Distributional Gradient Matching for Learning Uncertain Neural Dynamics Models

Lenart Treven[*] TREVENL@ETHZ.CH
Learning and Adaptive Systems Group & Automatic Control Lab
ETH Zürich, Switzerland
Philippe Wenk[*] WENKPH@ETHZ.CH
Learning and Adaptive Systems Group
ETH Zürich and Max Planck ETH Center for Learning Systems
Florian Dörfler DORFLER@ETHZ.CH
Automatic Control Lab
ETH Zürich, Switzerland
Andreas Krause KRAUSEA@ETHZ.CH
Learning and Adaptive Systems Group
ETH Zürich, Switzerland

Abstract

Differential equations in general and neural ODEs in particular are an essential technique in continuous-time system identification. While many deterministic learning algorithms have been designed based on numerical integration via the adjoint method, many downstream tasks such as active learning, exploration in reinforcement learning, robust control, or filtering require accurate estimates of predictive uncertainties. In this work, we propose a novel approach towards estimating epistemically uncertain neural ODEs, avoiding the numerical integration bottleneck. Instead of modeling uncertainty in the ODE parameters, we directly model uncertainties in the state space. Our algorithm – *distributional gradient matching (DGM)* – jointly trains a smoother and a dynamics model and matches their gradients via minimizing a Wasserstein loss. Our experiments show that, compared to traditional approximate inference methods based on numerical integration, our approach is faster to train, faster at predicting previously unseen trajectories, and in the context of neural ODEs, significantly more accurate.

1. Introduction

For continuous-time system identification and control, ordinary differential equations form an essential class of models, deployed in applications ranging from robotics (Spong et al., 2006) to biology (Jones et al., 2009). Here, it is assumed that the evolution of a system is described by the evolution of continuous state variables, whose time-derivative is given by a set of parametrized equations. Often, these equations are derived from first principles, e.g., rigid body dynamics (Wittenburg, 2013), mass action kinetics (Ingalls, 2013), or Hamiltonian dynamics (Greydanus et al., 2019), or chosen for computational convenience (e.g., linear systems (Ljung, 1998)) or parametrized to facilitate system identification (Brunton et al., 2016).

[*] Equal Contribution. Correspondence to trevenl@ethz.ch, wenkph@ethz.ch.

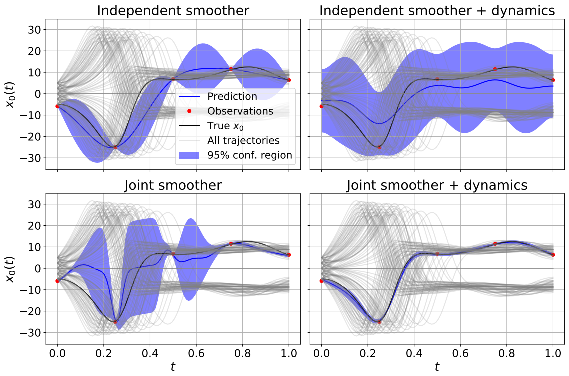

Figure 1: Illustration of DGM: Learning a joint smoother (first vs second row) across trajectories enables sharing observational data. Dynamics regularization (first vs second column) substantially improves prediction accuracy of joint smoother.

Such construction methods lead to intriguing properties, including guarantees on physical realizability (Wensing et al., 2017), favorable convergence properties (Ortega et al., 2018), or a structure suitable for downstream tasks such as control design (Ortega et al., 2002). However, such models often capture the system dynamics only approximately, leading to a potentially significant discrepancy between the model and reality (Ljung, 1999). Moreover, when expert knowledge is not available, or precise parameter values are cumbersome to obtain, system identification from raw time series data becomes necessary. In this case, one may seek more expressive *nonparametric* models instead (Rackauckas et al., 2020; Pillonetto et al., 2014). If the model is completely replaced by a neural network, the resulting model is called *neural ODE* (Chen et al., 2018). Despite their large number of parameters, as demonstrated by Chen et al. (2018); Kidger et al. (2020); Zhuang et al. (2020, 2021), deterministic neural ODEs can be efficiently trained, enabling accurate deterministic trajectory predictions.

For many practical applications however, accurate *uncertainty estimates* are essential, as they guide downstream tasks like reinforcement learning (Deisenroth and Rasmussen, 2011; Schulman et al., 2015), safety guarantees (Berkenkamp et al., 2017), robust control design (Hjalmarsson, 2005), planning under uncertainty (LaValle, 2006), probabilistic forecasting in meteorology (Fanfarillo et al., 2021), or active learning / experimental design (Srinivas et al., 2010). A common way of obtaining such uncertainties is via a Bayesian framework. However, as observed by Dandekar et al. (2021), Bayesian training of neural ODEs in a dynamics setting remains largely unexplored. They demonstrate that initial variational-based inference schemes for Bayesian neural ODEs suffer from several serious drawbacks and thus propose sampling-based alternatives. However, as surfaced by our experiments in Section 4, sampling-based approaches still exhibit serious challenges. These pertain both to robustness (even if highly informed priors are supplied), and reliance on frequent numerical integration of large neural networks, which poses severe computational challenges for many downstream tasks like sampling-based planning (Karaman and Frazzoli, 2011) or uncertainty propagation in prediction.

In this work, we propose a novel approach for uncertainty quantification in nonlinear dynamical systems (cf. Figure 1). Crucially, our approach avoids explicit costly and non-robust numerical integration, by employing a probabilistic smoother of the observational data, whose representation we learn jointly across multiple trajectories. To capture dynamics, we regularize our smoother with a dynamics model. Latter captures epistemic uncertainty in the gradients of the ODE, which we match with the smoother's gradients by minimizing a Wasserstein loss, hence we call our approach *Distributional Gradient Matching (*DGM*)*. In summary, our main contributions are:

- We develop DGM, an approach[1] for capturing epistemic uncertainty about nonlinear dynamical systems by *jointly* training a smoother and a neural dynamics model;
- We provide a computationally efficient and statistically accurate mechanism for prediction, by focusing directly on the posterior / predictive state distribution.
- We experimentally demonstrate the effectiveness of our approach on learning challenging, chaotic dynamical systems, and generalizing to new unseen inital conditions.

2. Background

Problem Statement Consider a continuous-time dynamical system whose K-dimensional state $\boldsymbol{x} \in \mathbb{R}^K$ evolves according to an unknown ordinary differential equation of the form

$$\dot{\boldsymbol{x}} = \boldsymbol{f}^*(\boldsymbol{x}). \tag{1}$$

Here, $\boldsymbol{f}^*$ is an arbitrary, unknown function assumed to be locally Lipschitz continuous, to guarantee existence and uniqueness of trajectories for every initial condition. In our experiment, we initialize the system at M different initial conditions $\boldsymbol{x}_m(0)$, $m \in \{1, \dots, M\}$, and let it evolve to generate M trajectories. Each trajectory is then observed at discrete (but not necessarily uniformly spaced) time-points, where the number of observations $(N_m)_{m\in\{1\dots M\}}$ can vary from trajectory to trajectory. Thus, a trajectory m is described by its initial condition $\boldsymbol{x}_m(0)$, and the observations $\boldsymbol{y}_m := [\boldsymbol{x}_m(t_{n,m}) + \boldsymbol{\epsilon}_{n,m}]_{n\in\{1\dots N_m\}}$ at times $\boldsymbol{t}_m := [t_{n,m}]_{n\in\{1\dots N_m\}}$, where the additive observation noise $\boldsymbol{\epsilon}_{n,m}$ is assumed to be drawn i.i.d. from a zero mean Gaussian, whose covariance is given by $\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} := \operatorname{diag}(\sigma_1^2, \dots, \sigma_K^2)$. We denote by $\mathcal{D}$ the dataset, consisting of M initial conditions $\boldsymbol{x}_m(0)$, observation times $\boldsymbol{t}_m$, and observations $\boldsymbol{y}_m$.

To model the unknown dynamical system, we choose a parametric Ansatz $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$. Depending on the amount of expert knowledge, this parameterization can follow a white-box, gray-box, or black-box methodology (Bohlin, 2006). In any case, the parametric form of $\boldsymbol{f}$ is fixed a priori (e.g., a neural network), and the key challenge is to infer a reasonable distribution over the parameters $\boldsymbol{\theta}$, conditioned on the data $\mathcal{D}$. For later tasks, we are particularly interested in the *predictive posterior state distribution*

$$p(\boldsymbol{x}_{\text{new}}(\boldsymbol{t}_{\text{new}})|\mathcal{D}, \boldsymbol{t}_{\text{new}}, \boldsymbol{x}_{\text{new}}(0)), \tag{2}$$

i.e., the posterior distribution of the states starting from a potentially unseen initial condition $\boldsymbol{x}_{\text{new}}(0)$ and evaluated at times $\boldsymbol{t}_{\text{new}}$. This posterior would then be used by the downstream or prediction tasks described in the introduction.

Bayesian Parameter Inference In the case of Bayesian parameter inference, an additional prior $p(\boldsymbol{\theta})$ is imposed on the parameters $\boldsymbol{\theta}$ so that the posterior distribution of Equation (2) can be inferred. Unfortunately, this distribution is not analytically tractable for most choices of $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$, which is especially true when we model $\boldsymbol{f}$ with a neural network. Formally, for fixed parameters $\boldsymbol{\theta}$, initial condition $\boldsymbol{x}(0)$ and observation time t, the likelihood of an observation $\boldsymbol{y}$ is given by

$$p(\boldsymbol{y}(t)|\boldsymbol{x}(0), t, \boldsymbol{\theta}, \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}) = \mathcal{N}\left(\boldsymbol{y}(t) \middle| \boldsymbol{x}(0) + \int_0^t \boldsymbol{f}(\boldsymbol{x}(\tau), \boldsymbol{\theta}) d\tau, \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}\right). \tag{3}$$

[1] Code will be released on github when the paper is published.

Using the fact that all noise realizations are independent, the expression (3) can be used to calculate the likelihood of all observations in $\mathcal{D}$. Most state-of-the-art parameter inference schemes use this fact to create samples $\hat{\boldsymbol{\theta}}_s$ of the posterior over parameters $p(\boldsymbol{\theta}|\mathcal{D})$ using various Monte Carlo methods. Given a new initial condition $\boldsymbol{x}(0)$ and observation time t, these samples $\hat{\boldsymbol{\theta}}_s$ can then be turned into samples of the predictive posterior state again by numerically integrating

$$\hat{\boldsymbol{x}}_s(t) = \boldsymbol{x}(0) + \int_0^t \boldsymbol{f}(\boldsymbol{x}(\tau), \hat{\boldsymbol{\theta}}_s) d\tau. \tag{4}$$

Clearly, both training (i.e., obtaining the samples $\hat{\boldsymbol{\theta}}_s$) and prediction (i.e., evaluating Equation (4)) require integrating the system dynamics $\boldsymbol{f}$ many times. Especially when we model $\boldsymbol{f}$ with a neural network, this can be a huge burden, both numerically and computationally (Kelly et al., 2020).

As an alternative approach, we can approximate the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ with variational inference (Bishop, 2006). However, we run into similar bottlenecks. While optimizing the variational objective, e.g., the ELBO, many integration steps are necessary to evaluate the unnormalized posterior. Also, at inference time, to obtain a distribution over state $\hat{\boldsymbol{x}}_s(t)$, we still need to integrate $\boldsymbol{f}$ several times. Furthermore, Dandekar et al. (2021) report poor forecasting performance by the variational approach.

3. Distributional Gradient Matching

In both the Monte Carlo sampling-based and variational approaches, all information about the dynamical system is stored in the estimates of the system parameters $\hat{\boldsymbol{\theta}}$. This makes these approaches rather cumbersome: Both for obtaining estimates of $\hat{\boldsymbol{\theta}}$ and for obtaining the predictive posterior over states, once $\hat{\boldsymbol{\theta}}$ is found, we need multiple rounds of numerically integrating a potentially complicated (neural) differential equation. We thus have identified two bottlenecks limiting the performance and applicability of these algorithms: namely, numerical integration of $\boldsymbol{f}$ and inference of the system parameters $\boldsymbol{\theta}$. In our proposed algorithm, we *avoid both* of these bottlenecks by directly working with the posterior distribution in the state space.

To this end, we introduce a probabilistic, differentiable *smoother model*, that directly maps a tuple $(t, \boldsymbol{x}(0))$ consisting of a time point t and an initial condition $\boldsymbol{x}(0))$ as input and maps it to the corresponding distribution over $\boldsymbol{x}(t)$. Thus, the smoother directly replaces the costly, numerical integration steps, needed, e.g., to evaluate Equation (2).

Albeit computationally attractive, this approach has one serious drawback. Since the smoother no longer explicitly integrates differential equations, there is no guarantee that the obtained smoother model follows any vector field. Thus, the smoother model is strictly more general than the systems described by Equation (1). Unlike ODEs, it is able to capture mappings whose underlying functions violate, e.g., Lipschitz or Markovianity properties, which is clearly not desirable. To address this issue, we introduce a regularization term, $\mathcal{L}_{\text{dynamics}}$, which ensures that a trajectory predicted by the smoother is encouraged to follow some underlying system of the form of Equation (1). The smoother is then trained with the multi-objective loss function

$$\mathcal{L} := \mathcal{L}_{\text{data}} + \lambda \cdot \mathcal{L}_{\text{dynamics}}, \tag{5}$$

where, $\mathcal{L}_{\text{data}}$ is a smother-dependent loss function that ensures a sufficiently accurate data fit, and λ is a trade-off parameter.

Regularization by Matching Distributions over Gradients To ultimately define $\mathcal{L}_{\text{dynamics}}$, first choose a parametric *dynamics model* similar to $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$ in Equation (3), that maps states to their derivatives. Second, define a set of *supporting points* $\mathcal{T}$ with the corresponding *supporting gradients* $\dot{\mathcal{X}}_{\text{supp}}$ as

$$\mathcal{T} := \left\{ \left(t_{\text{supp},l}, \boldsymbol{x}_{\text{supp},l}(0)\right)_{l \in \{1 \dots N_{\text{supp}}\}} \right\}, \quad \dot{\mathcal{X}}_{\text{supp}} := \left\{ \left(\dot{\boldsymbol{x}}_{\text{supp},l}\right)_{l \in \{1 \dots N_{\text{supp}}\}} \right\}.$$

Here, the l-th element represents the event that the dynamical system's derivative at time $t_{\text{supp},l}$ is $\dot{\boldsymbol{x}}_{\text{supp},l}$, after being initialized at time 0 at initial condition $\boldsymbol{x}_{\text{supp},l}(0)$.

Given both the smoother and the dynamics model, we have now two different ways to calculate distributions over $\dot{\mathcal{X}}$ given some data $\mathcal{D}$ and supporting points $\mathcal{T}$. First, we can directly leverage the differentiability and global nature of our smoother model to extract a distribution $p_S(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$ from the smoother model. Second, we can first use the smoother to obtain state estimates and then plug these state estimates into the dynamics model, to obtain a second distribution $p_D(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$. Clearly, if the solution proposed by the smoother follows the dynamics, these two distributions should match. Thus, we can regularize the smoother to follow the solution of Equation (3) by defining $\mathcal{L}_{\text{dynamics}}$ to encode the *distance* between $p_D(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$ and $p_S(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$ to be small in some metric. By minimizing the overall loss, we thus match the distributions over the gradients of the smoother and the dynamics model.

Smoothing jointly over Trajectories with Deep Gaussian Processes The core of DGM is formed by a smoother model. In principle, the posterior state distribution of Equation (2) could be modeled by any Bayesian regression technique. However, calculating $p_S(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$ is generally more involved. Here, the key challenge is evaluating this posterior, which is already computationally challenging, e.g., for simple Bayesian neural networks. For Gaussian processes, however, this becomes straightforward, since derivatives of GPs remain GPs (Solak et al., 2003). Thus, DGM uses a GP smoother. For scalability and simplicity, we keep K different, independent smoothers, one for each state dimension. However, if computational complexity is not a concern, our approach generalizes directly to multi-output Gaussian processes. Below, we focus on the one-dimensional case, for clarity of exposition. For notational compactness, all vectors with a superscript should be interpreted as vectors over time in this subsection. For example, the vector $\boldsymbol{x}^{(k)}$ consists of all the k-th elements of the state vectors $\boldsymbol{x}(t_{n,m}), n \in \{1, \dots, N_m\}, m \in \{1, \dots, M\}$.

Combining ideas from Rasmussen (2004) and Wilson et al. (2016), we define a Gaussian process with a differentiable mean function $\mu(\boldsymbol{z}_i)$ as well as a differentiable and positive-definite kernel function $\mathcal{K}(\boldsymbol{z}_i, \boldsymbol{z}_j)$. For the purpose of modeling our smoother model, we define $\boldsymbol{z}_i$ as being features obtained by mapping a tuple consisting of the initial condition and time, i.e. $\boldsymbol{z}_i := (\boldsymbol{x}_i(0), t_{n,i}) \in \mathbb{R}^{K+1}$, through a differentiable feature map $\boldsymbol{\phi}$ parametrized by a deep neural network. As observed by Solak et al. (2003), given fixed $\boldsymbol{x}_{\text{supp}}$, we can now calculate the joint density of $(\dot{\boldsymbol{x}}_{\text{supp}}^{(k)}, \boldsymbol{y}^{(k)})$. Let

$$\begin{aligned}
\boldsymbol{z}^{(k)} &:= \phi^{(k)}(\boldsymbol{x}^{(k)}(0), \boldsymbol{t}), \qquad \boldsymbol{z}_{\text{supp}}^{(k)} := \phi^{(k)}(\boldsymbol{x}_{\text{supp}}^{(k)}(0), \boldsymbol{t}_{\text{supp}}), \\
\boldsymbol{\mu}^{(k)} &:= \mu^{(k)}\left(\boldsymbol{x}^{(k)}(0), \boldsymbol{t}\right), \quad \dot{\boldsymbol{\mu}}^{(k)} := \frac{\partial}{\partial \boldsymbol{t}_{\text{supp}}} \mu^{(k)}\left(\boldsymbol{x}_{\text{supp}}^{(k)}(0), \boldsymbol{t}_{\text{supp}}\right),
\end{aligned}$$

$$\boldsymbol{\mathcal{K}}^{(k)} := \mathcal{K}_k(\boldsymbol{z}^{(k)}, \boldsymbol{z}^{(k)}), \quad \dot{\boldsymbol{\mathcal{K}}}^{(k)} := \frac{\partial}{\partial t_1} \mathcal{K}_k(\boldsymbol{z}_{\text{supp}}^{(k)}, \boldsymbol{z}^{(k)}), \quad \ddot{\boldsymbol{\mathcal{K}}}^{(k)} := \frac{\partial^2}{\partial t_1 \partial t_2} \mathcal{K}_k(\boldsymbol{z}_{\text{supp}}^{(k)}, \boldsymbol{z}_{\text{supp}}^{(k)}).$$

Then the joint density of $(\dot{\boldsymbol{x}}_{\text{supp}}^{(k)}, \boldsymbol{y}^{(k)})$ can be written as

$$\begin{pmatrix} \dot{\boldsymbol{x}}^{(k)}_{\text{supp}} \\ \boldsymbol{y}^{(k)} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \dot{\boldsymbol{\mu}}^{(k)} \\ \boldsymbol{\mu}^{(k)} \end{pmatrix}, \begin{pmatrix} \ddot{\boldsymbol{\mathcal{K}}}^{(k)} & \dot{\boldsymbol{\mathcal{K}}}^{(k)} \\ (\dot{\boldsymbol{\mathcal{K}}}^{(k)})^\top & \boldsymbol{\mathcal{K}}^{(k)} + \sigma_k^2 \boldsymbol{I} \end{pmatrix} \right). \tag{6}$$

Here we denote by $\frac{\partial}{\partial t_1}$ the partial derivative with respect to time in the first coordinate, by $\frac{\partial}{\partial t_2}$ the partial derivative with respect to time in the second coordinate, and with σ_k^2 the corresponding noise variance of $\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}$.

