THE UNIVERSITY *of* EDINBURGH

Parameter Estimation in Sparse State-Space Models

Benjamin Cox

Doctor of Philosophy
University of Edinburgh
2025

Declaration

I declare that this thesis was composed by myself, Benjamin Cox, under the guidance of my supervisor, Victor Elvira. This work has not been submitted, in whole or in part, for any other degree or professional qualification. Chapters 3, 4, and 6, are based on published works, of which I am the first of multiple co-authors and the primary contributor.

Chapter 3 is based on the following works:

- B. Cox and V. Elvira, "Sparse Bayesian Estimation of Parameters in Linear-Gaussian State-Space Models," in IEEE Transactions on Signal Processing, vol. 71, pp. 1922-1937, 2023, doi: 10.1109/TSP.2023.3278867.
- B. Cox and V. Elvira, "Parameter Estimation in Sparse Linear-Gaussian State-Space Models via Reversible Jump Markov Chain Monte Carlo," 2022 30th European Signal Processing Conference (EUSIPCO), Belgrade, Serbia, 2022, pp. 797-801

This work was jointly conceptualised by myself and Victor, with me performing the research, design, implementation, experimentation, and main write-up, and Victor providing editorial support and feedback.

Chapter 5 is based on the following works:

- B. Cox, É. Chouzenoux and V. Elvira, "GraphGrad: Efficient Estimation of Sparse Polynomial Representations for General State-Space Models," in IEEE Transactions on Signal Processing, vol. 73, pp. 1562-1576, 2025, doi: 10.1109/TSP.2025.3554876
- B. Cox, É. Chouzenoux and V. Elvira, "Learning a Sparse Polynomial Approximation to the Transition Function of General State-Space Models," ICASSP 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Hyderabad, India, 2025

This work was jointly conceptualised by myself, Émilie, and Victor, with me performing the research, design, implementation, experimentation, and main write-up, and Victor and Émilie providing mathematical support and guidance, editorial support, and feedback.

Chapter 6 is based on the following works:

- B. Cox, S. Segarra and V. Elvira, "Learning state and proposal dynamics in state-space models using differentiable particle filters and neural networks", in Signal Processing, vol. 234, 2025, doi: 10.1016/j.sigpro.2025.109998

- B. Cox, S. Pérez-Vieites, N. Zilberstein, M. Sevilla, S. Segarra and V. Elvira, "End-to-End Learning of Gaussian Mixture Proposals Using Differentiable Particle Filters and Neural Networks," ICASSP 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Seoul, Korea, Republic of, 2024, pp. 9701-9705.

The initial concept for this work was jointly conceptualised by myself, Sara, and Victor, with me performing the research, design, implementation, experimentation, and main write-up, Sara contributing significantly to the write-up, and all other authors providing editorial support and feedback for the first paper.

The extension leading to the second paper was conceptualised by myself, with me performing the research, design, implementation, experimentation, and main write-up, and Santiago and Victor providing editorial support and feedback.

Other published peer-reviewed works outwith this thesis include:

- M. Sevilla, N. Zilberstein, B. Cox, S. Pérez-Vieites, V. Elvira and S. Segarra, "State and Dynamics Estimation with the Kalman-Langevin filter," 2023 57th Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 2023, pp. 1372-1376.

This work has not been included as I was not the primary contributor.

(*Benjamin Cox*)

Lay summary

In several scientific disciplines, it is common to encounter systems which change in various ways over time. For example, in finance, modelling the behaviour of an asset in time can be used to reduce the risk of a set of investments of a pension, or in numerical weather prediction, modelling the impact of newly observed data on broader weather patterns can help to more accurately predict rain. This type of modelling is broadly known as time series analysis, and forms the backbone of many quantitative fields, such as econometrics, statistical ecology, signal processing, numerical weather prediction, and many more.

In order to utilise time series techniques to tackle a problem, we must first design or choose a model for the system we are investigating. It is common that we are interested in modelling a phenomena that we do not directly observe. For example, in finance, we are often interested in modelling the volatility of an asset, but only observe its price, or in ecology, we may be interested in the population of a species in an area, but observe only how many are seen on camera each day. This complication requires us to model the relationship between the quantity we are interested and the quantity that we observe. We can do this using a class of models called state-space models. State-space models allow us to model the unobserved underlying state as related to an observed noisy observation. As we model the relationship between the unobserved and observed components, as well as the noise present in both, we can obtain a very good representation of several real-life systems utilising these techniques.

In this thesis, we focus on state-space models. In order to utilise a state-space model to recover state estimates for a given series of observations, we must know both the form of the state-space model, and the parameters of the model. These parameters can be known from a baseline, which is common in epidemiology, or derived from an understanding of the system, such as is common in numerical weather prediction. However, if no such parameters are apparent, then we must estimate the model parameters from the data we have on the system. We focus on estimating the model parameters in such a way that they exhibit sparsity, which occurs when many of the parameter values are zero. Sparsity makes the model more interpretable, and allows the model parameters to be interpreted as representing the connectivity of the system. Estimating the connectivity of the state space is very useful, as it allows for inference as to what parts of the state impact others. For example, in ecology, we may jointly estimate the population of two species, and infer that a large population in one species drives down the population in the other species.

We present several techniques relating to sparse modelling of the driving parameters in state space models, and also present a method for estimating the state in a general

state-space model where only the observation dynamics are known. We also present methods for estimating state-space models when the form of the underlying model is not known. We show that the proposed methods perform well in a variety of scenarios, and can be broadly applied to several types of models.

Abstract

State-space models are a flexible framework for modelling sequential data in the presence of noise or incomplete observations, within which we model a system via a hidden state process and a related observation process. These processes are described via a pair of distributions encoding the state dynamics and the observation process, with the distribution of the current state depending only on the previous state, and the distribution of the current observation depending only on the current state. In general, the parameters of these distributions are unknown, and are challenging to estimate, with conventional estimation schemes failing due to the temporal dependence of the time series, and the resultant concentration of the likelihood function. In this work we present several methods to estimate the parameters of state-space models, as well as some methods for estimating the form of the model itself when this is unknown. In particular, we focus on methods that admit interpretable estimates via promoting sparsity in the parameters, thereby shrinking many parameter values to zero.

In the first contributing chapter of this work, Chapter 3, we propose a method to obtain sparse Bayesian estimates of the transition matrix of a linear-Gaussian state-space model by utilising reversible jump Markov chain Monte Carlo. We discuss the construction of the reversible jump kernel, and how to interpret the sampled sparsity in terms of a Bayesian causality. We demonstrate our method on several synthetic datasets, where we have the ground truth of causality, and on real-world weather data where we do not, comparing the performance to the existing state-of-the-art.

In Chapter 4, we propose a method to promote graphical clusters in the transition parameters of a linear-Gaussian state-space model by utilising a sparsity promoting estimation scheme in conjunction with a dynamically adaptive penalty. We design a general framework to construct state clustering methods within state-space models, and then construct a representative method as a case of this general framework, wherein we apply ideas from network analysis to design an iteratively applied cluster promoting penalty function. We test our method on a series of synthetic datasets, and compare the performance to the existing state of the art.

In Chapter 5, we propose a method to construct a polynomial representation of a

general state-space model, whereby we learn a sparse approximation of the transition function from a basis of polynomial terms. This allows us to infer the connectivity of the hidden states, thereby providing insight into the unknown underlying dynamics.

In the final main chapter, Chapter 6, we propose a method to approximate the intractable optimal proposal of a particle filter utilising a shallow neural network which parametrises a Gaussian mixture distribution. We compare this proposal to several standard proposals, and extend the work to simultaneous estimation of the transition and proposal distributions.

Finally, we provide some concluding remarks on the techniques developed, and present a number of potential avenues for future research.

Contents

Chapter 1

Introduction

In many scientific disciplines, it is required to study a system that evolves in time. Data collected from such systems is known as time series data, and the processing of this data is the focus of the discipline known as time series analysis [1]. Within this discipline, there are several ways to model time series data, such as graphically [2], autoregressively [3], or via black-box machine learning techniques [4]. Another method is state-space modelling, where the system is modelled as having an unobserved noisy latent state that evolves in time, and which is related to the observed series of noisy observations [5, 6]. This latent process is typically such that it encodes the variable that we are interested in modelling. For example, in numerical weather prediction, we may observe the wind using various instruments, but these instruments have error, and measure the wind through a process, not directly [7]. Another example is in finance, where we can often obtain the price of an asset, but do not directly observe its volatility, which is of interest for risk forecasting [8]. State-space models are commonly used in many areas, and hence methods for improving their efficiency and applicability are widely applicable.

In order to perform filtering in state-space models, the model must be fully specified with all parameters being known [5]. However, in practice, this is often not the case, and typically some model parameters are unknown. Therefore, we estimate the parameters, and in some cases the model form, from the available data [1]. This results in a difficult parameter estimation problem with a complicated and concentrated likelihood structure. Parameter estimation is a much more difficult task than performing filtering on the same models, and is in general more computationally expensive, as most parameter estimation algorithms perform filtering as part of their estimation [5, 9, 10]. In particular, estimating the parameters of a linear-Gaussian state space model, in many ways the simplest continuous state-space model, in all but the simplest cases requires multiple evaluations of the filtering equations to obtain the likelihood, which is far more expensive compared to evaluating the likelihood of

a typical statistical model [5, 10, 11]. Even more difficult, in the general setting it is not possible to evaluate the likelihood exactly, and we require approximations such as the extended Kalman filter or the particle filter. These methods are more expensive than the Kalman filter, and yield only an approximate likelihood, making parameter estimation even more challenging [5, 12, 9]. Fortunately, there exist several methods for performing estimation in state-space models, with techniques based on Markov chain Monte Carlo (MCMC) [10, 13] and expectation maximisation (EM) [5, 11] being commonly used methodological frameworks for parameter estimation in state-space models. However, in general, methods developed within these frameworks do not take into account the structure of the underlying parameters; this is a trade-off for their generic applicability. For a state-space model this structure is very informative, and therefore we should, if possible, incorporate it into the design of our estimation scheme.

In real-world systems, where each latent state dimension has a physical interpretation, the system dynamics are often capable of being represented by sparse interacting subsystems [14]. Furthermore, this between-step connectivity of the latent states can be represented as a graph, with edges implying that the source Granger-causes the destination [15, 16]. This graphical interpretation of parameter sparsity is particularly powerful, as, in addition to providing an easy to understand representation of the underlying dynamics, it allows us to potentially partition the state-space into interacting subsystems that can be studied in isolation [16]. In addition, if a state-space model can be represented by a graphical model where the underlying graph is made up of disconnected components, we can potentially parallelise the evaluation of the filtering equations, thereby reducing the cost of recovering the state in large and expensive to evaluate systems. As we generally wish to recover estimates that encapsulate the true behaviour of a system, whatever scheme we use to estimate the parameters of a state-space model should be able to exhibit this sparsity in the recovered estimates [16].

In this thesis we introduce several parameter estimation methods for both linear-Gaussian state-space models and general state space models. These methods aim at discovering latent structure in the modelled series, and cover a range of point-based and distributional approaches. The capability to recover sparse graphical models is a core feature of most of the methods we propose in this thesis, such as those proposed in Chapters 3, 4, and 5. However, if we are not interested in recovering the underlying structure, and instead are interested in only the state, then we can utilise additional methods such as that proposed in Chapter 6, which improve state recovery at the expense of model interpretability.

Chapter 2: Background on State-Space Models and Parameter Estimation

In this chapter, we provide background information on existing methods for parameter estimation in state-space models, including such methods as importance sampling and particle Markov chain Monte Carlo (MCMC).

Chapter 3: Sparse Bayesian Estimation of Transition Parameters in Linear-Gaussian State-Space Models

We proceed to develop a fully Bayesian method for estimating the transition matrix of a linear-Gaussian state-space model (LGSSM) such that the recovered transition matrix exhibits true sparsity in each sample. This stands in contrast to standard sparsity promoting schemes, that when applied within a Bayesian sampling framework do not yield exact zeros. We achieve this using a reversible jump MCMC sampler, with a novel jump proposal over the sparsity, which is interpreted as a model space. We demonstrate the performance of this method on several synthetic experiments, and apply it to real-world data.

Chapter 4: Cluster Discovery in Linear-Gaussian State-Space Models via Iterative Constraints

In this chapter, we propose a framework for estimating the transition matrix of a linear-Gaussian state-space model such that the estimate exhibits a clustered structure. Our framework can utilise several method for promoting cluster structure, such as betweenness centrality or minimum cuts. We evaluate the performance of this method compared to existing sparse parameter estimation methods for LGSSMs, and demonstrate a superior performance when the underlying model exhibits a clustered structure, and comparable performance when it does not.

Chapter 5: Approximate Learning of General State-Space Models via a Sparse Polynomial Library

In this chapter, we develop a method for estimating the state transition kernel of a non-linear state-space model utilising a polynomial approximation, in such a manner as to yield a sparse connectivity graph. First, we develop the polynomial approximation, and show that several popular dynamical systems can be exactly recovered within the implied basis. Next, we outline a method for estimating the parameters of this approximation utilising a sparsity-promoting gradient scheme. Then, we develop a variant of our method to automatically combat likelihood degeneration. Finally, we compare our method with a polynomial maximum likelihood scheme, and show that

our proposed sparsity promoting method yields better results in both recovering the underlying structure, and recovering the true value of the driving parameters.

Chapter 6: Neural Transition Kernels and Proposal Distributions for Particle Filtering

This chapter proposes a method to jointly estimate the transition kernel and optimal proposal distribution of a general state-space model. We do this using a neural network to learn the mean and variance functionals of the components of two Gaussian mixture distributions, which then serve as the transition kernel and state proposal distribution. We evaluate the performance of this method against several existing improved proposal schemes.

Chapter 7: Conclusion

This chapter contains our concluding remarks, and presents several avenues for further research and extension of proposed methods.

Chapter 2

Background on State-Space Models and Parameter Estimation

In many scientific disciplines, we require to model time variable systems. For example, in biology it is common to model the response to a course of treatment, in mathematical finance it is typical to model the price of a security, and in ecology the goal is often to model the population dynamics of some species. This type of data is collected sequentially in time, and is referred to as time series data. We can use time series data to build a model for the underlying system from which the data was gathered, with this model then being used to perform inference, such as forecasting the future values of the system based on the observed values, determining the most important factors in the evolution of the system, or probabilistically recovering the state of the series given a related series of observations [1].

Furthermore, we often require to estimate the parameters of these models, as they are not known beforehand. Parameter estimation is a very common problem in statistics, and therefore has many possible methods with which it can be addressed [17]. First, we will address parameter estimation in a general setting. Then, we will introduce the state-space model, for which standard parameter estimation schemes often fail [5]. Finally, we will give some methods for parameter estimation within state-space models.

2.1 Parameter estimation

In statistical inference, we often estimate the parameters of a model, as they are generally not known. Such parameter estimation in many ways forms the core of statistics, and thus there are many ways by which to proceed. Commonly, parameter estimation schemes are tailored to take advantage of the specific structure of the model for which they are designed. This is the case for the schemes discussed in Section 2.3,

however here we will discuss some more generic schemes which we build off to create more specific schemes.

2.1.1 Monte Carlo parameter estimation methods

In standard statistical inference, Monte Carlo methods are commonly used to estimate distributions and expectations via sets of samples. These methods are particularly useful when dealing with complex distributions where direct analytical solutions are not feasible.

We can estimate $p(\boldsymbol{\theta})$ using the set of K independent samples $\{\boldsymbol{\theta}_k\}_{k=1}^K$, with $\boldsymbol{\theta}_k \sim p(\boldsymbol{\theta})$ using the following empirical approximation:

$$\hat{p}(\boldsymbol{\theta}) = \frac{1}{K}\sum_{k=1}^{K}\delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}), \tag{2.1}$$

where $\delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta})$ is the Dirac delta function centred at $\boldsymbol{\theta}_k$ and evaluated at $\boldsymbol{\theta}$. To approximate the expectation of a function $f(\boldsymbol{\theta})$ with respect to $p(\boldsymbol{\theta})$, we can use the Monte Carlo estimator

$$\mathbb{E}_p(f(\boldsymbol{\theta})) = \int f(\boldsymbol{\theta})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta} \approx \frac{1}{K}\sum_{k=1}^{K} f(\boldsymbol{\theta}_k). \tag{2.2}$$

This estimator is unbiased, consistent, and converges to the true value of $\mathbb{E}_p(f(\boldsymbol{\theta}))$ at a rate of $O(\frac{1}{K})$ [18].

Methods of sampling $p(\boldsymbol{\theta})$ directly are often specific to a distribution, but a universally applicable method for distributions that admit an inverse CDF $F^{-1}(\boldsymbol{\theta})$is the probability inverse transform, whereby samples $u_k \sim U(0,1)$ are transformed to samples from $p(\boldsymbol{\theta})$ by $\boldsymbol{\theta}_k = F^{-1}(u_k) \sim p(\boldsymbol{\theta})$.

However, directly sampling from $p(\boldsymbol{\theta})$ is often not feasible. In such cases, indirect sampling methods can be employed. One approach involves transforming samples drawn from a proposal distribution $q(\boldsymbol{\theta})$ and modifying them utilising a series of weights so that they are valid samples from the target distribution $p(\boldsymbol{\theta})$. This requires only evaluating the density of the target distribution up to proportionality. This approach is known as importance sampling [18].

Another approach is to construct a Markov chain with the target distribution $p(\boldsymbol{\theta})$ as the stationary distribution, and generate samples from that Markov chain. This also typically requires only the density of the target distribution up to proportionality, but more advanced methods can require the gradient of the target density as well. This method of generating samples is known as Markov chain Monte Carlo (MCMC), and forms the basis for modern Bayesian statistical computing [19]. Note that other

methods such as variational inference [20] can be used to generate sample, but are excluded here as they are not relevant to this thesis.

Importance Sampling

Importance sampling is a technique by which we can evaluate expectations of a complex target distribution $p(\boldsymbol{\theta})$ using samples from a simpler proposal distribution $q(\boldsymbol{\theta})$ via a weighted average of the samples [18]. In order to perform importance sampling, $\text{supp}(p) \subseteq \text{supp}(q)$, where supp denotes the support [18]. First, note that we can construct a Monte Carlo estimate to $q(\boldsymbol{\theta})$ by sampling K samples $\boldsymbol{\theta}_k$ for $k=1,...,K$ from $q(\boldsymbol{\theta})$

$$\hat{q}(\boldsymbol{\theta}) = \frac{1}{K}\sum_{k=1}^{K}\delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}). \tag{2.3}$$

Therefore, we can construct an approximation to $p(\boldsymbol{\theta})$ by

$$p(\boldsymbol{\theta}) = \frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}q(\boldsymbol{\theta}) \approx \frac{1}{K}\sum_{k=1}^{K}\frac{p(\boldsymbol{\theta}_k)}{q(\boldsymbol{\theta}_k)}\delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}). \tag{2.4}$$

We denote by $w(\boldsymbol{\theta}) = \frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}$ the importance weight, and thereby obtain the importance estimator to $p(\boldsymbol{\theta})$, given by

$$\hat{p}(\boldsymbol{\theta}) = \frac{1}{K}\sum_{k=1}^{K}w(\boldsymbol{\theta})\delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}). \tag{2.5}$$

This estimator requires the target distribution to be known exactly, which is often not the case.

If we can evaluate the target density $p(\boldsymbol{\theta})$ only up to some constant of proportionality, where $p(\boldsymbol{\theta}) = \frac{\bar{p}(\boldsymbol{\theta})}{\int \bar{p}(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}}$, with $\bar{p}(\boldsymbol{\theta})$ being a function we can evaluate, we can then construct the self-normalised importance sampling estimate to $p(\boldsymbol{\theta})$, given by,

$$p(\boldsymbol{\theta}) = \frac{\bar{p}(\boldsymbol{\theta})}{\int \bar{p}(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}} \approx \widetilde{p}(\boldsymbol{\theta}) = \sum_{k=1}^{K}\frac{w(\boldsymbol{\theta}_k)}{\sum_{k=1}^{K}w(\boldsymbol{\theta}_k)}\delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}), \tag{2.6}$$

where $w(\boldsymbol{\theta})$ is now $\frac{\bar{p}(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}$. We can rewrite the estimator $\widetilde{p}(\boldsymbol{\theta})(\boldsymbol{\theta})$ using normalised weights, given by $\widetilde{w}(\boldsymbol{\theta}) = \frac{w(\boldsymbol{\theta})}{\sum_{k=1}^{K}w(\boldsymbol{\theta})}$, as

$$\widetilde{p}(\boldsymbol{\theta}) = \sum_{k=1}^{K}\widetilde{w}(\boldsymbol{\theta}_k)\delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}), \tag{2.7}$$

and approximate expectations of functions of the state using

$$\mathbb{E}_p(f(\boldsymbol{\theta})) \approx \sum_{k=1}^{K} \widetilde{w}(\boldsymbol{\theta}_k) f(\boldsymbol{\theta}_k). \tag{2.8}$$

Importance sampling can be efficient and fast to sample, however it is generally difficult to design the proposal distribution q to yield accurate results [21].

When estimating the posterior parameter distribution $p(\boldsymbol{\theta}|\mathbf{y}_{1:T})$, importance sampling is generally not directly useful, as the distribution of probability mass in the parameter distribution is generally very different from that of standard closed form distributions, and therefore the resulting importance samples are of low quality. Furthermore, the high cost of evaluating the density of $p(\boldsymbol{\theta}|\mathbf{y}_{1:T})$ is such that more expensive methods, such as MCMC, can be used to obtain a superior effective sample size per second for this purpose.

Multiple importance sampling. In order to use importance sampling, we must choose a proposal distribution. This is generally a difficult task, as in order for the effective sample size to consistently be high, the ratio of the densities of the target and the proposal should be close to 1. However, this presents an issue, as we require a distribution that globally approximates the target distribution to construct such a proposal.

In the case that we do not have such a distribution, we can use multiple distributions to leverage local knowledge to create a proposal. Multiple importance sampling (MIS) involves selecting a set of N component distributions, $\psi = \{q_1, q_2, ..., q_N\}$, which together form the proposal distribution. This makes designing the proposal simpler, as each component distribution can capture local behaviour, with their combination better representing the global behaviour of the target distribution than any single component.

Note that multiple importance sampling is not, in general, equivalent to using a mixture distribution as the proposal distribution, with there being many distinct and valid schemes for generating samples from an MIS proposal [22]. One way is to then uniformly sample a distribution q_j by drawing j from a categorical distribution such that $j \sim \text{Cat}([1,...,N],[N^{-1},...,N^{-1}])$, and then sample $\boldsymbol{\theta} \sim q_j$.

Furthermore, there are many valid ways to compute the weight w of a sample from an MIS scheme. Two typical ways are

$$w = \frac{p(\boldsymbol{\theta})}{q_j(\boldsymbol{\theta})}$$

or

$$w = \frac{p(\boldsymbol{\theta})}{N^{-1} \sum_{n=1}^{N} q_n(\boldsymbol{\theta})}.$$

These weighting functions both result in valid estimators, and are independent of the

index sampling method. Note that other weighting functions can be used, and that the proposal distribution set can be weighted such that some distributions are sampled more frequently than others [22].

Adaptive importance sampling. In many cases, the target distribution is largely unknown beforehand, and therefore it is not possible to design a proposal This can occur if, for example, the distribution results from a model with unknown parameters. To counteract this, we can adapt the importance distribution to the target distribution by learning the parameters of the proposal distribution that best represent the target distribution. This is known as adaptive importance sampling (AIS).

In general, AIS proceeds by iteratively sampling the proposal, then updating the proposal parameters. Mathematically, this involves initialising N proposal distributions $\psi_0 = \{q_{\boldsymbol{\phi}_{1,0}}, q_{\boldsymbol{\phi}_{2,0}}, ..., q_{\boldsymbol{\phi}_{N,0}}\}$, with $\phi_{i,j}$ parametrising the jth distribution at the ith iteration. At the ith iteration, we have $\psi_i = \{q_{\boldsymbol{\phi}_{1,1}}, q_{\boldsymbol{\phi}_{2,1}}, ..., q_{\boldsymbol{\phi}_{N,1}}\}$. We then sample K samples from each proposal distribution, yielding $\{\boldsymbol{\theta}_{n,i}^{(k)}\}$ for $k = 1,...,K$, $n = 1,...,N$. We then compute the importance weight for each sample, with $w_{n,i}^{(k)}$ being the weight for ${}_{n\boldsymbol{\theta},i}^{(k)}$. Finally, we update the parameters $\{\boldsymbol{\phi}_{n,i}\}_{n=1}^{N} \rightarrow \{\boldsymbol{\phi}_{n,i+1}\}_{n=1}^{N}$.

An overview of various AIS methods is provided in [23].

Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) allows us to sample from a distribution $p(\boldsymbol{\theta})$ given that we can evaluate it up to a constant of proportionality. In order to sample $p(\boldsymbol{\theta})$, MCMC constructs a Markov chain with $p(\boldsymbol{\theta})$ as its limiting distribution. We then generate samples from this Markov chain, and utilise the samples as we would samples from the distribution of interest. Typically an initial fraction of samples are discarded in order to allow the chain to converge to its limiting distribution. The limiting distribution of a Markov chain is determined by its transition kernel, which we denote by $T(\boldsymbol{\theta}, \boldsymbol{\theta}')$. There are several methods to construct a transition kernel such that the resulting chain admits the target distribution as the limiting distribution. We will present a standard method, Metropolis-Hastings, from which many methods can be developed, such as Hamiltonian Monte Carlo (also known as Hybrid Monte Carlo) [24] and Reversible Jump MCMC [25].

The Metropolis-Hastings algorithm. The Metropolis-Hastings algorithm is method to construct a transition kernel for a desired distribution [26]. This method uses an accept-reject step to correct samples from a proposal distribution to samples of the target distribution. Denote the target distribution by $p(x) = \bar{p}(\boldsymbol{\theta}) / \int \bar{p}(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}$, where we can evaluate $\bar{p}(\boldsymbol{\theta})$ but not $p(\boldsymbol{\theta})$. Furthermore, denote our proposal distribution by $q(\boldsymbol{\theta}' | \boldsymbol{\theta})$, where the value of the new value is conditional on the current value of the

chain. We can then construct the Metropolis-Hastings transition kernel by

$$T(\boldsymbol{\theta},\boldsymbol{\theta}')=\alpha(\boldsymbol{\theta}',\boldsymbol{\theta})q(\boldsymbol{\theta}'|\boldsymbol{\theta}), \tag{2.9}$$

where the Metropolis acceptance probability $\alpha(\boldsymbol{\theta}',\boldsymbol{\theta})$ is given by

$$\alpha(\boldsymbol{\theta}',\boldsymbol{\theta})=\min\left(\frac{q(\boldsymbol{\theta}|\boldsymbol{\theta}')}{q(\boldsymbol{\theta}'|\boldsymbol{\theta})}\frac{p(\boldsymbol{\theta}')}{p(\boldsymbol{\theta})},1\right). \tag{2.10}$$

We note that

$$\frac{p(\boldsymbol{\theta}')}{p(\boldsymbol{\theta})}=\frac{\bar{p}(\boldsymbol{\theta}')/\int\bar{p}(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}}{\bar{p}(\boldsymbol{\theta})/\int\bar{p}(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}}=\frac{\bar{p}(\boldsymbol{\theta}')}{\bar{p}(\boldsymbol{\theta})}, \tag{2.11}$$

and therefore we do not need to know the normalising constant of $p(\boldsymbol{\theta})$. We can now construct the Metropolis-Hastings algorithm, which we give in Alg. 1.

Algorithm 1 Metropolis-Hastings Markov chain Monte Carlo

Input: Proposal distribution $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$, number of samples K, initial value $\boldsymbol{\theta}_0$.
Output: Samples $\{\boldsymbol{\theta}_k\}_{k=1}^K$ from a Markov chain with limiting distribution $p(\boldsymbol{\theta})$.
for $k=1,...,K$ **do**
    Draw $\boldsymbol{\theta}'\sim q(\boldsymbol{\theta}'|\boldsymbol{\theta}_{k-1})$.
    Draw $u\sim\mathcal{U}(0,1)$.
    if $u\leq\alpha(\boldsymbol{\theta}',\boldsymbol{\theta}_{k-1})$ **then**
        Set $\boldsymbol{\theta}_k:=\boldsymbol{\theta}'$.
    else
        Set $\boldsymbol{\theta}_k:=\boldsymbol{\theta}_{k-1}$.
    end if
end for

Metropolis-Hastings MCMC, given in Alg. 1, gives us a method to sample from $p(\boldsymbol{\theta})$ given that we have a proposal distribution $q(\boldsymbol{\theta})$, however we do not know how to construct $q(\boldsymbol{\theta}'|\boldsymbol{\theta}_{k-1})$. One method used in Bayesian modelling, where $p(\boldsymbol{\theta})$ is the posterior distribution of some parameter, is to sample from the prior distribution. This can be efficient when the prior is highly informative, but often uninformative or diffuse priors are selected, or the data differs significantly from the prior, and therefore prior sampling is ineffective. We can mitigate this by utilising previous values of the sample chain to inform the distribution of new samples. A simple method implementing this is random walk Metropolis-Hastings (RWMH).

RWMH proposes samples from a distribution informed by the current value of the chain. An example of this is $q(\boldsymbol{\theta}'|\boldsymbol{\theta})=\mathcal{N}(\boldsymbol{\theta},\boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ is some selected covariance matrix. Furthermore, it is common that the distribution chosen is symmetrical, that is $q(\boldsymbol{\theta}'|\boldsymbol{\theta})=q(\boldsymbol{\theta}|\boldsymbol{\theta}')$, as is the case with the Gaussian given above. This simplifies the

Metropolis acceptance probability to

$$\alpha(\boldsymbol{\theta}',\boldsymbol{\theta}) = \min\left(\frac{q(\boldsymbol{\theta}|\boldsymbol{\theta}')}{q(\boldsymbol{\theta}'|\boldsymbol{\theta})}\frac{p(\boldsymbol{\theta}')}{p(\boldsymbol{\theta})},1\right) \tag{2.12}$$

$$= \min\left(\frac{p(\boldsymbol{\theta}')}{p(\boldsymbol{\theta})},1\right), \tag{2.13}$$

which, if q is such that the density is expensive to evaluate, can save significant computational time. It is clear that the design of the proposal distribution $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ is critical to the performance of a Metropolis-Hastings sampler. Generally, we want a proposal distribution that leads to good mixing, which occurs when the Markov chain rapidly converges to the desired distribution and has low auto-correlation between samples, and therefore explore the sample space well. We can always get a high acceptance probability using RWMH by setting the proposal $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ such that the bulk of the density occurs in a small region around $\boldsymbol{\theta}$, however this will converge extremely slowly and will not explore the sample space efficiently. Conversely, a proposal with a very diffuse density will also not explore the sample space, as the acceptance probability will be very low, and so the proposed samples will rarely be accepted. A rule of thumb for RWMH is to target an acceptance rate of 0.234 [27], although the conditions for this to be truly optimal are often not satisfied. We utilise RWMH alongside a model sampling scheme to design a sparsity promoting parameter estimation scheme for the transition matrix of a linear-Gaussian state-space model in chapter 3.

2.1.2 Optimisation-based pointwise estimators

An alternative approach to estimating a parameter of interest is finding the value that optimises a specific function, often referred to as a loss function. This approach does not generally yield distributional estimates, and therefore does not natively propagate uncertainty, as is the case with MCMC and importance sampling. However, often these approaches are either computationally infeasible due to the cost of evaluating the target density, or there is no method to design a usable proposal or sampling scheme. In these cases, it is typical to use a pointwise estimator optimising some criterion.

Perhaps the most common example of such an estimator is the Maximum likelihood estimator (MLE). When estimating the parameter $\boldsymbol{\theta}$ given the data $\mathbf{y}_{1:T}$, the MLE, denoted $\hat{\boldsymbol{\theta}}_{\text{MLE}}$, is obtained by maximising the likelihood of the data

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ p(\mathbf{y}_{1:T}|\boldsymbol{\theta}),$$

where $p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$ represents the likelihood of the data. This estimator does not incor-

porate prior information, and so can be used when there is either no prior information or when an improper diffuse prior is chosen. In this thesis we typically use it as a benchmark.

A similar estimator to the MLE is the maximum-a-posteriori (MAP) estimator. Instead of maximising the data likelihood, the MAP estimator maximises the posterior density of the parameter, hence:

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\ p(\boldsymbol{\theta})p(\mathbf{y}_{1:T}|\boldsymbol{\theta}),$$

where $p(\boldsymbol{\theta})$ is the prior distribution associated with $\boldsymbol{\theta}$. The MAP estimator allows us to incorporate prior beliefs about the parameter's value and structure through $p(\boldsymbol{\theta})$. Notably, many penalty schemes can be interpreted as optimising parameters under a prior distribution [28, 29]. Sparse estimation schemes can often be interpreted as computing a MAP under a prior, where the prior in some way promotes sparsity in the estimated parameters $\hat{\boldsymbol{\theta}}_{\text{MAP}}$ [16]. One well known sparsity promoting prior is the Laplace prior, which the the Bayesian equivalent of the frequentist LASSO [28, 29], and which is equivalent to penalising by the $L1$ norm of $\boldsymbol{\theta}$ when performing optimisation.

2.2 State-space models

One framework for analysing time series data is state-space models (SSMs). SSMs model the evolution of a latent state in time, which we do not observe. Instead, we obtain a series of related noisy observations. A SSM is hence completely defined by a state transition process an observation process, and an initial distribution for the hidden state [5].

The general state-space model, in absence of control inputs, is given by

$$\begin{aligned} \mathbf{x}_t &\sim p(\mathbf{x}_t|\mathbf{x}_{t-1};\boldsymbol{\theta}), \\ \mathbf{y}_t &\sim p(\mathbf{y}_t|\mathbf{x}_t;\boldsymbol{\theta}), \end{aligned} \tag{2.14}$$

where $t = 1,...,T$ denotes discrete time, $\mathbf{x}_t \in \mathbb{R}^{N_x}$ is the state of the system at time t, $\mathbf{y}_t \in \mathbb{R}^{N_y}$ is the observation at time t, $p(\mathbf{x}_t|\mathbf{x}_{t-1};\boldsymbol{\theta})$ is the conditional density of the state $\mathbf{x}_t$ given $\mathbf{x}_{t-1}$, $p(\mathbf{y}_t|\mathbf{x}_t;\boldsymbol{\theta})$ is the conditional density of the observation $\mathbf{y}_t$ given the hidden state $\mathbf{x}_t$, and $\boldsymbol{\theta}$ is a set of parameters. The initial value of the state is distributed as $\mathbf{x}_0 \sim p(\mathbf{x}_0|\boldsymbol{\theta})$. The state sequence $\{\mathbf{x}_t\}_{t\geq 0}$ is hidden, whilst the related sequence of $\{\mathbf{y}_t\}_{t\geq 1}$ is observed.

We are often interested in inferring $p(\mathbf{x}_t|\mathbf{y}_{1:t})$, the so-called filtering distribution, or $p(\mathbf{x}_t|\mathbf{y}_{1:T})$, the smoothing distribution. The filtering distribution infers $\mathbf{x}_t$ using

only observations collected until time t, whereas the smoothing distribution infers $\mathbf{x}_t$ using all observations including those collected after time t [5]. Consequently, filtering is suitable for online, real-time use, whereas smoothing can only be used offline, after all data has been gathered.

From Eq. (2.14), we observe that

1. the state transition process is Markov, i.e. the state at time t, $\mathbf{x}_t$, depends only on the state at time $t-1$, $\mathbf{x}_{t-1}$

2. the state transition process is not informed by the observation process

3. the observation $\mathbf{y}_t$ depends only on the current state $\mathbf{x}_t$, and not on previous values of the observation or state.

Note that the latent state $\mathbf{x}_t$ and the observation $\mathbf{y}_t$ can be in discrete or continuous space.

2.2.1 Linear-Gaussian state-space models and the Kalman filter

The linear-Gaussian state-space model (LGSSM) is a special case of the general state-space model, and is a particularly useful case, as it admits both an efficient closed-form solution to the filtering problem, and can represent many real-world problems [5]. The LGSSM, in absence of control inputs, is given by

$$\begin{aligned} \mathbf{x}_t &= \mathbf{A}\mathbf{x}_{t-1} + \mathbf{q}_t, \\ \mathbf{y}_t &= \mathbf{H}\mathbf{x}_t + \mathbf{r}_t, \end{aligned} \tag{2.15}$$

for $t \in \{1,\dots,T\}$, where $\mathbf{x}_t \in \mathbb{R}^{N_x}$ is the hidden state with associated observation $\mathbf{y}_t \in \mathbb{R}^{N_y}$ at time t, $\mathbf{A} \in \mathbb{R}^{N_x \times N_x}$ is the state transition matrix, $\mathbf{H} \in \mathbb{R}^{N_y \times N_x}$ is the observation matrix, $\mathbf{q}_t \sim \mathcal{N}(\mathbf{0},\mathbf{Q})$ is the state noise, and $\mathbf{r}_t \sim \mathcal{N}(\mathbf{0},\mathbf{R})$ is the observation noise, with $\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})$ denoting the normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. The state prior is $\mathbf{x}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0,\boldsymbol{\Sigma}_0)$, with $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ known.

For the LGSSM, we can obtain the optimal solution to the filtering equations by the Kalman filter [30], given by Alg. 2.

Algorithm 2 Kalman Filter

Input: Parameters $\mathbf{A},\mathbf{Q},\mathbf{H},\mathbf{R},\boldsymbol{\mu}_0,\boldsymbol{\Sigma}_0$, observations $\mathbf{y}_{1:T}$.

Output: Filtered state means and covariances $\{\boldsymbol{\mu}_t,\boldsymbol{\Sigma}_t\}_{t=1}^{T}$.

for $t\in\{1,...,T\}$ **do**

Prediction step:

$$\hat{\boldsymbol{\mu}}_t=\mathbf{A}\boldsymbol{\mu}_{t-1}$$
$$\hat{\boldsymbol{\Sigma}}_t=\mathbf{A}\boldsymbol{\Sigma}_{t-1}\mathbf{A}^\intercal+\mathbf{Q}$$

Update step:

$$\boldsymbol{\nu}_t=\mathbf{H}\hat{\boldsymbol{\mu}}_t$$
$$\mathbf{v}_t=\mathbf{y}_t-\boldsymbol{\nu}_t$$
$$\mathbf{S}_t=\mathbf{H}\hat{\boldsymbol{\Sigma}}_t\mathbf{H}^\intercal+\mathbf{R}$$
$$\mathbf{K}_t=\hat{\boldsymbol{\Sigma}}_t\mathbf{H}^\intercal\mathbf{S}_t^{-1}$$
$$\boldsymbol{\mu}_t=\hat{\boldsymbol{\mu}}_t+\mathbf{K}_t\mathbf{v}_t$$
$$\boldsymbol{\Sigma}_t=\hat{\boldsymbol{\Sigma}}_t-\mathbf{K}_t\mathbf{S}_t\mathbf{K}_t^\intercal$$

end for

Note that the likelihood of the observation series is given by

$$p(\mathbf{y}_{1:T}|\boldsymbol{\theta})=\prod_{t=1}^{T}p(\mathbf{y}_t|\mathbf{y}_{1:t-1},\boldsymbol{\theta}), \tag{2.16}$$

where

$$p(\mathbf{y}_t|\mathbf{y}_{1:t-1},\boldsymbol{\theta})=\mathcal{N}(\boldsymbol{\nu}_t,\mathbf{S}_t), \tag{2.17}$$

with $\boldsymbol{\nu}_t,\mathbf{S}_t$ as Alg. 2. In the LGSSM, the $\boldsymbol{\theta}$ parameter can inform the value of any (or all) of $\mathbf{A},\mathbf{H},\mathbf{R},\mathbf{Q},\boldsymbol{\Sigma}_0$, and $\boldsymbol{\mu}_0$. For example, if a method were inferring both the transition matrix and the state noise covariance matrix, we would have $\boldsymbol{\theta}=\{\mathbf{A},\mathbf{Q}\}$.

2.2.2 Particle filtering

The general state-space model can be described by Eq. (2.14). Filtering methods aim at estimating the hidden state at time t, denoted $\mathbf{x}_t$, typically utilising the posterior density function of the state conditional on the observations up to time t, denoted $\mathbf{y}_{1:t}$. Particle filters approximate this density, $p(\mathbf{x}_t|\mathbf{y}_{1:t};\boldsymbol{\theta})$, using a set of K Monte Carlo samples (particles) and their associated (normalised) weights, $\{\mathbf{x}_{1:T}^{(k)},\overline{w}_{1:T}^{(k)}\}_{k=1}^{K}$. The posterior density can then be approximated by

$$p(\mathbf{x}_t|\mathbf{y}_{1:t};\boldsymbol{\theta})\approx\sum_{k=1}^{K}\overline{w}_t^{(k)}\delta_{\mathbf{x}_t^{(k)}}, \tag{2.18}$$

with functions of the state having their expectation approximated by

$$\mathbb{E}(f(\mathbf{x}_t)) \approx \sum_{k=1}^{K} \overline{w}_t^{(k)} f(\mathbf{x}_t^{(k)}). \tag{2.19}$$

A commonly used particle filtering method is the sequential importance resampling algorithm [31, 32], given in Alg. 3. At every time-step t, the K particles and normalised weights, $\{\mathbf{x}_{1:T}^{(k)}, \overline{w}_{1:T}^{(k)}\}_{k=1}^{K}$, are calculated. First, we perform the resampling step (line 7), which generates K samples, sampling $\mathbf{x}_{t-1}^{(k)}$, $k=1,...,K$, with probability $\overline{w}_{t-1}^{(k)}$. The resampling step is vital to avoid the degeneracy of the filter by ensuring the diversity in the particle set and obtaining a more accurate approximation to the posterior distribution, $p(\mathbf{x}_t|\mathbf{y}_{1:t};\boldsymbol{\theta})$. Next, K particles $\mathbf{x}_t^{(k)}$, $k=1,...,K$, are drawn from the proposal distribution $\pi(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{y}_t;\boldsymbol{\theta})$ (line 10). This is analogous to the prediction step of the Kalman filter, although it potentially utilises observation information, which is not the case in the Kalman filter. Finally, we incorporate the observation $\mathbf{y}_t$ and the system dynamics, both of which are utilised when computing the particle weights, given by $w_t^{(k)}$, $k=1,...,K$, in line 12, and the normalised weights (line 13). These weights, in conjunction with the particles, form an importance sampling approximation to the filtering distribution $p(\mathbf{x}_t|\mathbf{y}_{1:T};\boldsymbol{\theta})$, where the sample $\mathbf{x}_t^{(k)}$ has weight $\overline{w}_t^{(k)}$. This step is analogous to the update step of the Kalman filter.

Algorithm 3 Sequential importance resampling (SIR) particle filter

1: **Input:** Observations $\mathbf{y}_{1:T}$, parameters $\boldsymbol{\theta}$
2: **Output:** Hidden state estimates $\mathbf{x}_{1:T}$, particle weights $w_{1:T}$
3: Draw $\mathbf{x}_0^{(k)} \sim p(\mathbf{x}_0|\boldsymbol{\theta})$, for $k=1,...,K$
4: Set $\overline{w}_0^{(k)} = 1/K$, for $k=1,...,K$
5: **for** $t=1,...,T$ and $k=1,...,K$ **do**
6: *Resampling step:*
7: Sample $a_t^{(k)} \sim \text{Categorical}(\overline{w}_{t-1})$
8: Set $\overline{w}_{t-1}^{(k)} = 1/K$
9: *Prediction step:*
10: Sample $\mathbf{x}_t^{(k)} \sim \pi(\mathbf{x}_t|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)},\mathbf{y}_t;\boldsymbol{\theta})$
11: *Update step:*
12: Compute $w_t^{(k)} = \frac{p(\mathbf{y}_t|\mathbf{x}_t^{(k)};\boldsymbol{\theta})p(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)};\boldsymbol{\theta})}{\pi(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)},\mathbf{y}_t;\boldsymbol{\theta})}$
13: Compute $\overline{w}_t^{(k)} = w_t^{(k)} / \sum_{i=1}^{K} w_t^{(i)}$
14: **end for**

In order to utilise the particle filter, one must determine the proposal distribution, as this is not a component of the state-space model for which estimation is being performed. The proposal distribution is particularly tricky to design, as it is unique

to the particle filter, with the other two distributions being exactly as in the definition of the state-space model (Eq. (2.14)). In some cases an optimal proposal distribution can be derived analytically, such as in the LGSSM where $p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{y}_t)$ can be derived using the properties of the multivariate Gaussian distribution, however in the general case the proposal distribution weighs computational complexity against statistical performance. One such proposal distribution is the bootstrap proposal [32], where the particles are sampled from the state transition distribution. This is both simple, and results in an analytic simplification of the weighting term, which allows both faster computation and for the bootstrap proposal to be applied in situations where the transition dynamics can be sampled, but do not admit a density, e.g., when the transition is derived from a general SDE. However, the bootstrap proposal is often not very statistically efficient. Proposals can be designed to utilise the observation information, such as in the auxiliary particle filter [12, 33]. However, these schemes are generally more expensive per iteration, as they require evaluating more densities.

