GraphGrad: Efficient Estimation of Sparse Polynomial Representations for General State-Space Models

Benjamin Cox, *Student Member, IEEE*, Émilie Chouzenoux, *Senior Member, IEEE*, and Víctor Elvira, *Senior Member, IEEE*

***Abstract*—State-space models (SSMs) are a powerful statistical tool for modelling time-varying systems via a latent state. In these models, the latent state is never directly observed. Instead, a sequence of observations related to the state is available. The SSM is defined by the state dynamics and the observation model, both of which are described by parametric distributions. Estimation of parameters of these distributions is a very challenging, but essential, task for performing inference and prediction. Furthermore, it is typical that not all states of the system interact. We can therefore encode the interaction of the states via a graph, usually not fully connected. However, most parameter estimation methods do not take advantage of this feature. In this work, we propose GraphGrad, a fully automatic approach for obtaining sparse estimates of the state interactions of a non-linear SSM via a polynomial approximation. This novel methodology unveils the latent structure of the data-generating process, allowing us to infer both the structure and value of a rich and efficient parameterisation of a general SSM. Our method utilises a differentiable particle filter to optimise a Monte Carlo likelihood estimator. It also promotes sparsity in the estimated system through the use of suitable proximity updates, known to be more efficient and stable than subgradient methods. As shown in our paper, a number of well-known dynamical systems can be accurately represented and recovered by our method, providing basis for application to real-world scenarios.**

I. Introduction

State-space models (SSMs) are a powerful tool for describing systems that evolve in time. SSMs are utilised in many fields, such as target tracking, [1], epidemiology [2], ecology [3], finance [4], and meteorology [5]. SSMs represent a dynamical system via an unobserved hidden state and a series of related observations. The objective is then to estimate the hidden state, conditional on the series of observations. Estimation of the state using only observations available at the time the state occurs is called the filtering problem, while estimation of the state using the entire observation series, is known as the smoothing problem. In order to solve either of these problems, one must know both the conditional densities of the hidden state and of the observation.

When the state transition and observation model are linear and admit Gaussian densities, a linear-Gaussian SSM is obtained, and one can solve the filtering and smoothing problems exactly using the Kalman filter [6] and RTS smoother [7] respectively. If the state dynamics or observation model are not linear, then we have a non-linear state-space model (NLSSM), and the Kalman filter cannot be applied without simplifying the model. In this case we can use Gaussian approximations, such as the Unscented Kalman filter [8] or the extended Kalman filter, although in many cases a Gaussian approximation is not sufficiently rich to capture the behaviour of the system. In such cases, more accurate results might be obtained by a particle filter, which approximates the posterior density of the state via a series of importance weighted Monte Carlo samples [9]. However, all of these techniques require knowing the form of the transition and observations models, and for all parameters therein to be known. It is common for the parameters to be unknown, and therefore they must be estimated. Furthermore, in many cases, the form of the model is not known, and therefore we must impose a form before we can estimate anything. In the case of dynamical systems where we observe the state directly, methods such as SINDy [10] can be used to recover the system dynamics in the presence of unknown dynamics. 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 thereof. 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. Thanks to the recent advent of differentiable particle filters [11]–[13], it recently became possible to obtain the gradient of the estimated likelihood of model parameters, and hence became much simpler to fit them using, e.g., maximum likelihood, assuming the form of the model is known.

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 [14] and Lorenz 96 [15] 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 paper, we propose GraphGrad, a method to model and learn the transition distribution of a non-linear

B.C. acknowledges support from the *Natural Environment Research Council* of the UK through a SENSE CDT studentship (NE/T00939X/1). The work of V. E. is supported by the ARL/ARO under grant W911NF-22-1-0235. É.C. acknowledges support from the European Research Council Starting Grant MAJORIS ERC-2019-STG850925.

SSM via a polynomial approximation of the state.[1]

- 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-likelihood, 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.

Structure. In Section II, we present the underlying particle filtering methodology that we build upon, and introduce the differentiable particle filter, as well as the notation used throughout this paper. We present the method in Section III, and discussion in Section IV. We present several numerical experiments in Section V, and conclude in Section VI.