Since the conditionals of a joint Gaussian random variable are again Gaussian distributed, p_S is again Gaussian, i.e., $p_S(\dot{\mathcal{X}}_k|\mathcal{D}_k, \mathcal{T}_k) = \mathcal{N}\left(\dot{\boldsymbol{x}}^{(k)}_{\text{supp}}|\boldsymbol{\mu}_S, \boldsymbol{\Sigma}_S\right)$ with

$$\begin{aligned} \boldsymbol{\mu}_S &\coloneqq \dot{\boldsymbol{\mu}}^{(k)} + \dot{\boldsymbol{\mathcal{K}}}^{(k)}(\boldsymbol{\mathcal{K}}^{(k)} + \sigma_k^2\boldsymbol{I})^{-1}\left(\boldsymbol{y}^{(k)} - \boldsymbol{\mu}^{(k)}\right), \\ \boldsymbol{\Sigma}_S &\coloneqq \ddot{\boldsymbol{\mathcal{K}}}^{(k)} - \dot{\boldsymbol{\mathcal{K}}}^{(k)}(\boldsymbol{\mathcal{K}}^{(k)} + \sigma_k^2\boldsymbol{I})^{-1}(\dot{\boldsymbol{\mathcal{K}}}^{(k)})^\top. \end{aligned} \tag{7}$$

Here, the index k is used to highlight that this is just the distribution for one state dimension. To obtain the final $p_S(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$, we take the product over all state dimensions k.

To fit our model to the data, we minimize the negative marginal log likelihood of our observations, neglecting purely additive terms (Rasmussen, 2004), i.e.,

$$\mathcal{L}_{\text{data}} \coloneqq \frac{1}{2}\left(\boldsymbol{y}^{(k)} - \boldsymbol{\mu}^{(k)}\right)^T \left(\boldsymbol{\mathcal{K}}^{(k)} + \sigma_k^2\boldsymbol{I}\right)^{-1}\left(\boldsymbol{y}^{(k)} - \boldsymbol{\mu}^{(k)}\right) + \frac{1}{2}\operatorname{logdet}\left(\boldsymbol{\mathcal{K}}^{(k)} + \sigma_k^2\boldsymbol{I}\right) \tag{8}$$

Furthermore, the predictive posterior for a new point $x^{(k)}_{\text{test}}$ given time t_{test} and initial condition $x^{(k)}_{\text{test}}(0)$ has the closed form

$$p_S(x^{(k)}_{\text{test}}|\mathcal{D}, t_{\text{test}}, \boldsymbol{x}_{\text{test}}) = \mathcal{N}\left(x^{(k)}_{\text{test}}\middle|\mu^{(k)}_{\text{post}}, \sigma^2_{\text{post},k}\right), \tag{9}$$

where

$$\mu^{(k)}_{\text{post}} = \mu^{(k)}(\boldsymbol{x}_{\text{test}}(0), t_{\text{test}}) + \mathcal{K}(\boldsymbol{z}^{(k)}_{\text{test}}, \boldsymbol{z}^{(k)})^T(\boldsymbol{\mathcal{K}}^{(k)} + \sigma_k^2\boldsymbol{I})^{-1}\boldsymbol{y}^{(\boldsymbol{k})}, \tag{10}$$

$$\sigma^2_{\text{post},k} = \mathcal{K}(\boldsymbol{z}_{\text{test}}, \boldsymbol{z}_{\text{test}}) - \mathcal{K}(\boldsymbol{z}^{(k)}_{\text{test}}, \boldsymbol{z}^{(k)})^T(\boldsymbol{\mathcal{K}}^{(k)} + \sigma_k^2\boldsymbol{I})^{-1}\mathcal{K}(\boldsymbol{z}^{(k)}_{\text{test}}, \boldsymbol{z}^{(k)}). \tag{11}$$

Representing Uncertainty in the Dynamics Model via the Reparametrization Trick As described at the beginning of this section, a key bottleneck of standard Bayesian approaches is the potentially high dimensionality of the dynamics parameter vector $\boldsymbol{\theta}$. The same is true for our approach. If we were to keep track of the distributions over all parameters of our dynamics model, calculating $p_D(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$ quickly becomes infeasible.

However, especially in the case of modeling $\boldsymbol{f}$ with a neural network, the benefits of keeping distributions directly over $\boldsymbol{\theta}$ is unclear due to overparametrization. For both the downstream tasks and our training method, we are mainly interested in the distributions in the state space. Usually, the state space is significantly lower dimensional compared to the parameter space of $\boldsymbol{\theta}$. Furthermore, since the exact posterior state distributions are generally intractable, they normally have to be approximated anyways with simpler distributions for downstream tasks (Schulman et al., 2015; Houthooft et al., 2016; Berkenkamp et al., 2017). Thus, we change the parametrization of our dynamics model as follows. Instead of working directly with $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{\theta})$ and keeping a distribution over $\boldsymbol{\theta}$, we model uncertainty directly on the level of the vector field as

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{\psi}) + \boldsymbol{\Sigma}_D^{\frac{1}{2}}(\boldsymbol{x}(t), \boldsymbol{\psi})\boldsymbol{\epsilon}, \tag{12}$$

where $\epsilon \sim \mathcal{N}(0, \boldsymbol{I}_K)$ is drawn once per rollout (i.e., fixed within a trajectory) and $\boldsymbol{\Sigma}_D$ is a state-dependent and positive semi-definite matrix parametrized by a neural network. Here, $\boldsymbol{\psi}$ are the parameters of the new dynamics model, consisting of both the original parameters $\boldsymbol{\theta}$ and the weights of the neural network parametrizing $\boldsymbol{\Sigma}_D$. To keep the number of parameters reasonable, we employ a weight sharing scheme, detailed in Section 4.

In spirit, this modeling paradigm is very closely related to standard Bayesian training of NODEs. In both cases, the random distributions capture a distribution over a set of deterministic, ordinary differential equations. This should be seen in stark contrast to stochastic differential equations, where the randomness in the state space, i.e., diffusion, is modeled with a stochastic process. In comparison to (12), the latter is a time-varying disturbance added to the vector field. In that sense, our model still captures the *epistemic* uncertainty about our system dynamics, while an SDE model captures the intrinsic process noise, i.e., *aleatoric* uncertainty. While this reparametrization does not allow us to directly calculate $p_D(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$, we obtain a Gaussian distribution for the marginals $p_D(\dot{\boldsymbol{x}}_{\text{supp}}|\boldsymbol{x}_{\text{supp}})$. To retrieve $p_D(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$, we use the smoother model's predictive state posterior to obtain

$$
\begin{aligned}
p_D(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T}) &= \int p_D(\dot{\boldsymbol{x}}_{\text{supp}}, \boldsymbol{x}_{\text{supp}}|\mathcal{D}, \mathcal{T}) d\boldsymbol{x}_{\text{supp}} && (13)\\
&\approx \int p_D(\dot{\boldsymbol{x}}_{\text{supp}}|\boldsymbol{x}_{\text{supp}}) p_S(\boldsymbol{x}_{\text{supp}}|\mathcal{T}, \mathcal{D}) d\boldsymbol{x}_{\text{supp}}. && (14)
\end{aligned}
$$

Comparing Gradient Distributions via the Wasserstein Distance To compare and eventually match $p_D(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$ and $p_S(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$, we propose to use the Wasserstein distance (Kantorovich, 1939), since it allows for an analytic, closed-form representation, and since it outperforms similar measures (like forward, backward and symmetric KL divergence) in our exploratory experiments. The squared type-2 Wasserstein distance gives rise to the term

$$
\mathbb{W}_2^2\left[p_S(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T}), p_D(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})\right] = \mathbb{W}_2^2\left[p_S(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T}), \mathbb{E}_{\boldsymbol{x}_{\text{supp}} \sim p_{\text{GP}}(\boldsymbol{x}_{\text{supp}}|\mathcal{D}, \mathcal{T})}\left[p_D(\dot{\boldsymbol{x}}_{\text{supp}}|\boldsymbol{x}_{\text{supp}})\right]\right] \tag{15}
$$

that we will later use to regularize the smoothing process. To render the calculation of this regularizaton term computationally feasible, we introduce two approximations. First, observe that an exact calculation of the expectation in Equation (15) requires mapping a multivariate Gaussian through the deterministic neural networks parametrizing $\boldsymbol{f}$ and $\boldsymbol{\Sigma}_D$ in Equation (12). To avoid complex sampling schemes, we carry out a certainty-equivalence approximation of the expectation, that is, we evaluate the dynamics model on the posterior smoother mean $\boldsymbol{\mu}_{\text{S, supp}}$. As a result of this approximation, observe that both $p_D(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$ and $p_S(\dot{\mathcal{X}}|\mathcal{D}, \mathcal{T})$ become Gaussians. However, the covariance structure of these matrices is very different. Since we use indepdent GPs for different state dimensions, the smoother only models the covariance between the state values within the same dimension, across different time points. Furthermore, since $\boldsymbol{\epsilon}$, the random variable that captures the randomness of the dynamics across all time-points, is only K-dimensional, the covariance of p_D will be degenerate. Thus, we do not match the distributions directly, but instead match the marginals of each state coordinate at each time point independently at the different supporting time points. Hence, using first marginalization and then the certainty equivalence, Equation (15) reduces to

$$\begin{aligned}\mathbb{W}_2^2\left[p_S(\dot{\mathcal{X}}|\mathcal{D},\mathcal{T}), p_D(\dot{\mathcal{X}}|\mathcal{D},\mathcal{T})\right] &\approx \sum_{k=1}^{K}\sum_{i=1}^{|\dot{\mathcal{X}}|}\mathbb{W}_2^2\left[p_S(\dot{x}^{(k)}_{\text{supp}}(t_{\text{supp},i})|\mathcal{D},\mathcal{T}), p_D(\dot{x}^{(k)}_{\text{supp}}(t_{\text{supp},i})|\mathcal{D},\mathcal{T})\right] \\ &\approx \sum_{k=1}^{K}\sum_{i=1}^{|\dot{\mathcal{X}}|}\mathbb{W}_2^2\left[p_S(\dot{x}^{(k)}_{\text{supp}}(t_{\text{supp},i})|\mathcal{D},\mathcal{T}), p_D(\dot{x}^{(k)}_{\text{supp}}(t_{\text{supp},i})|\boldsymbol{\mu}_{\text{S, supp}})\right]. \qquad (16)\end{aligned}$$

Conveniently, the Wasserstein distance can now be calculated analytically, since for two one-dimensional Gaussians $a \sim \mathcal{N}(\mu_a, \sigma_a^2)$ and $b \sim \mathcal{N}(\mu_b, \sigma_b^2)$, we have $\mathbb{W}_2^2[a, b] = (\mu_a - \mu_b)^2 + (\sigma_a - \sigma_b)^2$.

Final Loss Function As explained in the previous paragraphs, distributional gradient matching trains a smoother regularized by a dynamics model. Both the parameters of the smoother $\boldsymbol{\varphi}$, consisting of the trainable parameters of the GP prior mean $\boldsymbol{\mu}$, the feature map ϕ, and the kernel $\mathcal{K}$, and the parameters of the dynamics model $\boldsymbol{\psi}$ are trained concurrently, using the same loss function. This loss consists of two terms, of which the regularization term was already described in Equation (16). While this term ensures that the smoother follows the dynamics, we need a second term ensuring that the smoother also follows the data. To this end, we follow standard GP regression literature, where it is common to learn the GP hyperparameters by maximizing the marginal log likelihood of the observations, i.e. $p_{GP}(\boldsymbol{y}|\boldsymbol{\varphi})$ (Rasmussen, 2004). Combining these terms, we obtain the final objective

$$\mathcal{L}(\boldsymbol{\varphi}, \boldsymbol{\psi}) := \log\left(p_{GP}(\boldsymbol{y}|\boldsymbol{\varphi})\right) - \lambda \cdot \sum_{k=1}^{K}\sum_{i=1}^{|\dot{\mathcal{X}}|}\mathbb{W}_2^2\left[p_S(\dot{x}^{(k)}_{\text{supp}}(t_{\text{supp},i})|\mathcal{D},\mathcal{T}), p_D(\dot{x}^{(k)}_{\text{supp}}(t_{\text{supp},i})|\boldsymbol{\mu}_{\text{S, supp}})\right].$$

This loss function is a multi-criteria objective, where fitting the data (via the smoother) and identifying the dynamics model (by matching the marginals) regularize another. In our preliminary experiments, we found the objective to be quite robust w.r.t. different choices of λ. In the interest of simplicity, we thus set it in all our experiments in Section 4 to a default value of $\lambda = \frac{|\mathcal{D}|}{|\dot{\mathcal{X}}|}$, accounting only for the possibility of having different numbers of supporting points and observations. One special case worth mentioning is $\lambda \to 0$, which corresponds to conventional sequential smoothing, where the second part would be used for identification in a second step, as proposed by Pillonetto and De Nicolao (2010). However, as can be seen in Figure 1, the smoother fails to properly identify the system without any knowledge about the dynamics and thus fails to provide meaningful state or derivative estimates. Thus, especially in the case of sparse observations, joint training is strictly superior.

In its final form, unlike its pure Bayesian counterparts, DGM does not require any prior knowledge about the system dynamics. Nevertheless, if some prior knowledge is available, one could add an additional, additive term $\log(p(\boldsymbol{\psi}))$ to $\mathcal{L}(\boldsymbol{\varphi}, \boldsymbol{\psi})$. It should be noted however that this was not done in any of our experiments, and excellent performance can be achieved without.

4. Experiments

We now compare DGM against state-of-the-art methods. In a first experiment, we demonstrate the effects of an overparametrized, simple dynamics model on the performance of DGM as well as traditional, MC-based algorithms SGLD and SGHMC. We select our baselines based on the results of Dandekar et al. (2021), who demonstrate that both a variational approach and NUTS are inferior to these two. Subsequently, we will investigate and benchmark the ability of DGM to correctly identify neural dynamics models and to generalize across different initial conditions. Since SGLD and SGHMC reach their computational limits in the generalization experiments, we compare against Neural ODE Processes (NDP). Lastly, we will conclude by demonstrating the necessity of all of its components. For all comparisons, we use the julia implementations of SGLD and SGHMC provided by Dandekar et al. (2021), the pytorch implementation of NDP provided by Norcliffe et al. (2021), and our own JAX (Bradbury et al., 2018) implementation of DGM.

Setup We use known parametric systems from the literature to generate simulated, noisy trajectories. For these benchmarks, we use the two-dimensional *Lotka Volterra (LV)* system, the three-dimensional, chaotic *Lorenz (LO)* system, a four-dimensional *double pendulum (DP)* and a twelve-dimensional *quadrocopter (QU)* model. For all systems, the exact equations and ground truth parameters are provided in the Appendix A. For each system, we create two different data sets. In the first, we include just one densely observed trajectory, taking the computational limitations of the benchmarks into consideration. In the second, we include many, but sparsely observed trajectories (5 for LV and DP, 10 for LO, 15 for QU). This setting aims to study generalization over different initial conditions.

Metric We use the log likelihood as a metric to compare the accuracy of our probabilistic models. In the 1-trajectory setting, we take a grid of 100 time points equidistantly on the training trajectory. We then calculate the ground truth and evaluate its likelihood using the predictive distributions of our models. When testing for generalization, we repeat the same procedure for unseen initial conditions.

Effects of Overparametrization We first study a three-dimensional, linear system of the form $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t)$, where $\boldsymbol{A}$ is a randomly chosen matrix with one stable and two marginally stable eigenvalues. For the dynamics model, we choose a linear Ansatz $\dot{\boldsymbol{x}}(t) = \boldsymbol{B}\boldsymbol{x}(t)$, where $\boldsymbol{B}$ is parametrized as the product of multiple matrices. The dimension of the matrices of each factorization are captured in a string of integers of the form $(3, a_1, \dots, a_J, 3)$. For example, $(3, 3)$ corresponds to $\boldsymbol{B}$ being just one matrix, while $(3, 6, 6, 3)$ corresponds to $\boldsymbol{B} = \boldsymbol{B}_1\boldsymbol{B}_2\boldsymbol{B}_3$, with $\boldsymbol{B}_1 \in \mathbb{R}^{3\times 6}$, $\boldsymbol{B}_2 \in \mathbb{R}^{6\times 6}$ and $\boldsymbol{B}_3 \in \mathbb{R}^{6\times 3}$. All of these models can be interpreted as linear neural networks, forming a simple case of the nonparametric systems we study later. Unlike general neural networks, the expressiveness of the Ansatz is independent of the number of parameters, allowing us to isolate the effects of overparametrization. In Figure 2, we show the mean and standard deviation of the log likelihood of the ground truth over 10 different noise realizations. While SGLD runs into numerical issues after a medium model complexity, the performance of SGHMC continuously disintegrates, while DGM is unaffected. This foreshadows the results of the next two experiments, where we observe that the MC-based approaches are not suitable for the more complicated settings.

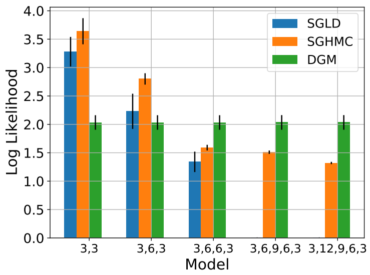

Figure 2: SGLD does not converge for strongly overparametrized models, the performance of SGHMC deteriorates. DGM is not noticeably affected.

Single Trajectory Benchmarks In Table 1, we evaluate the log-likelihood of the ground truth for the four benchmark systems, obtained when learning these systems using a neural ODE as a dynamics model (for more details, see appendix B). Clearly, DGM performs the best on all systems, even though we supplied both SGLD and SGHMC with very strong priors and fine-tuned them with

Table 1: Log likelihood and prediction times of 100 ground truth sample points, with mean and standard deviation taken over 10 independent noise realizations, for neural ODEs trained on a single, densely sampled trajectory.

	Log Likelihood			Prediction time [ms]		
	DGM	SGHMC	SGLD	DGM	SGHMC	SGLD
LV 1	**1.96 ± 0.21**	1.36 ± 0.0693	1.03 ± 0.0581	**0.68 ± 0.04**	14.98 ± 0.23	14.59 ± 0.15
LO 1	**−0.57 ± 0.11**	−3.02 ± 0.158	−2.67 ± 0.367	**0.99 ± 0.05**	98.93. ± 5.79	105.03 ± 12.22
DP 1	**2.13 ± 0.14**	1.88 ± 0.0506	1.85 ± 0.0501	**1.31 ± 0.05**	10.60 ± 0.21	11.34 ± 0.76
QU 1	**0.64 ± 0.07**	−5.00 ± 1.36	NaN	**3.76 ± 0.12**	24.68 ± 6.58	NaN

an extensive hyperparameter sweep (see Appendix C for more details). Despite this effort, we failed to get SGLD to work on Quadrocopter 1, where it always returned NaNs. This is in stark contrast to DGM, which performs reliably without any pre-training or priors.