Finally, note that the particle filter, just as the Kalman filter, requires the parameters of the state-space model to be known in order to be evaluated, which is not in general the case. We will now discuss several methods for learning parameters in the state-space model.

2.3 Parameter estimation in state-space models

As in practice the parameters of a SSM are in general unknown, we often require to estimate the $\boldsymbol{\theta}$ parameters of a state-space model before we can perform filtering. We will now describe several approaches to estimating the unknown parameter $\boldsymbol{\theta}$, allowing us to infer the hidden state of the system via the filtering equations.

2.3.1 Posterior distributions and parameter densities in state-space models

To infer the model parameters, we target the posterior distribution of the parameter, given by

$$p(\boldsymbol{\theta}|\mathbf{y}_{1:T}) \propto p(\boldsymbol{\theta})p(\mathbf{y}_{1:T}|\boldsymbol{\theta}), \tag{2.20}$$

with $p(\boldsymbol{\theta})$ the prior distribution of $\boldsymbol{\theta}$, and where we can compute $p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$ via

$$p(\mathbf{y}_{1:T}|\boldsymbol{\theta}) = \prod_{t=1}^{T} p(\mathbf{y}_t|\mathbf{y}_{1:t-1},\boldsymbol{\theta}), \tag{2.21}$$

with $p(\mathbf{y}_1|\mathbf{y}_{1:0},\boldsymbol{\theta}) := p(\mathbf{y}_1|\boldsymbol{\theta})$, and

$$p(\mathbf{y}_t|\mathbf{y}_{1:t-1};\boldsymbol{\theta}) = \int p(\mathbf{y}_t|\mathbf{x}_t,\boldsymbol{\theta})p(\mathbf{x}_t|\mathbf{y}_{1:t-1},\boldsymbol{\theta})\mathrm{d}\mathbf{x}_t, \tag{2.22}$$

which obeys the recursion

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{y}_{1:t-1},\boldsymbol{\theta}) &= \int p(\mathbf{x}_t|\mathbf{x}_{t-1},\boldsymbol{\theta})p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1},\boldsymbol{\theta})\mathrm{d}\mathbf{x}_{t-1}, \\ p(\mathbf{x}_t|\mathbf{y}_t,\boldsymbol{\theta}) &= \frac{p(\mathbf{y}_t|\mathbf{x}_t,\boldsymbol{\theta})p(\mathbf{x}_t|\mathbf{y}_{1:t-1},\boldsymbol{\theta})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1},\boldsymbol{\theta})}. \end{aligned} \tag{2.23}$$

Note that Eq. (2.23) are simply the Bayesian filtering equations, with the first line giving the prediction step, and the second line the update step [5].

Further, note that, given the output of the the particle filter (Alg. 3), we can estimate the posterior density of the observation series by the importance estimate

$$p(\boldsymbol{\theta}|\mathbf{y}_{1:T}) \propto p(\boldsymbol{\theta})p(\mathbf{y}_{1:T}|\boldsymbol{\theta}) \approx \hat{p}(\boldsymbol{\theta}|\mathbf{y}_{1:T}) = p(\boldsymbol{\theta})\prod_{t=1}^{T}\left(\frac{1}{N}\sum_{k=1}^{K} w_t^{(k)}\right). \tag{2.24}$$

There are many methods to estimate parameters $\boldsymbol{\theta}$ given its posterior density function $p(\boldsymbol{\theta}|\mathbf{y}_{1:T})$. We can broadly classify these parameter estimation methods as point estimation methods and distributional methods. Point estimation methods provide a single estimate that is, in some defined way, the optimal value. An example of a point estimation method is the maximum-a-posteriori (MAP) estimator, that defines the optimal value of $\boldsymbol{\theta}$ as the one that maximises the posterior density $p(\boldsymbol{\theta}|\mathbf{y}_{1:T})$. The method proposed in this work yields a MAP estimator.

In the case of a linear-Gaussian state-space model, we have simple iterative methods for the MLE of each parameter [5], and efficient methods for obtaining the MAP estimator for sparsity promoting priors for the transition matrix [16] and the state covariance matrix [34]. Distributional methods estimate the posterior distribution of the parameter, with common methods being importance sampling [18], Markov chain Monte Carlo [10], and variational inference [20]. For the linear-Gaussian state-space model, we can utilise reversible jump Markov chain Monte Carlo to obtain an estimate of the distribution of sparsity [35], however no similar method exists for the general state-space model. Our proposed method in this work is a point estimation method.

For general SSMs, the posterior density of the parameter $\boldsymbol{\theta}$, necessary to compute the posterior and hence to design a procedure to maximise it, cannot be obtained in closed form. We can estimate it using the particle filter, by the Monte Carlo estimate given in Eq. (2.24). where $w_t^{(k)}$ and $\widetilde{w}_{t-1}^{(k)}$ are the weights of the particle filter in Alg. 3 [5, Chapter 12]. Note that the weights are dependent on the parameter $\boldsymbol{\theta}$ through

their computation in Alg. 3. Using these weights, we construct an estimate of the log posterior density by

$$\begin{aligned}\log(p(\boldsymbol{\theta}|\mathbf{y}_{1:T})) &= \log(p(\boldsymbol{\theta})) + \log(p(\mathbf{y}_{1:T}|\boldsymbol{\theta})) + c, \\ &\approx \log(p(\boldsymbol{\theta})) + \sum_{t=1}^{T} \log\left(\sum_{k=1}^{K} w_t^{(k)} \widetilde{w}_{t-1}^{(k)}\right) + c,\end{aligned} \tag{2.25}$$

where c comes from the proportionality in Eq. (2.24), and we note that if resampling occurred at time $t-1$, which is always the case if following Alg. 3, we have $\widetilde{w}_{t-1} = 1/K$. Note that, in practice, log-weights are used for numerical stability, and we compute $\sum_{k=1}^{K} w_t^{(k)} \widetilde{w}_{t-1}^{(k)}$ using a logsumexp reparametrisation.

2.3.2 Parameter optimisation

In the linear-Gaussian state-space model, we can evaluate $p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$ and its gradients exactly for all parameters of the model [5]. Therefore, we can use first order gradient optimisation schemes to obtain point estimates of the model parameters. Note that if improper flat priors are used, then closed form solutions for conditional MLEs exist for all parameters of the LGSSM. Several efficient parameter estimation schemes for the linear-Gaussian state-space model are presented in [11] utilising the EM algorithm.

However, in the general state-space model when using the particle filter, $p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$ must be stochastically approximated through importance weights, and therefore both does not admit gradients, and is a noisy estimate. Therefore gradient-based optimisation schemes cannot be directly applied, and most gradient-free methods assume properties of the optimised function that are violated due to the stochastic nature of the estimated posterior density. Furthermore, the posterior density of a general state-space model is typically more complex than that of a linear-Gaussian state-space model, and therefore it is more difficult to perform parameter estimation regardless of the stochasticity of the density estimate.

However, recent advances in particle filtering have introduced schemes that allow gradient propagation through particle filters [36, 37]. These developments have expanded the range of optimisation techniques available for general state-space models. In particular, gradient methods such as Adam [38] can be used, which are robust to noisy density estimates by construction, as they are designed for use in deep learning, where batched learning, and hence stochastic losses and gradients, is standard.

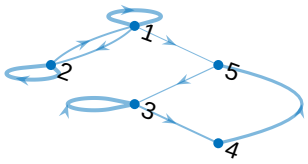
2.3.3 Sparse parameter estimation

When fitting and designing statistical models, the presence of sparsity in parameters is often desirable, as it reduces the number of relevant variables thus easing interpretation and simplifying inference. Furthermore, real systems are often made up of several interacting dense blocks that, when taken as a whole, exhibit complex dynamics [14]. Sparse estimation methods allow for this structure to be recovered, resulting in estimates that can reflect the structure of the underlying system. Sparsity is ubiquitous within signal processing, with signal decomposition into a sparse combination of components being very common [39, 40, 41], which can be parameterised via model parameters. Furthermore, within signal processing, there exist a number of existing sparse Bayesian methods, such as [42, 43, 44, 45], although these do not operate within the paradigm of state-space modelling.

There are several approaches to estimate model parameters such that sparsity is promoted. One approach may be to construct many models with unique combinations of sparse and dense elements, fit all of these models, and then select the best model according to some criteria (see [46, 47, 48] for examples). This approach is conceptually sound, but computationally expensive for even a small number of parameters p, as 2^p models would need to be fitted in order to obtain density estimates, or other goodness-of-fit metrics. Another approach is to estimate the model parameters under a sparsity inducing penalty, with the classic example of such a penalty being the LASSO [28, 29]. This approach, commonly called regularisation, allows for only one model to be fitted rather than many, but increases the computational complexity of fitting the model. This single regularised estimate is, in most cases, more expensive to compute than fitting a non-regularised estimate as required by the previous approach, but this cost is typically far less than the cumulative cost of all required estimates in the previous approach. Regularisation is a common way to obtain sparse estimates, and can be extended to Bayesian modelling and estimation in the form of sparsity inducing priors [29, 49].

In LGSSMs, a sparse estimate of the transition matrix $\mathbf{A}$ can be interpreted as the adjacency matrix of a weighed directed graph $\mathcal{G}$, with the nodes being the state elements, and the edges the corresponding elements of $\mathbf{A}$ [50, 16]. We illustrate this with an example in Fig. 2.1.

The graph $\mathcal{G}$ thus encodes the linear, between-step relationships of state elements. Under this graphical interpretation, A_{ij} being non-zero implies that knowledge of the jth element of the state improves the prediction of the ith value. The presence of an edge from node j to node i implies a Granger-causal relationship between the state elements, as knowledge of the past values of the jth element at time t improves the prediction of the ith element at time $t+1$, implying a Granger-causal relationship [51].



$$\mathbf{A} = \begin{pmatrix} 1 & 0.5 & 0 & 0 & 0 \\ 1 & -0.5 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -0.3 \\ 0 & 0 & 0.1 & 0 & 0 \\ 0.1 & 0 & 0 & -0.8 & 0 \end{pmatrix}$$

Figure 2.1: Example of a weighted directed graph associated with matrix. The edge thickness corresponds to the magnitude of the weight.

This interpretation can be extended to general state-space models. In this case, $\mathbf{A}$ is a binary matrix, with $A_{ij} = 1$ if the transition kernel for state i has a dependence on state j, and 0 otherwise. We then construct the graph $\mathcal{G}$ from $\mathbf{A}$, however, the graph is not weighted. This construction still implies state j Granger-causes state i if $A_{ij} = 1$, however there is no way to compare the magnitude of the effect without further information, as the interaction term may be more significant in some regions of the state than others.

2.3.4 Particle MCMC

In the general state-space model, we cannot evaluate the parameter posterior $p(\boldsymbol{\theta}|\mathbf{y}_{1:T})$ exactly using the particle filter. Instead, we rely on a stochastic estimate of the posterior, given by Eq. (2.24). It has been shown [10] that, under weak assumptions, the particle filter estimate of the parameter posterior density can be used to construct an MCMC proposal with which the Markov chain converges to the true parameter posterior. Several particle MCMC methods have been proposed, with one such method being particle marginal Metropolis-Hastings (PMMH).

Algorithm 4 Particle marginal Metropolis-Hastings

Input: Proposal distribution $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$, number of samples K, initial value $\boldsymbol{\theta}_0$, observations $\mathbf{y}_{1:T}$, prior distribution $p(\boldsymbol{\theta})$.
Output: K Samples from $p(\boldsymbol{\theta}|\mathbf{y}_{1:T})$ $\{\boldsymbol{\theta}_k\}_{k=1}^K$.
for $k = 1,...,K$ **do**
  Draw $\boldsymbol{\theta}' \sim q(\boldsymbol{\theta}'|\boldsymbol{\theta}_{k-1})$.
  Obtain $p(\mathbf{y}_{1:T}|\boldsymbol{\theta}')$ via a particle filter and Eq. (2.24).
  Draw $u \sim \mathcal{U}(0,1)$.
  if $u \leq \min\left(\frac{q(\boldsymbol{\theta}'|\boldsymbol{\theta}_{k-1})}{q(\boldsymbol{\theta}_{k-1}|\boldsymbol{\theta}')} \frac{\hat{p}(\mathbf{y}_{1:T}|\boldsymbol{\theta}')}{\hat{p}(\mathbf{y}_{1:T}|\boldsymbol{\theta}_{k-1})} \frac{p(\boldsymbol{\theta}')}{p(\boldsymbol{\theta}_{k-1})}, 1\right)$ **then**
    Set $\boldsymbol{\theta}_k := \boldsymbol{\theta}'$.
  else
    Set $\boldsymbol{\theta}_k := \boldsymbol{\theta}_{k-1}$.
  end if
end for

PMMH, described in Alg. 4, is a Metropolis-Hastings algorithm utilising the stochastic estimate of the parameter posterior density given by the particle filter. The

Markov chain resulting from this algorithm has the desired stationary distribution of $p(\boldsymbol{\theta}|\mathbf{y}_{1:T})$ [10].

PMMH is described as marginal due to the densities in line 5 of Alg. 4 being a marginal density, as we marginalise the dependence on the particles $\{\mathbf{x}_{0:T}^{(k)}\}_{k=1}^{K}$, hence reducing the dimension of the sampling.

The efficiency of PMMH, as with all Metropolis-based MCMC schemes, hinges on the choice of proposal distribution $q(\boldsymbol{\theta}'|\boldsymbol{\theta}_{k-1})$. In state-space models, which can be very difficult to design owing to the sequential dependency of the hidden state. It is common in practice to utilise the random walk proposal detailed in Section 2.1.1, where, for a Gaussian random walk, $\boldsymbol{\theta}' \sim \mathcal{N}(\boldsymbol{\theta}_{k-1}, \boldsymbol{\Sigma})$ for some selected covariance matrix $\boldsymbol{\Sigma}$. If $\boldsymbol{\theta}$ contains matrices, these are transformed to vectors.

Particle MCMC is computationally expensive, as each iteration requires a complete run of the particle filter, with a sufficiently larger number of particles K to yield an accurate estimate of the parameter posterior $p(\boldsymbol{\theta}|\mathbf{y}_{1:T})$. This gives a relatively expensive algorithm, however there are several advantages to using particle MCMC.

Firstly, as with regular MCMC, we obtain samples of the posterior distribution of the parameter $\boldsymbol{\theta}$, and can therefore estimate arbitrary functions of the parameter with uncertainty quantification. Second, the method works on models of many forms, and does not require gradient information, which is not readily available through the standard particle filter. Finally, the method is well tested, and is typically the baseline against which general parameter estimation methods are compared against when applied to general state-space models. Note that, in linear-Gaussian state-space models, we do not need to perform particle MCMC, and instead can perform MCMC utilising the parameter density recovered from the Kalman filter, not requiring a particle approximation of the posterior density.

Chapter 3

Sparse Bayesian Estimation of Transition Parameters in Linear-Gaussian State-Space Models

This chapter is based on the papers [52, 35].

3.1 Introduction

When estimating the parameters of a state-space model, it is crucial that the estimates reflect the structure of the underlying system. For instance, in real-world systems, the underlying dynamics are often composed of simple units, with each unit interacting with only a subset of the overall system, but when observed together these units exhibit complex behaviour [14]. This structure can be recovered by promoting sparsity in the parameter estimates. In addition to better representing the underlying system, sparse parameter estimates have several other advantages. By promoting sparsity uninformative terms are removed from the inference, thereby reducing the dimension of the parameter space, improving model interpretability. Furthermore, parameter sparsity allows us to infer the connectivity of the state space [50], which is useful in several applications, such as biology [53, 54], social networks [55], and neuroscience [56]. In state-space models the sparsity structure can be represented as a directed graph, with the nodes signifying the state variables, and edges signifying between connections between variables. In the LGSSM specifically, this graph can be represented by an adjacency matrix with identical sparsity to the transition matrix.

In this chapter, we propose a novel method for sampling the posterior distribution of the transition matrix of a linear-Gaussian state-space model in a fully Bayesian manner, whilst also obtaining true sparsity. Standard methods for sampling a con-

tinuous distribution do not allow for true sparsity, as the sampling probability of any given value is zero, and hence a true zero is almost never sampled. However, by interpreting sparsity constraints as encoding different models, we can utilise reversible jump Markov chain Monte Carlo (RJMCMC) to simultaneously sample the model space (and therefore the sparsity structure) and the parameter space (the values of the elements of the transition matrix). We design both specific transition kernels and parameter rejuvenation schemes, so the algorithm can efficiently explore both the parameter space, and the sparsity of the parameter in a hierarchical fashion. As RJMCMC is itself a modified Metropolis-Hastings method, the solid theoretical guarantees of both precursors are inherited by our proposed algorithm, such as the asymptotic correctness of distribution of both model and parameter [25].

Our method outperforms state-of-the-art methods for sparse parameter estimation in linear-Gaussian state-space models in several numerical experiments. We test our method in a synthetic example with dimension up to 144 in the parameter space. In this example, a total of 2^{144} models are to be explored (i.e., the number of different sparsity levels). Then, we run a numerical example with real data of time series measuring daily temperature. The novel probabilistic graphical interpretation allows recovery of a probabilistic (Granger) causal graph, showcasing the large impact that this novel approach can have in relevant applications of science and engineering. The model transition kernels used by our method are designed to allow the exploitation of sparse structures that are common in many applications, which reduces the computational complexity of the resulting (sparse) models once the transition matrix has been estimated (see for instance [14]). Our method retains strong theoretical guarantees, inherited from the underlying Metropolis-Hastings method, thanks to careful design of the transitions kernels, e.g., keeping the convergence properties of the algorithm. Extending our methodology to parameters other than the transition matrix is readily possible. In particular, we make explicit both a model and parameter proposal for extension to the state covariance parameter $\mathbf{Q}$.

Contributions. The main contributions of this chapter are summarised as follows:

- The proposed algorithm is the first method to estimate probabilistically the state transition matrix in LGSSMs (i.e, treating $\mathbf{A}$ as a random variable rather than a fixed unknown) under sparsity constraints. This is achieved by taking $\mathbf{A}$ to be a random variable, and sampling the posterior distribution $p(\mathbf{A}|\mathbf{y}_{1:T})$ under a unique interpretation of sparsity as a model. This new capability allows for powerful inference to be performed with enhanced interpretability in this relevant model, e.g., the construction of a probabilistic Granger causal network mapping the state space, which was not possible before.

- The proposed method is the first method to quantify the uncertainty associated with the occurrence of sparsity in SSMs, e.g., in the probability of sparsity occurring in a given element of the transition matrix. This capability is unique among parameter estimation techniques in this field.

- Our method proposes an interpretation of sparsity as a model, allowing the use of RJMCMC for sparsity detection in state-space modelling. This is the first RJMCMC method to have been applied to sparsity recovery in state-space models, probably because RJMCMC methods require careful design of several parts of the algorithm, especially for high dimensional parameter spaces as is the case for the matrix valued parameters of the LGSSM.

Structure. In Section 3.2 we present the components of the problem and present some of the underlying algorithms, as well as the notation we will use. Section 3.4 presents the method, with further elucidation in Section 3.5. We present several challenging numerical experiments in Section 3.6, showcasing the performance of our method, and comparing to a recent method with similar goals. We provide some concluding remarks in Section 3.7.

3.2 Background

3.2.1 Considered state-space model

In this chapter we consider the additive linear-Gaussian state-space model (LGSSM), given by

$$\begin{aligned} \mathbf{x}_t &= \mathbf{A}\mathbf{x}_{t-1} + \mathbf{q}_t, \\ \mathbf{y}_t &= \mathbf{H}\mathbf{x}_t + \mathbf{r}_t, \end{aligned} \tag{3.1}$$

for $t = 1,\ldots,T$, where $\mathbf{x}_t \in \mathbb{R}^{N_x}$ is the hidden state with associated observation $\mathbf{y}_t \in \mathbb{R}^{N_y}$ at time t, $\mathbf{A} \in \mathbb{R}^{N_x \times N_x}$ is the state transition matrix, $\mathbf{H} \in \mathbb{R}^{N_y \times N_x}$ is the observation matrix, $\mathbf{q}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ is the state noise, and $\mathbf{r}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is the observation noise. The state prior is $\mathbf{x}_0 \sim \mathcal{N}(\bar{\mathbf{x}}_0, \mathbf{P}_0)$, with $\bar{\mathbf{x}}_0$ and $\mathbf{P}_0$ known. We assume that the model parameters remain fixed.

A common task in state-space modelling is the estimation of the series of $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ for $t \in \{1,\ldots,T\}$, also known as the filtering distributions. In the case of the LGSSM, these distributions are obtained exactly via the Kalman filter equations [5, 30]. The linear-Gaussian assumption is not overly restrictive, as many systems can be approximated via linearisation, and for continuous problems Gaussian noises are very common.

3.2.2 Reversible jump Markov chain Monte Carlo

Reversible jump Markov chain Monte Carlo (RJMCMC) was proposed as a method for Bayesian model selection [25], and has since seen use in fields such as ecology [57], Gaussian mixture modelling [58], and hidden Markov modelling [59]. RJMCMC has even been applied within the realm of signal processing, with some relevant references being [60, 61, 62]. However, RJMCMC has not been applied to the estimation the sparsity of model parameters within signal processing.

RJMCMC is an extension of the Metropolis-Hastings algorithm that allows for the sampling of a discrete model space, and thus the inclusion of many models within a single sampling chain. RJMCMC is a hierarchical sampler, with an upper layer sampling the models, and a lower layer sampling the posterior distribution of the parameters within the model. This hierarchy allows the use of standard MCMC methods for the lower layer, with the difficulty coming in designing the upper layer [58].

RJMCMC traverses the model space via transition kernels between pairs of models, with the jumps occurring probabilistically. This lends the model space an interpretation as a directed graph, with nodes representing the models and edges representing the jumps between models. Let $\Theta^{(i)}$ be the parameter space associated with model $M^{(i)}$, and $\boldsymbol{\theta}^{(i)} \in \Theta^{(i)}$ an associated realisation of the model parameters. Denote by $\pi_{i,j}$ the probability of jumping from model $M^{(i)}$ to $M^{(j)}$. Note that $\pi_{i,j}$ is zero if and only if $\pi_{j,i}$ is zero. Let $M^{(1)}$ be the current model, and $M^{(2)}$ be a candidate model. In order to construct a Markov kernel for the transition between models, a symmetry constraint is imposed, i.e., if it is possible to jump from $M^{(1)}$ to $M^{(2)}$, it must also be possible to jump from $M^{(2)}$ to $M^{(1)}$ [63]. In general however, the dimension of the parameter spaces is not equal, hence it is not possible to construct an invertible mapping between them, violating the required symmetry. Reversible jump MCMC addresses this by introducing a dimension matching condition [25]; the spaces $\Theta^{(1)}$ and $\Theta^{(2)}$ are augmented with simulated draws from selected distributions such that

$$(\boldsymbol{\theta}^{(2)}, u_2) = T_{1,2}(\boldsymbol{\theta}^{(1)}, u_1), \quad u_1 \sim g_{1,2}(\cdot),\ u_2 \sim g_{2,1}(\cdot),$$

where $T_{1,2}$ is a differentiable bijection, and $g_{i,j}(\cdot)$ are known distributions.

The parameter mappings and stochastic draws change the equilibrium distribution of the chain, which means that sampling will not be asymptotically correct. This is counteracted by modifying the acceptance ratio; in the case of jumping from model $M^{(1)}$ to model $M^{(2)}$, the acceptance ratio is given by

$$\alpha^{(1,2)} = \left| \frac{\partial T_{1,2}(\boldsymbol{\theta}^{(1)}, u_1)}{\partial(\boldsymbol{\theta}^{(1)}, u_1)} \right| \frac{g_{2,1}(u_2)}{g_{1,2}(u_1)} \frac{\pi_{2,1}}{\pi_{1,2}} \frac{p_2(\boldsymbol{\theta}^{(2)})}{p_1(\boldsymbol{\theta}^{(1)})}, \tag{3.2}$$

where $p_i(\boldsymbol{\theta}^{(i)})$ is the density associated with model $M^{(i)}$ evaluated at $\boldsymbol{\theta}^{(i)}$. The proposal $(M^{(2)}, \boldsymbol{\theta}^{(2)})$ is accepted with probability $\min(\alpha^{(1,2)},1)$, and is otherwise rejected. On rejection, the previous value of the chain is kept, $(M^{(1)}, \boldsymbol{\theta}^{(1)})$.

In this way, given data $\mathbf{y}$, RJMCMC samples the joint posterior $p(\boldsymbol{\theta}, M|\mathbf{y}) \propto p(M|\mathbf{x})p(\boldsymbol{\theta}|M,\mathbf{y})$. However, in the case where only $\boldsymbol{\theta}$ is of interest, following standard Monte Carlo rules, we can discard the samples of M to obtain $p(\boldsymbol{\theta}|\mathbf{y})$ [63].

Reversible jump MCMC incorporates many models into a single chain, so it is simple to compare or average models. However, the parameter mappings and model jump probabilities must be well designed. Poor selection of these parameters will typically lead to poor mixing in the model space [25, 63]. Our method explores only a single overall model, which simplifies the mappings. We impose a pairwise structure on the model space, simplifying the jumps significantly.

3.2.3 Parameter definitions and notation

In order to use RJMCMC to explore sparsity in the transition matrix of a linear-Gaussian state-space model, we must first construct a set of candidate sub-models of the LGSSM that exhibit various sparsity levels.

Denote by M_n the model selected by the algorithm at iteration n. This model is uniquely defined by the associated set of indices of dense elements in $\mathbf{A}$, which we denote $\mathcal{M}_n$. Note that there are thus $2^{N_x^2}$ models in our model space, precluding parallel evaluation for even small N_x. It is therefore not possible to use methods such as Bayes factors or marginal likelihood to compare models, as is standard in Bayesian model selection in state-space models [46, 48], as the computational cost is infeasible, due to these requiring all models to be evaluated in order to be compared. The number of elements of $\mathcal{M}_n$, denoted by $D_n = |\mathcal{M}_n|$, is the number of dense elements at iteration n, and therefore the number of non-zero elements of $\mathbf{A}_n$. Denote by $S_n = N_x^2 - D_n$ the number of sparse elements at iteration n. If the true value of a parameter is known, then it will be presented without subscript or superscript. We denote by a superscript c the complement to a set, and note that $\mathcal{M}_n^{\mathsf{c}}$ denotes the set of indices of elements sparse in $\mathbf{A}$ at iteration n.

Each model has an associated parameter space, which we denote by Θ_n. As the parameter space of a sparse parameter is $\{0\}$, we therefore have

$$\Theta_n = \prod_{0<i,j\leq N_x} \boldsymbol{\theta}_{(i,j),n},\ \boldsymbol{\theta}_{(i,j),n} = \begin{cases} \mathbb{R}, & (i,j) \in \mathcal{M}_n, \\ \{0\}, & \text{otherwise}, \end{cases} \tag{3.3}$$

where $\boldsymbol{\theta}_{(i,j),n}$ is the support for $(\mathbf{A}_n)_{ij}$. In the Bayesian paradigm, we can interpret the sparsity of model M_n as a prior constraint induced by $p(\mathbf{A}|M_n)$, under which the

elements indexed by $\mathcal{M}_n^{\mathsf{c}}$ are always of value 0, and the elements indexed by $\mathcal{M}_n$ are distributed as $p(\mathbf{A})$. This is equivalent to fixing the value of the elements of $\mathbf{A}$ indexed by $\mathcal{M}_n^{\mathsf{c}}$ to 0, as in Eq. (3.3).

We denote the $k \times k$ identity matrix by $\mathbf{Id}_k$, and the k-vector with all elements equal to 1 by $\mathbf{1}_k$. We denote by $\cdot$ any unspecified parameters of a function or distribution. For example, $x \sim g(\cdot)$ means that the parameters of the distribution g are unspecified, typically as they are irrelevant to the discussion.

3.3 Model specification

We first specify the model that we impose. We define the marginal model prior $p(M)$ as being proportional to 1 for all models; this could be set such that a desired level of sparsity is promoted. Note that setting $p(M)$ as this implies that each element is sparse or dense with probability 0.5.

Next, we define the prior of the entire transition matrix conditional on a given model as

$$p(A_{ij}|M) \sim \begin{cases} p(A_{ij}), & (i,j) \in \mathcal{M}, \\ \delta_0, & \text{otherwise}, \end{cases} \tag{3.4}$$

with δ_0 denoting a point mass at 0. This yields $p(\mathbf{A}|M)$ when constructing the matrix distribution $p(A_{ij}|M)\ \forall 1 \leq i,j \leq N_x$. This leaves us to choose $p(A_{ij})$, which we set to be a Laplace distribution centred at 0 with scale σ.

By substituting $\mathbf{A}$ into Eq. (2.21) we obtain

$$p(\mathbf{y}_{1:T}|\mathbf{A},M) = p(y_1|\mathbf{A},M) \prod_{i=1}^{T} p(\mathbf{y}_i|\mathbf{y}_{1:i-1},\mathbf{A},M), \tag{3.5}$$

and can hence evaluate $p(\mathbf{A}|M)p(\mathbf{y}_{1:T}|\mathbf{A},M)$, allowing us to sample from $p(\mathbf{A},M|\mathbf{y}_{1:T})$.

From this we can construct our posterior $p(\mathbf{A},M|\mathbf{y}_{1:T}) \propto p(M)p(\mathbf{A}|M)p(\mathbf{y}_{1:T}|\mathbf{A},M)$, which we now propose a method to sample.

3.4 The SpaRJ algorithm

We now present the SpaRJ algorithm, a novel RJMCMC method to obtain sparse samples from the posterior distribution $p(\mathbf{A}|\mathbf{y}_{1:T})$ of the transition matrix of an LGSSM. We present our method in Algorithm 5 for estimating the transition matrix $\mathbf{A}$, although the algorithm can be adapted to estimating any unknown parameter of the LGSSM. Our method samples the joint posterior $p(\mathbf{A},M|\mathbf{y}_{1:T}) \propto p(M)p(\mathbf{A}|M)p(\mathbf{y}_{1:T}|\mathbf{A},M)$ by

first sampling M' from $p(M|\mathbf{y}_{1:T})$, then sampling $\mathbf{A}'$ from $p(\mathbf{A}|M',\mathbf{y}_{1:T})$ conditional on M', and then jointly accepting or rejecting the proposed values. However, as we are only interested in the posterior of the transition matrix $p(\mathbf{A}|\mathbf{y}_{1:T})$, we marginalise by discarding the samples of M when performing inference [63].

In order to apply our method, we must provide the values of all known parameters of the LGSSM, (used when evaluating the Kalman filter), and initial values for the unknown parameters $\mathbf{A}$ and M. We initialise the model sampling by setting M_0 to the fully dense model. The initial value $\mathbf{A}_0$ can be selected in a number of ways, such as randomly or via optimisation [63]. We obtain the initial log-likelihood, l_0, by running a Kalman filter with the chosen initial value $\mathbf{A}_0$, giving $l_0 = \log(p(\mathbf{y}_{1:T}|\mathbf{A}_0))$. We define a conditional prior $p(\mathbf{A}|M)$ on the transition matrix, as Section 3.3. The prior distribution incorporates our prior knowledge as to the value of the transition matrix $\mathbf{A}$, and can be used to promote sparsity. However, it is not required in order to recover sparse samples, and can be chosen to be uninformative or diffuse.

The method iterates N times, each iteration yielding a single sample, outputting N samples, $\{\mathbf{A}_n\}_{n=1}^{N}$. While adapting the number of iterations is possible, e.g. by following [64], we present with fixed N so as to provide a simpler algorithm. Note that, at each iteration we start in model M_{n-1}, with transition matrix $\mathbf{A}_{n-1}$, and log-likelihood of $l_{n-1}(\mathbf{A},M,\mathbf{y}_{1:T})$. Each iteration is split into three steps: model proposal (Step 1), parameter proposal (Step 2), and accept-reject (Step 3). We note that the model is not fixed, and is sampled at each iteration, allowing for evidence-based recovery of sparsity.

Step 1: Propose M'. At iteration n, we retain the previous model M_{n-1} with probability (w.p.) π_0, and hence setting $M' = M_{n-1}$. If a model jump occurs, we set the proposed model M' to be sparser than M_{n-1} w.p. π_{-1}, and denser otherwise. To create a model that is sparser, we select a number of dense elements to then make sparse. To create a denser model, we select a number of sparse elements to make dense. The number of elements to change, k, is drawn from a truncated Poisson distribution (see Appendix 3.8.2) with rate parameter $\lambda_{\text{jump}} \in [0,1)$, with this range required in order to bias the jump to models close to the previous model. This distribution is chosen as it exhibits the required property of an easily scaled support, needed as the maximal jump distance changes with the current model. Note that this distribution does not form a prior over the model space, but instead is used to generate jump kernels, which are then used to explore the model space. The model space prior $p(M)$ is discussed in Section 3.3, and is by default diffuse, i.e. $p(M) = 2^{-N_x^2} \forall M$. The truncated Poisson distribution is simple to sample, as it is a special case of the categorical distribution. The elements to change are then selected uniformly. The proposed model M' is always strictly denser than, strictly sparser than, or identical to M_{n-1}, following the construction of model jumps in Section 3.5.1.

Step 2: Propose $\mathbf{A}'$. If the proposed model M' differs from the previous model M_{n-1}, then the parameters $\boldsymbol{\theta}_{n-1}$ are mapped to $\boldsymbol{\theta}'$ via a modified identity mapping, which we discuss in Section 3.5.2. This mapping is augmented with stochastic draws if the dimension of the parameter space increases, and has elements removed if the dimension decreases. This mapping has identity Jacobian matrix, and is thus absent from the acceptance ratio.

If the proposed model M' is the same as the previous model M_{n-1}, then A' is sampled from the conditional posterior $p(\mathbf{A}|M')$. To achieve this, we use a random walk Metropolis-Hastings (RWMH) sampler. The RWMH sampler requires a single run of the Kalman filter per iteration, which is the most computationally expensive component of the algorithm. This single run follows from the joint accept-reject decision in Step 3, which allows us to ignore the accept-reject step of a RWMH sampler that jointly assesses all proposals, as in the case of SpaRJ. We cover the parameter proposal process further in Section 3.5.2. Note that any sampler can be used, even a non-MCMC method, with RWMH chosen for simplicity, computational speed, and to give a baseline statistical performance.

Step 3: Metropolis accept-reject. Once the model and parameter values have been proposed, a Metropolis-Hastings acceptance step is performed. We run a Kalman filter with $\mathbf{A}'$ to calculate the log-likelihood of the proposal, $l'(\mathbf{A},M,\mathbf{y}_{1:T}) = \log(p(\mathbf{y}_{1:T}|\mathbf{A}'))$.

Prior knowledge is included via a function of the prior probability densities, denoted Λ, which encodes both our prior knowledge of the parameter values and of the model (hence the sparsity). A wide range of prior distributions $p(\mathbf{A}|M)$ can be used, with our preference being the Laplace distribution, with the associated Λ given by

$$\Lambda(\mathbf{A}_{n-1},\mathbf{A}',\lambda) = \lambda(\|\mathbf{A}_{n-1}\|_1^1 - \|\mathbf{A}'\|_1^1) \tag{3.6}$$

after combining all $p(A_{ij}|M)$ to yield $p(\mathbf{A}|M)$, which is known to promote sparsity in Bayesian inference [29]. Note that the sparsity in our samples does not come from the choice of parameter prior, but from the model jump scheme, and therefore any suitable prior could be applied to the elements of $\mathbf{A}$, such as a Gaussian distribution or a t distribution. If we denote by $p(\mathbf{A}|M)$ our conditional prior on the transition matrix, then $\Lambda(\mathbf{A}_{n-1},\mathbf{A}') = \log(p(\mathbf{A}')) - \log(p(\mathbf{A}_{n-1}))$.

When defining the model space in Section 3.2.3, we note that each model is uniquely determined by the sparsity structure it imposes, with this structure being present in all samples of $\mathbf{A}$ generated from this model. We can therefore assess the model against our prior knowledge solely based on the sample structure, without a separate prior on the model space. An example of such a function is the L_0-norm, which penalises the number of non-zero elements, a property determined entirely by the model that can

be assessed via the samples. The log-acceptance ratio of the proposed values $\mathbf{A}'$ and M' is given by $\log(a_r(\mathbf{A},M,\mathbf{y}_{1:T}) = l'(\mathbf{A},M,\mathbf{y}_{1:T}) - l_{n-1}(\mathbf{A},M,\mathbf{y}_{1:T}) + \Lambda'_{n-1} + c$, where $c(\mathbf{A},M,\mathbf{y}_{1:T})$ is given in Appendix 3.8.3. The model and parameter proposals are jointly accepted with probability $a_r(\mathbf{A},M,\mathbf{y}_{1:T})$, and are otherwise rejected. If the proposals are accepted, then we set $M_n := M'$, $\mathbf{A}_n := \mathbf{A}'$, and $l_n(\mathbf{A},M,\mathbf{y}_{1:T}) := l'(\mathbf{A},M,\mathbf{y}_{1:T})$. Otherwise, we set $M_n := M_{n-1}$, $\mathbf{A}_n := \mathbf{A}_{n-1}$, and $l_n(\mathbf{A},M,\mathbf{y}_{1:T}) := l_{n-1}(\mathbf{A},M,\mathbf{y}_{1:T})$.

Algorithm 5 SpaRJ algorithm

Input: $\mathbf{y}_{1:T},\mathbf{A}_0,g(\cdot),\pi_0,\pi_{-1},N,\Lambda(\cdot,\cdot,\boldsymbol{\lambda}),\mathbf{P}_0,\mathbf{Q},\mathbf{R},\mathbf{H},\bar{\mathbf{x}}_0$
Output: Set of N samples $\{\mathbf{A}_n,l_n(\mathbf{A},M,\mathbf{y}_{1:T}),M_n\}_{n=1}^N$
Initialisation
Initialise M_0 as fully dense
Evaluate filtering equations, obtaining $l_0 := \log(\mathrm{p}(\mathbf{y}_{1:T}|\mathbf{A}_0,M_0))$
for $n=1,\ldots,N$ **do**
  Set $c(\mathbf{A},M,\mathbf{y}_{1:T}) := 0$.
  Step 1: Propose model (Section 3.5.1)
  Run Algorithm 6 to propose M'
  Step 2: Propose $\mathbf{A}'$ (Section 3.5.2)
  Run Algorithm 7 to propose $\mathbf{A}'$ and compute $c(\mathbf{A},M,\mathbf{y}_{1:T})$
  Step 3: MH accept-reject (Section 3.5.3)
  Evaluate Kalman filter with $\mathbf{A} := \mathbf{A}'$
  Set $l'(\mathbf{A},M,\mathbf{y}_{1:T}) := \log(p(\mathbf{y}_{1:T}|\mathbf{A}'))$
  Compute $\log(a_r(\mathbf{A},M,\mathbf{y}_{1:T})) := l'(\mathbf{A},M,\mathbf{y}_{1:T}) - l_{n-1}(\mathbf{A},M,\mathbf{y}_{1:T}) + \Lambda(\mathbf{A}_{n-1},\mathbf{A}',\boldsymbol{\lambda}) + c$
  Accept w.p. $\min(a_r(\mathbf{A},M,\mathbf{y}_{1:T}),1)$
  if *Accept* **then**
    Set $M_n := M'$, $\mathbf{A}_n := \mathbf{A}'$, $l_n(\mathbf{A},M,\mathbf{y}_{1:T}) := \log(p(\mathbf{y}_{1:T}|\mathbf{A}',M'))$
  else
    Set $M_n := M_{n-1}, \mathbf{A}_n := \mathbf{A}_{n-1}, l_n(\mathbf{A},M,\mathbf{y}_{1:T}) := l_{n-1}(\mathbf{A},M,\mathbf{y}_{1:T})$
  end if
end for

3.5 Algorithm design

We now detail the three steps of Algorithm 5 as presented in Section 3.4. This section is structured to follow the steps of the algorithm for clarity and reproducibility.

3.5.1 Step 1: Model sampling

In order to explore potential sparsity of $\mathbf{A}$ using RJMCMC, we design a model jumping scheme that exploits the structure inherent to the model space.

Algorithm 6 Model proposal routine

Input: M_{n-1},π_{-1},λ_{jump}
Output: M',$\{I_i\}_{i=1}^k$
Step 1: Determine jump
Retain w.p. π_0
if *Retain* **then**
  Set $M' := M_{n-1}$
else
  Step 1.1: Determine jump direction
  if $|\mathcal{M}_{n-1}| = 0$ **then**
    Jump denser
  else if $|\mathcal{M}_{n-1}| = N_x^2$ **then**
    Jump sparser
  else
    Jump sparser w.p. π_{-1}, else *jump denser*
  end if
  Step 1.2: Perform jump
  if *Jump sparser* **then** (Step 1.2s)
    Draw $k \sim \text{TPoi}(\lambda_{\text{jump}},1,D_n)$ (See App. 3.8.2)
    Select k elements $\{I_i\}_{i=1}^k$ of $\mathcal{M}_{n-1}$
    Set M' such that $\mathcal{M}' = \mathcal{M}_{n-1} \setminus \{I_i\}_{i=1}^k$
  else *Jump denser* (Step 1.2d)
    Draw $k \sim \text{TPoi}(\lambda_{\text{jump}},1,S_n)$ (See App. 3.8.2)
    Select k elements $\{I_i\}_{i=1}^k$ of $\mathcal{M}_{n-1}^{\mathsf{c}}$
    Set M' such that $\mathcal{M}' = \mathcal{M}_{n-1} \cup \{I_i\}_{i=1}^k$
  end if
end if

Model jumping scheme (steps 1.1 and 1.2 in Alg. 6)

At each iteration, the algorithm proposes to jump models with probability (w.p.) $1-\pi_0$, as in Step 1 of Algorithm 6. If the algorithm proposes a model jump, then the proposed model M' will be sparser than M_{n-1} w.p. π_{-1}, and denser than M_{n-1} otherwise. If no model jump is proposed, then $M' = M_{n-1}$.

There are thus three distinct outcomes of the model jumping step: retention of the previous model, proposing to jump to a sparser model, or proposing to jump to to a denser model. Note that in some cases it is not possible to jump in both directions, and hence if the jumping scheme proposes a model jump, the jump direction is deterministic. This changes the model jump probability, with the results detailed in Appendix 3.8.3.

Algorithm 7 Parameter proposal routine

Input: $\mathbf{A}_{n-1}$,$g(\cdot)$,M',$\{I_i\}_{i=1}^k$,c
Output: $\mathbf{A}'$,$c(\mathbf{A},M,\mathbf{y}_{1:T})$
Step 2: Determine jump
if Retain **then**
  Step 2.1: Sample posterior
  Sample $\mathbf{A}'$ from the conditional prior, $p(\mathbf{A}|M',\mathbf{y}_{1:T})$.
else
  Step 2.2: Map parameters
  if *Jump sparser* **then** (Step 2.2s)
    Set a_{I_i} equal to the value of $\mathbf{A}_{n-1}$ at index I_i for $i=1,...,k$.
    Set $\mathbf{A}':=\mathbf{A}_{n-1}$ with elements $\{I_i\}_{i=1}^k$ set to 0.
    Set $c(\mathbf{A},M,\mathbf{y}_{1:T}):=\sum_{i=1}^k \log(g(a_{I_i}))$.
  else (Step 2.2d)
    Draw $u_1,...,u_k \sim g(\cdot)$.
    Set $\mathbf{A}':=\mathbf{A}_{n-1}$ with elements $\{I_i\}_{i=1}^k$ set to u_i.
    Set $c(\mathbf{A},M,\mathbf{y}_{1:T}):=-\sum_{i=1}^k \log(g(u_i))$.
  end if
  Modify $c(\mathbf{A},M,\mathbf{y}_{1:T})$ as per Appendix 3.8.3.
end if

Model space adjacency (steps 1.2s and 1.2d in Alg. 6)

Given the jump direction from Step 1.1, we denote by k the number of elements that are to be made sparse or dense. We draw $k \sim \text{TPoi}(\lambda_{\text{jump}},1,m_n)$ (see Appendix 3.8.2), where m_n is the maximum jump distance in the chosen direction, equal to S_n if jumping denser, and D_n if jumping sparser. The rate λ_{jump} should be chosen with $\lambda_{\text{jump}} \in [0,1)$ to prefer jumps to closely related models. We find experimentally that $\lambda_{\text{jump}} = 0.1$ gives good results, and note that $\lambda_{\text{jump}} = 0$ is equivalent to a scheme in which the sparsity can change by one element only. Due to the small size of the space in λ_{jump} a grid search would also be possible. The resulting model jump probabilities are asymmetric, with the modification to the acceptance ratio given in Appendix 3.8.3.