II. Background

A. Particle filtering

The general SSM can be described by

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

where $t = 1, \dots, T$ 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, 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 weights, $\{\mathbf{x}_{1:T}^{(k)}, \widetilde{w}_{1:T}^{(k)}\}_{k=1}^K$. The posterior density can then be approximated by

$$p(\mathbf{x}_t|\mathbf{y}_{1:t};\boldsymbol{\theta}) \approx \sum_{k=1}^{K} \widetilde{w}_t^{(k)} \delta_{\mathbf{x}_t^{(k)}}. \tag{2}$$

A commonly used particle filtering method is the sequential importance resampling algorithm, given in Alg. 1. 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 $\mathbf{x}_{t-1}^{(k)}$, $k = 1, \dots, K$, 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 $\mathbf{x}_t^{(k)}$, $k = 1, \dots, K$, 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 $w_t^{(k)}$, $k = 1, \dots, K$, in line 9, and the normalised weights (line 10).

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

B. 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

$$p(\mathbf{y}_{1:T}|\boldsymbol{\theta}) = \prod_{t=1}^{T} p(\mathbf{y}_t|\mathbf{y}_{1:t-1};\boldsymbol{\theta}), \tag{3}$$

where $p(\mathbf{y}_1|\mathbf{y}_{1:0};\boldsymbol{\theta}) := p(\mathbf{y}_1|\boldsymbol{\theta})$ [17]. 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 [18].

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.

[1] A limited version of this work was presented by the authors in the conference paper [16], which presented a version of this work using automatic differentiation through the $L1$ norm regularisation, and which does not utilise observation batching to stabilise the learning. We propose here a more methodologically advanced method, which utilises the proximal operator for applying the $L1$ regularisation, thereby making the method more computationally efficient as it is faster to converge, and have added batch-stabilised learning for long observation series, allowing the method to be applied to large systems without careful initialisation. Furthermore, we include extensive methodological discussion, and provide practical guidance for practitioners. Finally, we include a large number of numerical experiments, which were not present in the prior work.

In the case of a linear-Gaussian SSM, we have closed form methods for the MAP estimator assuming a diffuse prior [17], and efficient methods for obtaining the MAP estimator for sparsity promoting priors [19]. Distributional methods estimate the posterior distribution of the parameter, with common methods being importance sampling [20], Markov chain Monte Carlo [21], and variational inference [22]. For the linear-Gaussian SSM, we can utilise reversible jump Markov chain Monte Carlo to obtain an estimate of the distribution of sparsity [23], however no similar method exists for the general SSM. Our proposed method in this work is a point estimation method.

For general SSMs, the likelihood 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 $\nu_t^{(k)} = w_t^{(k)} \widetilde{w}_{t-1}^{(k)}$ be the adjusted importance weight. We can estimate the parameter likelihood using the particle filter, by the Monte Carlo estimate

$$\begin{aligned} p(\boldsymbol{\theta}|\mathbf{y}_{1:T}) &\propto p(\boldsymbol{\theta}) p(\mathbf{y}_{1:T}|\boldsymbol{\theta}), \\ &\approx p(\boldsymbol{\theta}) \prod_{t=1}^{T} \left(\sum_{k=1}^{K} \nu_t \right), \end{aligned} \tag{4}$$

where $w_t^{(k)}$ and $\widetilde{w}_{t-1}^{(k)}$ are the weights of the particle filter as in Alg. 1 [17, Chapter 12]. Note that the weights are dependent on the parameter $\boldsymbol{\theta}$ through their computation in Alg. 1. Using these weights, we construct an estimate of the log-likelihood by

$$\begin{aligned} \log(p(\boldsymbol{\theta}|\mathbf{y}_{1:T})) + c &= \log(p(\boldsymbol{\theta})) + \log(p(\mathbf{y}_{1:T}|\boldsymbol{\theta})), \\ &\approx \log(p(\boldsymbol{\theta})) + \sum_{t=1}^{T} \log \left(\sum_{k=1}^{K} \nu_t^{(k)} \right), \end{aligned} \tag{5}$$

where the constant c results from the proportionality in Eq. (4), and we note that if resampling occurred at time $t-1$, which is always the case if following Alg. 1, we have $\widetilde{w}_{t-1}^{(k)} = 1/K$, 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 $\sum_{k=1}^{K} w_t^{(k)} \widetilde{w}_{t-1}^{(k)}$ using a logsumexp reparametrisation.

C. Differentiable particle filter

Algorithm 2 Stop-gradient differentiable particle filter (DPF) [13]

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

The outputs of the particle filter, as given in Alg. 1, namely $\{\mathbf{x}_{1:T}^{(k)}, w_{1:T}^{(k)}\}_{k=1}^{K}$, are not differentiable with respect to $\boldsymbol{\theta}$ [13], 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 [13].