Prediction speed To evaluate prediction speed, we consider the task of predicting 100 points on a previously unseen trajectory. To obtain a fair comparison, all algorithms' prediction routines were implemented in JAX (Bradbury et al., 2018). Furthermore, while we used 1000 MC samples when evaluating the predictive posterior for the log likelihood to guarantee maximal accuracy, we only used 200 samples in Table 1. Here, 200 was chosen as a minimal sample size guaranteeing reasonable accuracy, following a preliminary experiment visualized in Appendix C. Nevertheless, the predictions of DGM are 1-2 orders of magnitudes faster, as can be seen in Table 1. This further illustrates the advantage of relying on a smoother instead of costly, numerical integration to obtain predictive posteriors in the state space.

Table 2: Log likelihood of 100 ground truth sample points, with mean and covariance taken over 10 independent noise realizations, for neural ODEs trained on a multiple, sparsely sampled trajectory.

	Log Likelihood	
	DGM	NDP
LV 100	**1.81 ± 0.08**	0.62 ± 0.27
LO 125	**−2.18 ± 0.76**	−2.85 ± 0.05
DP 100	**1.86 ± 0.05**	0.88 ± 0.05
QU 64	**−0.54 ± 0.36**	−0.91 ± 0.07

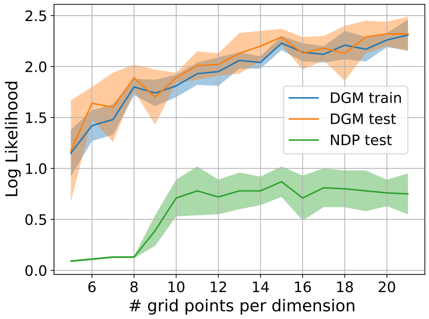

Figure 3: Log likelihood of the ground truth for Lotka Volterra for increasing number of trajectories with 5 observations each.

Multi-Trajectory Benchmarks Next, we take a set of trajectories starting on an equidistant grid of the initial conditions. Each trajectory is then observed at 5 equidistant observation times for LV and DP, and 10 equidistant observation times for the chaotic Lorenz and more complicated Quadrocopter. We test generalization by randomly sampling a new initial condition and evaluating the negative log likelihood of the ground truth at 100 equidistant time points. In Table 2, we compare the generalization performance of DGM against NDP, since despite serious tuning efforts, the MC methods failed to produce meaningful results in this setting. DGM clearly outperforms NDP, a fact which is further exemplified in Figure 3. There, we show the test log likeliood for Lotka Volterra trained on an increasing set of trajectories. Even though the time grid is fixed and we only decrease the distance between initial condition samples, the dynamics model helps the smoother to generalize across time as well. In stark contrast, NDP fails to improve with increasing data after an initial jump.

Ablation study We next study the importance of different elements of our approach via an ablation study on the Lorenz 125 dataset, shown in Figure 1. Comparing the two rows, we see that joint smoothing across trajectories is essential to transfer knowledge between different training trajectories. Similarly, comparing the two columns, we see that the dynamics model enables the smoother to reduce its uncertainty in between observation points.

Computational Requirements For the one trajectory setting, all DGM related experiments were run on a Nvidia RTX 2080 Ti, where the longest ones took 15 minutes. The comparison methods were given 24h, on Intel Xeon Gold 6140 CPUs. For the multi-trajectory setting, we used Nvidia Titan RTX, where all experiments finished in less than 3 hours. A more detailed run time compilation can be found in Appendix B. Using careful implementation, the run time of DGM scales linearly in the number of dimensions K. However, since we use an accurate RBF kernel for all our experiments reported in this section, we have cubic run time complexity in $\sum_{m=1}^{M} N_m$. In principle, this can be alleviated by deploying standard feature approximation methods (Rahimi et al., 2007; Liu et al., 2020). While this is a well known fact, we nevertheless refer the interested reader to a more detailed discussion of the subject in Appendix D.

5. Related work

Bayesian Parameter Inference with Gaussian Processes The idea of matching gradients of a (spline-based) smoother and a dynamics model goes back to the work of Varah (1982). For GPs, this idea is introduced by Calderhead et al. (2009), who first fit a GP to the data and then match the parameters of the dynamics. Dondelinger et al. (2013) introduce concurrent training, while Gorbach et al. (2017) introduce an efficient variational inference procedure for systems with a locally-linear parametric form. All these works claim to match the distributions of the gradients of the smoother and dynamics models, by relying on a product of experts heuristics. However, Wenk et al. (2019) demonstrate that this product of experts in fact leads to statistical independence between the observations and the dynamics parameters, and that these algorithms essentially match *point estimates* of the gradients instead. Thus, DGM is the first algorithm to actually match gradients on the level of distributions for ODEs. In the context of stochastic differential equations (SDEs) with constant diffusion terms, Abbati et al. (2019) deploy MMD and GANs to match their gradient distributions. However, it should be noted that their algorithm treats the parameters of the dynamics model *deterministically* and thus, they can not provide the epistemic uncertainty estimates that we seek here. Note that our work is *not* related to the growing literature investigating SDE approximations of Bayesian Neural ODEs in the context of classification (Xu et al., 2021). Similarly to Chen et al. (2018), these works emphasize learning a terminal state of the ODE used for other downstream tasks.

Gaussian Processes with Operator Constraints Gradient matching approaches mainly use the smoother as a proxy to infer dynamics parameters. This is in stark contrast to our work, where we treat the smoother as the main model used for prediction. While the regularizing properties of the dynamics on the smoother are explored by Wenk et al. (2020), Jidling et al. (2017) introduce an algorithm to incorporate linear operator constraints directly on the kernel level. Unlike in our work, they can provide strong guarantees that the posterior always follows these constraints. However, it remains unclear how to generalize their approach to the case of complex, nonlinear operators, potentially parametrized by neural dynamics models.

Other Related Approaches In some sense, the smoother is mimicking a probabilistic numerical integration step, but without explicitly integrating. In spirit, this approach is similar to the solution networks used in the context of PDEs, as presented by Raissi et al. (2019), which however typically disregard uncertainty. In the context of classical ODE parameter inference, Kersting et al. (2020) deploy a GP to directly mimic a numerical integrator in a probabilistic, differentiable manner. Albeit promising in a classical, parametric ODE setting, it remains unclear how these methods can be scaled up, as there is still the numerical integration bottleneck. Unrelated to their work, Ghosh et al. (2021) present a variational inference scheme in the same, classical ODE setting. However, they still keep distributions over all weights of the neural network (Norcliffe et al., 2021). A similar approach is investigated by Dandekar et al. (2021), who found it to be inferior to the MC methods we use as a benchmark. Variational inference was previously employed by Yildiz et al. (2019) in the context of latent neural ODEs parametrized by a Bayesian neural network, but their work mainly focuses on dimensionality reduction. Nevertheless, their work inspired a model called Neural ODE Processes by Norcliffe et al. (2021). This work is similar to ours in the sense that it avoids keeping distributions over network weights and models an ensemble of deterministic ODEs via a global context variable. Consequently, we use it as a benchmark in Section 4, showing that it does not properly capture epistemic uncertainty in a low data setting, which might be problematic for downstream tasks like reinforcement learning.

6. Conclusion

In this work, we introduced a novel, GP-based collocation method, that matches gradients of a smoother and a dynamics model on the distribution level using a Wasserstein loss. Through careful parametrization of the dynamics model, we manage to train complicated, neural ODE models, where state of the art methods struggle. We then demonstrate that these models are able to accurately predict unseen trajectories, while capturing epistemic uncertainty relevant for downstream tasks. In future work, we are excited to see how our training regime can be leveraged in the context of active learning of Bayesian neural ordinary differential equation for continuous-time reinforcement learning.

Acknowledgments

This research was supported by the Max Planck ETH Center for Learning Systems. This project has received funding from the European Research Council(ERC) under the European Union's Horizon 2020 research and innovation programme grant agreement No 815943 as well as from the Swiss National Science Foundation under NCCR Automation, grant agreement 51NF40_180545.

References

Gabriele Abbati, Philippe Wenk, Michael A Osborne, Andreas Krause, Bernhard Schölkopf, and Stefan Bauer. Ares and mars adversarial and mmd-minimizing regression for sdes. In *International Conference on Machine Learning*, pages 1–10. PMLR, 2019.

Emmanouil Angelis, Philippe Wenk, Bernhard Schölkopf, Stefan Bauer, and Andreas Krause. Sleipnir: Deterministic and provably accurate feature expansion for gaussian process regression with derivatives. *arXiv preprint arXiv:2003.02658*, 2020.

Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/766ebcd59621e305170616ba3d3dac32-Paper.pdf.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. doi: 10.1137/141000671. URL https://epubs.siam.org/doi/10.1137/141000671.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Torsten P Bohlin. *Practical grey-box process identification: theory and applications*. Springer Science & Business Media, 2006.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

Ben Calderhead, Mark Girolami, and Neil D Lawrence. Accelerating bayesian inference over nonlinear differential equations with gaussian processes. In *Advances in neural information processing systems*, pages 217–224. Citeseer, 2009.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

Raj Dandekar, Karen Chung, Vaibhav Dixit, Mohamed Tarek, Aslan Garcia-Valadez, Krishna Vishal Vemula, and Chris Rackauckas. Bayesian neural ordinary differential equations. *Symposium on Principles of Programming Languages, POPL*, 2021.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

Frank Dondelinger, Dirk Husmeier, Simon Rogers, and Maurizio Filippone. Ode parameter inference using adaptive gradient matching with gaussian processes. In *Artificial intelligence and statistics*, pages 216–228. PMLR, 2013.

Alessandro Fanfarillo, Behrooz Roozitalab, Weiming Hu, and Guido Cervone. Probabilistic forecasting using deep generative models. *GeoInformatica*, 25(1):127–147, 2021.

Sanmitra Ghosh, Paul Birrell, and Daniela De Angelis. Variational inference for nonlinear ordinary differential equations. In *International Conference on Artificial Intelligence and Statistics*, pages 2719–2727. PMLR, 2021.

Nico S Gorbach, Stefan Bauer, and Joachim M Buhmann. Scalable variational inference for dynamical systems. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/e71e5cd119bbc5797164fb0cd7fd94a4-Paper.pdf`.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/26cd8ecadce0d4efd6cc8a8725cbd1f8-Paper.pdf`.

Håkan Hjalmarsson. From experiment design to closed-loop control. *Automatica*, 41(3):393–438, 2005.

Rein Houthooft, Xi Chen, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper/2016/file/abd815286ba1007abfbb8415b83ae2cf-Paper.pdf`.

Brian P Ingalls. *Mathematical modeling in systems biology: an introduction*. MIT press, 2013.

Carl Jidling, Niklas Wahlström, Adrian Wills, and Thomas B Schön. Linearly constrained gaussian processes. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/71ad16ad2c4d81f348082ff6c4b20768-Paper.pdf`.

Douglas Samuel Jones, Michael Plank, and Brian D Sleeman. *Differential equations and mathematical biology*. CRC press, 2009.

Leonid V Kantorovich. The mathematical method of production planning and organization. *Management Science*, 6(4):363–422, 1939.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

Jacob Kelly, Jesse Bettencourt, Matthew J Johnson, and David K Duvenaud. Learning differential equations that are easy to solve. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4370–4380. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/2e255d2d6bf9bb33030246d31f1a79ca-Paper.pdf.

Hans Kersting, Nicholas Krämer, Martin Schiegg, Christian Daniel, Michael Tiemann, and Philipp Hennig. Differentiable likelihoods for fast inversion of'likelihood-free'dynamical systems. In *International Conference on Machine Learning*, pages 5198–5208. PMLR, 2020.

Patrick Kidger, Ricky TQ Chen, and Terry Lyons. " hey, that's not an ode": Faster ode adjoints with 12 lines of code. *arXiv preprint arXiv:2009.09457*, 2020.

Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When gaussian process meets big data: A review of scalable gps. *IEEE transactions on neural networks and learning systems*, 31(11): 4405–4423, 2020.

Lennart Ljung. *System Identification*, pages 163–173. Birkhäuser Boston, Boston, MA, 1998. ISBN 978-1-4612-1768-8. doi: 10.1007/978-1-4612-1768-8_11. URL https://doi.org/10.1007/978-1-4612-1768-8_11.

Lennart Ljung. *Model validation and model error modeling*. Linköping University Electronic Press, 1999.

Alexander Norcliffe, Cristian Bodnar, Ben Day, Jacob Moss, and Pietro Liò. Neural {ode} processes. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=27acGyyI1BY.

Romeo Ortega, Arjan Van Der Schaft, Bernhard Maschke, and Gerardo Escobar. Interconnection and damping assignment passivity-based control of port-controlled hamiltonian systems. *Automatica*, 38(4):585–596, 2002.

Romeo Ortega, Laurent Praly, Stanislav Aranovskiy, Bowen Yi, and Weidong Zhang. On dynamic regressor extension and mixing parameter estimators: Two luenberger observers interpretations. *Automatica*, 95:548–551, 2018.

Gianluigi Pillonetto and Giuseppe De Nicolao. A new kernel-based approach for linear system identification. *Automatica*, 46(1):81–93, 2010.

Gianluigi Pillonetto, Francesco Dinuzzo, Tianshi Chen, Giuseppe De Nicolao, and Lennart Ljung. Kernel methods in system identification, machine learning and function estimation: A survey. *Automatica*, 50(3):657–682, 2014.

Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

Ali Rahimi, Benjamin Recht, et al. Random features for large-scale kernel machines. In *NIPS*, volume 3, page 5. Citeseer, 2007.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Carl Edward Rasmussen. *Gaussian Processes in Machine Learning*, pages 63–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-28650-9. doi: 10.1007/978-3-540-28650-9_4. URL `https://doi.org/10.1007/978-3-540-28650-9_4`.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

Ercan Solak, Roderick Murray-Smith, William E Leithead, Douglas J Leith, and Carl Edward Rasmussen. Derivative observations in gaussian process models of dynamic systems. 2003.

Mark W Spong, Seth Hutchinson, Mathukumalli Vidyasagar, et al. *Robot modeling and control*, volume 3. wiley New York, 2006.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 1015–1022, 2010.

James M Varah. A spline least squares method for numerical parameter estimation in differential equations. *SIAM Journal on Scientific and Statistical Computing*, 3(1):28–46, 1982.

Philippe Wenk, Alkis Gotovos, Stefan Bauer, Nico S Gorbach, Andreas Krause, and Joachim M Buhmann. Fast gaussian process based gradient matching for parameter identification in systems of nonlinear odes. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1351–1360. PMLR, 2019.

Philippe Wenk, Gabriele Abbati, Michael A Osborne, Bernhard Schölkopf, Andreas Krause, and Stefan Bauer. Odin: Ode-informed regression for parameter and state inference in time-continuous dynamical systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6364–6371, 2020.

Patrick M Wensing, Sangbae Kim, and Jean-Jacques E Slotine. Linear matrix inequalities for physically consistent inertial parameter identification: A statistical perspective on the mass distribution. *IEEE Robotics and Automation Letters*, 3(1):60–67, 2017.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016.

Jens Wittenburg. *Dynamics of systems of rigid bodies*, volume 33. Springer-Verlag, 2013.

Winnie Xu, Ricky TQ Chen, Xuechen Li, and David Duvenaud. Infinitely deep bayesian neural networks with stochastic differential equations. *arXiv preprint arXiv:2102.06559*, 2021.

Cagatay Yildiz, Markus Heinonen, and Harri Lahdesmaki. Ode2vae: Deep generative second order odes with bayesian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/99a401435dcb65c4008d3ad22c8cdad0-Paper.pdf`.

Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ode. In *International Conference on Machine Learning*, pages 11639–11649. PMLR, 2020.

Juntang Zhuang, Nicha C Dvornek, Sekhar Tatikonda, and James S Duncan. Mali: A memory efficient and reverse accurate integrator for neural odes. *arXiv preprint arXiv:2102.04668*, 2021.

Appendix A. Dataset description

In this section, we describe the datasets we use in our experiments.

A.1 Lotka Volterra

The two dimensional Lotka Volterra system is governed by the parametric differential equations

$$\frac{dx}{dt} = \alpha x - \beta xy$$
$$\frac{dy}{dt} = \delta xy - \gamma y,$$

where we selected $(\alpha, \beta, \gamma, \delta) = (1, 1, 1, 1)$. These equations were numerically integrated to obtain a ground truth, where the initial conditions and observation times depend on the dataset. All observations were then created by adding additive, i.i.d. noise, distributed according to a normal distribution $\mathcal{N}\left(0, 0.1^2\right)$.

LV 1 consists of one trajectory starting from initial condition $(1, 2)$. The trajectory is observed at 100 equidistant time points from the interval $(0, 10)$.

LV 100 consists of 100 trajectories. Initial conditions for these trajectories are located on a grid, i.e.,

$$\left\{ \left(\frac{1}{2} + \frac{i}{9}, \frac{1}{2} + \frac{j}{9} \right) \middle| i \in \{0, \ldots, 9\}, j \in \{0, \ldots, 9\} \right\}.$$

Each trajectory is then observed at 5 equidistant time points from the interval $(0, 10)$, which leads to a total of 500 observations.

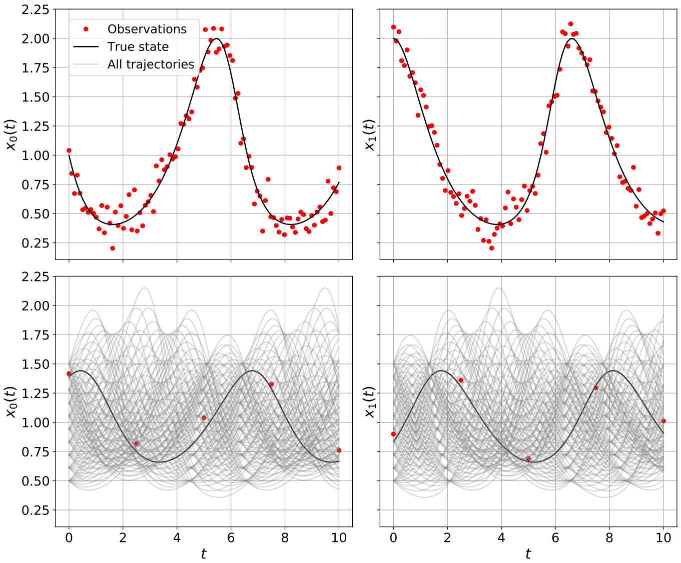

Figure 4: The first row represents the true states and all observations of LV 1 with random seed 0. In the second row we plot all ground truth trajectories from the dataset LV 100. One particular trajectory is highlighted in black, together with the corresponding observations of that trajectory (red dots).