In order to provide a set of candidate models for M', we impose an adjacency condition. We say model $M^{(1)}$ is densely k-adjacent to model $M^{(2)}$ if both $|\mathcal{M}^{(1)} \backslash \mathcal{M}^{(2)}| = k$ and $|\mathcal{M}^{(2)} \backslash \mathcal{M}^{(1)}| = 0$, and sparsely k-adjacent if both $|\mathcal{M}^{(2)} \backslash \mathcal{M}^{(1)}| = k$ and $|\mathcal{M}^{(1)} \backslash \mathcal{M}^{(2)}| = 0$. In other words, if the model $M^{(1)}$ differs in k elements from $M^{(2)}$ in a given direction only, then it is k-adjacent in that direction; if the model $M^{(1)}$ differs from $M^{(2)}$ in both the sparse and dense directions, e.g. two elements are sparser and one element is denser, then it is not adjacent to $M^{(2)}$. Note that, if $M^{(1)}$ is sparsely k-adjacent to $M^{(2)}$, then $M^{(2)}$ is densely k-adjacent to $M^{(1)}$, satisfying the reversibility condition of RJMCMC. With this adjacency condition, for a given k and jump direction, the proposed M' is uniformly selected from the models k-adjacent to

M_{n-1} in the given direction. Note that, given $\lambda_{\text{jump}} > 0$, it is theoretically possible to reach any model in a maximum of two jumps: one to the maximally sparse model and one jump denser to the desired model.

It is possible to extend this proposal technique to jumping in both direction simultaneously, rather than requiring combinations of birth-death moves to achieve the result. This is omitted for simplicity. The natural solution to this is evaluating each jump with an accept-reject step, which effectively recovers the current scheme. It is also possible to propose $\mathcal{M}'$ as a randomly selected list of indices with length D_n, effectively shuffling the sparse elements around. This could potentially allow for more robust exploration of the posterior, but these moves would be very unlikely to be accepted. We therefore believe our proposal method to be a good compromise between simplicity and robustness, with good performance as evidenced by Section 3.6.

Choice of parameters for model jumps

The value of the hyper-parameters π_0 and π_{-1} affects the acceptance rate of the proposed models and parameter values. It is known that an acceptance rate close to 0.234 is optimal for a random walk Metropolis-Hastings sampler [65], and works well as a rule of thumb for RJMCMC algorithms [66]. We aim to have our within-model samples accepted at close to this rate, and thus must not propose to change model too often. This is because model changes can significantly alter the conditional posterior $p(\mathbf{A}|M')$, often leading to a low acceptance probability. We recommend using a model retention probability of $\pi_0 \approx 0.8$. We find this gives enough iterations per model to average close to the optimal acceptance rate, whilst also proposing to jump models relatively frequently, allowing for exploration of sparsity. We recommend setting $\pi_{-1} = 0.5$, making the model proposal process symmetric, although π_{-1} can reflect prior knowledge of the sparsity of $\mathbf{A}$, with larger π_{-1} indicating a preference for sparsity. The algorithm is relatively insensitive to the value of π_0 and π_{-1}, allowing for these parameters to be chosen easily. However, π_0 and π_{-1} can be tuned during the burn-in period, with the objective of reaching a given acceptance rate. We find that using a value of 0.5 works well for both parameters, due to the structure of the model space and restricted parameter space of stable LGSSMs

3.5.2 Step 2: Parameter sampling and mapping

Since our method applies MCMC to sample the posterior distribution $p(\mathbf{A}|\mathbf{y}_{1:T})$, we must define a parameter proposal routine. Once a model has been proposed, the algorithm proposes a parameter value, $\mathbf{A}'$, which is constrained to the parameter space of M'. The process by which we generate the parameter proposal depends on whether

or not the proposed model M' is the same as the previous model M_{n-1}. If $M' = M_{n-1}$, the conditional posterior of the transition matrix, $p(\mathbf{A}|\mathbf{y}_{1:T}, M')$, is sampled. Otherwise, the parameter value is mapped to the parameter space of M'.

Sampling A under a given model (Step 2.1)

To generate the parameter proposal $\mathbf{A}'$, we sample from $p(\mathbf{A}|M', \mathbf{y}_{1:T})$. We propose to sample from this distribution using a random walk Metropolis-Hastings (RWMH) sampler. For the walk distribution, we use a Laplace distribution for each element of $\mathbf{A}$, with all steps element-wise distributed i.i.d. Laplace$(0,\sigma)$, with σ discussed below. We can thus view our parameter proposal as drawing from

$$(A')_{ij} \sim \begin{cases} \text{Laplace}((A_{n-1})_{ij}, \sigma), & (i,j) \in \mathcal{M}', \\ \delta_0, & \text{otherwise.} \end{cases} \tag{3.7}$$

The Laplace distribution is selected, primarily due to its relationship with our proposed prior distribution for $\mathbf{A}$, itself a Laplace distribution [29]. In addition, the mass concentration of the Laplace distribution means that the walk will primarily propose values close to the previous value, increasing the acceptance rate, but can also propose values that are further from the accepted value, improving the mixing of the sample chain. The value of σ is chosen to give a within-model acceptance rate near the optimal rate of 0.234 [65], with $\sigma = 0.1$ consistently yielding rates close to this. A grid search suffices to select the value of this parameter as, for stable systems, the space of feasible values is small ($\sigma < 1$ is recommended).

Completion distributions (steps 2.2s, 2.2d)

In our algorithm, when a model jump occurs the dimension of the parameter space always changes. For example, jumping to a sparser model is equivalent in the parameters space to discarding parameters and decreasing the dimension of the parameter space. However, if jumping to a denser model, hence increasing the dimension of the parameter space, we require a method to assign a value to the new parameter. RJMCMC accomplishes this by augmenting the parameter mapping from model $M^{(i)}$ to $M^{(j)}$ with draws from a completion distribution $g_{i,j}(\cdot)$, defined for each possible model jump [25].

Rather than defining a distribution for every pair of models, we exploit the numerical properties of sparsity to define a global completion distribution $g(\cdot)$. In our samples, sparse elements take the value zero. In order to propose parameters close to the previous parameters, we draw the value of newly dense elements such that the value is close to zero. In order to accomplish this, we choose a Laplace$(0,\sigma_c)$ as the

global completion distribution $g(\cdot)$. The σ_c parameter is subject to choice, and for stable systems we find that $\sigma_c \approx 0.1$ performs well, although the parameter could be tuned during the burn-in period. A simple grid search suffices to select the value of this parameter as the method is robust to parameter specification via the accept-reject step [63]. For stable systems, the search space for this parameter is small, approximately (0,0.5], so a grid search is most efficient in terms of computational cost. If the prior is chosen as per the recommendations in Section 3.5.3, then we can interpret this renewal process as drawing the values for newly dense elements from the prior.

Mapping between parameter spaces (steps 2.2s, 2.2d)

In order to jump models, we must be able to map between the parameter spaces of the previous model and the proposal. This is, in general, a difficult task [25, 63], but is eased in our case as the models we are sampling are specific cases of the same model, and thus the parameters are the same between models. We therefore use an augmented identity mapping to preserve the interpretation of parameter values between models. Written in terms of $\mathbf{A}$, this mapping is given by

$$(A')_{ij} = \begin{cases} (A_{n-1})_{ij}, & \text{if } A_{ij} \text{ is unchanged,} \\ u_{ij}, & \text{if } A_{ij} \text{ becomes dense,} \\ 0, & \text{if } A_{ij} \text{ becomes sparse,} \end{cases} \tag{3.8}$$

with u_{ij} i.i.d. $g(\cdot)$. In order to obtain the Jacobian required to evaluate Eq. (3.2), the transformation must be written as applied to the parameter space, giving

$$(A')_{ij} = \begin{cases} (A_{n-1})_{ij}^{(n-1)}, & (i,j) \in \mathcal{M}_{n-1} \cap \mathcal{M}', \\ u_{ij}, & (i,j) \in \mathcal{M}' \setminus \mathcal{M}_{n-1}. \end{cases} \tag{3.9}$$

As sparse elements are, by construction, not in the parameter space, they are not present in the transformation, and are taken to be zero in $\mathbf{A}'$ by definition. The Jacobian of the parameter mapping given by Eq. (3.9) is a $D' \times D'$ identity matrix, and hence the Jacobian determinant term in Eq. (3.2) is constant and equal to one.

3.5.3 Step 3: MH accept-reject

In this section, we discuss the MH acceptance ratio, and how this relates to the model space. We then discuss the implications of sparsity in the samples.

Modified acceptance ratio

The modified Metropolis-Hastings acceptance ratio for our method is given by

$$a_r(\mathbf{A},M,\mathbf{y}_{1:T})^{(n-1,n)} = \frac{g(u_n)}{g(u_{n-1})}\frac{\pi_{n,n-1}}{\pi_{n-1,n}}\frac{p(\mathbf{A}')p(\mathbf{y}_{1:T}|\mathbf{A}')}{p(\mathbf{A}_{n-1})p(\mathbf{y}_{1:T}|\mathbf{A}_{n-1})}. \tag{3.10}$$

The first term is a correction for detailed balance, required due to the stochastic completion of the parameter mappings. The second term is analogous to the modification required when using an asymmetric proposal in RWMH, but relating to the model space. The last term of the expression is the standard symmetric Metropolis acceptance ratio. Note that the Jacobian term from Eq. (3.2) is equal to one in our case, and is therefore omitted.

Probabilistic Granger causality

In LGSSMs, an element x_i of the state space Granger-causes element x_j if knowledge of x_i at time t improves the prediction of x_j at time $t+1$. We can therefore derive probabilistic Granger-causal relationships from our samples of the transition matrix, as the sampler assesses the proposals using their likelihood, which is equivalent to assessing their predictive capabilities. These probabilistic Granger-causal relationships are powerful, as they allow the probability of a relationship between variables to be quantified.

Note that A_{ji} being zero in the transition matrix does not necessarily mean independence of the state elements, but directed conditional independence on the scale of one time step. This conditional independence means that x_i does not Granger-cause x_j, however x_i may indirectly affect x_j through another variable over multiple time steps.

3.5.4 Extending SpaRJ to other parameters

Our method can be used to obtain sparse estimates of any of the parameters of the LGSSM, although some modifications are required to extend the formulation given above (for the transition matrix). Extending our method to the matrix $\mathbf{H}$ requires only for the parameter and model proposal to be changed to reflect the size of $\mathbf{H}$.

Extending the method to covariance matrices requires the proposal value to be constrained such that the resulting matrix is positive semi-definite. If the method does not jump models, remaining in model M_{t-1}, a possible proposal distribution for the state covariance proposal $\mathbf{Q}'$ is

$$\mathbf{Q}' \sim \text{Wishart}(\mathbf{Q}_{t-1}/p,p), \tag{3.11}$$

where p is larger than N_x, with $p > 30N_x$ working well experimentally. This proposal has expectation close to $\mathbf{Q}_{t-1}$, and is positive semi-definite. Note that this distribution is not applied to $\mathbf{Q}$ if estimating only $\mathbf{A}$, e.g., in Section 3.4 or in other subsections of Section 3.5, and applies only to the extension to sampling the covariance parameter. We enforce the sparsity structure of $\mathbf{Q}_{t-1}$ in $\mathbf{Q}'$, by setting elements sparse under M_{t-1} to 0 after sampling but before the accept-reject step, replacing Step 2.1 in Algorithm 7, where the model does not change.

The model proposal step (Step 2.2 in Algorithm 7) would also need to be modified, with the diagonal being dense at all times, and enforcing indices (i,j) and (j,i) to have the same sparsity. The model space is therefore reduced, and model adjacency is assessed via only the upper triangular. This modification to the model proposal process completes the alterations required to use our method to sparsely sample the state covariance matrix. Note that the covariance parameters cannot be interpreted as encoding state connections, and therefore cannot be interpreted graphically in the same way as in the transition matrix.

3.6 Numerical study

We now present the results of three sets of simulation studies to evaluate our method, showcasing the performance of SpaRJ in several scenarios. The section is divided into three synthetic data experiments and one real data problem. First, we evaluate the method with isotropic covariance matrices over variable N_x and T. Next, we investigate the effect of known and unknown anisotropic state covariance $\mathbf{Q}$ over variable N_x and λ. The third experiment explores the effect of the true level of sparsity D in the transition matrix on the quality of inference. We then use real data to recover geographical relationships from global temperature data. Finally, we explore the convergence characteristics of the method and check guarantees.

For the synthetic experiments, we generate observations following Eq. (3.1), with $N_x = N_y$, $\mathbf{H} = \mathbf{Id}_{N_x}$, and take $\bar{\mathbf{x}}_0 = \mathbf{1}_{N_x}$ and $T = 100$ unless stated otherwise. The state covariance matrix $\mathbf{Q}$ is specified per study. We generate transition matrices and synthetic data for $N_x \in \{3,6,12\}$. Whilst this may seem limited in dimension, this equates to performing inference in 9,36, and 144 dimensional spaces, as each element of the transition matrix is an independent parameter. Furthermore, we sample the model space, which is of size $2^{N_x^2}$, e.g. 2^{144} when $N_x = 12$. In all experiments, we run SpaRJ for $N = 15000$ iterations, discarding the first 5000 as burn-in. The matrix $\mathbf{A}_0$ is generated using an EM scheme, initialised at a random element-wise standard normal matrix. We set $\pi_0 = 0.8$ and $\pi_{-1} = 0.5$ in all cases. The LASSO penalty is used, with λ chosen per

experiment. We use a truncated Poisson distribution for the jump size, with $\lambda_{\text{jump}} = 0.1$.

We contrast our proposed method method with GraphEM [50, 16], an algorithm with similar goals based on proximal optimisation. In addition, we compare with the conditional Granger causality (CGC) method of [54] and the DAG-based method (DAGMA) of [67]. These methods do not exploit the state-space model structure, and are trained only on the observations $\mathbf{y}_{1:T}$. We note that there are not other RJMCMC-based methods that are applicable to this problem. We therefore compare with a reference MCMC implementation that does not exploit sparsity, and is hence dense in all elements of the estimate. This is equivalent to running our method, but with $p(M) = 0$ except for the M corresponding to the fully dense $\mathbf{A}$ matrix, hence effectively removing Step 1 in Algorithm 5 and Section 3.4. We compare the metrics of RMSE, precision, recall, specificity (true negative rate), and F1 score, with an element being sparse encoded as a positive, and dense as a negative.

We use these metrics, which are associated with classification rather than regression, as our method outputs truly sparse samples, and therefore allows for parameters to be classified as sparse or dense without thresholding on their numerical value, or using confidence/credible intervals. We take an element to be sparse under SpaRJ by majority vote of the samples. We average the metrics over 100 independent runs of each algorithm. The average time taken runs to complete is given, with the runs being performed in parallel on an 8 core processor. No special effort was put into optimising any single method, and all methods that use the Kalman filter utilise the same implementation thereof. Note that the DAGMA implementation is GPU accelerated, whereas all other methods utilise only the CPU. RMSE for SpaRJ and for the reference MCMC is calculated with respect to the mean of post-burnin samples for each chain. Note the RMSE is computed relative to $\mathbf{A}$, not the sequence of underlying hidden states as is often the case. RMSE is not meaningful for CGC, as it estimates only connectivity. We generate our $\mathbf{A}$ matrices by drawing the dense elements from a standard normal, and then divide $\mathbf{A}$ by the magnitude of its maximal singular value to give a stable system.

3.6.1 Synthetic data validation

Isotropic covariances Q and R

We test the performance of the method with isotropic covariance matrices $\mathbf{Q}$ and $\mathbf{R}$.
Dimension 3 matrix. We generate $\mathbf{A}$ for dimension $N_x = 3$ with sparsity in one element per row and one element per column. We set $\mathbf{Q} = \mathbf{R} = \mathbf{Id}_{N_x}, \mathbf{P} = 10^{-8}\mathbf{Id}_{N_x}$, and $\lambda = 1 = \exp(0)$.
Dimension 6 block diagonal matrix. We generate $\mathbf{A}$ for dimension $N_x = 6$ as a

block diagonal matrix with 2×2 blocks. We set $\mathbf{Q}=\mathbf{R}=10^{-2}\mathbf{Id}_{N_x}$, $\mathbf{P}=10^{-8}\mathbf{Id}_{N_x}$, and $\lambda=\exp(-1)\approx0.367$.

Dimension 12 block diagonal matrix. We generate $\mathbf{A}$ for dimension $N_x=12$ as a block diagonal matrix with 2×2 blocks. We set $\mathbf{Q}=\mathbf{R}=10^{-2}\mathbf{Id}_{N_x}$, $\mathbf{P}=10^{-8}\mathbf{Id}_{N_x}$, and $\lambda=\exp(-1)\approx0.367$.

Table 3.1: Results for systems with known isotropic state covariance $\mathbf{Q}$, alongside average time per run.

N_x	method	RMSE	spec.	recall	prec.	F1	Time (s)
3	GraphEM	0.099	0.86	0.98	0.79	0.88	**0.043**
	SpaRJ	**0.092**	**0.98**	**0.99**	**0.99**	**0.99**	0.68
	CGC	–	0.85	0.95	0.87	0.92	0.32
	DAGMA	0.49	0.17	0.67	0.29	0.40	0.25
	MCMC	0.103	1	0	–	0	0.65
6	GraphEM	0.103	0.83	0.90	0.91	0.91	**0.22**
	SpaRJ	**0.094**	**0.88**	**0.96**	**0.94**	**0.95**	3.2
	CGC	–	0.87	0.93	0.91	0.90	1.6
	DAGMA	0.27	0.17	0.96	0.69	0.87	0.62
	MCMC	0.114	1	0	–	0	3.3
12	GraphEM	0.090	0.85	0.77	**0.96**	0.85	**0.93**
	SpaRJ	**0.071**	0.83	0.89	0.91	0.90	14.5
	CGC	–	0.80	0.67	0.75	0.71	6.5
	DAGMA	0.16	0.25	**0.98**	0.86	**0.92**	1.0
	MCMC	0.107	1	0	–	0	14.4

Table 3.1 evidences a good performance from SpaRJ, exhibiting the capability to extract the sparsity structure in all examples. Furthermore, point estimates resulting from SpaRJ are consistently closer to the true value than those from comparable methods, as evidenced by the lower RMSE. Note that in all cases the DAG based method recovered overly sparse graphs, as evidenced by the poor specificity scores. We further note that DAGMA is designed to recover acyclic graphs, with all graphs here being cyclical, further degrading performance.

In order to test the relationship between the recovered values and the number of observations T, we now demonstrate our method for different values of $T\in[10,150]$ using the same 3×3 system as previously. In Figure 3.1, we show averaged metrics over 100 independent runs for SpaRJ and GraphEM. We see that the longer the series the better the overall performance, with SpaRJ giving a better overall performance than GraphEM.

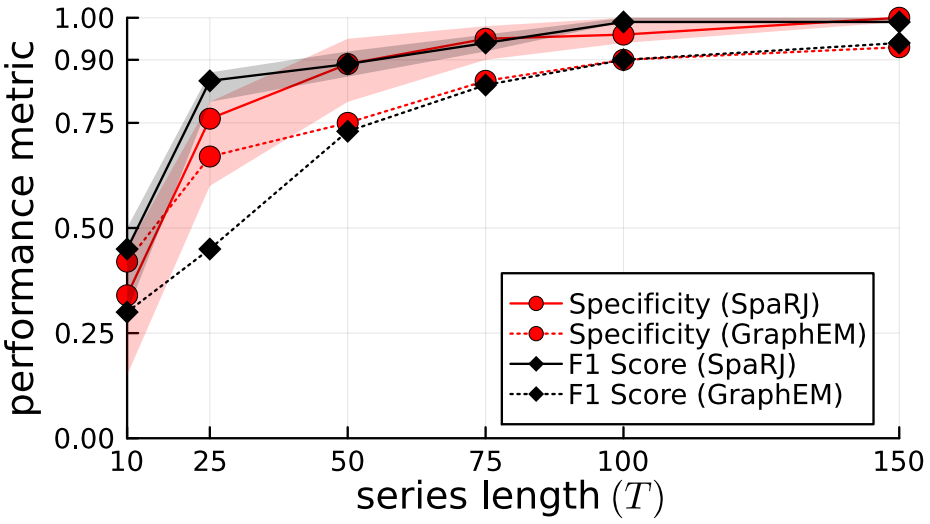


Figure 3.1: Sparsity metrics for variable series length T for a 3×3 system with known isotropic state covariance. Shaded regions denote 95% HPDIs (highest posterior density intervals), markers denote means. The dotted line indicates the mean performance of GraphEM.

The change in the quality of inference with the times series length T illustrated in Figure 3.1 is typical for parameter estimation methods in state-space modelling, as a longer series gives more statistical information with which to perform inference.

Known anisotropic state covariance Q

We now generate synthetic data using a less favourable regime, under an anisotropic state covariance $\mathbf{Q}$. In order to do this, we note that all $n\times n$ covariance matrices $\mathbf{\Sigma}$ can be expressed in the form

$$\mathbf{\Sigma}=\mathbf{G}^T\mathrm{Diag}(e_1,e_2,...,e_n)\mathbf{G}, \tag{3.12}$$

where $\mathbf{G}$ is an orthogonal matrix, and e_k are the eigenvalues of $\mathbf{\Sigma}$, with $e_1\geq e_2\geq\cdots\geq e_n>0$.

To generate a covariance matrix, we first generate an orthogonal matrix G following the algorithm of [68]. We then draw $e_i\sim\mathrm{U}(0.5,1.5)$, and sort in descending order. Finally, we obtain $\mathbf{\Sigma}$ via evaluation of Eq. (3.12). In this way, we generate a random positive definite matrix, with all elements non-zero.

To allow direct comparison with the previous results, we use the same set of model parameters as before, except we randomly generate the covariance $\mathbf{Q}$ for each system as above.

In Table 3.2 see that there is only a small apparent difference in performance between isotropic covariance and non-isotropic state covariances, providing that the

Table 3.2: Results for systems with known anisotropic state covariance $\mathbf{Q}$, alongside average time per run.

N_x	method	RMSE	spec.	recall	prec.	F1	Time (s)
3	GraphEM	0.093	0.98	0.85	0.97	0.88	**0.052**
	SpaRJ	**0.087**	**0.98**	**0.99**	**0.98**	**0.99**	0.67
	CGC	–	0.72	0.93	0.43	0.59	0.32
	DAGMA	0.61	0.17	0.33	0.17	0.22	0.25
	MCMC	0.107	1	0	–	0	0.68
6	GraphEM	0.09	**0.99**	0.48	**0.98**	0.63	**0.24**
	SpaRJ	**0.07**	0.88	0.90	0.94	**0.92**	3.3
	CGC	–	0.63	0.65	0.32	0.45	1.7
	DAGMA	0.26	0.25	**0.96**	0.71	0.82	0.62
	MCMC	0.09	1	0	–	0	3.3
12	GraphEM	0.097	**0.98**	0.30	**0.99**	0.46	**0.95**
	SpaRJ	**0.082**	0.95	0.83	0.99	0.90	14.4
	CGC	–	0.75	0.57	0.43	0.49	6.5
	DAGMA	0.16	0.25	**0.97**	0.86	**0.91**	1.0
	MCMC	0.099	1	0	–	0	14.4

covariance is known. This is expected, as a known covariance would not affect the estimation of the value of the state transition matrix. However, when estimating sparsity, the anisotropic nature of the state covariance does have an effect. This is due to the value of the state elements affecting each other in more than one way, as is the case in the isotropic covariance case. There is thus a small drop in metrics in all cases due to this additional source of error. We note that whilst DAGMA may seem to perform well due to the high F1 scores, it does this by recovering an overly sparse graph as indicated by the low specificity. For example, only 9 elements are recovered as dense in the 12 dimensional system, out of a true 24 dense elements, which does not well represent the underlying system.

We now perform a sensitivity analysis, in which we vary the strength of the prior by varying λ, and observe the effect on the results. We will perform this analysis on the $N_x = 12$ system with a known anisotropic covariance. The results of this analysis are presented in Table 3.3. We see that the results are not dependent on the prior parameter, meaning that the parameter can be chosen without excess computation or prior knowledge required.

Estimated unknown anisotropic covariance

In many scenarios, the true value of the state covariance $\mathbf{Q}$ is unknown, and must be estimated. As we wish to assess the performance of our method in this scenario, we use the same true state covariance as Section 3.6.1, but input an estimated state

Table 3.3: Results for variable penalty in the $N_x = 12$ known anisotropic system, alongside average time per run.

λ	RMSE	spec.	recall	prec.	F1	Time (s)
exp(−1)	0.082	**0.95**	**0.83**	**0.99**	**0.90**	14.2
exp(0)	0.085	0.95	0.83	0.98	0.90	**14.1**
exp(1)	0.081	0.94	0.83	0.99	0.89	14.3
exp(−1.5)	0.083	0.93	0.82	0.99	0.90	14.3
exp(−2)	**0.081**	0.94	0.82	0.99	0.89	14.2
0	0.082	0.94	0.82	0.99	0.90	14.2

covariance. However, as both $\mathbf{A}$ and $\mathbf{Q}$ are now unknown, we must estimate both parameters in order to obtain an estimate for $\mathbf{Q}$. We therefore iteratively estimate $\mathbf{A}$ and $\mathbf{Q}$ using their analytic maximisers, and input the resulting estimate for $\mathbf{Q}$ into the tested methods. The estimated $\mathbf{A}$ resulting from this initialisation is discarded, and is not used in our method, nor in any other method.

Table 3.4: Results for systems with estimated anisotropic state covariance $\mathbf{Q}$, alongside average time per run.

N_x	method	RMSE	spec.	recall	prec.	F1	Time (s)
3	GraphEM	0.123	0.75	0.72	0.62	0.65	**0.05**
	SpaRJ	**0.099**	**0.87**	**0.98**	**0.80**	**0.89**	0.64
	CGC	–	0.72	0.93	0.43	0.59	0.32
	DAGMA	0.61	0.17	0.33	0.17	0.22	0.25
	MCMC	0.127	1	0	–	0	0.68
6	GraphEM	0.097	0.68	0.38	0.70	0.49	**0.21**
	SpaRJ	**0.078**	**0.75**	0.53	**0.81**	0.63	3.2
	CGC	–	0.63	0.65	0.32	0.45	1.7
	DAGMA	0.26	0.25	**0.96**	0.71	**0.82**	0.62
	MCMC	0.152	1	0	–	0	3.5
12	GraphEM	0.102	**0.76**	0.34	0.88	0.49	**0.92**
	SpaRJ	**0.074**	0.60	0.53	**0.88**	0.65	14.7
	CGC	–	0.75	0.57	0.43	0.49	6.5
	DAGMA	0.16	0.25	**0.97**	0.86	**0.91**	1.0
	MCMC	0.124	1	0	–	0	15.0

We see in Table 3.4 that our method performs well under these challenging conditions, consistently outperforming existing methods. Note that the CGC and DAGMA metrics are unchanged from the previous section, as these methods do not require estimates of $\mathbf{Q}$. The deterioration of metrics is expected in this experiment, as we are inferring both the value of $\mathbf{Q}$ and the value of $\mathbf{A}$ from the same data, with the estimation of $\mathbf{A}$ being conditional on the estimated value of $\mathbf{Q}$. However, the sparsity structures of the estimates are better than comparable methods, and the parameter

value is still well estimated, as evidenced by the RMSE value.

Variable levels of sparsity

We now explore the performance of our method under variable levels of sparsity. To facilitate direct comparison, all other parameters of the state-space model remain the same between sparsity levels, as well as the matrix from which $\mathbf{A}$ is generated. This experiment is performed on a 4×4 transition matrix, with $\mathbf{Q}=\mathbf{R}=\mathbf{Id}_4$, $\mathbf{P}=10^{-8}\mathbf{Id}_4$, $T=50$, and $\lambda=0.5$.

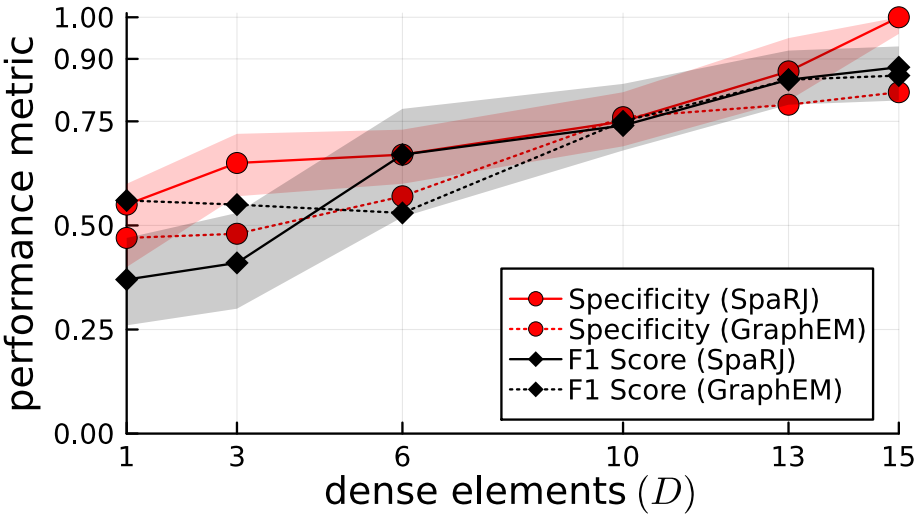


Figure 3.2: Sparsity metrics over variable sparsity in a 4×4 system. Shaded regions denote 95% HPDIs, markers denote means. The dotted line indicates the mean performance of GraphEM.

Note that, in systems with many dense elements, the effect of each element can be emulated by changing the values of a number of other elements, making sparsity recovery difficult in these cases. Our algorithm performs well in general, consistently outperforming other methods in this case.

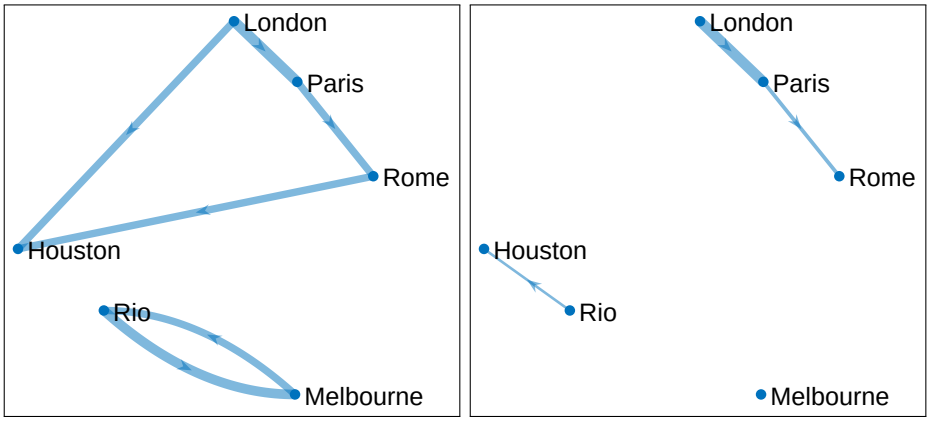
We observe from Figure 3.2 that both methods generally perform better as the number of sparse elements increases. This is due to sharper likelihood changes occurring when sparsity changes when assessing these models. For particularly dense transition matrices, GraphEM outperforms SpaRJ, due to the model proposal step of SpaRJ requiring more transitions to walk the larger space. This could be remedied with a prior encoding more information for these sampling regimes. For example, a penalty relating to the number of sparse elements could be incorporated into the prior. This ease of encoding prior preference in the model space is a strength of SpaRJ, and is not possible in comparable methods. Furthermore, SpaRJ can be assisted by GraphEM via the provision of a sparse initial value $\mathbf{A}_0$, which will greatly speed convergence.

We have not done this for any of the numerical experiments, but in practice we recommended doing so.

3.6.2 Application to global temperature data

We now apply our method to real data. We use the average daily temperature of 324 cities from 1995 to 2021, curated by the United States Environmental Protection Agency [69]. We subset the data to the cities of London (GB), Paris (FR), Rome (IT), Melbourne (AU), Houston (US), and Rio de Janeiro (BR) in 2010. We subset to a single year to avoid missing data.

We set the parameters as follows: $\pi_{-1}=0.5, \pi_0=0.8, \lambda=0.5$, and $\lambda_{\text{jump}}=0.2$. We estimate $\mathbf{Q}$ using the EM scheme detailed in Section 3.6.1, and set $\mathbf{H}=\mathbf{Id}_6, \mathbf{R}=0.5\mathbf{Id}_6$ as per the data specification. The results for GraphEM are given graphically in Figure 3.3a, while Figure 3.3b displays the results for SpaRJ. In Figure 3.3, edge thickness for GraphEM is proportional to the number of times an edge appears in 100 independent runs, whereas for SpaRJ edge thickness is proportional to the number of post-burnin samples from 100 chains with the edge present.



(a) Graph recovered via GraphEM. (b) Graph recovered via SpaRJ.

Figure 3.3: Graphs of state relationships recovered from temperature data using both GraphEM and SpaRJ. Self-self edges are omitted for clarity, but present in all instances (i.e., the elements in the diagonal of $\mathbf{A}$ are different from zero).

It is well known that weather phenomena are highly non-linear, and are driven by both local and global factors. Not all driving factors are recorded in the data, therefore making it a challenging task to extract statistically causal relationships between locations. For example, neither barometric pressure nor rainfall are utilised, which would assist with localisation [7].

We see that the geographical relationships between the cities are well recovered by SpaRJ. We see that the European cities form one cluster, Melbourne is separate,

and Rio weakly affects Houston in Figure 3.3b.

Second, note that there exists no ground truth to compare against in this problem. GraphEM recovers the graph given in Figure 3.3a, which does not reflect the geographical positioning of the cities, as there are connections across large distances which is not physically reasonable. In addition, this graph cannot be interpreted probabilistically, as is possible with SpaRJ. On the other hand, GraphEM is generally faster compared to SpaRJ.

The SpaRJ estimate, presented in Figure 3.3b, offers the capacity for additional inference. For instance, in Figure 3.3a, all edges recovered by GraphEM are of a similar thickness, indicating that they are recovered by many independent runs of the algorithm. This is a desirable characteristic of GraphEM, as it is indicative of good convergence, although it does not admit a probabilistic interpretation of state connectivity.

SpaRJ recovers the edges probabilistically, which is made apparent in Figure 3.3b by the variable edge thicknesses. In SpaRJ, the number of post-burnin samples in which a given edge is present gives an estimate of the probability that this edge is present. This is of particular interest when inferring potential causal relationships, as in this example. This property follows from the broader capacity of SpaRJ to provide Monte Carlo uncertainty quantification. For example, a credible interval for the probability of an edge being present can easily be obtained via bootstrapping with the output of SpaRJ, which is not possible with GraphEM, as it is designed to converge to a point.

We used a value of $\lambda = 1.2$ in GraphEM for this estimation. However, increasing λ would make the self-self edges disappear (i.e., zeros would appear in the diagonal of $\mathbf{A}$ before edges among cities would be removed). Comparing the results in Figure 3.3, we see that the output from SpaRJ is more feasible when accounting for geophysical properties. It is not reasonable for the weather of cities to affect each other across very large distances and oceans over a daily timescale, and the parameter estimate should reflect this. This spatial isolation is present in the SpaRJ estimate in Figure 3.3b, but is absent from the GraphEM result in Figure 3.3a.

3.6.3 Assessing convergence

As our method is a MCMC method, it is not desirable to converge in a point-wise sense, however it is desirable to converge in distribution to the target distribution. However, we cannot use standard metrics such as $\hat{R}$ [17] to assess convergence, as these assume that the same parameters are being estimated at all times, which is not the case in our method. In the literature there are specific methods to assess

convergence for RJMCMC algorithms, with proposed methods [70] breaking down for large model spaces with few visited models (as is the case here). However, due to the tight linking between the model space and the parameter space, we can assess convergence via a combination of model metrics and parameter statistics [63].

In order to properly assess convergence, we must take into account both the model space and parameter space, and assess convergence in both. As our model space is closely linked to the values in the parameter space, we are able to assess convergence by jointly observing parameter and model metrics.

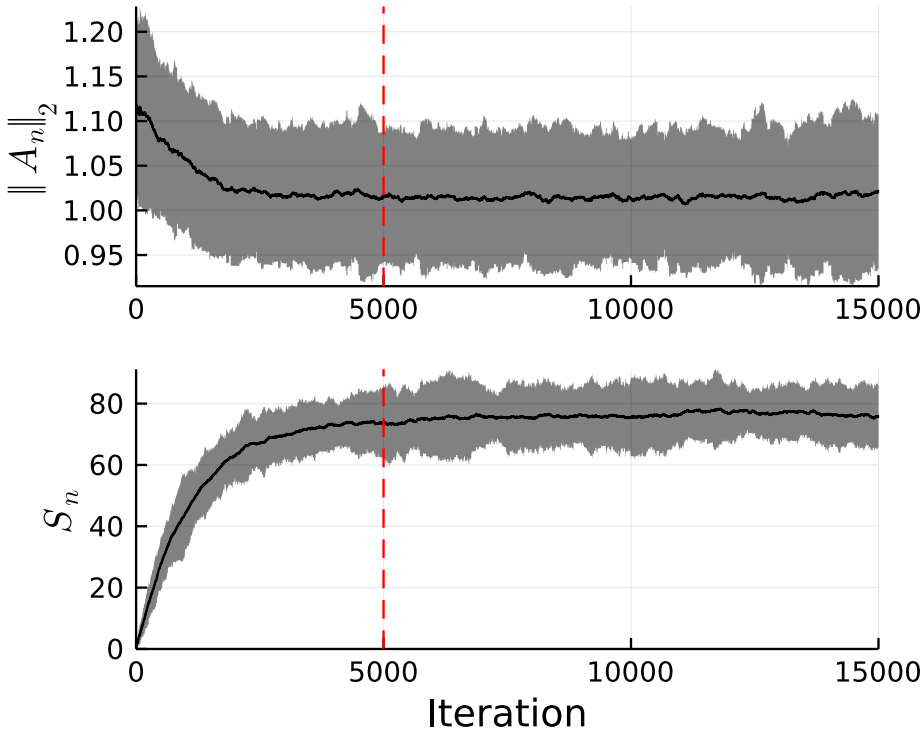


Figure 3.4: Progression of sample metrics in the 12×12 estimated state covariance model. The red line indicates the end of the burn-in period. Shaded regions are 95% HPDIs. Black line indicates the mean.

We track both the spectral norm of the sampled $\mathbf{A}_n$ (parameter metric) and the number of sparse elements (model metric), and plot them in Figure 3.4. We observe that convergence in both the parameter space and the model space occur quickly, and that convergence seems to occur before the burn-in period ends. This is the case for all examples, with the exemplar system being the slowest to converge. It is possible to decrease the time to convergence in several ways, such as better estimates of the model parameters. However, parameters such as π_0 and π_{-1} will also alter the speed of convergence, although the manner in which they do so is dependent on the true dynamics. We note that convergence speed is faster for lower dimensional $\mathbf{A}$, and conversely is slower

for larger $\mathbf{A}$. This is due to a larger $\mathbf{A}$ matrix having more variables to estimate, and hence requiring sampling from a higher dimensional space. Furthermore, the sampler converges faster for longer series lengths. Experimentally, sparser models benefit from a larger π_{-1}, whereas denser models benefit from a smaller π_{-1}. Increasing π_0 increases convergence speed in the parameter space, but decreases convergence speed in the model space. Convergence could also be improved by using a gradient-based sampler for the parameter posterior, as the gradient is available in closed form [5]. We find that the increase in computational cost and the reduction in modularity is not worth the increased speed of convergence. Finally, note that our method inherits the convergence guarantees of RWMH and RJMCMC, and therefore for a finite dimensional parameter space we are guaranteed to converge to the target sampling distribution given sufficient iterations.

3.7 Conclusion

In this chapter we have proposed the SpaRJ algorithm, a novel Bayesian method for recovering sparse estimates of the transition matrix of a linear-Gaussian state-space model. In addition, SpaRJ provides Bayesian uncertainty quantification of Granger causality between state elements, following from the interpretation of the transition matrix as representing information flow within an LGSSM. The method, built on reversible jump Markov chain Monte Carlo, has strong theoretical guarantees, displays performance exceeding state-of-the-art methods in both challenging synthetic experiments and when operating on real-world data, and exhibits great potential for extension.

3.7.1 Identifiability issues

We note that when running the method with the uniform prior over the models, we observe that in Figure 3.4 the method recovers approximately 72 sparse elements on average. This indicates that the posterior distribution over the model space essentially reduces to the prior in this example, with the prior in this case being that each element is independently dense (or sparse) with probability 0.5. Therefore, S is a priori distributed Binomial(144,0.5), having mean 72 and standard deviation 6. The use of the Laplace prior on the elements of $\mathbf{A}$ further exacerbates this problem, as we expect the magnitude of elements of $\mathbf{A}$ to be relatively small given that the system is stable. In order to remedy this, one could potentially use a different prior for the dense elements of $p(\mathbf{A}|M)$, which would place little probability mass near zero.

3.7.2 Sampling in the case of the uniform model prior

In the case that every model is a priori equally likely, we note that RJMCMC is not required in order to generate samples from our posterior. This is possible as the marginal prior

$$p(\mathbf{A}) = \sum_m p(\mathbf{A}|M=m)p(M=m) \tag{3.13}$$

can easily be evaluated as it is the product of equally weighted mixtures of a Laplace distribution and point masses at zero. As $p(\mathbf{y}_{1:T}|\mathbf{A},M) = p(\mathbf{y}_{1:T}|\mathbf{A})$ in this model, can evaluate

$$p(\mathbf{A}|\mathbf{y}_{1:T}) \propto p(\mathbf{A})p(\mathbf{y}_{1:T}|\mathbf{A}) \tag{3.14}$$

up to a normalising constant.

We can therefore generate samples from the posterior distribution $p(\mathbf{A}|\mathbf{y}_{1:T})$ using a standard Metropolis Hastings algorithm which proposes to update each element of $\mathbf{A}$ by sampling an equal weighted mixture of a Laplace distribution and a point mass at zero. However, for a more general $p(M)$ which encodes more complex prior beliefs, it may not be simple to integrate out the model prior, and therefore such a simplification might not be made.

3.8 Appendix items for Chapter 3

3.8.1 Guidance for choice of parameters

Table 3.5: Hints for choosing parameter values

Parameter	Hint	Recommended value(s)
λ	Increase to promote sparsity	$\in[\exp(-2),\exp(2)]$
λ_{jump}	Decrease to increase acceptance rate	$\in[0,0.5],\approx 0.2,$
π_0	Increase to increase acceptance rate	$\in[0.75,0.99],\approx 0.8$
π_{-1}	Increase to promote sparsity	$\in[0.4,0.6],\approx 0.5$
σ,σ_c	Decrease to increase acceptance rate	$\in[0.05,0.15],\approx 0.1$

3.8.2 Truncated Poisson distribution

Denote the Poisson distribution with rate λ that is left-truncated at a and right-truncated at b by $\text{TPoi}(\lambda,a,b)$. This distribution has support $n\in\mathbb{N}\cap[a,b]$, and probability mass function of

$$\text{TPoi}(n;\lambda,a,b)=\frac{\lambda^n e^{-\lambda}}{Z\cdot n!},\text{ with } Z=\sum_{n=a}^{b}\frac{\lambda^n e^{-\lambda}}{n!}. \tag{3.15}$$

3.8.3 Correction terms

In order to maintain detailed balance in the sampling chain, we must account for the unequal model transition probabilities, which is done via a correction term. These terms arise from the RJMCMC acceptance probability,

$$\frac{l'(\mathbf{A},M,\mathbf{y}_{1:T})}{l(\mathbf{A},M,\mathbf{y}_{1:T})}\frac{\pi_{n+1,n}}{\pi_{n,n+1}}\frac{g(u_n)}{g(u_{n+1})}\left|\frac{\partial T_{n,n+1}(\boldsymbol{\theta}_n,u_n)}{\partial(\boldsymbol{\theta}_n,u_n)}\right|, \tag{3.16}$$

in which $\pi_{n+1,n}/\pi_{n,n+1}$ is the ratio of the probability of the reverse jump to that of the forward jump. All other terms in the acceptance ratio are calculated in Algorithm 5, with the Jacobian term ignored as per Section 3.5.2.

As we are using log likelihoods and log acceptance ratios, we compute our correction on the log scale. For a given jump distance J in either the sparse or dense direction, we denote this log correction term $c_{j,J}(\mathbf{A},M,\mathbf{y}_{1:T})$, with $j=s$ when jumping sparser, and $j=d$ when jumping denser. This term is equal to the $\log(\pi_{n+1,n}/\pi_{n,n+1})$ term in the

acceptance ratio in Step 3 of Algorithm 5. The calculations for both sparser jumps and denser jumps proceed similarly, thus we detail only the derivation for sparser jumps.

