the experiment.

Numerous methods of analysis exist, the most popular of which is Statistical Parametric Mapping (SPM) [76], with software of the same name, suitable for activation studies. Analysis of covariance (ANCOVA) is suitable for functional connectivity studies.

The most commonly used technique for effective connectivity studies to date is Structural Equation Modelling (SEM) [2; 3], examples including [66; 67]. Dynamic Causal Modelling (DCM) [4] has been proposed as an alternative to SEM, developed specifically for brain imaging data such as fMRI. Examples of studies employing DCM are [70; 79].

We provide a more detailed review of both SEM and DCM in the following sections.

7.5.1 Structural Equation Modelling

SEM is a multivariate regression technique where each dependent variable may be a linear combination of both independent variables and other dependent variables. It supports both observed and latent variables, assuming a multivariate Gaussian distribution across all of these. Models are fit by matching the sample covariance with the covariance implied by the model parameters.

As applied to effective connectivity analysis using fMRI data, SEM is static and ignores the temporal dimension of the data. It assumes that all volumes are temporally independent of each other, and that causal influences between regions of interest are immediate and wholly evident within individual volumes. Furthermore, it does not distinguish between neural activity, hemodynamic activity and the measured BOLD signal, in essence only identifying relationships between observations. This is limiting, as interactions between regions at the neural level are not necessarily evident at the hemodynamic level [80].

SEM is a confirmatory rather than exploratory technique. It begins with a hypothesised model of the causal influences and correlations between regions of interest. From this an estimated covariance matrix $\hat{\Sigma}$ is calculated and compared to the sample covariance matrix S to assess the fit of the proposed model against actual data. Based on this, the estimated covariance matrix is updated and the process repeated.

An excellent overview of many issues related to SEM is provided by [81], and some of the issues as they particularly relate to fMRI analysis in [82]. Some of the shortcomings of SEM in its particular application to effective connectivity analysis are discussed in [83] and [84].

Despite its shortcomings, SEM is undoubtedly the most popular analysis technique applied to effective connectivity studies. The studies of [66], [67], [68] and [69], mentioned in §7.3 all employ the use of SEM.

7.5.1.1 Formulation

A number of formulations exist for SEM. We present here the Bentler–Weeks model [2], which explicitly distinguishes between dependent and independent variables [3]. It is the model of choice in the practical introduction to SEM given in [85, ch.14]. Other formulations include the Reticular Action Model (RAM) [3] and Linear Structural Relationship (LISREL) model [86; 87], used in one of the most established of SEM software packages, also named LISREL [86]. The RAM and Bentler–Weeks formulations may be reduced to LISREL, however [3].

The Bentler–Weeks model is formulated as:

$$\boldsymbol{\eta} = B\boldsymbol{\eta} + \gamma\boldsymbol{\xi} \tag{7.8}$$

where $\boldsymbol{\eta}$ is a size m vector of dependent variables, $\boldsymbol{\xi}$ a size n vector of independent variables, B an $m \times m$ matrix of regression coefficients between dependent variables, and γ an $m \times n$ matrix of regression coefficients between dependent and independent variables.

In addition, Φ is an $n \times n$ matrix of covariances between independent variables. Only independent variables have explicit covariances, other covariances are implied. Residuals may be treated as dependent or independent variables as appropriate.

Any of the entries in B, γ and Φ may be fixed.

Assuming $I - B$ is non–singular and rewriting (7.8) above, the dependent variables $\boldsymbol{\eta}$ may be expressed as a linear combination of the independent variables $\boldsymbol{\xi}$:

$$\begin{aligned} \boldsymbol{\eta} &= B\boldsymbol{\eta} + \gamma\boldsymbol{\xi} & (7.9)\\ I\boldsymbol{\eta} - B\boldsymbol{\eta} &= \gamma\boldsymbol{\xi} & (7.10)\\ (I - B)\boldsymbol{\eta} &= \gamma\boldsymbol{\xi} & (7.11)\\ \boldsymbol{\eta} &= (I - B)^{-1}\gamma\boldsymbol{\xi} & (7.12) \end{aligned}$$

Let q be the number of observed dependent variables, and r the number of observed independent variables. Let G_y be a $q \times m$ selection matrix which selects the observed dependent variables from $\boldsymbol{\eta}$, and G_x be an $r \times n$ selection matrix which selects the observed independent variables from $\boldsymbol{\xi}$.

7.5.1.2 Model estimation

The task is to estimate the unknown parameters B, γ and Φ, collectively labelled $\boldsymbol{\theta}$. The estimated matrices are denoted $\hat{B}$, $\hat{\gamma}$ and $\hat{\Phi}$, respectively. The parameters are estimated iteratively in order to minimise some objective function $Q(\Theta)$ between the covariance $\hat{\Sigma}$ implied by the paramters and the covariance of the data S.

$\hat{\Sigma}$ may be calculated in three parts as follows. Firstly the covariances between dependent variables:

$$\hat{\Sigma}_{yy} = (G_y(I - \hat{B})^{-1}\hat{\gamma})\hat{\Phi}(G_y(I - \hat{B})^{-1})\hat{\gamma})^T \,, \tag{7.13}$$

then the covariances between dependent variables and independent variables:

$$\hat{\Sigma}_{yx} = (G_y(I - \hat{B})^{-1}\hat{\gamma})\hat{\Phi}G_x^T \,, \tag{7.14}$$

and finally the covariances between independent variables:

$$\hat{\Sigma}_{xx} = G_x\hat{\Phi}G_x^T \,. \tag{7.15}$$

$Q(\Theta)$ may take any form, but a reasonable selection is to use the Maximum Likelihood function:

$$Q(\Theta) = \log|\hat{\Sigma}| - \log|S| + tr(S\hat{\Sigma}^{-1}) - (q + r) \tag{7.16}$$

Other sensible selections include least squares functions, see [85, ch.14,p.696] for an extensive list. Any optimisation or estimation technique, such as Newton methods or gradient descent, may be used to search the space of equations for a minimum to $Q(\theta)$.

SEMs are in general unidentifiable, although advocates appeal to the concept of *local identifiability* – that is, in the neighbourhood of the parameters there are no other sets of parameters which give an equivalent value for the objective function [84]. See [85, ch.14, pp.691-693] for practical issues related to model identification, and [2] for a more theoretical treatment.

Parameters need to be suitably initialised at the start of the estimation process. For unknown covariances of observed variables, it is usually reasonable to use sample covariances for initialisation.

7.5.1.3 Model validation

Once the parameters $\boldsymbol{\theta}$ which minimise $Q(\boldsymbol{\theta})$ are found, the fit of the determined model against the available data may be assessed. This is usually performed using a χ^2 test. A number of specially devised goodness-of-fit indices have also been developed for SEM, such as the Goodness of Fit Index, Adjusted Goodness of Fit Index and Parsimony Normed Fit Index. Many of these are described in [85, ch.14].

7.5.1.4 Extensions

The SEM methods described here generally fit into the category of Confirmatory Factor Analysis (CFA). Some extensions to these models are available, such as Latent Growth Models (LGMs), which can include a non-zero mean structure as well as covariance structure. Multi-level models for properly dealing with the independence assumptions of hierarchical data are also available. These are also appropriate for group studies. [88] describes many of these extensions to CFA.

7.5.2 Dynamic Causal Modelling

DCM [4] is a method developed specifically for effective connectivity analysis using neuroimaging techniques such as fMRI. Like SEM, DCM requires that a hypothesised model of connectivity be provided upfront. Unlike SEM, however, it treats this model as a representation of interactions at the underlying neural level, not simply interactions at the hemodynamic level. Neural activity is explicitly modelled, and the output of this model fed into a hemodynamic model which generates a simulated signal for comparison with the observed BOLD signal.

The experimental design is directly input to the neural level, and all neural activity is assumed to be the result of this stimulus. Neural activity feeds into a layer of hemodynamic activity, from which the BOLD signal at any time may be predicted. The neural model is a differential model allowing direct inter-regional excitation, as well as the experimental inputs both directly exciting regions and modulating the connectivity between them. Noise is introduced only at the observation level. This is the greatest limitation of DCM – while posing a dynamic and biologically explainable model, it is entirely deterministic.

While SEMs are linear, static and stochastic, DCMs are nonlinear, dynamic but deterministic. One DCM study of note is [70], although as modulatory connections are not used, the model is a simple linear fit that misses out on some of the more interesting features of DCM. In addition to [4], DCMs are described thoroughly in [73, ch.52].

7.5.2.1 Model

The neural level model is defined as:

$$\frac{d\mathbf{z}}{dt} = \left(A + \sum_{i=1}^{U} B_i u_i \right) \mathbf{z} + C\mathbf{u} \tag{7.17}$$

where $\mathbf{z}$ is a vector of neural activities in the regions of interest, $\mathbf{u}$ are experimental inputs, A is an $M \times M$ matrix of connectivity between regions, $B_1, \ldots, B_U$ are $M \times M$ matrices of input modulations between regions, and C is an $M \times U$ matrix of input efficacies.

Note the two distinct ways in which a particular input u_i can affect neural activity. Firstly, through the matrix C, it may influence activity in the regions $\mathbf{z}$ directly in a linear fashion. Secondly, through its associated matrix B_i, it may influence the strength of the connections between regions in a linear fashion. This influence may be at different rates for different regions, and in net constitutes a bilinearity. Only inputs are able to modulate the strength of connections in this fashion, the activity in the neural regions $\mathbf{z}$ is unable to influence their own or others' connections.

If no inputs are specified, the model reduces to a simple linear autoregressive model, as in the case of [70]:

$$\frac{d\mathbf{z}}{dt} = A\mathbf{z} \tag{7.18}$$

Neural activity in the regions of interest, represented by $\mathbf{z}$ in (7.17) is fed directly into the balloon model to produce BOLD signal estimates $\mathbf{y}$. Gaussian noise is added to each predicted measurement.

7.5.2.2 Model estimation

The parameters of a DCM are estimated using Expectation-Maximisation (EM) [4], with priors constraining the space of possible solutions.

7.6 Summary

This chapter has provided a brief introduction to fMRI in order to better motivate the real world applications of this work. It has introduced image acquisition, given a flavour of the data and common preprocessing steps, as well as reviewing the current state of the art in data analysis techniques.

Chapter 8

Applications

In this chapter we take the methodology developed throughout Part I and apply it to real world problems in fMRI research. We begin by constructing a model of the biophysical processes involved in §8.2, combining input (§8.2.1), neural (§8.2.2) and hemodynamic (§8.2.3) activity into a single continuous-time dynamical system based on the balloon model and DCM. We extend this by carefully introducing noise to account for model uncertainty and intrinsic stochasticity. A measurement model (§8.2.4) is then used to couple in noisy observations of the BOLD signal acquired through fMRI.

In conjunction with the kernel smoothers developed in Part I, the model is employed for a number of problems, notably to deconvolve the hemodynamic response from neural activity in §8.5 and to effective connectivity studies in §8.6.

8.1 Motivation

Temporal analysis of fMRI is significantly confounded by the fact that it does not measure neural activity directly, but instead does so via hemodynamic activity, which applies a form of temporal smoothing. For studies of higher level patterns of activity, such as effective connectivity [65], it becomes necessary to strip away the hemodynamic activity to reveal the underlying neural interactions. In the first instance, this is because interactions between regions at the neural level are not necessarily evident at the hemodynamic level [80]. In the second, analyses increasingly benefit from the temporal quality of the data, and the hemodynamic response is itself a form of temporal blurring.

Unfortunately, the relationship between neural activity, hemodynamic activity and the BOLD signal is still not thoroughly understood. There is, of course, the argument that a biological understanding and associated biophysical models are entirely unnecessary. Empirically derived curves of the response are available, such as a canonical hemodynamic response function that may be convolved with expected neural acitivyt, as in SPM [76]. Generative models are attractive for two reasons, however. Firstly, the hemodynamic response is known to vary across regions of the brain. Generative models provide parameters which allow the response to be tuned in meaningful and constrained ways across regions rather than arbitrarily in the case of a simple response curve. Secondly, with multimodal studies now an active area of research, generative models may provide some sort of interface for integrating data from multiple modes into the model estimation or validation procedures.

We have therefore chosen to limit ourselves to generative models of the hemodynamic response. The most established of these is the Balloon model [57]. The balloon model represents hemodynamic activity in one region only, whereas we are interested in interactions across many regions. This gives some idea of the magnitude of the models we are dealing with – large differential systems of neural and hemodynamic variables interrelated at various levels. Into all this we add stochasticity, reflecting the inherent noise and uncertainty in both the biological processes being modelled and the MRI machinery used to measure them. This has naturally led to considering these models in the more general framework of SDEs.

8.2 Model

We define a model of the neural and hemodynamic interactions between M regions of interest in the brain. A region consists of neural tissue and a venous compartment. The state $\mathbf{x}_i(t)$ of region i at time t is given by:

$$\mathbf{x}_i(t) = \begin{cases} f_i(t) & \text{normalised blood flow into the venous compartment} \\ s_i(t) & \text{vasodilatory signal} \\ q_i(t) & \text{normalised dHb content of the venous compartment} \\ v_i(t) & \text{normalised blood volume of the venous compartment} \\ z_i(t) & \text{neural activity} \end{cases}$$

The complete state at time t is given by $\mathbf{x}(t) = (\mathbf{x}_1(t)^T, \ldots, \mathbf{x}_M(t)^T)^T$.

Interactions between regions are modelled in four parts – the input model, the neural model, the hemodynamic model and the measurement model.

8.2.1 Input model

The input model represents the stimulus associated with the experimental task during an fMRI session. In general this is a function $\mathbf{u}(t)$, with U denoting its dimensionality. For a simple block design paradigm a one-dimensional box-car function is adequate.

8.2.2 Neural model

Neural interactions between the regions are given by:

$$d\mathbf{z} = \left[\left(A + \sum_{i=1}^{U} B_i u_i\right)\mathbf{z} + C\mathbf{u} + \mathbf{d}\right] dt + \Sigma_{\mathbf{z}}\, d\mathbf{W} \tag{8.1}$$

where $d\mathbf{W}$ is the M-dimensional standard (zero mean, unit variance) Wiener process, A an $M \times M$ matrix of efficacies between regions, $B_1, \ldots, B_U$ matrices of input modulation upon neural efficacies, C an $M \times U$ matrix of efficacies between inputs and regions and $\Sigma_{\mathbf{z}}$ an $M \times M$ diagonal diffusion matrix with diagonal $\boldsymbol{\sigma}_z$.

8.2.3 Hemodynamic model

Within each region, the variables f_i, s_i, q_i, v_i and z_i interact according to a stochastic extension of the balloon model. It is assumed that regions are sufficiently separate that their hemodynamic activity is independent given neural activity [62].

