RESEARCH ARTICLE | OCTOBER 06 2023

Generative modeling of time-dependent densities via optimal transport and projection pursuit

Jonah Botvinick-Greenhouse; Yunan Yang; Romit Maulik

Chaos 33, 103108 (2023)
https://doi.org/10.1063/5.0155783

Chaos ARTICLE pubs.aip.org/aip/cha

Generative modeling of time-dependent densities via optimal transport and projection pursuit

Cite as: Chaos **33**, 103108 (2023); doi: 10.1063/5.0155783
Submitted: 23 April 2023 · Accepted: 11 September 2023 ·
Published Online: 6 October 2023

Jonah Botvinick-Greenhouse,[1,a)] Yunan Yang,[2,b)] and Romit Maulik[3,4,a)]

AFFILIATIONS

[1] Center for Applied Mathematics, Cornell University, Ithaca, New York 14850, USA
[2] Department of Mathematics, Cornell University, Ithaca, New York 14853, USA
[3] College of Information Science and Technology, Pennsylvania State University, University Park, Pennsylvania 16802, USA
[4] Joint Appointment Faculty, Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, Illinois 60439, USA

[a)] **Authors to whom correspondence should be addressed:** jrb482@cornell.edu and rmaulik@anl.gov
[b)] **Electronic mail:** yunan.yang@cornell.edu

ABSTRACT

Motivated by the computational difficulties incurred by popular deep learning algorithms for the generative modeling of temporal densities, we propose a cheap alternative that requires minimal hyperparameter tuning and scales favorably to high-dimensional problems. In particular, we use a projection-based optimal transport solver [Meng *et al., Advances in Neural Information Processing Systems* (Curran Associates, 2019), Vol. 32] to join successive samples and, subsequently, use transport splines (Chewi *et al.,* 2020) to interpolate the evolving density. When the sampling frequency is sufficiently high, the optimal maps are close to the identity and are, thus, computationally efficient to compute. Moreover, the training process is highly parallelizable as all optimal maps are independent and can, thus, be learned simultaneously. Finally, the approach is based solely on numerical linear algebra rather than minimizing a nonconvex objective function, allowing us to easily analyze and control the algorithm. We present several numerical experiments on both synthetic and real-world datasets to demonstrate the efficiency of our method. In particular, these experiments show that the proposed approach is highly competitive compared with state-of-the-art normalizing flows conditioned on time across a wide range of dimensionalities.

Deep learning algorithms for generative modeling of time-dependent data may face difficulties related to optimal hyperparameter selection and can be computationally expensive. In this work, we instead propose using a generative model based on numerical linear algebra and projection pursuit optimal transport, which requires only a few hyperparameter choices. We demonstrate the effectiveness of our model on synthetic data of stochastic dynamical systems and real-life examples of fish schooling and embryoid growth. These experiments show that our proposed approach is highly competitive with a state-of-the-art deep learning algorithm that learns to sample from time-dependent densities in terms of efficiency, accuracy, and interpretability.

I. INTRODUCTION

A. Motivation

In recent times, there have been several studies that have investigated the use of machine learning algorithms for learning time-varying stochastic processes. Such processes are visible in many applications ranging from geoscience, bioscience, and engineering to computer vision. In particular, deep learning algorithms, such as neural network parameterized normalizing flows, neural ordinary differential equations, diffusion models, and generative adversarial networks, have shown remarkable advances in learning and enabling rapid sampling from these stochastic processes. Such advances are further pronounced for very high-dimensional systems where classical methods are seen to saturate their effectiveness. However, the effective use of deep learning is frequently hampered by difficulties associated with computational cost as well as optimal hyperparameter selection. In this article, we propose a novel approach based on projection-pursuit optimal transport, which learns to sample from the densities of time-varying stochastic processes. It is competitive (both in terms of computational cost and accuracy) with a state-of-the-art deep learning algorithm (given by the neural spline flow). Crucially, our proposed method requires few hyperparameter choices by the user in contrast with most neural network-based methodologies. Thus, our main contributions to this work are as follows:

(1) We implement a projection-pursuit optimal transport-based method to learn maps between time-varying densities from snapshots of particles sampled from these densities.
(2) We utilize a state-of-the-art cubic spline-based interpolation technique for trajectories of samples from the approximated densities, allowing us to smoothly interpolate between training data snapshots in time.
(3) We thoroughly compare our approach with a state-of-the-art neural network parameterized normalizing flow, demonstrating competitive performance for several systems over a wide range of dimensions.

Throughout, a "snapshot" refers to a collection of samples drawn from the underlying density at a fixed time.

B. Related work

Data-driven approaches for constructing models of complex dynamical systems have received considerable attention within several fields of study. Stochastic differential equations (SDEs) are often used to model physical processes whose dynamics are either probabilistic in nature or well approximated by randomness. By leveraging prior knowledge about the functional form of a given SDE, one can use parameter estimation techniques to calibrate governing equations to available sample path data.[1–4] Moreover, advancements in machine learning have led to models that parameterize the drift and diffusion of SDEs by neural networks.[5–7] Other methods include the use of variational inference,[8] which assumes a Gaussian structure for the target probability density function. However, such approaches rely on prior knowledge of the noise process that drives the dynamics and may be infeasible if this information is unavailable.

Notably, Refs. 9–13 overcome this challenge by deploying generative machine learning frameworks based on normalizing flows to learn the time-dependent density of trajectories without imposing any assumptions on a noise process or underlying functional form of the dynamics. A normalizing flow constructs a map from a known reference density to a target density via a sequence of parameterized invertible transformations.[14] These transformations may also rely on the neural ordinary differential equation architecture[15] as the parameterization algorithm in which they are considered continuous normalizing flows.[16] One can also learn a time-dependent target density using the so-called conditional normalizing flows.[17]

Other assumption-free methods for learning time-varying stochastic processes include using generative adversarial networks,[18] which is a class of algorithms where two neural networks compete with each other in the form of a zero-sum game. A discriminator is trained to detect fake samples of the distribution, while a generator attempts to fool the discriminator. Given samples from a training distribution, this technique learns to generate new data with the same statistics as the training set. However, these methods are challenging to train and require significant inductive biases to adapt to scientific applications.[19,20] Newer techniques for learning densities include diffusion models inspired by non-equilibrium thermodynamics, which define a Markov chain of diffusion steps that adds random noise to training data and, subsequently, learn the inversion of this process, using a neural network to generate synthetic data.[21]

Training neural network-based models, such as normalizing flows is compute-intensive, and this is due to the requirement of large-scale nonconvex optimizations as well as expensive hyperparameter and neural architecture searches. Therefore, it is desirable to investigate alternatives that can accelerate workflows reasonably by reducing the number of restarts for new hyperparameters and architecture choices. In the following, we put forth a competitive new approach.

C. Our approach

As mentioned above, normalizing flows often require large neural network architectures, which demand a long training time and careful hyperparameter tuning. Due to these challenges, we propose an alternative approach based upon optimal transport (OT)[22] and projection pursuit,[23] which can circumvent these difficulties. Rather than learning an invertible map between a reference density and several snapshots of samples, we learn an optimal transport map (OTM) from a reference density to the first snapshot of samples and join the remaining snapshots with successive optimal transport maps. This provides a generative model for the dynamics when samples were collected. With additional assumptions on the regularity of the stochastic process, the learned optimal transport maps can also be used to interpolate between snapshots smoothly. The contrast with a normalizing flow-based generative model is illustrated in Fig. 1. Modeling stochastic systems using optimal transport in this way has several computational benefits. Notably, if the inference data are sampled at a sufficiently high frequency, then the optimal transport maps will not deviate significantly from the identity and will, therefore, be cheap to approximate. Moreover, the procedure can be easily paralleled since the approximation of the optimal map between two snapshots is unaffected by the optimal maps at other times. While OT has been used to model time-dependent densities in Refs. 24 and 25 and has also been combined with normalizing flows,[26] our approach investigates how it may be deployed for high-dimensional systems, representative of real-world applications, by using projection pursuit regression, thereby making it competitive with techniques such as the normalizing flow.

We accelerate the process of finding the OT map using the Projection Pursuit Monge Map (PPMM) algorithm presented in Ref. 27. The approach is inspired by projection pursuit regression[23] as it seeks to estimate high-dimensional optimal transport maps by learning several one-dimensional optimal transport maps along the most "informative" projection directions. The PPMM algorithm is an iterative approach, and at the *k*th step of the algorithm, the projection direction is selected to maximize the discrepancy [measured using the so-called sliced average variance estimation (SAVE)[28]] between projections of the current samples and target samples. The PPMM algorithm empirically exhibits fast convergence and scales well to high-dimensional problems. It must be noted that the PPMM procedure used to approximate optimal transport maps is based solely on numerical linear algebra instead of minimizing a nonconvex objective function. It can be analyzed and controlled using standard tools in numerical analysis.

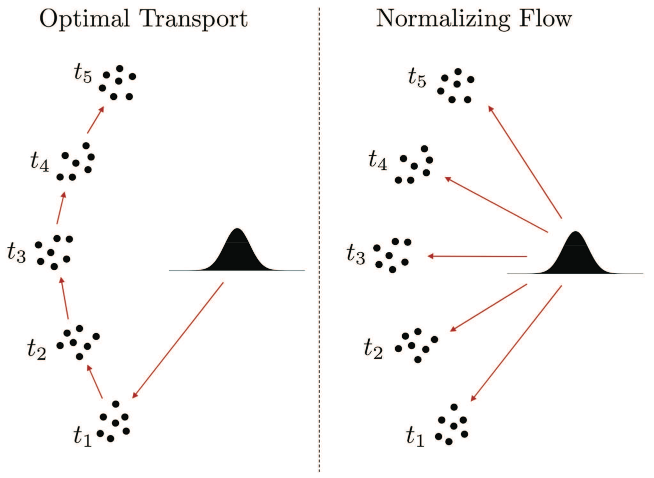

FIG. 1. Illustration of the two generative models. The left panel shows a generative model obtained by mapping between successive snapshots via optimal transport maps, whereas the right panel shows the generative model obtained by mapping a known reference distribution to each snapshot.