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 $\boldsymbol{\theta}$ [12], [13], [24], [25]. 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 [11], to optimal transport [12]. An overview of the DPF and its interplay with deep learning methods can be found in [24], which motivates our choice of differentiable particle filter, and our usage of stochastic gradient descent methods.

In this work, we built upon the DPF approach from [13], which utilises a stop gradient operator to make the resampling step differentiable. We summarise this method in Alg. 2. 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. 2, at each time-step t, we first sample the previous particle set, sampling $a_t^{(k)}$, $k = 1, \ldots, K$, 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. 1, described in Sec. II-A, 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 likelihood 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. 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 [26], [27].

D. Notation

We here present our notation, used throughout the paper. 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 [13].

III. 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 [14], [15] or the Rabinovich-Fabrikant system [28], the Lotka-Volterra model and many of its extensions [29], many compartmental epidemiological models [30], and several ODEs resulting from PDE discretisations (such as from the Brusselator model [31] and Oregonator model [32]). 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

$$\mathbf{x}_t \sim p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{C}) := \mathcal{N}(f(\mathbf{x}_{t-1}, \mathbf{C}; \mathbf{D}), \boldsymbol{\Sigma}_v), \tag{6}$$

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

$$f(\mathbf{x}, \mathbf{C}; \mathbf{D}) = \sum_{j=1}^{M} \left(\mathbf{C}_{\cdot,j} \cdot \prod_{i=1}^{N_x} x_i^{D_{i,j}} \right). \tag{7}$$

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 III-A. 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 III-B 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 III-C. We move to the presentation of our approach to estimate $\mathbf{C}$, starting from the problem definition in Section III-D, a single batch algorithm S-GraphGrad in Section III-E, and a practical mini-batch implementation, leading to our final algorithm B-GraphGrad, in Section III-F.

A. 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

$$M = \sum_{n=0}^{d} \binom{n + N_x - 1}{N_x - 1} \tag{8}$$

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. (7). 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.

1) 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 $j \in \{1, \ldots, M\}$, $\mathbf{D}_{\cdot,j} = [0, 1, 2]$, 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. 3, where 'count' and 'cwr' are defined in Sec. II-D. Alg. 3 takes the union of all possible combinations with replacement of the set $\{1, 2, \dots, N_x\}$ 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 $i \in \{1, \dots, N_x\}$ and $j \in \{1, \dots, M\}$.

Algorithm 3 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 $\mathcal{Q} = \bigcup_{\delta=0}^{d} \mathrm{cwr}([1, 2, \dots, N_x], \delta)$
4: $D_{i,j} = \mathrm{count}(i, \mathcal{Q}_j)\ \forall\, i \in \{1, \dots, N_x\},\ j \in \{1, \dots, M\}$.

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 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 work, 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. 3, by iterating over degrees.

2) 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. (7). 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. (1), where $\boldsymbol{\theta}$ is the set of learnt parameters, in our case $\mathbf{C}$.

B. A graphical interpretation of matrices $\mathbf{C}$ *and* $\mathbf{D}$

Within time series modelling, we can often interpret the driving parameters of a sparse model as a graph encoding the connectivity of the system [19], [23], [33]–[37]. This is the case here as well, where we observe that, for $(a, b) \in \{1, \dots, N_x\}^2$, state dimension b affects state dimension a in our estimated dynamics if $(\mathrm{abs}(\mathbf{C})\mathbf{D}^\top)_{a,b} \neq 0$, where we note that $\mathrm{abs}(\mathbf{C})\mathbf{D}^\top$ 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 $(\mathrm{abs}(\mathbf{C})\mathbf{D}^\top)_{a,b} \neq 0$.

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 $\mathbf{A} = \mathbb{1}_{\neq 0}(\mathrm{abs}(\mathbf{C})\mathbf{D}^\top) \in \{0, 1\}^{N_x \times N_x}$, 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 $(a, b) \in \{1, \dots, N_x\}^2$ 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 [38]. 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 [39].

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 $\mathrm{abs}(\mathbf{C}_{\cdot,j})\mathbf{D}_{\cdot,j}^\top$. 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. IV-D, 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 [40], 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.