Stochasticity is carefully introduced in line with the intuition of the model. On top of the noise already incorporated into the neural model, blood flow in and out of each venous compartment is considered stochastic. This is important given that the coupling between neural activity and blood flow is not well understood, and so this stochasticity can be used to account for this uncertainty. Inflow is diffused with parameter σ_f. Outflow is more difficult as it is not a state variable. Given its dependence on blood volume v, however, we introduce a diffusion with parameter σ_v to blood volume. We introduce further stochasticity to blood oxygenation, diffusing with σ_q, permitting some nondeterministic decoupling of the metabolic rate of oxygen, potentially important in light of some studies [61; 64].

The balloon model will not behave sensibly in the case that f_i, q_i or v_i is negative. The deterministic model will never violate these constraints from any sensible initial state, although the stochastic extension may. One solution is to introduce noise into the log space of f_i, q_i and v_i, redefining the derivatives relative to $\ln f_i$, $\ln q_i$ and $\ln v_i$ via the simple identity $\frac{d\ln x}{dt} = \frac{dx/dt}{x}$. While workable, this introduces additional divisions and exponentials that, at the level of loop nesting where these calculations are performed, become a significant computational burden. We instead find multiplicative noise to be effective, essentially reducing noise as these variables approach 0 to reduce the likelihood of them ever crossing the threshold.

Dropping the subscript i for clarity, the stochastic balloon model becomes (c.f. 7.1-7.6).

$$\begin{aligned}
df &= s\,dt + f\sigma_f\,dW && (8.2)\\
ds &= \left(\epsilon z - \frac{s}{\tau_s} - \frac{f-1}{\tau_f}\right) dt && (8.3)\\
dq &= \frac{1}{\tau_0}\left(f\frac{1-(1-E_0)^{\frac{1}{f}}}{E_0} - v^{\frac{1}{\alpha}-1}q\right) dt + q\sigma_q\,dW && (8.4)\\
dv &= \frac{1}{\tau_0}(f - v^{\frac{1}{\alpha}})\,dt + v\sigma_v\,dW\,. && (8.5)
\end{aligned}$$

In the extreme, floating point arithmetic may result in any of these variables becoming zero, introducing singularities. We correct for these singularities using the limits:

$$dq = \begin{cases} \frac{1}{\tau_0}\left(f\frac{1-(1-E_0)^{\frac{1}{f}}}{E_0} - v^{\frac{1}{\alpha}-1}q\right) dt + q\sigma_q\, dW & \text{when } v \neq 0 \text{ and } f \neq 0\,, \\ \left(-\frac{1}{\tau_0}v^{\frac{1}{\alpha}-1}q\right) dt + q\sigma_q\, dW & \text{when } v \neq 0 \text{ and } f = 0\,, \\ \frac{1}{\tau_0}\left(f\frac{1-(1-E_0)^{\frac{1}{f}}}{E_0}\right) dt + q\sigma_q\, dW & \text{when } v = 0 \text{ and } f \neq 0\,, \\ q\sigma_q\, dW & \text{when } v = 0 \text{ and } f = 0\,, \end{cases} \tag{8.6}$$

and,

$$dv = \begin{cases} \frac{1}{\tau_0}(f - v^{\frac{1}{\alpha}})\, dt + v\sigma_v\, dW & \text{when } v \neq 0 \text{ and } f \neq 0\,, \\ -\frac{v^{\frac{1}{\alpha}}}{\tau_0}\, dt + v\sigma_v\, dW & \text{when } v \neq 0 \text{ and } f = 0\,, \\ \frac{f}{\tau_0}\, dt & \text{when } v = 0 \text{ and } f \neq 0\,, \\ 0 & \text{when } v = 0 \text{ and } f = 0\,. \end{cases} \tag{8.7}$$

All of this combined emphasises the most significant uncertainty in the hemodynamic model itself – the coupling between neural activity and induced blood flow – while also accounting for the noise that we expect in a biological system such as this. We find this more controlled introduction of correlated noise more appealing, and more consistent with the neuroscientific theory, than simply slapping an independent Wiener process onto each component of the model.

8.2.4 Measurement model

The relative BOLD signal change at any time for a particular region is given by (c.f. 7.7)[1]:

$$\Delta y = V_0\left[k_1(1-q) + k_2\left(1 - \frac{q}{v}\right) + k_3(1-v)\right]. \tag{8.8}$$

This may be converted to an absolute measurement $\hat{y}$ for comparison with actual measurements by introducing a baseline signal b_i for each region and an independent noise source $\xi \sim \mathcal{N}(0,1)$:

$$\hat{y} = b(1 + \Delta y) + \sigma_y \xi\,. \tag{8.9}$$

The model is completely defined by (8.1-8.9).

[1]Again, subscript is have been eliminated for clarity

8.2.5 Contrast to existing models

As proposed, the model is close to that of DCM. The main innovation is incorporating noise at all levels of the system, making this a truly stochastic model compared to the determinism of DCM. This is important, particularly in light of the significant knowledge gap in the understanding of the processes connecting neural and hemodynamic activity in the brain. DCM, on the other hand, introduces noise only at the observation level, and consequently has a much simplified noise structure that may require additional preprocessing such as whitening to be effective.

Additional superficial differences are the introduction of the constant terms $\mathbf{d}$ in the neural model (8.1) and BOLD baselines $\mathbf{b}$ in the measurement model (8.9). These replace the standardisation prepreprocessing usually performed on data before the application of DCM.

Like DCM, the model proposed has numerous appealing advantages over SEM. In particular, it incorporates a biophysical model of hemodynamic activity to infer interactions between regions at the level of latent neural activity. SEM, on the other hand, detects interactions at the observation level only, and is confounded by convolution of the hemodynamic response. Furthermore, the model, as proposed, allows variation of the hemodynamic response across regions by adjusting the biophysical parameters such as ϵ and α, and rate constants τ_0, τ_f and τ_s.

8.2.6 Prior

The prior for the system is designed to ensure stationarity, as is observed and expected from domain knowledge. Observe that the model is constructed hierarchically with all state variables ultimately dependent on the behaviour of $\mathbf{z}$ in the neural model. Consequently, by ensuring stationarity of the neural model, stationarity of the entire system is ensured.

For the particular implementation of the model used in this work, we have a single box-car input function $u(t)$ taking value 1 or 0 according to the presence or absence of the experimental stimulus, respectively. While the second term over B_i matrices in (8.1) is bilinear over arbitrary input functions, in these constrained circumstances it simply represents a switching factor between two different linear configurations – one during experimental stimulus and one during rest. We may therefore consider the neural model to be linear.

For this single input scenario, we bring to bear the theory of discrete-time linear dynamical systems via an Euler–Maruyama discretisation of (8.1) over a time step of 1, giving the autoregressive:

$$\begin{aligned}\mathbf{x}_{n+1} &= \mathbf{x}_n + (A + Bu)\mathbf{x}_n + C\mathbf{u} + \mathbf{d} + \Sigma_{\mathbf{z}}\boldsymbol{\xi} && (8.10)\\ &= (I + A + Bu)\mathbf{x}_n + C\mathbf{u} + \mathbf{d} + \Sigma_{\mathbf{z}}\boldsymbol{\xi}\,, && (8.11)\end{aligned}$$

where $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, I)$. Note that as the system is linear, the discretisation is exact and the choice of time step arbitrary.

For $u(t) = 1$, let $\Phi = I + A + Bu = I + A + B$. In order for the system to be stable, $|\lambda_i|$ must be ≤ 1 for all eigenvalues λ_i of Φ. A more conservative but facilitative constraint is to ensure that $\sigma_i \leq 1$ for all singular values σ_i of Φ, on the basis that $\sigma_{\min} \leq |\lambda_i| \leq \sigma_{\max}$ for the smallest and largest singular values, $\sigma_{\min}$ and $\sigma_{\max}$, and all λ_i.

The identity matrix I is stable, and the set of all stable matrices convex. Consequently, by generating stable matrices A and B, the stability of Φ is assured.

Any stable matrix Φ may be generated using Algorithm 8.1:

Algorithm 8.1 *Begin by producing matrices X_U and X_V of random elements uniformly distributed between 0 and 1, with linearly independent columns. Decompose each into $X_U = UR_U$ and $X_V = VR_V$ using QR decomposition. U and V are now orthonormal matrices.*

Generate a random diagonal matrix D of singular values ≤ 1 and reconstruct Φ using its singular value decomposition $\Phi = UDV^T$.

Note that stability is also assured for the case of $u(t) = 0$, where $\Phi = I + A + Bu = I + A$. The method generalises straightforwardly to multiple box-car inputs.

The above provides prior parameter samples for the neural model only. Other parameters are sampled from a Gaussian distribution of independent components given by Tables 8.1-8.3. Together, these constitute the prior over parameters

$p(\boldsymbol{\theta})$. Note that in this work, hemodynamic and measurement model parameters are fixed for all experiments, although these could be selectively freed to allow variation in the hemodynamic response across regions (see [89] for discussion).

Conditioned on such a parameter sample, the system is stationary. The full prior is given by $p_s(\mathbf{x}_0 \mid \boldsymbol{\theta})p(\boldsymbol{\theta})$. In order to obtain a sample from this, for each parameter sample $\boldsymbol{\theta}_0^{(i)}$, a starting state is drawn from the multivariate Gaussian of independent components given by Table 8.4. The system is then simulated with fixed input $u(0)$ for some burn time until the stationary distribution is reached. The state at this point constitutes a sample from $p_s(\mathbf{x}_0 \mid \boldsymbol{\theta} = \boldsymbol{\theta}_o^{(i)})$, and $\mathbf{x}_0^{(i)}$ and $\boldsymbol{\theta}_0^{(i)}$ together a complete sample from the prior.

8.3 Numerical scheme

We perform an initial study into the model that is similar to that for the double well model (§4.4.2), comparing various numerical schemes. The same caveats apply as those noted at the end of §4.4.2, with the higher dimensionality in this case making it difficult to adjust for any perceived autocorrelation, equilibrium or other qualitative differences across results.

In contrast to the double well model, where each numerical scheme was made to perform a fixed number of steps, for the fMRI model each scheme is made to progress a fixed length of time. This is important given the switching input of the model – the most difficult part of the integration – ensuring that each numerical scheme progresses through the same number of switches.

We set error bounds of $\delta_{\text{abs}} = 10^{-3}$ and $\delta_{\text{rel}} = 10^{-2}$ for all schemes.

For a single region, Figure 8.1 plots a histogram of step sizes for each numerical scheme. A fixed time step of .025 is used for the EM(1)FIXED scheme on the basis of these results. A performance comparison of the schemes is given in Table 8.5. On the basis of these results the RK(4) scheme is chosen.

Parameter (θ)	μ_θ	σ_θ
ϵ	.8	16
E_0	.4	.0024
τ_0	1.02	.1
τ_f^{-1}	.41	.002
τ_s^{-1}	.65	.015
α	.32	.0013

Table 8.1: Gaussian prior over hemodynamic model parameters (based on [4]).

Parameter (θ)	μ_θ	σ_θ
σ_z	.1	.01
σ_f	.01	.01
σ_v	.01	.01
σ_y	2	.5

Table 8.2: Gaussian prior over noise parameters.

Parameter (θ)	μ_θ	σ_θ
b	190	5
V_0	.018	.01
k_1	.28	1
k_2	2	1
k_3	.4	1

Table 8.3: Gaussian prior over measurement model parameters (based on [57]).

Variable (x)	μ_x	σ_x
z	0	.1
f	1	.1
s	0	.1
q	1	.1
v	1	.1

Table 8.4: Gaussian prior over state variables.

Scheme	Runtime	Steps	$\frac{\text{Progression}}{\text{Runtime}}$	$\frac{\text{Progression}}{\text{Steps}}$
EM(1)FIXED	2.88	1660000	14236	.0247
EM(1)	3.70	559057	11081	.0733
RK(2)	6.87	662253	5968	.0619
RK(4)	**1.45**	74639	**28276**	.5493
RK(8)	3.27	**61174**	12538	**.6702**
EM(1)IMP	7.92	782229	5177	.0524
RK(2)IMP	8.76	792096	4680	.0518
RK(4)IMP	14.89	826810	2754	.0496

Table 8.5: Comparison of numerical schemes for the fMRI deconvolution model. Results in all cases are based on a time progression of 4.1×10^4s.

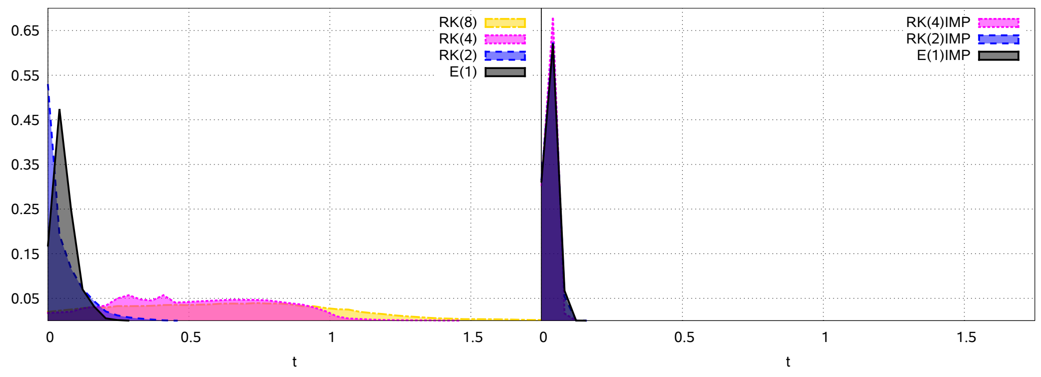


Figure 8.1: Histogram over step sizes for numerical schemes.

8.4 Data

We use both a simulated and a real data set for our experiments. The simulated data set is important for validation of methods as it has a known ground truth.

For a given TR, number of observations T and number of regions M, a simulated data set may be constructed simply by drawing a single sample from the prior and simulating for $T \times TR$ seconds. Given that the prior ensures stability, any such sample can be propagated indefinitely to obtain a data set of the desired length. Different random number seeds produce different data sets. Fixing hemodynamic, noise and measurement model parameters to their prior means as in Tables 8.1-8.3, the diagonal elements of A to -1, and starting from a single sample drawn from the remaining prior, we generate both single region and 4 region sequences using a TR of 2.05s with 200 observations. We refer to this as the *Sim* data set.

Real experimental data is used in the form of the *Session Effects*, or *SessFX* set. This was collected during a simple finger tapping exercise. Using a Siemens Vision at 2T with a TR of 4.1s, a healthy 23-year-old right-handed male was scanned on 33 separate days over a period of two months. In each session, 80 whole volumes were taken, with the first two discarded to account for T1 saturation effects. The experimental paradigm consists of alternating $6TR$ blocks of rest and tapping of the right index finger at 1.5Hz, where tapping frequency is provided by a constant audio cue, present during both rest and tapping phases.