After using PPMM to learn optimal transport maps between snapshots, we use the transport splines framework in Ref. 25 to quickly interpolate the samples. While various methods for performing interpolation in Wasserstein space exist, e.g., Ref. 24, we elect to use transport splines for their ease of implementation and computational feasibility. Indeed, the transport splines approach reduces the problem of interpolating a curve of measures to the Euclidean interpolation of Lagrangian trajectories, which are determined by the optimal coupling between snapshots. Through several numerical experiments on synthetic and real-world datasets, we illustrate that the proposed method exhibits faster convergence and higher accuracy in many situations than state-of-the-art neural spline flows.[29] As the dimensionality of the state space becomes sufficiently large (e.g., $d = 30$), we observe that this performance gap becomes small, but that our proposed method is still more computationally efficient.

The rest of the paper is structured as follows. In Sec. II, we review the necessary background on optimal transport, the PPMM algorithm, and normalizing flows. In Sec. III, we discuss how we adapt both PPMM and normalizing flows to the time-dependent setting and review the generative algorithms shown in Fig. 1 in detail. In Sec. IV, we provide numerical results based on synthetic snapshot data from high-dimensional Fokker–Planck equations and real-world biological applications to fish schooling and embryoid growth. Conclusions follow in Sec. V.

II. BACKGROUND

This section provides an overview of optimal transport and normalizing flows as tools for constructing generative models of stationary densities. In Sec. II A, we review the theory of optimal transport and introduce the PPMM algorithm. In Sec. II C, we provide an overview of normalizing flows as well as the neural spline flow architecture,[29] which we use for our numerical experiments in Sec. IV.

A. Projection pursuit Monge map

The projection pursuit Monge map (PPMM) algorithm introduced in Ref. 27 provides an efficient approach to computing optimal transport maps in high dimensions. The method utilizes tools from projection pursuit regression,[23] as well as the sufficient dimension reduction technique of sliced average variance estimation (SAVE).[28] At each iteration, the SAVE procedure is used to choose a projection direction that maximizes the discrepancy between the variance of projections of the current and target samples. A one-dimensional OT map is then learned along this projection direction and is used to transport the collection of current samples accordingly. To discuss the PPMM algorithm more precisely, we now introduce a few concepts from the theory of optimal transport.

Let $X : \mathbb{R}^d \to \mathbb{R}$ and $Y : \mathbb{R}^d \to \mathbb{R}$ be two continuous random variables, with corresponding densities p_X and p_Y. We say that a measurable function $\phi : \mathbb{R}^d \to \mathbb{R}^d$ is a transport map between p_X and p_Y if for all Borel measurable sets B, it holds that $\mu_X(\phi^{-1}(B)) = \mu_Y(B)$, where μ_X and μ_Y denote the measures induced by the densities p_X and p_Y, respectively. Since μ_X and μ_Y are absolutely continuous by assumption, we also have that ϕ is a transport map if ϕ is invertible and (1) holds

$$p_Y(x) = p_X(\phi^{-1}(x))|\det D\phi^{-1}(x)|. \tag{1}$$

Though there may be many such transport maps, the goal is to recover one that is optimal in some sense. Toward this, we introduce a function $c(x, y)$, which represents the cost of transporting a point mass $x \in \mathbb{R}^d$ to another point mass $y \in \mathbb{R}^d$. Given a transport map $\phi : \mathbb{R}^d \to \mathbb{R}^d$ and cost $c : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, we define the overall transport cost as

$$\int_{\mathbb{R}^d} c(x, \phi(x)) p_X(x) dx. \tag{2}$$

Typically, one fixes $p \in [1, \infty)$ and uses the cost $c(x, y) = |x - y|^p$, where $|\cdot|$ denotes the Euclidean norm. Provided that X and Y have finite moments of order p, this construction gives rise to the p-Wasserstein distance

$$W_p(p_X, p_Y) := \left(\inf_{\phi \in \Phi} \int_{\mathbb{R}^d} |x - \phi(x)|^p p_X(x) dx \right)^{1/p}. \tag{3}$$

In (3), Φ denotes the collection of all transport maps from p_X to p_Y. For a particular choice of $p \in [1, \infty)$, we will hereafter denote by $\phi_* \in \Phi$ the transport map for which the infimum (3) is achieved.

The optimal transport map ϕ_* and p-Wasserstein distance are popular tools in the machine learning community for training generative models. While (3) can be used as the loss function to train a given model,[30] one can also use the mapping ϕ_* as the generative model itself.[31] We adopt the latter approach. Given N samples $\mathbf{X}$, $\mathbf{Y} \in \mathbb{R}^{N \times d}$ drawn from the densities p_X and p_Y respectively, the PPMM algorithm constructs k-step approximations $\widehat{\phi}_k$ (see Algorithm 1) and $\widehat{W}_2^{(k)}$ [see (5)] of the true OT-map ϕ_* and the 2-Wasserstein distance (3), respectively.

We remark that the output $\widehat{\phi}_k$ of the PPMM algorithm is defined recursively by setting $\widehat{\phi}_0(x) = x$ and

Algorithm 1. Projection Pursuit Monge Map (PPMM)[27]

```
Input: $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{N\times d}$.
Set $\mathbf{X}_0 = \mathbf{X}$ and $k = 0$.
Until convergence:
    (a) Use SAVE to compute the projection direction $p_k \in \mathbb{R}^d$ between $\mathbf{X}_k$ and $\mathbf{Y}$.
    (b) Compute the one-dimensional OT-map $\eta_k$ between the projections $\mathbf{X}_k p_k$ and $\mathbf{Y} p_k$.
    (c) Set $\mathbf{X}_{k+1} = \mathbf{X}_k + \big(\eta_k(\mathbf{X}_k p_k) - \mathbf{X}_k p_k\big)p_k^T$ and $k = k + 1$.
Output: $\{p_\ell\}_{\ell=0}^{k-1}$ and $\{\eta_\ell\}_{\ell=0}^{k-1}$, which form the OT approximation $\widehat{\phi}_k : \mathbb{R}^d \to \mathbb{R}^d$ given in (4).
```

$$\widehat{\phi}_{\ell+1}(x) = \widehat{\phi}_\ell(x) + \left(\eta_\ell\left(\widehat{\phi}_\ell(x)p_\ell\right) - \widehat{\phi}_\ell(x)p_\ell\right)p_\ell^T, \qquad 0 \le \ell \le k-1. \tag{4}$$

In step (a) of Algorithm 1, the projection direction p_k is chosen to maximize the discrepancy between $\mathrm{Var}(\mathbf{X}^{(k)}p_k)$ and $\mathrm{Var}(\mathbf{Y}p_k)$. In step (b), a one-dimensional (1D) OT-map η_k is computed between the projected samples $\mathbf{X}_k p_k$ and $\mathbf{Y}p_k$. Finally, in step (c), the collection $\mathbf{X}_k$ of current samples is updated according to the transport map η_k. We note that in Ref. 27, the 1D OT maps $\{\eta_\ell\}_{\ell=0}^{k-1}$ from (b) are computed via sorting and interpolation. This is because the 1D optimal transport problem has an analytical solution, which the Quicksort algorithm can efficiently compute in $\mathcal{O}(N \log N)$ time. In Sec. II B, we propose an alternative approach for computing the 1D maps η_ℓ, which is more efficient when N is large, acts as a helpful regularization when N is small, and extends the domain of η_ℓ to all of $\mathbb{R}$. By modeling each η_ℓ as a mapping from $\mathbb{R} \to \mathbb{R}$, we then have that the approximation $\widehat{\phi}_k$ is a mapping from $\mathbb{R}^d \to \mathbb{R}^d$, as given in (4). This perspective motivates the use of PPMM as a generative model. We remark that weak convergence of Algorithm 1 has been proven (see Ref. 27, Theorem 3) and that its computational complexity has been studied (see Ref. 27, Sec. 2).

In our implementation of PPMM in Sec. IV, we also enforce a relative stopping criterion that terminates the algorithm after convergence of the approximate 2-Wasserstein distance

$$\widehat{W}_2^{(k)}(\mathbf{X}, \mathbf{Y}) := \left(\frac{1}{N}\sum_{i=1}^{N}\left\|\widehat{\phi}_k(x_i) - x_i\right\|^2\right)^{1/2} \tag{5}$$

begins to slow down. That is, we specify a tolerance $\alpha \in [0, 1]$ and terminate PPMM when

$$\frac{\left|\widehat{W}_2^{(k(\alpha))}(\mathbf{X}, \mathbf{Y}) - \widehat{W}_2^{(k(\alpha)-1)}(\mathbf{X}, \mathbf{Y})\right|}{\left|\widehat{W}_2^{(k(\alpha))}(\mathbf{X}, \mathbf{Y})\right|} \le \alpha, \tag{6}$$

where $k(\alpha) \in \mathbb{Z}_+$ is the last iteration number. Note that the convergence criteria (6) does not rely on a knowledge of $W_2(p_X, p_Y)$. Thus, we can use it in applications where the ground truth solution is unknown. Moreover, since we will ultimately be interested in applying PPMM to many pairs of snapshots simultaneously, using the convergence criteria (6) ensures that we spend most of our computational efforts learning the optimal transport maps along the dominant directions. While we have stated the PPMM algorithm for the case when $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{N\times d}$, we note that it can also be used to learn the OTM between distributions when the sample sizes are not equal.