To test generalization, we created 10 new trajectories. The initial conditions of these trajectories were obtained by sampling uniformly at random on $[0.5, 1.5]^2$. To evaluate the log likelihood, we used 100 equidistant time points from the interval $(0, 10)$.

A.2 Lorenz

The 3 dimensional, chaotic Lorenz system is governed by the parametric differential equations

$$\begin{aligned}\frac{dx}{dt} &= \sigma(y - x)\\ \frac{dy}{dt} &= x(\rho - z) - y\\ \frac{dz}{dt} &= xy - \tau y,\end{aligned}$$

where we selected $(\sigma, \rho, \tau) = (10, 28, 8/3)$. These equations were numerically integrated to obtain a ground truth, where the initial conditions and observation times depend on the dataset. All observations were then created by adding additive, i.i.d. noise, distributed according to a normal distribution $\mathcal{N}(0, 1)$.

LO 1 consists of one trajectory starting from initial condition $(-2.5, 2.5, 2.5)$. The trajectory is observed at 100 equidistant time points from the interval $(0, 1)$.

LO 125 consists of 125 trajectories. Initial conditions for these trajectories are located on a grid, i.e.,

$$\left\{\left(-5 + \frac{5i}{2}, -5 + \frac{5j}{2}, -5 + \frac{5k}{2}\right) \middle| i \in \{0, \ldots, 4\}, j \in \{0, \ldots, 4\}, k \in \{0, \ldots, 4\}\right\}.$$

Each trajectory is then observed on 10 equidistant time points from the interval $(0, 1)$, which leads to a total of 1250 observations.

To test generalization, we created 10 new trajectories. The initial conditions of these trajectories were obtained by sampling uniformly at random on $[-5, 5]^3$. To evaluate the log likelihood, we used 100 equidistant time points from the interval $(0, 1)$.

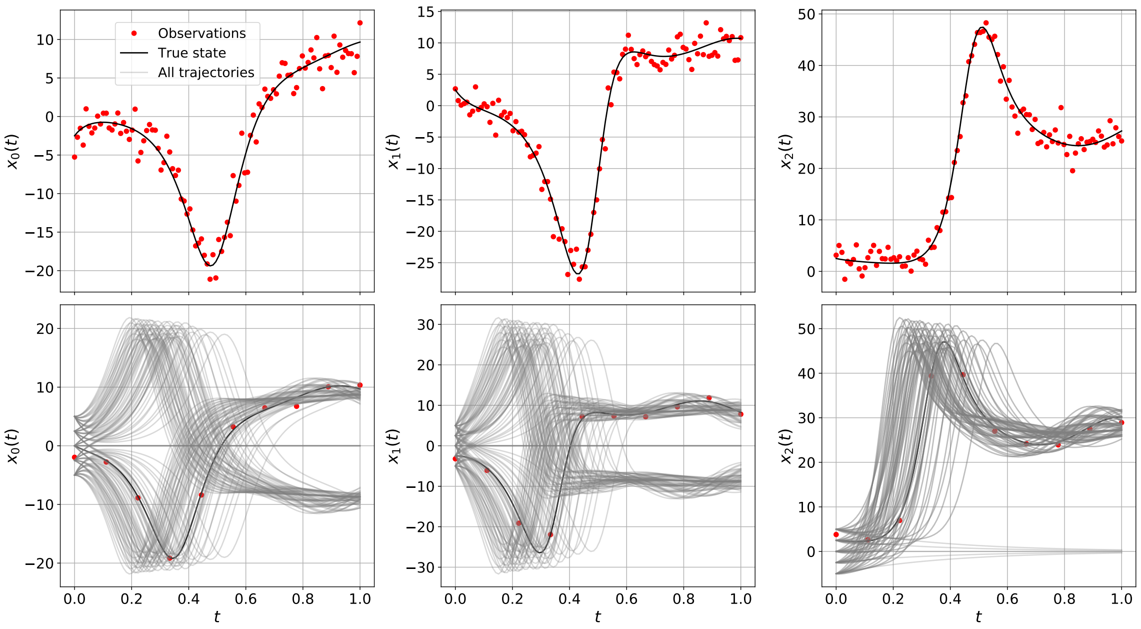

Figure 5: The first row represents the true states and all observations of LO 1 with random seed 0. In the second row we plot all ground truth trajectories from the dataset LO 125. One particular trajectory is highlighted in black, together with the corresponding observations of that trajectory (red dots).

A.3 Double Pendulum

The 4 dimensional Double pendulum system is governed by the parametric differential equations

$$
\begin{aligned}
\dot{\theta}_1 &= \frac{6}{ml^2}\frac{2p_{\theta_1} - 3\cos(\theta_1 - \theta_2)p_{\theta_2}}{16 - 9\cos^2(\theta_1 - \theta_2)} \\
\dot{\theta}_2 &= \frac{6}{ml^2}\frac{8p_{\theta_2} - 3\cos(\theta_1 - \theta_2)p_{\theta_1}}{16 - 9\cos^2(\theta_1 - \theta_2)}. \\
\dot{p}_{\theta_1} &= -\tfrac{1}{2}ml^2\left(\dot{\theta}_1\dot{\theta}_2\sin(\theta_1 - \theta_2) + 3\frac{g}{l}\sin\theta_1\right) \\
\dot{p}_{\theta_2} &= -\tfrac{1}{2}ml^2\left(-\dot{\theta}_1\dot{\theta}_2\sin(\theta_1 - \theta_2) + \frac{g}{l}\sin\theta_2\right),
\end{aligned}
$$

where we selected $(g, m, l) = (9.81, 1, 1)$. In these equations, θ_1 and θ_2 represent the offset angles, while p_{θ_1} and p_{θ_1} represent the momentum of the upper and lower pendulum. These equations were numerically integrated to obtain a ground truth, where the initial conditions and observation times depend on the dataset. All observations were then created by adding additive, i.i.d. noise, distributed according to a normal distribution $\mathcal{N}\left(0, 0.1^2\right)$.

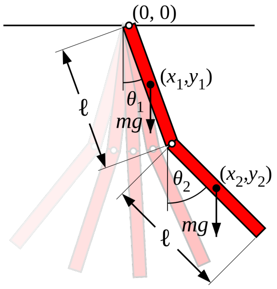

Figure 6: Double Pendulum where both rods have equal length and mass.

DP 1 consist of one trajectory starting from initial condition $(-\pi/6, -\pi/6, 0, 0)$. The trajectory is observed at 100 equidistant time points from the interval $(0, 1)$.

DP 100 consists of 100 trajectories. Initial conditions for these trajectories are located on a grid, i.e.,

$$\left\{ \left(-\frac{\pi}{6} + \frac{\pi i}{27}, -\frac{\pi}{6} + \frac{\pi j}{27}, 0, 0 \right) \middle| i \in \{0, \dots, 9\}, j \in \{0, \dots, 9\} \right\}.$$

Each trajectory is then observed at 5 equidistant time points from the interval $(0, 1)$, which leads to a total of 500 observations.

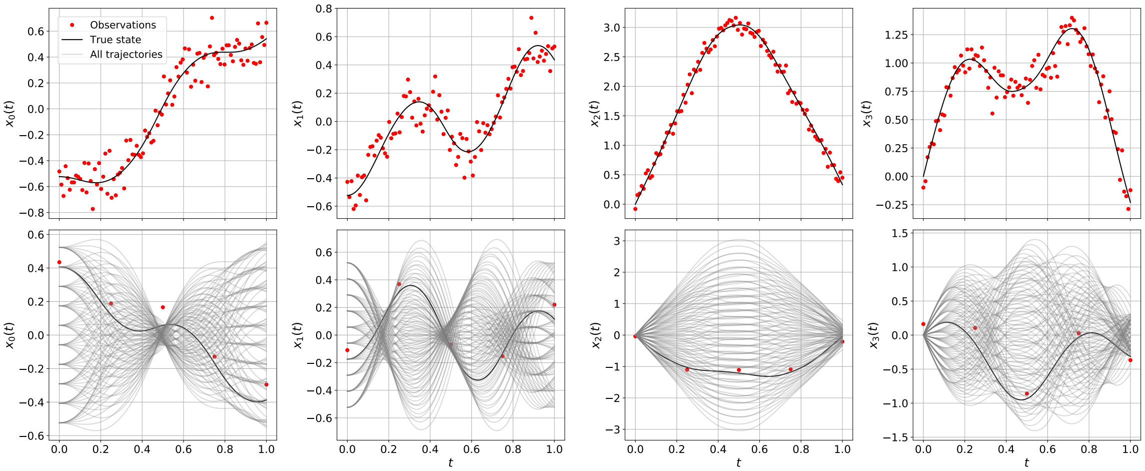

Figure 7: The first row represents the true states and all observations of DP 1 with random seed 0. In the second row we plot all ground truth trajectories from the dataset DP 100. One particular trajectory is highlighted in black, together with the corresponding observations of that trajectory (red dots).

To test generalization, we created 10 new trajectories. The initial conditions of these trajectories were obtained by sampling uniformly at random on $[-\frac{\pi}{6}, \frac{\pi}{6}]^2 \times \{0\}^2$. To evaluate the log likelihood, we used 100 equidistant time points from the interval $(0, 1)$.

A.4 Quadrocopter

The 12 dimensional Quadrocopter system is governed by the parametric differential equations

$$\begin{aligned}
\dot{u} &= -g\sin(\theta) + rv - qw \\
\dot{v} &= g\sin(\phi)\cos(\theta) - ru + pw \\
\dot{w} &= -F_z/m + g\cos(\phi)\cos(\theta) + qu - pv \\
\dot{p} &= \left(L + (I_{yy} - I_{zz})qr\right)/I_{xx} \\
\dot{q} &= \left(M + (I_{zz} - I_{xx})pr\right)/I_{yy} \\
\dot{r} &= (I_{xx} - I_{yy})pq/I_{zz} \\
\dot{\phi} &= p + (q\sin(\phi) + r\cos(\phi))\tan(\theta) \\
\dot{\theta} &= q\cos(\phi) - r\sin(\phi) \\
\dot{\psi} &= (q\sin(\phi) + r\cos(\phi))\sec(\theta) \\
\dot{x} &= \cos(\theta)\cos(\psi)u + (-\cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi))v + \\
&\quad + (\sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\phi))w \\
\dot{y} &= \cos(\theta)\sin(\psi)u + (\cos(\phi)\cos(\psi) + \sin(\phi)\sin(\theta)\sin(\psi))v + \\
&\quad + (-\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\phi))w \\
\dot{z} &= \sin(\theta)u - \sin(\phi)\cos(\theta)v - \cos(\phi)\cos(\theta)w,
\end{aligned}$$

where

$$\begin{aligned}
F_z &= F_1 + F_2 + F_3 + F_4 \\
L &= (F_2 + F_3)d_y - (F_1 + F_4)d_x \\
M &= (F_1 + F_3)d_x - (F_2 + F_4)d_x.
\end{aligned}$$

We fixed the input control forces to $(F_1, F_2, F_3, F_4) = (0.496, 0.495, 0.4955, 0.4955)$ (not to be inferred) and selected $(m, I_{xx}, I_{yy}, I_{zz}, d_x, d_y, g) = (0.1, 0.62, 1.13, 0.9, 0.114, 0.0825, 9.85)$. These equations were numerically integrated to obtain a ground truth, where the initial conditions and observation times depend on the dataset. All observations were then created by adding additive, i.i.d. noise, distributed according to a normal distribution $\mathcal{N}(0, \Sigma)$, where

$$\Sigma = \mathrm{diag}(1, 1, 1, 0.1, 0.1, 0.1, 1, 0.1, 0.1, 5, 5, 5).$$

QU 1 consists of one trajectory starting from initial condition $(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$. The trajectory is observed at 100 equidistant time points from the interval $(0, 10)$.

QU 64 consists of 64 trajectories. Initial conditions for these trajectories are located on a grid, i.e.,

$$\left\{ \left(0, 0, 0, 0, 0, 0, -\frac{\pi}{18} + \frac{\pi i}{27}, -\frac{\pi}{18} + \frac{\pi j}{27}, -\frac{\pi}{18} + \frac{\pi k}{27}, 0, 0, 0\right) \middle| (i, j, k) \in \{0, \dots, 4\}^3 \right\}.$$

Each trajectory is then observed at 15 equidistant time points from the interval $(0, 10)$, which leads to a total of 960 observations.

To test generalization, we created 10 new trajectories. The initial conditions of these trajectories were obtained by sampling uniformly at random on $\{0\}^6 \times [-\frac{\pi}{18}, \frac{\pi}{18}]^2 \times \{0\}^3$. To evaluate the log likelihood, we used 100 equidistant time points from the interval $(0, 10)$.

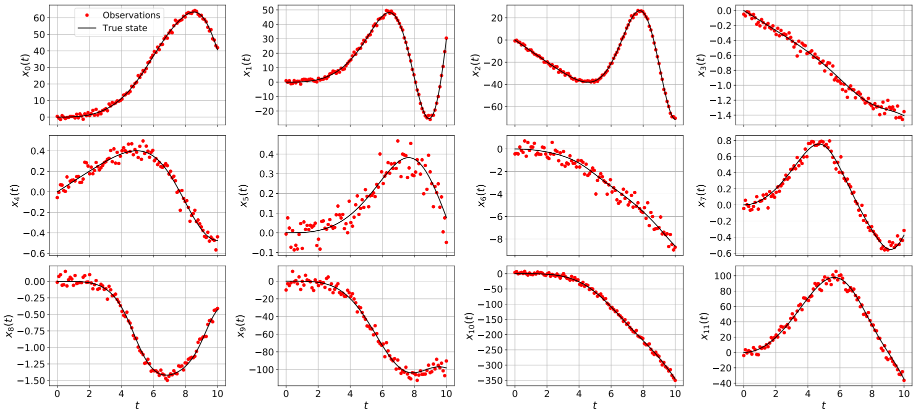

Figure 8: Visualization showing the true states and all observations of QU 1 with random seed 0.

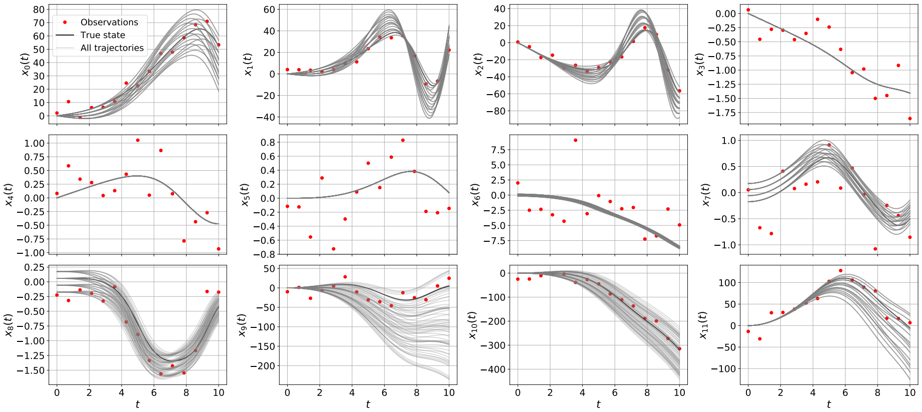

Figure 9: Visualization showing all ground truth trajectories from the dataset QU 64. One particular trajectory is highlighted in black, together with the corresponding observations of that trajectory (red dots).

Appendix B. Implementation details of DGM

In this section we discuss all implementation details of DGM. As described in Section 3, DGM consists of a smoother and a dynamics model. At training time, the dynamics model is then used to regularize the smoother via the squared type-2 Wasserstein distance.

B.1 Dynamics model

The role of the dynamics model is to map every state $\boldsymbol{x}$ to a distribution over derivatives, which we decide to parameterize as $\mathcal{N}\left(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\psi}), \boldsymbol{\Sigma}_D(\boldsymbol{x}, \boldsymbol{\psi})\right)$. In the paper we focused in the case where we do not have any prior knowledge about the dynamics and we model both $\boldsymbol{f}$ and $\boldsymbol{\Sigma}_D$ with neural network. Nevertheless, if some prior knowledge about the dynamics $\boldsymbol{f}$ is available, it can be used to inform the mean function $\boldsymbol{f}$ (potentially also covariance $\boldsymbol{\Sigma}_D$) appropriately. In particular, if the parametric form of the dynamics is known, we can use them as a direct substitute for $\boldsymbol{f}$. This is the special case of ODE-parameter inference, which we investigate empirically in Appendix E. Next we present implementation details of both non-parametric (i.e. $\boldsymbol{f}$ given by a neural network) and parametric (i.e. $\boldsymbol{f}$ given by some parametric form) dynamics models.

Non-parametric dynamics In the non-parametric case, we model both the dynamics' mean function $\boldsymbol{\mu}_D$ and covariance function $\boldsymbol{\Sigma}_D$ with a neural networks. Across all experiments, we choose a simple 3-layer neural network with 20, 20 and $2n$ nodes, where n denotes the number of state dimensions of a specific system. After each layer, we apply the sigmoid activation function, except for the last one. The first n output nodes represent the mean function. Here, no activation function is necessary for the last layer. The second n output nodes are used to construct the diagonal covariance $\boldsymbol{\Sigma}_D$. To ensure positivity, we use $x \mapsto \log(1 + \exp(x))^2$ as an activation function on the last n nodes of the last layer.

Parametric dynamics In the parametric case, we model $\boldsymbol{\mu}_D$ using the parametric form of the vector field. Across all experiments, we choose a simple 3-layer neural network with 10, 10 and n nodes, where n denotes the number of state dimensions of a specific system. After each lyer, we apply the sigmoid activation function, except for the last one. The n nodes are then used to construct the diagonal covariance $\boldsymbol{\Sigma}_D$. To ensure positivity, we use $x \mapsto \log(1 + \exp(x))^2$ as an activation function on the last layer.

B.2 Smoother model

The role of the smoother model is to map every tuple $(\boldsymbol{x}(0), t)$ consisting of initial condition $\boldsymbol{x}(0)$ and time t to $\boldsymbol{x}(t)$, which is the state at time t of a trajectory starting at $\boldsymbol{x}(0)$ at time 0. In the paper, we model the smoother using a Gaussian process with a deep mean function $\boldsymbol{\mu}$ and a deep feature map ϕ. Both of them take as input the tuple $(\boldsymbol{x}(0), t)$. This tuple is then mapped through a dense neural network we call core. For all experiments, we chose a core with two layers, with 10 and 5 hidden nodes and sigmoid activation on both. The output of the core is then fed into two linear heads. The head for $\boldsymbol{\mu}$ builds a linear combination of the core's output to obtain a vector of the same shape as $\boldsymbol{x}(t)$. The head for ϕ builds a linear combination of the core's output to obtain a vector of length 3, the so called features. These features are then used as inputs to a standard RBF kernel with ARD (Rasmussen, 2004). For each state dimension, we keep a separate ϕ-head, as well as separate kernel hyperparameters. However, the core is shared across dimensions, while $\boldsymbol{\mu}$ is directly introduced as multidimensional.

In the paper, we set the variance of the RBF to 1 and learned the lengthscales together with all other hyperparameters. However, due to the expressiveness of the neural network, the lengthscales are redundant and could easily be incorporated into the linear combination performed by the head. Thus, in the scaling experiments, we fix the lengthscale to one and approximate the RBF kernel with a feature expansion, as detailed in Appendix D.