The forward jump is a jump sparser, which occurs with probability π_{-1}. When jumping sparser we truncate the jump distribution at D_n. When using the truncated Poisson distribution, the probability of drawing a given jump length J given that the jump is in the sparse direction is $\mathrm{TPoi}(J;\lambda,1,D_n)$. Given J, the probability of choosing a given set of elements in the forward jump is $\binom{D_n}{J}^{-1}$. Multiplying these terms we obtain

$$\pi_{n,n+1} = \pi_{-1}\ \mathrm{TPoi}(J;\lambda,1,D_n)\ J!\ (D_n - J)!(D_n!)^{-1}.$$

The reverse jump is a jump denser, which occurs with probability $1-\pi_{-1}$. When jumping sparser, we truncate the jump distribution at S_n+J. Given these choices, the probability of drawing a given J for the jump distance is $\mathrm{TPoi}(J;\lambda,1,S_n+J)$. Given J, the probability of choosing a given set of elements in the reverse jump is $\binom{S_n+J}{J}^{-1}$. Multiplying these terms we obtain

$$\pi_{n+1,n} = (1-\pi_{-1})\mathrm{TPoi}(J;\lambda,1,S_n+J)J!S_n!(S_n+J)!^{-1}.$$

From which we obtain the acceptance ratio

$$\frac{\pi_{n+1,n}}{\pi_{n,n+1}} = \frac{(1-\pi_{-1})\mathrm{TPoi}(J;\lambda,1,S_n+J)S_n!D_n!}{\pi_{-1}\mathrm{TPoi}(J;\lambda,1,D_n)(S_n+J)!(D_n-J)!},$$

Writing $r := (1-\pi_{-1})(\pi_{-1})^{-1}$ we then have

$$\exp(c_{\mathrm{s},J}(\mathbf{A},M,\mathbf{y}_{1:T})) = r\frac{\mathrm{TPoi}(J;\lambda,1,S_n+J)S_n!D_n!}{\mathrm{TPoi}(J;\lambda,1,D_n)(S_n+J)!(D_n-J)!}, \tag{3.17}$$

with the corresponding term for the denser jump being

$$\exp(c_{\mathrm{d},J}(\mathbf{A},M,\mathbf{y}_{1:T})) = \frac{1}{r}\frac{\mathrm{TPoi}(J;\lambda,1,D_n+J)S_n!D_n!}{\mathrm{TPoi}(J;\lambda,1,S_n)(D_n+J)!(S_n-J)!}. \tag{3.18}$$

In the case of jumping to and from maximal density (MD) and maximal sparsity (MS) further adjustment is required. In these cases we replace r following Table 3.6.

Table 3.6: r terms for edge cases.

Jump	to MS	to MD	from MS	from MS
r	$(\pi_{-1})^{-1}$	$1-\pi_{-1}$	$(\pi_{-1})^{-1}$	$1-\pi_{-1}$

Chapter 4

Learning Partitions of Linear-Gaussian State-Space Models via Iterative Constraints

4.1 Introduction

In this chapter, we focus on the linear Gaussian state-space model, in which the between-step model dynamics are encoded by a transition matrix. This matrix can be interpreted as a graph encoding the connectivity of the between-step transition of the state space [16, 50]. Furthermore, it is known that real world dynamical systems are often made up of interacting subsystems [14], and therefore will exhibit clusters in the transition graph. These clusters are groups of state dimensions, in which within group connections are common, and between group connections are infrequent. The transition graph can be estimated using sparse estimation techniques such as [16, 35, 71], however whilst these methods promote sparsity in the estimate, they do not promote the desired cluster structure. In particular, we note that the method previously developed in Chapter 3 promotes sparsity without regard for a cluster structure. This method both extends and specifies that presented in Chapter 3, as we develop a general methodological framework by which to promote a more realistic sparse structure, but we develop this framework for pointwise estimates only, whereas the method in Chapter 3 yields distributional sparsity.

In this chapter, we develop the iterative graphical clustering for linear Gaussian state-space models (IGC-LGSSM) framework, a generic framework for performing clustering on the transition matrix of a LGSSM. This framework iteratively incorporates structural information into parameter estimation via a conditioning step, thereby promoting clustering in the sense explained above. As our framework promotes clustering, it therefore also promotes sparsity in the transition matrix.

These sparse clustered estimates have several advantages, in addition to typically better representing the model dynamics [14, 16, 35]. One such advantage is that quality of inference is typically improved, as promoting sparsity results in an implicit reduction in the dimension of the parameter space [28]. Furthermore, sparsity in state-space model parameter estimates allows us to infer the connectivity of the state dimensions, via interpreting the connectivity of the parameter estimates as in graphical models [72]. In doing so, we make the underlying system more interpretable, and allow for further inference to be carried out.

We exemplify our framework with a fully described algorithm within the new framework, itself a novel clustering method for estimating the between-step transition matrix of a LGSSM, and compare against existing state of the art methods for sparse parameter estimation in LGSSMs. Our method can be applied to estimate the transition matrix in models where the Kalman filter is applicable. Furthermore, by promoting separate clusters in the state space, Kalman filtering equations can be parallelised, thus decreasing the cost of future inference of the hidden states.

Contribution. The main contributions of this chapter are summarised as follows:

- We propose a new method to recover common structures exhibited by real-world systems by promoting clusters in the estimated transition matrix, resulting in a physically interpretable estimate of the system dynamics.
- The proposed framework is extensible, and can utilise several schemes for cluster recovery, such as edge betweenness centrality or minimum cut.
- Our method outperforms state of the art methods for structure recovery in LGSSMs, demonstrated by our numerical experiments.

4.2 Background

4.2.1 Notation

Denote by $G(V,E,W)$ a weighed directed graph with vertices V, edges E, and weights W, with weight $W(s,d)$ the weight of the edge with source s and destination d, with this edge denoted (s,d). We denote by $|G|$ the cardinality of the graph G, given by the number of vertices in the graph $|V|$. We define the adjacency matrix $\mathbf{A}$ of the weighed directed graph G as the matrix such that A_{ds} is the weight associated with the edge (s,d), and is zero if that edge is not present. This index ordering is typical when utilising graphs in dynamical systems. We denote by $\mathbf{Id}_n$ the $n \times n$ identity matrix.

4.2.2 State-space modelling and filtering

In this chapter, we consider the additive linear-Gaussian state-space model (LGSSM), given by

$$\begin{aligned} \mathbf{x}_t &= \mathbf{A}\mathbf{x}_{t-1} + \mathbf{q}_t, \\ \mathbf{y}_t &= \mathbf{H}\mathbf{x}_t + \mathbf{r}_t, \end{aligned} \tag{4.1}$$

for $t = 1,...,T$, where $\mathbf{x}_t \in \mathbb{R}^{N_x}$ is the hidden state at time t, $\mathbf{y}_t \in \mathbb{R}^{N_y}$ is the observation at time t, $\mathbf{A} \in \mathbb{R}^{N_x \times N_x}$ is the between-step state transition matrix, $\mathbf{H} \in \mathbb{R}^{N_y \times N_x}$ maps the hidden state to the observations, $\mathbf{q}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ is a realisation of the state noise, and $\mathbf{r}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is a realisation of the observation noise. The prior for is $\mathbf{x}_0$ is $\mathcal{N}(\bar{\mathbf{x}}_0, \mathbf{P}_0)$, where the hyper-parameters $\bar{\mathbf{x}}_0$ and $\mathbf{P}_0$ are typically assumed known [5].

4.2.3 Graphical clustering

In network analysis, it is common to partition graphs into clusters for further analysis [73, 74]. In order to recover clusters it is commonly assumed that that clusters are densely connected within the cluster, with few connections between clusters. Several method exist to recover these structures, with two commonly used methods being the Highly Connected Sub-graphs (HCS) algorithm [73], which utilises minimum cuts, and the Girvan-Newman algorithm [74], which utilises the edge betweenness centrality (EBC).

Minimum cut

A graph cut is a set of edges which partitions a graph into two disjoint subsets. We define the cost of a cut as equal to the sum of the weights of the edges of the cut. Therefore, the minimum cut of the weighted directed graph G is the cut with minimum cost.

HCS partitions a graph into clusters by iteratively performing minimum cuts on G and the resulting disconnected sub-graphs. This proceeds until all sub-graphs $G_i \subseteq G$ each have either a single vertex, or have edge connectivity $\lambda(G_i) > \frac{|G_i|}{2}$, where edge connectivity $\lambda(G)$ is the minimum number of edges that must be removed to disconnect G.

Edge betweenness centrality

The edge betweenness centrality (EBC) of the edge (s,d) is given by the fraction of minimum paths between all vertices of G that contain (s,d). We expect that edges between clusters will have a larger EBC than edges within a cluster, as shortest paths between nodes in separate clusters will traverse these edges. By iteratively computing the EBC, and removing the edge(s) with the highest EBC, we can recover clusters. This algorithm was proposed in [74], and stops when all edges have been removed.

A simple extension of this algorithm is to impose a stopping criteria, such as in [73], reducing the amount of computation required.

4.3 Iterative re-estimation under graphical sparsity constraints

We now propose a methodological framework for estimating the transition matrix $\mathbf{A}$ of a LGSSM whilst promoting a cluster structure in the resulting estimate. We achieve this via a sparsity promoting estimating scheme with iteratively learnt constraints on the implied connectivity graph to recover the cluster structure. Furthermore, we present our framework in a generic manner, allowing for different estimation schemes and cluster recovery methods to be utilised.

4.3.1 Proposed framework algorithm

Algorithm 8 IGC-LGSSM framework algorithm

1: **Input:** Series of observations $\mathbf{y}_{1:T}$, initial transition matrix $\mathbf{A}_0$, LGSSM parameters $\mathbf{H}$,$\mathbf{Q}$,$\mathbf{R}$,$\bar{\mathbf{x}}_0$,$\mathbf{P}_0$, maximum number of iterations N, stopping condition $S(\cdot)$.
2: **Output:** Series of estimates $\{\mathbf{A}_i\}$, best estimate $\hat{\mathbf{A}}$.
3: Construct $G_0(V_0,E_0,W_0)$ with adjacency matrix $\mathbf{A}_0$.
4: Initialise $i:=0$, $C_0=\{\}$.
5: **while** $|E_i|>1$ **and** $i<N$ **and not** $S(\cdot)$ **do**
6: Increment i such that $i:=i+1$.
7: Determine edges $R_i:=\{(s_i^{(m)},d_i^{(m)})\}_{m=1}^{M_i}$ to remove from G_{i-1}.
8: Set $C_i:=C_{i-1}\cup R_i$.
9: Obtain $\mathbf{A}_i$, the estimate of $\mathbf{A}$ with C_i constrained to equal zero. Note that this will require evaluating the Kalman filtering equations.
10: Construct $G_i(V_i,E_i,W_i)$ as the graph with adjacency matrix $\mathbf{A}_i$.
11: **end while**
12: Set $I:=i$ the number of iterations performed.
13: Determine which best estimate $\hat{\mathbf{A}}$ to output.
14: Output $\{\mathbf{A}_i\}_{i=1}^{I}$ and $\hat{\mathbf{A}}$.

In Alg. 8, we describe our framework, iterative graphical clustering for linear Gaussian state-space models (IGC-LGSSM), which estimates the transition matrix $\mathbf{A}$ of a LGSSM such that the resulting estimate exhibits a clustered structure.

As input to out method, (line 1), we take values of the model parameters except $\mathbf{A}$, an initial estimate $\mathbf{A}_0$, a maximum number of iterations N, and if desired, an early stopping criterion $S(\cdot)$, which we discuss in Sec. 4.3.4.

Our algorithm returns (line 2) the sequence of estimates $\{\mathbf{A}_i\}_{i=1}^{I}$, as well as a single, best, estimate $\hat{\mathbf{A}}$, (see Sec. 4.3.5 for discussion).

We then construct the graph $G_0(V_0,E_0,W_0)$ with adjacency matrix equal to the provided $\mathbf{A}_0$ (line 3), as well as initialise our iteration counter i to zero and the initial set of constrained sparse edges C_0 to the empty set (line 4). We then iterate (line 5) until either

- there are one or fewer edges remaining in the estimated graph ($|E_i|>1$), or
- the maximum number of iterations is reached ($i<N$), or
- the early stopping criterion is met (**not** $S(\cdot)$).

At the ith iteration, we perform the following steps. First, we determine the additional set of edges R_i that we will constrain to be sparse at this iteration (line 7), which we discuss in Sec. 4.3.2. We then combine these new constraints with our existing set of constraints C_{i-1} by setting $C_i=C_{i-1}\cup R_i$ (line 8). Next, we estimate the transition matrix $\mathbf{A}$, constraining the edges contained in C_i to be sparse (line 9), denoting the result $\mathbf{A}_i$, with the method by which estimation is performed detailed in Sec. 4.3.3. Note that this is by far the most expensive step of the method, and requires evaluating the Kalman filter and RTS smoother several times, as per [16]. Finally for each iteration, we construct the graph $G_i(V_i,E_i,W_i)$ with adjacency matrix $\mathbf{A}_i$ (line 10).

After we finish iterating, we denote the number of iterations by I (line 12). Then, we determine the best estimate $\hat{\mathbf{A}}$ of $\mathbf{A}$ from $\{\mathbf{A}_i\}_{i=1}^{I}$ (line 13), which we discuss in Sec. 4.3.5. Finally, we return both the sequence of estimates $\{\mathbf{A}_i\}_{i=1}^{I}$, as well as the best estimate $\hat{\mathbf{A}}$ (line 14).

4.3.2 Learning cluster-inspired constraints

We promote cluster recovery through iteratively constraining elements that violate the cluster structure. We propose a pair of schemes for iteratively imposing constraints which promote a cluster structure; one scheme based on minimum cuts, and another scheme based on EBC. Each scheme utilises methods from different disciplines, with minimum cuts coming from cluster analysis, and EBC heralding from community detection. Algorithm 8 changes only in line 7 when changing the constraining scheme, with all other steps remaining the same. At iteration i, we have the estimated transition matrix $\mathbf{A}_{i-1}$, which is the adjacency matrix for the weighted directed graph G_{i-1}. Further, we have constrained that the edges $C_i=\{(s_c,d_c)\}_c$ are not present in G_{i-1}.

Minimum cut clustering

We separate G_{i-1} into its weakly connected components $\{K_{i-1}^{(n)}\}_{n=1}^{N_i}$, where N_i is the number of weakly connected components, and we define a weakly connected (di)graph as a graph where it is possible to reach any node from any other node by traversing edges in any direction (i.e. not necessarily the direction in which the edges point). Denote by $R_c = \{(s_n,d_n)\}_{n=1}^{N_c}$ the N_c edges removed by the minimum cut of the sub-graph $K_{i-1}^{(c)}$. Next, we estimate $\mathbf{A}_i$ in parallel, with the pth parallel branch imposing that the elements of $\mathbf{A}_i$ corresponding to $R_p \cup C_i$ are zero, denoting the resulting estimate by $\mathbf{A}_i^{(p)}$. We then set $\mathbf{A}_i$ to the $\mathbf{A}_i^{(\hat{p})}$ with the highest likelihood, and set $C_{i+1} = C_i \cup R_{\hat{p}}$.

EBC clustering

For each edge (s,d) of G_{i-1}, we compute the EBC, which we denote by $\mathrm{EBC}(s,d)$. We find all edges (s,d) such that $(s,d) = \mathrm{argmin}_{(s,d)} \mathrm{EBC}(s,d)$, denoting this set by $\{(s_i^{(c)},d_i^{(c)})\}_{c=1}^{C_i}$. We denote the edge(s) removed by $\{(s_i^{(m)},d_i^{(m)})\}_{m=1}^{M_i}$, where M_i is the number of edges removed. Finally, we estimate the transition matrix whilst constraining edges $C_{i+1} = C_i \cup \left(\bigcup_{k=1}^{i} \{(s_k^{(m)},d_k^{(m)})\}_{m=1}^{M_k} \right)$ constrained to equal 0, which we denote by $\mathbf{A}_i$.

4.3.3 Estimating the transition matrix under constraints

In this chapter, we require to estimate the transition matrix under constraints. The constraints that we wish to apply are that specific elements of $\mathbf{A}$ are fixed to be zero, which is equivalent to imposing cuts on the graph with $\mathbf{A}$ as adjacency matrix. Specifically, if at iteration i we remove edges $\{(s_i,d_i)\}_i$, we require that the elements $A_{d_i,s_i} = 0 \ \forall i$. Furthermore, it is desirable for the estimate to be sparse in elements that are not explicitly constrained to be sparse, as this will accelerate inference by reducing the number of iterations required.

In this chapter, we use GraphEM [16], which utilises the proximal operator of the $L1$ norm to promote sparsity within the estimate for $\mathbf{A}$. However, this method does not natively admit constraints, but we can impose that $A_{ij} = 0$ for any $1 \leq i,j \leq N_x$ by modifying the update step [16, Alg. 4], such that we set the values of the constrained elements to 0, and then perform estimation updates only on the elements corresponding to unconstrained elements of the transition matrix.

4.3.4 Online stopping condition

We utilise an early stopping criterion to reduce the computational cost of our method. This condition is designed to balance the cost of exploring the parameter space by

performing more iterations, with the potential improvements from performing more iterations. Experimentally, we observe that, when performing iteration with increasing numbers of constraints, the likelihood of the estimate at each iteration, which encodes the capability of the estimate to recover the observed behaviour, is generally lower than at the preceding iteration, owing to the use of a maximisation scheme when performing estimation.

Therefore, we propose as our stopping criteria to cease estimation once the likelihood of the estimate decreases below a given threshold. This is implemented via the stopping condition $S(\cdot)$, which is set to be

$$S(\cdot)=S(\mathbf{A}_i)=\begin{cases}\text{true} & p(\mathbf{y}_{1:T}|\mathbf{A}_i)<\alpha p(\mathbf{y}_{1:T}|\mathbf{A}_0),\\ \text{false} & \text{otherwise,}\end{cases} \tag{4.2}$$

where $0<\alpha<1$ is a hyper-parameter. Note that this assumes the initial value $\mathbf{A}_0$ is an estimate of $\mathbf{A}$, not a random value. The likelihood can be cheaply obtained via the Kalman filter, which we already evaluate when estimating $\mathbf{A}$, so this method add very little overhead to each iteration, and drastically decreases the cost overall, as estimates that would never be used due to poor explanatory capability are not calculated.

4.3.5 Offline estimate selection

We propose two methods for selecting an estimate from the set of recovered estimates. The first method is designed explicitly to balance interpretability and representative power, whereas the second allows an interpretation of our scheme as optimising a heuristic aiming to minimise some other cost.

In the first instance, balancing interpretability against the recovery of dynamics, we aim to recover the estimate that has the most constraints before the likelihood begins to be significantly adversely impacted. We can recover this point by fitting a smoothing spline $s(i)$ to the likelihood function $l(i)=p(\mathbf{y}_{1:T}|\mathbf{A}_i)$, and finding the (integral) value $\hat{i}$ such that the second derivative is minimised, i.e.,

$$\hat{\mathbf{A}}=\mathbf{A}_{\hat{i}},\quad \hat{i}=\underset{i}{\operatorname{argmin}}(s''(i)),\ i\in\mathbb{N}_{\leq I}. \tag{4.3}$$

The second scheme comes about if we choose the estimate that minimises a cost function. An appropriate cost would be one that is intractable to directly optimise, such as an $L0$ penalised loss, as many costs with smooth convex penalties can be directly optimised by our chosen estimation scheme [16, 34]. For example, we as we

wish to obtain a sparse model, we may wish to minimise

$$f(\mathbf{A}) = \lambda ||\mathbf{A}||_0 - \log(p(\mathbf{y}_{1:T}|\mathbf{A}_i)), \tag{4.4}$$

where a larger λ gives a preference for higher sparsity.

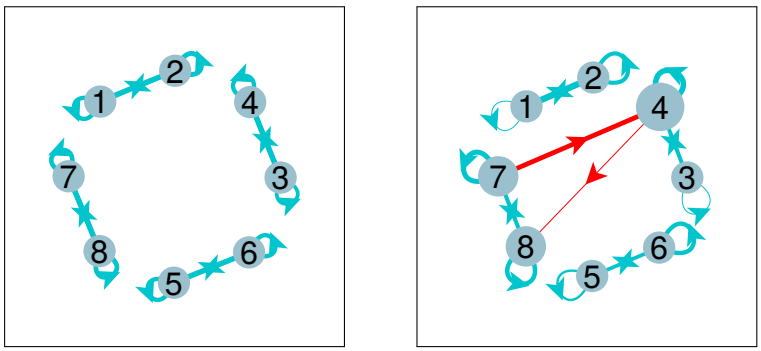
Note that, when $\lambda = 1$, minimising Eq. (4.4) is equivalent to minimising the Akaike information criterion (AIC), thereby lending our estimation scheme a similar interpretation to automatic regression using AIC as the selection mechanism; however, in our case, the set of terms is fixed, and we impose constraints rather than adding in new terms to a regression, thereby making the scheme more principled than typical automatic regression methods. If this scheme is used, our scheme becomes a cost heuristic optimiser, where we optimise a heuristic (given by [16, Eq. (31)]) under several conditions (in this case, the iterative constraints), and select the estimate which minimises our cost function.

4.4 Numerical study

Experimental setup. In order to demonstrate our method, we generate observations following Eq. (4.1), with $N_x = N_y = 8$, $\mathbf{R} = \mathbf{P}_0 = \mathbf{Q} = \mathbf{H} = \mathbf{Id}_{N_x}$, $\bar{\mathbf{x}}_0 = \mathbf{1}_{N_x}$ and a series length of $T = 100$. We generate $\mathbf{A}$ to be a block diagonal matrix, with 2×2 blocks of i.i.d. Uniform(0,1) elements, which we normalise so as to have absolute maximal singular value of 1, yielding a stable system. The resulting graph of state relationships is given in Fig. 4.1a. We will compare IGC-LGSSM (Alg. 8) with GraphEM [16], a recent method that provides sparse estimates of the transition matrix in a LGSSM. We compare both methods to a dense maximum likelihood (ML) method as a baseline [5, Chapter 13] [11].

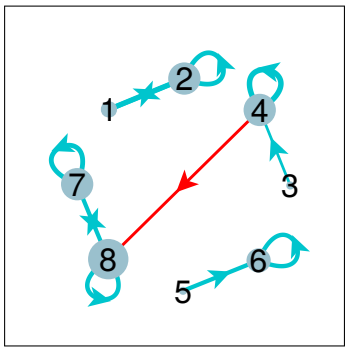
We utilise the minimum cut method outlined in Section 4.3.2 to determine which elements of $\mathbf{A}$ to constrain at each iteration, noting that in our testing both methods proposed in Section 4.3.2 performed comparably. We set our stopping condition as Section 4.3.4, with $\alpha = 0.9$. We set λ as the penalty for GraphEM, with λ selected via coordinate descent targeting F1 on a separate system used only for this purpose. We utilise GraphEM as the estimation method for IGC-LGSSM, with the same λ utilised as the penalty parameter. The initial value $\mathbf{A}_0$ for GraphEM and IGC-LGSSM is the ML estimate; $\mathbf{A}_0$ for ML is an element-wise normally distributed matrix.

Initial demonstration. We present the ground truth connectivity in Fig. 4.1a. We see in Fig. 4.1b that GraphEM recovers false edges in addition to the true edges. Furthermore, increasing the penalty parameter λ causes GraphEM to remove true edges before falsely recovered edges, as seen in Fig. 4.1c. Our method recovers the state relationships in their entirety with no false connections, as seen in Fig. 4.1d.

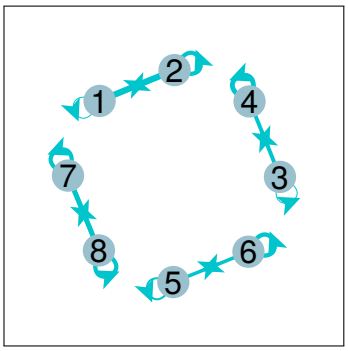


(a) True graph

(b) GraphEM estimate ($\lambda = 1.3$)



(c) GraphEM estimate ($\lambda = 1.4$)



(d) IGC-LGSSM estimate

Figure 4.1: True and estimated graphs. Edge thickness is proportional to the magnitude of the elements, with arrows denoting direction. The size of nodes is proportional to their degree. Blue denotes an estimated edge that exists in the true graph, and red denotes an estimated edge that does not exist in the true graph.

Variable state dimension. We now compare the performance of GraphEM and IGC-LGSSM applied to a number of systems. The true transition matrix in all systems is a block diagonal composed of 2×2 blocks, with the blocks generated as before, resulting in matrices with many sparse elements. All other parameters are set as previously, except the number of time-steps T, which we now set to $T = 250$, with $N_x = N_y$ varied per experiment.

We compare the metrics of specificity (spec.), precision (prec.), recall, and F1 score when assessing sparsity. We denote an edge being present as positive and no edge as negative for these calculations. We also compute the root mean square error (RMSE) of the estimates for each method. Performance metrics are averaged over 100 independently initialised runs. We present the results of these experiments in Table 4.1. We see that IGC-LGSSM generally outperforms GraphEM in terms of edge detection and in recovering the graph structure, especially on combined metrics such as F1 score. In particular, GraphEM recovers many dense edges as sparse, which IGC-LGSSM avoids, as these edges are required to represent the dynamics in the absence of the

Table 4.1
Metrics of GraphEM and IGC-LGSSM over variable state dimension.

N_x	method	RMSE	spec.	recall	prec.	F1
8	GraphEM	0.08	0.97	0.73	0.97	0.83
	IGC-LGSSM	**0.07**	**0.97**	**0.97**	**0.99**	**0.99**
	ML	0.10	–	–	–	–
16	GraphEM	0.14	**0.98**	0.69	0.96	0.81
	IGC-LGSSM	**0.12**	0.97	**0.95**	**0.97**	**0.95**
	ML	0.16	–	–	–	–
36	GraphEM	0.20	**0.97**	0.65	**0.96**	0.79
	IGC-LGSSM	**0.17**	0.96	**0.90**	0.95	**0.92**
	ML	0.25	–	–	–	–
124	GraphEM	0.23	**0.96**	0.57	**0.98**	0.71
	IGC-LGSSM	**0.20**	0.96	**0.89**	0.94	**0.92**
	ML	0.34	–	–	–	–

incorrectly initially recovered inter-cluster edges. Furthermore, IGC-LGSSM has better RMSE scores than GraphEM, with both methods consistently beating the maximum likelihood baseline. On inspection of the estimates, this improvement in RMSE is also due to the superior recovery of dense elements.

We also test IGC-LGSSM for variable noise magnitude $\mathbf{Q}$ for $N_x = 36$, which is displayed in Table 4.2. We see that IGC-LGSSM yields better estimates than GraphEM over all tested covariances. We see that IGC-LGSSM generally recovers dense elements more reliably than GraphEM, which is apparent from the recall metric being consistently higher.

Table 4.2
Metrics of GraphEM and IGC-LGSSM over variable noise covariance.

$\mathbf{Q}$	method	RMSE	spec.	recall	prec.	F1
$0.1\mathbf{Id}_{36}$	GraphEM	0.15	0.98	0.78	**0.97**	0.85
	IGC-LGSSM	**0.13**	**0.98**	**0.93**	0.96	**0.95**
	ML	0.18	–	–	–	–
$\mathbf{Id}_{36}$	GraphEM	0.20	**0.97**	0.65	**0.96**	0.79
	IGC-LGSSM	**0.17**	0.96	**0.90**	0.95	**0.92**
	ML	0.25	–	–	–	–
$1.5\mathbf{Id}_{36}$	GraphEM	0.26	0.95	0.62	**0.96**	0.80
	IGC-LGSSM	**0.21**	0.92	0.90	0.95	**0.92**
	ML	0.32	–	–	–	–
$2\mathbf{Id}_{36}$	GraphEM	0.32	**0.94**	0.58	0.94	0.73
	IGC-LGSSM	**0.25**	0.89	**0.85**	**0.95**	**0.88**
	ML	0.36	–	–	–	–

4.5 Conclusion

In this chapter, we have proposed the IGC-LGSSM framework, a novel methodology that combines a graphical interpretation of the state-space dynamics with a robust and extensible clustering method to promote a clustered structure when estimating the transition matrix of a linear-Gaussian state-space model. Our method is interpretable by design, and exhibits superior performance to the existing state-of-the-art for sparse parameter estimation in LGSSMs. Furthermore, due to the modular design of our method allowing the use of any parameter estimation scheme for the transition matrix, so long as element-wise sparsity constraints can be applied, our method will benefit from any future research in the field.

Chapter 5

Approximate Learning of Transition Dynamics in General State-Space Models

This chapter is based on the papers [75, 76].

5.1 Introduction

In practice many systems are non linear, and thus the methods of Chapters 3 and 4 are approximations at best. Therefore, we develop a method for sparse parameter estimation in non-linear state-space models, using the particle filter for state recovery rather than the Kalman filter, thereby allowing the methodology to be applied more generally.

The general state-space model does not have a form imposed as in the linear-Gaussian state-space model, and in order to perform filtering of a state-space model utilising the particle filter, we must know the form of the transition and observations models, and the value of all parameters therein. However, in practice, it is common for the parameters to be unknown, and therefore require estimation. Furthermore, in many cases, the form of the model is not known, and therefore we must impose or learn a form before we can estimate anything. In the case of dynamical systems where we observe the state directly, methods such as SINDy [77] can be used to estimate the system dynamics in the case they are unknown. However, these methods cannot be utilised for incompletely observed noisy dynamics, as is the case in state-space models.

Estimating the parameters of a general non-linear SSM is a difficult task, even when the model is known. One challenge is that the exact likelihood is usually intractable for general non-linear models, and we must use a stochastic estimate utilising the particle weights of the particle filter. When using this estimate, a further problem is that we

cannot efficiently compute gradients of the likelihood, and hence we cannot easily compute the maximum likelihood estimator, nor utilise standard modern sampling schemes such as Hamiltonian Monte Carlo. However, thanks to the recent advent of differentiable particle filters [36, 78, 79], it recently became possible to obtain the gradient of the estimated likelihood of model parameters with respect to the parameter values, and hence became much simpler to fit parameters of non-linear state-space models using, e.g., maximum likelihood, assuming the form of the model is known.

In the unknown model case, we can make use of several simplifying assumptions to design a model form that well approximates a large family of models. In dynamical systems, many phenomena occur with rates proportional to a polynomial function of the state dimensions. For example, in population modelling, birth-immigration-death-emigration models are often used, within which the dynamics of the rates are dependent on 2nd degree polynomials of the state-space. Furthermore, the well known Lorenz 63 [80] and Lorenz 96 [81] models both have rates of change described by 2nd degree polynomials, and are known to be highly chaotic systems of equations. We can therefore see that polynomial systems can describe complex phenomena, and hence a crucial problem is to learn the coefficients of a polynomial approximation to the transition function of a NLSSM.

Contribution. In this chapter, we propose GraphGrad, a method to model and learn the transition distribution of a non-linear SSM via a polynomial approximation of the state.

- The proposed approximation can represent a rich class of systems, as many dynamical systems are described by a series of differential equations that are polynomial functions of the states. The resulting systems are interpretable, with the interactions between variables having interpretation similar to the rate terms in a dynamical system.

- Our method uses a differentiable particle filter, allowing us to use a first order optimisation scheme to perform parameter estimation, improving robustness whilst decreasing computation time. Furthermore, we promote sparse systems by the use of a proximity update, thereby increasing interpretability, and improving stability compared to naive subgradient updates of a penalised loss.

- GraphGrad is unsupervised, as we optimise the (penalised) parameter log posterior density, and therefore do not require knowledge of the underlying hidden state to be trained.

- GraphGrad is fast and efficient to evaluate, and provides excellent performance

in challenging scenarios, even when the underlying system is not of polynomial form, i.e., under model mismatch.

5.2 Background

5.2.1 Considered state-space model

In this chapter, we consider a general SSM, described by

$$\begin{aligned} \mathbf{x}_t &\sim p(\mathbf{x}_t|\mathbf{x}_{t-1};\boldsymbol{\theta}), \\ \mathbf{y}_t &\sim p(\mathbf{y}_t|\mathbf{x}_t;\boldsymbol{\theta}), \end{aligned} \tag{5.1}$$

where $t=1,...,T$ denotes discrete time, $\mathbf{x}_t \in \mathbb{R}^{N_x}$ is the state of the system at time t, $\mathbf{y}_t \in \mathbb{R}^{N_y}$ is the observation at time t, $p(\mathbf{x}_t|\mathbf{x}_{t-1};\boldsymbol{\theta})$ is the density of the hidden state $\mathbf{x}_t$, given the previous state $\mathbf{x}_{t-1}$, $p(\mathbf{y}_t|\mathbf{x}_t;\boldsymbol{\theta})$ is the density of the observation $\mathbf{y}_t$ given the hidden state $\mathbf{x}_t$, and $\boldsymbol{\theta}$ is a set of model parameters. The initial value of the hidden state is distributed $\mathbf{x}_0 \sim p(\mathbf{x}_0|\boldsymbol{\theta})$. The state sequence $\mathbf{x}_{0:T}$ is typically hidden, whilst the related sequence of observations $\mathbf{y}_{1:T}$ is known.

Filtering methods aim at estimating the hidden state at time t, typically utilising the posterior density function of the state conditional on the observations up to time t, denoted $p(\mathbf{x}_t|\mathbf{y}_{1:t};\boldsymbol{\theta})$. Particle filters approximate $p(\mathbf{x}_t|\mathbf{y}_{1:t};\boldsymbol{\theta})$ using a set of K Monte Carlo samples, called particles, and their associated weights, collectively denoted by $\{\mathbf{x}_{1:T}^{(k)}, \overline{w}_{1:T}^{(k)}\}_{k=1}^{K}$. The posterior density can then be approximated by

$$p(\mathbf{x}_t|\mathbf{y}_{1:t};\boldsymbol{\theta}) \approx \sum_{k=1}^{K} \overline{w}_t^{(k)} \delta_{\mathbf{x}_t^{(k)}}. \tag{5.2}$$

Algorithm 9 Sequential importance resampling (SIR) particle filter

1: **Input:** Observations $\mathbf{y}_{1:T}$, parameters $\boldsymbol{\theta}$
2: **Output:** Hidden state estimates $\mathbf{x}_{1:T}$, particle weights $w_{1:T}$
3: Draw $\mathbf{x}_0^{(k)} \sim p(\mathbf{x}_0|\boldsymbol{\theta})$, for $k=1,...,K$
4: Set $\overline{w}_0^{(k)} = 1/K$, for $k=1,...,K$
5: **for** $t=1,...,T$ and $k=1,...,K$ **do**
6: Sample $a_t^{(k)} \sim \text{Categorical}(\overline{w}_{t-1})$.
7: Set $\overline{w}_{t-1}^{(k)} = 1/K$.
8: Sample $\mathbf{x}_t^{(k)} \sim \pi(\mathbf{x}_t|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)}, \mathbf{y}_t;\boldsymbol{\theta})$.
9: Compute $w_t^{(k)} = \frac{p(\mathbf{y}_t|\mathbf{x}_t^{(k)};\boldsymbol{\theta}) p(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)};\boldsymbol{\theta})}{\pi(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)}, \mathbf{y}_t;\boldsymbol{\theta})}$.
10: Compute $\overline{w}_t^{(k)} = w_t^{(k)} / \sum_{i=1}^{K} w_t^{(i)}$.
11: **end for**

A commonly used particle filtering method is the sequential importance resampling algorithm, given in Alg. 3. We reproduce this in Alg. 9, with minor modifications for ease of reference. At every time-step t, the K particles and normalised weights, $\{\mathbf{x}_{1:T}^{(k)}, \overline{w}_{1:T}^{(k)}\}_{k=1}^{K}$, are calculated. First, we perform the resampling step (line 6), which generates K samples, sampling $\mathbf{x}_{t-1}^{(k)}$, $k=1,...,K$, with probability $\overline{w}_{t-1}^{(k)}$. The resampling step is vital to avoid the degeneracy of the filter, i.e., to ensure diversity in the particle set and obtain more accurate approximations of the posterior distribution, $p(\mathbf{x}_t|\mathbf{y}_{1:t};\boldsymbol{\theta})$. Next, K particles $\mathbf{x}_t^{(k)}$, $k=1,...,K$, are drawn from the proposal distribution $\pi(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{y}_t;\boldsymbol{\theta})$ (line 8). Finally, we incorporate the observation $\mathbf{y}_t$, which is done via the particle weights, given by $w_t^{(k)}$, $k=1,...,K$, in line 9, and the normalised weights (line 10).

5.2.2 Parameter estimation in state-space models

In many problems of interest, the parameter $\boldsymbol{\theta}$ is not known, and must be estimated. The posterior distribution of $\boldsymbol{\theta}$ in the SSM can be factorised as $p(\boldsymbol{\theta}|\mathbf{y}_{1:T}) \propto p(\boldsymbol{\theta})p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the parameter of interest, $p(\boldsymbol{\theta})$ is the prior distribution of the parameter $\boldsymbol{\theta}$, and $p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$ is given by

$$p(\mathbf{y}_{1:T}|\boldsymbol{\theta}) = \prod_{t=1}^{T} p(\mathbf{y}_t|\mathbf{y}_{1:t-1};\boldsymbol{\theta}), \tag{5.3}$$

where $p(\mathbf{y}_1|\mathbf{y}_{1:0};\boldsymbol{\theta}) := p(\mathbf{y}_1|\boldsymbol{\theta})$ [5]. The prior distribution $p(\boldsymbol{\theta})$ encodes our pre-existing beliefs as to the value and structure of the parameter $\boldsymbol{\theta}$, and can be used for regularisation, e.g., by the Lasso [28].

There are many methods to estimate parameters $\boldsymbol{\theta}$ given its posterior density function $p(\boldsymbol{\theta}|\mathbf{y}_{1:T})$. We can broadly classify these parameter estimation methods as point estimation methods and distributional methods. Point estimation methods provide a single estimate that is, in some defined way, the optimal value. An example of a point estimation method is the maximum-a-posteriori (MAP) estimator, that defines the optimal value of $\boldsymbol{\theta}$ as the one that maximises the posterior density $p(\boldsymbol{\theta}|\mathbf{y}_{1:T})$. The method proposed in this chapter yields a MAP estimator.

In the case of a linear-Gaussian SSM, we have closed form methods for the MAP estimator assuming a diffuse prior [5], and efficient methods for obtaining the MAP estimator for sparsity promoting priors [16]. Distributional methods estimate the posterior distribution of the parameter, with common methods being importance sampling [18], Markov chain Monte Carlo [10], and variational inference [20]. For the linear-Gaussian SSM, we can utilise reversible jump Markov chain Monte Carlo to obtain an estimate of the distribution of sparsity [35], however no similar method exists

for the general SSM. Our proposed method in this chapter is a point estimation method.

For general SSMs, the posterior density of the parameter $\boldsymbol{\theta}$, necessary to compute the posterior and hence to design a procedure to maximise it, cannot be obtained in closed form. Let $\nu_t^{(k)} = w_t^{(k)} \overline{w}_{t-1}^{(k)}$ be the adjusted importance weight. We can estimate the parameter likelihood using the particle filter via the Monte Carlo estimate

$$p(\mathbf{y}_{1:T}|\boldsymbol{\theta}) \approx \prod_{t=1}^{T} \left(\sum_{k=1}^{K} \nu_t^{(k)} \right), \tag{5.4}$$

from which we obtain an estimate of the posterior density of our parameter

$$\begin{aligned} p(\boldsymbol{\theta}|\mathbf{y}_{1:T}) &\propto p(\boldsymbol{\theta}) p(\mathbf{y}_{1:T}|\boldsymbol{\theta}), \\ &\approx p(\boldsymbol{\theta}) \prod_{t=1}^{T} \left(\sum_{k=1}^{K} \nu_t^{(k)} \right), \end{aligned} \tag{5.5}$$

where $w_t^{(k)}$ and $\overline{w}_{t-1}^{(k)}$ are the weights of the particle filter as in Alg. 9 [5, Chapter 12]. Note that the weights are dependent on the parameter $\boldsymbol{\theta}$ through their computation in Alg. 9. Further, we construct an estimate of the log posterior density by

$$\begin{aligned} \log(p(\boldsymbol{\theta}|\mathbf{y}_{1:T})) + c &= \log(p(\boldsymbol{\theta})) + \log(p(\mathbf{y}_{1:T}|\boldsymbol{\theta})), \\ &\approx \log(p(\boldsymbol{\theta})) + \sum_{t=1}^{T} \log \left(\sum_{k=1}^{K} \nu_t^{(k)} \right), \end{aligned} \tag{5.6}$$

where the constant c results from the proportionality in Eq. (5.5), and we note that if resampling occurred at time $t-1$, which is always the case if following Alg. 9, we have $\overline{w}_{t-1}^{(k)} = 1/K$, and hence $\nu_t^{(k)} = w_t^{(k)}$ due to the weights being normalised. Note that, in practice, log-weights are used for numerical stability, and we compute $\sum_{k=1}^{K} w_t^{(k)} \overline{w}_{t-1}^{(k)}$ using a logsumexp reparametrisation.

5.2.3 Differentiable particle filter

The outputs of the particle filter, as given in Alg. 9, namely $\{\mathbf{x}_{1:T}^{(k)}, w_{1:T}^{(k)}\}_{k=1}^{K}$, are not differentiable with respect to $\boldsymbol{\theta}$ [79], because of the resampling step on line 6. The resampling step requires sampling a multinomial distribution. Sampling a multinomial distribution is not differentiable, as an infinitesimal change in the input probabilities can lead to a discrete change in the output sample value [79].

Differentiable particle filters (DPFs) are recently introduced tools that modify the resampling step of the SIR particle filter to be differentiable, and therefore lead to a particle filtering algorithm that is differentiable with respect to $\boldsymbol{\theta}$ [78, 37, 79, 82].

Algorithm 10 Stop-gradient differentiable particle filter (DPF) [79]

1: **Input:** Observations $\mathbf{y}_{1:T}$, parameters $\boldsymbol{\theta}$
2: **Output:** Hidden state estimates $\mathbf{x}_{1:T}$, particle weights $w_{1:T}$
3: Draw $\mathbf{x}_0^{(k)} \sim p(\mathbf{x}_0|\boldsymbol{\theta})$, for $k=1,...,K$
4: Set $\overline{w}_0^{(k)} = 1/K$, for $k=1,...,K$
5: **for** $t=1,...,T$ and $k=1,...,K$ **do**
6: Sample $a_t^{(k)} \sim \text{Categorical}(\perp(\overline{w}_{t-1}))$.
7: Set $\overline{w}_{t-1}^{(k)} = \frac{1}{K}\, \overline{w}_{t-1}^{a_t^{(k)}} / \perp(\overline{w}_{t-1}^{a_t^{(k)}})$.
8: Sample $\mathbf{x}_t^{(k)} \sim \pi(\mathbf{x}_t|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)}, \mathbf{y}_t; \boldsymbol{\theta})$.
9: Compute $w_t^{(k)} = \frac{p(\mathbf{y}_t|\mathbf{x}_t^{(k)};\boldsymbol{\theta}) p(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)};\boldsymbol{\theta})}{\pi(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)}, \mathbf{y}_t;\boldsymbol{\theta})}$.
10: Compute $\overline{w}_t^{(k)} = w_t^{(i)} / \sum_{k=1}^{K} w_t^{(k)}$.
11: **end for**

where $\perp(x)$ is the stop gradient operator as defined in [79], which yields x when evaluated, but always has a gradient of 0.

There exist a number of DPF methods, based on various techniques for making the resampling step differentiable. These range from weight retention in soft resampling [36], to optimal transport [78]. An overview of the DPF and its interplay with deep learning methods can be found in [37], which motivates our choice of differentiable particle filter, and our usage of stochastic gradient descent methods.

In this chapter, we built upon the DPF approach from [79], which utilises a stop gradient operator to make the resampling step differentiable. We summarise this method in Alg. 10. This algorithm yields gradient estimates with minimal computational overhead, and does not modify the behaviour of the forward pass of the particle filter.

In Alg. 10, at each time-step t, we first sample the previous particle set, sampling $a_t^{(k)}$, $k=1,...,K$, with probability $\overline{w}_{t-1}$ (line 6). The value of $a_t^{(k)}$ determines the ancestry of the k-th particle at time t. Note that we apply a stop gradient operator to the weights in the sampling method, and therefore do not attempt to propagate gradients through sampling a discrete distribution. We then set the weights of all K particles to $1/K$ whilst preserving gradient information in the weights by dividing the weights by themselves, applying a stop-gradient operation to the divisor, and then multiplying by the constant $1/K$ (line 7). The rest of the DPF proceeds as Alg. 9, described in Sec. 5.2.1, with the particle sampling, weighting, and normalisation steps unmodified.