B. Regularizing the optimal transport maps

The one-dimensional optimal maps η_ℓ in Algorithm 1 are computed according to the quicksort algorithm, which runs in $\mathcal{O}(N \log N)$ time. However, when N is sufficiently large, it can be computationally advantageous to instead form histogram approximations using $B \ll N$ cells of the underlying samples and learn the optimal map according to the weighted histogram cell centers. More specifically, assume we want to learn the optimal map between densities $\rho_1, \rho_2 : \mathbb{R} \to [0, \infty)$ from the observed samples $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$ with $\mathbf{x} \sim \rho_1$ and $\mathbf{y} \sim \rho_2$. The true optimal map is given by $G^{-1} \circ F$, where F and G denote the cumulative distribution functions (CDFs) of ρ_1 and ρ_2, respectively.[32] Thus, we use a kernel density estimate (KDE) based upon the Gaussian kernel to obtain approximations $\hat{F}$ and $\hat{G}$ of the CDFs and compute the underlying optimal map as

$$\eta(x) := \begin{cases} f_2(f_1(x)) & x \in \Upsilon \\ \mathrm{Id}(x) & x \notin \Upsilon \end{cases}, \qquad f_1 := \mathtt{Interp}(\mathbf{z}, \hat{F}), \qquad f_2 := \mathtt{Interp}(\hat{G}, \mathbf{z}). \tag{7}$$

In (7), $\mathtt{Interp}(\cdot, \cdot)$ denotes a piecewise linear interpolation and $\mathbf{z} \in \mathbb{R}^B$ is a vector containing B equally spaced histogram cell centers over the interval $\Upsilon := [\min\{\mathbf{x}, \mathbf{y}\} - L, \max\{\mathbf{x}, \mathbf{y}\} + L] \subseteq \mathbb{R}$. The constant $L > 0$ should be chosen large enough such that the effective support of the approximate densities is strictly contained in Υ. We remark that the KDE of the binned samples can be evaluated at the equidistant grid points $\mathbf{z} \in \mathbb{R}^B$ in $\mathcal{O}(B \log B)$ time based on an approach using the fast Fourier transform (FFT).[33] Moreover, to enforce strict monotonicity of the approximate CDFs, $\hat{F}$ and $\hat{G}$, we rescale the densities obtained from the KDE by adding a small positive constant $\varepsilon > 0$ and renormalizing.

The density estimation can heavily depend on the KDE bandwidth, so the bandwidth should be chosen carefully and in accordance with the observed data. A bandwidth that is too small will overfit the training data, whereas a bandwidth that is too large may result in an over-smoothed distribution. While the use of an FFT-based KDE can be computationally advantageous when modeling the OT maps η_ℓ for large values of N, it can also serve as a useful regularization in the low-data limit. Indeed, when N is small, the selection of a sufficiently large bandwidth for the KDE effectively injects additional noise in the under-representative training samples. This helps compensate for the small number of observations and prevents overfitting. Otherwise, we select the bandwidth according to either Scott's method[34] when the data are close to unimodal or the improved Sheather–Jones (ISJ) method[35] when the data are multimodal.

C. Neural spline flows

Normalizing flows deliver alternative approaches to the generative modeling of samples drawn from a probability distribution. While the PPMM algorithm only provides a generative model for samples of the underlying density, a normalizing flow learns the density itself. Moreover, while OT-based methods seek to construct an optimal mapping between samples between reference and target densities, normalizing flows learn a transport map that is not necessarily optimal in the sense of (3).

We now describe the optimization procedure by which a normalizing flow learns a transport map from a known reference density to a target density. Specifically, assume that $p_X : \mathbb{R}^d \to \mathbb{R}$ is the probability density function of the continuous random variable $X : \mathbb{R}^d \to \mathbb{R}$ and that $g : \mathbb{R}^d \to \mathbb{R}^d$ is invertible with $f := g^{-1}$. For the random variable $Y := g \circ X$ with corresponding density p_Y, the change of variables formula for probability densities is given in (1) with the map ϕ replaced by g here. By rewriting $g = g_K \circ \cdots \circ g_1$ as the composition of K bijective functions, the determinant $\det Df$ can be evaluated as

$$\det Df(x) = \prod_{i=1}^{K} \det Df_i(x_i), \qquad x_i := \begin{cases} x & i = K \\ (f_{i+1} \circ \cdots \circ f_K)(x) & 1 \le i < K \end{cases}, \qquad f_i := g_i^{-1}.$$

If one samples the data matrix $\mathbf{Y} \in \mathbb{R}^{N \times d}$ from an underlying density p_Y, a normalizing flow can be used to estimate p_Y by parameterizing the family of bijections $g = g_\theta$ by $\theta \in \Theta$ and performing the optimization

$$\max_{\theta \in \Theta} \mathcal{L}_\theta, \qquad \mathcal{L}_\theta := \frac{1}{N} \sum_{n=1}^{N} \Big(\log p_X(f_\theta(\mathbf{Y}_n)) + \log \big| \det Df_\theta(\mathbf{Y}_n) \big| \Big). \tag{8}$$

In (8), we have written $\mathbf{Y}_n$ to denote the nth sample of $\mathbf{Y}$ and again $f_\theta := g_\theta^{-1}$. The density p_X is a known reference distribution (usually taken to be Gaussian). During training, the flow learns to transport the samples $\mathbf{Y}$ to samples of p_X via the mapping f_θ. Since f_θ is bijective with an inverse g_θ, the flow can also be used to generate new samples from p_Y once it has been trained, simply by applying the mapping g_θ to new samples drawn from p_X.

We use neural spline flows[29] for their state-of-the-art capabilities in modeling high-dimensional densities. Specifically, we study neural spline flows based upon coupling transforms $\psi_m : \mathbb{R}^d \to \mathbb{R}^d$ of the form

$$\psi_m(x_1, \dots, x_{m-1}, x_m, \dots, x_d) = (x_1, \dots, x_{m-1}, h_{\theta_m}(x_m), \dots, h_{\theta_d}(x_d)), \tag{9}$$

where the family $\{h_{\theta_j}\}_{j=m}^{d}$ of functions are invertible and parameterized by $(\theta_m, \dots, \theta_d)$, which is set as the output of a neural network with inputs $\{x_1, \dots, x_{m-1}\}$, i.e., $\mathrm{NN}(x_1, \dots, x_{m-1})$. Essentially, (9) splits an input vector at the index $1 \le m \le d$ and applies a transformation parameterized by $(x_1, \dots, x_{m-1})$ to the entries $(x_m, \dots, x_d)$. It is also common to permute the entries of the input $x \in \mathbb{R}^d$ before passing through the transform (9). Coupling transforms enable a seamless computation of the Jacobian determinant, as $D\psi_m$ is by construction block triangular. Indeed, the Jacobian determinant can be evaluated as the product

$$\det D\psi_m = \prod_{j=m}^{d} \frac{\partial h_{\theta_j}}{\partial x_j}.$$

To improve the approximation power of coupling transforms (9), neural spline flows parameterize the functions h_{θ_j} as families of monotonic rational-quadratic splines. To allow all parameters of the flow to interact with one another, the flow architecture typically alternates between spline-based coupling layers and linear layers. The linear layers are based upon the *PLU* decomposition, where at the beginning of training, the permutation matrix P is fixed and the matrices L and U are learned. As in Ref. 29, a single step of the flow consists of a spline layer and a linear layer.

Following Ref. 29, we refer to the number of knots in the monotonic rational-quadratic splines h_{θ_j} as the number of bins. A tail bound of $b > 0$ is prescribed for the splines h_{θ_j} such that the functions $h_{\theta_j}(x)$ are extended to be strictly linear when $|x| > b$. This allows the flow to evaluate unconstrained inputs (see Ref. 29, Fig. 1). Moreover, the neural network, which computes the parameters θ_j in the coupling transform, is taken as a residual network with pre-activation blocks. Our experiments in Sec. IV utilize ten-step RQ-NSFs with eight bins, two residual blocks with 128 hidden features, and a tail bound of $b = 3$. This is the same architecture that achieved state-of-the-art density estimation results in Ref. 29. Since we are interested in efficient density estimation, we also test a Reduced RQ-NSF architecture, which consists of only two steps. This architecture can be trained quickly and provides an expedient means for calculating likelihoods. However, it reduces the approximation power of the flow compared to the ten-step framework.

III. METHODS

In this section, we formally introduce dynamic PPMM and adapt the RQ-NSF framework to the time-dependent setting. We begin in Sec. III A by discussing an alternative approach to computing the one-dimensional OT maps in the PPMM algorithm. We then proceed by formally stating the dynamic PPMM algorithm, reviewing the transport splines algorithm presented in Ref. 25 and commenting on the computational complexity of training the model. We conclude in Sec. III B by explaining how conditional normalizing flows[36] can be used to adapt the neural spline flow framework from Sec. II C to the time-dependent setting.

A. Dynamic PPMM

We will assume that $\rho(x,t)$ is a time-dependent density on $\mathbb{R}^d$ and that at each observation time $t_1 < t_2 < \cdots < t_M$, we draw N_j samples $\mathbf{X}^{(t_j)} \in \mathbb{R}^{N_j \times d}$ from $\rho(x,t_j)$. We now aim to use our collection $\{\mathbf{X}^{(t_j)}\}_{j=1}^M$ of time-dependent samples to sample from $\rho(x,t)$, for any $t \in [t_1, t_M]$. Our proposed approach involves using the PPMM algorithm to join successive snapshots via optimal transport maps and, subsequently, applying a Euclidean interpolation algorithm to individual trajectories between the snapshots.

1. The algorithm

We now state the Dynamic PPMM algorithm, which for convenience, will be frequently referred to as D-PPMM.

Note that $\mathbf{Y}^{(t_0)}$ contains $\tilde{N}$ samples, where $\tilde{N}$ is the number of new samples we wish to generate using D-PPMM. The D-PPMM algorithm has only a handful of tunable parameters, most of which are related to the construction of the one-dimensional optimal maps (see (7)). The hyperparameters that can affect the performance of Algorithm 2 include the convergence tolerance $\alpha \in [0,1]$, the KDE bandwidth selection method, the number of histogram cells B, the base normal distribution $\mathcal{N}(\nu, \Gamma)$, and the constants $L, \varepsilon > 0$ (see Sec. II B). We find that the algorithm's performance is relatively insensitive to the choice of ν, Γ, L, and ε. Thus, throughout our experiments in Sec. IV, we leave the base normal distribution fixed with $\nu = 0$ and $\Gamma = 0.01I$ where I is the identity matrix, and we set $\varepsilon = 10^{-8}$ and $L = 0.1$ or $L = 0.25$.

2. Interpolation via transport splines

To extend Algorithm 2 to a generative model for $\rho(x,t)$, for any $t \in [t_1, t_M]$, we use the transport splines algorithm introduced in Ref. 25. This approach involves first coupling the snapshots via optimal transport maps, e.g., computing the output $\{\mathbf{Y}^{(t_j)}\}_{j=1}^M$ of Algorithm 2. Then, for each $n \in \{1, \ldots, \tilde{N}\}$, the trajectories $\{(t_j, \mathbf{Y}_n^{(t_j)})\}_{j=1}^M$ are interpolated in Euclidean space via cubic splines.