All scans across all sessions were then realigned using SPM5 [76] and a two-level random effects analysis performed, from which 13 voxels were selected to represent regions of interest. No smoothing or normalisation was applied to the data. Of the 13 voxels, one is selected for deconvolution experiments, located in right M1, and four are selected for use in effective connectivity experiments, located in the left posterior parietal cortex, left M1, left S1 and left premotor cortex. A single session is used to provide the measurements for these experiments. A more driven enquiry would use all sessions – our motivation here is to demonstrate the effectiveness of the proposed methods rather than provide biological insight.

It is worth noting that the preprocessing steps typically performed on fMRI data sets are not particularly conducive to temporally sensitive analyses such as those performed here. In particular, whole volumes are generally assumed to have been acquired at the same time. Perhaps worse, if not assumed to be so, interpolation and resampling may be performed to match slice timings. Our preprocessing of the SessFX data set has been limited in order to preserve as much temporal information as possible.

8.5 Deconvolution

The fMRI deconvolution problem is to infer the posterior distribution over neural activity given a particular observation of the BOLD signal. For this task the model is defined over a single region only, so that $M = 1$.

For the deconvolution problem over the Sim data set, we fix all hemodynamic, noise and measurement model parameters to their prior means as in Tables 8.1-8.3, and A to -1 for regularisation. Other parameters – B, C and d – are allowed to vary freely.

For the SessFX data set we do the same, only this time fixing b to the mean of all measurements in the session. We find it particularly useful to fix A and b when working with real data so as to limit the number of parameterisations which are equally favoured given the significant noise in the data.

Method	Sim	SessFX
Auxiliary + regularised filter	14.00	15.26
Kernel forward-backward smoother	17.39	11.48
Kernel two-filter smoother	30.95	17.97

Table 8.6: Runtime performance results for deconvolution problem with both data sets. All runs are over 4 nodes with $P = 1000$ particles. Times given in wallclock seconds.

8.5.1 Results

Figures 8.2 & 8.3 provide filter results using the auxiliary + regularised filter for both the Sim and SessFX data sets. Gaussian kernels are used for the regularised particle filter with bandwidth $h = .5h_{\text{opt}}$ and no standardisation. $P = 1000$ particles are used, distributed over 4 nodes.

Figures 8.4 & 8.5 smooth these results using the kernel forward-backward smoother, and Figures 8.6 & 8.7 using the kernel two-filter smoother. In both cases a kernel bandwidth of $h = .5h_{\text{opt}}$ is used, with no standardisation.

Table 8.6 provides performance results for both data sets.

8.5.2 Discussion

In the case of the simulated data set, results are consistent with the known ground truth. We note that in this case the kernel two-filter smoother delivers a more successful smoothing than the kernel forward-backward smoother in the first block of activity between 24.6s and 49.2s.

In the case of the SessFX data, quite a clear pattern of neural activity is extracted from the very noisy BOLD signal for the single region. Close to zero activity is apparent during periods of rest, while inhibition of activity is evident during periods of stimulus. This is consistent with expectations given that the data is extracted from the right motor cortex and the experimental condition is tapping of the right index finger. Even when smoothed, some misbehaviour is apparent within the first few blocks. We observe such behaviour consistently across voxels

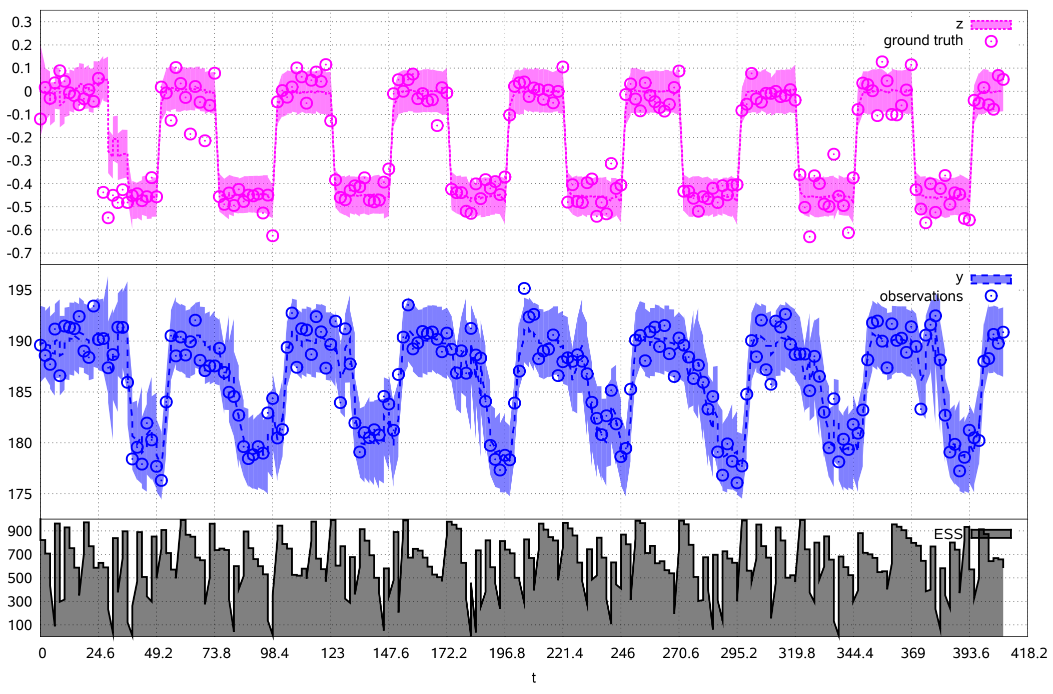


Figure 8.2: Deconvolution of a region in the Sim data set using an auxiliary + regularised particle filter with bandwidth $h = .5h_{\text{opt}}$, no standardisation and $P = 1000$ particles. Lines indicate means and shaded regions two standard deviations either side. Known ground truth of the neural activity (z) and actual observations are marked for comparison.

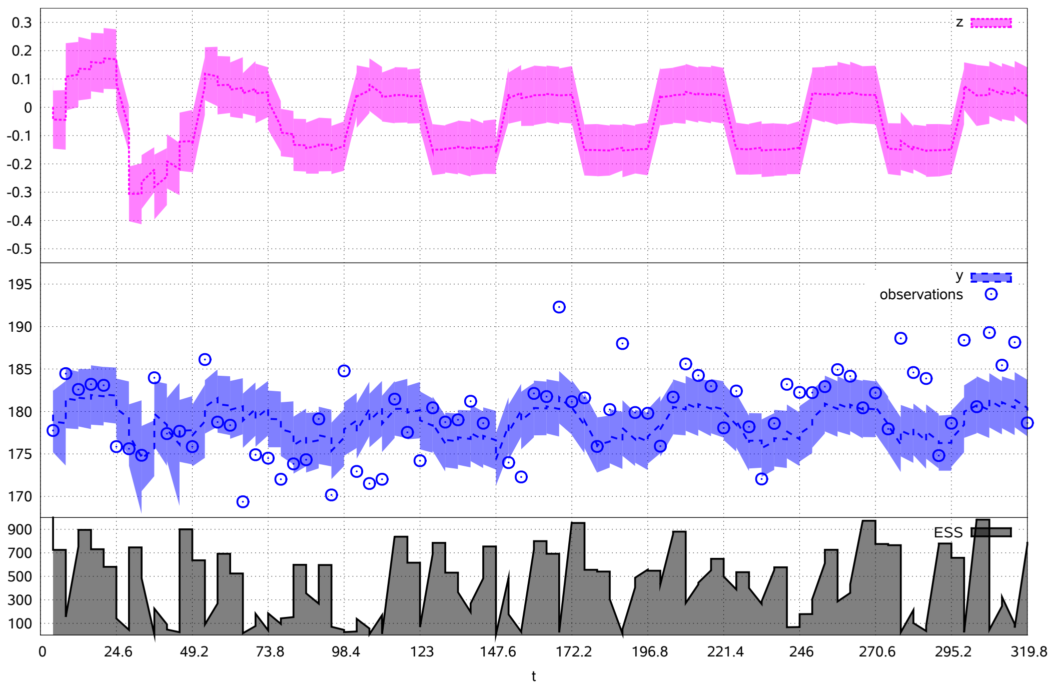


Figure 8.3: Deconvolution of the right M1 region in the SessFX data set using an auxiliary + regularised particle filter with bandwidth $h = .5h_{\text{opt}}$, no standardisation and $P = 1000$ particles. Lines indicate means and shaded regions two standard deviations either side. Actual observations are marked for comparison.

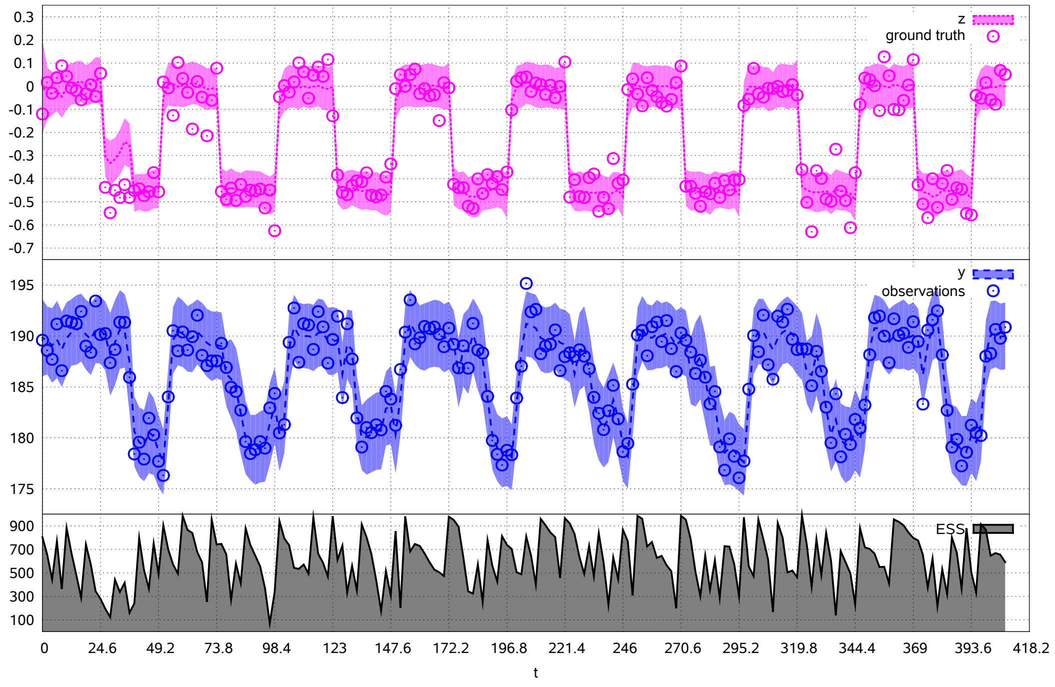


Figure 8.4: Deconvolution of a region in the Sim data set using a kernel forward-backward smoother across the results of the auxiliary + regularised particle filter. Bandwidth is set to $h = .5h_{\text{opt}}$, with no standardisation and $P = 1000$ particles. Lines indicate means and shaded regions two standard deviations either side. Known ground truth of the neural activity (z) and actual observations are marked for comparison.

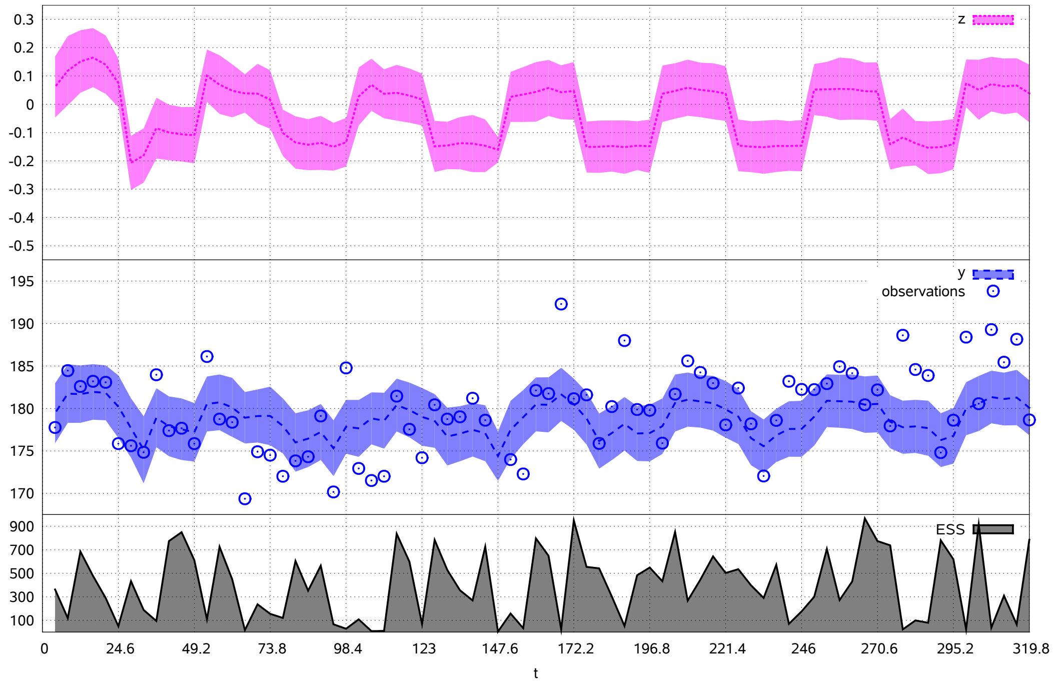


Figure 8.5: Deconvolution of the right M1 region in the SessFX data set using a kernel forward-backward smoother across the results of the auxiliary + regularised particle filter. Bandwidth is set to $h = .5h_{\text{opt}}$, with no standardisation and $P = 1000$ particles. Lines indicate means and shaded regions two standard deviations either side. Actual observations are marked for comparison.

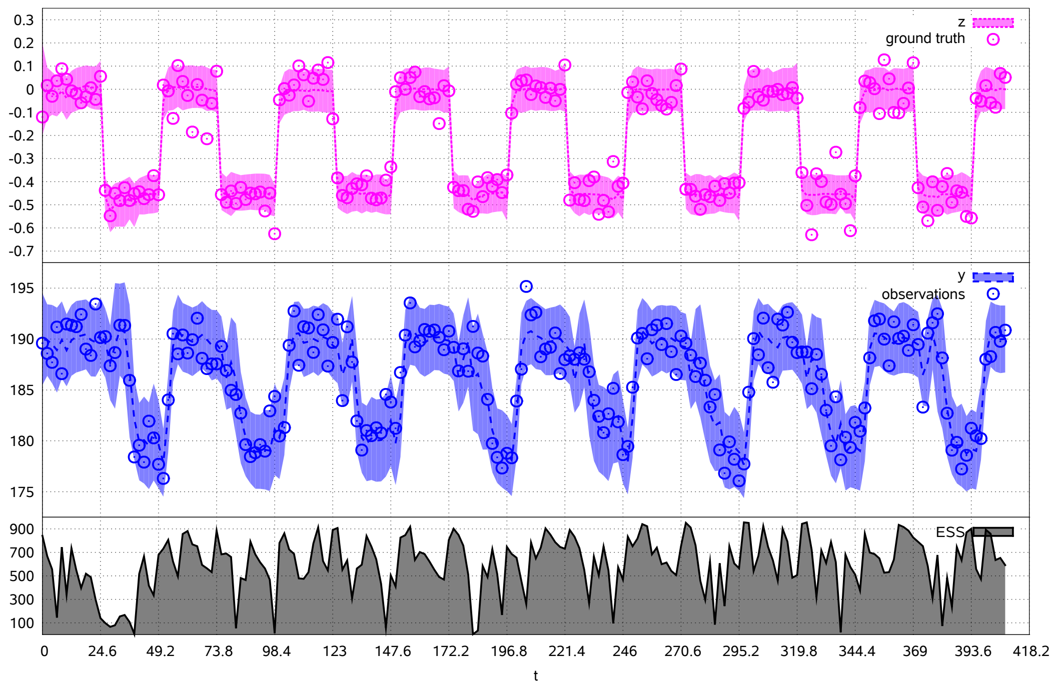


Figure 8.6: Deconvolution of a region in the Sim data set using a kernel two-filter smoother across the results of the auxiliary + regularised particle filter. Bandwidth is set to $h = .5h_{\text{opt}}$, with no standardisation and $P = 1000$ particles. Lines indicate means and shaded regions two standard deviations either side. Known ground truth of the neural activity (z) and actual observations are marked for comparison.