B.3 Evaluation metric

To evaluate the quality of our models' predictions, we use the log likelihood. To obtain the log likelihood, we first use the model to predict the mean and standard deviation at 100 equidistant times. Then we calculate the log likelihood of the ground truth for every predicted point. We take the mean over dimensions, over times, and over trajectories. When reporting the training log likelihood, as done e.g. in Table 1, we use the training trajectories for evaluation. When reporting the generalization log likelihood, as done e.g. in Table 2, we use 10 unseen trajectories. This evaluation is then repeated for 10 different , meaning that we retrain the model 10 times on a data set with the same ground truth, but a different noise realization. We then report the mean and standard deviation of the log likelihood across these repetitions.

Weight decay To prevent overfitting, we use weight decay on the parameters of both the dynamics and the smoother neural networks. We denote by wd_D the weight decay parameter of the dynamics model, and wd_S the weight decay parameters of the smoother model. While we keep the wd_S constant during all three phases of training, we gradually increase wd_D from 0 to its final value, which is the same as wd_S. The increase follows a polynomial schedule with power 0.8.

B.4 Training details

The training of DGM, i.e. optimizing Equation (5), can be split into three distinct phases: *transition*, *training*, and *fine-tuning*. In the *transition* phase we gradually increase the value of both λ and the weight decay regularization parameter of the dynamics (wd_D) from 0 to its final value. When these parameters reach their final value, we reach the end of the transition phase and start the *training* phase. In this phase, all optimization parameters are left constant. It stops when the last 1000 steps are reached. Then, the *fine-tune* phase starts, where we decrease learning rate to 0.01. The gradual increase of λ and wd_D follows polynomial schedule with power 0.8. As an optimizer, we use Adam.

Supporting points The selection of the supporting points in $\mathcal{T}$ is different for data sets consisting of one or multiple trajectories. If there is only one trajectory in the dataset, we match the derivatives at the same places where we observe the state. If there are multiple trajectories, we match the derivatives at 30 equidistant time points on each training trajectory.

Selection of λ The loss of Equation (5) is a multi-objective optimization problem with a trade-off parameter λ. Intuitively, if λ is too small, the model only tries to fit the data and neglects the dynamics. On the other hand, with too large λ, the model neglects the data fit and only cares about the dynamics. In Figure 10 we show a plot of log likelihood score on the 10 test trajectories of the LV 100 dataset with varying λ. We train the model for $\lambda \cdot |\dot{\mathcal{X}}|/|\mathcal{D}| \in \{2^i | i = -20, \dots, 6\}$. To estimate the robustness of the experiment, we show the mean and standard deviation over 5 different noise realizations.

Parameter selection To search for the best performing parameters we performed sweep over learning rate value lr in the transition and training phase and over the weight decay parameters wd_S. For lr, we considered the values $0.02, 0.05$ and 0.1. For wd_S, we considered $0.1, 0.5$ and 1.0.

Training time We did not optimize our hyperparameters and code for training time. Nevertheless, we report the length of each phase and the total time in the Table 3.

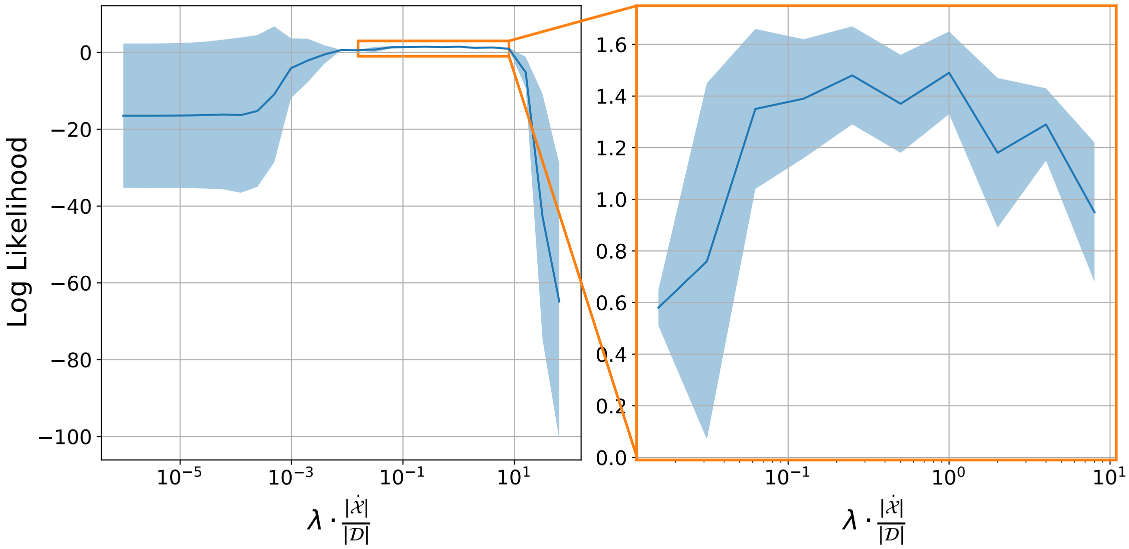

Figure 10: If λ is too small, the dynamics model does not regularize the smoother sufficiently, the model overfits to the data and the test log likelihood score is worse. If λ is too large, the observation term gets dominated, the model underfits and the log likelihood score on the test data is worse. Empirically, we found that we achieve the best log likelihood on the test data with $\lambda = |\mathcal{D}|/|\dot{\mathcal{X}}|$.

Table 3: Number of steps for each training phase and total training time for different datasets. For the times, we report mean ± standard deviation over 10 different random seeds.

	Transition	Training	Fine-Tuning	Time[s]
LV 1	1000	0	1000	329 ± 15
LO 1	1000	0	1000	399 ± 6
DP 1	1000	1000	1000	535 ± 42
QU 1	1000	2000	1000	1121 ± 41
LV 100	1000	0	1000	408 ± 8
LO 125	1000	0	1000	753 ± 8
DP 100	5000	4000	1000	2988 ± 261
QU 64	6000	3000	1000	8387 ± 36

Appendix C. Bayesian NODE training

In this section, we describe the specifics of the experiments and the implementation of SGLD and SGHMC, the Bayesian integration benchmarks used in the paper. For all experiments, we use a slightly changed version of the code provided by Dandekar et al. (2021), which is written in Julia (Bezanson et al., 2017).

C.1 Effects of Overparametrization

Here we provide further details regarding the experiment presented in Figure 2. For the ground truth dynamics $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$, we selected the matrix $\boldsymbol{A}$ such that it has 1 stable and 2 marginally stable modes. The eigenvalue of the stable mode is selected uniformly at random from $[-0.5, -0.1]$. To create marginally stable modes, we create a block $\boldsymbol{C} \in \mathbb{R}^{2\times 2}$, where its components are sampled i.i.d. uniformly at random from $[0, 1]$. The marginally stable part is then created as $\boldsymbol{A}$ as $\frac{\pi}{2\rho(\boldsymbol{C}-\boldsymbol{C}^\top)}\left(\boldsymbol{C} - \boldsymbol{C}^\top\right)$, where $\rho(.)$ denotes the spectral radius. Using the spectral radius in the normalization ensures that the period of the mariginally stable mode is bounded with $\pi/2$. We selected the initial condition for the trajectory uniformly at random from the unit sphere in $\mathbb{R}^3$. We evolved the trajectory on the time interval $(0, 10)$ and observed 100 noisy observations, where every ground truth value was perturbed with additive, independent Gaussian noise, drawn from $\mathcal{N}\left(0, 0.1^2\right)$.

While DGM performed without any pre-training, we observed that both SGLD and SGHMC struggle without narrow priors. To obtain reasonable priors, we followed the following procedure:

First, we pretrain every set of matrices $\boldsymbol{B}_1, \ldots, \boldsymbol{B}_k$ on the ground truth, such that the

$$\sum_{i=1}^{100} \left\| \dot{\boldsymbol{x}}(t_i) - \prod_{j=1}^{k} \boldsymbol{B}_j \boldsymbol{x}(t_i) \right\|_2^2 \leq 10^{-5},$$

where t_i are times at which we observed the state. We selected the prior as Gaussian centered around the pretrained parameters. The standard deviation was chosen such that the standard deviation of the product of the matrices $\prod_{j=1}^{k} \boldsymbol{B}_j$ stays at 0.01, independent of the number of matrices used. For SGHMC, we selected learning rate 1.5×10^{-7} and momentum decay 0.1 as hyperparameters. For SGLD,we chose the hyperparameters $a = 0.001, b = 1$ and $\gamma = 0.9$, which SGLD then uses to calculate a polynomial decay $a(b + k)^{-\gamma}$ (at step k) for its learning rate. For sampling we used 5 chains with 20000 samples on each chain. The last 2000 samples of each chain were used for evaluation. With this setting we ensured that the r-hat score was smaller than 1.1.

C.2 Finetuning for Benchmark Systems

Both SGLD and SGHMC require hyperparameters, that influence their performance quite strongly. In this subsection, we explain all the tuning steps and requirements we deployed for these algorithms to produce the results shown in the main paper. For both algorithms, we used a setup of 6 chains, that were sampled in parallel, with 10000 samples per chain, where the final 2000 samples were taken as predictions. These numbers were selected using the r-hat value to determine if the chains had sufficiently converged.

Hyperparameters of SGLD For SGLD, we additionally searched over the hyperparameters a, b, and γ. All three are used to calculate the learning rate of the algorithm. These parameters were chosen by evaluating the log likelihood of the ground truth on a grid, which was given by the values $a \in \{0.0001, 0.001, 0.005, 0.01, 0.05, 0.1\}$, $b \in \{0.3, 0.6, 1.0, 1.5, 2\}$ and $\gamma \in \{0.5001, 0.55, 0.6, 0.7, 0.8, 0.99\}$. Clearly, using the log likelihood of the ground truth to tune the hyperparameters overestimates the performance of the algorithm, but it provides us with an optimistic estimate of its real performance in practice. For computational reasons, we only performed a grid search on the one trajectory experiments, and then reused the same hyperparameters on the multi-trajectory experiments. All hyperparameters are shown in Table 4.

Hyperparameters of SGHMC For SGHMC, we additionally searched over the hyperparameters the learning rate and the momentum decay. Again, these parameters were chosen by evaluating the log likelihood of the ground truth on a grid, where learning rate was chosen from the set $\{1\text{e}{-8}, 5\text{e}{-8}, 1.5\text{e}{-7}, 5\text{e}{-7}, 1\text{e}{-6}, 5\text{e}{-6}, 1\text{e}{-5}, 5\text{e}{-5}, 1\text{e}{-4}, 5\text{e}{-4}\}$ and momentum decay was chosen from the set $\{0.0001, 0.001, 0.05, 0.1, 0.5, 1, 5\}$. Since we used the log likelihood of the ground truth again to tune the hyperparameters, we overestimate the performance of the algorithm and obtain thus an optimistic estimate of its real performance in practice. For computational reasons, be only performed a grid search on the one trajectory experiments, and then reused the same hyperparameters on the multi-trajectory experiments. All hyperparameters are shown in Table 4.

Table 4: Hyperparameters with best performance evaluated on the likelihood of the ground truth. The hyperparameters are different for the parametric (p) and the non-parametric (n) dynamics models, which is indicated with the last letter.

	SGLD			SGHMC	
	a	b	γ	learning rate	momentum decay
Lotka Volterra p	1e−3	2	0.5001	5e−7	0.1
Lorenz p	0.001	1.5	0.5001	1e−5	0.05
Double Pendulum p	0.1	0.3	0.5001	1e−6	0.1
Quadrocopter p	0.0001	1.5	0.7	5e−7	0.5
Lotka Volterra n	0.005	1.5	0.7	5e−7	0.5
Lorenz n	0.001	1.5	0.55	5e−6	0.1
Double Pendulum n	0.01	2	0.55	1e−6	0.05
Quadrocopter n	F	F	F	5e−7	0.05

Choice of Priors for SGLD **and** SGHMC Since SGLD and SGHMC are both Bayesian methods, they need to be supplied with a prior. As we discovered in our experiments, this prior plays a crucial role in the stability of the algorithm. In the end, we did not manage to get them to converge without some use of ground truth. In particular, if the priors were not chosen narrowly around some ground truth, the algorithms just returned NaNs, since their integration scheme runs into numerical issues. For the parametric case shown in Appendix E, where we assume access to the true parametric form of the system, we thus chose narrow priors around the ground truth of the parameter values, that were used to create the data set. For Lotka Volterra, we chose a uniform distribution around the ground truth ±0.5, i.e. $\theta_i \sim$ Uniform$[0.5, 1]$ for all components of $\boldsymbol{\theta}$. For Lorenz, we chose $\alpha \sim$ Uniform$[8, 12]$, $\beta \sim$ Uniform$[25, 31]$ and $\gamma \sim$ Uniform$[6/3, 10/3]$. For Double Pendulum, we chose $m \sim$ Uniform$[0.5, 1.5]$ and $l \sim$ Uniform$[0.5, 1.5]$. For Quadrocopter, we chose independent Gaussians, centered around the ground truth, with a standard deviation of 0.005. For all experiments, the prior on the observation noise was set to $\sigma \sim \text{InverseGamma}[2, 3]$, except for SGLD when inferring the Lorenz system. There, we had to fix the noise standard deviation to its ground truth, to get convergence.

For the non-parametric case shown in the main paper, we needed a different strategy, since no ground truth information was available for the weights of the neural dynamics model. Thus, we first trained a deterministic dynamics model. As data, we sampled 100 tuples $\boldsymbol{x}, \dot{\boldsymbol{x}}$ equidistantly in time on the trajectory. Note that we implicitly assume access to the ground truth of the dynamics model, i.e. we assume we are provided with accurate, noise free $\dot{\boldsymbol{x}}$. The neural dynamics model was then pre-trained on these pairs, until the loss was almost zero (up to $1\mathrm{e}{-5}$). SGLD and SGHMC were then provided with Gaussian priors, independent for each component of $\boldsymbol{\theta}$, centered around the pre-trained weights, with a standard deviation of 0.1.

C.3 Number of integrations for prediction

SGLD and SGHMC both return samples of the parameters of the dynamics model. To obtain uncertainties in the state space at prediction time, each one of these samples needs to be turned into a sample trajectory, by using numerical integration. To obtain maximum accuracy, we would ideally integrate all parameter samples obtained by the chains. However, due to the computational burden inflicted by numerical integration, this is not feasible. We thus need to find a trade-off between accuracy and computational cost, by randomly subsampling the number of available parameter samples.

In Figure 11 we show how the log likelihood of the ground truth changes with increasing number of sample trajectories on the LV 1 dataset. After initial fluctuations, the log likelihood of the ground truth stabilizes after approximately 200 steps. To obtain the results of Table 1, we thus chose 200 integration steps.

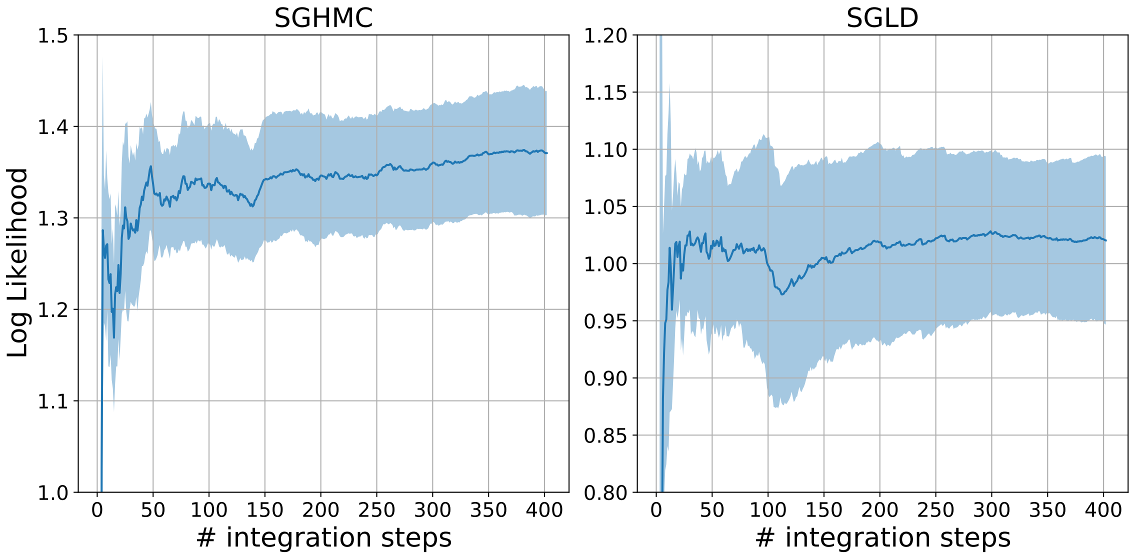

Figure 11: We select the number of sample trajectories for uncertainty prediction to be 200, since we observe that the log likelihood of the ground truth stops fluctuating after 200 steps.

Appendix D. Scaling to many observations or trajectories

Let N be the total number of observations, summed over all training trajectories. In this section, we will analyze the computational complexity of DGM in terms of N and demonstrate how this can be drastically reduced using standard methods from the literature. For notational compactness, we will assume that the supporting points in $\mathcal{T}$ are at the same locations as the observations in $\mathcal{D}$. However, this is by no means necessary. As long as they are chosen to be constant or stand in a linear relationship to the number of observations, our analysis still holds. We will thus use $\boldsymbol{x}$ and $\boldsymbol{x}_{\text{supp}}$ and the corresponding quantities interchangeably. Similarly, we will omit the k that was used for indexing the state dimension and assume one-dimensional systems. The extension to multi-dimensional systems is straight forward and comes at the cost of an additional factor K.

Fortunately, most components of the loss of DGM given by Equation (5) can be calculated in linear time. In particular, it is worth noting that the independence assumption made when calculating the Wasserstein distance in Equation (16) alleviates the need to work with the full covariance matrix and lets us work with its diagonal elements instead. Nevertheless, there are several terms that are not straight forward to calculate. Besides the marginal log likelihood of the observations, these are the posteriors

$$p_S(\boldsymbol{x}|\mathcal{D},\mathcal{T}) = \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}_{\text{post}}, \boldsymbol{\Sigma}_{\text{post}}\right), \tag{17}$$

$$p_S(\dot{\mathcal{X}}|\mathcal{D},\mathcal{T}) = \mathcal{N}\left(\dot{\boldsymbol{x}}_{\text{supp}}|\boldsymbol{\mu}_S, \boldsymbol{\Sigma}_S\right), \tag{18}$$

where

$$\boldsymbol{\mu}_{\text{post}} = \boldsymbol{\mu} + \boldsymbol{\mathcal{K}}^T(\boldsymbol{\mathcal{K}} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y}, \tag{19}$$

$$\boldsymbol{\Sigma}_{\text{post}} = \boldsymbol{\mathcal{K}} - \boldsymbol{\mathcal{K}}^T(\boldsymbol{\mathcal{K}} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{\mathcal{K}}, \tag{20}$$

$$\boldsymbol{\mu}_S = \dot{\boldsymbol{\mu}} + \dot{\boldsymbol{\mathcal{K}}}(\boldsymbol{\mathcal{K}} + \sigma^2\boldsymbol{I})^{-1}\left(\boldsymbol{y} - \boldsymbol{\mu}\right), \tag{21}$$

$$\boldsymbol{\Sigma}_S = \ddot{\boldsymbol{\mathcal{K}}} - \dot{\boldsymbol{\mathcal{K}}}(\boldsymbol{\mathcal{K}} + \sigma^2\boldsymbol{I})^{-1}\dot{\boldsymbol{\mathcal{K}}}^\top. \tag{22}$$

Here, Equation (17) is used for prediction, while its mean is also used in the approximation of Equation (16). On the other hand, Equation (18) is used directly for Equation (16). Note that in both cases, we only need the diagonal elements of the covariance matrices, a fact that will become important later on.