Algorithm 2. Dynamic PPMM (D-PPMM)

```
**Input:** Measurement times $\{t_j\}_{j=1}^M$ and samples $\{\mathbf{X}^{(t_j)}\}_{j=1}^M$.
**Training:**
  Sample the rows of $\mathbf{X}^{(t_0)} \in \mathbb{R}^{N_1 \times d}$ i.i.d. from $\mathcal{N}(\nu, \Gamma)$.
  **For** $j \in \{0, \ldots, M-1\}$ **do**:
    Use PPMM (Algorithm 1) to estimate the OTM $\widehat{\phi}_{k_j(\alpha)}^{(t_j)}$ between
      $\mathbf{X}^{(t_j)}$ and $\mathbf{X}^{(t_{j+1})}$,
    where each 1D OTM is given by the procedure in Sec. II B.
**Testing:**
  Sample the rows of $\mathbf{Y}^{(t_0)} \in \mathbb{R}^{\tilde{N} \times d}$ i.i.d. from $\mathcal{N}(\nu, \Gamma)$.
  **For** $j \in \{1, \ldots, M\}$ **do**:
    Assign $\mathbf{Y}^{(t_j)} = \widehat{\phi}_{k_j(\alpha)}^{(t_{j-1})}(\mathbf{Y}^{(t_{j-1})})$
**Output:** New samples $\{\mathbf{Y}^{(t_j)}\}_{j=1}^M$.
```

That is, for each fixed n, we compute its spline interpolant $p_n : [t_1, t_M] \to \mathbb{R}^d$,

$$p_n := \texttt{Spline}\Big([t_1, t_2, \ldots, t_M], [\mathbf{Y}_n^{(t_1)}, \mathbf{Y}_n^{(t_2)}, \ldots, \mathbf{Y}_n^{(t_M)}]\Big), \tag{10}$$

where $\texttt{Spline}(\cdot,\cdot)$ denotes a cubic B-spline interpolation. Then, for arbitrary $t \in [t_1, t_M]$, we form the interpolating snapshot $\mathbf{Y}^{(t)}$ by evaluating

$$\mathbf{Y}^{(t)} = \begin{bmatrix} — & p_1(t) & — \\ — & p_2(t) & — \\ & \vdots & \\ — & p_{\tilde{N}}(t) & — \end{bmatrix} \in \mathbb{R}^{\tilde{N} \times d}. \tag{11}$$

We remark that no regularity assumptions are required if we simply aim to construct a generative model at the measurement times $t_1 < t_2 < \cdots < t_M$, as in Algorithm 2. However, additional knowledge of the underlying stochastic process may be required to justify the use of cubic spline interpolation between measurement times, as the approach[25] was devised for stochastic processes with C^2 sample paths. Thus, the use of transport spline interpolation is not expected to be effective for discontinuous stochastic processes, such as Cauchy noise or a jump process. Therefore, our application of transport splines to real data in Secs. IV D and IV E assumes sufficient regularity of the underlying processes. The transport splines algorithm has theoretical approximation guarantees (see Ref. 25, Theorem 2) and is computationally expedient as it reformulates the problem of interpolating measures into the Euclidean interpolation of individual trajectories. As the time between observations decreases, so does the approximation power of the transport spline interpolation.

3. Computational complexity

We now discuss the computational requirements for training Algorithm 2. For simplicity, we will assume that the Scott bandwidth selection rule is used, and we will write $k_j(\alpha) = k_j$ as the number of PPMM iterations, which are performed at each timestep. Using PPMM to estimate each optimal map $\hat{\phi}_{k_j}^{(t_j)}$ incurs a cost of $\mathcal{O}(k_j N_j d^2 + k_j B \log B)$ (see Ref. 27 and Sec. II B). We also remark that empirical evidence suggests that $k_j = \mathcal{O}(d)$ is expected to yield good convergence of the PPMM algorithm (see Ref. 27, Fig. 8). The first term in the complexity bound is due to the computation of the most informative projection direction via the SAVE algorithm, whereas the second term is the cost associated with learning a one-dimensional optimal map via the procedure outlined in Sec. II B. Thus, if we let $K = \max_{0 \le j \le M-1} k_j$ and $N = \max_{0 \le j \le M-1} N_j$, we have that the total cost of training D-PPMM is given by $\mathcal{O}(KNd^2 + KB \log B)$. Note that in this analysis, the cost associated with pushing the N_j samples through the OT maps at each step of training, as well as the KDE bandwidth selection are absorbed in the $\mathcal{O}$-notation.

B. Time-conditioned RQ-NSF

We now discuss how the normalizing flows from Sec. II C may be adapted to the time-dependent setting. As in Sec. III A, we consider the problem of learning the temporal density $\rho(x, t)$ when $t \in [t_1, t_M]$, given the observations $\{\mathbf{X}^{(t_j)}\}_{j=1}^{M}$ at the times $\{t_j\}_{j=1}^{M}$. Similar to Ref. 11, we accomplish this by viewing the underlying density as a distribution conditioned upon the time t, which we then model via a conditional normalizing flow.[17] In analogy with (1), we now seek a mapping $f : \mathbb{R}^d \times [t_1, t_M] \to \mathbb{R}^d$ such that

$$\rho(x, t) = p_{X|t}(f(x, t)) |\det D_x f(x, t)|, \qquad x \in \mathbb{R}^d, \qquad t \in [t_1, t_M], \tag{12}$$

where $f(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ is a bijection for each fixed $t \in [t_1, t_M]$. Note that in (12), the base distribution $p_X = p_{X|t}$ is now conditioned on time as well. Our goal is to learn a suitable base distribution $p_{X|t}$ and bijection $f(x, t)$ from the data $\{\mathbf{X}^{(t_j)}\}_{j=1}^{M}$ such that (12) holds. Toward this, we use **nflows**[36] to simultaneously train an RQ-NSF (see Sec. II C) along with two additional feed-forward networks, $\mathrm{NN}^{(1)} : \mathbb{R} \to \mathbb{R}$ and $\mathrm{NN}^{(2)} : \mathbb{R} \to \mathbb{R}^{2d}$, which learn the time-conditioning in the bijection $f(x, t)$ and the base distribution $p_{X|t}$, respectively. More specifically, the function $f(x, t)$ is modeled as an RQ-NSF with an additional dimension concatenated to account for the sampling time, the inputs of which are given by $\mathrm{NN}^{(1)} : \mathbb{R} \to \mathbb{R}$. Moreover, we parameterize $p_{X|t}$ as a conditional diagonal normal distribution $\mathcal{N}(\nu(t), \mathrm{diag}(\sigma(t)))$ and model the functions $\nu : \mathbb{R} \to \mathbb{R}^d$ and $\sigma : \mathbb{R} \to \mathbb{R}^d$ using $\mathrm{NN}^{(2)} : \mathbb{R} \to \mathbb{R}^{2d}$.

We use mini-batch training in which each step of the optimization includes exactly the same number of samples from each observation time t_j. We also use the Adam optimizer and decay the learning rate from 5×10^{-4} to 0 on a cosine schedule every five epochs. Moreover, we parameterize $\mathrm{NN}^{(1)}$ as a four layer fully connected ReLU network with 100 nodes in each layer, and we parameterize $\mathrm{NN}^{(2)}$ as a fully connected ReLU network with a single layer of 500 nodes. We note that the network architecture of both time-conditioned Reduced RQ-NSF and time-conditioned standard RQ-NSF have hundreds of thousands of tunable parameters. There are clearly also many user-specified hyperparameters that can affect the performance of the time-conditioned normalizing flow. We have tried to make our conditional RQ-NSFs competitive through a manual hyperparameter search. In this regard, a clear benefit of the D-PPMM algorithm is that it has only a select few hyperparameters which can be tuned (see Sec. III A 1).

To interpolate the density $\rho(x, t)$ between two measurement times $\{t_j, t_{j+1}\}$ using the conditioned normalizing flow, we simply evaluate the right hand side of (12) for $t \in [t_j, t_{j+1}]$. Note that the interpolation technique here differs from Sec. III A, where we instead utilized the optimal coupling between snapshots to interpolate individual trajectories.

IV. NUMERICAL RESULTS

In this section, we compare D-PPMM with a time-conditioned RQ-NSF through several numerical experiments (our implementations of D-PPMM and the time-conditioned RQ-NSF are publicly available at https://github.com/jrbotvinick/Dynamic-PPMM). All experiments were conducted using an Intel i7-1165G7 CPU. In Sec. IV A, we introduce the synthetic dynamical system examples and explain the data generation process. In Sec. IV B, we present results comparing the two methods in the low-dimensional setting when $d < 5$. In Sec. IV C, we move onto the high-dimensional setting and study the convergence behavior of the generative models when $d \in \{10, 20, 30\}$. Finally, in Secs. IV D and IV E, we study the interpolation capabilities of D-PPMM applied to two real-world datasets from biological applications and compare the results, both in terms of speed and accuracy, with the interpolation of a conditioned RQ-NSF.

A. Data preparation and experimental setup

Throughout all experiments, we rescale the sample data to belong to the cube $[-1, 1]^d \subseteq \mathbb{R}^d$ via an affine transformation. We also rescale the trajectory time interval to $[0, 1] \subseteq \mathbb{R}$. We now discuss the preparation of the data used in Secs. IV B and IV C. Throughout, we seek generative models for the sample paths of stochastic differential equations of the form

$$dX_t = \nu(X_t; \theta) dt + \sqrt{2D} dW_t, \tag{13}$$

where $\nu(\cdot; \theta)$ denotes a parameter-dependent velocity, $D > 0$ is the diffusion, which for simplicity, we assume to be constant, and W_t denotes a Brownian motion. We remark that the sample paths (13) are described by the probability flow of a Fokker–Planck equation, i.e., a density $\rho(x, t)$ governed by the partial differential equation (PDE),

$$\frac{\partial}{\partial t} \rho(x, t) + \nabla \cdot (\rho(x, t) \nu(x; \theta)) = D \Delta \rho(x, t). \tag{14}$$

We will denote by N the number of sample paths of (13), which evolve over the time interval $[0, T]$. To obtain a numerical approximation of (13), we use the Euler–Maruyama discretization (Ref. 37, Sec. 5.2), which assigns

$$\mathbf{X}_n^{j+1} = \mathbf{X}_n^j + \nu(\mathbf{X}_n^j) \Delta t + \xi_j \sqrt{2D\Delta t}, \tag{15}$$

TABLE I. Equations and parameters for the synthetic dynamical systems studied throughout this section.