C. An example: Lorenz 63

We present here a worked example utilising the Lorenz 63 model. The Lorenz 63 model [14] is a popular chaotic oscillator

model, with a discrete time variant given by,

$$\begin{aligned} x_{1,t} &= x_{1,t-1} + \Delta t(\sigma(x_{2,t-1} - x_{1,t-1})) + v_{1,t}, \\ x_{2,t} &= x_{2,t-1} + \Delta t(x_{1,t-1}(\rho - x_{3,t-1}) - x_{2,t-1}) + v_{2,t}, \\ x_{3,t} &= x_{3,t-1} + \Delta t(x_{1,t-1}x_{2,t-1} - \beta x_{3,t-1}) + v_{3,t}, \\ \mathbf{y}_t &= \mathbf{x}_t + \mathbf{r}_t, \end{aligned} \tag{9}$$

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. (9) 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. (9) 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. (6) and Eq. (7) yields the system given in Eq. (9). Furthermore, we construct the adjacency matrix $\mathbf{A} = \mathbb{1}_{\neq 0}(\mathrm{abs}(\mathbf{C})\mathbf{D}^\top)$ from the above $\mathbf{C}$ and $\mathbf{D}$ matrices, yielding the adjacency matrix and associated directed graph given in Fig. 1.

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

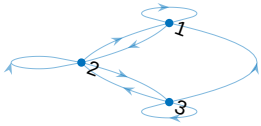

Figure 1: Graph and adjacency matrix encoding the connectivity of the Lorenz 63 system.

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.

D. 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 penalised negative log-likelihood ℓ_R, given by

$$\ell_R(\mathbf{C}|\mathbf{y}_{1:T}, \lambda) = \ell(\mathbf{C}; \mathbf{y}_{1:T}) + \lambda R(\mathbf{C}), \tag{10}$$

where

$$\ell(\mathbf{C}|\mathbf{y}_{1:T}) = -\log(p(\mathbf{y}_{1:T}|\mathbf{C})), \tag{11}$$

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

$$\widehat{\mathbf{C}} = \underset{\mathbf{C} \in \mathbb{R}^{N_x \times M}}{\operatorname{argmin}} \ \ell_R(\mathbf{C}|\mathbf{y}_{1:T}, \lambda). \tag{12}$$

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

$$(\forall \mathbf{C} \in \mathbb{R}^{N_x \times M}) \quad R(\mathbf{C}) = ||\mathbf{C}||_1 := \sum_{j=1}^{M} \sum_{i=1}^{N_x} |C_{i,j}|. \tag{13}$$

We propose to solve Eq. (12), 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.

1) Estimating the parameter likelihood and its gradient: In order to resolve Eq. (12) 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. (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. 1, are subject to a non-differentiable resampling step.

Thankfully, we can utilise the DPF approach discussed in Sec. II-C, 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 [27], [41] to compute our gradient updates for the negative log-likelihood, as it is robust to gradient outliers [41].

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

$$\widetilde{\mathbf{C}}_s = \mathrm{update}_\eta(\mathbf{C}_{s-1}, \nabla\ell(\mathbf{C}_{s-1}|\mathbf{y}_{1:T})), \tag{14}$$

where $\mathrm{update}_\eta(\mathbf{A}, \nabla\ell(\mathbf{B}|\mathbf{y}_{1:T}))$ 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 $\mathbf{y}_{1:T}$.

2) Proximal update on the penalty term: Given that we can estimate the negative log-likelihood $\ell(\mathbf{C}; \mathbf{y}_{1:T})$ and its gradient $\nabla\ell(\mathbf{C}; \mathbf{y}_{1:T})$, 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 [42], 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

$$\begin{aligned} \widetilde{\mathbf{C}}_s &= \mathrm{update}_\eta(\mathbf{C}_{s-1}, \nabla\ell(\mathbf{C}_{s-1}|\mathbf{y}_{1:T})), \\ \mathbf{C}_s &= \mathcal{T}_{\eta\cdot\lambda}(\widetilde{\mathbf{C}}_s), \end{aligned} \tag{15}$$

where the soft-thresholding operator,

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

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 [43], [44].

Note that automatic differentiation can also be used when applying the $L1$ penalty, as the $L1$ penalty admits a subgradient [45], [46]. 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 [47].

E. S-GraphGrad algorithm

We have now all the elements for solving (12). We first start with a full batch implementation, in Alg. 4, which we call S-GraphGrad, and which operates on the entire series of observations at once.