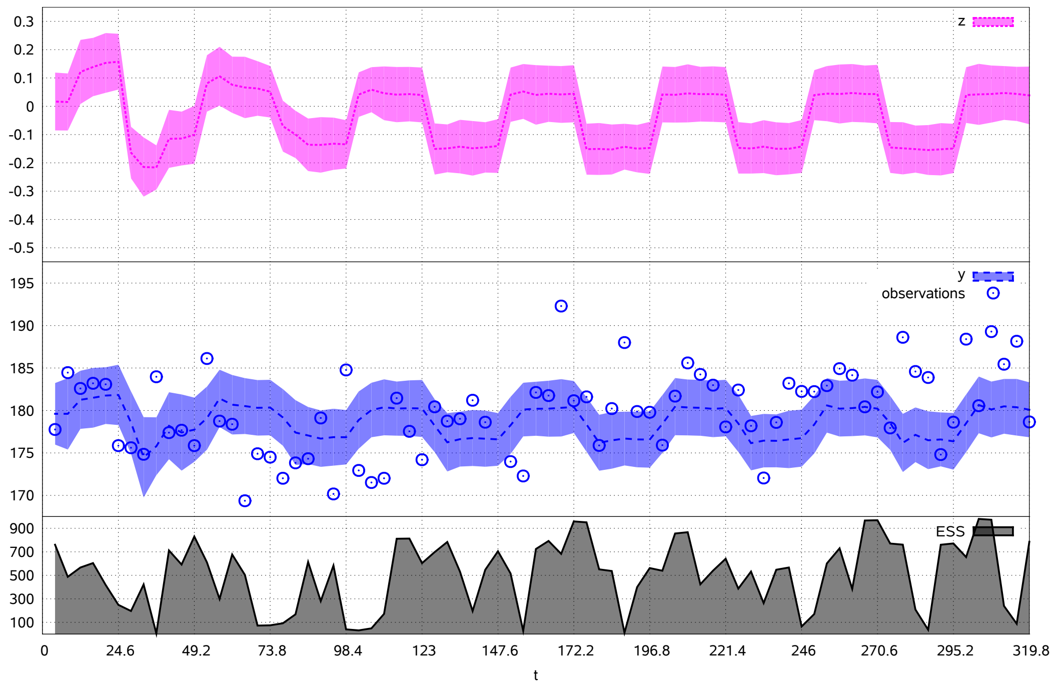


Figure 8.7: Deconvolution of a region in the SessFX data set using a kernel two-filter smoother across the results of the auxiliary + regularised particle filter. Bandwidth is set to $h = .5h_{\text{opt}}$, with no standardisation and $P = 1000$ particles. Lines indicate means and shaded regions two standard deviations either side. Actual observations are marked for comparison.

in the data set, and this may in fact represent characteristic behaviour under the experimental stimulus rather than methodological limitation, particularly given that no such behaviour is apparent with the idealised simulated data.

Unfortunately, despite the nice neural signal extracted, the fit of the model based on predicted versus actual observations is fairly poor, as evident by the number of outliers visible in Figures 8.3, 8.5 and 8.7. This is not overly surprising given the low signal-to-noise ratio of the data. Indeed, we might conversely argue that the results are encouraging given that the fit is obtained from a single session only. A second-level analysis across such results obtained for each session could potentially be used for stronger inference. We note that taking the mean across all sessions produces a much clearer signal and much better fit [46], although this is not ideal given that such aggregates may smudge the sensitive temporal information used to infer the causal relationships of interest for effective connectivity. A few other tricks can improve fit, notably increasing measurement noise, but this produces broader distributions over neural activity such that the two states are then not well defined, and is questionable practice regardless given that measurement noise is assumed known *a priori*. We leave the results in their current state as a work in progress, without such compromises.

At least one previous study has attempted to use particle filtering techniques for fMRI deconvolution [90]. This study relied on linear discretisation of similar SDEs to those proposed here, albeit without the same controlled introduction of noise. As demonstrated, our own methods can make use of faster Runge–Kutta schemes. To our knowledge no study has attempted to smooth the results of this filtering, however. On its own, a filter does not provide an accurate deconvolution of the hemodynamic response, particularly for times close to the starting point, and provides little basis for assessing model likelihood or fit.

8.6 Effective connectivity

For studies of higher level patterns of activity, such as effective connectivity [65], it becomes necessary to strip away the hemodynamic activity to reveal the underlying neural interactions. In the first instance, this is because interactions between regions at the neural level are not necessarily evident at the hemody-

Method	Sim	SessFX
Auxiliary + regularised filter	24	21
Kernel forward-backward smoother	605	281

Table 8.7: Performance results for effective connectivity problem with both data sets. All runs over 40 nodes with $P = 10^5$ particles. Times given in wallclock minutes. Smoothing times do not include that of the preceding filter.

namic level [80]. In the second, analyses increasingly benefit from the temporal qualities of the data, and the hemodynamic response itself is a form of temporal blurring.

We perform 4 region effective connectivity analyses on both the Sim and the SessFX data sets. Even with 4 regions we have a reasonable sized problem with 24 state variables and 36 free parameters. We set $P = 10^5$ and run our methods across 40 nodes of the Eddie cluster of the Edinburgh Data and Compute Facility[2].

8.6.1 Results

Figure 8.8 presents smoothed results using the kernel forward-backward smoother for underlying neural activity in the Sim data set. For model fit, Figure 8.9 presents predicted measurements against actual observations.

Figures 8.10-8.17 provide parameter estimation results for the Sim data set. Figures 8.18-8.25 provide results for the SessFX data set.

Table 8.7 provides runtime performance results.

8.6.2 Discussion

On the surface, it appears as though the filter has failed to capture the ground truth parameter values for the Sim data set, despite exhibiting good fit to both observations (Figure 8.9) and ground truth neural activity (Figure 8.8). While the estimated parameters and their ground truth represent two different dynamical systems, closer investigation reveals that there is much in common.

[2]`http://www.ecdf.ed.ac.uk/`

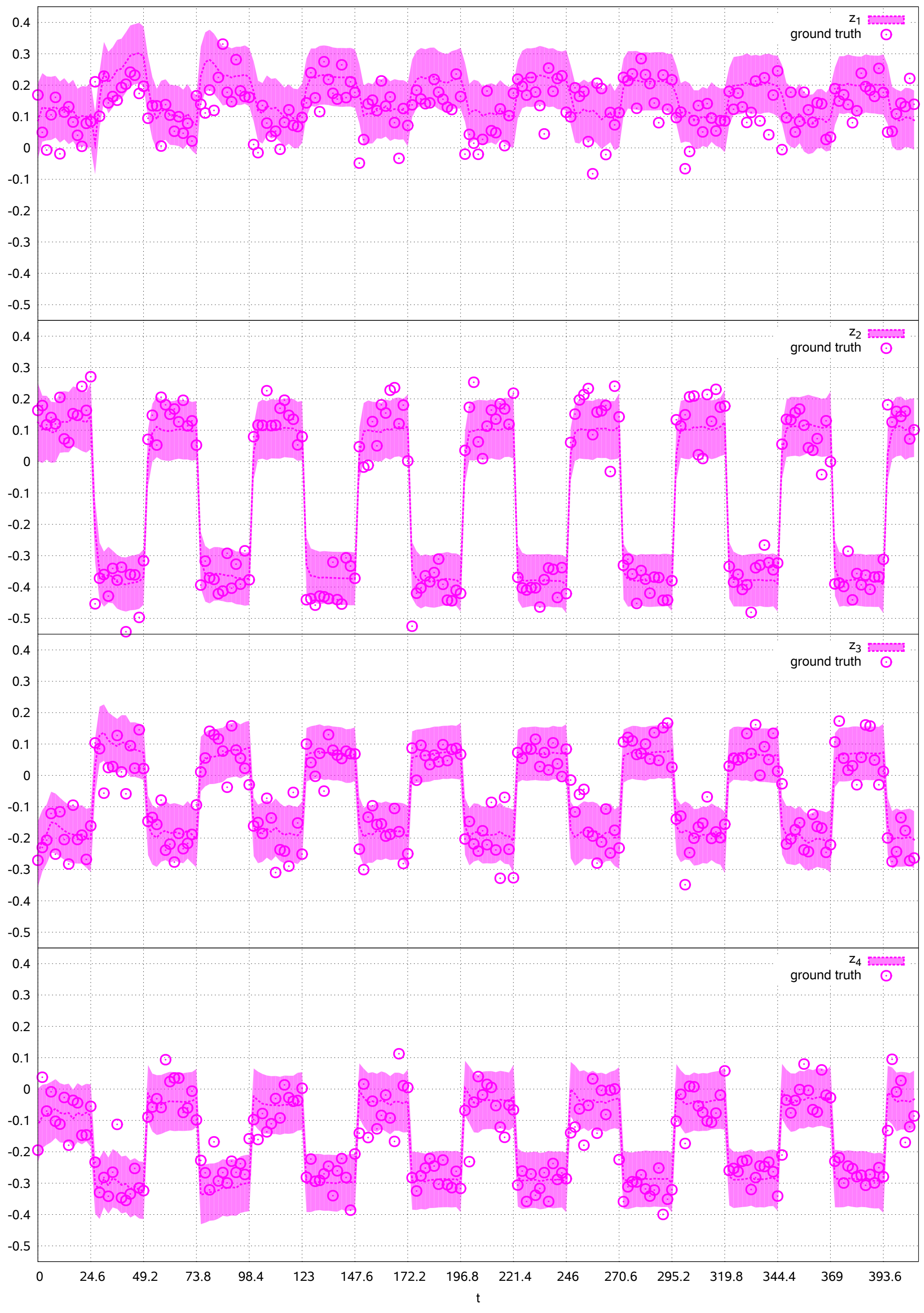


Figure 8.8: Smoothed neural activity $\mathbf{z}$ on the Sim data set for the effective connectivity problem. Lines indicate means and shaded regions two standard deviations either side. Known ground truth is marked for comparison.

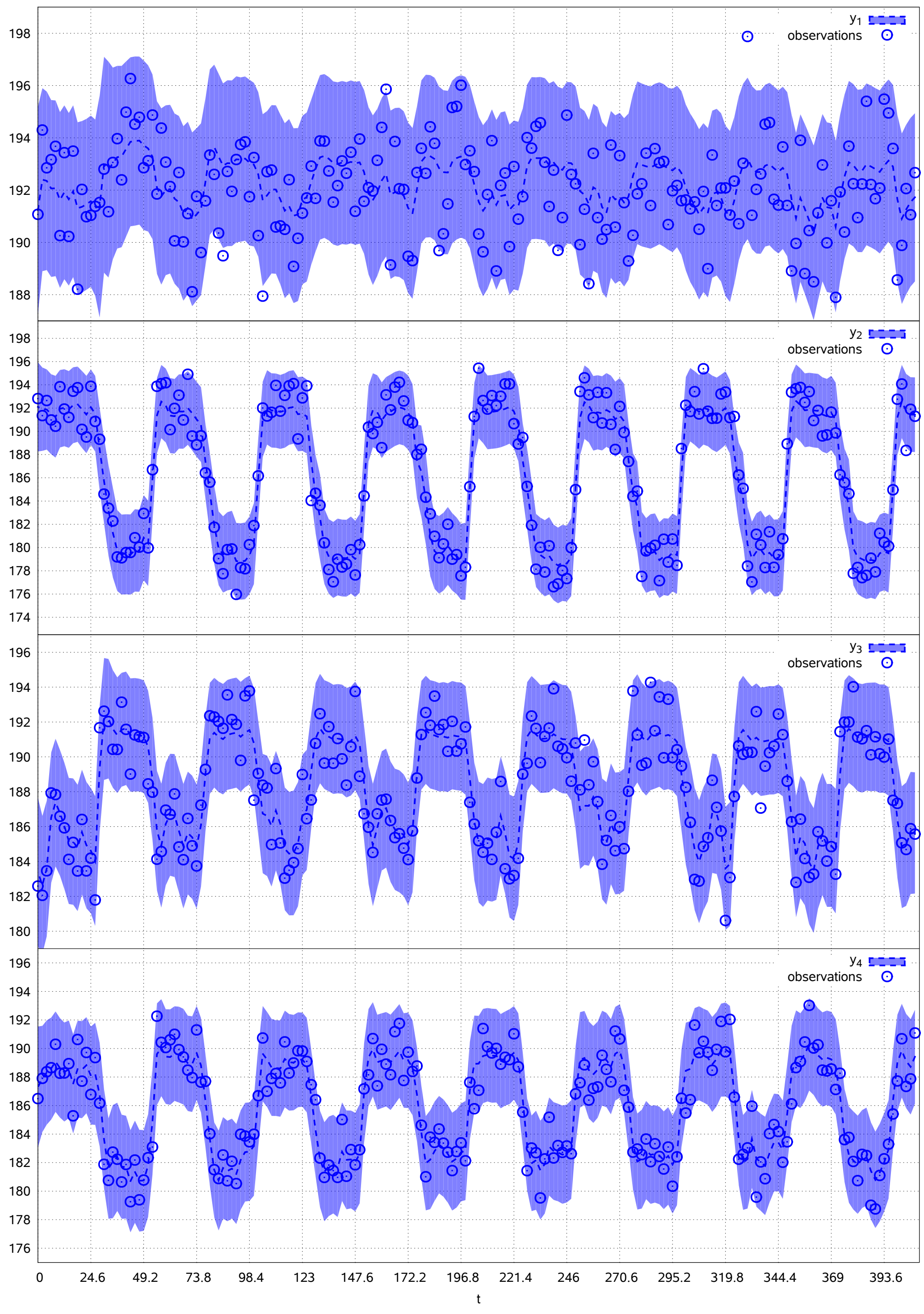


Figure 8.9: Smoothed predicted measurements against actual observations on the Sim data set for the effective connectivity problem. Lines indicate means and shaded regions two standard deviations either side. Actual observations are marked for comparison.

$$A = \begin{pmatrix} -1.00 & -.09 & .48 & -.36 \\ -.01 & -1.00 & -.20 & .01 \\ .12 & .13 & -1.00 & -.44 \\ .32 & -.24 & -.17 & -1.00 \end{pmatrix} \pm \begin{pmatrix} \text{na} & 10^{-2} & 10^{-2} & 10^{-2} \\ 10^{-1} & \text{na} & 10^{-1} & 10^{-2} \\ 10^{-1} & 10^{-2} & \text{na} & 10^{-2} \\ 10^{-1} & 10^{-1} & 10^{-2} & \text{na} \end{pmatrix}$$

Figure 8.10: Final estimate of A for Sim data set, mean plus standard deviation. Recall that the diagonal is fixed.