System	ν	D	θ	$\mathbf{X}^{(t_0)}$	T
Van der Pol	$\dot{x}_1 = x_2$ $\dot{x}_2 = c(1 - x_1^2)x_2 - x_1$	2.5×10^{-3}	$c = 1$	$\mathcal{N}(\nu, \sigma^2 I)$ $\nu = (1, 1)$ $\sigma = 5 \times 10^{-2}$	6
OU-process	$\dot{x}_i = -\lambda x_i$ $1 \leq i \leq d, \quad d \geq 2$	5×10^{-2}	$\lambda = 10^{-1}$	$\frac{1}{2}\left(\mathcal{N}(\nu_1, \sigma^2 I) + \mathcal{N}(\nu_2, \sigma^2 I)\right)$ $\nu_i = (10(-1)^i, 10, \ldots, 10)$ $\sigma = 5 \times 10^{-2}$	15
Lorenz-96	$\dot{x}_i = (x_{i+1} - x_{i-2})x_{i-1} - x_i + F$ $x_{-1} = x_{d-1},\, x_0 = x_d,\, x_{d+1} = x_1$ $1 \leq i \leq d, \quad d \geq 4$	5×10^{-3}	$F = 2$	$\mathcal{N}(\nu, \sigma^2 I)$ $\nu = (4, 0, \ldots, 0)$ $\sigma = 10^{-1}$	≤ 5

where $\xi_j \sim \mathcal{N}(0, I)$ are i.i.d from the standard d-dimensional normal distribution and $\Delta t > 0$.

Throughout our experiments in Secs. IV B and IV C, we study the Van der Pol oscillator, an Ornstein–Uhlenbeck process (OU-process), and the Lorenz-96 system with additive stochastic forcing. While the Van der Pol oscillator has a fixed dimensionality of $d = 2$, we can manually adjust the dimensionality of the OU-process and Lorenz-96 system. Table I details the specific parameter choices, diffusion coefficients, and initializations that we use throughout our experiments. After approximating the sample paths (13) via (15) over the time interval $[0, T]$, we select M snapshots $\{\mathbf{X}^{(t_j)}\}_{j=1}^{M}$ evenly spaced in time as the inference data during our experiments. Each experiment in Secs. IV B and IV C uses a total of $M = 11$ snapshots and $N = 10^4$ samples per snapshot.

We now discuss the process by which we quantify the error of the two generative models. Both models are trained using the same inference samples, while error is quantified by comparing new samples from the generative models with a separate held-out testing set from the ground truth dynamical systems. The error in the generated samples from a given model is then determined using the maximum mean discrepancy (MMD). Given collections of samples $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{N \times d}$, where $\mathbf{X} \sim \mu$ and $\mathbf{Y} \sim \nu$, Ref. 38, Lemma 5 characterizes the empirical MMD as

$$\mathrm{MMD}_{\sigma}^{2}(\mathbf{X}, \mathbf{Y}) := \frac{1}{N(N-1)} \sum_{n=1}^{N} \sum_{m \neq n}^{N} \Big(\gamma_{\sigma}(\mathbf{X}_n, \mathbf{X}_m) + \gamma_{\sigma}(\mathbf{Y}_n, \mathbf{Y}_m) \Big) - \frac{2}{N^2} \sum_{n=1}^{N} \sum_{m=1}^{N} \gamma_{\sigma}(\mathbf{X}_n, \mathbf{Y}_m), \tag{16}$$

where we denote by $\mathbf{X}_n$ the n-th row (or sample) of $\mathbf{X}$ and γ_σ is the Gaussian kernel with bandwidth σ. An analogous formulation also holds when the data matrices $\mathbf{X}$ and $\mathbf{Y}$ differ in size. Given that the complexity of (16) is $\mathcal{O}(N^2)$, we instead use the so-called linear-MMD approximation (see Ref. 38, Lemma 14) when N is large (e.g., $N = 10^4$ in Secs. IV B and IV C). Moreover, due to the sensitivity of the MMD metric on the choice of kernel bandwidth σ, we follow Ref. 39 by computing the so-called generalized MMD. This involves evaluating (16) for a finite collection of bandwidths $\mathcal{I} \subseteq \mathbb{R}$ and reporting the maximum value. Throughout our tests, we choose the elements of $\mathcal{I}$ logarithmically spaced between 10^{-2} and 10^{2} with $|\mathcal{I}| = 15$. In situations when we compare the accuracy of the generative models at only a single snapshot (see Sec. IV E), we report the Generalized MMD, which we denote by $\mathrm{GMMD}_{\mathcal{I}}^{2}$. Moreover, when we compare the accuracy of the generative models at several snapshots (see Secs. IV B–IV D), we report the average of the Generalized MMD across all such snapshots, which we denote by $\widetilde{\mathrm{GMMD}}_{\mathcal{I}}^{2}$.

B. Low-dimensional comparison

We begin by comparing the speed and accuracy of D-PPMM with both the standard RQ-NSF and Reduced RQ-NSF architectures conditioned on time. Specifically, we will consider inference sample path data $\{\mathbf{X}^{(t_j)}\}_{j=1}^{M}$ arising from the Van der Pol oscillator ($d = 2$), an OU-process ($d = 3$), and the Lorenz-96 system ($d = 4$); see Sec. A. For these experiments, the terminal time of the Lorenz-96 system is taken as $T = 5$. Given that the problem of interpolation is highly application-dependent, we will first be concerned with quantifying the speed and accuracy of these frameworks as generative models exactly at the measurement times $\{t_j\}_{j=1}^{M}$.

Figure 2 shows an initial qualitative comparison between D-PPMM and a time-conditioned Reduced RQ-NSF, in which both models were permitted to train for exactly the same amount of wall-clock time. This was achieved by choosing a convergence tolerance $\alpha > 0$, recording the wall-clock time t_α after which D-PPMM finishes training, and then training a time-conditioned RQ-NSF for exactly the same amount of time t_α. Throughout our comparisons, we do not track the time during which the RQ-NSF framework partitions the training set into mini-batches. In Fig. 2, we see for each test that D-PPMM produces new samples, which appear more representative of the original dynamics and visually have smaller

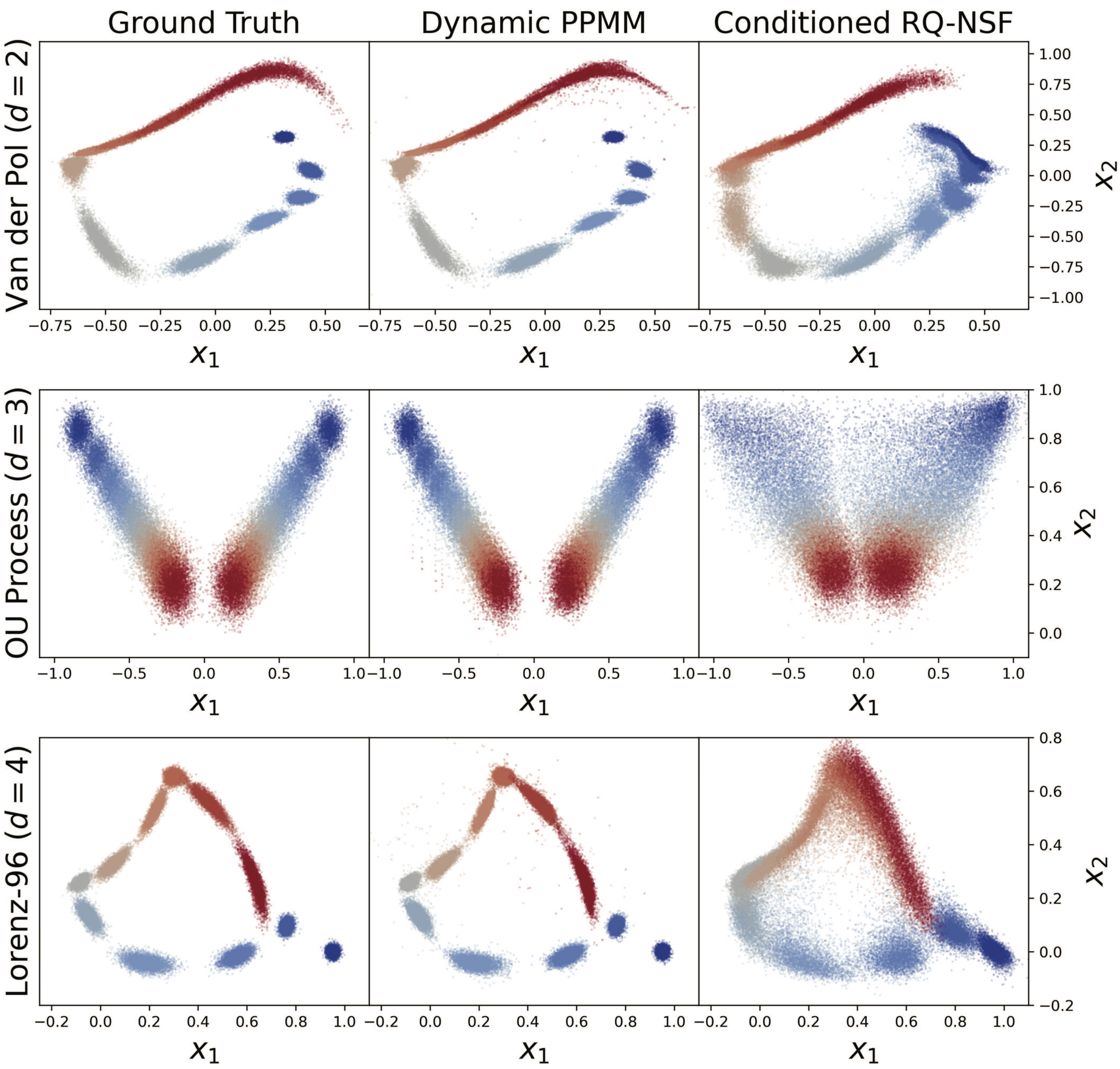

FIG. 2. Low-dimensional qualitative comparison. We show samples from the D-PPMM and time-conditioned Reduced RQ-NSF generative models in which both models are permitted the same amount of wall-clock training time (< 10 s). The coloring shows time-evolution, with blue indicating $t = 0$ and red indicating $t = T$.

variance than the samples of the Reduced RQ-NSF. For the Van der Pol oscillator and Lorenz-96 system, D-PPMM uses $B = 500$ and $L = 0.1$. For the OU-process, D-PPMM uses $B = 2000$ and $L = 0.1$. The RQ-NSFs use a mini-batch size of 100, with the remaining hyperparameter choices detailed in Sec. B.