Algorithm 4 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. 2 with $p(\mathbf{x}_t|\mathbf{x}_{t-1}; \mathbf{C}_{s-1}, \mathbf{D})$ and observations $\mathbf{y}_{1:T}$.
5: Estimate $\ell(\mathbf{C}_{s-1}|\mathbf{y}_{1:T})$ and $\nabla\ell(\mathbf{C}_{s-1}|\mathbf{y}_{1:T})$ via Eq. (4) and back-propagation.
6: Set $\widetilde{\mathbf{C}}_s = \mathrm{update}_\eta(\mathbf{C}_{s-1}, \nabla\ell(\mathbf{C}_{s-1}|\mathbf{y}_{1:T}))$.
7: Set $\mathbf{C}_s = \mathcal{T}_{\eta\cdot\lambda}(\widetilde{\mathbf{C}}_s)$.
8: **end for**
9: Output $\mathbf{C} = \mathbf{C}_S$.

1) 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. (6), 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. (4), and use automatic differentiation to obtain an estimate of $\nabla\ell(\mathbf{C}^{s-1}|\mathbf{y}_{1:T})$ (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 $\nabla\ell(\mathbf{C}_{s-1}|\mathbf{y}_{1:T})$ are those of the negative log-likelihood, as we are aiming to maximise the 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.

2) Combating likelihood degeneracy: In practice, using S-GraphGrad is hampered by likelihood degeneracy, which causes the likelihood 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. (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.

F. 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. 4 (S-GraphGrad) in Sec. III-E, 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 $b = 1, \dots, B$. Note that these batches are of increasing size, with $\mathbf{y}^{(b)} \subseteq \mathbf{y}^{(b+1)}$ for $b = 1, \dots, B$. Within each batch of observations, we perform S runs of Alg. 4 (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. 2 [9]. 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. 5 (B-GraphGrad), and describe it below.

Algorithm 5 Batched GraphGrad algorithm (B-GraphGrad)

1: **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.
2: **Output:** Sparse $\mathbb{R}^{N_x \times M}$ matrix $\mathbf{C}$ of polynomial coefficients.
3: Construct $\mathbf{D}$ by running Alg. 3.
4: Randomly initialise $\mathbf{C}_0 \in \mathbb{R}^{N_x \times M}$ element-wise by sampling a $\mathcal{U}(-1, 1)$ distribution.
5: **for** $b = 1, \dots, B$ **do**
6: Set $\mathbf{y}^{(b)} := \mathbf{y}_{1:\lceil bT/B \rceil}$.
7: Run Alg. 4 (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.
8: **end for**
9: Output $\mathbf{C} := \mathbf{C}_B$

1) B-GraphGrad description: As stated in Sec. III-F, 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. 3 described in Sec. III-A1. 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. 5 (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. 4 (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. (12).

IV. Discussion

We now discuss our algorithm, B-GraphGrad, as given in Alg. 5, and provide some potential extensions and modifications to particular systems.

A. 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 reparameterisation trick [48]. 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.

B. Computational cost

B-GraphGrad, as given in Alg. 5, 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. (7) 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 [45].

C. Exploiting parallelisation to decrease runtime

B-GraphGrad, if implemented following Alg. 5, 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 $||\mathbf{C} - \mathbf{C}^{(p)}|| < \epsilon \; \forall p \in \{1, \ldots, P\}$ 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.

D. 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 [10], 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. (7). 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.

V. 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. V-B) and Lorenz 96 (Sec. V-C) oscillators, and estimating a non-polynomial system, the Kuramoto oscillator (Sec. V-D). 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).

A. Experimental setup

For our numerical experiments, we use the following settings, unless stated otherwise. We use the Novograd optimisation scheme [27], [41], for update in Alg. 4 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. 2. T denotes the length of the observation series. For a given polynomial degree d, we construct the degree matrix $\mathbf{D}$ following Sec. III-A1.

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 $[-5, 2]$.

B. Lorenz 63 model

We start our experiments, with the Lorenz 63 system [14], which we transform into an NLSSM using an Euler discretisation with a timestep of $\Delta t = 0.025$, yielding the system

$$\begin{aligned} x_{1,t+1} &= x_{1,t} + \Delta t(\sigma(x_{2,t} - x_{1,t})) + \sqrt{\Delta t} v_{1,t+1}, \\ x_{2,t+1} &= x_{2,t} + \Delta t(x_{1,t}(\rho - x_{3,t}) - x_{2,t}) + \sqrt{\Delta t} v_{2,t+1}, \\ x_{3,t+1} &= x_{3,t} + \Delta t(x_{1,t}x_{2,t} - \beta x_{3,t}) + \sqrt{\Delta t} v_{3,t+1}, \end{aligned} \quad (16)$$

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. III-C 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.