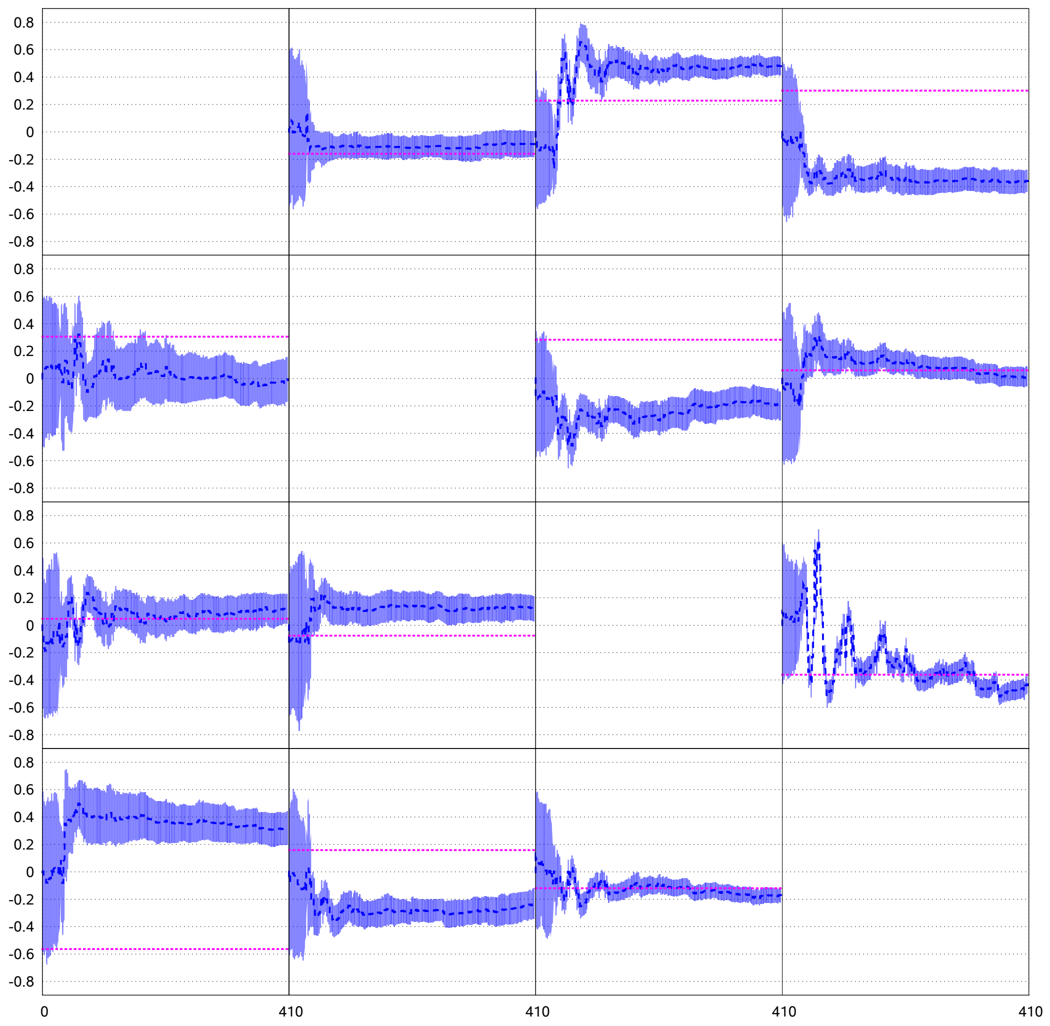


Figure 8.11: Converging estimate of A during filter for Sim data set. Horizontal dotted lines indicate known ground truth parameter values.

$$B_1 = \begin{pmatrix} -.64 & -.39 & -.46 & .07 \\ -.36 & -.86 & -.64 & -.17 \\ .53 & .28 & -1.03 & .26 \\ -.15 & .20 & .15 & -.42 \end{pmatrix} \pm \begin{pmatrix} 10^{-2} & 10^{-2} & 10^{-2} & 10^{-2} \\ 10^{-2} & 10^{-2} & 10^{-2} & 10^{-2} \\ 10^{-2} & 10^{-2} & 10^{-2} & 10^{-2} \\ 10^{-2} & 10^{-2} & 10^{-2} & 10^{-2} \end{pmatrix}$$

Figure 8.12: Final estimate of B for Sim data set, mean plus standard deviation.

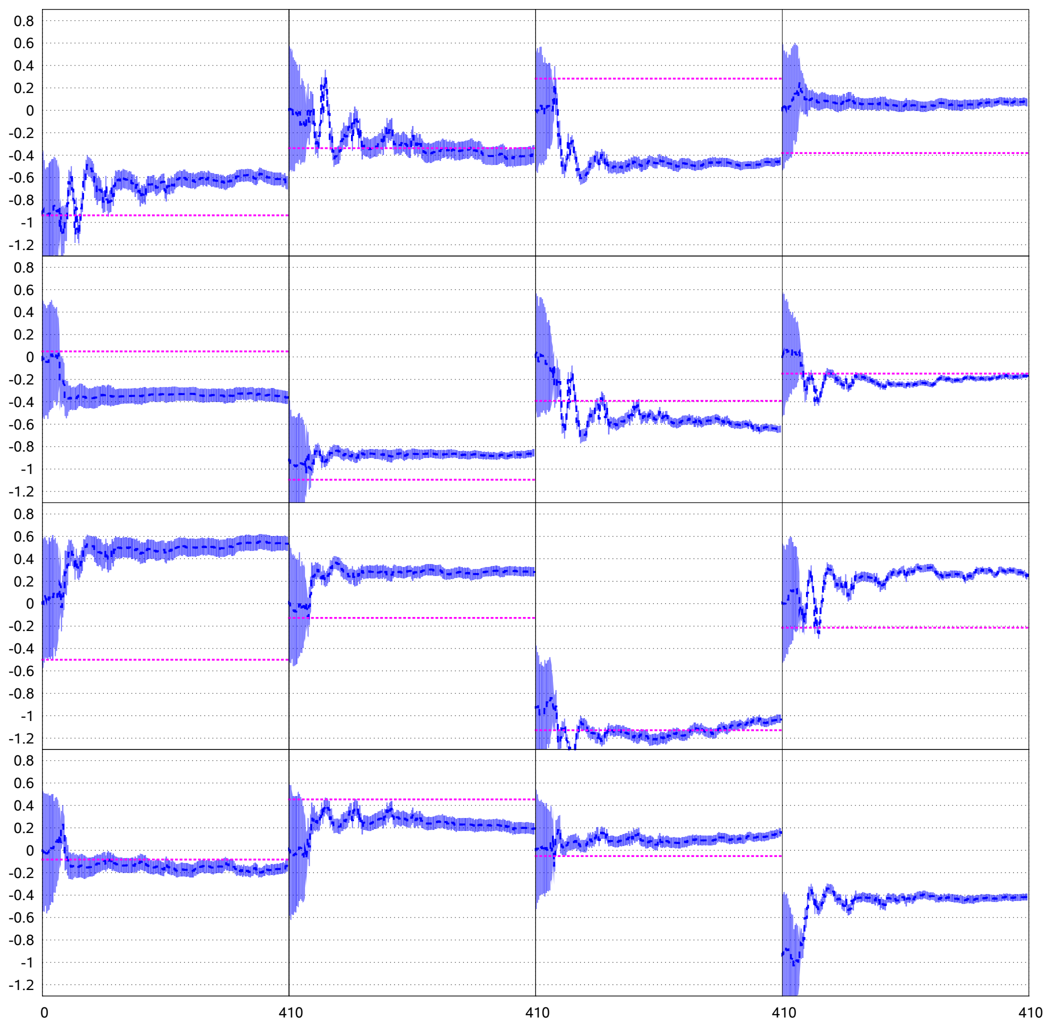


Figure 8.13: Converging estimate of B during filter for Sim data set. Horizontal dotted lines indicate known ground truth parameter values.

$$C = \begin{pmatrix} -.11 \\ -.68 \\ .35 \\ -.39 \end{pmatrix} \pm \begin{pmatrix} 10^{-2} \\ 10^{-2} \\ 10^{-2} \\ 10^{-2} \end{pmatrix}$$

Figure 8.14: Final estimate of C for Sim data set, mean plus standard deviation.

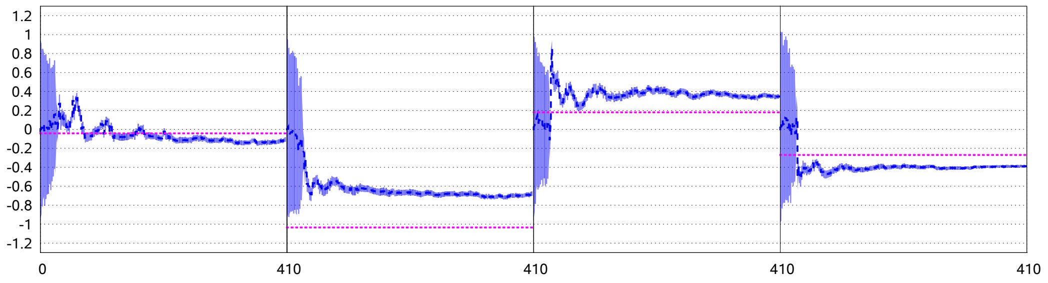


Figure 8.15: Converging estimate of C during filter for Sim data set. Horizontal dotted lines indicate known ground truth parameter values.

$$\mathbf{d} = \begin{pmatrix} .18 \\ .07 \\ -.24 \\ -.07 \end{pmatrix} \pm \begin{pmatrix} 10^{-3} \\ 10^{-3} \\ 10^{-3} \\ 10^{-3} \end{pmatrix}$$

Figure 8.16: Final estimate of $\mathbf{d}$ for Sim data set, mean plus standard deviation. Horizontal dotted lines indicate known ground truth parameter values.

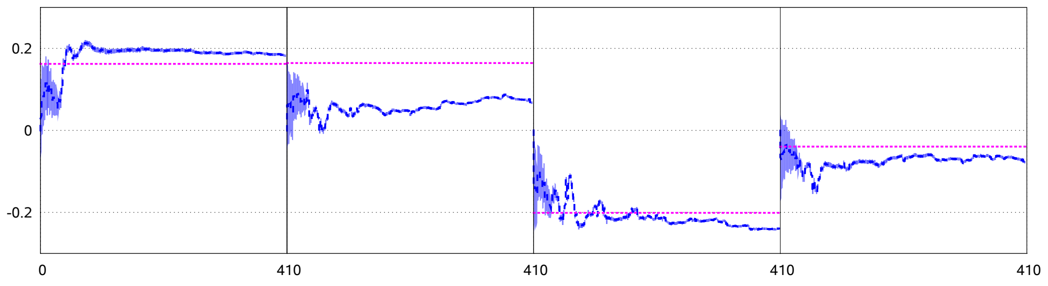


Figure 8.17: Converging estimate of $\mathbf{d}$ during filter for Sim data set. Horizontal dotted lines indicate known ground truth parameter values.

$$A = \begin{pmatrix} -1.00 & -.92 & -.12 & -.03 \\ -.03 & -1.00 & -1.10 & -.76 \\ .16 & .10 & -1.00 & -.05 \\ .41 & -.08 & -.38 & -1.00 \end{pmatrix} \pm \begin{pmatrix} \text{na} & 10^{-6} & 10^{-7} & 10^{-8} \\ 10^{-4} & \text{na} & 10^{-7} & 10^{-8} \\ 10^{-5} & 10^{-7} & \text{na} & 10^{-8} \\ 10^{-5} & 10^{-7} & 10^{-7} & \text{na} \end{pmatrix}$$

Figure 8.18: Final estimate of A for SessFX data set, mean plus standard deviation. Recall that the diagonal is fixed.

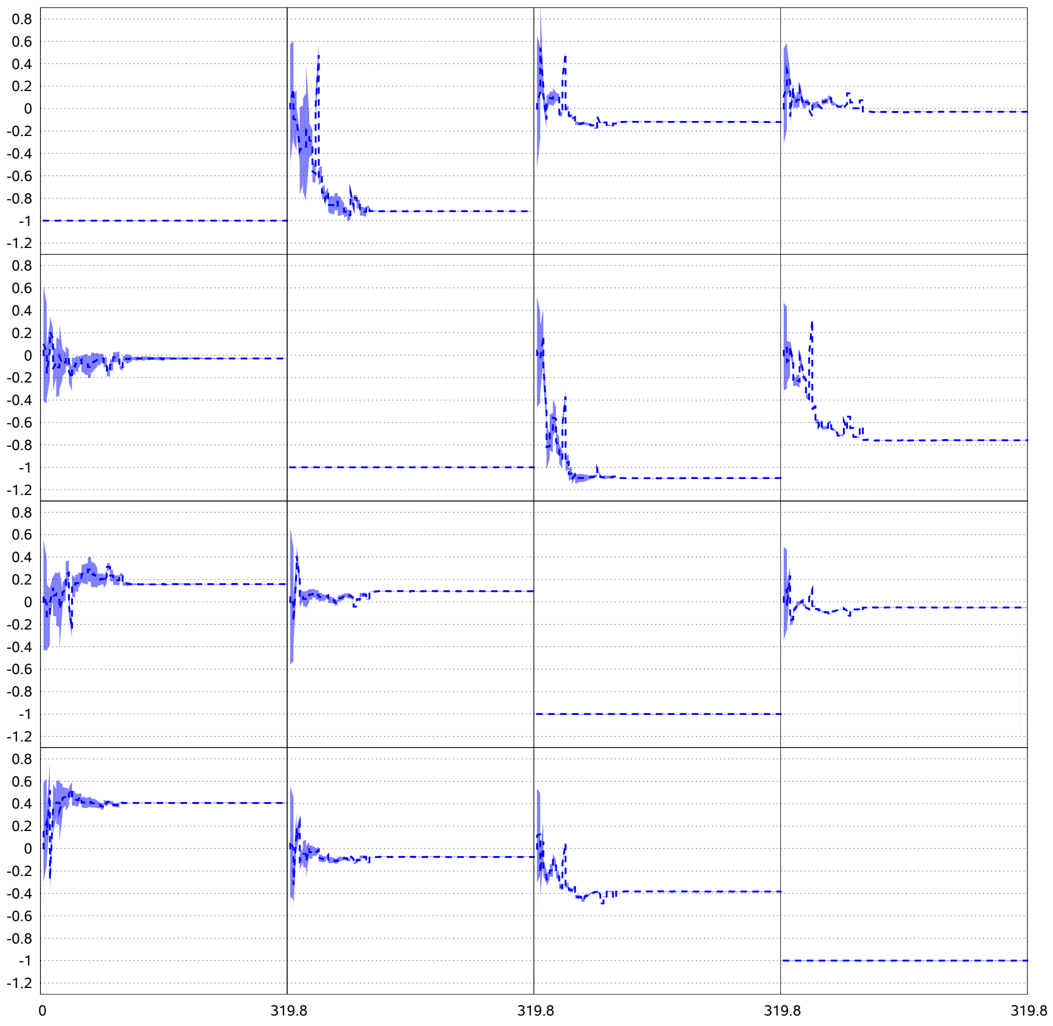


Figure 8.19: Converging estimate of A during filter for SessFX data set.

$$B_1 = \begin{pmatrix} -.14 & .34 & .12 & .16 \\ .26 & -.85 & -.07 & -.66 \\ -.89 & .11 & -.75 & .45 \\ .02 & .68 & -.49 & -1.03 \end{pmatrix} \pm \begin{pmatrix} 10^{-15} & 10^{-15} & 10^{-14} & 10^{-16} \\ 10^{-14} & 10^{-13} & 10^{-16} & 10^{-14} \\ 10^{-15} & 10^{-15} & 10^{-15} & 10^{-15} \\ 10^{-15} & 10^{-14} & 10^{-16} & 10^{-15} \end{pmatrix}$$

Figure 8.20: Final estimate of B for SessFX data set, mean plus standard deviation.