In Fig. 3, we quantitatively compare the efficiency and accuracy of D-PPMM, the time-conditioned RQ-NSF, and time-conditioned Reduced RQ-NSF frameworks. Since the time to compute the MMD is quadratic in the number of samples $N = 10^4$, the error we use in Fig. 3 is based on the linear-MMD approximation to the true

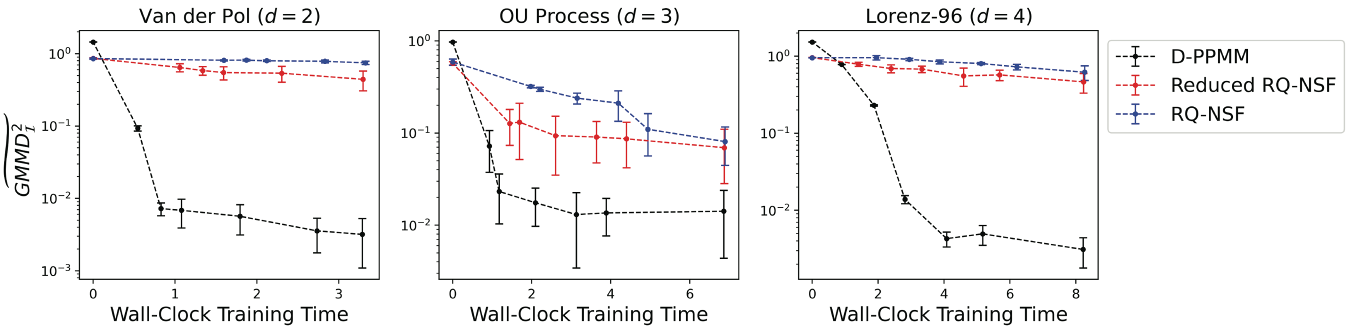

FIG. 3. Low-dimensional convergence comparison. The convergence behavior of D-PPMM and the time-conditioned RQ-NSFs as generative models at the observed measurement times for low-dimensional dynamical systems.

MMD. In Fig. 3, we find that D-PPMM consistently converges faster and with smaller error than both time-conditioned RQ-NSF frameworks. The hyperparameters used to train the generative models are the same as in Fig. 2. To account for the randomness during the training, testing, and error quantification processes, we train each model 10 times with different random seeds and visualize both the mean testing error and the standard deviation over each set of trials for a fixed training time.

C. High-dimensional comparison

We next turn to the high-dimensional setting and consider both the OU-process and Lorenz-96 system with $d = 10$, 20, and 30. For these experiments, we set the final time $T = 3.5$ for the Lorenz-96 system. Conducting a similar convergence analysis as in Fig. 3, we find in Fig. 4 that D-PPMM remains competitive with the time-conditioned RQ-NSF in the high-dimensional setting.

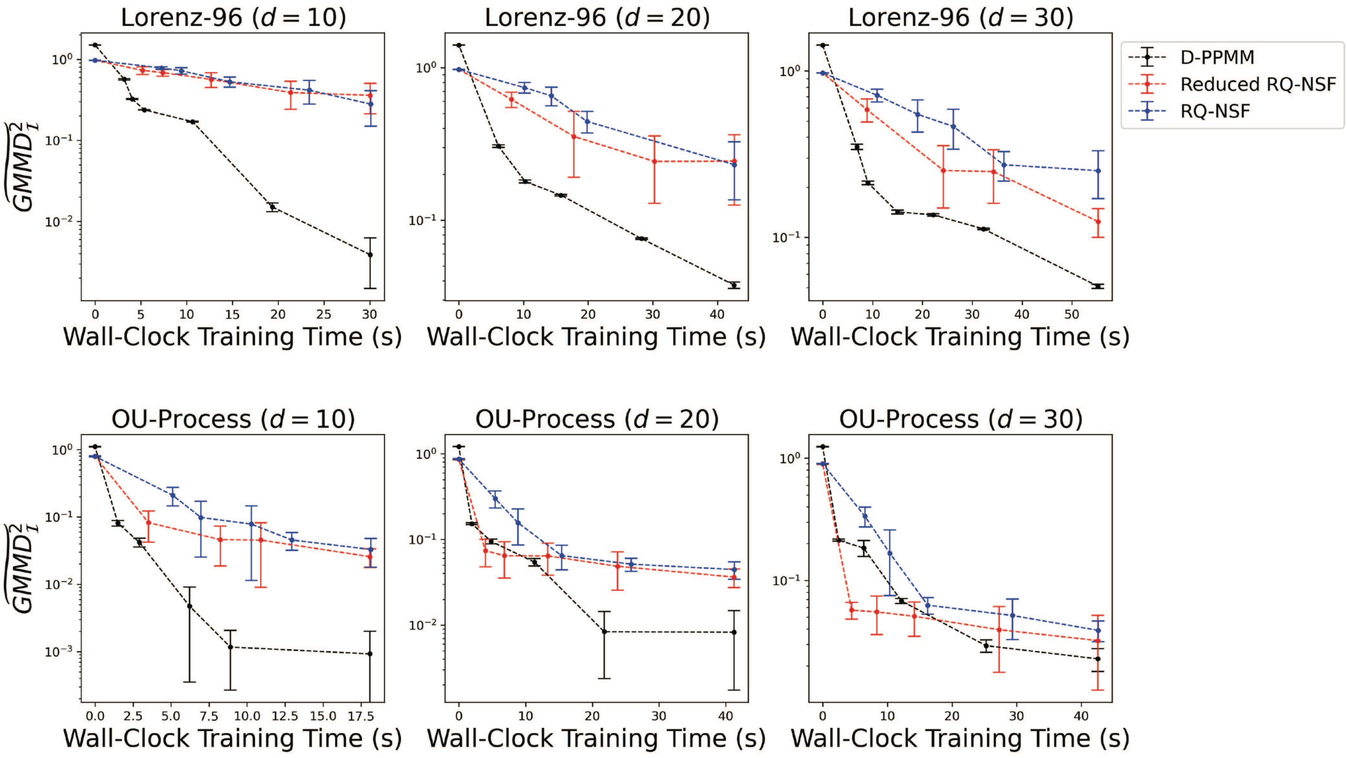

FIG. 4. High dimensional convergence comparison. We show the convergence behavior of D-PPMM and the time-conditioned RQ-NSFs for the Lorenz-96 system and OU-process with $d \in \{10, 20, 30\}$ with $N = 10^4$. Each model is trained 10 times with different random seeds for each training time shown. The error is computed according to the linear-MMD approximation.

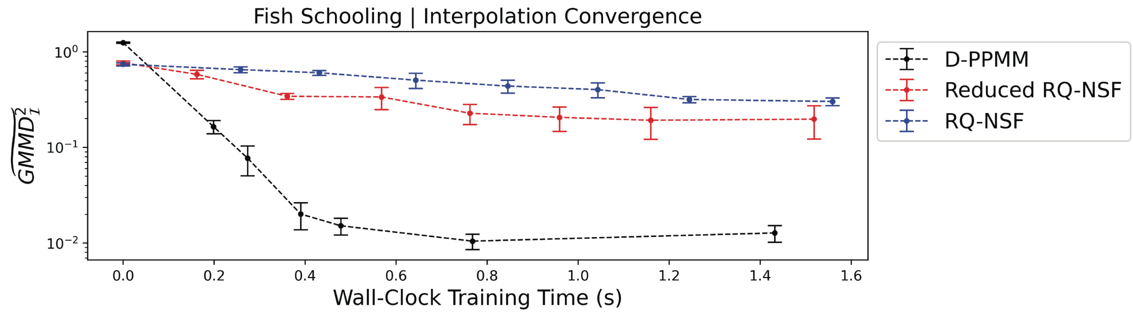

FIG. 5. Fish schooling convergence comparison. Error comparison between D-PPMM and the time-conditioned RQ-NSFs at the held-out snapshots S_2 as the training time is varied.

We observe that in these examples, the performances of the time-conditioned RQ-NSFs start to catch up with those from D-PPMM as the dimension increases.