In its original form, calculating the matrix inverses of both Equation (17) and Equation (18) has cubic complexity in N. To alleviate this problem, we follow Rahimi et al. (2007) and Angelis et al. (2020) by using a feature approximation of the kernel matrix and its derivatives. In particular, let $\boldsymbol{\Phi} \in \mathbb{R}^{F\times N}$ be a matrix of F random Fourier features as described by Rahimi et al. (2007). Furthermore, denote $\dot{\boldsymbol{\Phi}}$ as its derivative w.r.t. the time input variable, as defined by Angelis et al. (2020). We can now approximate the kernel matrix and its derivative versions as

$$\boldsymbol{K} \approx \boldsymbol{\Phi}^\top\boldsymbol{\Phi}, \quad \dot{\boldsymbol{\mathcal{K}}}^\top \approx \dot{\boldsymbol{\Phi}}^\top\boldsymbol{\Phi}, \quad \text{and} \quad \ddot{\boldsymbol{\mathcal{K}}} \approx \dot{\boldsymbol{\Phi}}^\top\dot{\boldsymbol{\Phi}}. \tag{23}$$

Using these approximations, we can leverage the Woodbury idendity to approximate

$$(\boldsymbol{\mathcal{K}} + \sigma^2\boldsymbol{I})^{-1} \approx \frac{1}{\sigma^2}\left[\boldsymbol{I} - \boldsymbol{\Phi}^\top\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \sigma^2\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}\right]. \tag{24}$$

This approximation allows us to invert a $F \times F$ matrix, to replace the inversion of a $N \times N$ matrix. This can be leveraged to calculate

$$\begin{aligned}\boldsymbol{\mu}_S &= \dot{\boldsymbol{\mu}} + \dot{\boldsymbol{\mathcal{K}}}(\boldsymbol{\mathcal{K}} + \sigma^2 \boldsymbol{I})^{-1} (\boldsymbol{y} - \boldsymbol{\mu}) && (25)\\ &\approx \dot{\boldsymbol{\mu}} + \frac{1}{\sigma^2} \dot{\boldsymbol{\Phi}}^\top \boldsymbol{\Phi} \left[\boldsymbol{I} - \boldsymbol{\Phi}^\top \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \sigma^2 \boldsymbol{I} \right)^{-1} \boldsymbol{\Phi} \right] (\boldsymbol{y} - \boldsymbol{\mu}) && (26)\end{aligned}$$

and

$$\begin{aligned}\boldsymbol{\Sigma}_S &= \ddot{\boldsymbol{\mathcal{K}}} - \dot{\boldsymbol{\mathcal{K}}}(\boldsymbol{\mathcal{K}} + \sigma^2 \boldsymbol{I})^{-1} \dot{\boldsymbol{\mathcal{K}}}^\top && (27)\\ &\approx \dot{\boldsymbol{\Phi}}^\top \dot{\boldsymbol{\Phi}} - \frac{1}{\sigma^2} \dot{\boldsymbol{\Phi}}^\top \boldsymbol{\Phi} \left[\boldsymbol{I} - \boldsymbol{\Phi}^\top \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \sigma^2 \boldsymbol{I} \right)^{-1} \boldsymbol{\Phi} \right] \boldsymbol{\Phi}^\top \dot{\boldsymbol{\Phi}}. && (28)\end{aligned}$$

Evaluating the matrix multiplications of Equation (26) in the right order leads to a computational complexity of $\mathcal{O}(NF^2 + F^3)$. Similarly, the diagonal elements of the covariance given by Equation (18) can be calculated with the same complexity, by carefully summarizing everything in between $\dot{\boldsymbol{\Phi}}^\top$ and $\dot{\boldsymbol{\Phi}}$ as one $F \times F$ matrix and then calculating the N products independently.

Since the components of Equation (17) have the exact same form as the components of Equation (18), they can be approximated in the exact same way to obtain the exact same computational complexity. Thus, the only components that need further analysis are the components of the marginal log likelihood of the observations, particularly

$$\boldsymbol{y}^\top (\boldsymbol{\mathcal{K}} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y} \approx \boldsymbol{y}^\top \frac{1}{\sigma^2} \left[\boldsymbol{I} - \boldsymbol{\Phi}^\top \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \sigma^2 \boldsymbol{I} \right)^{-1} \boldsymbol{\Phi} \right] \boldsymbol{y} \qquad (29)$$

and

$$\begin{aligned}\text{logdet}(\boldsymbol{\mathcal{K}} + \sigma^2 \boldsymbol{I}) &\approx \text{logdet}(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \sigma^2 \boldsymbol{I}) && (30)\\ &\approx \text{logdet}(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \sigma^2 \boldsymbol{I}) + (N - F)\text{log}(\sigma^2). && (31)\end{aligned}$$

In the last line, we used the fact that the nonzero eigenvalues of the transposed of a matrix stay the same.

Combining all these tricks, it is clear that the overall complexity of DGM can be reduced to $\mathcal{O}(NF^2 + F^3)$. Since F is a constant controlling the quality of the approximation scheme and is usually chosen to be constant, we thus get essentially linear computational complexity in the number of observations. Note that these derivations are completely independent of what scheme is chosen to obtain the feature matrix $\boldsymbol{\Phi}$. For ease of implementation, we opted for random Fourier features though in our experiments.

Experimental Proof of Concept To demonstrate that this approximation scheme can be used in the context of DGM, we tested it on the multi-trajectory experiment of Lotka Volterra. To this end, we increased the grid from 10 points per dimension to 25, leading to a total number of 3125 observations instead of 500. As an approximation, we used 50 random Fourier features. Through this approximation, DGM became slightly more sensitive to the optimization hyperparameters. Nevertheless, it reached comparable accuracy within roughly 440 seconds of training, compared to the 408 seconds needed to train the approximation free version on LV 100.

Appendix E. Additional experiments

In this section, we first show the state predictions of DGM on the datasets with multiple trajectories. Then, we compare DGM with SGLD and SGHMC for the parametric case, i.e. where we assume to have access to the true parametric form of the dynamics. Since most datasets have too many trajectories to be fully visualized, we show a random subset instead.

E.1 Sample plots from trained trajectories

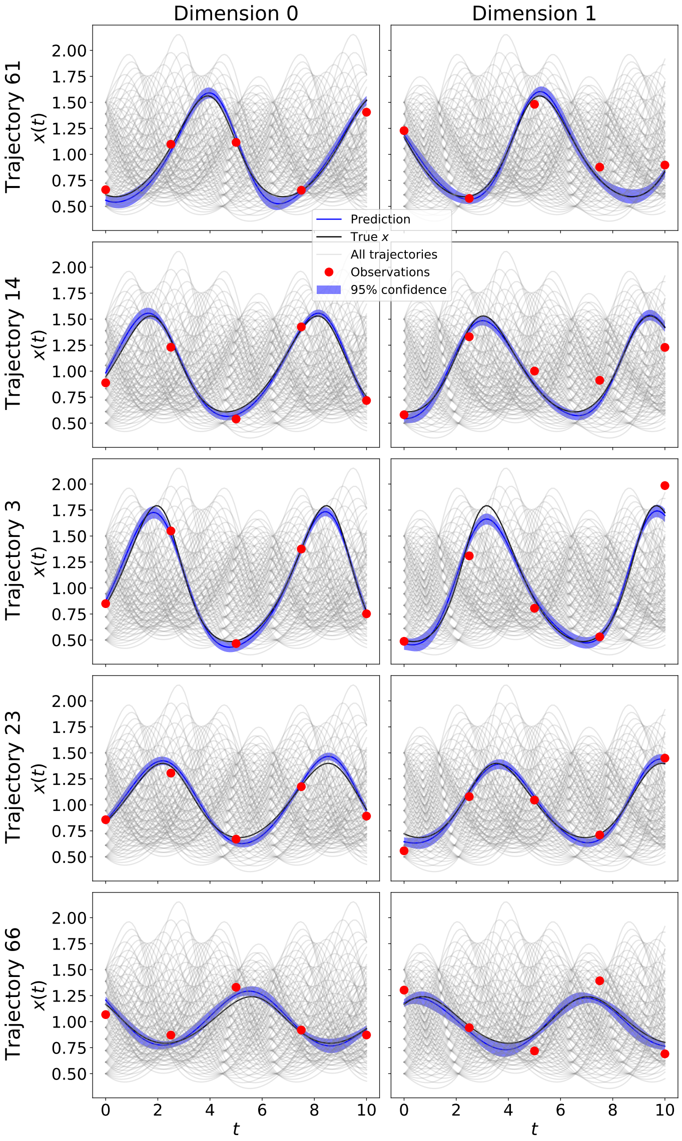

Figure 12: DGM's prediction on 5 randomly sampled training trajectories of LV 100.

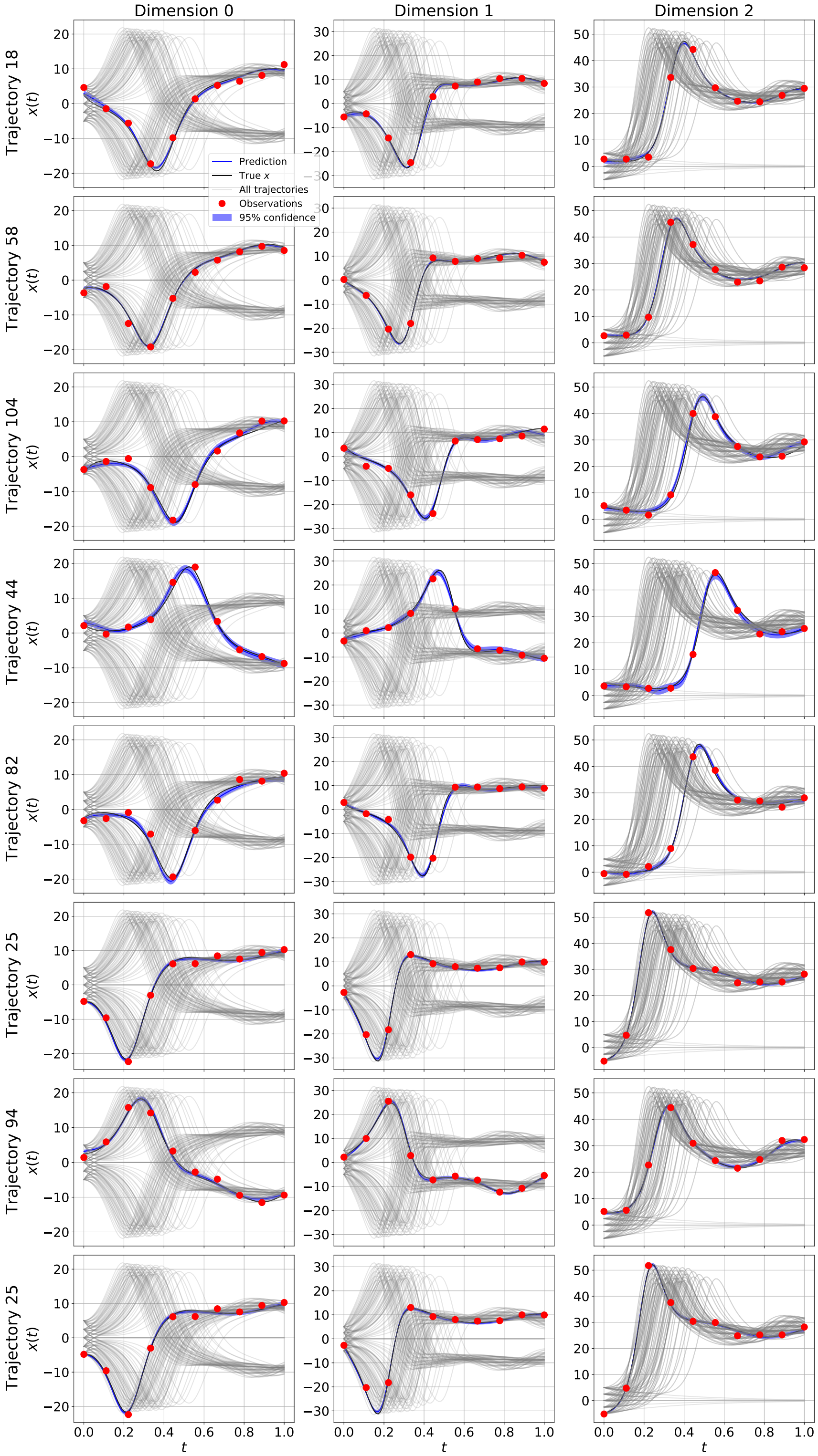

Figure 13: DGM's prediction on 8 randomly sampled training trajectories of LO 125.

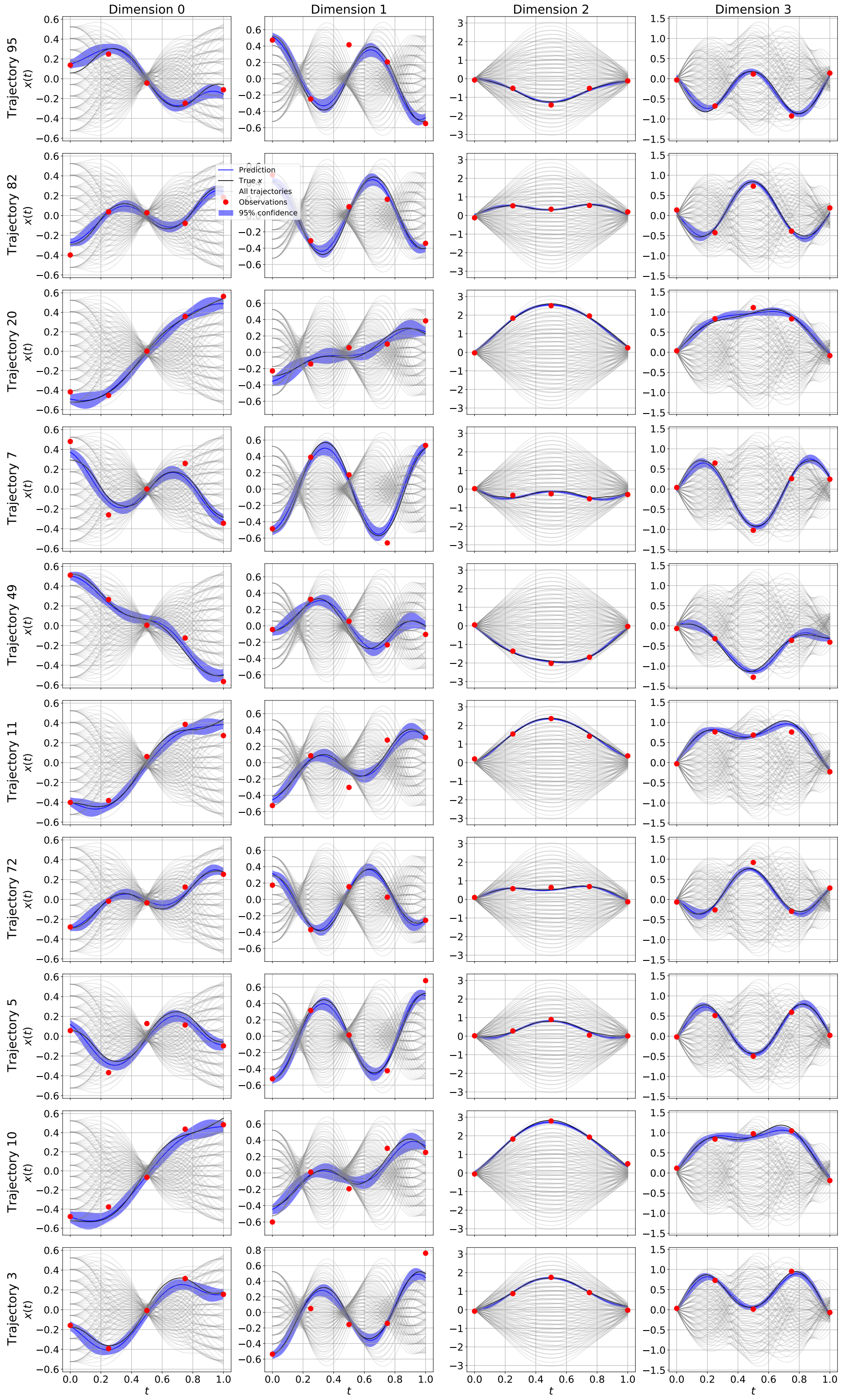

Figure 14: DGM's prediction on 10 randomly sampled training trajectories of DP 100.

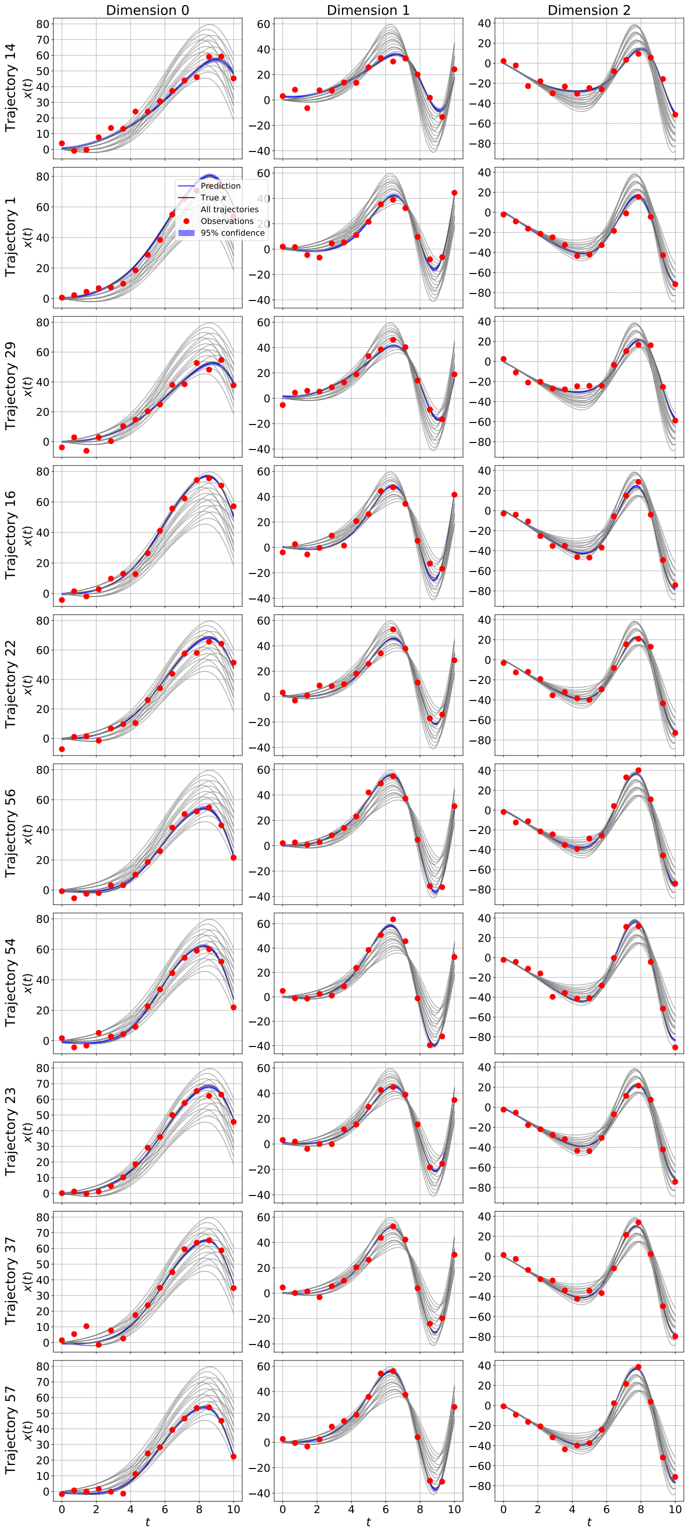

Figure 15: DGM's prediction on 10 randomly sampled training trajectories of QU 64, for state dimensions 0-2.