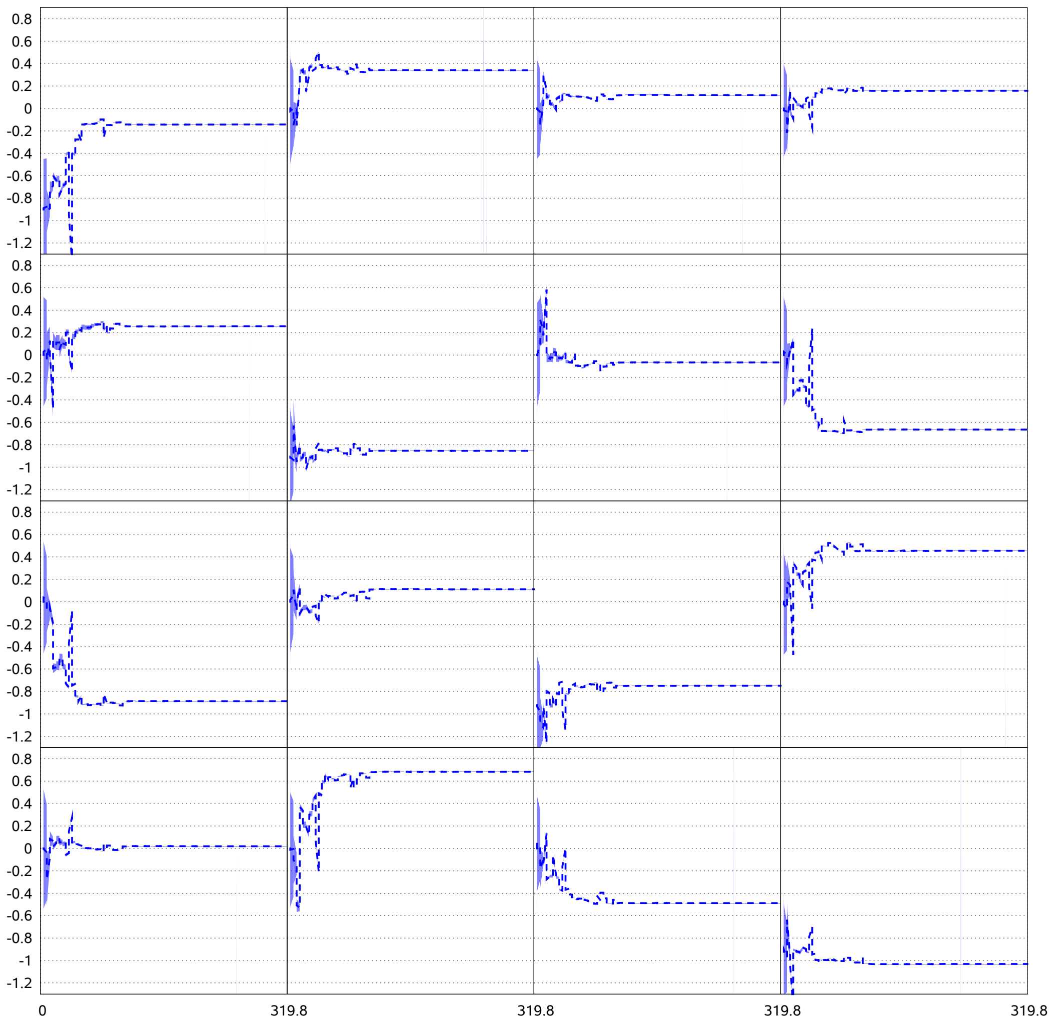


Figure 8.21: Converging estimate of B during filter for SessFX data set.

$$C = \begin{pmatrix} -.95 \\ .66 \\ -.51 \\ .37 \end{pmatrix} \pm \begin{pmatrix} 10^{-14} \\ 10^{-15} \\ 10^{-15} \\ 10^{-15} \end{pmatrix}$$

Figure 8.22: Final estimate of C for SessFX data set, mean plus standard deviation.

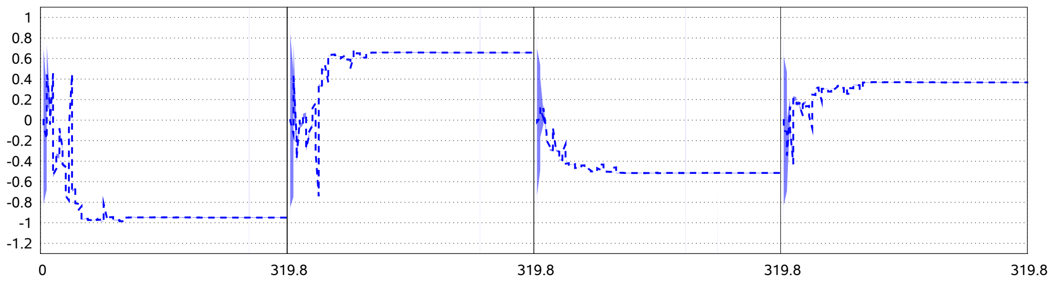


Figure 8.23: Converging estimate of C during filter for SessFX data set.

$$\mathbf{d} = \begin{pmatrix} .09 \\ .15 \\ .03 \\ .12 \end{pmatrix} \pm \begin{pmatrix} 10^{-16} \\ 10^{-17} \\ 10^{-17} \\ 10^{-16} \end{pmatrix}$$

Figure 8.24: Final estimate of $\mathbf{d}$ for SessFX data set, mean plus standard deviation.

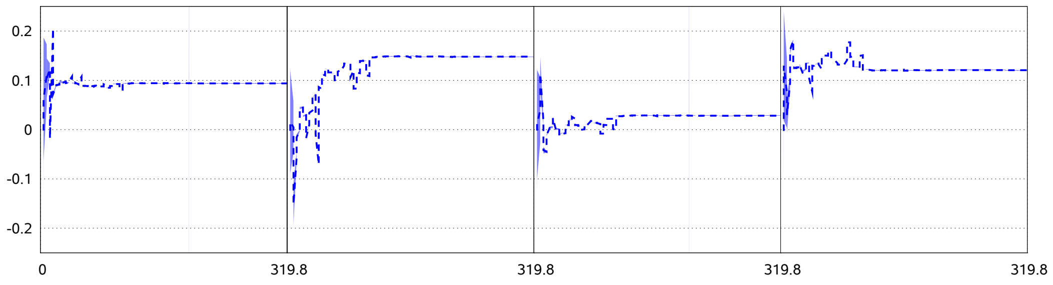


Figure 8.25: Converging estimate of $\mathbf{d}$ during filter for SessFX data set.

Consider the equilibrium state of the neural model (8.1). We approximate this by simulating a single trajectory from the model with fixed input, taking 10^5 samples at 1s intervals, and taking the maximum likelihood Gaussian fit to these points, given the linear-Gaussian nature of the neural model:

$$
= \begin{cases}
\mathcal{N}\left(\begin{matrix} .15 \\ .15 \\ -.19 \\ -.04 \end{matrix}, \begin{matrix} 1.0 \times 10^{-2} & 9.0 \times 10^{-6} & 9.0 \times 10^{-6} & -1.0 \times 10^{-5} \\ 9.0 \times 10^{-6} & 1.0 \times 10^{-2} & -2.0 \times 10^{-5} & 2.0 \times 10^{-5} \\ 9.0 \times 10^{-6} & -2.0 \times 10^{-5} & 1.0 \times 10^{-2} & 1.0 \times 10^{-5} \\ -1.0 \times 10^{-5} & 2.0 \times 10^{-5} & 1.0 \times 10^{-5} & 9.0 \times 10^{-3} \end{matrix}\right) & \mathrm{u} = 0 \\
\mathcal{N}\left(\begin{matrix} .16 \\ -.38 \\ -.12 \\ -.19 \end{matrix}, \begin{matrix} 1.0 \times 10^{-2} & -8.0 \times 10^{-5} & -4.0 \times 10^{-5} & -9.0 \times 10^{-5} \\ -8.0 \times 10^{-5} & 1.0 \times 10^{-2} & 1.0 \times 10^{-4} & 1.0 \times 10^{-4} \\ -4.0 \times 10^{-5} & 1.0 \times 10^{-4} & 1.0 \times 10^{-2} & 1.0 \times 10^{-4} \\ -9.0 \times 10^{-5} & 1.0 \times 10^{-4} & 1.0 \times 10^{-4} & 1.0 \times 10^{-2} \end{matrix}\right) & u = 1
\end{cases}
$$

Now consider the same after fixing parameters to their posterior mean:

$$
= \begin{cases}
\mathcal{N}\left(\begin{matrix} .17 \\ .06 \\ -.22 \\ -.07 \end{matrix}, \begin{matrix} 1.0 \times 10^{-2} & -4.0 \times 10^{-5} & 2.0 \times 10^{-5} & -4.0 \times 10^{-5} \\ -4.0 \times 10^{-5} & 1.0 \times 10^{-2} & 3.0 \times 10^{-5} & -6.0 \times 10^{-6} \\ 2.0 \times 10^{-5} & 3.0 \times 10^{-5} & 1.0 \times 10^{-2} & -1.0 \times 10^{-6} \\ -4.0 \times 10^{-5} & -6.0 \times 10^{-6} & -1.0 \times 10^{-6} & 1.0 \times 10^{-2} \end{matrix}\right) & u = 0 \\
\mathcal{N}\left(\begin{matrix} .14 \\ -.30 \\ -.07 \\ -.29 \end{matrix}, \begin{matrix} 1.0 \times 10^{-2} & -7.0 \times 10^{-5} & -1.0 \times 10^{-5} & -7.0 \times 10^{-5} \\ -7.0 \times 10^{-5} & 1.0 \times 10^{-2} & 3.0 \times 10^{-5} & 1.0 \times 10^{-5} \\ -1.0 \times 10^{-5} & 3.0 \times 10^{-5} & 1.0 \times 10^{-2} & 8.0 \times 10^{-6} \\ -7.0 \times 10^{-5} & 1.0 \times 10^{-5} & 8.0 \times 10^{-6} & 1.0 \times 10^{-2} \end{matrix}\right) & u = 1
\end{cases}
$$

On inspection, the two configurations have very similar equilibrium states. Note that the comparison is between the ground truth parameter setting and a configuration based on the mean of the posterior over parameters, such that the uncertainty represented by the whole posterior distribution may account for discrepancies.

The equilibrium state dominates any other in the data, but the difference between the two configurations arises predominantly when the input is switched, and relates to the rapidity with which the system decays to the equilibrium distribution. In either case neural activity decays to this well within the TR spacing of measurements. Such subtle differences in decay rate are indistinguishable given the relatively small number and sparsity of measurements, and further drowned in noise regardless. The method therefore has no reason to favour one over the other, and has no incentive to settle on the ground truth.

Ultimately, it may simply be the case that, regardless of the method, there is insufficient information in the data to identify the model without greater temporal resolution or a more informative prior that favours particular configurations. Similar can be said for the deterministic form of the model used in DCM.

One may therefore wonder why the simpler method of SEM, which essentially identifies parameters in the equilibrium state only, is not a more attractive option than these differential models. Recall, however, that SEM does not deconvolve the hemodynamic response, and estimates interactions at the measurement level only. Introducing latent variables into an SEM to represent neural activity falls short of achieving this, as their relationship with observed variables is non-linear.

8.7 Summary

The major contributions of this work in the fMRI domain are establishing a stochastic model of latent neural and hemodynamic activity (§8.2), formulating a filtering and smoothing approach for inference in this model, overcoming the basic practical difficulties associated with this, and identifying areas where further model and methodological advances are needed to successfully solve effective connectivity problems.

This chapter has demonstrated the potential of the proposed model and methods for the deconvolution (§8.5) problem. It has attempted the same for the effective connectivity (§8.6) problem, but results have fallen short of expectations. Arguably, this is related more to the model and data than the method. Further discussion will follow on this in §9.1.5.

Chapter 9

Conclusion

The ambitious goal of this work was to develop and deploy sophisticated machine learning methods to reveal meaningful patterns of effective connectivity beneath an observed BOLD signal in fMRI. Bayesian filtering methods have a natural appeal in combining mathematical models of biophysical processes with a proper statistical treatment of uncertainty, both intrinsic to the phenomena under study, and representative of the confidence in the model itself. The biophysical processes underlying fMRI are at present best described by the balloon model. Rather than settling for simplified models – linearity, Gaussianity, discrete time – the project has taken hold of this state of the art continuous-time model, in spite of its unwieldiness, and sought to develop the core methodology required to bring it to bear for inference. This has meant accepting a myriad of challenges – nonlinearity, non-Gaussianity, continuous time, difficult parameterisation, and others – but a biological realism and real world relevance.

Methodology for working with such models is still in its infancy, and has led to considerable effort in this work on the filtering, smoothing and parameter estimation problems in continuous time using particle methods.