The DPF is particularly useful when estimating parameters, as we can compute the gradients of functions of the posterior density and of the particle trajectories, and therefore compute parameters optimising a chosen loss function. In particular, we typically use likelihood-based losses when performing inference where the hidden state is unknown in the training data, and trajectory-based losses when the sequence of hidden states is known for the training data. Examples of likelihood-based losses are the negative log-likelihood and the ELBO, as well as the (negative) posterior density.

The mean-square error of the inferred state is a typical trajectory-based loss. Note that the trajectory of the hidden states depends on the weights, and therefore the weights must be differentiable even if the loss function is based only on the trajectory.

Given a loss function based on the weights and/or particles of a differentiable particle filter, we can compute the minimiser using a gradient-based optimisation scheme. This is efficient, especially compared to the gradient-free methods previously required, and is more robust to the stochasticity of the likelihood and state estimates, as many gradient schemes are designed with noisy gradients in mind [38, 83].

5.2.4 Notation

We here present our notation, used throughout the chapter. We denote by $\mathbf{A}_{\cdot,i}$ the i-th column vector of matrix $\mathbf{A}$, and by $\mathbf{A}_{i,\cdot}$ the i-th row vector of matrix $\mathbf{A}$. We denote by $\mathbf{Id}_n$ the $n \times n$ identity matrix.

We denote by $\mathbb{1}_{\text{cond}}(x)$ the binary indicator function, which returns 1 when x satisfies cond, and 0 otherwise. We denote by $\text{sgn}(x) = \mathbb{1}_{\geq 0}(x) - \mathbb{1}_{\leq 0}(x)$ the sign function. We denote by $\text{abs}(x)$ the absolute value function. All three functions can be applied element-wise to a matrix or vector.

We denote by $\mathbb{N}_0$ the natural numbers including 0, and by $\mathbb{N}$ the natural numbers excluding 0.

We denote by $\text{count}(a,b)$ the number of times the item a occurs in the list b, and by $\text{cwr}(S,r)$ the set of all length r combinations of the elements of the set S with replacement, up to reordering. For example, we have $\text{count}(1,[1,2,2,3,1]) = 2$ and $\text{cwr}(\{a,b,c\},2) = \{[a,a],[a,b],[a,c],[b,b],[b,c],[c,c]\}$.

We denote by $\perp(x)$ the stop-gradient operator applied to x, with properties as defined in [79].

5.3 Sparse estimation of non-linear SSMs

This section presents our main contribution, a novel approach for modelling and learning the state transition distribution of a general SSM utilising a polynomial approximant. Several dynamical systems can be exactly represented as polynomials, such as the chaotic Lorenz 63 and 96 systems [80, 81] or the Rabinovich-Fabrikant system [84], the Lotka-Volterra model and many of its extensions [85], many compartmental epidemiological models [86], and several ODEs resulting from PDE discretisations (such as from the Brusselator model [87] and Oregonator model [88]). These systems with different properties and dynamics can all be represented exactly by a polynomial

model, demonstrating the wide array of systems that a polynomial approximation can exactly capture.

Our approach builds a polynomial approximation to the transition distribution, parametrised by $\mathbf{C} \in \mathbb{R}^{N_x \times M}$, a matrix of real numbers encoding polynomial coefficients, to be learnt, and $\mathbf{D} \in \mathbb{N}_0^{N_x \times M}$ a fixed (i.e., known) integer matrix of monomial degrees associated with $\mathbf{C}$.

The positive integer d denotes the fixed maximum degree of our polynomial approximation. The number of monomials of degree d in N_x variables is $M = \sum_{n=0}^{d} \binom{n+N_x-1}{N_x-1}$. No other model parameters are assumed unknown, hence $\boldsymbol{\theta}$ is equal in our case to $\mathbf{C}$, and, in particular, $p(\mathbf{x}_t|\mathbf{x}_{t-1};\boldsymbol{\theta}) = p(\mathbf{x}_t|\mathbf{x}_{t-1};\mathbf{C})$. For example, in the additive zero-mean Gaussian noise case, we have

$$\mathbf{x}_t \sim p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{C}) := \mathcal{N}(f(\mathbf{x}_{t-1},\mathbf{C};\mathbf{D}),\boldsymbol{\Sigma}_v), \tag{5.7}$$

where $\boldsymbol{\Sigma}_v$ is the covariance of the state noise distribution, and, for every $\mathbf{x} = [x_1,x_2,...,x_{N_x}]^\top \in \mathbb{R}^{N_x}$,

$$f(\mathbf{x},\mathbf{C};\mathbf{D}) = \sum_{j=1}^{M} \left(\mathbf{C}_{\cdot,j} \cdot \prod_{i=1}^{N_x} x_i^{D_{i,j}} \right). \tag{5.8}$$

is our considered polynomial approximation. The degree matrix $\mathbf{D}$ is constructed from the number of hidden states N_x and the maximum degree d, and is a static parameter.

The remainder of this section is structured as follows. Our polynomial model is thoroughly described in Section 5.3.1. We then design an approach to learn the coefficient matrix $\mathbf{C}$ using a MAP estimator under a sparsity inducing penalty. We next discuss in Section 5.3.2 the graphical interpretation of $\mathbf{C}$ and the key role of sparsity. We then use the Lorenz 63 dynamical system as a pedagogical example to better illustrate the usefulness of our model, in Section 5.3.3. We move to the presentation of our approach to estimate $\mathbf{C}$, starting from the problem definition in Section 5.3.4, a single batch algorithm S-GraphGrad in Section 5.3.5, and a practical mini-batch implementation, leading to our final algorithm B-GraphGrad, in Section 5.3.6.

5.3.1 Constructing a polynomial approximant

In order to learn a polynomial approximation to the state transition distribution, let us first define the learnt and static parameters of our polynomials. A polynomial is the result of summing a number of monomials, and hence can be constructed as a sum of estimated monomial terms. The degree of a monomial is given by the sum of the powers of its constituent terms, with a polynomial having degree equal to the maximum degree of its constituent monomials. In our model, we assume a fixed

maximum degree $d \in \mathbb{N}$ for our polynomial approximation.

Once d is set, we construct all length N_x sequences of positive integers that sum to $n \leq d, n \in \mathbb{N}_0$, resulting in

$$M = \sum_{n=0}^{d} \binom{n+N_x-1}{N_x-1} \tag{5.9}$$

unique sequences. This simple procedure allows us to generate the powers of all monomial terms in a polynomial of degree d, that we store in an $N_x \times M$ matrix, denoted $\mathbf{D}$, with the term $D_{i,j}$ corresponding to the power of state dimension i in the j-th monomial term. The polynomial expression is then defined by matrix $\mathbf{C}$, following Eq. (5.8). The ordering of the monomials is arbitrary but must be consistent, as it implies the order of the columns of $\mathbf{C}$ and $\mathbf{D}$.

In our construction, d must be a non-zero natural number, as we construct polynomials from positive integer powers of the state components. However, the method could easily be extended to utilise rational powers of the state, of the form $1/p$, $p \in \mathbb{N}$. For example, one could construct an approximant utilising square root terms with maximal polynomial degree d simply by constructing polynomials up to degree $2d$, and then substituting in the square root term. More generally, approximants of maximal degree d using terms of power $1/p$ could be utilised by constructing polynomials up to degree pd, and then dividing $\mathbf{D}$ by p. We choose to focus here on the integer power polynomials for notational simplicity, ease of implementation, and the capability to be interpreted in a similar fashion to Taylor approximants.

Generating the degree matrix

We will now specify the construction of $\mathbf{D}$, the degree matrix. As before, $\mathbf{D}$ is such that $D_{i,j}$ is the power of state dimension i in the j-th monomial term. For example, if $N_x = 3$, and, for some $j \in \{1,...,M\}$, $\mathbf{D}_{:,j} = [0,1,2]$, then the value of the j-th monomial term when evaluating the transition of the i-th state dimension is $C_{i,j} x_1^0 x_2^1 x_3^2$, where $C_{i,j}$ is a learnt coefficient. The degree matrix $\mathbf{D}$ is static, and hence the same for every state, meaning that all states fit a polynomial of the same maximum degree.

Our method for construction the degree matrix $\mathbf{D}$ is given in Alg. 11, where 'count' and 'cwr' are defined in Sec. 5.2.4. Alg. 11 takes the union of all possible combinations with replacement of the set $\{1,2,...,N_x\}$ of length less than or equal to d, denoting by $\mathcal{Q}$ the resulting set. We then construct $\mathbf{D}$ by setting each entry $D_{i,j}$ equal to the number of times i occurs in the j-th element of $\mathcal{Q}$, for $i \in \{1,...,N_x\}$ and $j \in \{1,...,M\}$.

We observe that $|\mathcal{Q}| = M$. Note that the set $\mathcal{Q}$ has no inherent ordering, but we access it by index. We must therefore impose an ordering on the set $\mathcal{Q}$. One such

Algorithm 11 Generating the degree matrix $\mathbf{D}$

1: **Input:** State size N_x, maximal degree d.
2: **Output:** Matrix $\mathbf{D} \in \mathbb{R}^{N_x \times M}$ of monomial degrees.
3: Compute $\mathcal{Q} = \bigcup_{\delta=0}^{d} \text{cwr}([1,2,...,N_x],\delta)$
4: $D_{i,j} = \text{count}(i, \mathcal{Q}_j)\ \forall\ i \in \{1,...,N_x\},\ j \in \{1,...,M\}$.

ordering is lexicographical ordering. To apply this ordering, we first count how many times each number appears in an element of $\mathcal{Q}$. We then order these elements by the number of times 1 appears, and in case of equality comparing the number of times 2 appears, and so on until N_x. Note that the ordering of the degree matrix does not change the properties of the algorithm. In this chapter, for interpretability, we sort monomials in ascending order by degree, then sorting by lexicographical order as a tiebreaker within degree, as this aligns closely with how the degree matrix is generated in Alg. 11, by iterating over degrees.

Initialising the coefficient matrix and evaluating the polynomial

Now that $\mathbf{D}$ is constructed, to evaluate the resulting polynomial for each state dimension, we require $\mathbf{C}$, encoding the coefficients of each monomial term in every state dimension. These coefficients are our object of inference, and therefore should be stored in a manner admitting efficient evaluation of the polynomial.

We have M monomials for each of the N_x state dimensions, and therefore we propose to store the monomial coefficients in an $N_x \times M$ matrix $\mathbf{C}$, where $C_{i,j}$ corresponds to the coefficient of the j-th monomial when computing state i. This gives us a total of $N_x \cdot M = N_x \sum_{n=0}^{d} \binom{n+N_x-1}{N_x-1}$ parameters to estimate.

Following usual broadcasting rules, given $\mathbf{x}$, $\mathbf{D}$, and $\mathbf{C}$, we can now evaluate the value of our polynomial at any $\mathbf{x} \in \mathbb{R}^{N_x}$ by Eq. (5.8). Note that $\prod_{i=1}^{N_x} x_i^{D_{i,j}}$ is scalar, with $\mathbf{C}_{\cdot,j} \prod_{i=1}^{N_x} x_i^{D_{i,j}}$ being the vector $\mathbf{C}_{\cdot,j}$ multiplied by the scalar $\prod_{i=1}^{N_x} x_i^{D_{i,j}}$. In effect, $\prod_{i=1}^{N_x} x_i^{D_{i,j}}$ evaluates the j-th monomial term with coefficient 1, and this calculation is reused for every state dimension, with $\mathbf{C}_{\cdot,j} \prod_{i=1}^{N_x} x_i^{D_{i,j}}$ applying the coefficients, which are unique to each state dimension. Once the model is initialised, our goal is to learn the coefficient matrix $\mathbf{C}$, since this matrix, in conjunction with the known fixed degree matrix $\mathbf{D}$, defines the transition density $p(\mathbf{x}_t|\mathbf{x}_{t-1};\boldsymbol{\theta}) = p(\mathbf{x}_t|\mathbf{x}_{t-1};\mathbf{C})$ in Eq. (5.1), where $\boldsymbol{\theta}$ is the set of learnt parameters, in our case $\mathbf{C}$.

5.3.2 A graphical interpretation of matrices C and D

Within time series modelling, we can often interpret the driving parameters of a sparse model as a graph encoding the connectivity of the system [16, 35, 71, 89, 90, 34]. This

is the case here as well, where we observe that, for $(a,b) \in \{1,...,N_x\}^2$, state dimension b affects state dimension a in our estimated dynamics if $(\text{abs}(\mathbf{C})\mathbf{D}^\top)_{a,b} \neq 0$, where we note that $\text{abs}(\mathbf{C})\mathbf{D}^\top$ is an $N_x \times N_x$ matrix.

We can interpret this in terms of Granger causality, where we see that including information from state b improves the knowledge on state a, and therefore we say that state b Granger-causes state a if $(\text{abs}(\mathbf{C})\mathbf{D}^\top)_{a,b} \neq 0$.

We are therefore able to construct a directed graph encoding the network topology of our estimated system from the matrices $\mathbf{C}$ and $\mathbf{D}$. This graph has adjacency matrix $\mathbf{A}$, defined by $\mathbf{A} = \mathbb{1}_{\neq 0}(\text{abs}(\mathbf{C})\mathbf{D}^\top) \in \{0,1\}^{N_x \times N_x}$, where we have an edge from node b to node a if $A_{a,b} = 1$, and no edge if $A_{a,b} = 0$. If this graph is not fully connected, or equivalently if there exist $(a,b) \in \{1,...,N_x\}^2$ such that $A_{a,b} = A_{b,a} = 0$, then some state dimensions do not directly interact in our estimated system. Note that this graph represents the connectivity of the state space within the system dynamics, and therefore it is distinct from graphical SSMs that perform state estimation over graphs, such as [91]. However, the interpretation of the above graph as encoding relationships between state dimensions is the same as that in methods that estimate the state as a graph, or methods that infer network structure as a graph [92].

We can also interpret our system as a collection of M graphs, $\{\mathcal{G}_j\}_{j=1}^M$, where the j-th graph $\mathcal{G}_j$ has adjacency matrix $\text{abs}(\mathbf{C}_{:,j})\mathbf{D}_{:,j}^\top$. Each $\mathcal{G}_j$ can be interpreted as encoding the connectivity resulting from the j-th monomial. A sparsity promoting prior on $\mathbf{C}$ also promotes sparsity in these graphs. We can therefore interpret our method of estimating $\mathbf{C}$ as estimating multiple interacting sparse graphical models, one for each of the underlying monomials. We can then recover the overall connectivity graph by taking a union of the graphs encoding the connectivity of the individual monomials. These graphs can be constructed for more general forms of model, such as those discussed in Sec. 5.4.4, thereby generalising the sparse graphical interpretation to parametric non-linear model discovery methods.

Once estimated, these graphs can be utilised in a variety of ways. For example, it is possible that the system could be broken down into sparsely interacting sub-systems, which is a common structure for real-world systems [14], or, if possible, used to separate the system dynamics into non-interacting systems that can be filtered in parallel, assuming the observation model also allows them to be separated. The graph estimate can also be used to determine the most influential state dimensions, as the nodes relating to these will have a higher out degree than nodes relating to dimensions that affect fewer other dimensions. These examples are not exhaustive, and many potential uses of this graph interpretation are system specific, for example, estimating the network structure of a power grid if observations are related to such a system.

5.3.3 An example: Lorenz 63

We present here a worked example utilising the Lorenz 63 model. The Lorenz 63 model [80] is a popular chaotic oscillator model, with a discrete time variant given by,

$$\begin{aligned} x_{1,t} &= x_{1,t-1} + \Delta t(\sigma(x_{2,t-1} - x_{1,t-1})) + v_{1,t}, \\ x_{2,t} &= x_{2,t-1} + \Delta t(x_{1,t-1}(\rho - x_{3,t-1}) - x_{2,t-1}) + v_{2,t}, \\ x_{3,t} &= x_{3,t-1} + \Delta t(x_{1,t-1}x_{2,t-1} - \beta x_{3,t-1}) + v_{3,t}, \\ \mathbf{y}_t &= \mathbf{x}_t + \mathbf{r}_t, \end{aligned} \tag{5.10}$$

where Δt is the time elapsed between observations, $\mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_v)$ the state noise term, and $\mathbf{r}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_r)$ the observation noise term, and β, ρ, σ are real scalar parameters, often taken as $\beta = 8/3$, $\sigma = 10$, $\rho = 28$. The initial condition $\mathbf{x}_0$ is arbitrary, so long as it is non-zero, and is often taken to be [1,0,0].

We can see that the transition state system in Eq. (5.10) is described by a degree $d = 2$ polynomial in $N_x = 3$ variables. Therefore, using our notations, we have

$$Q = \{[1,1],[1,2],[1,3],[1],[2,2],[2,3],[2],[3,3],[3],[]\}$$

under lexicographical ordering, and

$$Q = \{[],[1],[2],[3],[1,1],[1,2],[1,3],[2,2],[2,3],[3,3]\}$$

under lexicographical-in-degree ordering. From the lexicographical-in-degree ordering, the resulting degree matrix $\mathbf{D}$ is

$$\mathbf{D} = \begin{pmatrix} 0 & 1 & 0 & 0 & 2 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 2 \end{pmatrix}.$$

Given the above degree matrix $\mathbf{D}$, we can extract from Eq. (5.10) the true coefficient matrix $\mathbf{C}$,

$$\mathbf{C} = \begin{pmatrix} 0 & 1-\sigma\Delta t & \sigma\Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \rho\Delta t & 1-\Delta t & 0 & 0 & 0 & -\Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 1-\beta\Delta t & 0 & \Delta t & 0 & 0 & 0 & 0 \end{pmatrix},$$

which, notably, is sparse with 23 elements out of 30 equal to zero. We can verify that inputting the above $\mathbf{C}$ and $\mathbf{D}$ into Eq. (5.7) and Eq. (5.8) yields the system given in Eq. (5.10). Furthermore, we construct the adjacency matrix $\mathbf{A} = \mathbb{1}_{\neq 0}(\text{abs}(\mathbf{C})\mathbf{D}^\top)$ from the above $\mathbf{C}$ and $\mathbf{D}$ matrices, yielding the adjacency matrix and associated directed

graph given in Fig. 5.1.

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix},$$

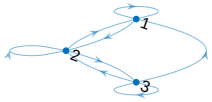


Figure 5.1: Graph and adjacency matrix encoding the connectivity of the Lorenz 63 system.

In this example, the coefficient matrix $\mathbf{C}$ is sparse, while the adjacency matrix $\mathbf{A}$ gives a highly connected graph. Imposing a sparse $\mathbf{C}$ at the estimation stage therefore gives more information than the adjacency matrix $\mathbf{A}$, as we also obtain estimates of the type of interactions that occur between the hidden states.

5.3.4 Parameter estimation

We now move to the learning procedure, for the unknown parameter $\boldsymbol{\theta} := \mathbf{C} \in \mathbb{R}^{N_x \times M}$. This is done via a MAP approach, that is by minimising the negative posterior log density ℓ_R, given by

$$\ell_R(\mathbf{C}|\mathbf{y}_{1:T}, \lambda) = \ell(\mathbf{C}|\mathbf{y}_{1:T}) + \lambda R(\mathbf{C}), \tag{5.11}$$

where

$$\ell(\mathbf{C}|\mathbf{y}_{1:T}) = -\log(p(\mathbf{y}_{1:T}|\mathbf{C})), \tag{5.12}$$

is the negative log-likelihood of $\mathbf{C}$, and $\lambda R(\mathbf{C})$ is a sparsity promoting penalty term acting on $\mathbf{C}$, with penalty weight $\lambda > 0$. Hence, we aim to compute

$$\widehat{\mathbf{C}} = \underset{\mathbf{C} \in \mathbb{R}^{N_x \times M}}{\operatorname{argmin}} \ \ell_R(\mathbf{C}|\mathbf{y}_{1:T}, \lambda). \tag{5.13}$$

We propose to adopt the $L1$ penalty to promote sparsity, given by

$$(\forall \mathbf{C} \in \mathbb{R}^{N_x \times M}) \quad R(\mathbf{C}) = ||\mathbf{C}||_1 := \sum_{j=1}^{M} \sum_{i=1}^{N_x} |C_{i,j}|. \tag{5.14}$$

We propose to solve Eq. (5.13), using a first-order optimisation scheme, combining a gradient step on the log-likelihood, and a proximity step over λR. We will now describe the gradient step, and then describe the proximity step.

Estimating the parameter likelihood and its gradient

In order to resolve Eq. (5.13) using a first order optimisation scheme, we require to evaluate the negative log-likelihood and its first derivative with respect to $\mathbf{C}$, given

the observation series and the SSM. We propose to estimate the likelihood through Eq. (5.4), where we have $\boldsymbol{\theta} := \mathbf{C}$, our parameter of interest. We then transform this quantity to the negative log-likelihood through negating the logarithm of the resulting estimate. For stability reasons, in practice, log-weights are used, and the log-likelihood is thus computed directly. The obtained estimate of the log-likelihood depends on the particle weights, which, in the standard SIR particle filter of Alg. 9, are subject to a non-differentiable resampling step.

Thankfully, we can utilise the DPF approach discussed in Sec. 5.2.3, to obtain an estimate of the negative log-likelihood with respect to our parameter $\mathbf{C}$, and the gradient of the negative log-likelihood with respect to $\mathbf{C}$. The DPF yields a stochastic estimate of the gradient, which we can use to perform gradient based minimisation of the negative log-likelihood. In order to be robust to gradient noise, we propose to rely on first-order updates from the deep learning literature, where stochastic gradients are common due to stochastic input batches. In our experiments, we utilise the Novograd optimiser [83, 93] to compute our gradient updates for the negative log-likelihood, as it is robust to gradient outliers [93].

We denote the result of the gradient update utilising the negative log-likelihood at iteration s,

$$\widetilde{\mathbf{C}}_s = \text{update}_\eta(\mathbf{C}_{s-1}, \nabla \ell(\mathbf{y}_{1:T} | \mathbf{C}_{s-1})), \tag{5.15}$$

where $\text{update}_\eta(\mathbf{A}, \nabla \ell(\mathbf{y}_{1:T}|\mathbf{B}))$ is a step of a given minimisation scheme with learning rate η applied to $\mathbf{A}$ with gradient of the negative log-likelihood obtained from running the particle filter with parameter $\mathbf{B}$ and observations $\mathbf{y}_{1:T}$.

Proximal update on the penalty term

Given that we can estimate the negative log-likelihood $\ell(\mathbf{y}_{1:T}|\mathbf{C})$ and its gradient $\nabla \ell(\mathbf{y}_{1:T}|\mathbf{C})$, we now need to account for the penalty term $\lambda R(\mathbf{C})$. Note that the $L1$ penalty is not differentiable when a coordinate is 0, hence it cannot be optimised using standard gradient descent if we aim to recover sparse estimates. It is however convex, and particularly well suited to the use of a proximity operator update [94], that can be understood as an implicit subgradient step.

The proximity operator update, for the $L1$ penalty, reads as a simple soft thresholding, so that our resulting scheme; combining both steps, is

$$\begin{aligned} \widetilde{\mathbf{C}}_s &= \text{update}_\eta(\mathbf{C}_{s-1}, \nabla \ell(\mathbf{y}_{1:T} | \mathbf{C}_{s-1})), \\ \mathbf{C}_s &= \mathcal{T}_{\eta \cdot \lambda}(\widetilde{\mathbf{C}}_s), \end{aligned} \tag{5.16}$$

where the soft-thresholding operator,

$$\mathcal{T}_\alpha(x) = \max(|x| - \alpha, 0) \cdot \text{sgn}(x),$$

is applied element-wise. The above algorithm belongs to the class of stochastic proximal gradient methods, the convergence of which is well studied, for instance in [95, 96].

Note that automatic differentiation can also be used when applying the $L1$ penalty, as the $L1$ penalty admits a subgradient [97, 98]. However, the proximal operator update is more computationally efficient, and more stable. The proximal operator is also more versatile, as it allows using other penalties such as low rank, or $L0$ penalty, which cannot be applied by the subgradient method [99].

5.3.5 S-GraphGrad algorithm

We have now all the elements for solving (5.13). We first start with a full batch implementation, in Alg. 12, which we call S-GraphGrad, and which operates on the entire series of observations at once.

Algorithm 12 Series GraphGrad algorithm (S-GraphGrad)

1: **Input:** Series of observations $\mathbf{y}_{1:T}$, number of steps S, penalty parameter λ, $\mathbb{N}^{N_x \times M}$ degree matrix $\mathbf{D}$, initial coefficient value $\mathbf{C}_0$, learning rate η.
2: **Output:** Sparse $\mathbb{R}^{N_x \times M}$ matrix $\mathbf{C}$ of polynomial coefficients.
3: **for** $s = 1,\ldots,S$ **do**
4: Run Alg. 10 with $p(\mathbf{x}_t|\mathbf{x}_{t-1};\mathbf{C}_{s-1},\mathbf{D})$ and observations $\mathbf{y}_{1:T}$.
5: Estimate $\ell(\mathbf{y}_{1:T}|\mathbf{C}_{s-1})$ and $\nabla\ell(\mathbf{y}_{1:T}|\mathbf{C}_{s-1})$ via Eq. (5.4) and backpropagation.
6: Set $\widetilde{\mathbf{C}}_s = \text{update}_\eta(\mathbf{C}_{s-1}, \nabla\ell(\mathbf{y}_{1:T}|\mathbf{C}_{s-1}))$.
7: Set $\mathbf{C}_s = \mathcal{T}_{\eta\cdot\lambda}(\widetilde{\mathbf{C}}_s)$.
8: **end for**
9: Output $\mathbf{C} = \mathbf{C}_S$.

S-GraphGrad description

S-GraphGrad takes as input the series of observations $\mathbf{y}_{1:T}$, the number of steps S, the penalty parameter λ, the $\mathbb{N}_0^{N_x \times M}$ degree matrix $\mathbf{D}$, the initial coefficient value $\mathbf{C}_0$, and the learning rate η, producing as output a sparse $\mathbb{R}^{N_x \times M}$ matrix $\mathbf{C}$ of polynomial coefficients. S-GraphGrad iterates for S steps, with the s-th step proceeding as follows.

First, we run a differentiable particle filter with the estimate of the coefficients from the previous step $\mathbf{C}_{s-1}$ with observations $\mathbf{y}_{1:T}$ (line 4). When running the filter, we assume that we either know or have suitable estimates of the observation model $p(\mathbf{y}_t|\mathbf{x}_t)$ and the proposal distribution $\pi(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{y}_t)$, as well as the state noise. For example, if the noise is additive and Gaussian, we have Eq. (5.7), and would require

a suitable estimate of $\mathbf{\Sigma}_v$. Other noises can be accounted for, such as multiplicative Gaussian, assuming a definition of how the estimated polynomial model and the noise interact to compute $p(\mathbf{x}_t|\mathbf{x}_{t-1};\mathbf{C},\mathbf{D})$.

While running the filter, we process the weights to obtain an estimate of the likelihood of the parameter $\mathbf{C}_{s-1}$ using Eq. (5.4), and use automatic differentiation to obtain an estimate of $\nabla\ell(\mathbf{y}_{1:T}|\mathbf{C}^{s-1})$ (line 5). We then apply a gradient-based update step, on the negative log-likelihood, of a given minimisation scheme update with learning rate η to $\mathbf{C}_{s-1}$, yielding $\widetilde{\mathbf{C}}_s$. Namely, the gradients $\nabla\ell(\mathbf{y}_{1:T}|\mathbf{C}_{s-1})$ are those of the negative log-likelihood, as we are aiming to maximise the log posterior density (equivalent to the penalised log-likelihood), and hence we minimise the negative log-likelihood using the minimisation scheme update (line 6).

We then apply the proximal update at $\widetilde{\mathbf{C}}_s$, yielding $\mathbf{C}_s$, the estimated coefficient matrix at step s (line 7). In the case of the $L1$ penalty the proximity operator is the soft thresholding operator, depending on the learning rate of η and a penalty parameter of λ.

Combating likelihood degeneracy

In practice, using S-GraphGrad is hampered by likelihood degeneracy, which causes the likelihood estimate of Eq. (5.4) to evaluate numerically as zero due to the limited precision of floating point arithmetic. This could result in the gradient vanishing, and hence numerical failure of the scheme.

A typical way to combat this is by using log weights, different floating point representations, or gradient scaling, but these do not address the core problem, which is that the likelihood becomes more concentrated the longer the observations series is. We will now discuss this phenomenon, and how we mitigate it.

For observation series longer than a few elements, the likelihood function, given in Eq. (5.4), when evaluated with parameters that do not yield dynamics close to the true dynamics, is numerically 0. Numerical zeros are a limitation of computer arithmetic, and in this case result from the multiplication of the very low likelihoods that occur in an unadapted filter to yield a result that is numerically zero. Note that, if operating on a log scale, we instead obtain $-\infty$ rather than 0, but the root cause is the same. Note that even when the dynamics are perfectly known we still observe this phenomenon, and here we aim to mitigate the effect of unadapted parameters, not the fundamental issue of likelihood accumulation in particle filters.

There are several standard ways to mitigate the effects of likelihood concentration within SSMs, such as variance inflation, where we increase the magnitude of the state and observation covariances to decrease likelihood concentration, or simply running the particle filter with a larger number of particles. However, both of these methods

break down as the length of the observation series increases, as they do not address the underlying issue of the parameters not being adapted to the observed data.

The likelihood degenerating causes the gradients of the log-likelihood to explode, and hence makes it impossible to perform gradient-based inference. We can address this issue by running our method multiple times, first to fit a coarse estimate, and then refining this estimate over several subsequent iterations. We fit this coarse estimate using only the first few observations, and then gradually introduce more observations, refining our estimate and mitigating likelihood degeneracy due to unadapted parameters.

5.3.6 B-GraphGrad algorithm

We end this section by presenting B-GraphGrad, our final method for estimating the dynamics of a general non-linear SSM via a polynomial approximation. This method builds upon that described Alg. 12 (S-GraphGrad) in Sec. 5.3.5, proposing a sequentially-batched implementation of the method, utilising an observation batching strategy to mitigate likelihood concentration issues.

The proposed method is doubly iterative: we iterate over telescoping batches of observations, within which we iteratively estimate the coefficients of our polynomial approximation. We create B batches of observations by $\mathbf{y}^{(b)} := \mathbf{y}_{1:\lceil bT/B \rceil}$ for $b = 1,...,B$. Note that these batches are of increasing size, with $\mathbf{y}^{(b)} \subseteq \mathbf{y}^{(b+1)}$ for $b = 1, \ldots, B$. Within each batch of observations, we perform S runs of Alg. 12 (S-GraphGrad). We perform batching to avoid numerical errors, as the first sampled trajectories, with parameters close the the initial random initialisation, will likely have an extremely small log-likelihood, which compounds numerical errors when computing the weights in Alg. 10 [31]. Note that, in the case that the true system can represented as a polynomial, few batches may be needed. Indeed, in the case of the Lorenz 63 oscillator, we require only a batch of 10 observations to initialise the coefficients, and can then proceed with estimation on series of lengths exceeding 1000. However, in general the true system is not polynomial, so we proceed with fixed-size batches for simplicity and robustness. We present the method in Alg. 13 (B-GraphGrad), and describe it below.

Algorithm 13 Batched GraphGrad algorithm (B-GraphGrad)

Input: Series of observations $\mathbf{y}_{1:T}$, number of batches B, steps per batch S, penalty parameter λ, learning rate η, maximal degree d, hidden state size N_x.
Output: Sparse $\mathbb{R}^{N_x \times M}$ matrix $\mathbf{C}$ of polynomial coefficients.
Construct $\mathbf{D}$ by running Alg. 11.
Randomly initialise $\mathbf{C}_0 \in \mathbb{R}^{N_x \times M}$ element-wise by sampling a $\mathcal{U}(-1,1)$ distribution.
for $b=1,...,B$ **do**
    Set $\mathbf{y}^{(b)} := \mathbf{y}_{1:\lceil bT/B \rceil}$.
    Run Alg. 12 (S-GraphGrad) with observations $\mathbf{y}^{(b)}$, number of steps S, penalty parameter λ, degree matrix $\mathbf{D}$, initial coefficient value $\mathbf{C}_{b-1}$, and learning rate η, setting $\mathbf{C}_b$ to the output.
end for
Output $\mathbf{C} := \mathbf{C}_B$

B-GraphGrad description

As stated in Sec. 5.3.6, B-GraphGrad builds upon S-GraphGrad, implementing observation batching to mitigate likelihood generation for unadapted models. In particular, B-GraphGrad runs for B iterations, with the b-th iteration running S-GraphGrad with S iterations on the observation series $\mathbf{y}^{(b)} := \mathbf{y}_{1:\lceil bT/B \rceil} \subseteq \mathbf{y}_{1:T}$. This allows us to 'warm up' the coefficient parameter $\mathbf{C}$, so that when we are performing estimation on the entire series we do not encounter issues due to likelihood concentration. We avoid these issues by learning $\mathbf{C}$ first on small series, which have relatively dispersed likelihoods due to their length. By learning an initial estimate of the transition dynamics on the initial batches, we have a better model when we come to estimating $\mathbf{C}$ on the longer series that can display likelihood issues when the model is not adapted.

B-GraphGrad proceeds as follows. First, we construct the degree matrix $\mathbf{D}$ given the selected maximum degree d and hidden state dimension N_x. $\mathbf{D}$ is constructed following Alg. 11 described in Sec. 5.3.1. We note that $\mathbf{D}$ is a static parameter, and is not learnt through our training. We then generate an initial value for $\mathbf{C}$, denoted $\mathbf{C}_0$. We can choose this value randomly, as we avoid likelihood related issues through our batching procedure. In Alg. 13 (B-GraphGrad) we draw $\mathbf{C}_0$ element-wise by sampling a uniform $\mathcal{U}(-1,1)$ distribution, however in principle any real valued distribution could be used, such as a standard normal distribution. After initialising $\mathbf{C}$, we generate the B observation batches. We then iterate over the B batches of observations. For batch b, we run Alg. 12 (S-GraphGrad) with the parameter estimate from the previous batch, $\mathbf{C}_{b-1}$, and label the resulting updated parameter by $\mathbf{C}_b$. The B-th batch is the final batch and trains using the entire series of observations, outputting the final value $\mathbf{C}_B$, which is learnt so as to optimise our objective function Eq. (5.13).

5.4 Discussion

We now discuss our algorithm, B-GraphGrad, as given in Alg. 13, and provide some potential extensions and modifications to particular systems.

5.4.1 GraphGrad prerequisites

In order to apply GraphGrad, we must either know or have estimates of the observation model and its noise process, and must know the form of the state noise. The observation model are typically assumed known in dynamical systems, such as by the intrinsic properties of the sensors used, or are estimated from previous studies. We require that the likelihood of the observation is differentiable with respect to $\mathbf{C}$, as otherwise we cannot apply the differentiable particle filter. If we use a proposal distribution $\pi(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{y}_t) \neq p(\mathbf{x}_t|\mathbf{x}_{t-1}) = p(\mathbf{x}_t|\mathbf{x}_{t-1};\mathbf{C},\mathbf{D})$, then we must be able to sample from this distribution differentiably with respect with $\mathbf{C}$, and it must admit a differentiable likelihood with respect with $\mathbf{C}$.

Furthermore, we assume that the state noise is such that we can sample it in a differentiable manner. An example of such a noise is the Gaussian distribution, from which we can generate sample differentiably with regard to the distribution parameters using the reparametrisation trick [100]. Many distributions can be sampled differentiably with regard to their parameters using similar tricks, such as the multivariate t distribution (with known degrees of freedom), which could be used if heavier tails are required.

5.4.2 Computational cost

B-GraphGrad, as given in Alg. 13, requires SB evaluations of the particle filter to obtain the negative log-likelihood and its gradient. The particle filter is of time complexity $\mathcal{O}(KT)$, and therefore our method is of complexity $\mathcal{O}(SBKT)$. Indeed, we evaluate the particle filter SB times, and obtain the gradients via reverse-mode automatic differentiation, which is of the same time complexity as the function we differentiate. Note that we neglect here the complexity cost of sampling distributions and evaluating the likelihoods in the particle filter, as these vary from model to model, and occur the same number of times across filter runs.

The cost of evaluating the polynomial in Eq. (5.8) is small, and of complexity $\mathcal{O}(MN_x^2)$. We note that the polynomial evaluation can be efficiently parallelised as $\mathbf{D}$ is fixed, so it is possible to evaluate the polynomial by performing an associative scan over evaluating the monomials. That is, we can evaluate the $x_j^{D_{i,j}}$ terms in parallel, and then evaluate their product and sum in parallel. Furthermore, the vector-scalar

element product is typically automatically abstracted to an SIMD operation, speeding evaluation up by a factor of N_x.

Finally, we note that the computation of the particle filter and its gradient can be accelerated by observing that the computations in the differentiable particle filter depend on the previous state only through the particles and the weights. Under this restriction, we can implement the particle filter as a scan-with-carry. Therefore, it is possible to construct the computational graph of the particle filter, and therefore of its gradient, from the computational graph of the scanned function, yielding a much smaller graph than from the entire filter [97].

5.4.3 Exploiting parallelisation to decrease runtime

B-GraphGrad, if implemented following Alg. 13, is a sequential algorithm. Sequential operation is required as the estimate at each step depends on the estimate at the previous step. However, we can use parallel computing to reduce the elapsed time of the computation by computing fewer batches.

In particular, the batching proposed in B-GraphGrad is very conservative, in that the batch size increases by the same increment in all instances. In practice, as the approximand uses few parameters, with each parameter having a distinct effect, we rapidly adapt to the system within the first few batches.

We hence propose a more efficient procedure. We initialise P independent coefficient matrices, $\{\mathbf{C}^{(p)}\}_{p=1}^{P}$, and learn each independently in parallel for B_I batches of observations. Then, we check if there exists a subset of the P coefficient matrices such that the coefficient matrices are close in value, for example by attempting to find $\mathbf{C} \in \mathbb{R}^{N_x \times M}$ such that $||\mathbf{C} - \mathbf{C}^{(p)}|| < \epsilon \; \forall p \in \{1,...,P\}$ for some matrix norm $||\cdot||$ and $\epsilon > 0$. If this matrix exists, then this indicates that the coefficient matrices have adapted to the system. We then cease batching, and learn on the entire series of observations using the coefficient matrices in our subset, aggregating after finishing optimisation, e.g., via an element-wise mean. If the subset cannot be constructed, we continue learning for another batch of observations, and then recheck the above condition, stopping when either we exhaust the set of observations or the subset of matrices can be constructed.

This batching method can take advantage of parallel processing by performing optimisation on each independent parameter in parallel, with the subset construction and matrix calculations being cheap in comparison. This accelerates our inference by reducing the number of batches that need to be constructed, therefore reducing the run time of the algorithm.

5.4.4 Interpretation as a library regression method

Our method, at its core, fits a series of polynomial functions aiming to recover the transition kernel of a SSM. Therefore, B-GraphGrad can be interpreted as performing function library regression, similar to SINdY [77], with the function library comprising all polynomials in N_x variables of degree less than or equal to d, and fitted performed simultaneously for each state dimension.

Given this interpretation, it is possible to expand the library of terms to include additional terms, such as trigonometric or exponential terms. These terms would allow for an even larger class of systems to be exactly represented by our model, but come at the downside of requiring additional machinery to evaluate, whereas polynomial terms admit a simple vectorised expression in Eq. (5.8). However, non-polynomial terms should be evaluated as separate expressions, and then combined with the polynomial terms after computation. For example, if the system is defined in terms of angles, trigonometric terms could be included, or functions of the differences between states could be included for systems involving potentials.

5.5 Numerical study

We here present our experimental results, to illustrate and discuss the performance of our proposed approach. We are interested in the recovery of the underlying model in dynamical systems described by polynomial ordinary differential equations (ODEs), namely the Lorenz 63 (Sec. 5.5.2) and Lorenz 96 (Sec. 5.5.3) oscillators, and estimating a non-polynomial system, the Kuramoto oscillator (Sec. 5.5.4). In each case, we use an Euler discretisation to build a discretised form for the model, and we map it with our problem formulation where the goal is to recover an estimate of a ground truth matrix $\mathbf{C}$.

We implement the proposed B-GraphGrad algorithm, and compare it against the fitting of a polynomial of the same degree using a maximum likelihood scheme, which we denote pMLE, for polynomial maximum likelihood estimator. This scheme is identical to our B-GraphGrad scheme, except that we remove the proximal steps. pMLE fits a fully dense model, so in its case we present only RMSE, as the sparsity metrics are predetermined (i.e., no sparsity is recovered).

5.5.1 Experimental setup

For our numerical experiments, we use the following settings, unless stated otherwise. We use the Novograd optimisation scheme [83, 93], for update in Alg. 12 line 6, with a fixed learning rate of $\eta = 10^{-3}$. We split our observation series into B batches such

that $B = \lceil T/10 \rceil$, therefore giving a batch size increment of approximately 10. We use $K = 100$ particles in our particle filter in Alg. 10. T denotes the length of the observation series. For a given polynomial degree d, we construct the degree matrix $\mathbf{D}$ following Sec. 5.3.1.

Performance assessment is performed using several quantitative metrics, either based on the recovery of the sparse support of $\mathbf{C}$ (in terms of specificity, precision, recall, and F1 score), or of its entries (in terms of root mean square error, RMSE). We define an element of $\mathbf{C}$ as numerically zero if it is lower than 10^{-6} in absolute value, as this is approximately the precision of single precision floating point arithmetic. Perfect recovery of positive and negative values is indicated by 1.0 in all sparsity metrics, and 0.0 RMSE.

We choose the penalty parameter λ for a system with state dimension N_x and maximal degree d by tuning it on a synthetic system. This system is not used to generate the data for fitting, and is only used to tune $\boldsymbol{\lambda}$. This system is such that it has the same N_x dimension state and a maximal degree of d as the model we are fitting. We generate the $\mathbf{C}$ matrix of this system such that it is 75% sparse, with the dense elements drawn from $\mathrm{U}(-N_x, N_x)$, and then scaled such that the maximal singular value of $\mathbf{C}$ is 1. We discretise this system using an Euler discretisation with a timestep Δt equal to the timestep of the system we aim to estimate, adding zero mean Gaussian noise terms $\mathbf{v}$ and $\mathbf{r}$ to the state and observations respectively, with $\boldsymbol{\Sigma}_v = \boldsymbol{\Sigma}_r = \Delta t \mathbf{Id}_{N_x}$. We then optimise λ via setting $\lambda = 10^l$, with l chosen to maximise the accuracy of the estimated $\mathbf{C}$ of this system using B-GraphGrad, via 10 iterations of bisection over the interval $[-5, 2]$.

5.5.2 Lorenz 63 model

We start our experiments, with the Lorenz 63 system [80], which we transform into an NLSSM using an Euler discretisation with a timestep of $\Delta t = 0.025$, yielding the system

$$\begin{aligned} x_{1,t+1} &= x_{1,t} + \Delta t(\sigma(x_{2,t} - x_{1,t})) + \sqrt{\Delta t} v_{1,t+1}, \\ x_{2,t+1} &= x_{2,t} + \Delta t(x_{1,t}(\rho - x_{3,t}) - x_{2,t}) + \sqrt{\Delta t} v_{2,t+1}, \\ x_{3,t+1} &= x_{3,t} + \Delta t(x_{1,t} x_{2,t} - \beta x_{3,t}) + \sqrt{\Delta t} v_{3,t+1}, \end{aligned} \tag{5.17}$$

with the observation model $p(\mathbf{y}_t | \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_t, \boldsymbol{\Sigma}_r)$. Hence, $N_x = N_y = 3$. We choose $\mathbf{x}_0$ such that $\mathbf{x}_0$ is equal to 1 in the first element, and 0 elsewhere. We set $\boldsymbol{\Sigma}_v = \boldsymbol{\Sigma}_r = \sigma^2 \mathbf{Id}_3$, with $\sigma = 1$ unless stated otherwise. Note that we present the true $\mathbf{C}$ and $\mathbf{D}$ matrices for this system in Sec. 5.3.3 We use $\rho = 28, \sigma = 10, \beta = 8/3$, as these parameters are known to result in a chaotic system, and we set $t_0 = 0$. Our particle filter is initialised with $p(\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_0, \mathbf{Id}_3)$.

We average the results, for our method GraphGrad, and pMLE, a scheme that

fits the same coefficient matrix $\mathbf{C}$ but does not promote sparsity, on 150 independent realisations of the specified dynamical system. Hereafter we present our results, on three scenarios, namely assuming maximum degree $d=2$ or 3, and varying the series length, then considering a varying noise amplitude.