Table I
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	-	-	-	-

1) Varying series length, maximum degree $d = 2$: Table I presents the results for learning over a range of series lengths T

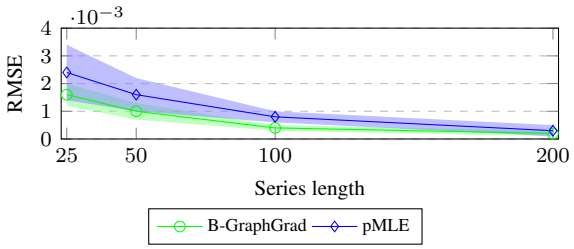

Figure 2: Comparison of B-GraphGrad with pMLE over variable series length on the Lorenz 63 oscillator, with maximum polynomial degree $d = 2$. Markers denote mean performance, with the ribbons being symmetric 95% intervals.

Table II
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

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 II, 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. III-F to remove the quadratic scaling by fixing the number of batches, in which case the runtime scales linearly in T.

2) Varying series length and maximum degree $d = 3$: Table III presents the results of our GraphGrad method over

Table III
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	-	-	-	-

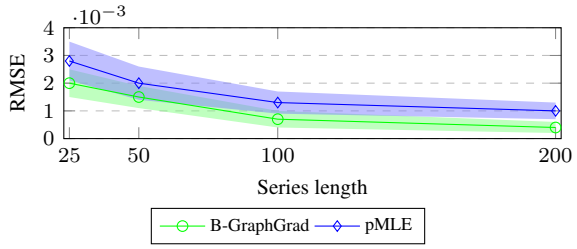

Figure 3: Comparison of B-GraphGrad with pMLE over variable series length on the Lorenz 63 oscillator, with maximum polynomial degree $d = 3$. Markers denote mean performance, with the ribbons being symmetric 95% intervals.

Table IV
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	-	-	-	-

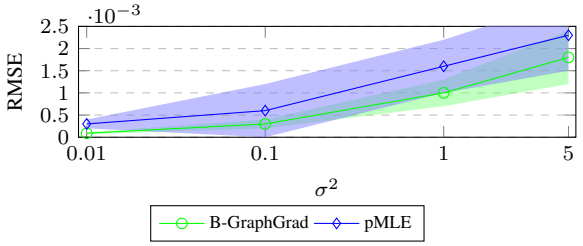

Figure 4: Comparison of B-GraphGrad with pMLE over variable noise magnitude on the Lorenz 63 oscillator, with maximum polynomial degree $d = 2$. Markers denote mean performance, with the ribbons being symmetric 95% intervals.

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 [49] 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 I 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.

3) Variable noise magnitude: In Table IV, 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 I and III, 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.

4) Likelihood degeneracy: In Sec. III-E2, 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. (9).

Table V
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 V 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.

5) 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 VI 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. V-B1, except d, which we vary. We present the results in Table VI.

From Table VI, 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

Table VI
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

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. (7) 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.

6) Prox update compared to subgradient update: B-GraphGrad uses the proximal operator of the $L1$ norm to apply a sparsity promoting penalty, following Sec. III-D2. 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 $\operatorname{sgn}(x)$, using the convention $\operatorname{sgn}(0) = 0$. 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. V-B1. 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. V-B1.

Table VII
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 negligable difference in computational cost, this illustrates the superiority of the proximal over the subgradient method for this problem.

C. 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 [15], which we transform into a NLSSM using an Euler discretisation, yielding the system

$$\begin{aligned} d_{i,t+1} &= x_{i-1,t}(x_{i+1,t} - x_{i-2,t}) - x_{i,t} + F, \\ x_{i,t+1} &= x_{i,t} + \Delta t \cdot d_{i,t+1} + \sqrt{\Delta t} \cdot v_{i,t+1}, \\ y_{i,t+1} &= x_{i,t+1} + \sqrt{\Delta t} \cdot r_{i,t+1}, \end{aligned} \tag{17}$$

for $i = \{1, \dots, N_x\}$, 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. III-B, is given in Figure 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 $k \in \mathbb{Z}$.