We have provided a substantial review of Bayesian filtering methodology for discrete-time models (§2) and fused this with existing theory in SDEs (§3) to develop a framework for particle filtering in continuous time (§4). In doing so we have identified a number of challenges, most notably:

- the expensive computational cost of particle propagations, and

- the unavailability of a closed form transition density $p(\mathbf{x}_n \,|\, \mathbf{x}_{n-1})$.

The first of these has been tackled by introducing higher-order Runge-Kutta schemes for numerical integration into the Bayesian filtering framework. We have provided substantial experimental results throughout that clearly demonstrate the advantage of these higher order schemes over commonly used linear discretisation schemes in terms of runtime performance.

The second of these undermines the assumed analytical tractability of the transition density in most particle filtering and smoothing methods. §4 identified only those particle filtering methods using resampling proposals as being relevant for continuous-time models. At the same time it highlighted particular problems in conventional smoothers, and their very poor performance.

In response to this, §5 introduced the kernel forward-backward and kernel two-filter smoothers. Tailored for the continuous time case, these very fast, but approximate methods, may facilitate efficient model fit assessment and iterative parameter estimation schemes. They are applicable to a broader range of dynamical systems than both conventional methods and related work, in particular the general class of models expressed using stochastic differential equations. By their construction they facilitate use of the higher-order Runge-Kutta methods advocated earlier in the work. Ultimately, they are substantially more computationally efficient than conventional methods in terms of both space and runtime resources. In addition, they establish an importance sampling scheme that provides a handle into addressing the degeneracy problem identifiable in all conventional particle smoothing techniques.

§6 provided a concrete implementation of these and other approaches suitable for large-scale, high-performance distributed computing environments. This has permitted the use of such methodology at an unprecedented scale, and provided open source code to repeat the same in the form of the `dysii` C++ library.

Finally, we have taken the biologically motivated but deterministic balloon model for fMRI and extended it into the stochastic setting. We have then applied this novel model and our methods to fMRI analysis, in particular the deconvolution and effective connectivity problems.

9.1 Future work

Despite the contributions of this work, the picture is still incomplete. We discuss potential areas for future research in this section.

Perhaps most outstanding is the matter of parameter estimation, where this work has brutishly relied on its high-performance implementation to use large numbers of particles for Bayesian parameter estimation. Parameter estimation using particle methods is still in its infancy, even for discrete time. While the smoothing methods presented here may provide a vehicle to efficient parameter estimation schemes, they are not an end in their own right in this regard. We outline a few ideas for further work here.

9.1.1 Parameter estimation

Particle filtering, being an importance sampling technique, relies heavily on the design of a suitable proposal distribution in order to be effective. While any proposal distribution that is non-zero where the target density is non-zero is theoretically tenable, proposals vary considerably in their effectiveness. A well chosen proposal will draw samples strongly supporting the whole target density with approximately equal weights. A poorly chosen proposal will draw misdirected samples which lie in a limited region of the target's probability mass, possibly even its tails, with high variance in weights. More samples are required in the latter case, with heuristic measures such as ESS making this clear.

As dimensionality expands, it becomes increasingly difficult to design appropriate proposals. As an importance sampler, a particle filter in high-dimensional spaces may be destined to failure. The matter is additionally confounded when taking a Bayesian approach to parameter estimation – adding parameters into the state space increases dimensionality further. Indeed, for the fMRI effective connectivity problem, the number of parameters scales $\mathcal{O}(M^2)$ in the number of regions considered, M, and similar would be expected of any similar network-type problem.

Iterative parameter estimation schemes can at least hope to minimise the dimensionality of the state by witholding the parameters from it. Generally, these require the conjecture of a fixed parameter configuration, followed by filtering and smoothing to obtain the expected likelihood of the model under this configuration. The fast kernel smoothers introduced in this work potentially facilitate the efficient calculation of the expected likelihood. The parameter estimate may then be updated based on this likelihood, and the entire process repeated. Possible update schemes include Expectation Maximisation (EM) [91] and gradient ascent [33].

Observe that for the fMRI model presented, the parameters of most interest are the A, $B_{1:U}$ and C matrices and $\mathbf{d}$ vector of the neural model (8.1). For a single box-car input $u_1(t)$ of value 0 or 1, and equispaced measurements at time Δt, the neural model is a two-state switching linear model. When $u_1(t) = 0$ the model is given by:

$$d\mathbf{z} = (A\mathbf{z} + \mathbf{d})\,dt + \Sigma_z\,d\mathbf{W} \tag{9.1}$$

which corresponds exactly to the autoregressive:

$$\mathbf{z}^*_{t+1} = (I + \Delta t A^*)\mathbf{z}^*_t\,, \tag{9.2}$$

where $\mathbf{z}^* = (\mathbf{z}^T, 1)^T$ and $A^* = (A, \mathbf{d})$. Similarly, when $u_1(t) = 1$ the system is given by:

$$\mathbf{z}^*_{t+1} = (I + \Delta t B^*)\mathbf{z}^*_t\,, \tag{9.3}$$

where $B^* = (A + B_1, \mathbf{c} + \mathbf{d})$, with $\mathbf{c}$ being the single column of C as a vector.

Given these linear systems we can proceed with an EM algorithm. For the E-step parameters are fixed and a smoother used to obtain a weighted sample set $\{(\mathbf{s}^{(i)}_n, \psi^{(i)}_n)\}$ representing the smooth density at each time t_n. For the M-step, parameters may be updated as follows:

$$A^* \leftarrow \Sigma_0^{-1} \sum_{n \in T_0} \sum_{i=1}^{P} \psi^{(i)}_n \mathbf{s}^{(i)}_n (\mathbf{s}'^{(i)}_{n+1})^T \tag{9.4}$$

$$B^* \leftarrow \Sigma_1^{-1} \sum_{n \in T_1} \sum_{i=1}^{P} \psi^{(i)}_n \mathbf{s}^{(i)}_n (\mathbf{s}'^{(i)}_{n+1})^T \tag{9.5}$$

where $\mathbf{s}^{(i)}_*$ is the propagation of the particle at time t_n to time t_{n+1}, T_0 is the set of times where $u_1(t) = 0$ and likewise T_1 the set of times where $u_1(t) = 1$, and:

$$\Sigma_0 = \sum_{n \in T_0} \sum_{i=1}^{P} \psi_n^{(i)} \mathbf{s}_n^{(i)} (\mathbf{s}_n^{(i)})^T \tag{9.6}$$

$$\Sigma_1 = \sum_{n \in T_1} \sum_{i=1}^{P} \psi_n^{(i)} \mathbf{s}_n^{(i)} (\mathbf{s}_n^{(i)})^T \,. \tag{9.7}$$

New estimates for A, B_1, C and $\mathbf{d}$ may then be recovered straightforwardly from A^* and B^*.

The problem with this approach is that neural activity is dominated by the flatline equilibrium distribution where the autocorrelation of the function is zero and the particular parameter configuration is essentially arbitrary. Consequently, runs exhibit a convergence toward zero regardless of initial conditions. Further work is needed in this regard, although we do note that preliminary results on this would not be possible at all without the efficient smoothers of this work.

9.1.2 MCMC possibilities

An entirely different approach may be warranted. Markov Chain Monte Carlo (MCMC) techniques require a number of samples that is theoretically independent of the number of dimensions. In situations where all data is available ahead of time, a batch MCMC approach to sampling the state across all times may be an alternative to the recursive filtering approaches advocated here. Some work has already been conducted in this regard [92].

Consider a Metropolis-Hastings scheme where the current state of the chain is given by $\mathbf{s}_{1:T}^{\tau}$. A proposed new state $\mathbf{s}_{1:T}^{*}$ is drawn from a proposal $q(\mathbf{s}_{1:T} \,|\, \mathbf{s}_{1:T}^{\tau})$ and accepted with probability:

$$A(\mathbf{s}_{1:T}^{*}, \mathbf{s}_{1:T}^{\tau}) = \min\left(1, \frac{p(\mathbf{x}_{1:T} = \mathbf{s}_{1:T}^{*} \,|\, \mathbf{y}_{1:T}) q(\mathbf{s}_{1:T}^{\tau} \,|\, \mathbf{s}_{1:T}^{*})}{p(\mathbf{x}_{1:T} = \mathbf{s}_{1:T}^{\tau} \,|\, \mathbf{y}_{1:T}) q(\mathbf{s}_{1:T}^{*} \,|\, \mathbf{s}_{1:T}^{\tau})}\right) \,. \tag{9.8}$$

At a glance, we would expect the transition density to appear when expanding this acceptance rate. By way of example, consider a block Gibbs sampling scheme where the the state $\mathbf{x}_k$ for some time t_k is resampled conditioned on the remaining trajectory. The proposal reduces to:

$$\begin{aligned}
q(\mathbf{s}^*_{1:T} \,|\, \mathbf{s}^\tau_{1:T}) &= p(\mathbf{x}_k = \mathbf{s}^*_k \,|\, \mathbf{x}_{1:k-1} = \mathbf{s}^\tau_{1:k-1}, \mathbf{x}_{k+1:T} = \mathbf{s}^\tau_{k+1:T}) && (9.9)\\
&= p(\mathbf{x}_k = \mathbf{s}^*_k \,|\, \mathbf{x}_{k-1} = \mathbf{s}^\tau_{k-1}, \mathbf{x}_{k+1} = \mathbf{s}^\tau_{k+1}) && (9.10)\\
&= \frac{p(\mathbf{x}_{k+1} = \mathbf{s}^\tau_{k+1} \,|\, \mathbf{x}_{k-1} = \mathbf{s}^\tau_{k-1}, \mathbf{x}_k = \mathbf{s}^*_k) p(\mathbf{x}_k = \mathbf{s}^*_k \,|\, \mathbf{x}_{k-1} = \mathbf{s}^\tau_{k-1})}{p(\mathbf{x}_{k+1} = \mathbf{s}_{k+1} \,|\, \mathbf{x}_{k-1} = \mathbf{s}^\tau_{k-1})} && (9.11)\\
&= \frac{p(\mathbf{x}_{k+1} = \mathbf{s}^\tau_{k+1} \,|\, \mathbf{x}_k = \mathbf{s}^*_k) p(\mathbf{x}_k = \mathbf{s}^*_k \,|\, \mathbf{x}_{k-1} = \mathbf{s}^\tau_{k-1})}{\int p(\mathbf{x}_{k+1} = \mathbf{s}^\tau_{k+1} \,|\, \mathbf{x}_k) p(\mathbf{x}_k \,|\, \mathbf{x}_{k-1} = \mathbf{s}^\tau_{k-1})\, d\mathbf{x}_k}, && (9.12)
\end{aligned}$$

and again the transition density appears, twice in fact. It is not immediately clear how one can draw from this distribution, or provide a derivation which eliminates it.

9.1.3 Other interpolation methods

Kernel density approximations are a simple non-parametric approach to density estimation that are easy to apply, but even with efficient partition tree calculations are limited to $\mathcal{O}(\lg P \lg P)$ runtime performance for each smoothing step. One can imagine other interpolation techniques also, such as EM or variational fitting of Gaussian mixtures or other analytical distributions. It may even be possible to tailor a filter around these, such as propagating selected sigma point samples from each component of a Gaussian mixture in a manner inspired by some sort of continuous-time extension to the unscented Kalman filter [19]. These may prove more efficient, although such parametric approaches may be model dependent.

9.1.4 Other continuous time behaviours

While SDEs may be used to represent the class of continuous-time stochastic diffusions, they do not encompass the full range of continuous-time stochastic processes. The *Chapman-Kolmogorov equation* [17, ch.3] is a more general means of representing stochastic processes. From this general equation and its differential form, the families of jump processes, Fokker-Planck equations, diffusions and others may be derived. The precise form of the Chapman-Kolmogorov equation is unimportant here. What is important is that it neatly breaks down into the three main properties of a stochastic process:

drift which is the deterministic component of the process,
diffusion which is the continuous uncertainty of the process, and
jumps which are the discontinuous uncertainties of the process.

By selectively zeroing out one or more of these components, various process families are obtained. Processes with only a drift component form deterministic *Liouville equations* [17, §3.5.3], only a jump component *jump processes* [17, §3.5.1], and only drift and diffusion components *diffusion processes*, often described by *Fokker-Planck equations* [17, §3.5.2] or SDEs.

This appears a useful way of thinking of stochastic processes, and of positioning SDEs relative to other theory. This work is currently only applicable to diffusion process models described using SDEs. A reasonable extension would be to consider other families, particularly jump processes, and how these could be fit into the framework developed, or the framework extended to suit.

9.1.5 fMRI developments

Improving the results of the fMRI experimentation in this work requires a more integrated approach to model development and data collection that requires additional expertise in the field.

Unfortunately, the data sets available for use in this work are not particularly conducive to the temporal analysis desired, having high TR and block experimental design. More rapidly acquired images on more recent hardware, at greater temporal resolution and using event-related experimental paradigms, are probably more useful for this sort of analysis. Bringing in data from other more temporally sensitive modalities, such as Electroencephalography (EEG) or Magnetoencephalography (MEG), may also prove useful.

Expert knowledge may also be used to fix a subset of connectivity parameters, or at least to provide more informative priors over them. The effect of this would be to constrain the number of configurations manifesting equivalent equilibrium distributions and facilitate identification of a single most likely set of parameter values.

Bibliography

[1] Heeger, D. J. and Ress, D. (2002) What does fMRI tell us about neuronal activity? *Nature Reviews: Neuroscience*, **3**, 142–151.

[2] Bentler, P. M. and Weeks, D. G. (1980) Linear structural equations with latent variables. *Psychometrika*, **45**, 289–307.

[3] McArdle, J. J. and McDonald, R. P. (1984) Some algebraic properties of the reticular action model for moment structures. *British Journal of Mathematical and Statistical Psychology*, **37**, 234–251.

[4] Friston, K., Harrison, L., and Penny, W. (2003) Dynamic causal modelling. *NeuroImage*, **19**, 1273–1302.

[5] Golightly, A. and Wilkinson, D. J. (2006) Bayesian sequential inference for nonlinear multivariate diffusions. *Statistics and Computing*, **16**, 323–338.

[6] Fearnhead, P., Papaspiliopoulos, O., and Roberts, G. O. (2008) Particle filters for partially observed diffusions. *Journal of the Royal Statistical Society Series B*, **70**, 755–777.

[7] Beskos, A., Papaspiliopoulos, O., Roberts, G., and Fearnhead, P. (2006) Exact and efficient likelihood-based inference for discretely observed diffusion processes (with discussion). *Journal of the Royal Statistical Society Series B*, **68**, 333–382.

[8] Ozaki, T. (1993) A local linearization approach to nonlinear filtering. *International Journal on Control*, **57**, 75–96.

[9] Archambeau, C., Cornford, D., Opper, M., and Shawe-Taylor, J. (2007) Gaussian process approximations of stochastic differential equations. *Journal*

of Machine Learning Research Workshop and Conference Proceedings, **1**, 1–16.