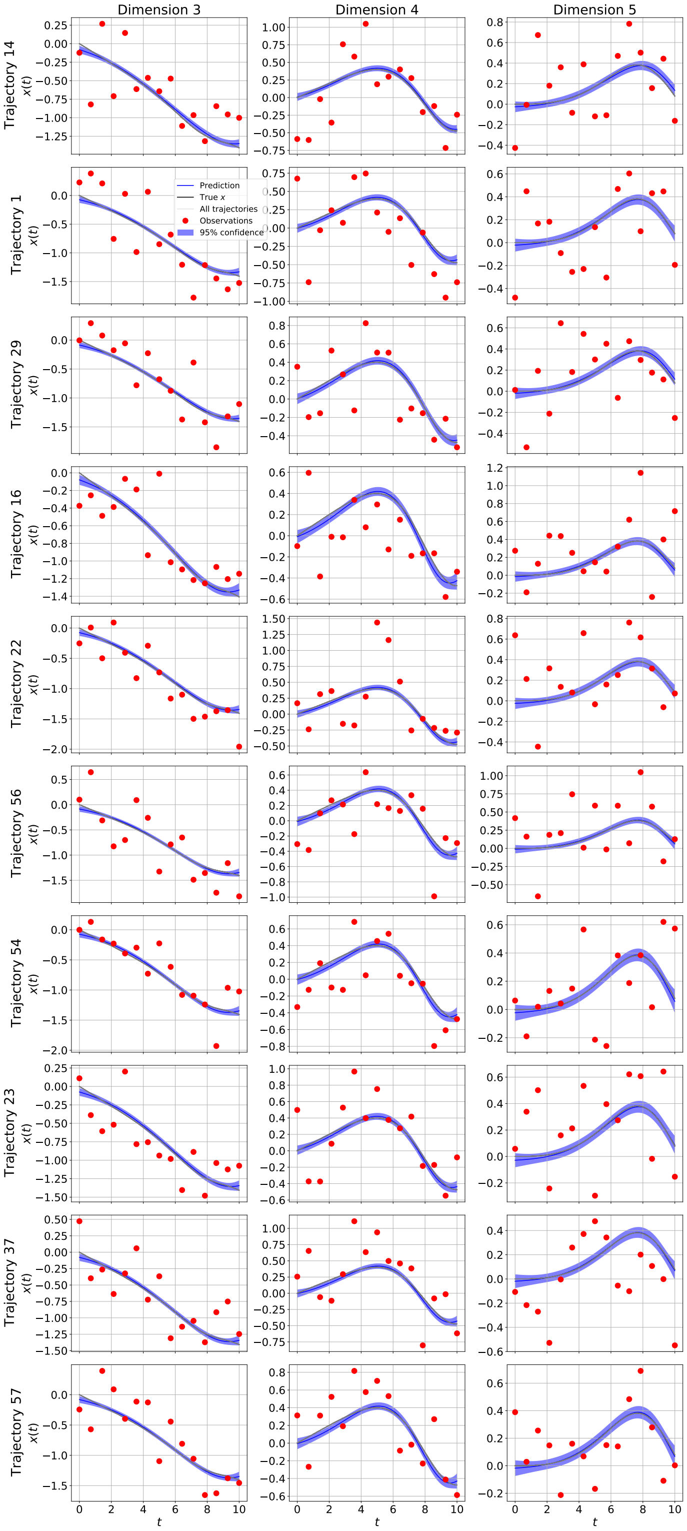

Figure 16: DGM's prediction on 10 randomly sampled training trajectories of QU 64, for state dimensions 3-5.

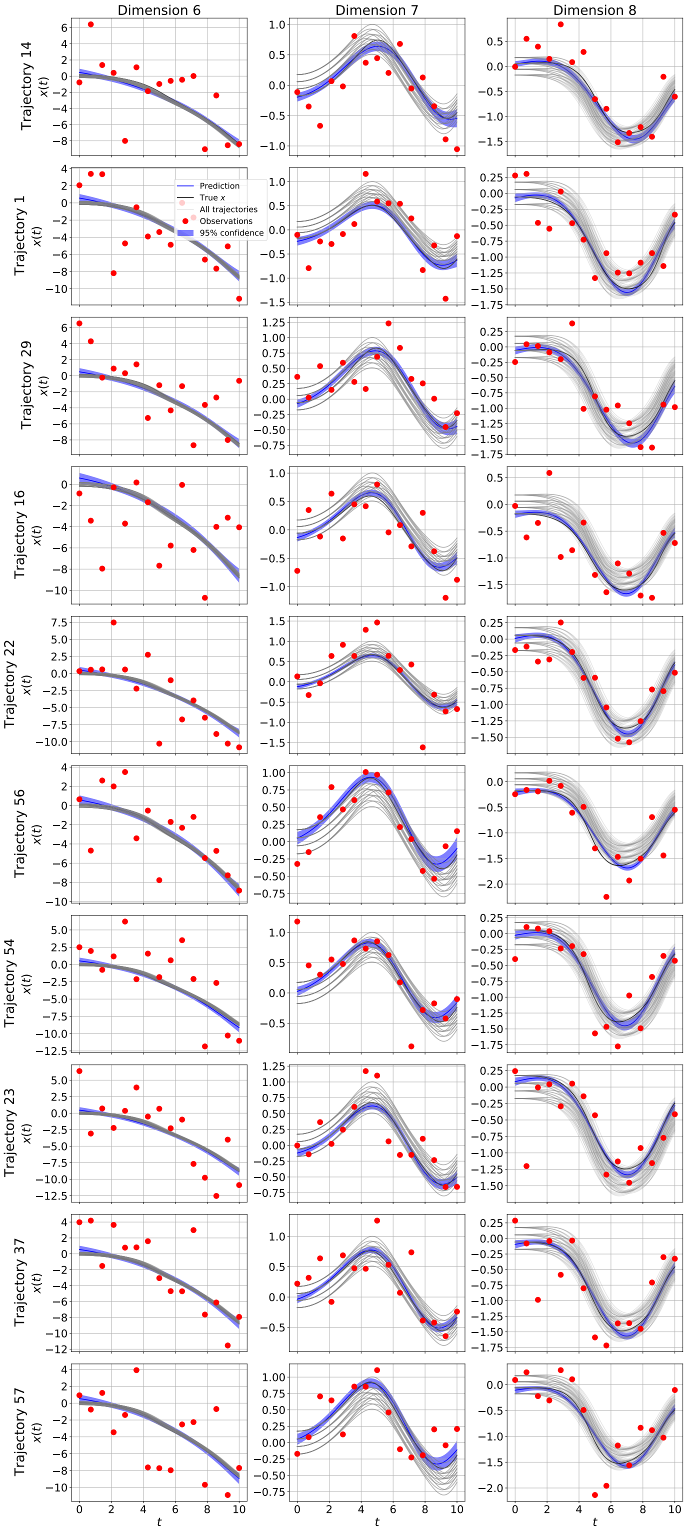

Figure 17: DGM's prediction on 10 randomly sampled training trajectories of QU 64, for state dimensions 6-8.

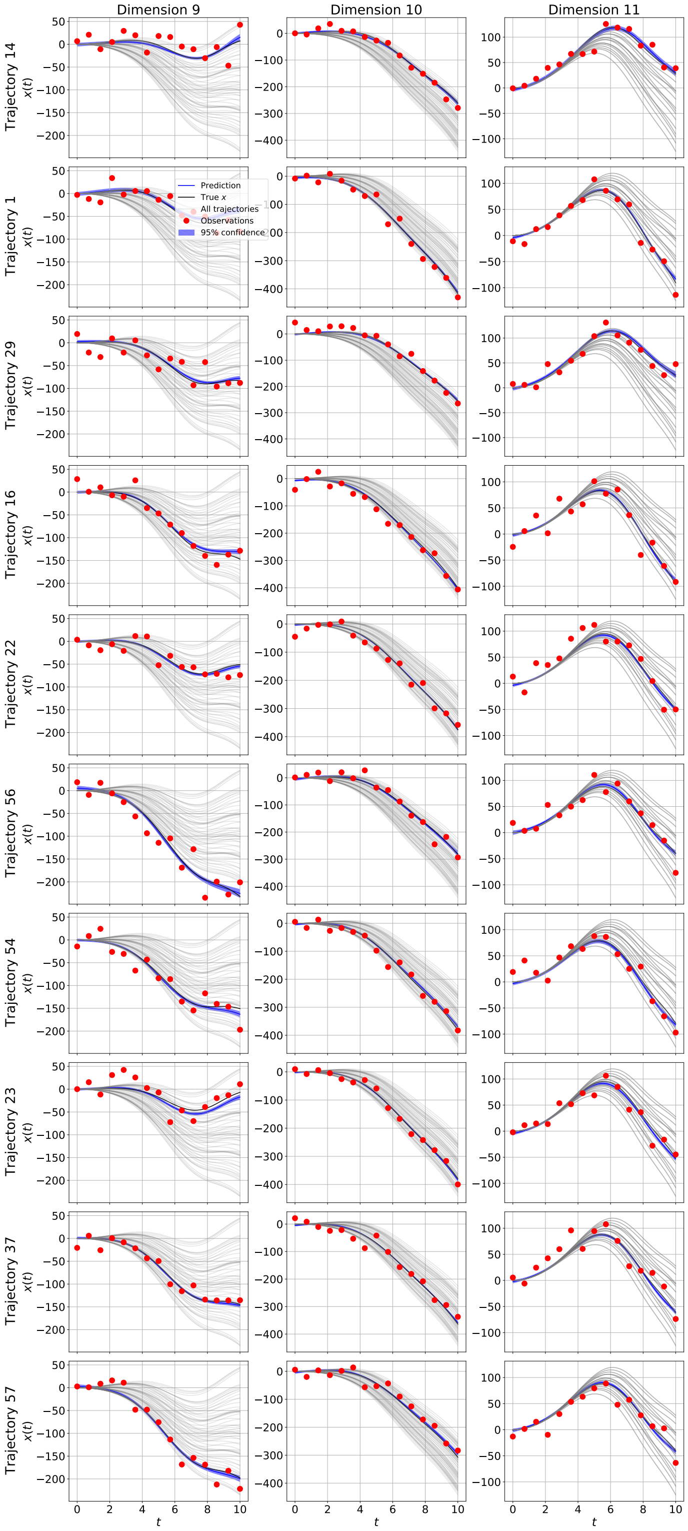

Figure 18: DGM's prediction on 10 randomly sampled training trajectories of QU 64, for state dimensions 9-11.

E.2 Sample plots from test trajectories

Here, we show DGM's predictions on the test trajectories used to test generalization, as introduced in Appendix A. Since LV 100 is a two dimensional system, we also show the placement of the train and test initial conditions in Figure 19.

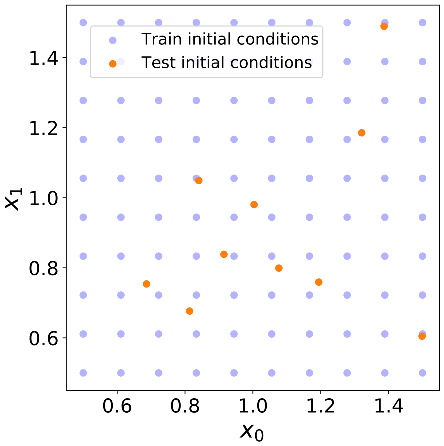

Figure 19: Placement of the initial conditions for the train and test trajectories of the LV 100 dataset. We selected the initial conditions for the train trajectories by gridding $[0.5, 1.5]^2$ with 10 points in every dimension. We select initial conditions for test trajectories independently, uniformly at random from the cube $[0.5, 1.5]^2$.

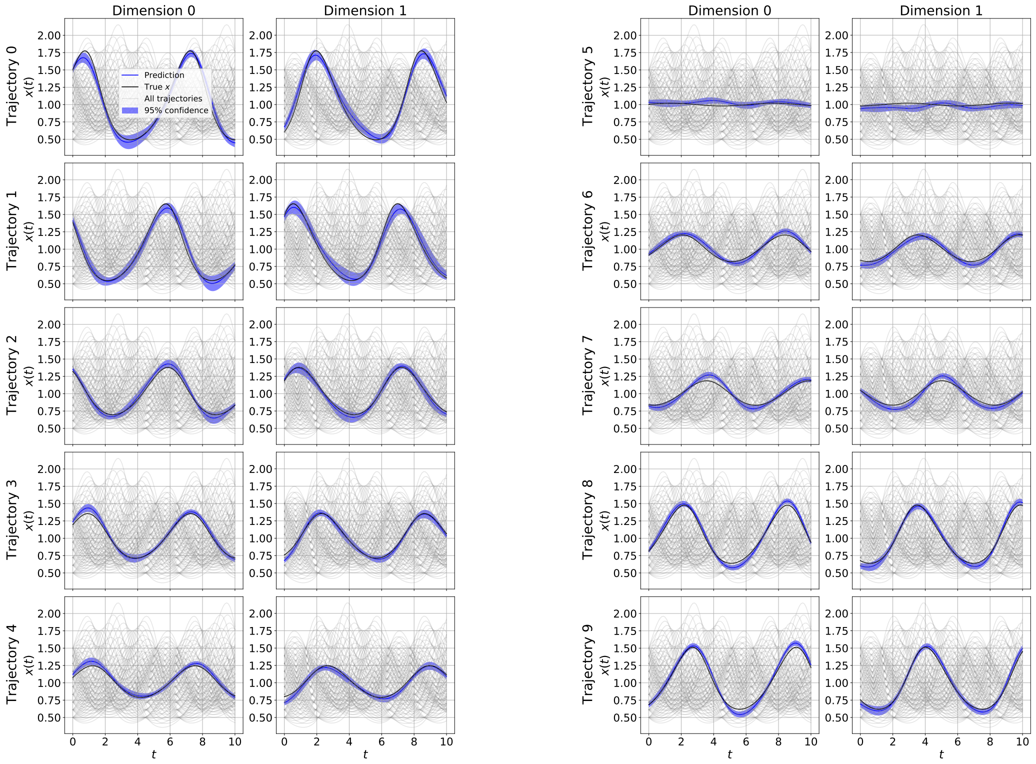

Figure 20: DGM's prediction on 10 randomly sampled test trajectories for the LV 100 dataset.

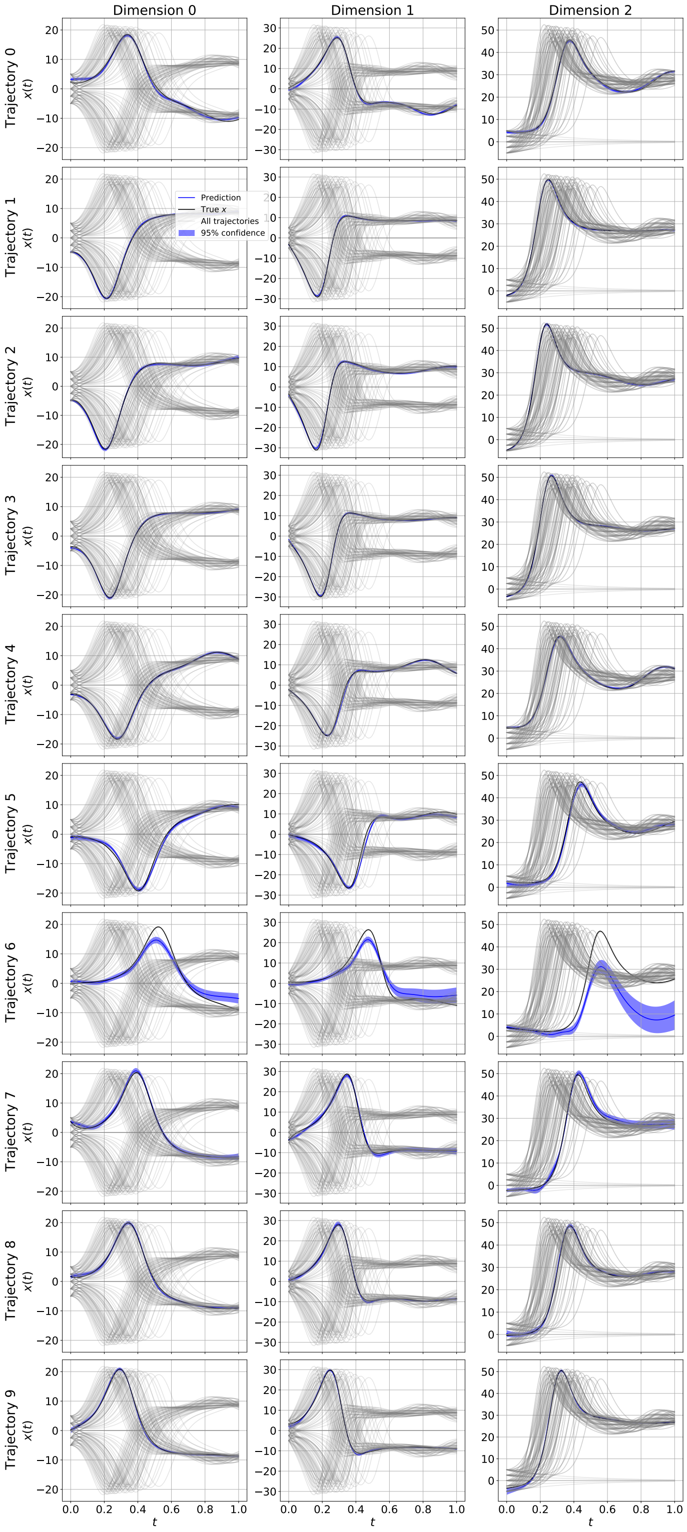

Figure 21: DGM's prediction on 10 randomly sampled test trajectories for the LO 125 dataset.