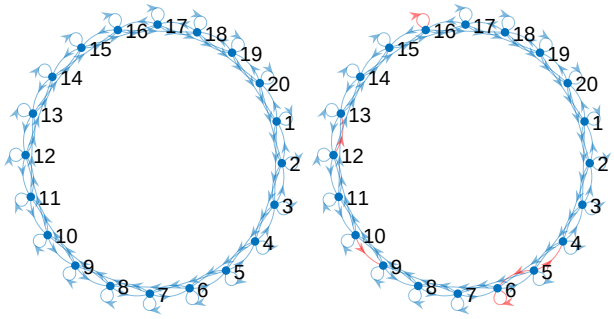

Figure 5: Graph encoding the connectivity of the Lorenz 96 system for $N_x = 20$. The left plot represents the true connectivity of the system under the graphical perspective described in Section III-B. The right plot represents the connectivity retrieved by the B-GraphGrad algorithm. The links in blue 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 $d = 3$, GraphGrad performs well, outperforming pMLE by a considerable margin. We see no significant deterioration of performance in GraphGrad in this setting, noting that we are estimating 4620 parameters in the $d = 2$ case, and 35420 parameters in the $d = 3$ case.

D. Kuramoto oscillator

The Kuramoto oscillator [49] is a mathematical model that describes the behaviour of a system of phase-coupled oscillators.

Table VIII
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	-	-	-	-

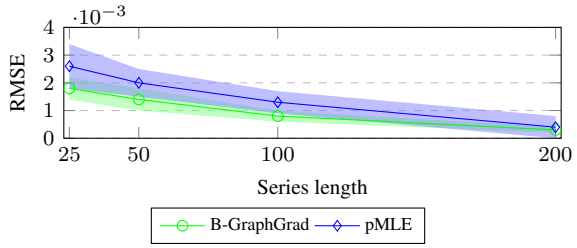

Figure 6: Comparison of B-GraphGrad with pMLE over variable series length on the Lorenz 96 oscillator, with maximum polynomial degree $d = 2$. Markers denote mean performance, with the ribbons being symmetric 95% intervals.

Table IX
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	-	-	-	-

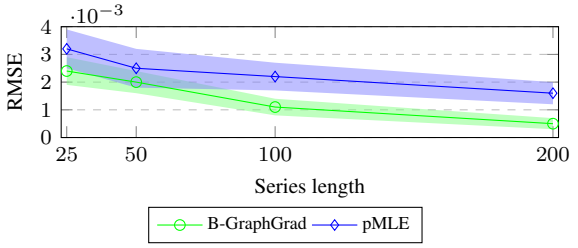

Figure 7: Comparison of B-GraphGrad with pMLE over variable series length on the Lorenz 96 oscillator, with maximum polynomial degree $d = 3$. Markers denote mean performance, with the ribbons being symmetric 95% intervals.

The model is described, for all $i \in \{1, \dots, N_x\}$, by

$$\frac{\mathrm{d}\phi_i}{\mathrm{d}t} = \eta_i + N_x^{-1} \sum_{o=1}^{N_x} K \sin(\phi_i - \phi_o), \tag{18}$$

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, diverge to $\pm\infty$ as $t \to \infty$. To address this, we transform Eq. (18) by introducing derived parameters R and ψ such that

$$\begin{aligned} R\exp(\sqrt{-1}\psi) &= N_x^{-1} \sum_{o=1}^{N_x} \exp(\sqrt{-1}\phi_o), \\ (\forall i \in \{1, \dots, N_x\})\ \frac{\mathrm{d}\phi_i}{\mathrm{d}t} &= \eta_i + KR\sin(\psi - \phi_i), \end{aligned} \tag{19}$$

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. (19) into a NLSSM using an Euler discretisation, yielding

$$\begin{aligned} R\exp(\sqrt{-1}\psi) &= N_x^{-1} \sum_{o=1}^{N_x} \exp(\sqrt{-1}x_{o,t}), \\ d_{i,t+1} &= \eta_i + KR\sin(\psi - x_i), \\ x_{i,t+1} &= x_{i,t} + \Delta t d_{i,t+1} + \sqrt{\Delta t} v_{i,t+1}, \\ y_{i,t+1} &= x_{i,t+1} + \sqrt{\Delta t} r_{i,t+1}, \end{aligned} \tag{20}$$

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, $\mathrm{nRMSE} = 1$ 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. (20) is known, and estimates the $\boldsymbol{\eta}$ and K parameters using maximum likelihood.

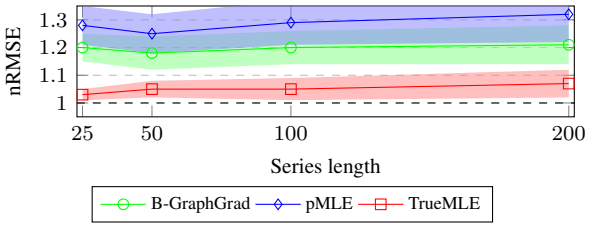

Figure 8: Comparison of B-GraphGrad with pMLE and TrueMLE over variable series length on the Kuramoto oscillator. Quantities are divided by the RMSE of a filter utilising the true model. Markers denote mean performance, with the ribbons being symmetric 95% intervals.