We remark that the convergence of the D-PPMM algorithm will remain independent of the dimension *d* only if the underlying density changes along a fixed number of directions as *d* increases. Due to the *d*-dimensional Brownian noise used in the experiments in Fig. 4, as well as the fact that the first optimal map we learn is from a *d*-dimensional Gaussian, we do not expect the convergence of D-PPMM to be dimension independent. In situations when each component of the state vector of the underlying ODE/SDE evolves independently, such as the OU-process, it is possible to decompose the pushforward map along *d*-directions, corresponding to the standard Euclidean basis vectors, which can be learned in

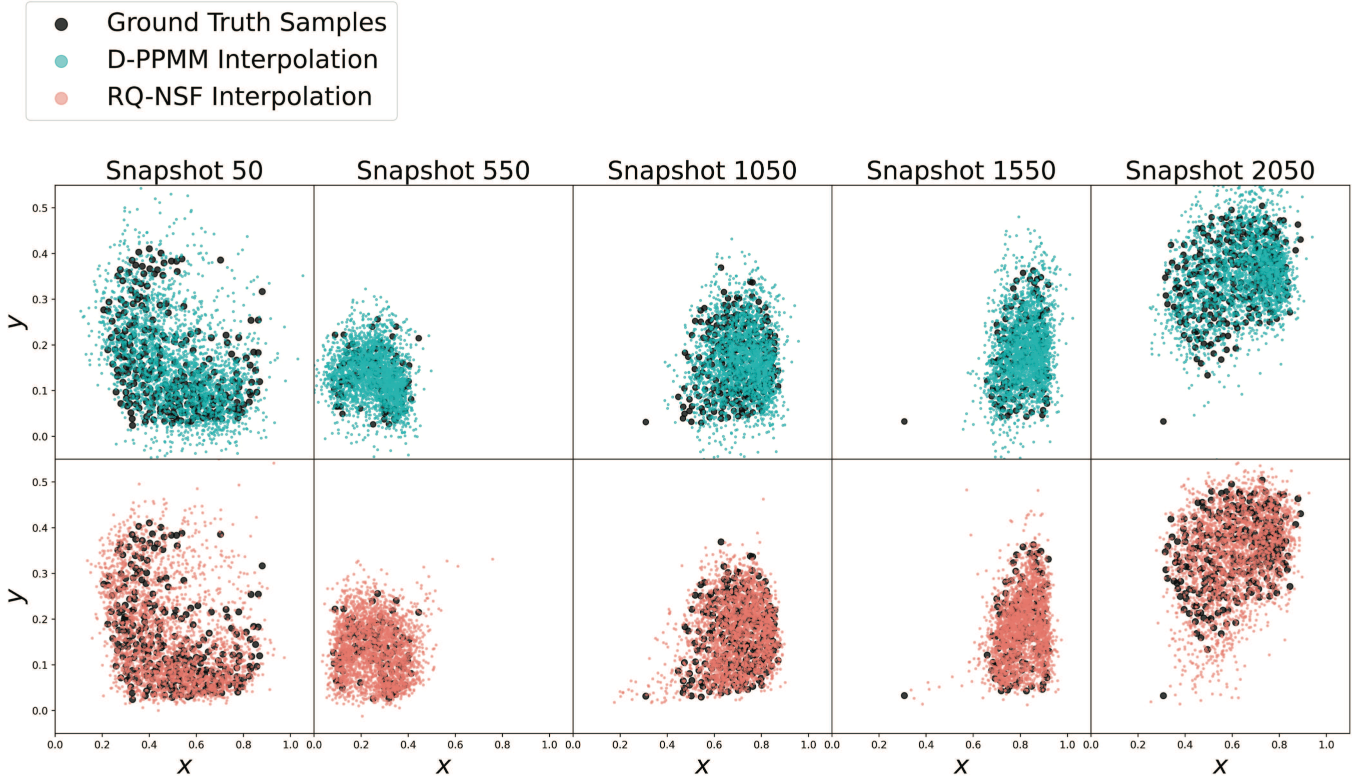

FIG. 6. Fish schooling sample comparison. Samples generated by D-PPMM using cubic transport spline interpolation and the conditioned RQ-NSF at the held-out snapshots S_2. While D-PPMM trained for less than 1 s, the RQ-NSF was permitted 1200 s of training time.

parallel without the use of SAVE. This approach relies on prior knowledge about the functional form of the underlying dynamics, and we do not consider this case here.

D. Fish schooling dataset

Having studied D-PPMM for synthetic stochastic processes in Secs. IV B and IV C, we now consider its application to real-world datasets originating from biological applications. Moreover, instead of comparing D-PPMM and the conditioned RQ-NSF at the observed measurement times, as in Secs. IV B and IV C, we will now examine their interpolation capabilities. Recall that D-PPMM can interpolate densities using the transport splines algorithm (see Sec. III A 2), whereas the conditional normalizing flow can simply be evaluated at the desired interpolation time (see Sec. III B). We begin by studying a dataset of fish schooling behavior.[40] It features trajectories of 300 golden shiners in a rectangular water tank and contains snapshot data, which records the position $(x_i, y_i) \in \mathbb{R}^2$ of the shiners, sampled at 30 Hz frequency. We consider the first 2301 snapshots of the dataset in our comparison. We also note that the recorded sample size varies per snapshot based on which fish are detectable by the tracking software used.[40] Thus, it is particularly advantageous that PPMM can be used to learn OT maps between distributions with different sample sizes (see Sec. II A). Given the sparsity of observations in this particular dataset, we also find it useful to manually select a relatively large bandwidth when performing the KDE for the 1D projected samples (see Sec. II B).

TABLE II. Quantitative comparison of D-PPMM and the conditioned RQ-NSF at the held-out snapshots S_2. The average and standard deviation of the errors are computed over ten different simulations while holding the same training seed fixed. The boldface values are used to highlight the experiments which either used the least amount of training time or achieved the lowest error during testing.

Method	Training time (s)	$\widetilde{\mathrm{GMMD}}^2_{\mathcal{I}}$ $(\times 10^{-3})$
D-PPMM	**0.81**	13.51 ± 4.35
Reduced RQ-NSF	0.81	319.25 ± 5.49
RQ-NSF	1200	$\mathbf{8.76 \pm 0.85}$

To study the effectiveness of interpolation with D-PPMM and the conditioned RQ-NSF, we construct the generative models only from snapshots with indices in $S_1 := \{100j : 0 \leq j \leq 23\}$ and leave out all other data (note that for this example we index our data starting with 0, rather than 1). This effectively reduces the sampling rate from 30 to 0.3 Hz. We then compare the models based on their success at inferring the distribution of golden shiners at the

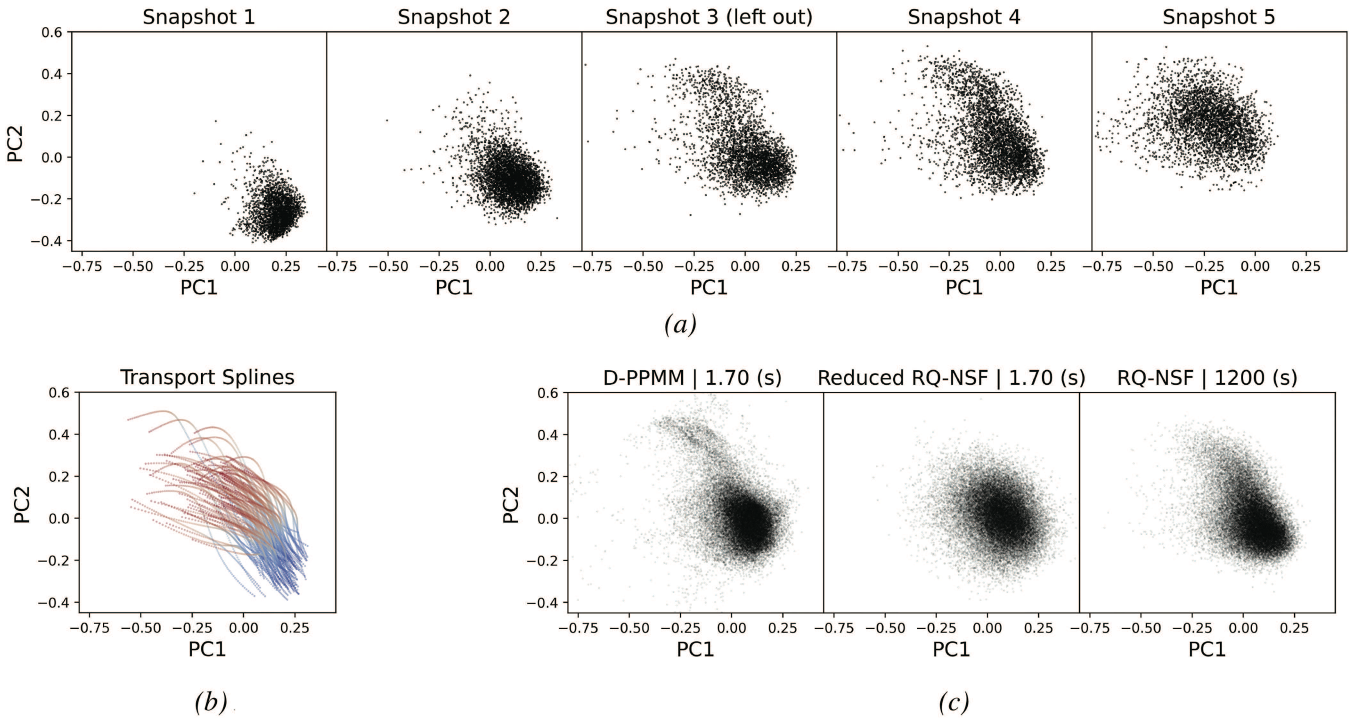

FIG. 7. Embryoid body dataset interpolation comparison. Top row: embryoid body training and testing datasets. Bottom left: Transport splines that D-PPMM uses to interpolate the left-out snapshot. Bottom right: interpolation comparison of the third snapshot using D-PPMM with cubic transport spline interpolation and a time-conditioned RQ-NSF. We show results for 1.70 and 1200 s of wall-clock training time. (a) Inference data; the third snapshot is left out of the training set for testing. (b) D-PPMM transport splines. (c) Interpolation results for D-PPMM and RQ-NSFs with various training times.

snapshots with indices in $S_2 := \{50, 550, 1050, 1550, 2050\}$. In Fig. 5, we study the initial convergence behavior of D-PPMM and the conditioned RQ-NSF when inferring the distribution of shiners at the interpolation snapshots S_2. While D-PPMM converges almost instantly, both normalizing flow frameworks make minimal progress during the same training time. These results are consistent with the rapid convergence of D-PPMM displayed in Fig. 3. For the experiment in Fig. 5, D-PPMM uses a bandwidth of 7.5×10^{-2}, $B = 75$, and $L = 0.25$, and the RQ-NSFs use a mini-batch size of 46. Figure 5 shows the mean and standard deviation of each model's error after training 10 times with different random seeds.