Varying series length, maximum degree $d=2$

Table 5.1
Lorenz 63: Average recovery metrics for variable series length for 150 independent systems. Maximum polynomial degree $d=2$.

method	T	RMSE (10^{-3})	spec.	recall	prec.	F1
B-GraphGrad	25	1.6	0.90	0.92	0.96	0.94
pMLE	25	2.4	-	-	-	-
B-GraphGrad	50	1.0	0.98	0.97	0.98	0.98
pMLE	50	1.6	-	-	-	-
B-GraphGrad	100	0.4	1.00	1.00	1.00	1.00
pMLE	100	0.8	-	-	-	-
B-GraphGrad	200	0.2	1.00	1.00	1.00	1.00
pMLE	200	0.3	-	-	-	-

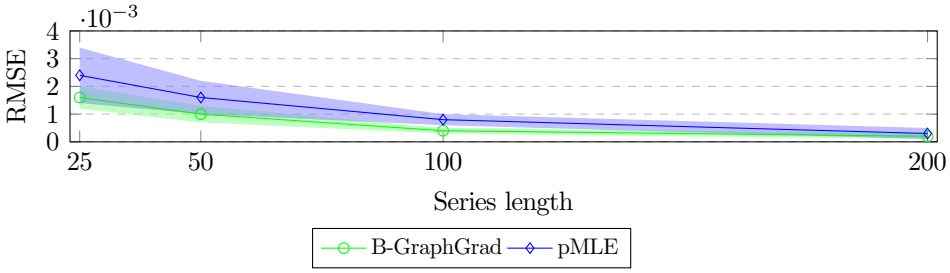


Figure 5.2: Comparison of B-GraphGrad with pMLE over variable series length on the Lorenz 63 oscillator, with maximum polynomial degree $d=2$. Markers denote mean performance, with the ribbons being symmetric 95% intervals.

Table 5.2
Lorenz 63: Median runtime relative to $T=25$ for variable series length for 150 independent systems. Maximum polynomial degree $d=2$.

method	T	Relative runtime
B-GraphGrad	25	1
B-GraphGrad	50	3.86
B-GraphGrad	100	15.64
B-GraphGrad	200	62.25

Table 5.1 presents the results for learning over a range of series lengths T with a maximal degree of $d=2$. In this case, estimating $\mathbf{C}$ requires learning 30 parameters. Here, we assumed the knowledge of the correct degree of the underlying system, so

these results serve to provide a baseline for the performance of our method in the scenario where the degree is known. We note that, for many dynamical systems, a default degree of 2 is a sensible modelling choice, as this type of interaction in the rate is very common in many systems, such as in chemical and biological networks, population modelling, and meteorological systems. We observe that the performance of our method improves as the number of observations T increases, indicating that our method well incorporates information from new observations. Furthermore, we see that, even for a small number of observations, our method recovers the system connectivity well, even if the RMSE is not as good for low values of T. This is due to changes in the system connectivity and the presence or absence of system terms having a large effect, and thus being relatively easy to infer compared the the specific value of the terms, the effects of which are obscured by the noise inherent to the system. Overall, we observe excellent performance, with good recovery of all parameters at all tested series lengths. We see that pMLE does not perform well, mostly because it recovers a fully dense system, and therefore yields matrix $\mathbf{C}$ with many non-zero parameters that are, in truth, zero. Hence, the RMSE is poor, as the value of the truly non-zero parameters is affected by the value of the falsely non-zero parameters.

From the results given in Table 5.2, we see that our runtime seems to scale quadratically in T. This follows from the linear complexity of the particle filter in T, and the number of batches B also scaling linearly in T as per the experimental setup. Each batch runs a particle filter of a fixed length, with each batch running a longer filter than the previous, and therefore the algorithm runtime scales quadratically with T. However, one can use the batching method presented in the introduction of Sec. 5.3.6 to remove the quadratic scaling by fixing the number of batches, in which case the runtime scales linearly in T.

Varying series length and maximum degree $d=3$

Table 5.3
Lorenz 63 average recovery metrics for
variable series length for 150 independent systems. Maximum allowed degree $d=3$.

method	T	RMSE (10^{-3})	spec.	recall	prec.	F1
B-GraphGrad	25	2.0	0.84	0.90	0.92	0.91
pMLE	25	2.8	-	-	-	-
B-GraphGrad	50	1.5	0.96	0.95	0.97	0.96
pMLE	50	2.0	-	-	-	-
B-GraphGrad	100	0.7	0.97	0.98	0.97	0.97
pMLE	100	1.3	-	-	-	-
B-GraphGrad	200	0.4	1.00	1.00	1.00	1.00
pMLE	200	1.0	-	-	-	-

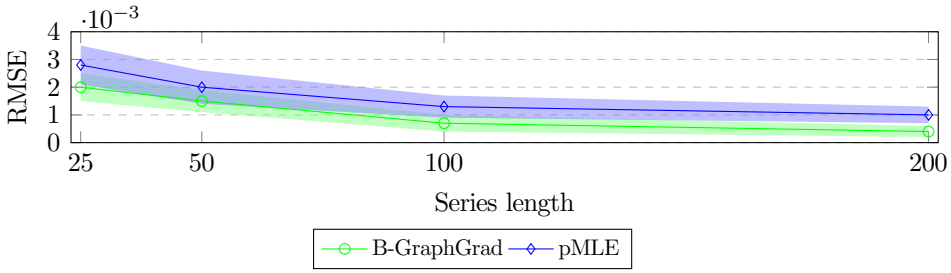


Figure 5.3: Comparison of B-GraphGrad with pMLE over variable series length on the Lorenz 63 oscillator, with maximum polynomial degree $d=3$. Markers denote mean performance, with the ribbons being symmetric 95% intervals.

Table 5.3 presents the results of our GraphGrad method over a range of series lengths T with maximal degree of 3. In this case estimating $\mathbf{C}$ requires estimating 60 parameters. A degree of 3 is greater than the true degree of the system, which, as we recall, is not in general known beforehand. Systems with a degree of 3 are uncommon, however it is possible to approximate non-polynomial systems, with higher degree allowing better approximation of the dynamics. For example, the Kuramoto oscillator [101] can be well approximated via a centralised Taylor approximation, which our method can learn directly.

We observe that the performance of our method improves as the number of observations increases, indicating that our method well incorporates information from new observations, and does not over fit given the over specification of the model relative to the system we are learning. We note a decrease in performance relative to Table 5.1 where the degree is that of the underlying system, however the results improve as T increases.

Furthermore, we note that the change in RMSE is more severe than the changes in sparsity metrics, as a small number of degree 3 terms are fit, where in reality they should be zero. As RMSE is computed only on terms recovered as non-zero, these contribution of these deviations is large relative to that of the degree 2 terms. We see that our method performs well, with the sparsity metrics being close to those of the $d=2$ system. The RMSE is inferior to the $d=2$ system, but still significantly outperforms the comparable polynomial MLE.

Variable noise magnitude

In Table 5.4, we present the results of our method over a range of noise magnitudes σ^2, now fixing the number of observations to $T=50$. We remind that the results from Tables 5.1 and 5.3, were obtained using $\sigma^2=1$. We observe that our method performs well for all tested values of σ^2, especially in sparsity metrics. As expected, the recovery quality degrades as the signal to noise ratio decreases when we increase σ^2. We note that a large σ^2 affects both the system and the observation, so increasing it has a particularly pronounced effect.

Table 5.4
Lorenz 63: Average recovery metrics for variable noise magnitude for 150 independent systems. Maximum polynomial degree $d=2$. $T=50$.

method	σ^2	RMSE (10^{-3})	spec.	recall	prec.	F1
B-GraphGrad	0.01	0.09	1.00	1.00	1.00	1.00
pMLE	0.01	0.3	-	-	-	-
B-GraphGrad	0.1	0.3	1.00	1.00	1.00	1.00
pMLE	0.1	0.6	-	-	-	-
B-GraphGrad	1	1.0	0.98	0.97	0.98	0.98
pMLE	1	1.6	-	-	-	-
B-GraphGrad	5	1.8	0.90	0.85	0.87	0.86
pMLE	5	2.3	-	-	-	-

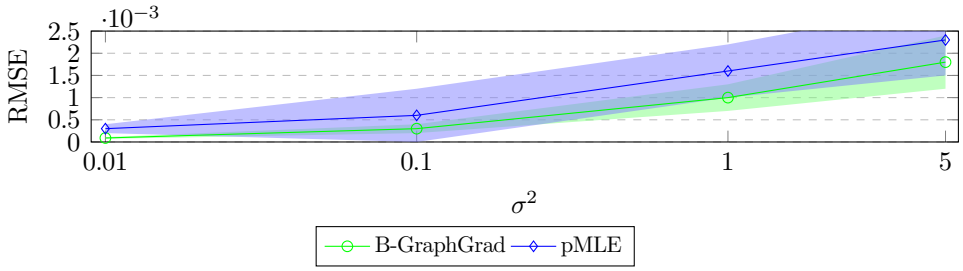


Figure 5.4: Comparison of B-GraphGrad with pMLE over variable noise magnitude on the Lorenz 63 oscillator, with maximum polynomial degree $d=2$. Markers denote mean performance, with the ribbons being symmetric 95% intervals.

Likelihood degeneracy

In Sec. 5.3.5, we discussed the effect of likelihood concentration on parameter estimation in the particle filter. We demonstrate this phenomenon below, giving the mean number of timesteps before the likelihood becomes numerically zero for a stochastic variant of the Lorenz 63 model, given by Eq. (5.10).

Table 5.5
Average number of timesteps/observations ($\Delta t=0.025$) before likelihood degeneracy with a random $\mathbf{C}$ matrix for the Lorenz 63 model ($d=2$) over 200 independent systems initialised at random points.

number of particles K	5	10	20	50	100	250	500	1000
timesteps before degeneracy	7.8	12.2	15.7	23.8	35.3	57.5	76.8	111.3

It is clear from Table 5.5 that increasing the number of particles does combat the likelihood degeneracy, though it does not solve it, with computational cost increasing significantly for small gains. It is for this reason that we do not display the results of S-GraphGrad, instead displaying only B-GraphGrad, as S-GraphGrad fails to converge using single precision arithmetic due to these likelihood issues.

Varying maximal degree d

The required number of parameters to estimate increases as d increases, and therefore the computational cost increases as well. In the Lorenz 63 oscillator we have $N_x = 3$, and therefore for $d = 2$, the true degree of the system, we have to estimate 30 parameters; 10 parameters per state dimension. For $d = 3$ this increases to 60 parameters, which is 20 parameters per state dimension. We present in Table 5.6 the cost of running B-GraphGrad with different d parameters relative to the cost of running with $d = 2$. All parameters are set per Sec. 5.5.2, except d, which we vary. We present the results in Table 5.6.

Table 5.6
Lorenz 63: Average relative runtime of B-GraphGrad when run with different values of d. Runtime is presented relative to $d = 2$.

maximum polynomial degree d	1	2	3	4	5	6
average relative runtime	0.41	1.0	1.98	3.46	5.52	8.27
relative number of parameters	0.4	1.0	2.0	3.5	5.6	8.4

From Table 5.6, we observe that the runtime of our method scales as the number of rows in $\mathbf{C}$, or equivalently, the runtime grows as the number of elements of $\mathbf{C}$ increase. For example, there are 84 monomials in 3 variables of of degree $d \leq 6$, and we observe that running our method with $d = 6$ takes 8.27 times longer than with $d = 2$ (which has 10 monomials). The runtime foe larger values of d is slightly lower than expected, due to constant computational overhead introduced in other areas of the implementation. This overhead makes up proportionally more of the runtime for smaller values of d, therefore resulting in smaller relative runtimes for larger values of d. Finally, we note that this experiment was performed without parallelising the evaluation of Eq. (5.8) or its gradients. When parallelisation is enabled, the relative runtime is limited by computational resources, and in the presence of unlimited resources the runtimes are approximately equal. However, the total number of operations performed will scale similarly to as before.

Prox update compared to subgradient update

B-GraphGrad uses the proximal operator of the $L1$ norm to apply a sparsity promoting penalty, following Sec. 5.3.4. However, as noted, we can also use subgradient methods to minimise our cost function including the $L1$ penalty. Using standard convex analysis definition, a subgradient of the $L1$ norm at an element x (i.e., an element of the subdifferential of L1 function, at x), is a vector of same dimension than x, with entries equal to $\text{sgn}(x)$, using the convention $\text{sgn}(0) = 0$. Note that, numerically, we never encountered exactly zero entries when running the subgradient descent method, and

that the subgradient of the $L1$ norm at 0^- (resp. 0^+) entry is taken as -1 (resp. $+1$). We use the same learning rate of $\eta=10^{-3}$ as in the proximal version, and the same (B,S) parameters for the number of inner/outer iterations. The computational cost differences within each step are negligible, however the proximal method is overall more efficient, as the proximal version appears to converge faster.

We test both the subgradient approach and the proximal approach on the same system as Sec. 5.5.2. We present both the average absolute value of elements where the ground truth is 0, and the RMSE of the estimate. Parameters are set per Sec. 5.5.2.

Table 5.7
Lorenz 63: Average absolute value of elements of $\mathbf{C}$ where ground truth is 0 over 150 independent systems. Maximum polynomial degree $d=2$.

method	T	Recall	RMSE (10^{-3})
B-GraphGrad (prox)	25	0.92	1.6
B-GraphGrad (subgrad)	25	0.76	2.0
B-GraphGrad (prox)	50	0.97	1.0
B-GraphGrad (subgrad)	50	0.80	1.4
B-GraphGrad (prox)	100	1.00	0.4
B-GraphGrad (subgrad)	100	0.91	0.6
B-GraphGrad (prox)	200	1.00	0.2
B-GraphGrad (subgrad)	200	0.93	0.4

The proximal operator method is more efficient per iteration, in particular, we obtain estimates closer to the ground truth in the same number of iterations as the subgradient method. Further, as the methods have negligible difference in computational cost, this illustrates the superiority of the proximal over the subgradient method for this problem.

5.5.3 Lorenz 96 model

The Lorenz 63 system, while well studied and challenging to estimate, is only 3 dimensional. In order to test the applicability of our method to higher dimensional chaotic systems, we test on the Lorenz 96 system [81], which we transform into a NLSSM using an Euler discretisation, yielding the system

$$\begin{aligned} d_{i,t+1} &= x_{i-1,t}(x_{i+1,t}-x_{i-2,t})-x_{i,t}+F, \\ x_{i,t+1} &= x_{i,t}+\Delta t\cdot d_{i,t+1}+\sqrt{\Delta t}\cdot v_{i,t+1}, \\ y_{i,t+1} &= x_{i,t+1}+\sqrt{\Delta t}\cdot r_{i,t+1}, \end{aligned} \tag{5.18}$$

for $i=\{1,\ldots,N_x\}$, with $\mathbf{v}_t\sim\mathcal{N}(\mathbf{0},\mathbf{\Sigma}_v)$,$\mathbf{r}_t\sim\mathcal{N}(\mathbf{0},\mathbf{\Sigma}_r)$, where $x_{,\cdot}=x_{N_x}$, $x_{-1,\cdot}=x_{N_x-1}$, and $x_{-2,\cdot}=x_{N_x-2}$. For this experiment we choose $N_x=N_y=20$. Therefore, in the case of $d=2$, estimating $\mathbf{C}$ requires estimating 4620 parameters, and the case of $d=3$

estimating $\mathbf{C}$ requires estimating 35420 parameters. We choose a forcing constant of $F=8$, which is known known to result in a chaotic system. The connectivity graph of this system, constructed following Sec. 5.3.2, is given in Figure 5.5, where we observe that state i is affected by states $i-2$,$i-1$,i, and $i+1$, with index boundaries such that state $N_x \cdot k+i$ is equivalent to state i for any $k \in \mathbb{Z}$.

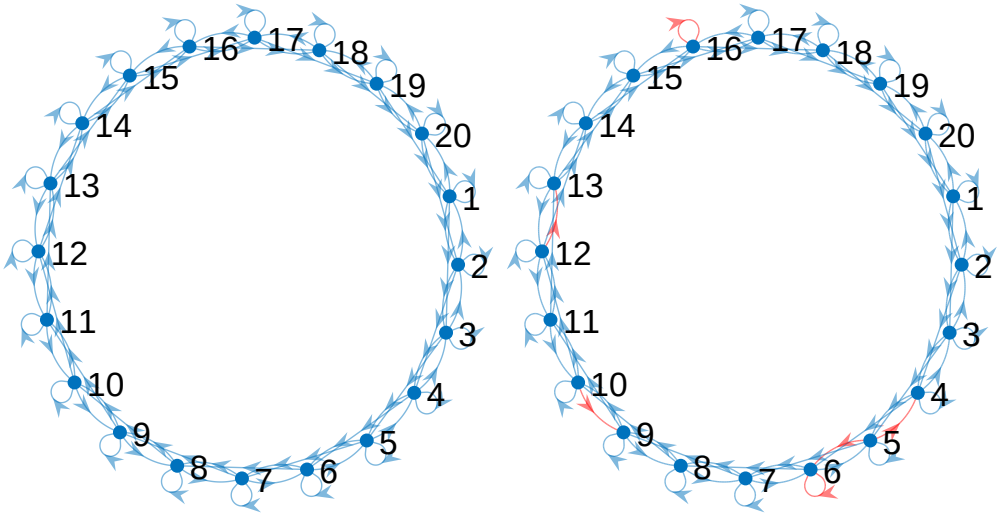


Figure 5.5: Graph encoding the connectivity of the Lorenz 96 system for $N_x=20$. The left plot represents the true connectivity of the system under the graphical perspective described in Section 5.3.2. The right plot represents the connectivity retrieved by the B-GraphGrad algorithm. The links in blue correspond to connections where the monomial is also correctly identified. The links in red correspond to connections where the monomial is incorrectly identified. In this example, the node connectivity was perfectly retrieved by B-GraphGrad, however in 6 out of the 80 links, the link was found but with an incorrect monomial (e.g., the self loop of node 16 was identified as quadratic while it should be a linear term).

We set $\mathbf{\Sigma}_v=\mathbf{\Sigma}_r=\sigma^2\mathbf{Id}_{N_x}$, with $\sigma=1$ unless stated otherwise. We discretise this system with a timestep of $\Delta t=0.025$. We choose $t_0=0$, and fix $\mathbf{x}_0$ such that $\mathbf{x}_0$ is equal to 1 in the first element, and 0 elsewhere. Our particle filter is initialised with $p(\mathbf{x}_0)=\mathcal{N}(\mathbf{x}_0,\mathbf{Id}_{N_x})$. We note that the system is of polynomial form, and therefore can be exactly recovered by our model, assuming d equal or larger than two.

We see that in both cases of $d=2$ and $d=3$, B-GraphGrad performs well, outperforming pMLE by a considerable margin. We see no significant deterioration of performance in B-GraphGrad in this setting, noting that we are estimating 4620 parameters in the $d=2$ case, and 35420 parameters in the $d=3$ case.

Table 5.8
Lorenz 96: Average recovery metrics for variable series length for 150 independent systems. Maximum polynomial degree $d=2$.

method	T	RMSE (10^{-3})	spec.	recall	prec.	F1
B-GraphGrad	25	1.8	0.92	0.93	0.93	0.93
pMLE	25	2.6	-	-	-	-
B-GraphGrad	50	1.4	0.96	0.95	0.94	0.95
pMLE	50	2.0	-	-	-	-
B-GraphGrad	100	0.8	0.99	0.97	0.98	0.98
pMLE	100	1.3	-	-	-	-
B-GraphGrad	200	0.3	1.00	1.00	1.00	1.00
pMLE	200	0.8	-	-	-	-

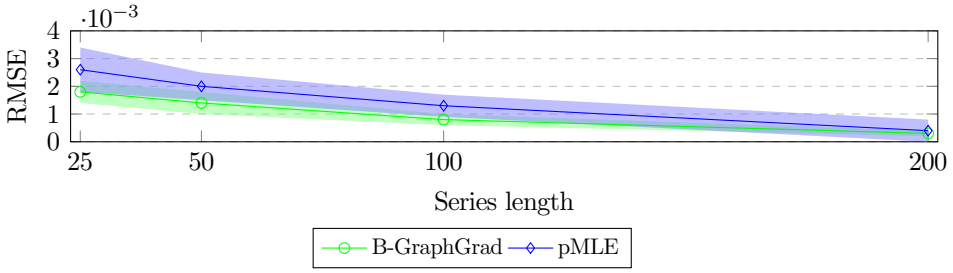


Figure 5.6: Comparison of B-GraphGrad with pMLE over variable series length on the Lorenz 96 oscillator, with maximum polynomial degree $d=2$. Markers denote mean performance, with the ribbons being symmetric 95% intervals.

Table 5.9
Lorenz 96: Average recovery metrics for variable series length for 150 independent systems. Maximum polynomial degree $d=3$.

method	T	RMSE (10^{-3})	spec.	recall	prec.	F1
B-GraphGrad	25	2.4	0.80	0.92	0.73	0.82
pMLE	25	0.32	-	-	-	-
B-GraphGrad	50	2.0	0.89	0.96	0.86	0.91
pMLE	50	2.5	-	-	-	-
B-GraphGrad	100	1.1	0.95	0.98	0.96	0.97
pMLE	100	2.2	-	-	-	-
B-GraphGrad	200	0.5	1.00	1.00	1.00	1.00
pMLE	200	1.6	-	-	-	-

5.5.4 Kuramoto oscillator

The Kuramoto oscillator [101] is a mathematical model that describes the behaviour of a system of phase-coupled oscillators. The model is described, for all $i\in\{1,...,N_x\}$, by

$$\frac{\mathrm{d}\phi_i}{\mathrm{d}t}=\eta_i+N_x^{-1}\sum_{o=1}^{N_x}K\sin(\phi_i-\phi_o), \tag{5.19}$$

where $\phi_i\in\mathbb{R}$ denotes the phase of the i-th oscillator, and $K\in\mathbb{R}$ is the coupling constant between oscillators. However, this does not restrict $\boldsymbol{\phi}$, which will, in general,

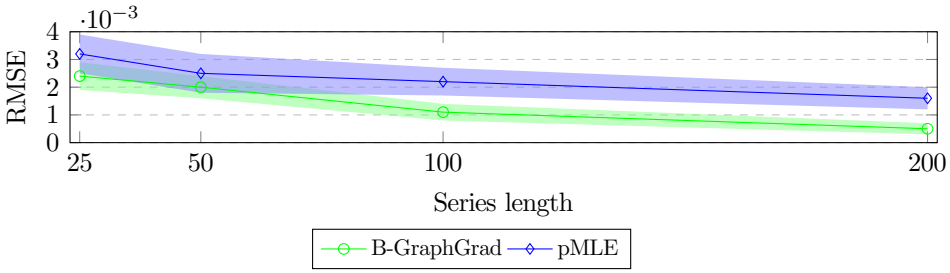


Figure 5.7: Comparison of B-GraphGrad with pMLE over variable series length on the Lorenz 96 oscillator, with maximum polynomial degree $d=3$. Markers denote mean performance, with the ribbons being symmetric 95% intervals.

diverge to $\pm\infty$ as $t\to\infty$. To address this, we transform Eq. (5.19) by introducing derived parameters R and ψ such that

$$\begin{aligned} R\exp(\sqrt{-1}\psi) &= N_x^{-1}\sum_{o=1}^{N_x}\exp(\sqrt{-1}\phi_o), \\ (\forall i\in\{1,\dots,N_x\})\ \frac{\mathrm{d}\phi_i}{\mathrm{d}t} &= \eta_i + KR\sin(\psi-\phi_i), \end{aligned} \tag{5.20}$$

which restricts $\boldsymbol{\phi}\in[-\pi,\pi]^{N_x}$. This is done purely for computational reasons, and does not affect the properties of the system or interpretation of $\boldsymbol{\phi}$ in any way. We transform Eq. (5.20) into a NLSSM using an Euler discretisation, yielding

$$\begin{aligned} R\exp(\sqrt{-1}\psi) &= N_x^{-1}\sum_{o=1}^{N_x}\exp(\sqrt{-1}x_{o,t}), \\ d_{i,t+1} &= \eta_i + KR\sin(\psi-x_i), \\ x_{i,t+1} &= x_{i,t} + \Delta t d_{i,t+1} + \sqrt{\Delta t}v_{i,t+1}, \\ y_{i,t+1} &= x_{i,t+1} + \sqrt{\Delta t}r_{i,t+1}, \end{aligned} \tag{5.21}$$

for $i\in\{1,\dots,N_x\}$,where $\mathbf{x}$ denotes the phases in the discretised system to ease comparison with the rest of the work. We choose $N_x=20$, and $K=0.8$. We set $\boldsymbol{\Sigma}_v=\boldsymbol{\Sigma}_r=\sigma^2\mathbf{Id}_{N_x}$, with $\sigma=0.1$. We discretise this model with a timestep of $\Delta t=0.05$. We sample $\eta_i\sim\mathcal{N}(0.5,0.5^2)$ and $\mathbf{x}_{i,0}\sim U(-\pi,\pi)\ \forall i\in\{1,\dots,N_x\}$. Our particle filter is initialised with $p(\mathbf{x}_0)=\mathcal{N}(\mathbf{x}_0,0.2\mathbf{Id}_{20})$. We run the system until $t=10$, and then begin collecting observations.

We note that this model cannot be represented exactly as a polynomial, and therefore the classification metrics used to illustrate sparsity recovery do not make sense. Therefore, we only present a normalised RMSE metric, where we compare the relative error in recovering the hidden state. nRMSE is given by $\mathrm{nRMSE}(\mathbf{x}_{EST},\mathbf{x}_{TM},\mathbf{x}_{GT})=\frac{\mathrm{RMSE}(\mathbf{x}_{EST},\mathbf{x}_{GT})}{\mathrm{RMSE}(\mathbf{x}_{TM},\mathbf{x}_{GT})}$, where $\mathbf{x}_{EST}$ is the sequence of means recovered by an estimator, $\mathbf{x}_{TM}$

is the sequence of means recovered by a particle filter running with the true model, and $\mathbf{x}_{GT}$ is the sequence of states we generate when creating our synthetic data. Therefore, nRMSE $= 1$ indicates identical performance in terms of state recovery between the true model and an approximation.

We compare the performance of GraphGrad against the pMLE as above, but also against the true model, where we estimate the parameters using maximum likelihood, which we denote by TrueMLE. TrueMLE is an oracle algorithm, and serves as a baseline for the case there the form of the model is known. TrueMLE assumes the form the model in Eq. (5.21) is known, and estimates the $\boldsymbol{\eta}$ and K parameters using maximum likelihood.

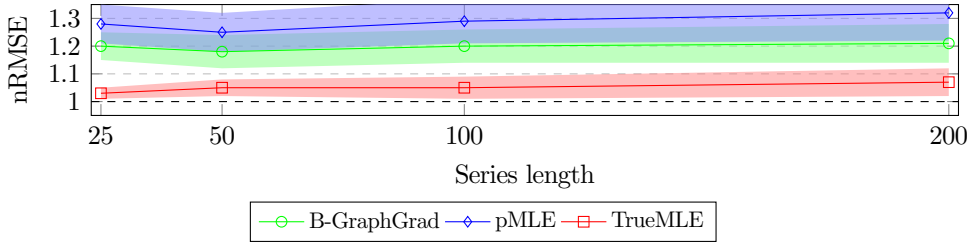


Figure 5.8: Comparison of B-GraphGrad with pMLE and TrueMLE over variable series length on the Kuramoto oscillator. Quantities are divided by the RMSE of a filter utilising the true model. Markers denote mean performance, with the ribbons being symmetric 95% intervals.

We see in Figure 5.8 that TrueMLE gives the best results, which is expected given that it is an oracle algorithm with respect to the form of the dynamics, whilst B-GraphGrad estimates both the form of the dynamics and the parameters thereof. Within these methods, GraphGrad performs significantly better than the polynomial MLE, and on average has only 20% greater RMSE than state recovery with the true model and known parameters, whilst also assuming no knowledge of the underlying dynamics.

5.6 Conclusion

In this chapter, we have proposed GraphGrad, a method for fitting a sparse polynomial approximation to the transition function of a general SSM. GraphGrad utilises a differentiable particle filter to learn a polynomial parametrisation of a general SSM using gradient methods, from which we infer the connectivity of the state. GraphGrad promotes sparsity using a proximal operator, which is computationally efficient and stable. Our method is memory efficient, requiring only to keep track of a few parameters. Moreover, it is expressive, as many well known systems can be exactly represented by polynomial

rates. Our method can utilise long observation series without vanishing gradients as we utilise observation batching to mitigate likelihood degeneracy. The method displays strong performance inferring the connectivity of a chaotic system, and keeps performing well when the true system cannot be exactly represented by the approximation.

Chapter 6

Neural Transition Kernels and Proposal Distributions for Particle Filtering

This chapter is based on the papers [102, 103].

6.1 Introduction

In previous chapters, we made relatively strong assumptions on the form of the state-space model. In particular, in Chapters 3 and 4 we assumed the system is linear-Gaussian, and in Chapter 5 we assumed that the transition dynamics could be represented in polynomial form. This is not in general the case, and therefore we propose a method that is suitable for state estimation in a general unknown model setting. However, we lose much of the interpretability of the sparse model approaches developed in Chapters 3, 4, and 5, as well as increasing the cost of evaluation and fitting. This trades off against the very general applicability of the methods proposed in this chapter, which can be utilised to learn any state-space model that evolves in continuous space.

In particle filters (PFs), we must evaluate the density of the distribution of the state given the previous state, known as the transition kernel. If the form of the transition kernel is known, with only the parameters unknown, then several methods exist to estimate the system parameters, and hence estimate the transition kernel [5, Chapter 12]. Furthermore, learning parameters in the PF has been greatly eased with the advances in differentiable particle filters (DPFs) [78, 79, 37, 82], which allow the use of gradient-based optimisation methods for parameter estimation.

Differentiable particle filters (DPFs) modify the resampling step of the PF to be differentiable, and therefore, when combined with differentiable transition and

observation densities, makes the overall particle filter differentiable. There exist a number of differentiable particle filters, which utilise many different methods for making the resampling step differentiable, such as soft resampling [36], stop-gradient operators [79], or optimal transport schemes [78].

If the form of the transition kernel is unknown, current methods require one to be assumed. In practice, it is often the case that the transition kernel has unknown form, such as when the underlying dynamics are not known. In these cases, it is required to perform filtering without prior knowledge of the transition kernel, and therefore the form and parameters of the transition kernel must be estimated. In addition to the transition kernel, in the PF we must provide a proposal distribution to generate Monte Carlo samples. The diversity of the samples, which is critically important to the method's ability to represent the system, is heavily dependent on the proposal distribution.

One approach to choosing the proposal distribution is to use the bootstrap particle filter (BPF) [32], which uses the state transition kernel as the proposal. However, the bootstrap proposal does not incorporate the observation, and therefore does not utilise all available information at each time step. Not using observation information can cause significant issues, for example, if the state transition is a very diffuse distribution but the observation is very informative, few, if any, particles will have high posterior probability, and therefore the effective sample size will be very low. Several methods incorporate the observation in their proposal distribution, such as the auxiliary particle filter (APF) [12, 104, 105, 33], which adjusts the particle weights based on both the transition dynamics and the likelihood of the new observation. Deep learning methods have also been used within the sequential Monte Carlo (SMC) and PF framework to learn the proposal distribution, such as neural adaptive SMC [106] and the variational SMC [107].

Contribution. In this chapter, we propose StateMixNN, a method to approximate the transition kernel and proposal distribution of a particle filter using a pair of adaptive Gaussian mixtures. StateMixNN improves upon methods that learn only the transition dynamics, as we also estimate the optimal proposal density, and therefore incorporate more information in our estimates than would otherwise be possible.

StateMixNN learns the mean and covariance parameters of the components of a multivariate Gaussian mixture as the output of a dense neural network. By estimating both the transition and proposal distributions, we can estimate the hidden state from an observation series given only the observation model, which cannot be estimated here for identifiability reasons. This approximation allows us to estimate distributions resulting from complex models, becoming more expressive as the number of mixture components increases.

In order to train the neural networks, we utilise the DPF framework of [79], allowing

gradient propagation through the resampling step of the particle filter, and therefore the use of gradient-based optimisation schemes. StateMixNN is semi-supervised since it trains targeting the observation log-likelihood, which requires the sequence of observations, but does not require the underlying hidden states.

6.2 Problem statement and background

6.2.1 Considered state-space model

In this chapter, we consider state-space models of the form

$$\begin{aligned} \mathbf{x}_t &\sim f(\mathbf{x}_t|\mathbf{x}_{t-1};\boldsymbol{\theta}^{(f)}), \\ \mathbf{y}_t &\sim g(\mathbf{y}_t|\mathbf{x}_t;\boldsymbol{\theta}^{(g)}), \end{aligned} \tag{6.1}$$

where $t \in \{1,...,T\}$ denotes discrete time, $\mathbf{x}_t \in \mathbb{R}^{d_x}$ is the state of the system at time t, $\mathbf{y}_t \in \mathbb{R}^{d_y}$ is the observation associated with $\mathbf{x}_t$, $\boldsymbol{\theta}^{(g)}$ and $\boldsymbol{\theta}^{(f)}$ are sets of parameters relating to the observation and state dynamics respectively, and the distributions f and g encode the transition and observation model respectively.

In terms of densities, $f(\mathbf{x}_t|\mathbf{x}_{t-1};\boldsymbol{\theta}^{(f)})$ is the conditional density of the state $\mathbf{x}_t$ given $\mathbf{x}_{t-1}$, and $g(\mathbf{y}_t|\mathbf{x}_t;\boldsymbol{\theta}^{(g)})$ is the conditional density of the observation $\mathbf{y}_t$ given the hidden state $\mathbf{x}_t$. The initial value of the state, $\mathbf{x}_0$, is distributed as $\mathbf{x}_0 \sim p(\mathbf{x}_0|\boldsymbol{\theta}^{(p)})$. The hidden state sequence $\mathbf{x}_{0:T}$ is not observed, while the related sequence of observations $\mathbf{y}_{1:T}$ is observed.

6.2.2 Particle filtering

We reproduce the SIR particle filter of Alg. 3 in Alg. 14, with minor adjustments for clarity in explaining our method. In this method, at time step t, we compute a set of particles and associated importance weights via the following steps. First, we draw K ancestor indices $\{a_t^{(k)}\}_{k=1}^K$, (line 4), which is equivalent to resampling the particle set K times with replacement, with the probability of drawing particle $\mathbf{x}_{t-1}^{(k)}$ equal to its weight $\bar{w}_{t-1}^{(k)}$. Resampling is important to avoid weight degeneracy in the filter, as it helps to avoid the weights becoming concentrated in a diminutive subset of the sampled particles. However, resampling results in path degeneracy [108], where all particles will eventually share a common ancestor trajectory. Path degeneracy does not impact the properties of the filter at each time step however, so is considered far less damaging than weight degeneracy. Next, we draw K particles $\{\mathbf{x}_t^{(k)}\}_{k=1}^K$ from the proposal distribution $\pi(\mathbf{x}_t|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)},\mathbf{y}_t;\boldsymbol{\theta}^{(\pi)})$ (line 6). Then, we compute the importance weights $\{w_t^{(k)}\}_{k=1}^K$

(line 7), which takes into account the transition kernel and the current observation. Finally, we normalise the weights to sum up to 1, obtaining $\{\bar{w}_t^{(k)}\}_{k=1}^K$ (line 8).

Algorithm 14 Sequential importance resampling (SIR)

1: Draw $\mathbf{x}_0^{(k)} \sim p(\mathbf{x}_0|\boldsymbol{\theta}^{(p)})$, for $k \in 1,...,K$.
2: Set $\widetilde{w}_0^{(k)} = 1/K$, for $k \in 1,...,K$.
3: **for** $t \in 1,...,T$ and $k \in 1,...,K$ **do**
4: Sample $a_t^{(k)} \sim \text{Categorical}(\bar{w}_{t-1})$.
5: Set $\widetilde{w}_t^{(k)} = \frac{1}{K}$.
6: Sample $\mathbf{x}_t^{(k)} \sim \pi(\mathbf{x}_t|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)})$.
7: Compute $w_t^{(k)} = \frac{g(\mathbf{y}_t|\mathbf{x}_t^{(k)};\boldsymbol{\theta}^{(g)}) f(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)};\boldsymbol{\theta}^{(f)})}{\pi(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)})}$.
8: Compute $\overline{w}_t^{(k)} = w_t^{(k)} / \sum_{i=1}^K w_t^{(i)}$.
9: **end for**

Given the sequential importance resampling (SIR) algorithm, it is apparent that the choice of proposal distribution $\pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)})$ is critical to the success of the estimation; the particles should be concentrated in regions of the state space with high probability mass. There exist several methods to choose the proposal distribution.

One such method is the BPF [32], where the proposal distribution is set equal to the transition kernel of the state-space model (SSM), with $\pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)}) = f(\mathbf{x}_t|\mathbf{x}_{t-1}; \boldsymbol{\theta}^{(f)})$. Such an approach is simple, and results in less computation due to cancellation in line 7 of Alg. 14, resulting in $w_t^{(k)} = g(\mathbf{y}_t|\mathbf{x}_t^{(k)}; \boldsymbol{\theta}^{(g)})$ for the bootstrap particle filter. However, the BPF does not incorporate the observation $\mathbf{y}_t$ in the proposal distribution, and therefore omits potentially important information. For example, if the transition kernel is far more diffuse than the observation distribution, then the observation contains a lot of information relative to the previous state, which the BPF does not use.

The optimal proposal distribution incorporates the observation at the current time step t, i.e., $\pi(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)}) = p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t)$. In general distribution is intractable or unknown. Methods such as the APF and its derivatives [12, 33, 104, 105] incorporate the observation information via a pre-weighting step.

[109, 110] propose several methods to learn the proposal distribution in a general setting, such as modelling it as a Gaussian distribution with mean and covariance output by a neural network, or utilising normalising flows. These methods learn a distinct network for each time step, and therefore do not generalise outside of a single series of observations.

6.2.3 Differentiable particle filters

following The particle filter as given in Alg. 14 contains non-differentiable operations using the weights and particle values, and hence is not differentiable with respect to the model parameters $\boldsymbol{\theta}^{(f)}$,$\boldsymbol{\theta}^{(\pi)}$, or $\boldsymbol{\theta}^{(g)}$, from the two sampling steps (lines 6 and 4). The proposal sampling step (line 6) can often be rewritten to be differentiable with respect to the parameters $\boldsymbol{\theta}^{(f)}$ and $\boldsymbol{\theta}^{(\pi)}$ using distribution specific reparametrisations, such as the reparametrisation trick for Gaussian distributions[1] [100].

The resampling step (line 4) remains, which requires sampling a categorical or multinomial distribution, depending on implementation. Sampling either of these distributions is a non differentiable operation, as infinitesimal changes in the weights result in discrete changes in the sampled indices [111]. A heuristic reasoning for this non-differentiability can be made by noting that $a \sim \text{Categorical}(p)$ can be sampled by drawing $u \sim \text{Uniform}(0,1)$, and then setting $a = \max N$ s.t. $\sum_{n=1}^{N} p_n \leq u$. If we have $\sum_{n=1}^{N} p_n = u$ for some I, then an infinitesimal increase in p_N by any $\epsilon > 0$, will result in a discrete change in a from N to $N+1$. Therefore, sampling Categorical(p) is not differentiable with respect to p.

Due to resampling not being differentiable, any function of the weights is non-differentiable, with one such function being the likelihood $\ell(\mathbf{y}_{1:T}|\boldsymbol{\theta})$, which is a common target function for parameter estimation in particle filtering. Note that the particle trajectories are also functions of the weights, so functions of trajectories are also non-differentiable if using Alg. 14.

Differentiable particle filtering (DPF) modifies the resampling step to achieve differentiability. Therefore, when combined with a differentiable transition step, the overall particle filter is differentiable [78, 37, 79, 82]. There exist several DPF methods, each utilising a different method to make the resampling step differentiable. In this chapter, we use the stop-gradient DPF of [79], which we present in Alg. 15. The stop-gradient DPF yields gradient estimates of a given loss function with minimal computational overhead [79]. This method achieves differentiability using two steps. First, the stop-gradient operator is applied to the weights when sampling particle ancestry in line 3, and thereby we do not require to compute gradients through the discrete sampling. However, gradient information is then propagated through a transformation of the weights in line 4. Therefore, we can take gradients of filter outputs, such as the particle trajectories and the particle weights, with respect to the filter inputs. An overview of differentiable particle filter methods is given in [37], and

[1] The Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ can be sampled differentiably by noting that $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \sim \boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{\varepsilon}$, where $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T$, and $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$, and taking gradients of this transformation accordingly, noting that $\boldsymbol{\varepsilon}$ is independent of the parameters of the initial distribution, and is therefore a constant for the purposes of the derivative.

a detailed treatment of the specific filter we use is given in [79].

Algorithm 15 Stop-gradient differentiable particle filter (DPF) [79]

1: Draw $\mathbf{x}_0^{(k)} \sim p(\mathbf{x}_0|\boldsymbol{\theta}^{(p)})$, for $k \in 1,...,K$. State Set $\overline{w}_0^{(k)} = 1/K$, for $k = 1,...,K$
2: **for** $t \in 1,...,T$ and $k \in 1,...,K$ **do**
3: Sample $a_t^{(k)} \sim \text{Categorical}(\perp(\overline{w}_{t-1}))$.
4: Set $\overline{w}_{t-1}^{(k)} = \frac{1}{K}\, \overline{w}_{t-1}^{\left(a_t^{(k)}\right)} / \perp(\overline{w}_{t-1}^{\left(a_t^{(k)}\right)})$.
5: Sample $\mathbf{x}_t^{(k)} \sim \pi(\mathbf{x}_t|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)})$.
6: Compute $w_t^{(k)} = \frac{g(\mathbf{y}_t|\mathbf{x}_t^{(k)};\boldsymbol{\theta}^{(g)}) f(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)};\boldsymbol{\theta}^{(f)})}{\pi(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)}, \mathbf{y}_t; \boldsymbol{\theta}^{(\pi)})}$.
7: Compute $\overline{w}_t^{(k)} = w_t^{(k)} / \sum_{i=1}^{K} w_t^{(i)}$.
8: **end for**

where $\perp(x)$ is the stop gradient operator as defined in [79], which yields x when evaluated, but always has a gradient of 0.

We can use differentiable particle filters to estimate the gradient of the log-likelihood, and therefore apply gradient methods to estimate unknown parameters of our state-space model. However, as the log-likelihood is estimated, we will also have noise in gradients thereof, and therefore the parameter estimation scheme must be robust to noisy gradients. Noisy gradients are common in deep learning, as it is typical to compute the gradient of a stochastically selected subset of the training data, which results in gradient noise due to sampling. Several optimisation methods have been proposed for use in deep learning, with a common feature being robustness to stochastic gradients. We can utilise these methods when estimating parameters using differentiable particle filters, with schemes such as ADAM [38], RADAM [112], and Novograd [93, 83] all being applicable to this problem.

6.3 Proposed algorithm

We propose StateMixNN, a method to learn the transition and proposal distribution of a generic state-space model. We approximate the transition kernel and the proposal distribution by a pair of multivariate Gaussian mixtures parametrised by neural networks.

By combining learnt estimates of both the transition kernel and the proposal distribution, we can learn a generic state-space model conditional only on knowledge of the observation model. Note that we discuss and justify several of the choices made here, such as the use of Gaussian mixtures, equal mixture weights, and diagonal covariances, in Section 6.4.

6.3.1 Parametrising the Gaussian mixture

StateMixNN learns the transition kernel conditional on only the previous state value, thus preserving the Markovianity of the SSM. We approximate $f(\mathbf{x}_t|\mathbf{x}_{t-1};\boldsymbol{\theta}^{(f)})$ by an equally weighted multivariate Gaussian mixture of S_f components with diagonal covariances

$$f(\mathbf{x}_t|\mathbf{x}_{t-1};\boldsymbol{\theta}^{(f)}) := (S^{(f)})^{-1}\sum_{s=1}^{S_f} f_s(\mathbf{x}_t|\mathbf{x}_{t-1};\boldsymbol{\theta}^{(f)}), \tag{6.2}$$

with $s \in \{1,...,S_f\}$, where $\boldsymbol{\theta}^{(f)}$ are the parameters of the state neural network, and f_s is given by

$$f_s(\mathbf{x}_t|\mathbf{x}_{t-1};\boldsymbol{\theta}^{(f)}) = \mathcal{N}\big(\mu_s^{(f)}(\mathbf{x}_{t-1}), \Sigma_s^{(f)}(\mathbf{x}_{t-1})\big). \tag{6.3}$$

StateMixNN learns the proposal distribution conditioned on the previous state value and the current observation, as is the case for the intractable optimal proposal distribution. We parametrise $\pi(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{y}_t;\boldsymbol{\theta}^{(\pi)})$ as an equally weighted mixture of S_π multivariate Gaussian distributions with diagonal covariances

$$\pi(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{y}_t;\boldsymbol{\theta}^{(\pi)}) := (S^{(\pi)})^{-1}\sum_{s=1}^{S_\pi} \pi_s(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{y}_t;\boldsymbol{\theta}^{(\pi)}), \tag{6.4}$$

with $s \in \{1,...,S_\pi\}$, where $\boldsymbol{\theta}^{(\pi)}$ are the parameters of the proposal neural network, and π_s is given by

$$\pi_s(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{y}_t;\boldsymbol{\theta}^{(\pi)}) = \mathcal{N}\big(\mu_s^{(\pi)}(\mathbf{x}_{t-1},\mathbf{y}_t), \Sigma_s^{(\pi)}(\mathbf{x}_{t-1},\mathbf{y}_t)\big). \tag{6.5}$$

Mixtures of multivariate Gaussians can represent a wide range of possible distributions, offering flexibility and expressiveness that is beneficial for modelling complex systems [23].