We see in Figure 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.

VI. Conclusion

In this work, 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 parameterisation 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.

References

[1] X. Wang, T. Li, S. Sun, and J. M. Corchado, "A survey of recent advances in particle filters and remaining challenges for multitarget tracking," *Sensors*, vol. 17, no. 12, p. 2707, 2017.

[2] T. A. Patterson, A. Parton, R. Langrock, P. G. Blackwell, L. Thomas, and R. King, "Statistical modelling of individual animal movement: an overview of key methods and a discussion of practical challenges," *AStA Advances in Statistical Analysis*, vol. 101, pp. 399–438, 2017.

[3] 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.

[4] A. Virbickaitė, H. F. Lopes, M. C. Ausín, and P. Galeano, "Particle learning for Bayesian semi-parametric stochastic volatility model," *Econometric Reviews*, 2019.

[5] A. M. Clayton, A. C. Lorenc, and D. M. Barker, "Operational implementation of a hybrid ensemble/4d-Var global data assimilation system at the Met Office," *Quarterly Journal of the Royal Meteorological Society*, vol. 139, no. 675, pp. 1445–1461, 2013.

[6] 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.

[7] H. E. Rauch, F. Tung, and C. T. Striebel, "Maximum likelihood estimates of linear dynamic systems," *AIAA Journal*, vol. 3, no. 8, pp. 1445–1450, 1965.

[8] E. A. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*. IEEE, 2000, pp. 153–158.

[9] 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.

[10] 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.

[11] 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, 2018, pp. 169–178.

[12] A. Corenflos, J. Thornton, A. Doucet, and G. Deligiannidis, "Differentiable particle filtering via entropy-regularized optimal transport," *arXiv preprint arXiv:2102.07850*, 2021.

[13] A. Ścibior and F. Wood, "Differentiable particle filtering without modifying the forward pass," *arXiv preprint arXiv:2106.10314*, 2021.

[14] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of atmospheric sciences*, vol. 20, no. 2, pp. 130–141, 1963.

[15] ——, "Predictability: A problem partly solved," in *Proc. Seminar on predictability*, vol. 1, no. 1. Reading, 1996.

[16] 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.

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

[18] 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, 1996.

[19] V. Elvira and É. Chouzenoux, "Graphical inference in linear-Gaussian state-space models," *IEEE Transactions on Signal Processing*, vol. 70, pp. 4757–4771, 2022.

[20] S. T. Tokdar and R. E. Kass, "Importance sampling: a review," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 1, pp. 54–60, 2010.

[21] 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, 2010.

[22] 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, 2017.

[23] 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.

[24] X. Chen and Y. Li, "An overview of differentiable particle filters for data-adaptive sequential Bayesian inference," *arXiv preprint arXiv:2302.09639*, 2023.

[25] 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.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[27] 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.

[28] 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.

[29] P. J. Wangersky, "Lotka-volterra population models," *Annual Review of Ecology and Systematics*, vol. 9, pp. 189–218, 1978.

[30] F. Brauer, "Compartmental models in epidemiology," *Mathematical epidemiology*, pp. 19–79, 2008.

[31] 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.

[32] 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.

[33] E. Chouzenoux and V. Elvira, "Graphit: Iterative reweighted l1 algorithm for sparse graph inference in state-space models," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2023, pp. 1–5.

[34] ——, "Sparse graphical linear dynamical systems," *Journal of Machine Learning Research*, vol. 25, no. 223, pp. 1–53, 2024.

[35] ——, "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.

[36] 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.

[37] E. Chouzenoux and V. Elvira, "Sparse graphical linear dynamical systems," *Journal of Machine Learning Research*, vol. 25, no. 223, pp. 1–53, 2024.

[38] D. Zambon, A. Cini, L. Livi, and C. Alippi, "Graph state-space models," *arXiv preprint arXiv:2301.01741*, 2023.

[39] 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.

[40] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[41] 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

[42] 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.

[43] L. Rosasco, S. Villa, and B. C. Vũ, "Convergence of stochastic proximal gradient algorithm," *Applied Mathematics & Optimization*, vol. 82, pp. 891–917, 2020.

[44] 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.

[45] 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

[46] 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.

[47] 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.

[48] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[49] Y. Kuramoto, *Chemical Oscillations, Waves, and Turbulence*. Springer, 1984.