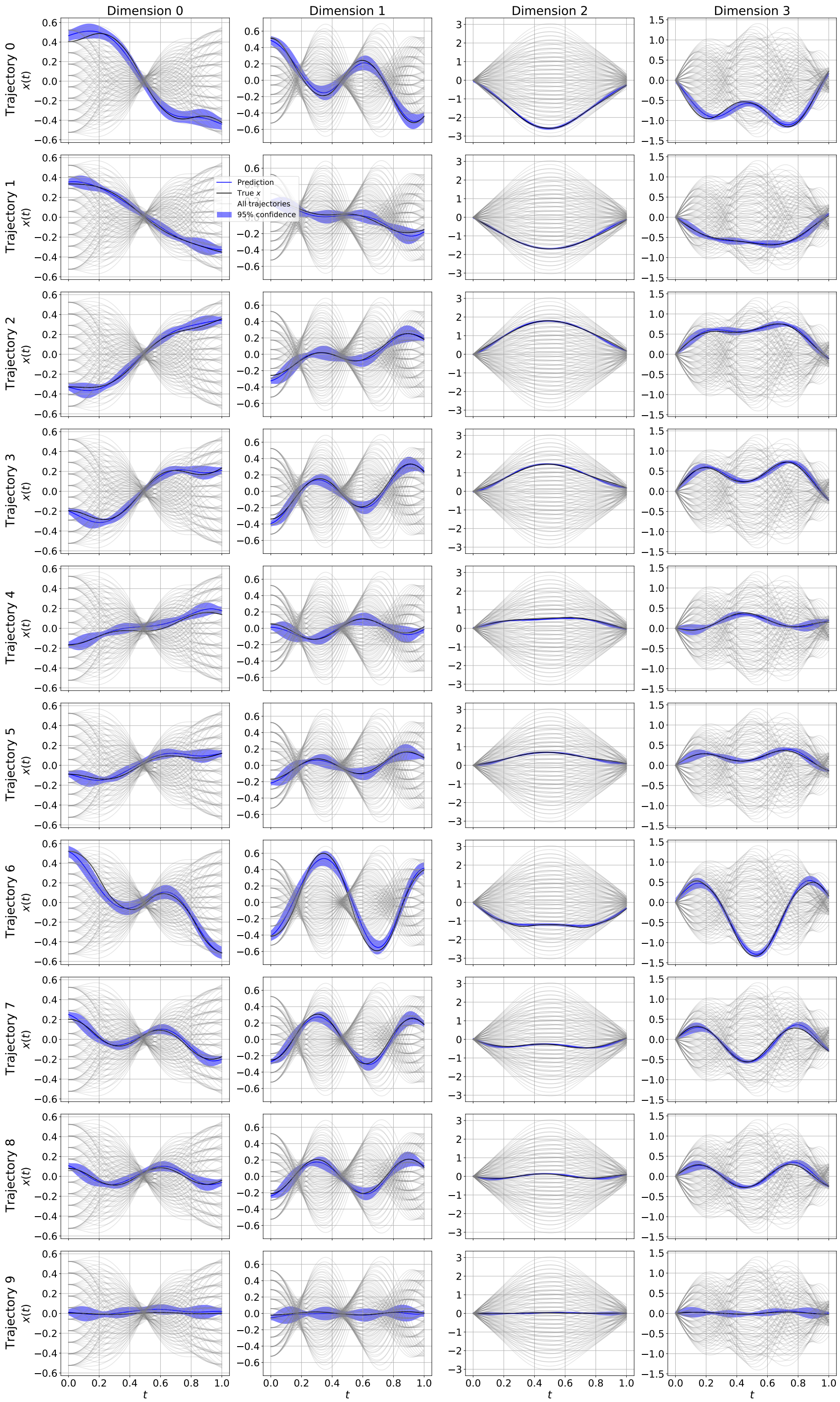

Figure 22: DGM's prediction on 10 randomly sampled test trajectories for the DP 100 dataset.

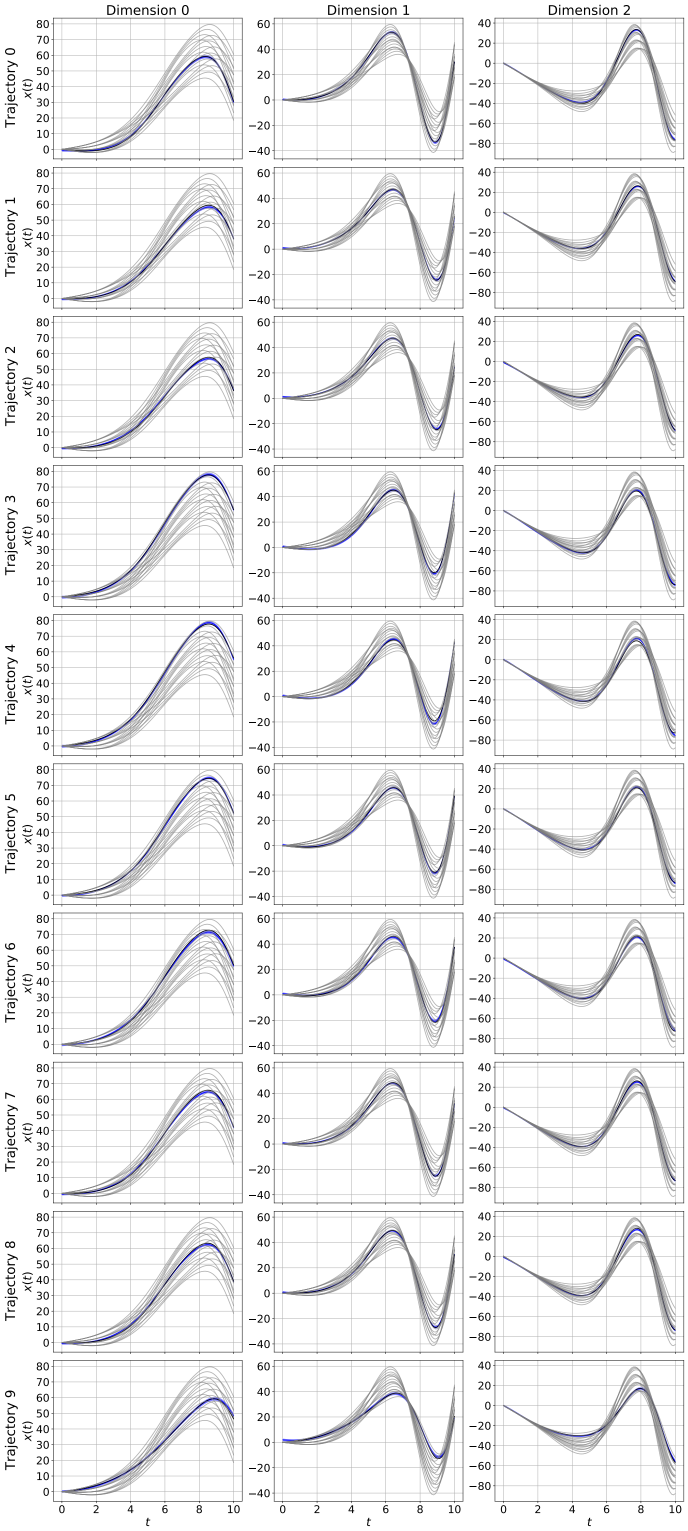

Figure 23: DGM's prediction on 10 randomly sampled test trajectories of QU 64, for state dimensions 0-2.

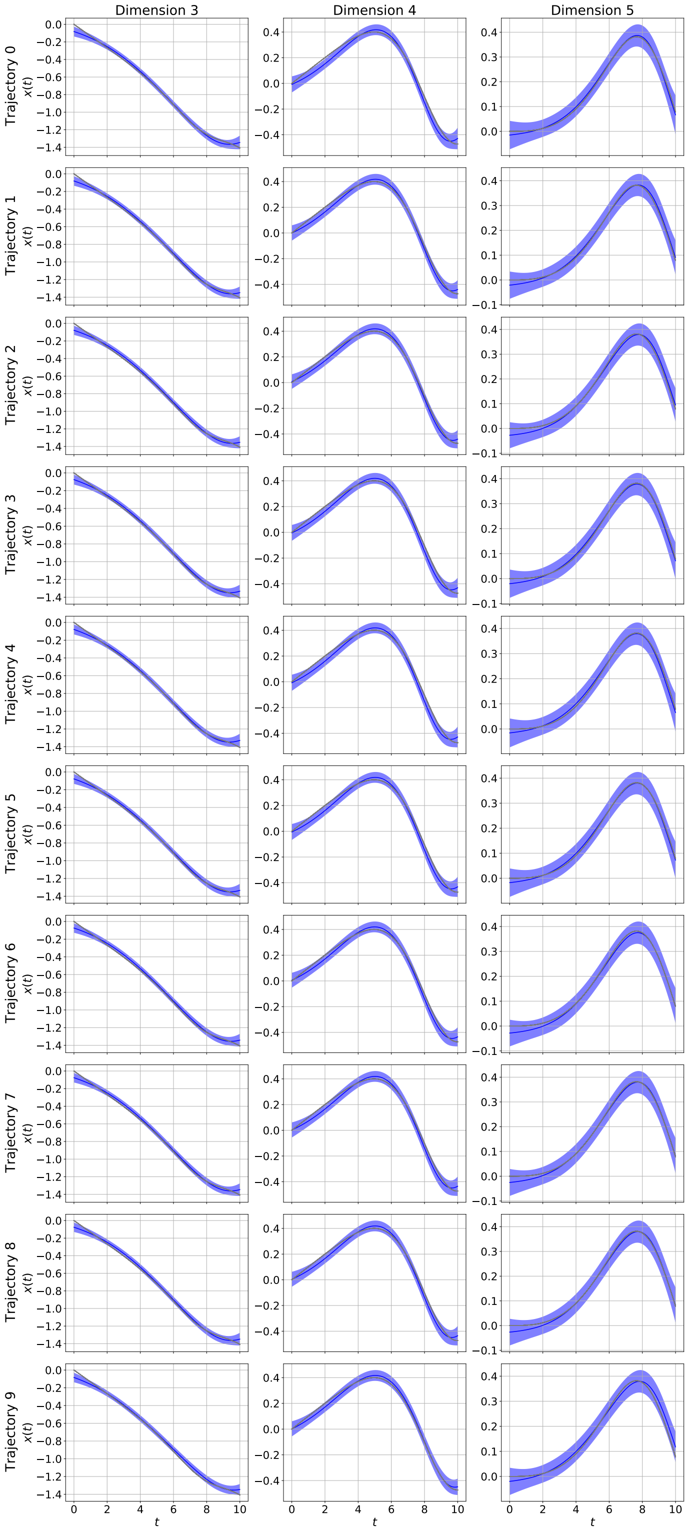

Figure 24: DGM's prediction on 10 randomly sampled test trajectories of QU 64, for state dimensions 3-5.

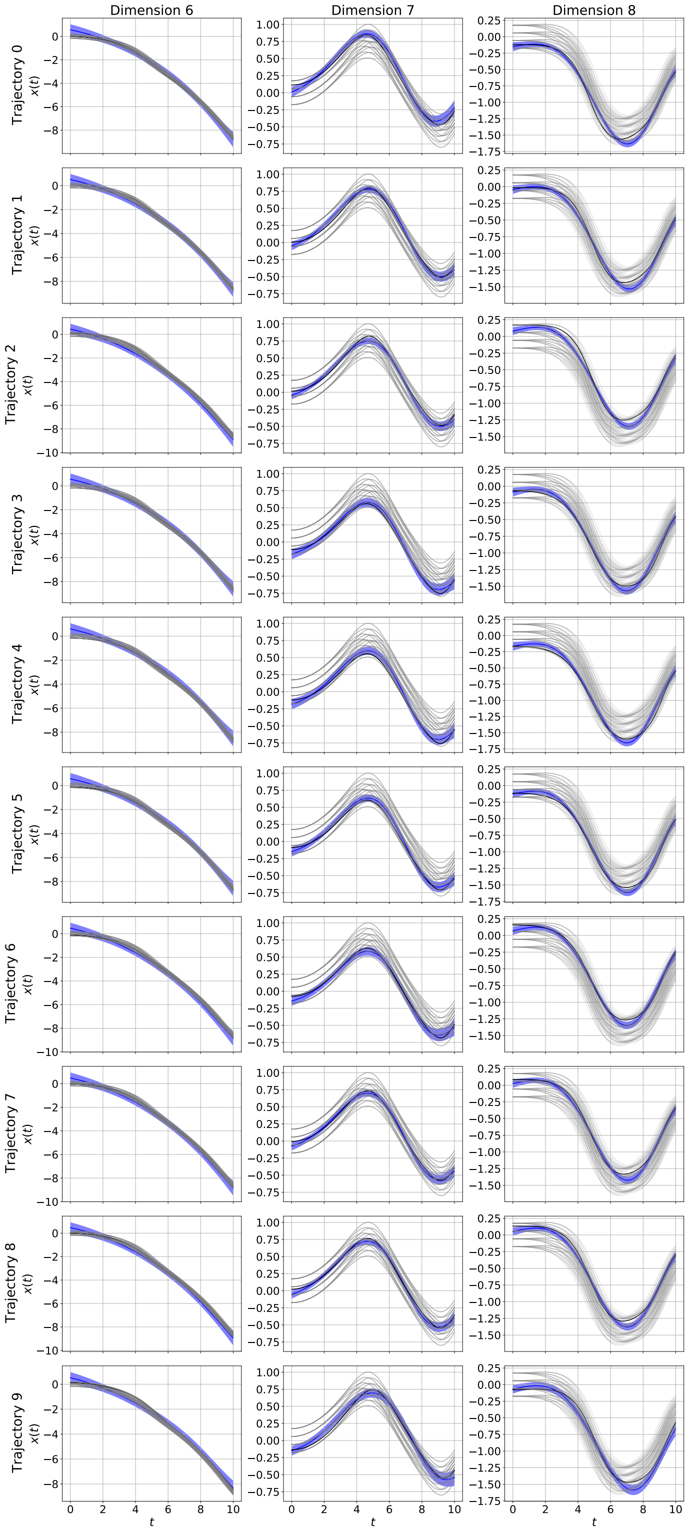

Figure 25: DGM's prediction on 10 randomly sampled test trajectories of QU 64, for state dimensions 6-8.

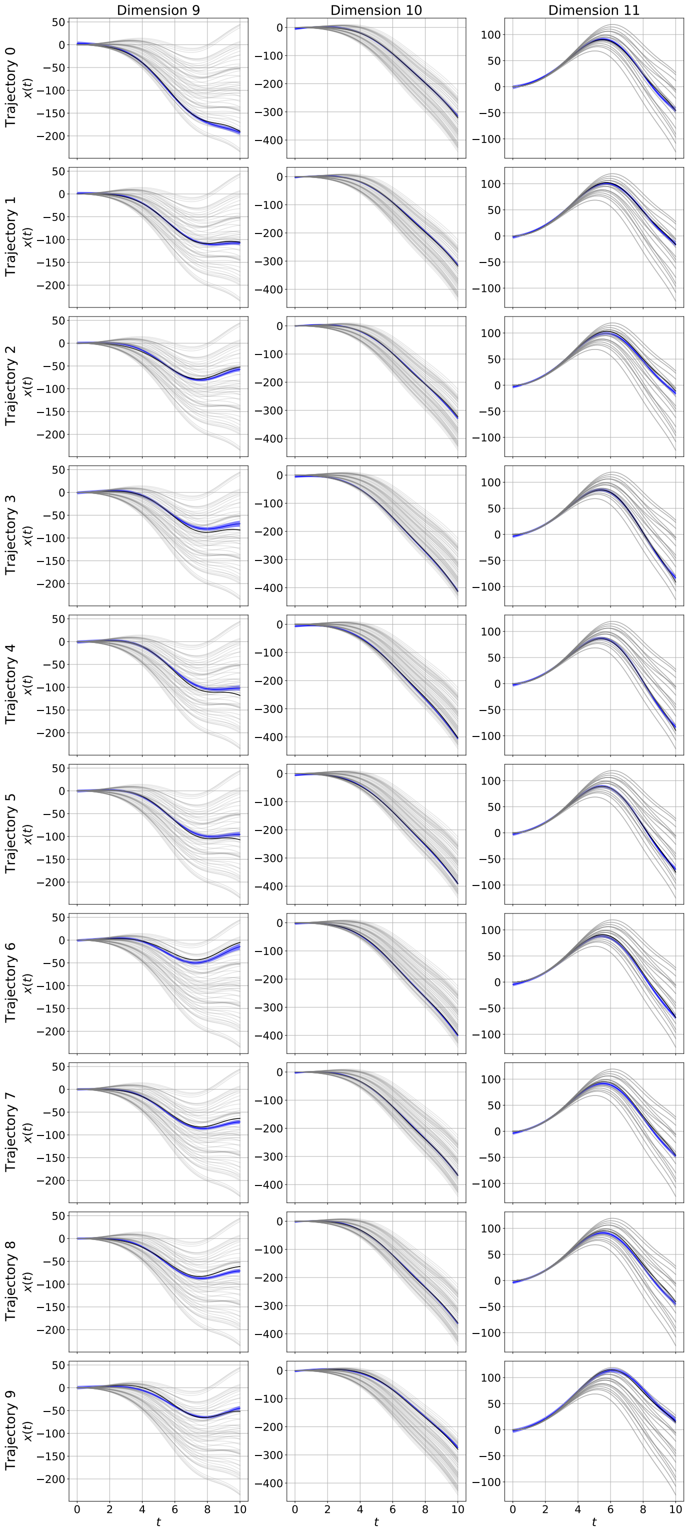

Figure 26: DGM's prediction on 10 randomly sampled test trajectories of QU 64, for state dimensions 9-11.

E.3 Comparison with parameteric integration

In this subsection, we compare DGM against SGLD and SGHMC in the parametric setting, i.e. where we assume access to the parametric form of the true dynamics $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$. Despite serious tuning efforts outlined in Appendix C.2, we were unable to make SGLD and SGHMC perform on any multitrajectory experiments except for Lotka Volterra 100. As can be seen in Table 5, the sampling based methods seem to perform quite well. However, it should be noted that we were unable to get a stable performance without using ground truth information, as outlined in Appendix C.2. Given this caveat and the results of the non-parametric case in the main paper, we conclude the following. If strong and accurate expert knowledge is available that can be used to fix strong priors on simple systems, the sampling-based approaches are certainly a good choice. For more complex systems or in the absence of any expert knowledge, DGM seems to have a clear edge.

Table 5: Log likelihood of the ground truth of 100 points on the training trajectories. SGHMC and SGLD were provided with strong, ground-truth-inspired priors and received an extensive hyperparameter sweep using the ground truth as metric. Nevertheless, DGM performs decently in comparison, without using neither priors nor ground truth.

	Log Likelihood		
	DGM	SGLD	SGHMC
LV 1	1.98 ± 0.18	$\mathbf{3.07 \pm 0.685}$	3.06 ± 0.517
LO 1	-0.52 ± 0.09	$\mathbf{2.01 \pm 0.548}$	F
DP 1	2.16 ± 0.13	$\mathbf{3.43 \pm 0.396}$	2.96 ± 0.795
QU 1	0.71 ± 0.07	$\mathbf{2.42 \pm 0.322}$	1.38 ± 0.00
LV 100	1.85 ± 0.11	$\mathbf{4.28 \pm 0.184}$	4.26 ± 0.178