[10] Archambeau, C., Opper, M., Shen, Y., Cornford, D., and Shawe-Taylor, J. (2008) Variational inference for diffusion processes. Platt, J., Koller, D., Singer, Y., and Roweis, S. (eds.), *Advances in Neural Information Processing Systems 20*, Cambridge, MA, MIT Press.

[11] Stavropoulos, P. and Titterington, D. (2001) *Sequential Monte Carlo Methods in Practice*, chap. Improved Particle Filters and Smoothing. Springer.

[12] Klaas, M., Briers, M., de Freitas, N., Doucet, A., Maskell, S., and Lung, D. (2006) Fast particle smoothing: If I had a million particles. *Proceedings of the 23rd International Conference on Machine Learning*.

[13] Thrun, S., Langford, J., and Fox, D. (1999) Monte Carlo hidden Markov models: Learning non-parametric models of partially observable stochastic processes. *Proceedings of the International Conference on Machine Learning*, Bled, Slovenia.

[14] Alspach, D. L. and Sorenson, H. W. (1972) Nonlinear Bayesian estimation using Gaussian sum approximations. *IEEE Transactions on Automatic Control*, **17**, 439–448.

[15] Kitagawa, G. (1994) The two-filter formula for smoothing and an implementation of the Gaussian-sum smoother. *Annals of the Institute of Statistical Mathematics*, **46**, 605–623.

[16] van der Merwe, R. and Wan, E. (2003) Gaussian mixture sigma-point particle filters for sequential probabilistic inference in dynamic state-space models. *IEEE International Conference on Acoustics, Speech and Signal Processing*.

[17] Gardiner, C. W. (2004) *Handbook of Stochastic Methods for Physics, Chemistry and the Natural Sciences*. Springer, third edn.

[18] Kalman, R. (1960) A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, **80**, 35–45.

[19] Julier, S. J. and Uhlmann, J. K. (1997) A new extension of the Kalman filter to nonlinear systems. *The Proceedings of AeroSense: The 11th International*

Symposium on Aerospace/Defense Sensing, Simulation and Controls, Multi Sensor Fusion, Tracking and Resource Management.

[20] Wan, E. A. and van der Merwe, R. (2000) The unscented Kalman filter for nonlinear estimation. *Proceedings of IEEE Symposium on Adaptive Systems for Signal Processing Communications and Control*, pp. 153–158.

[21] Gordon, N., Salmond, D., and Smith, A. (1993) Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings-F*, **140**, 107–113.

[22] Isard, M. and Blake, A. (1998) Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, **29**, 5–28.

[23] Kitagawa, G. (1996) Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, **5**, 1–25.

[24] Pitt, M. and Shephard, N. (1999) Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, **94**, 590–599.

[25] Musso, C., Oudjane, N., and Gland, F. L. (2001) *Sequential Monte Carlo Methods in Practice*, chap. Improving Regularised Particle Filters. Springer.

[26] Silverman, B. (1986) *Density Estimation for Statistics and Data Analysis*. Chapman and Hall.

[27] van der Merwe, R., Doucet, A., de Freitas, N., and Wan, E. (2000) The unscented particle filter. *Advances in Neural Information Processing Systems*, **13**.

[28] Doucet, A., de Freitas, N., and Gordon, N. (eds.) (2001) *Sequential Monte Carlo Methods in Practice*. Springer.

[29] Liu, J. S. and Chen, R. (1995) Blind deconvolution via sequential imputations. *Journal of the American Statistical Association*, **90**, 567–576.

[30] Isard, M. and Blake, A. (1998) A smoothing filter for Condensation. *Proceedings of the 5th European Conference on Computer Vision*, **1**, 767–781.

[31] Doucet, A., Godsill, S., and West, M. (2000) Monte Carlo filtering and smoothing with application to time-varying spectral estimation. *IEEE International Conference on Acoustics, Speech and Signal Processing*.

[32] Kitagawa, G. (1998) A self-organising state-space model. *Journal of the American Statistical Association*, **93**, 1203–1215.

[33] Doucet, A. and Tadic, V. B. (2003) Parameter estimation in general state-space models using particle methods. *Annals of the Institute of Statistical Mathematics*, **55**, 409–422.

[34] Fehlberg, E. (1968) Classical fifth, sixth, seventh and eighth order Runge-Kutta formulas with stepsize control. Tech. Rep. R-287, National Aeronautics and Space Administration.

[35] Kloeden, P. E. and Platen, E. (1992) *Numerical Solution of Stochastic Differential Equations*. Springer.

[36] Goodman, J., Moon, K.-S., Szepessy, A., Tempone, R., and Zouraris, G. (2006) Stochastic and partial differential equations with adapted numerics, lecture notes.

[37] Greiner, A., Strittmatter, W., and Honerkamp, J. (1988) Numerical integration of stochastic differential equations. *Journal of Statistical Physics*, **51**, 95–107.

[38] Wilkie, J. (2004) Numerical methods for stochastic differential equations. *Physical Review E*, **70**.

[39] Gaines, J. and Lyons, T. (1997) Variable step size control in the numerical solution of stochastic differential equations. *SIAM Journal on Applied Mathematics*, **57**, 1455–1484.

[40] Fehlberg, E. (1969) Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems. Tech. Rep. R-315, National Aeronautics and Space Administration.

[41] Prince, P. and Dormand, J. (1981) High order embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, **7**, 67–75.

[42] Bastani, A. F. and Hosseini, S. M. (2007) A new adaptive Runge-Kutta method for stochastic differential equations. *Journal of Computational and Applied Mathematics*, **206**, 631–644.

[43] Rauch, H., Tung, F., and Striebel, C. (1965) Maximum likelihood estimates of linear dynamic systems. *American Institute of Aeronautics and Astronautics*, **3**, 1445.

[44] Gray, A. G. and Moore, A. W. (2001) 'N-body' problems in statistical learning. *Advances in Neural Information Processing Systems*, **13**.

[45] Chorin, A. J. and Krause, P. (2004) Dimensional reduction for a Bayesian filter. *PNAS*, **101**, 15013–15017.

[46] Murray, L. and Storkey, A. (2008) Continuous time particle filtering for fMRI. *Advances in Neural Information Processing Systems*, **20**, to appear.

[47] Daumas, M. and Evripidou, P. (2000) Parallel implementations of the selection problem: A case study. *International Journal of Parallel Programming*, **28**, 103–131.

[48] Gray, A. G. and Moore, A. W. (2003) Rapid evaluation of multiple density models. *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*.

[49] Ogawa, S., Lee, T., Kay, A., and Tank, D. (1990) Brain magnetic resonance imaging with contrast dependent on blood oxygenation. *Proceedings of the National Academy of Sciences*, **87**, 9868–9872.

[50] Cohen, M. S. and Bookheimer, S. Y. (1994) Localization of brain function with magnetic resonance imaging. *Trends in Neurosciences*, **17**, 268–277.

[51] Buxton, R. B. (2002) *Introduction to Functional Magnetic Resonance Imaging: Principles & Techniques*. Cambridge University Press.

[52] Niessing, J., Ebisch, B., Schmidt, K. E., Niessing, M., Singer, W., and Galuske, R. (2005) Hemodynamic signals correlate tightly with synchronized gamma oscillations. *Science*, **309**, 948–951.

[53] Logothetis, N. K., Pauls, J., Augath, M., Trinath, T., and Oeltermann, A. (2001) Neurophysiological investigation of the basis of the fMRI signal. *Nature*, **412**, 150–157.

[54] Mukamel, R., Gelbard, H., Arieli, A., Hasson, U., Fried, I., and Malach, R. (2005) Coupling between neuronal firing, field potentials and fMRI in human auditory cortex. *Science*, **309**, 951–954.

[55] Wang, R., Foniok, T., Wamsteeker, J. I., Qiao, M., Tomaneka, B., Vivanco, R. A., and Tuor, U. I. (2006) Transient blood pressure changes affect the functional magnetic resonance imaging detection of cerebral activation. *NeuroImage*, **31**, 1–11.

[56] Behzadi, Y. and Liu, T. T. (2006) Caffeine reduces the initial dip in the visual bold response at 3T. *NeuroImage*, **32**, 9–15.

[57] Buxton, R. B., Wong, E. C., and Frank, L. R. (1998) Dynamics of blood flow and oxygenation changes during brain activation: The balloon model. *Magnetic Resonance in Medicine*, **39**, 855–864.

[58] Glaser, D., Friston, K., Mechelli, A., Turner, R., and Price, C. (2004) *Human Brain Function*, chap. 41, pp. 823–842. Elsevier.

[59] Mandeville, J. B., Marot, J. J. A., Ayata, C., Zaharchuk, G., Moskowitz, M. A., Rosen, B. R., and Weisskoff, R. M. (1999) Evidence of a cerebrovascular postarteriole Windkessel with delayed compliance. *Journal of Cerebral Blood Flow & Metabolism*, **19**, 679–689.

[60] Friston, K. J., Mechelli, A., Turner, R., and Price, C. J. (2000) Nonlinear responses in fMRI: The balloon model, Volterra kernels, and other hemodynamics. *NeuroImage*, **12**, 466–477.

[61] Vanzetta, I. and Grinvald, A. (1999) Increased cortical oxidative metabolism due to sensory stimulation: implications for functional brain imaging. *Science*, **286**, 1555–1558.

[62] Zarahn, E. (2001) Spatial localization and resolution of BOLD fMRI. *Current Opinion in Neurobiology*, **11**, 209–212.

[63] Zheng, Y., Martindale, J., Johnston, D., Jones, M., Berwick, J., and Mayhew, J. (2002) A model of the hemodynamic response and oxygen delivery to brain. *NeuroImage*, **16**, 617–637.

[64] Buxton, R. B., Uludag, K., Dubowitz, D. J., and Liu, T. T. (2004) Modeling the hemodynamic response to brain activation. *NeuroImage*, **23**, S220–S233.

[65] Friston, K. and Buchel, C. (2004) *Human Brain Function*, chap. 49, pp. 999–1018. Elsevier.

[66] Schlosser, R., Gesierich, T., Kaufmann, B., Vucurevic, G., Hunsche, S., Gawehn, J., and Stoeterb, P. (2003) Altered effective connectivity during working memory performance in schizophrenia: a study with fMRI and structural equation modeling. *NeuroImage*, **19**, 751–763.

[67] Au Duong, M., et al. (2005) Modulation of effective connectivity inside the working memory network in patients at the earliest stage of multiple sclerosis. *NeuroImage*, **24**, 533–538.

[68] de Marco, G., de Bonis, M., Vrignaud, P., Henry-Feugeas, M., and Peretti, I. (2006) Changes in effective connectivity during incidental and intentional perception of fearful faces. *NeuroImage*, **30**, 1030–1037.

[69] Fu, C. H., McIntosh, A. R., Kim, J., Chau, W., Bullmore, E. T., Williams, S. C., Honey, G. D., and McGuire, P. K. (2006) Modulation of effective connectivity by cognitive demand in phonological verbal fluency. *NeuroImage*, **30**, 266–271.

[70] Ethofer, T., Anders, S., Erb, M., Herbert, C., Wiethoff, S., Kissler, J., Grodd, W., and Wildgruber, D. (2006) Cerebral pathways in processing of affective prosody: A dynamic causal modeling study. *NeuroImage*, **30**, 580–587.

[71] Friston, K. (2004) *Human Brain Function*, chap. 31, pp. 599–634. Elsevier.

[72] Amaro Jr., E. and Barker, G. J. (2006) Study design in fMRI: Basic principles. *Brain and Cognition*, **60**, 220–232.

[73] Frackowiak, R. S., Friston, K. J., Frith, C. D., Dolan, R. J., Price, C. J., Zeki, S., Ashburner, J., and Penny, W. (2004) *Human Brain Function*. Elsevier Academic Press, second edn.

[74] Ashburner, J. and Friston, K. (2004) *Human Brain Function*, chap. 32, pp. 635–654. Elsevier.

[75] Ashburner, J. and Friston, K. (2004) *Human Brain Function*, chap. 34, pp. 655–672. Elsevier.

[76] Wellcome Department of Imaging Neuroscience (2006), Statistical parametric mapping. Online at www.fil.ion.ucl.ac.uk/spm/.

[77] Kiebel, S. and Holmes, A. (2004) *Human Brain Function*, chap. 37, pp. 725–760. Elsevier.

[78] Friman, O., Borga, M., Lundberg, P., and Knutssonb, H. (2004) Detection and detrending in fMRI data analysis. *NeuroImage*, **22**, 645–655.

[79] Stephan, K. E., Penny, W. D., Marshall, J. C., Fink, G. R., and Friston, K. J. (2005) Investigating the functional role of callosal connections with dynamic causal models. *Annals of the New York Academy of Sciences*, **1064**, 16–36.

[80] Gitelman, D. R., Penny, W. D., Ashburner, J., and Friston, K. J. (2003) Modeling regional and psychophysiologic interactions in fMRI: the importance of hemodynamic deconvolution. *NeuroImage*, **19**, 200–207.

[81] McDonald, R. P. and Ho, M.-H. R. (2002) Principles and practice in reporting structural equation analyses. *Psychological Methods*, **7**, 64–82.

[82] Bullmore, E., Horwitz, B., Honey, G., Brammer, M., Williams, S., and Sharma, T. (2000) How good is good enough in path analysis of fMRI data? *NeuroImage*, **11**, 289–301.

[83] Harrison, L. and Friston, K. (2004) *Human Brain Function*, chap. 50, pp. 1019–1047. Elsevier.

[84] Penny, W., Stephan, K., Mechelli, A., and Friston, K. (2004) Modelling functional integration: a comparison of structural equation and dynamic causal models. *NeuroImage*, **23**, S264–S274.

[85] Tabachnick, B. G. and Fidell, L. S. (2001) *Using Multivariate Statistics*. Allyn & Bacon, international student edn.

[86] Scientific Software International (2006), LISREL web site. Online.

[87] SAS (2006), SAS/STAT user's guide. Online.

[88] Kline, R. B. (2005) *Principles and Practice of Structural Equation Modeling*. The Guilford Press, second edn.

[89] Stephan, K. E., Weiskopf, N., Drysdale, P. M., Robinson, P. A., and Friston, K. J. (2007) Comparing hemodynamic models with dcm. *NeuroImage*, **38**, 387–401.

[90] Riera, J. J., Watanabe, J., Kazuki, I., Naoki, M., Aubert, E., Ozaki, T., and Kawashim, R. (2004) A state-space model of the hemodynamic approach: nonlinear filtering of BOLD signals. *NeuroImage*, **21**, 547–567.

[91] Ninness, B. and Gibson, S. (2001) The EM algorithm for multivariable dynamic system estimation. Tech. Rep. EE200101, Department of Electrical and Computer Engineering, University of Newcastle.

[92] Golightly, A. and Wilkinson, D. (2008) Bayesian inference for nonlinear multivariate diffusion models observed with error. *Computational Statistics & Data Analysis*, **52**, 1674–1693.