6.3.2 Network architecture and learning

For real data, the transition kernel of a state-space model is, in general, unknown. By estimating the transition kernel, we can utilise the state-space model framework to perform estimation in an unknown model scenario. Furthermore, the optimal proposal distribution is separate from the state-space model, and is in general intractable. Using a proposal distribution that approximates the optimal proposal distribution will improve the performance of the particle filter by way of making the sampling more efficient, thereby yielding more accurate estimates of state quantities by way of increased effective sample size.

The transition kernel can be approximated utilising neural networks, which are

efficient to evaluate, and can approximate any general continuous function (in the infinite width/depth limit) due to the universal approximation theorem. Small neural networks are simple to fit via gradient descent, and easy to implement due to the proliferation of deep learning frameworks. In this chapter we restrict ourselves to multilayer perceptrons, however the method could be extended to other forms of network.

Our method learns the mean functions $\mu^{(f)}(\cdot)$, $\mu^{(\pi)}(\cdot)$ and the covariance functions $\Sigma^{(f)}(\cdot)$, $\Sigma^{(\pi)}(\cdot)$ for the transition and proposal distributions using a pair of dense neural networks. We denote these networks by $\mathrm{NN}^{(f)}(\mathbf{x}_{t-1};\boldsymbol{\theta}^{(f)})$ for the network associated with the transition kernel and by $\mathrm{NN}^{(\pi)}(\mathbf{x}_{t-1},\mathbf{y}_t;\boldsymbol{\theta}^{(\pi)})$ for the network associated with the proposal distribution. We denote by $\mathrm{NN}(\cdot;\boldsymbol{\theta})$ a general dense neural network from which the proposal and transition networks are parametrised.

The transition network takes as input only the previous particle value, therefore preserving the Markovianity assumption of the SSM, which is often violated when using neural networks [4, 113, 114, 115]. The general dense network $\mathrm{NN}(\cdot;\boldsymbol{\theta})$ is comprised of L layers, with

$$\mathrm{NN}(\mathbf{z}_0;\boldsymbol{\theta})=\mathbf{z}_L, \qquad \mathbf{z}_l=\rho_l(\mathbf{A}_l\mathbf{z}_{l-1}+\mathbf{b}_l), \tag{6.6}$$

where $\mathbf{A}_l$ and $\mathbf{b}_l$ are the weight matrix and bias vector of the lth layer, ρ_l is the non-linear activation function of the lth layer, and $\mathbf{z}_l$ is the output of the lth layer for $l\in 1,...,L$. Therefore, $\boldsymbol{\theta}=\{\mathbf{A}_1,\mathbf{b}_1,...,\mathbf{A}_L,\mathbf{b}_L\}$ are the learned parameters for a network. For the proposal network the initial value $\mathbf{z}_0^{(\pi)}=[\mathbf{x}_{t-1}^\intercal,\mathbf{y}_t^\intercal]^\intercal$ is the concatenation of the previous state $\mathbf{x}_{t-1}$ and the current observation $\mathbf{y}_t$. Thus, $\mathbf{z}_0^{(\pi)}$ has dimension $d_0^{(\pi)}=d_x+d_y$. For the state network the initial value $\mathbf{z}_0^{(f)}=\mathbf{x}_{t-1}$ is the previous state $\mathbf{x}_{t-1}$. Thus, $\mathbf{z}_0^{(f)}$ has dimension $d_0^{(f)}=d_x$.

For both networks, the dimension of the output

$$\mathbf{z}_L=[\boldsymbol{\mu}^{(1)\intercal},\mathbf{c}^{(1)\intercal},...,\boldsymbol{\mu}^{(S)\intercal},\mathbf{c}^{(S)\intercal}]^\intercal \tag{6.7}$$

is $d_L:=2Sd_x$, as for each of the mixture components of both distributions, we require a d_x-dimensional mean vector $\boldsymbol{\mu}^{(n)}$ and a d_x-dimensional covariance scale vector $\mathbf{c}^{(n)}$.

In both cases, we construct a Gaussian mixture distribution from a neural network $\mathrm{NN}(\cdot;\boldsymbol{\theta})$ by

$$\mathcal{GM}(\mathrm{NN}(\cdot;\boldsymbol{\theta}))=\sum_{s=1}^{S}S^{-1}\mathcal{N}(\boldsymbol{\mu}^{(s)},\mathbf{C}^{(s)}), \tag{6.8}$$

where S is the number of mixture components, with $\boldsymbol{\mu}^{(s)},\boldsymbol{c}^{(s)},s\in 1,...,S$ extracted from $\mathbf{z}_L$ by indexing per Eq. (6.7), where we discard the components of $\boldsymbol{\theta}$ that are not relevant to NN, and $\mathbf{C}^{(s)}=\mathrm{diag}(\mathbf{c}^{(s)})^2$. For example, when constructing f, we use

$\mathcal{GM}(\text{NN}^{(f)}(\cdot;[\boldsymbol{\theta}^{(f)},\boldsymbol{\theta}^{(\pi)}]))$, discarding $\boldsymbol{\theta}^{(\pi)}$ as it is not used in $\text{NN}^{(f)}$. We make use of this property in Alg. 18.

The learned parameters for our neural networks are the weights $\mathbf{A}_l$ and the biases $\mathbf{b}_l$, with $l \in 1,...,L$. The number of layers L, the dimension of each layer $d_l, l \in 1,...,L$, and the activation functions $\rho_l, l \in 1,...,L$ are fixed as part of the network architecture.

We learn separate networks for the transition and proposal distributions, denoted by $\text{NN}^{(f)}$ and $\text{NN}^{(\pi)}$ respectively. Therefore, when learning the transition kernel we estimate $\boldsymbol{\theta}^{(f)} := \{\mathbf{A}_1^{(f)},\mathbf{b}_1^{(f)},...,\mathbf{A}_{L^{(f)}}^{(f)},\mathbf{b}_{L^{(f)}}^{(f)}\}$, and estimate $\boldsymbol{\theta}^{(\pi)} := \{\mathbf{A}_1^{(\pi)},\mathbf{b}_1^{(\pi)},...,\mathbf{A}_{L^{(\pi)}}^{(\pi)},\mathbf{b}_{L^{(\pi)}}^{(\pi)}\}$ when learning the proposal distribution.

To train the networks we train to maximise the estimated log-likelihood, given by

$$\ell(\boldsymbol{\theta}|\mathbf{y}_{1:T}) \propto \sum_{t=1}^{T}\left(\sum_{k=1}^{K}\log\left(w_t^{(k)}\right)\right), \tag{6.9}$$

where $w_t^{(k)}$ and $\widetilde{w}_{t-1}^{(k)}$ are the unnormalised and normalised weights of the PF in Alg. 14, and we assume a uniform flat improper prior $p(\boldsymbol{\theta}) \propto 1$ [5, Chapter 12]. Note that the weights are dependent on the parameters $\boldsymbol{\theta}^{(f)},\boldsymbol{\theta}^{(g)}$, and $\boldsymbol{\theta}^{(\pi)}$, as the weights are computed using densities dependent on these parameters.

In our case, we assume that the observation model g is known, and hence we omit the dependence on $\boldsymbol{\theta}^{(g)}$. The log-likelihood is maximised when all weights are equal, a desirable property as it reduces weight degeneracy over time [5]. Furthermore, maximising the log-likelihood does not require knowledge of the true value of the hidden state, which is often unavailable, requiring only the observation series.

6.3.3 StateMixNN algorithm

Algorithm 16 StateMixNN algorithm $\quad$ StateMixNN($B,J,A,\mathbf{y},\text{NN}^{(f)},\text{NN}^{(\pi)}$)

Input: Number of batches B, steps per batch J, number of iterations A, observations $\mathbf{y}$, transition network $\text{NN}^{(f)}$, proposal network $\text{NN}^{(\pi)}$.
Initialise $\boldsymbol{\theta}_0^{(\pi)} := \{\mathbf{A}_1^{(\pi)},\mathbf{b}_1^{(\pi)},...,\mathbf{A}_{L^{(\pi)}}^{(\pi)},\mathbf{b}_{L^{(\pi)}}^{(\pi)}\}$.
Initialise $\boldsymbol{\theta}_{-1}^{(f)} := \{\mathbf{A}_1^{(f)},\mathbf{b}_1^{(f)},...,\mathbf{A}_{L^{(f)}}^{(f)},\mathbf{b}_{L^{(f)}}^{(f)}\}$.
Set $\boldsymbol{\theta}_0^{(f)} := \text{ConditionalUpdate}(B,J,\boldsymbol{\theta}_{-1}^{(f)},\boldsymbol{\theta}_{-1}^{(f)},\mathbf{y})$ using Alg. 17
for $a \in 1,...,A$ **do**
    Set $\boldsymbol{\theta}_a^{(\pi)} := \text{ConditionalUpdate}(B,J,\boldsymbol{\theta}_{a-1}^{(\pi)},\boldsymbol{\theta}_{a-1}^{(f)},\mathbf{y})$ using Alg. 17.
    Set $\boldsymbol{\theta}_a^{(f)} := \text{ConditionalUpdate}(B,J,\boldsymbol{\theta}_{a-1}^{(f)},\boldsymbol{\theta}_a^{(\pi)},\mathbf{y})$ using Alg. 17.
end for
return $\boldsymbol{\theta}_A^{(f)}$, $\boldsymbol{\theta}_A^{(\pi)}$.

StateMixNN is described in Alg. 16. We first describe the overall algorithm, given in Alg. 16, and proceed to describe the subordinate algorithms, Alg. 17 and Alg. 18 in turn.

In Alg. 16, we begin by initialising the network parameters (lines 2 and 3 of Alg. 16). It is not crucial how these parameters are initialised, but we note that our software implementation utilises element wise random uniform initialisations for $\mathbf{A}_l$ and $\mathbf{b}_l$, with values in the range $\pm\sqrt{d_l}$, following [116]. We then learn an initial value for the transition kernel (line 4 of Alg. 16) by optimising the network parameters of the transition kernel in a bootstrap particle filter, wherein the transition and proposal distributions are the same.

We then perform A alternating conditional updates. We optimise each of $\boldsymbol{\theta}^{(f)}$ and $\boldsymbol{\theta}^{(\pi)}$ conditional on the other, similar to a coordinate descent method. At iteration a, we first optimise the proposal distribution parameters $\boldsymbol{\theta}_a^{(\pi)}$, conditional on the value of the transition kernel parameters $\boldsymbol{\theta}_{a-1}^{(f)}$. We then optimise the transition kernel parameters $\boldsymbol{\theta}_a^{(f)}$ conditional on the value of the proposal distribution parameters $\boldsymbol{\theta}_a^{(\pi)}$.

Algorithm 17 Conditional update algorithm ConditionalUpdate(B,J,$\boldsymbol{\theta}_0$,$\boldsymbol{\theta}_{\text{static}}$,$\mathbf{y}$)

1: **Input:** Number of batches B, steps per batch J, initial parameters $\boldsymbol{\theta}_0$, static parameters $\boldsymbol{\theta}_{\text{static}}$, observations $\mathbf{y}$.
2: Initialise $\boldsymbol{\theta}_{0,J} := \boldsymbol{\theta}_0$.
3: **for** $b \in 1,\ldots,B$ **do**
4: Set $\mathbf{y}^{(b)} := \mathbf{y}_{1:\lceil bT/B \rceil}$.
5: Set $\boldsymbol{\theta}_{b,0} := \boldsymbol{\theta}_{b-1,J}$.
6: **for** $j \in 1,\ldots,J$ **do**
7: Set $\boldsymbol{\theta}_{b,j} := \text{UpdateStep}(\boldsymbol{\theta}_{b,j-1}, \boldsymbol{\theta}_{\text{static}}, \mathbf{y}^{(b)})$ using Alg. 18.
8: **end for**
9: **end for**
10: **return** $\boldsymbol{\theta}_{B,J}$.

We now describe the conditional update, given in Alg. 17. The conditional update algorithm takes an initial value of the parameters of the distribution of interest $\boldsymbol{\theta}_0$, and the parameters of the distribution we are conditioning on $\boldsymbol{\theta}_{static}$. We split the observation series $\mathbf{y}_{1:T}$ into B batches, with the b-th batch given by $\mathbf{y}^{(b)} = \mathbf{y}_{1:\lceil bT/B \rceil}$. Note that this construction implies $\mathbf{y}^{(1)} \subseteq \mathbf{y}^{(2)} \subseteq \cdots \subseteq \mathbf{y}^{(B-1)} \subseteq \mathbf{y}^{(B)}$. We construct these telescoping batches of observations to avoid likelihood issues stemming from unadapted parameters, since the first sampled trajectories often have an extremely small log-likelihood, which causes numerical errors when computing the weights in Alg. 14, leaving to the gradients of the log-likelihood exploding [31, 79].

We iterate over each of the b batches, and for each batch perform J optimisation steps. For the j-th optimisation step of batch b, we run Alg. 18 with $\boldsymbol{\theta}_{\text{learn}} = \boldsymbol{\theta}_{b,j-1}, \boldsymbol{\theta}_{\text{static}} = \boldsymbol{\theta}_{\text{static}}$ and observations $\mathbf{y} = \mathbf{y}^{(b)} := \mathbf{y}_{1:\lceil bT/B \rceil}$. Note that Alg. 17 runs the particle filter a total of JB times, once for each of the J steps taken in each

of the B batches. As we perform 2 runs of Alg. 17 in each of the A iterative steps of Alg. 16, we run the particle filter a total of $2ABJ$ times in total.

Algorithm 18 Update step UpdateStep($\boldsymbol{\theta}_{\text{learn}}$,$\boldsymbol{\theta}_{\text{static}}$,$\mathbf{y}$)

1: **Input:** Parameters to learn $\boldsymbol{\theta}_{\text{learn}}$, static parameters $\boldsymbol{\theta}_{\text{static}}$ observations $\mathbf{y}$.
2: Set $f(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{GM}(\text{NN}^{(f)}(\mathbf{x}_{t-1};[\boldsymbol{\theta}_{\text{learn}},\boldsymbol{\theta}_{\text{static}}]))$.
3: $\pi(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{y}_t) := \mathcal{GM}(\text{NN}^{(\pi)}(\mathbf{x}_{t-1},\mathbf{y}_t;[\boldsymbol{\theta}_{\text{learn}},\boldsymbol{\theta}_{\text{static}}]))$.
4: Run a particle filter (Alg. 15) with transition kernel f, proposal distribution π, and observations $\mathbf{y}$.
5: Obtain $\ell(\boldsymbol{\theta}_{learn})$ and $\nabla\ell(\boldsymbol{\theta}_{\text{learn}})$ from the particle filter via Eq. (6.9) and autodifferentiation.
6: Obtain $\boldsymbol{\theta}_{out}$ by applying a gradient based update to $\boldsymbol{\theta}_{\text{learn}}$ with gradients $\nabla\ell(\boldsymbol{\theta}_{\text{learn}})$.
7: **return** $\boldsymbol{\theta}_{out}$.

Finally, we describe a single step of the inner update algorithm, given by Alg. 18. In each step, we construct the transition kernel (line 2 of Alg. 18) and proposal distribution (line 3 of Alg. 18) corresponding to the parameters at that step. We construct each distribution using Eq. (6.8), and thereby discard the parameters not relevant to a given network. In particular, $\text{NN}^{(\cdot)}(\cdot;[\boldsymbol{\theta}_{\text{learn}},\boldsymbol{\theta}_{\text{static}}])$ takes as arguments both the static parameter, and the parameter we are learning when constructing both distributions. However, each distribution takes either the static parameter, or the parameter we are learning, and we discard the unused parameter for the purposes of constructing that distribution. For example, when learning the the transition kernel f, $\text{NN}^{(f)}$ takes both $\boldsymbol{\theta}_{\text{learn}}$, corresponding to $\boldsymbol{\theta}^{(f)}$ and $\boldsymbol{\theta}_{\text{static}}$, corresponding to $\boldsymbol{\theta}^{(\pi)}$, in line 2 of Alg. 18, but discards $\boldsymbol{\theta}_{\text{static}}$ as $\boldsymbol{\theta}_{\text{static}}$ parametrises the proposal distribution π when learning f.

After constructing the transition kernel f and the proposal distribution π, we run a particle filter with these distributions (line 4 of Alg. 18), using the observations $\mathbf{y}$, which may be a subset of the overall series of observations. From the particle filter we obtain an estimate of the log-likelihood of the parameter $\boldsymbol{\theta}_{\text{learn}}$, denoted by $\ell(\boldsymbol{\theta}_{\text{learn}})$, and an estimate of the gradient of the log-likelihood with respect to $\boldsymbol{\theta}_{\text{learn}}$, denoted by $\nabla\ell(\boldsymbol{\theta}_{\text{learn}})$ (line 5). We then utilise a gradient update scheme, such as ADAM [38] or Novograd [93, 83], to apply a gradient update to the parameter $\boldsymbol{\theta}_{\text{learn}}$, and output the result of this update to be used within line 7 of Alg. 17.

6.3.4 Discussion

Our method approximates the transition and proposal distributions by multivariate Gaussian mixture distributions. These mixtures are capable of representing complex unknown distributions, and are in many situations both more interpretable and more reliable than methods common in machine learning literature such as normalising

flows [109, 110, 117, 118]. Furthermore, in the context of particle filters, normalising flows have been observed to be susceptible to overfitting, in addition to being less intuitive and harder to train [110, 109].

We note that the particle weights w_t depend on the samples $\mathbf{x}_{0:t}$, which are drawn from the proposal distribution that we are learning. Furthermore, the computation of the weights requires evaluating the density of the transition kernel. Therefore, we require a way to propagate gradients through particle resampling and sampling the proposal mixture distribution. We use the stop-gradient differentiable particle filter of [79], given in Alg. 15, to propagate gradients through the resampling step of the particle filter. By including resampling in the training using this method we improve the convergence characteristics of StateMixNN, and allow our method to be utilised to learn complex systems, as weight degeneracy is addressed in training via the resampling step [36, 37].

In order to sample the multivariate Gaussian mixture distributions f and π, we draw the component from a categorical distribution, and then sample the associated multivariate Gaussian distribution. We reparametrise the categorical distribution using the Gumbel-Softmax reparametrisation [119], thereby allowing gradient propagation through the categorical sampling. However, it is equally valid to use a stop-gradient sampling for the categorical distribution, which is computationally more efficient. We can propagate gradients through sampling a multivariate Gaussian using the reparametrisation trick [100], where we externalise the randomness to a input of an affine transformation, which, crucially, is independent $\boldsymbol{\theta}^{(f)}$ and $\boldsymbol{\theta}^{(\pi)}$, and therefore has zero gradient with respect to our learned parameters. The above, in combination with the differentiable particle filter method of [79], allow us to compute the gradient of Eq. (6.9) with respect to our parameters $\boldsymbol{\theta}^{(p)} = \{\mathbf{A}_1^{(p)}, \mathbf{b}_1^{(p)}, \ldots, \mathbf{A}_{L^{(p)}}^{(p)}, \mathbf{b}_{L^{(p)}}^{(p)}\}, p \in \{\pi, f\}$, and therefore train the proposal and transition networks using a gradient scheme, such as [38, 112].

6.4 Discussion about the StateMixNN framework

We now discuss some characteristics of Alg. 16, and explain several of the choices we have made. First, we discuss the reasoning for using the multivariate Gaussian mixture distribution for our approximating distributions. Then, we justify the restriction to diagonal covariance and equal mixture weights. Third, we justify the alternating estimation scheme of Alg.16. Finally, we discuss how we combat the phenomena of likelihood degeneracy and concentration in our method.

6.4.1 Choice of a Gaussian mixture distribution for π and f, and extensions

In this chapter, we propose (in Eq. (6.8)) to use a multivariate Gaussian mixture distribution to approximate the optimal proposal distribution and the state transition kernel. The Gaussian mixture is able to approximate a wide range of distributions, whilst retaining computational speed. Furthermore, Gaussian mixtures are interpretable, and are easier to understand and infer from than approximating distributions such as normalising flows [120].

Most deep learning frameworks have fast, differentiable implementations of the Gaussian distribution, and typically will have the capability to combine distributions resulting in a mixture distribution. Using standard methods such as these allows for rapid, optimised implementation of our method in a given differentiable programming framework, for example PyTorch [98] and JAX [97], which both have standard constructs which can be used to construct efficient multivariate Gaussian mixtures.

We note that our method can be trivially extended to mixture distributions for which the components can be parametrised by their location and scale parameters, with an example of such an extension being a mixture of multivariate t distributions with a fixed degree of freedom ν. To do so, we would replace the $\mathcal{N}(\boldsymbol{\mu}^{(s)},\mathbf{C}^{(s)})$ mixands in Eq. (6.8) with $t_\nu(\boldsymbol{\mu}^{(s)},\mathbf{C}^{(s)})$, noting that ν is fixed. Using the t distribution may capture tail behaviour better; however, the t distribution is more expensive both to sample from and to evaluate the density of. For example, the density of the multivariate t distribution requires evaluating the Γ function, whereas the density for the Gaussian distribution requires only exponentiation and standard linear algebra operations.

6.4.2 Restriction to diagonal covariance and equal weights in the approximating mixture

In Section 6.3.2 we present our method, restricting the covariance of mixture components, $\mathbf{C}^{(s)}$, to a diagonal form, and the mixture weights to be uniformly equal to S^{-1} in Eq. (6.8). We present our reasoning for these restrictions below.

Covariances As we estimate both the state and proposal distributions f and π, we infer complex dependencies between state dimensions in the interaction between the distributions, and therefore do not need to estimate full covariance matrices. We assume the form of the covariance matrices in order to reduce the number of parameters required; we can parametrise a N dimensional distribution by $2N$ parameters, whereas estimating the full covariance matrix would require $N+N(N+1)/2$ parameters, N for the mean $\boldsymbol{\mu}^{(s)}$, and $N(N+1)/2$ for the covariance $\mathbf{C}^{(s)}$, as covariance matrices are

positive semi-definite and are therefore symmetric. Furthermore, increasing the number of estimated parameters also increases the required dimension of the output of each network, $\mathbf{z}_L$, and therefore increases the dimension of the hidden layers parameters, $\mathbf{A}_l$ and $\mathbf{b}_l$, $l \in 1,...,L$, required to obtain a comparable estimation power, drastically increasing the number of network parameters that must be learnt at each step.

A diagonal covariance also allows for a efficient implementation of the multivariate Gaussian distribution. In general, evaluating the likelihood of a multivariate Gaussian in the covariance form requires performing either a matrix inverse, or utilising the Cholesky of the covariance and performing a (triangular) solve and a vector norm. However, if we know that the covariance $\mathbf{C}^{(s)}$ is diagonal, we can simply invert the diagonal element-wise, as $\mathbf{C}^{(s)^{-1}} = \text{diag}(1/\mathbf{c}^{(s)})$ for diagonal $\mathbf{C}^{(s)}$.

Mixture weights In Eq. (6.8) we impose that the mixture weights are uniformly equal to S^{-1}, where S is the number of mixture components. We can loosen this restriction and learn the mixture weights, but this is a difficult task. Much of this difficulty comes from the fact that changes in the log-likelihood can now be brought about by changing either the component parameters $\boldsymbol{\mu}^{(s)}$ and $\mathbf{C}^{(s)}$, or the mixture weights, which we denote here by $m^{(s)}$.

For each mixture component, the mean parameter $\boldsymbol{\mu}^{(s)}$ and the covariance parameter $\mathbf{C}^{(s)}$ are vector valued and matrix valued, although we restrict the covariance to be diagonal as Eq. (6.8). The mean and covariance of each component are thus determined by $2d_x$ values; however, the mixture weight $m^{(s)}$ is a single value, which has a far larger impact on the likelihood than any single element of the mean or covariance. Therefore, the gradient of the log-likelihood, computed via Eq. (6.9) with backpropagation, with respect to the network parameters $\boldsymbol{\theta}^{(\pi)}$ and $\boldsymbol{\theta}^{(f)}$, will be primarily attributable to the effect these parameters have on the mixture weight, which will result in convergence issues as only the mixture weights will meaningfully change between iterations. Identifiability would also be a problem, as changes in the likelihood could be attributed to changes in either the mean/covariance, or changes to the mixture weights.

The parameters would thus be significantly more difficult to train, and the method given in Alg. 16 would not be sufficient. A potential solution would be to introduce two more networks, each outputting only the mixture weights for the transition and proposal distributions respectively, and then to train these networks as part of the iterative step of Alg. 17, thereby training 4 networks in an alternating scheme. For the sake of brevity, and for the clarity of this chapter, we utilise the equal weighted mixture, which we note can well approximate arbitrarily weighted mixtures with a sufficiently large number of components.

6.4.3 Use of an alternating scheme when estimating network parameters

In order to learn the network parameters $\boldsymbol{\theta}^{(f)}$ and $\boldsymbol{\theta}^{(\pi)}$, Alg. 16 uses an alternating scheme to learn each parameter conditional on the value of the other. Learning one parameter conditional on the other stabilises inference, as the parameters heavily influence each other given that the proposal and transition interact in the weighting step of Alg. 15. Learning in such a manner may lead to identifiability issues, as any change in the weights or particles, and hence in the log-likelihood, can be attributed to either a change in the transition kernel, or a change in the proposal distribution. Furthermore, changes in one distribution should be reflected by changes in the other distribution, as both are tightly linked in interpretation and usage within the filter.

However, if we were to update networks $\text{NN}^{(f)}$ and $\text{NN}^{(\pi)}$ simultaneously, i.e., we replace Alg. 17 with a gradient update of both $\boldsymbol{\theta}^{(f)}$ and $\boldsymbol{\theta}^{(\pi)}$ at the same time, we cannot attribute changes in the likelihood to a specific network, nor can we update each network conditional on the changes made to the other network. In addition, learning both networks at the same time can lead to divergence, as each network may change in such a way that they yield numerically incompatible distributions; in such distributions the transition kernel has near zero probability mass where the proposal has large probability mass, thereby leading to numerically zero weights when computed in line 6 of Alg. 15.

In Alg. 16 we learn each distribution conditional on the other, and therefore do not suffer from an identifiability problem, as any changes are attributable only to the learned distribution. Further, we learn the effects the changes made to each distribution have on the other, hence stabilising the learning, as the distributions do not adapt at the same time. Alternating between learning and conditioning on each distribution allows the distribution to adapt to each other, and mitigates the potential problem of distributions diverging in density.

The order of the alternating scheme does not matter asymptotically, and we obtain nearly identical results when we learn $\boldsymbol{\theta}^{(f)}$ in each and then learn $\boldsymbol{\theta}^{(\pi)}$ in each iteration, instead of learning $\boldsymbol{\theta}^{(\pi)}$ and then $\boldsymbol{\theta}^{(f)}$, however, more iterations of the scheme are required. As we initialise $\boldsymbol{\theta}^{(f)}$ using an estimate based on the bootstrap particle filter, learning $\boldsymbol{\theta}^{(\pi)}$ first in the first iteration, we have a better value for $\boldsymbol{\theta}^{(\pi)}$ when we learn $\boldsymbol{\theta}^{(f)}$ conditional on $\boldsymbol{\theta}^{(\pi)}$. This improves the convergence of the algorithm, and reduces the number of iterations required as both parameters are initially learnt conditioned on reasonable estimates of the other parameter.

6.4.4 Combating likelihood degeneracy when estimating parameters using the particle filter

Our method learns $\boldsymbol{\theta}^{(f)}$ and $\boldsymbol{\theta}^{(\pi)}$ by maximising the estimated log likelihood, $\ell([\boldsymbol{\theta}^{(f)},\boldsymbol{\theta}^{(\pi)}])$, using an alternating scheme. We estimate the log-likelihood using Eq. (6.9), which, in turn, uses the particle weights $w_t^{(k)}$, $t=1,...,T$, $k=1,...,K$, from all iterations of the filter. Therefore, the success of our method hinges on the accurate and stable computation of these weights, which is a challenge in particle filtering, as the weight $w_t^{(k)}$ of a given particle $\mathbf{x}_t^{(k)}$ can be very close to zero. One way to numerically stabilise weight computations is to use log weights [5]. Implementing log weights is a simple change, as we need only rewrite the weight calculation in line 6 of Alg. 15 to use log likelihoods, and rewrite the normalisation to remain on the log scale.

We define $\text{logsumexp}([x_1,x_2,...,x_N]) := \log\left(\sum_{n=1}^{N}\exp(x_n)\right)$.[2] Note that

$$\begin{aligned}\log\left(w_t^{(k)}\right) &= \log\left(\frac{g(\mathbf{y}_t|\mathbf{x}_t^{(k)};\boldsymbol{\theta}^{(g)})f(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)};\boldsymbol{\theta}^{(f)})}{\pi(\mathbf{x}_t|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)},\mathbf{y}_t;\boldsymbol{\theta}^{(\pi)})}\right),\\ &= \log\left(g\left(\mathbf{y}_t|\mathbf{x}_t^{(k)};\boldsymbol{\theta}^{(g)}\right)\right)+\log\left(f\left(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)};\boldsymbol{\theta}^{(f)}\right)\right)-\log\left(\pi\left(\mathbf{x}_t|\mathbf{x}_{t-1}^{\left(a_t^{(k)}\right)},\mathbf{y}_t;\boldsymbol{\theta}^{(\pi)}\right)\right).\end{aligned} \tag{6.10}$$

By writing $p_t^{(k)} = \widetilde{w}_{t-1}^{(k)} w_t^{(k)}$, and noting that $\log\left(p_t^{(k)}\right) = \log\left(\widetilde{w}_{t-1}^{(k)}\right)+\log\left(w_t^{(k)}\right)$, we have

$$\begin{aligned}\bar{w}_t^{(k)} &= \frac{p_t^{(k)}}{\sum_{k=1}^{K}p_t^{(k)}} = \frac{\exp\left(\log\left(p_t^{(k)}\right)\right)}{\sum_{k=1}^{K}\exp\left(\log\left(p_t^{(k)}\right)\right)},\\ \log\left(\bar{w}_t^{(k)}\right) &= \log\left(p_t^{(k)}\right)-\log\left(\sum_{k=1}^{K}\exp\left(\log\left(p_t^{(k)}\right)\right)\right),\\ &= \log\left(p_t^{(k)}\right)-\text{logsumexp}\left(\left[\log\left(p_t^{(k)}\right)\right]_{k=1}^{K}\right),\\ &= \log\left(\widetilde{w}_{t-1}^{(k)}\right)+\log\left(w_t^{(k)}\right)-\text{logsumexp}\left(\left[\log\left(\widetilde{w}_{t-1}^{(k)}\right)+\log\left(w_t^{(k)}\right)\right]_{k=1}^{K}\right),\end{aligned} \tag{6.11}$$

and can construct the log-likelihood estimator following Eq. (6.9) directly using the log weights. The use of log weights significantly improves numerical stability; nevertheless, it does not address the issue of likelihood concentration, nor does it mitigate the problem of learning parameters with initial value having very low likelihood.

[2] The logsumexp can be additionally stabilised by observing that $\text{logsumexp}([x_n]_{n=1}^N) = \max([x_n]_{n=1}^N)+\text{logsumexp}([x_n-\max([x_m]_{m=1}^N)]_{n=1}^N)$, which helps to avoid numerical overflow when evaluating logsumexp. Note that this modification is typically already present in pre-existing implementations of logsumexp such as those present in [98, 97].

State-space models in general suffer from likelihood concentration, where $\ell(\boldsymbol{\theta}|\mathbf{y}_{1:T})$, which we estimate by Eq. (6.9), becomes increasingly concentrated around a single value of $\boldsymbol{\theta}$ as T increases. Likelihood concentration thereby results in vanishingly small likelihoods for parameter values that are not well adapted to the system, leading to the gradient of the parameter values being numerically zero, and the parameter is hence unable to be learnt, without careful initialisation near a value with high likelihood. The issue is therefore that we must choose an initial parameter that has high likelihood, but we assume we do not know the parameter beforehand; these statements contradict each other.

We address likelihood concentration, and hence the problem of learning parameters under random initialisation, using observation batching. We infer our parameters, $\boldsymbol{\theta}^{(f)}$ and $\boldsymbol{\theta}^{(\pi)}$, in Alg. 18 using B increasingly large batches $\mathbf{y}^{(b)}$, $b\in\{1,...,B\}$, of the observation series, where $\mathbf{y}^{(1)}\subseteq\mathbf{y}^{(2)}\subseteq\cdots\subseteq\mathbf{y}^{(B-1)}\subseteq\mathbf{y}^{(B)}$, choosing $\mathbf{y}^{(B)}:=\mathbf{y}$. Warming up the $\boldsymbol{\theta}^{(f)}$ and $\boldsymbol{\theta}^{(\pi)}$ parameters before learning on the entire series mitigates the effect of likelihood concentration for unadapted parameters, and, when combined with log weights, allows our method to be used on long observation series for complex problems.

We initially learn our parameters, $\boldsymbol{\theta}^{(f)}$ and $\boldsymbol{\theta}^{(\pi)}$, on a small subset, $\mathbf{y}^{(1)}$, of the observation series $\mathbf{y}$. Thereby, we obtain a relatively diffuse parameter likelihood function compared to that obtained with a longer observation series. Therefore, we can initialise the parameters of the neural networks, $\mathbf{A}_l$, $\mathbf{b}_l$, $l\in 1,...,L$, randomly, as the likelihood and its gradient will not be numerically zero for unadapted parameters, due to the relatively diffuse parameter likelihood function. Further, under the above method, our estimated f and π distributions incorporate information from the observation series $\mathbf{y}$ in sequence, adapting to one batch of observations $\mathbf{y}^{(b)}$ before taking in more. Sequentially incorporating observation information into our estimation prevents behaviour where the start and end of the series are well represented by the learnt parameters, but the middle is not; this is a common problem when using neural networks to learn between-step dynamics in time series models [121, 122, 123].

6.5 Numerical study

We will now illustrate the performance of our proposed method on two systems. First, we will test our method on a non-linear polynomial system: the Lorenz 96 chaotic oscillator. We will then apply our method to a non-linear non-polynomial system: the Kuramoto oscillator. In both instances we compare our method to the Improved APF [33] and the bootstrap particle filter (BPF) [32], as well as StateMixNN [102], which uses similar techniques but learns only the proposal distribution. The Improved APF utilises measurement information to improve the proposal distribution in a

pre-weighting step, aiming to improve the characteristics of the proposal distribution. Note that all of the methods that we compare against require more information than StateMixNN, as the transition kernel must be known. Finally, we note that all methods are run using the same differentiable particle filter, given in Alg. 15, which does not modify the forward pass behaviour of the SIR particle filter.

6.5.1 Lorenz 96 model

We consider a stochastic version of the Lorenz 96 model [81], a dynamical system known to exhibit chaotic behaviour. We insert additive noise terms to obtain a stochastic system to use for testing, and discretise using the Euler-Maruyama scheme, resulting in the system

$$\begin{aligned} x_{i,t+1} &= x_{i,t} + \Delta t(x_{i-1,t}(x_{i+1,t} - x_{i-2,t}) - x_{i,t} + F) + \sqrt{\Delta t} \cdot v_{i,t+1}, \\ y_{i,t+1} &= x_{i,t+1} + \sqrt{\Delta t} \cdot r_{i,t+1}, \end{aligned} \tag{6.12}$$

for $i \in \{1, \dots, d_x\}$ and $t \in \{0, \dots, T\}$, where we define $x_{-1} := x_{d_x - 1}, x_0 := x_{d_x}$, and $x_{d_x+1} := x_1$, $v_t \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_v)$, $r_t \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_r)$, and F is a forcing constant; we use $F = 8$.

We set $\Delta t = 0.05$ in Eq. (6.12). Unless explicitly stated otherwise, we choose the dimension of the system as $d_x = d_y = 20$, and set $\mathbf{\Sigma}_v = 0.25\mathbf{Id}_{d_x}$ and $\mathbf{\Sigma}_r = 0.1\mathbf{Id}_{d_x}$. We initialise the hidden state at $\mathbf{x}_0$ such that $x_{1,0} = 1$, with all other elements equal to 0. We assess the proposed method in terms of relative improvement in mean square error (MSE), showing the accuracy of the method as a fraction of the MSE obtained with the BPF [32]. The relative improvement in MSE is given by

$$\mathrm{RI}_{\mathrm{MSE}} = \frac{\mathrm{MSE}_{\mathrm{method}}}{\mathrm{MSE}_{\mathrm{baseline}}}, \tag{6.13}$$

where $\mathrm{MSE}_{\mathrm{method}}$ is the MSE of the method we are testing, and $\mathrm{MSE}_{\mathrm{baseline}}$ is the MSE of our baseline method in the same task. The MSE compares the weighted mean of the samples (the estimated state) with the true underlying hidden state. For this model metric, lower is better, as it indicates a lower MSE than the baseline. We use the bootstrap particle filter as our baseline method in all instances. Note that the MSE is not the optimisation target of our method.

We compare our method with the Improved APF [33], to illustrate the performance of a standard improved proposal utilising observation information. Note that there do exist other methods that utilise observation information to attempt to improve the proposal, such as the Gaussian particle filter [124] and the auxiliary particle filter [12], however of those tested the Improved APF performed best on our problem set. Further, we compare our method to PropMixNN [102], the method from which StateMixNN

is extended, which we fix at $S=6$ components, and use the parameters given in the originating chapter. Note that the Improved APF, the BPF, and PropMixNN, all require the state transition model to be known, and therefore cannot be applied to a general system as StateMixNN can. We note that all methods tested are implemented using the same stop-gradient DPF [79] used by StateMixNN, however, no parameters are learned for the Improved APF and the BPF. We compute the MSE for 200 independent runs of the filter, and plot the mean and symmetric 95% intervals.

We test the proposed method using a variable number of mixture components, with $S \in \{1,6,10\}$. All variants utilise the same network architecture, with 3 layers of output sizes $d_1=128, d_2=256, d_3=2Sd_x$ for both networks. For the activation function ρ_l in Eq. (6.6), we set $\rho_{1:2}(x)=\text{relu}(x)=\max(0,x)$, and $\rho_3(x)=x$, with this applying to both the proposal and transition networks. We train StateMixNN using the ADAM optimiser [38], using a fixed learning rate of $3\cdot 10^{-3}$, and setting the parameters of Alg. 16 to $B=\lceil T/5 \rceil, J=50, A=20$. We train the method using a series of observations distinct from those on which we test StateMixNN; however, all series are instances of the Lorenz 96 system.

Variable number of particles. We test StateMixNN for a variable number of particles K, with $K \in \{30,50,100,200\}$. We use a fixed series length $T=100$. We present the results in Fig. 6.1, which shows that StateMixNN outperforms the BPF for all given values of K, obtaining at most 0.9 times the MSE of the BPF, and typically less than 0.75 the MSE. StateMixNN with $S=10$ components suffers with few particles, performing worse than the other parametrisations of StateMixNN, as few samples are taken from each component, therefore the gradient estimates are less reliable, as the gradients relating to each component are dependent on only a few evaluations. In the case of $S=6$ there are more evaluations of each component than in the case of $S=10$, which allows for sufficient information to learn the transition dynamics. Our method outperforms the Improved APF at all tested numbers of particles, by an increasingly large margin as the number of particles increases.

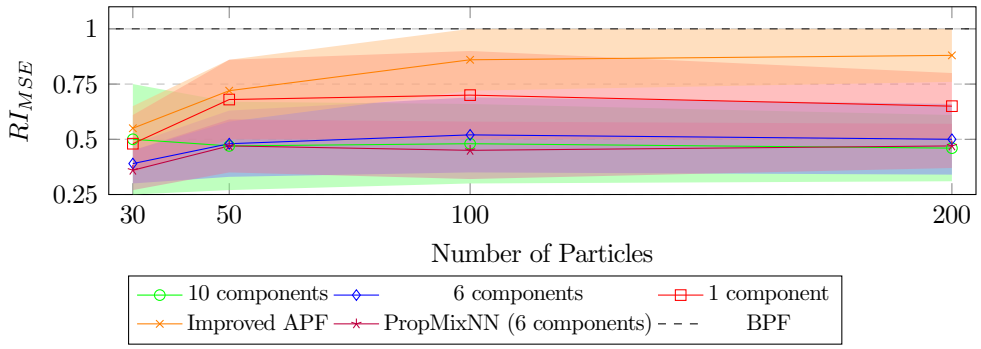


Figure 6.1: Comparison of StateMixNN with the BPF, Improved APF, and PropMixNN over variable numbers of particles. The lines denote mean performance, with bands denoting symmetric 95% intervals.

Variable series length. Next, we test StateMixNN with a variable series length T, with $T \in \{30,60,100,200,500\}$. In this case we fix the number of particles $K = 100$. We show in Fig. 6.2 that the proposed method obtains lower values of MSE than the BPF and Improved APF for all given values of T. The $S = 10$ component method slightly outperforms the $S = 6$ component method, which significantly outperforms the $S = 1$ component method.

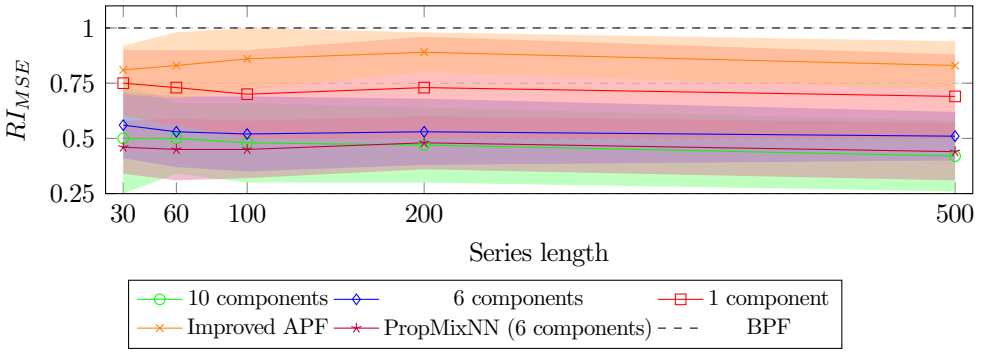


Figure 6.2: Comparison of StateMixNN with the BPF, Improved APF, and PropMixNN over variable series length. The lines denote mean performance, with bands denoting symmetric 95% intervals.

Variable state noise. We now test StateMixNN for a variable state noise $\mathbf{\Sigma}_v = \sigma_v^2 \mathbf{Id}_{20}$, with $\sigma_v^2 \in \{0.05,0.1,0.25,0.5,1\}$. In this case, we fix the number of particles $K = 100$ and the series length $T = 100$. Fig. 6.3 shows that StateMixNN is superior to the BPF for all given values of σ_v^2. The improvement in accuracy is lesser for small noise variances, as small perturbations give a concentrated distribution of the state values at the next time step. However, the performance improves for larger values of σ_v^2. This is due to the chaotic behaviour of the system, which leads to multimodal distributions for the next state, as the state follows one of several diverging paths. The mixture in the proposed method captures this behaviour, with components representing different

modes, thereby outperforming both the BPF and the Improved APF.

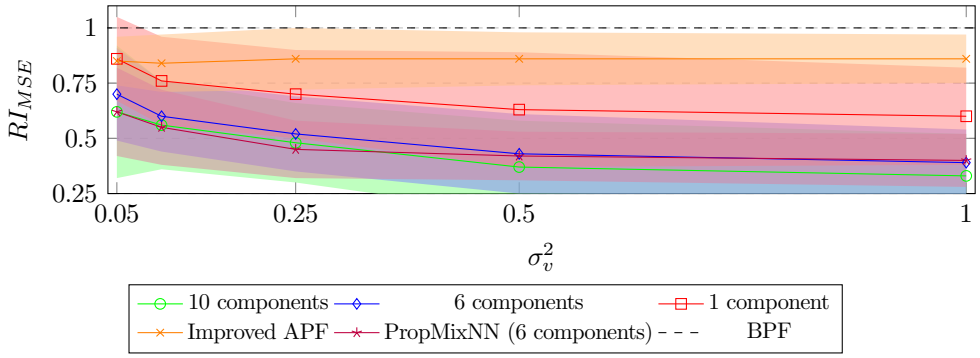


Figure 6.3: Comparison of StateMixNN with the BPF, Improved APF, and PropMixNN over variable state noise magnitude. The lines denote mean performance, with bands denoting symmetric 95% intervals.

Variable system dimension. Finally, we test StateMixNN over variable state and observation dimension $d_x = d_y$, with $d_x \in \{5,10,20,40,60\}$. We fix the number of particles $K = 100$, and the series length $T = 100$. Fig. 6.4 shows that StateMixNN is superior to the BPF for all given values of d_x. We observe that the margin of out-performance decreases as the state dimension increases, but seems to stabilise after $d_x = 40$. This is due to the static number of iterations used in training, as the state-space is easier to learn for smaller d_x, and therefore requires fewer iterations to achieve the same level of performance. We observe that StateMixNN performs well at all tested values of d_x, and does not rapidly deteriorate in performance when increasing d_x without changing the training regime. Furthermore, the filters with a larger number of components in the learned distributions outperform those with a smaller number of components at all times, displaying the mixture distributions ability to approximate complex systems.

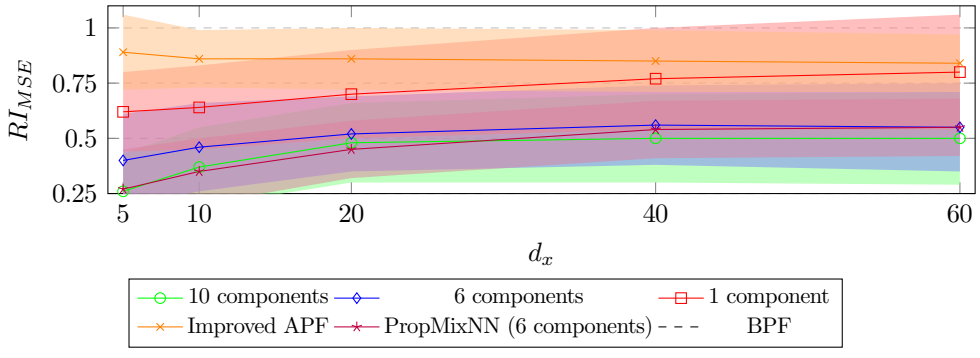


Figure 6.4: Comparison of StateMixNN with the BPF, Improved APF, and PropMixNN over variable system dimension. The lines denote mean performance, with bands denoting symmetric 95% intervals.

6.5.2 Kuramoto oscillator

The Kuramoto oscillator [101] is a mathematical model that describes the behavior of a system of d_x phase-coupled oscillators. The model is described by

$$\frac{\mathrm{d}\theta_i}{\mathrm{d}t} = \omega_i + d_x^{-1}\sum_{j=1}^{d_x} C\sin(\theta_j - \theta_i), \tag{6.14}$$

where θ_i denotes the phase of the ith oscillator, and $C \in \mathbb{R}$ is the coupling constant between oscillators. This does not restrict $\boldsymbol{\theta}$ however, which will, in general, diverge to an infinity as $t \to \infty$. To address this, we transform Eq. (6.14) by introducing derived parameters $R \in \mathbb{R}^+$ and $\phi \in [-\pi,\pi]$ such that

$$\begin{aligned} R\exp(\sqrt{-1}\phi) &= d_x^{-1}\sum_{j=1}^{d_x}\exp\left(\sqrt{-1}\theta_j\right), \\ \frac{\mathrm{d}\theta_i}{\mathrm{d}t} &= \omega_i + CR\sin(\phi - \theta_i), \end{aligned} \tag{6.15}$$

which restricts $\boldsymbol{\theta} \in [-\pi,\pi]^{d_x}$. We insert additive Gaussian noise to Eq. (6.15), and discretise using the Euler-Maruyama scheme, yielding the NLSSM

$$\begin{aligned} R\exp(\sqrt{-1}\phi) &= d_x^{-1}\sum_{j=1}^{d_x}\exp\left(\sqrt{-1}x_{j,t}\right), \\ x_{i,t+1} &= x_{i,t} + \Delta t(\omega_i + CR\sin(\phi - x_i)) + \sqrt{\Delta t}\cdot v_{i,t+1}, \\ y_{i,t+1} &= x_{i,t+1} + \sqrt{\Delta t}\cdot r_{i,t+1}, \end{aligned} \tag{6.16}$$

for $i \in \{1,\ldots,d_x\}$. We choose $d_x = 20$, and $C = 0.8$. We set $\boldsymbol{\Sigma}_v = \sigma_v^2\mathbf{Id}_{d_x}, \boldsymbol{\Sigma}_r = \sigma_r^2\mathbf{Id}_{d_x}$, with $\sigma_v = 0.1, \sigma_r = 0.005$ unless stated otherwise, which tests the performance of our method in the case that the observation is much more informative than the state, where the standard BPF is known to suffer. We discretise this model with a time step of $\Delta t = 0.05$. We sample $\omega_i \sim \mathcal{N}(0.5, 0.5^2)$, and $\mathbf{x}_{i,0} \sim U(-\pi,\pi)$. We run the system until $t = 10$, and then begin collecting observations.

Variable series length. We test StateMixNN with a variable series length T, with $T \in \{30,60,100,200,500\}$ on the Kuramoto oscillator In this case we fix the number of particles $K = 100$. We show in Fig. 6.5 that the proposed method obtains lower values of MSE than the BPF and Improved APF for all given values of T. The $S = 10$ component method outperforms the $S = 6$ component method, which in turn outperforms the $S = 1$ component method.

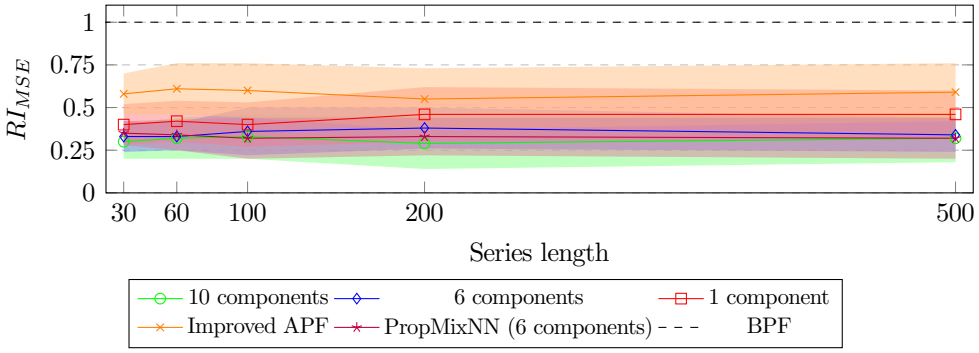


Figure 6.5: Comparison of StateMixNN with the BPF, Improved APF, and PropMixNN over variable series length on the Kuramoto oscillator. The lines denote mean performance, with bands denoting symmetric 95% intervals.

Variable number of particles. We test StateMixNN for a variable number of particles K, with $K \in \{30,50,100,200\}$. We use a fixed series length $T = 100$. We present the results in Fig. 6.6, which shows that StateMixNN outperforms the BPF and Improved APF for all given values of K. StateMixNN with $S = 10$ components suffers with few particles, performing worse than the other parametrisations of StateMixNN, as few samples are taken from each component, therefore the gradient estimates are more dispersed in the $S = 10$ case, making training less reliable.

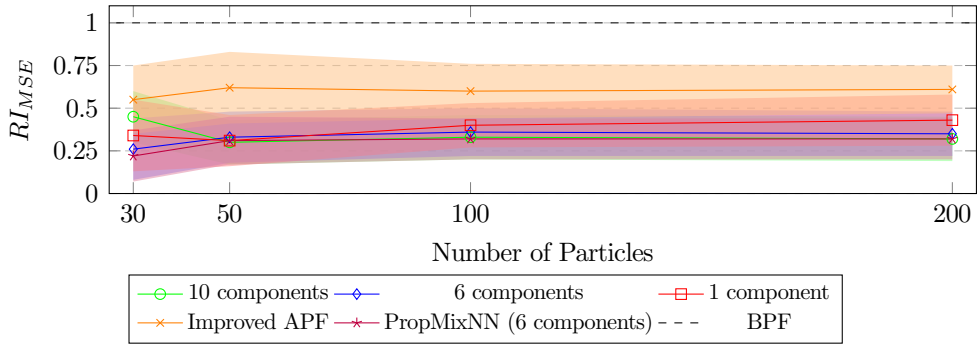


Figure 6.6: Comparison of StateMixNN with the BPF, Improved APF, and PropMixNN over variable number of particles on the Kuramoto oscillator. The lines denote mean performance, with bands denoting symmetric 95% intervals.

Variable state noise. We now test StateMixNN on the Kuramoto oscillator with variable state noise $\mathbf{\Sigma}_v = \sigma_v^2 \mathbf{Id}_{d_x}$, with $\sigma_v \in \{0.025,0.05,0.1,0.25,0.5\}$. In this case, we fix the number of particles $K = 100$ and the series length $T = 100$.

We observe in Fig. 6.7 that the performance remains consistent over state noise magnitudes when compared to the performance of the bootstrap particle filter. We observe an apparent decrease in performance for very low state noise magnitude for all tested methods, which is explained by the bootstrap particle filter performing better the closer σ_v is to σ_r.

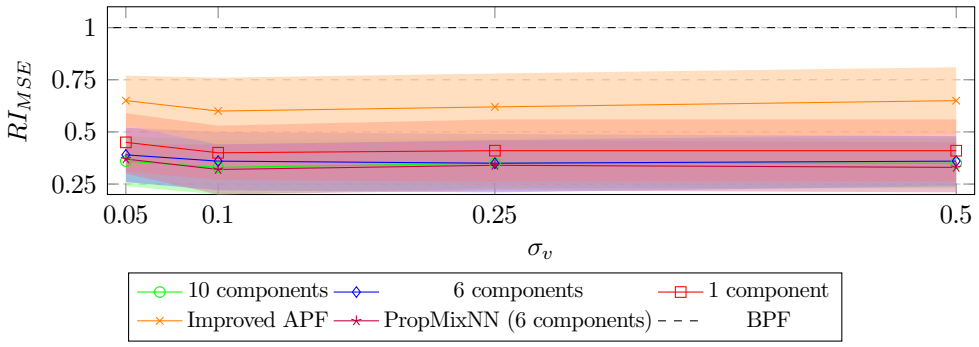


Figure 6.7: Comparison of StateMixNN with the BPF, Improved APF, and PropMixNN over variable state noise magnitude. The lines denote mean performance, with bands denoting symmetric 95% intervals.

Variable system dimension. Finally, we test StateMixNN over variable state and observation dimension $d_x = d_y$, with $d_x \in \{5, 10, 20, 40, 60\}$. We fix the number of particles $K = 100$, and the series length $T = 100$. We present the results in Fig. 6.8.

We observe that StateMixNN is superior to the BPF for all given values of d_x. Furthermore, the margin of out-performance is approximately constant after $d_x = 20$, with StateMixNN outperforming standard methods by a larger margin for smaller values of d_x. This is likely due both to a smaller number of parameters being learnt, and less complex transition and proposal distributions being required due to the reduced dimensionality of the problem. We observe that StateMixNN performs well at all tested values of d_x, and does not rapidly deteriorate in performance when increasing d_x without changing the training regime. Furthermore, the filters with a larger number of components in the learned distributions outperform those with a smaller number of components at all times, displaying the mixture distributions ability to approximate complex systems.

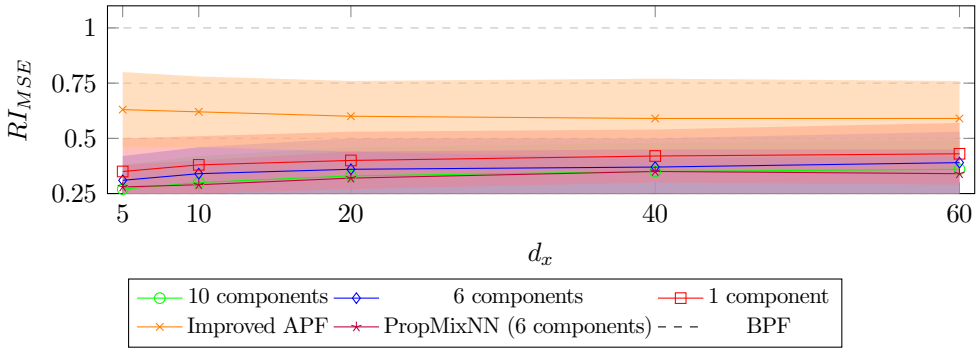


Figure 6.8: Comparison of StateMixNN with the BPF, Improved APF, and PropMixNN over variable system dimension. The lines denote mean performance, with bands denoting symmetric 95% intervals.

6.6 Conclusion

This chapter proposes a novel method, called StateMixNN, which simultaneously learns the transition and proposal distributions of a particle filter. We utilise a pair of multivariate Gaussian mixture distributions to approximate the transition and proposal distributions, with the means and covariances of the mixands given by the output of a dense neural network. StateMixNN uses a standard neural network architecture to produce its results, and could potentially be extended or modified to use a different architecture, such as a graph neural network to estimate state connectivity, or a recurrent neural network for non-Markovian systems.

We present numerical results for a stochastic Lorenz 96 model, which has highly chaotic behaviour, and for the Kuramoto oscillator, which has a restricted state-space and cannot be expressed using polynomial terms. We observe that our method outperforms the bootstrap particle filter, a standard method, as well as the improved auxiliary particle filter, a state-of-the-art method for improving the proposal distribution, both of which require more information about the underlying system than StateMixNN. Furthermore, our method here performs similarly to PropMixNN, a method that uses a similar neural proposal, but requires the state model to be known. The proposed method does not require knowledge of the hidden state, as we optimise the observation likelihood, which requires only the observations to be known. The ability of StateMixNN to operate without knowledge of the hidden state dynamics makes it particularly well-suited for real-world applications where the underlying model is unknown.

Chapter 7

Conclusions and future work

7.1 Conclusion and contributions

In this thesis, we have proposed several methods which share the theme of parameter estimation in linear and non-linear state-space models, and proposes a method to address the challenge of proposal design in particle filtering.

First, in Chapter 3, we proposed a method to perform fully Bayesian inference of the sparsity structure of the transition matrix of a linear-Gaussian state-space model. The method we propose has strong theoretical guarantees, and provides a reliable and efficient technique for obtaining sparse samples of the parameter posterior. Our method utilises reversible jump Markov chain Monte Carlo, in conjunction with a novel model space construction, to jointly sample the sparsity structure and the parameter value of the state transition matrix. We showed that it has excellent numerical characteristics, and compares favourably to state-of-the-art techniques for sparse parameter estimation in LGSSMs.

Next, in Chapter 4, we developed an algorithm that promotes sparse clusters in the transition matrix of a linear-Gaussian state-space model. The method proposed is extensible, and inherits the guarantees of the estimation subroutine used. By iteratively imposing cluster promoting constraints, our method recovers the cluster structure present in many real-world systems, whilst leveraging existing estimation techniques to perform parameter estimation. We have shown that the proposed method outperforms existing estimation schemes for structure recovery in sparse clustered systems, yielding superior parameter estimates that exhibit the correct clustered structure.

Then, in Chapter 5, we proposed a methodology for estimating the transition kernel of a general state-space model when the form of the transition kernel is unspecified. Our method estimates the transition kernel using a sparse polynomial basis, which we encode as a pair of matrices representing the degree and coefficient of the polynomial

terms. We estimate the coefficient matrix under a sparsity promoting penalty, which we apply using the proximity operator, resulting in a scheme that is more efficient than standard gradient descent, and which can be extended to use other convex penalties without the requirement for (sub)differentiability. We demonstrate that our method is capable of recovering the transition model in several well known polynomial systems, and furthermore, can well estimate the transition kernel in non-polynomial systems.

Finally, in Chapter 6, we proposed an algorithm for jointly estimating the transition kernel and the optimal proposal distribution for a general state-space model within a particle filtering framework. This method allows for filtering to be performed within a general class of models, where the hidden state dynamics are unknown, but the observation dynamics are assumed known. We show that our method yields superior recovery of the hidden state sequence compared to several improved proposals, whilst requiring less information to be known about the system.

7.2 Potential future work and extensions

We now discuss several potential extensions to the work proposed in this thesis, and sketch initial ideas for their exploration.

7.2.1 Fully Bayesian estimation of a sparse transition kernel in general state-space models

Given the fully Bayesian sparsity recovery for linear-Gaussian state-space models proposed in Chapter 3, one might wish to apply the same methodology to general state-space models following the polynomial approach proposed in Chapter 5. However, the concentration of the likelihood function, coupled with the high dimension of the parameter space, make a direct extension of SPaRJ unlikely to succeed, as the MCMC scheme utilised will be inefficient at exploring the parameter space. Instead, we can use the differentiable particle filter to design a Hamiltonian Monte Carlo styled proposal for the parameter sampling, and utilise a similar model jump kernel as in Chapter 3 to select polynomial basis terms in the model sampling.

However, due to the increased dimension of the parameter space, as well as the requirement to run a particle filter several times to generate each sample, which is required due to the leapfrog steps in HMC, the method would be orders of magnitude more expensive than GraphGrad, as many samples will be required to explore the model space, with each sample being relatively expensive to generate. Therefore much of the design of such a method would therefore be in reducing the computational cost. An initial avenue for exploration regarding designing better model jump kernels would be to take inspira-

tion from methods such as [125], automatically tuning a runtime-accuracy trade-off in the intermediate runs of the particle filter, or by parallel exploration of model moves.

7.2.2 Extended sparse library regression

In Chapter 5, we noted that our method could be interpreted as a library regression, similar to [77]. This naturally poses an extension to our proposed method: what if we extend the library of terms outside of polynomials. We chose the polynomial basis for three reasons. First, the polynomial basis can be thought of similarly to the Taylor expansion, which is well known, and works well for smoothly changing functions close to the point of expansion. Second, a polynomial in a known number of variables can be easily written as a matrix expression, and therefore quickly and efficiently evaluated as a vectorised expression, with the coefficients stored in a statically-sized format that easily admits the use of automatic differentiation. Third, the polynomial expansion is rather objective, and does not require terms to be included and excluded based on opinion, such as in many library regression methods.

Extending GraphGrad to a library regression is conceptually relatively simple; estimate an additional library term in addition to the existing polynomial basis. For example, non-polynomial terms could be separated into their own matrices, and therefore could be evaluated separately from the polynomial terms. However, this approach is limited by the requirement to specify the term library and interactions therein beforehand; incorporating binary interactions between terms, such as ratios of polynomial terms, would require a prohibitively large library.

7.2.3 Symbolic regression for transition terms

A potential further extension of Chapter 5, beyond a fixed library of terms estimated simultaneously, is to estimate the form of the transition kernel via symbolic regression. This would involve estimating the term currently approximated by a polynomial as the output of a symbolic regression. The symbolic search method of [126] could be utilised to perform symbolic regression, with a modification to the objective such that the method aims to minimise the negative log-likelihood of the estimated parameters, rather than the RMSE as is common in symbolic regression.

This would improve over a method that estimates all terms simultaneously in several ways. Firstly, the terms would be better estimated, as there is more statistical information available per coefficient estimated. Second, the model is more interpretable, as terms are explicitly included or excluded, rather than potentially set to a almost-but-not-quite zero value in estimation. This also eases the requirement for sparsity

promoting parameter estimation by estimating sparsity via an auxiliary process, similar to that in Chapter 3. Finally, this would allow for binary functions (e.g. products and ratios of polynomial terms) of arbitrary terms to be more easily incorporated, as these would not need to be explicitly encoded in the learnable parameters, but instead dynamically learnt through the symbolic search.

7.2.4 Concluding remarks

In this thesis, we have proposed several modelling approaches that allow for the estimation of state-space model parameters whilst promoting sparsity, thereby improving the interpretability of the resulting model. We have also proposed two methods for the automatic estimation of the transition kernel in a general state-space model, with one method being a sparse parametric approach, and one method being a dense deep learning approach.

Furthermore, the methods we have proposed here have several immediately apparent avenues for extension, both in broadening their scope and in improving their efficiency. It is likely that parameter estimation in state-space models will be an area of active research going forward, and as interpretable models give explainable results, we believe that the methods proposed here will be useful to future research.

Bibliography

[1] J. Durbin and S. J. Koopman, *Time Series Analysis by State Space Methods*. Oxford university press, 2012.

[2] D. Barber and A. T. Cemgil, "Graphical Models for Time-Series," *IEEE Signal Processing Magazine*, vol. 27, no. 6, pp. 18–28, Nov. 2010.

[3] S. Bisgaard and M. Kulahci, *Time Series Analysis and Forecasting by Example*. Hoboken, N.J: Wiley, 2011.

[4] L. Medsker and L. C. Jain, *Recurrent Neural Networks: Design and Applications*. CRC press, 1999.

[5] S. Särkkä, *Bayesian Filtering and Smoothing*. Cambridge, United Kingdom: Cambridge University Press, 2013.

[6] K. Newman, R. King, V. Elvira, P. de Valpine, R. S. McCrea, and B. J. Morgan, "State-space models for ecological time-series data: Practical model-fitting," *Methods in Ecology and Evolution*, vol. 14, no. 1, pp. 26–42, 2023.

[7] P. Bauer, A. Thorpe, and G. Brunet, "The Quiet Revolution of Numerical Weather Prediction," *Nature*, vol. 525, no. 7567, pp. 47–55, 2015.

[8] A. Virbickaitė, H. F. Lopes, M. C. Ausín, and P. Galeano, "Particle Learning for Bayesian Semi-Parametric Stochastic Volatility Model," *Econometric Reviews*, 2019.

[9] N. Kantas, A. Doucet, S. S. Singh, J. Maciejowski, and N. Chopin, "On Particle Methods for Parameter Estimation in State-Space Models," *Statistical Science*, vol. 30, no. 3, pp. 328–351, Aug. 2015. [Online]. Available: http://projecteuclid.org/euclid.ss/1439220716

[10] C. Andrieu, A. Doucet, and R. Holenstein, "Particle Markov Chain Monte Carlo Methods," *Journal of the Royal Statistical Society: Series B (Statistical*

Methodology), vol. 72, no. 3, pp. 269–342, Jun. 2010. [Online]. Available: http://doi.wiley.com/10.1111/j.1467-9868.2009.00736.x

[11] E. E. Holmes, "Derivation of an EM algorithm for constrained and unconstrained multivariate autoregressive state-space (MARSS) models," *arXiv preprint arXiv:1302.3919*, 2013.

[12] M. K. Pitt and N. Shephard, "Filtering via Simulation: Auxiliary Particle Filters," *Journal of the American Statistical Association*, vol. 94, no. 446, pp. 590–599, Jun. 1999. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01621459.1999.10474153

[13] F. Lindsten, M. I. Jordan, and T. B. Schon, "Particle Gibbs with Ancestor Sampling," *Journal of Machine Learning Research*, vol. 15, pp. 2145–2184, 2014.

[14] D. J. Watts and S. H. Strogatz, "Collective Dynamics of 'Small-World' Networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[15] V. Elvira, E. Chouzenoux, J. Cerda, and G. Camps-Valls, "Graphs in State-Space Models for Granger Causality in Climate Science," *arXiv preprint arXiv:2307.10703*, 2023.

[16] V. Elvira and E. Chouzenoux, "Graphical Inference in Linear-Gaussian State-Space Models," *IEEE Transactions on Signal Processing*, vol. 70, pp. 4757–4771, 2022. [Online]. Available: http://arxiv.org/abs/2209.09969

[17] A. Gelman, J. Carlin, H. Stern, D. Dunson, A. Vehtari, and D. Rubin, *Bayesian Data Analysis, Third Edition*, ser. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis, 2013.

[18] S. T. Tokdar and R. E. Kass, "Importance Sampling: A Review," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 1, pp. 54–60, 2010.

[19] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. A. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A Probabilistic Programming Language." *Grantee Submission*, vol. 76, no. 1, pp. 1–32, 2017.

[20] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational Inference: A Review for Statisticians," *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, Apr. 2017. [Online]. Available: http://arxiv.org/abs/1601.00670

[21] V. Elvira and L. Martino, "Advances in Importance Sampling," *Wiley StatsRef: Statistics Reference Online*, pp. 1–22, 2021.

[22] V. Elvira, L. Martino, D. Luengo, and M. F. Bugallo, "Generalized Multiple Importance Sampling," *Statistical Science*, vol. 34, no. 1, pp. 129–155, Feb. 2019. [Online]. Available: https://projecteuclid.org/euclid.ss/1555056039

[23] M. F. Bugallo, V. Elvira, L. Martino, D. Luengo, J. Miguez, and P. M. Djuric, "Adaptive Importance Sampling: The Past, the Present, and the Future," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 60–79, Jul. 2017.

[24] R. M. Neal, "MCMC Using Hamiltonian Dynamics," *arXiv:1206.1901 [physics, stat]*, vol. 2, no. 11, p. 2, Jun. 2012. [Online]. Available: http://arxiv.org/abs/1206.1901

[25] P. J. Green, "Reversible Jump Markov Chain Monte Carlo Computation and Bayesian Model Determination," *Biometrika*, vol. 82, no. 4, pp. 711–732, Dec. 1995. [Online]. Available: https://www.jstor.org/stable/2337340

[26] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[27] M. Bédard, "Optimal Acceptance Rates for Metropolis Algorithms: Moving beyond 0.234," *Stochastic Processes and their Applications*, vol. 118, no. 12, pp. 2198–2222, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304414907002177

[28] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, Jan. 1996. [Online]. Available: http://doi.wiley.com/10.1111/j.2517-6161.1996.tb02080.x

[29] T. Park and G. Casella, "The Bayesian Lasso," *Journal of the American Statistical Association*, vol. 103, no. 482, pp. 681–686, 2008.

[30] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[31] A. Doucet, A. M. Johansen *et al.*, "A Tutorial on Particle Filtering and Smoothing: Fifteen Years Later," *Handbook of nonlinear filtering*, vol. 12, no. 656-704, p. 3, 2009.

[32] N. Gordon, D. Salmond, and A. Smith, "Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation," *IEE Proceedings F Radar and Signal Processing*, vol. 140, no. 2, p. 107, 1993. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/ip-f-2.1993.0015

[33] V. Elvira, L. Martino, M. F. Bugallo, and P. M. Djuric, "In Search for Improved Auxiliary Particle Filters," in *2018 26th European Signal Processing Conference (EUSIPCO)*, IEEE. Rome: IEEE, Sep. 2018, pp. 1637–1641. [Online]. Available: https://ieeexplore.ieee.org/document/8553361/

[34] E. Chouzenoux and V. Elvira, "Sparse Graphical Linear Dynamical Systems," *Journal of Machine Learning Research*, vol. 25, no. 223, pp. 1–53, 2024.

[35] B. Cox and V. Elvira, "Sparse Bayesian Estimation of Parameters in Linear-Gaussian State-Space Models," *IEEE Transactions on Signal Processing*, vol. 71, pp. 1922–1937, 2023. [Online]. Available: https://doi.org/10.1109/TSP.2023.3278867

[36] P. Karkus, D. Hsu, and W. S. Lee, "Particle Filter Networks with Application to Visual Localization," in *Proceedings of the Conference on Robot Learning*, PMLR. PMLR, 2018, pp. 169–178.

[37] X. Chen and Y. Li, "An Overview of Differentiable Particle Filters for Data-Adaptive Sequential Bayesian Inference," *arXiv preprint arXiv:2302.09639*, 2023.

[38] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[39] S. C. Scott, L. D. David, and A. S. Michael, "Atomic Decomposition by Basis Pursuit," *SIAM journal on scientific computing*, vol. 20, no. 1, pp. 33–61, 1998.

[40] H. Mohimani, M. Babaie-Zadeh, and C. Jutten, "A Fast Approach for Overcomplete Sparse Decomposition Based on Smoothed 'L0-norm'," *IEEE Transactions on Signal Processing*, vol. 57, no. 1, pp. 289–301, 2008.

[41] H. Zayyani, M. Babaie-Zadeh, and C. Jutten, "An Iterative Bayesian Algorithm for Sparse Component Analysis in Presence of Noise," *IEEE Transactions on Signal Processing*, vol. 57, no. 11, pp. 4378–4390, 2009.

[42] S. Ji, Y. Xue, and L. Carin, "Bayesian Compressive Sensing," *IEEE Transactions on signal processing*, vol. 56, no. 6, pp. 2346–2356, 2008.

[43] P. Schniter, L. C. Potter, and J. Ziniel, "Fast Bayesian Matching Pursuit," in *2008 Information Theory and Applications Workshop*. IEEE, 2008, pp. 326–333.

[44] H. Zayyani, M. Babaie-Zadeh, and C. Jutten, "Bayesian Pursuit Algorithm for Sparse Representation," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2009, pp. 1549–1552.

[45] M. Korki, J. Zhang, C. Zhang, and H. Zayyani, "Iterative Bayesian Reconstruction of Non-IID Block-Sparse Signals," *IEEE Transactions on Signal Processing*, vol. 64, no. 13, pp. 3297–3307, 2016.

[46] F. Llorente, L. Martino, D. Delgado, and J. Lopez-Santiago, "Marginal Likelihood Computation for Model Selection and Hypothesis Testing: An Extensive Review," *arXiv preprint arXiv:2005.08334*, 2020.

[47] F. Llorente, L. Martino, E. Curbelo, J. López-Santiago, and D. Delgado, "On the Safe Use of Prior Densities for Bayesian Model Selection," *Wiley Interdisciplinary Reviews: Computational Statistics*, p. e1595, 2022.

[48] L. Martino, J. Read, V. Elvira, and F. Louzada, "Cooperative Parallel Particle Filters for Online Model Selection and Applications to Urban Mobility," *Digital Signal Processing*, vol. 60, pp. 172–185, Jan. 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1051200416301610

[49] C. M. Carvalho, N. G. Polson, and J. G. Scott, "The Horseshoe Estimator for Sparse Signals," *Biometrika*, vol. 97, no. 2, pp. 465–480, 2010.

[50] E. Chouzenoux and V. Elvira, "Graphem: EM Algorithm for Blind Kalman Filtering under Graphical Sparsity Constraints," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE. Barcelona, Spain: IEEE, May 2020, pp. 5840–5844. [Online]. Available: https://ieeexplore.ieee.org/document/9053646/

[51] C. W. J. Granger, "Investigating Causal Relations by Econometric Models and Cross-Spectral Methods," *Econometrica : journal of the Econometric Society*, vol. 37, no. 3, pp. 424–438, 1969. [Online]. Available: http://www.jstor.org/stable/1912791

[52] B. Cox and V. Elvira, "Parameter Estimation in Sparse Linear-Gaussian State-Space Models via Reversible Jump Markov Chain Monte Carlo," in *Proceedings of the 30th European Signal Processing Conference*. IEEE, 2022.

[53] A. Pirayre, C. Couprie, L. Duval, and J. Pesquet, "BRANE Clust: Cluster-assisted Gene Regulatory Network Inference Refinement," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 15, no. 3, pp. 850–860, May 2018.

[54] D. Luengo, G. Rios-Munoz, V. Elvira, C. Sanchez, and A. Artes-Rodriguez, "Hierarchical Algorithms for Causality Retrieval in Atrial Fibrillation Intracavitary Electrograms," *IEEE Journal of Biomedical and Health Informatics*, vol. 23, no. 1, pp. 143–155, Jan. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8290986/

[55] C. Ravazzi, R. Tempo, and F. Dabbene, "Learning Influence Structure in Sparse Social Networks," *IEEE Transactions on Control of Network Systems*, vol. PP, pp. 1–1, Dec. 2017.

[56] J. Richiardi, S. Achard, B. Horst, and a. D. V. D. Ville, "Machine Learning with Brain Graphs," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 58–70, 2013.

[57] M. Pagel and A. Meade, "Bayesian Analysis of Correlated Evolution of Discrete Characters by Reversible-Jump Markov Chain Monte Carlo," *The American Naturalist*, vol. 167, no. 6, pp. 808–825, 2006.

[58] Sylvia. Richardson and P. J. Green, "On Bayesian Analysis of Mixtures with an Unknown Number of Components (with Discussion)," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 59, no. 4, pp. 731–792, 1997. [Online]. Available: http://doi.wiley.com/10.1111/1467-9868.00095

[59] O. Cappé, C. P. Robert, and T. Rydén, "Reversible Jump, Birth-and-Death and More General Continuous Time Markov Chain Monte Carlo Samplers," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 65, no. 3, pp. 679–700, 2003.

[60] C. Andrieu and A. Doucet, "Joint Bayesian Model Selection and Estimation of Noisy Sinusoids via Reversible Jump MCMC," *IEEE Transactions on Signal Processing*, vol. 47, no. 10, pp. 2667–2676, 1999.

[61] J. Vermaak, C. Andrieu, A. Doucet, and SJ. Godsill, "Reversible Jump Markov Chain Monte Carlo Strategies for Bayesian Model Selection in Autoregressive Processes," *Journal of Time Series Analysis*, vol. 25, no. 6, pp. 785–809, 2004.

[62] P. Bunch, J. Murphy, and S. Godsill, "Bayesian Learning of Degenerate Linear Gaussian State Space Models Using Markov Chain Monte Carlo," *IEEE Transactions on Signal Processing*, vol. 64, no. 16, pp. 4100–4112, 2016.

[63] C. Robert and G. Casella, *Monte Carlo Statistical Methods*. Springer Science & Business Media, 2013.

[64] D. Vats, J. M. Flegal, and G. L. Jones, "Multivariate Output Analysis for Markov Chain Monte Carlo," *Biometrika*, vol. 106, no. 2, pp. 321–337, Apr. 2019. [Online]. Available: https://doi.org/10.1093/biomet/asz002

[65] A. Gelman, W. R. Gilks, and G. O. Roberts, "Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms," *The Annals of Applied Probability*, vol. 7, no. 1, pp. 110–120, 1997. [Online]. Available: https://doi.org/10.1214/aoap/1034625254

[66] P. Gagnon, M. Bédard, and A. Desgagné, "Weak Convergence and Optimal Tuning of the Reversible Jump Algorithm," *Mathematics and Computers in Simulation*, vol. 161, pp. 32–51, 2019.

[67] K. Bello, B. Aragam, and P. Ravikumar, "DAGMA: Learning Dags via m-Matrices and a Log-Determinant Acyclicity Characterization," *arXiv preprint arXiv:2209.08037*, 2022.

[68] F. Mezzadri, "How to Generate Random Matrices from the Classical Compact Groups," *Notices of the American Mathematical Society*, vol. 54, no. 5, pp. 592–604, 2007.

[69] "United States Environmental Protection Agency Average Daily Temperature Archive,"United States Environmental Protection Agency. [Online]. Available: https://academic.udayton.edu/kissock/http/Weather/default.htm

[70] S. P. Brooks and P. Giudici, "Convergence Assessment for Reversible Jump MCMC Simulations," in *Bayesian Statistics 6: Proceedings of the Sixth Valencia International Meeting June 6-10, 1998*. Oxford University Press, 08 1999. [Online]. Available: https://doi.org/10.1093/oso/9780198504856.003.0033

[71] E. Chouzenoux and V. Elvira, "GraphIT: Iterative Reweighted $_1$ Algorithm for Sparse Graph Inference in State-Space Models," in *ICASSP 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2023, pp. 1–5.

[72] M. Eichler, "Graphical Modelling of Multivariate Time Series," *Probability Theory and Related Fields*, vol. 153, no. 1, pp. 233–268, 2012.

[73] E. Hartuv and R. Shamir, "A Clustering Algorithm Based on Graph Connectivity," *Information Processing Letters*, vol. 76, no. 4, pp. 175–181, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020019000001423

[74] M. Girvan and M. E. Newman, "Community Structure in Social and Biological Networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[75] B. Cox, E. Chouzenoux, and V. Elvira, "Learning a sparse polynomial approximation to the transition function of general state-space models," in *ICASSP 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2025.

[76] ——, "GraphGrad: Efficient Estimation of Sparse Polynomial Representations for General State-Space Models," *IEEE Transactions on Signal Processing*, vol. 73, pp. 1562–1576, 2025.

[77] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the national academy of sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[78] A. Corenflos, J. Thornton, A. Doucet, and G. Deligiannidis, "Differentiable Particle Filtering via Entropy-Regularized Optimal Transport," *arXiv:2102.07850 [cs, stat]*, Feb. 2021. [Online]. Available: http://arxiv.org/abs/2102.07850

[79] A. Ścibior and F. Wood, "Differentiable Particle Filtering without Modifying the Forward Pass," *arXiv preprint arXiv:2106.10314*, 2021.

[80] E. N. Lorenz, "Deterministic Nonperiodic Flow," *Journal of atmospheric sciences*, vol. 20, no. 2, pp. 130–141, 1963.

[81] ——, "Predictability: A Problem Partly Solved," in *Proc. Seminar on Predictability*, vol. 1, ECMWF. Reading, 1996.

[82] W. Li, X. Chen, W. Wang, V. Elvira, and Y. Li, "Differentiable Bootstrap Particle Filters for Regime-Switching Models," *arXiv preprint arXiv:2302.10319*, 2023.

[83] J. Li, V. Lavrukhin, B. Ginsburg, R. Leary, O. Kuchaiev, J. M. Cohen, H. Nguyen, and R. T. Gadde, "Jasper: An End-to-End Convolutional Neural Acoustic Model," *arXiv preprint arXiv:1904.03288*, 2019.

[84] M. I. Rabinovich and A. L. Fabrikant, "Stochastic self-modulation of waves in nonequilibrium media," *J. Exp. Theor. Phys*, vol. 77, pp. 617–629, 1979.

[85] P. J. Wangersky, "Lotka-Volterra population models," *Annual Review of Ecology and Systematics*, vol. 9, pp. 189–218, 1978.

[86] F. Brauer, "Compartmental models in epidemiology," *Mathematical epidemiology*, pp. 19–79, 2008.

[87] I. Prigogine and R. Lefever, "Symmetry breaking instabilities in dissipative systems. II," *The Journal of Chemical Physics*, vol. 48, no. 4, pp. 1695–1700, 1968.

[88] R. J. Field and R. M. Noyes, "Oscillations in chemical systems. IV. Limit cycle behavior in a model of a real chemical reaction," *The Journal of Chemical Physics*, vol. 60, no. 5, pp. 1877–1884, 1974.

[89] E. Chouzenoux and V. Elvira, "Graphical Inference in Non-Markovian Linear-Gaussian State-Space Models," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2024, pp. 13 141–13 145.

[90] E. Tan, D. Corrêa, T. Stemler, and M. Small, "A Backpropagation Algorithm for Inferring Disentagled Nodal Dynamics and Connectivity Structure of Dynamical Networks," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 1, pp. 613–624, 2024.

[91] D. Zambon, A. Cini, L. Livi, and C. Alippi, "Graph state-space models," *arXiv preprint arXiv:2301.01741*, 2023.

[92] G. Mateos, S. Segarra, A. G. Marques, and A. Ribeiro, "Connecting the dots: Identifying network structure via graph signal processing," *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 16–43, 2019.

[93] B. Ginsburg, P. Castonguay, O. Hrinchuk, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, H. Nguyen, and J. M. Cohen, "Stochastic Gradient Methods with Layer-wise Adaptive Moments for Training of Deep Networks," *CoRR*, vol. abs/1905.11286, 2019. [Online]. Available: http://arxiv.org/abs/1905.11286

[94] P. L. Combettes and J.-C. Pesquet, "Proximal Splitting Methods in Signal Processing," *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, pp. 185–212, 2011.

[95] L. Rosasco, S. Villa, and B. C. Vũ, "Convergence of Stochastic Proximal Gradient Algorithm," *Applied Mathematics & Optimization*, vol. 82, pp. 891–917, 2020.

[96] P. L. Combettes and J.-C. Pesquet, "Stochastic Approximations and Perturbations in Forward-Backward Splitting for Monotone Operators," *Pure and Applied Functional Analysis*, vol. 1, no. 1, pp. 13–37, 2016.

[97] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/google/jax

[98] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[99] F. Bach, R. Jenatton, J. Mairal, G. Obozinski *et al.*, "Optimization with Sparsity-Inducing Penalties," *Foundations and Trends® in Machine Learning*, vol. 4, no. 1, pp. 1–106, 2012.

[100] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[101] Y. Kuramoto, *Chemical Oscillations, Waves, and Turbulence*. Springer, 1984.

[102] B. Cox, S. Pérez-Vieites, N. Zilberstein, M. Sevilla, S. Segarra, and V. Elvira, "End-to-End Learning of Gaussian Mixture Proposals Using Differentiable Particle Filters and Neural Networks," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2024, pp. 9701–9705.

[103] B. Cox, S. Segarra, and V. Elvira, "Learning state and proposal dynamics in state-space models using differentiable particle filters and neural networks," *Signal Processing*, vol. 234, p. 109998, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165168425001124

[104] V. Elvira, L. Martino, M. F. Bugallo, and P. M. Djuric, "Elucidating the Auxiliary Particle Filter via Multiple Importance Sampling [Lecture Notes]," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 145–152, Nov. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8887554/

[105] N. Branchini and V. Elvira, "Optimized Auxiliary Particle Filters: Adapting Mixture Proposals via Convex Optimization," in *Uncertainty in Artificial Intelligence*. PMLR, 2021, pp. 1289–1299.

[106] S. S. Gu, Z. Ghahramani, and R. E. Turner, "Neural Adaptive Sequential Monte Carlo," *Advances in neural information processing systems*, vol. 28, 2015.

[107] C. Naesseth, S. Linderman, R. Ranganath, and D. Blei, "Variational Sequential Monte Carlo," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2018, pp. 968–977.

[108] V. Elvira, L. Martino, D. Luengo, and M. F. Bugallo, "Improving Population Monte Carlo: Alternative Weighting and Resampling Schemes," *Signal Processing*, vol. 131, pp. 77–91, Feb. 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0165168416301633

[109] F. Gama, N. Zilberstein, R. G. Baraniuk, and S. Segarra, "Unrolling Particles: Unsupervised Learning of Sampling Distributions," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 5498–5502.

[110] F. Gama, N. Zilberstein, M. Sevilla, R. G. Baraniuk, and S. Segarra, "Unsupervised Learning of Sampling Distributions for Particle Filters," *IEEE Transactions on Signal Processing*, vol. 71, pp. 3852–3866, 2023.

[111] M. Zhu, K. Murphy, and R. Jonschkowski, "Towards Differentiable Resampling," *arXiv preprint arXiv:2004.11938*, 2020.

[112] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the Variance of the Adaptive Learning Rate and Beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[113] M. Schuster and K. K. Paliwal, "Bidirectional Recurrent Neural Networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[114] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, "Recent Advances in Recurrent Neural Networks," *arXiv preprint arXiv:1801.01078*, 2017.

[115] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 11 106–11 115.

[116] P. Kidger and C. Garcia, "Equinox: Neural Networks in JAX via Callable PyTrees and Filtered Transformations," *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

[117] D. Rezende and S. Mohamed, "Variational Inference with Normalizing Flows," in *International Conference on Machine Learning*. PMLR, 2015, pp. 1530–1538.

[118] B. L. Trippe and R. E. Turner, "Conditional Density Estimation with Bayesian Normalising Flows," *arXiv preprint arXiv:1802.04908*, 2018.

[119] E. Jang, S. Gu, and B. Poole, "Categorical Reparameterization with Gumbel-Softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[120] E. G. Tabak and C. V. Turner, "A Family of Nonparametric Density Estimation Algorithms," *Communications on Pure and Applied Mathematics*, vol. 66, no. 2, pp. 145–164, 2013.

[121] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[122] H. Yan, J. Du, V. Y. Tan, and J. Feng, "On robustness of neural ordinary differential equations," *arXiv preprint arXiv:1910.05513*, 2019.

[123] C. Rackauckas, M. Innes, Y. Ma, J. Bettencourt, L. White, and V. Dixit, "Diffeqflux. jl-A julia library for neural differential equations," *arXiv preprint arXiv:1902.02376*, 2019.

[124] J. H. Kotecha and P. M. Djuric, "Gaussian sum particle filtering," *IEEE Transactions on signal processing*, vol. 51, no. 10, pp. 2602–2612, 2003.

[125] X. Liang, S. Livingstone, and J. Griffin, "Adaptive random neighbourhood informed Markov chain Monte Carlo for high-dimensional Bayesian variable selection," *Statistics and Computing*, vol. 32, no. 5, p. 84, 2022.

[126] M. Cranmer, "Interpretable Machine Learning for Science with PySR and SymbolicRegression. Jl," *arXiv preprint arXiv:2305.01582*, 2023.