While Fig. 5 shows that D-PPMM with transport spline interpolation exhibits rapid initial convergence compared to the conditioned RQ-NSF models, it is also desirable to compare the accuracy of its interpolation with a conditional flow that has been trained for considerably longer than one second. Toward this, in Fig. 6, we compare the output of the D-PPMM interpolation with a conditioned RQ-NSF which was trained for 1200 s wall-clock time. Note that in Fig. 6, we use the generative models to produce approximately 10× as many samples as were available during training. For this comparison, the hyperparameters of the models are the same as in Fig. 5. Based on the results in Table II, the use of RQ-NSF results in more accurate interpolation for this example but also requires significantly more training time. On the other hand, D-PPMM can still produce reasonable interpolants with several orders of magnitude less training time (about 1/1500 of what the RQ-NSF required).

E. Embryoid body dataset

Next, we study cellular trajectories from an embryoid body dataset.[41] We remark that this embryoid dataset is used as an example in Refs. 26, Figs. 7 and 8. The dataset features five evenly spaced snapshots of a single cell RNA sequencing over a 27-day period with approximately 3×10^4 cells in total. We access a pre-processed version of the dataset from Ref. 26, which filters out rare genes and dead cells. This dataset contains approximately 1.6×10^4 cells and is projected onto its first 5 principal components. The time evolution of these samples is shown in the top row of Fig. 7. Similar to Ref. 26, we leave the third snapshot out from training and attempt to infer it from the remaining four snapshots.

The samples generated by D-PPMM and the conditioned RQ-NSF are shown in the bottom row of Fig. 7. Note that for this example, D-PPMM finished training with a convergence tolerance $\alpha = 10^{-5}$, $B = 500$, and $L = 0.1$ after 1.70 s. For comparison, we show the progress that the Reduced RQ-NSF makes after the same amount of time, as well as the samples of an RQ-NSF, which has been allotted 1200 s of training time. For this experiment, the RQ-NSFs use a mini-batch size of 92 samples. We note that in this case, the time-conditioned RQ-NSF converged with comparable error with D-PPMM with the transport spline interpolation, but that D-PPMM required several orders of magnitude less training time (about 1/700 of what the RQ-NSF required) (see Table III).

TABLE III. Quantitative comparison of D-PPMM and the conditioned RQ-NSF at the held-out snapshot 3. The average and standard deviation of the error are computed over 10 generations with the same training seed. The boldface values are used to highlight the experiments which either used the least amount of training time or achieved the lowest error during testing.

Method	Training time (s)	$\mathrm{GMMD}_{\mathcal{I}}^2$ $(\times 10^{-3})$
D-PPMM	**1.70**	24.97 ± 1.26
Reduced RQ-NSF	1.70	54.99 ± 2.28
RQ-NSF	1200	$\mathbf{24.14 \pm 1.04}$

V. CONCLUSIONS

We have introduced D-PPMM, which constructs a generative model of the sample paths of an evolving density by coupling observed snapshots via OTMs and subsequently applying transport spline interpolation. Though the concept of joining samples from successive snapshots via OTMs is not new, we have shown that the proposed approach is a computationally practical option that can circumvent the large neural network architectures associated with many popular generative models of dynamical systems. Specifically, we have conducted a thorough comparison of the convergence behavior of D-PPMM with state-of-the-art RQ-NSFs conditioned on time. In many of the examples we studied, D-PPMM was more efficient and accurate than the time-conditioned RQ-NSFs. As the dimension d was increased, the difference in performance between the generative models became less noticeable, but D-PPMM still remained competitive. We remark that in our computations in Sec. IV, we did not parallelize the training procedure for D-PPMM, but that doing so may lead to even faster performance. More specifically, the computation of the optimal map between two successive snapshots does not affect the computation of the optimal map between two different successive snapshots. Thus, the training for-loop in Algorithm 2 naturally lends itself to parallelization.

While we have shown that the D-PPMM method is competitive with the conditional normalizing flow in several settings, there are also situations in which the use of the normalizing flow may be preferable. In particular, D-PPMM can suffer from error accumulation when a large number of snapshots are considered, whereas the normalizing flow does not encounter this problem (see Fig. 1). Furthermore, the normalizing flow can be trained on irregularly sampled data, while D-PPMM is expected to struggle when there are relatively few samples at a given observation time. Lastly, the normalizing flow offers additional flexibility for modeling conditional probabilities, as one can use the statistics of previous timesteps as conditioning factors and the sampling time itself to model the density at future timesteps. This feature is expected to be helpful when one wishes to predict the dynamics of a stochastic process rather than interpolate between known measurement times, as in Ref. 12. Future work may focus on extending D-PPMM to remain competitive with the normalizing flow in these settings.

ACKNOWLEDGMENTS

This work was supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research (ASCR), under Contract No. DEAC02–06CH11357, at Argonne National Laboratory. We also acknowledge funding support from ASCR for DOE-FOA-2493, "Data-intensive scientific machine learning.” This research was supported, in part, by an appointment with the National Science Foundation (NSF) Mathematical Sciences Graduate Internship (MSGI) Program sponsored by the NSF Division of Mathematical Sciences. This program is administered by the Oak Ridge Institute for Science and Education (ORISE) through an interagency agreement between the U.S. Department of Energy (DOE) and NSF. ORISE is managed for DOE by ORAU. All opinions expressed in this paper are the author’s and do not necessarily reflect the policies and views of NSF, ORAU/ORISE, or DOE. This paper was supported, in part, by a fellowship award under Contract No. FA9550-21-F-0003 through the National Defense Science and Engineering Graduate (NDSEG) Fellowship Program, sponsored by the Air Force Research Laboratory (AFRL), the Office of Naval Research (ONR), and the Army Research Office (ARO). Y. Yang acknowledges support from Dr. Max Rössler, the Walter Haefner Foundation, and the ETH Zürich Foundation.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Jonah Botvinick-Greenhouse: Formal analysis (lead); Methodology (equal); Writing – original draft (lead); Writing – review & editing (equal). **Yunan Yang:** Methodology (equal); Writing – original draft (supporting); Writing – review & editing (equal). **Romit Maulik:** Conceptualization (lead); Methodology (equal); Writing – original draft (supporting); Writing – review & editing (equal).

DATA AVAILABILITY

The data used in Secs. IV B and IV C are available from the corresponding authors upon reasonable request. The data used in Sec. IV D are openly available in Oregon State University, Ref. 40, and the data used in Sec. IV E are openly available in Mendeley Data, Refs. 41 and 26.

REFERENCES

[1] A. Golightly and D. J. Wilkinson, “Markov chain Monte Carlo algorithms for SDE parameter estimation,” in *Learning and Inference for Computational Systems Biology* (MIT Press, 2010), pp. 253–276.

[2] I. S. Mbalawata, S. Särkkä, and H. Haario, “Parameter estimation in stochastic differential equations with Markov chain Monte Carlo and non-linear Kalman filtering,” Comput. Stat. **28**(3), 1195–1223 (2013).

[3] J. Nygaard Nielsen, H. Madsen, and P. C. Young, “Parameter estimation in stochastic differential equations: An overview,” Annu. Rev. Control **24**, 83–94 (2000).

[4] X. Yang, Y. Liu, and G.-K. Park, “Parameter estimation of uncertain differential equation with application to financial market,” Chaos Soliton. Fract. **139**, 110026 (2020).

[5] J. Jia and A. R. Benson, “Neural jump stochastic differential equations,” in *Advances in Neural Information Processing Systems* (Curran Associates, 2019), Vol. 32.

[6] P. Kidger, J. Foster, X. Li, and T. J. Lyons, “Neural SDEs as infinite-dimensional GANS,” in *International Conference on Machine Learning* (PMLR, 2021), pp. 5453–5463.

[7] X. Liu, T. Xiao, S. Si, Q. Cao, S. Kumar, and C.-J. Hsieh, “Neural SDE: Stabilizing neural ODE networks with stochastic noise,” arXiv:1906.02355 (2019).

[8] T. Salimans, D. Kingma, and M. Welling, “Markov chain Monte-Carlo and variational inference: Bridging the gap,” in *International Conference on Machine Learning* (PMLR, 2015), pp. 1218–1226.

[9] G.-J. Both and R. Kusters, “Temporal normalizing flows,” arXiv:1912.09092 (2019).

[10] X. Feng, L. Zeng, and T. Zhou, “Solving time dependent Fokker-Planck equations via temporal normalizing flow,” arXiv:2112.14012 (2021).

[11] Y. Lu, R. Maulik, T. Gao, F. Dietrich, I. G. Kevrekidis, and J. Duan, “Learning the temporal evolution of multivariate densities via normalizing flows,” Chaos **32**(3), 033121 (2022).

[12] K. Rasul, A.-S. Sheikh, I. Schuster, U. M. Bergmann, and R. Vollgraf, “Multivariate probabilistic time series forecasting via conditioned normalizing flows,” in *International Conference on Learning Representations* (ICLR, 2021).

[13] C. Schöller and A. Knoll, “Flomo: Tractable motion prediction with normalizing flows,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2021), pp. 7977–7984.

[14] I. Kobyzev, S. J. D. Prince, and M. A. Brubaker, “Normalizing flows: An introduction and review of current methods,” IEEE Trans. Pattern Anal. Mach. Intell. **43**(11), 3964–3979 (2020).

[15] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, “Neural ordinary differential equations,” in *Advances in Neural Information Processing Systems*, edited by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Curran Associates, Inc., 2018), Vol. 31.

[16] R. Deng, M. A. Brubaker, G. Mori, and A. Lehrmann, “Continuous latent process flows,” in *Advances in Neural Information Processing Systems*, edited by A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (Curran Associates, 2021).

[17] C. Winkler, D. Worrall, E. Hoogeboom, and M. Welling, “Learning likelihoods with conditional normalizing flows,” arXiv:1912.00042 (2019).

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” Commun. ACM **63**(11), 139–144 (2020).

[19] M. Wiese, R. Knobloch, R. Korn, and P. Kretschmer, “Quant GANS: Deep generation of financial time series,” Quant. Finance **20**(9), 1419–1440 (2020).

[20] L. Yang, D. Zhang, and G. Em Karniadakis, “Physics-informed generative adversarial networks for stochastic differential equations,” SIAM J. Sci. Comput. **42**(1), A292–A317 (2020).

[21] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, “Score-based generative modeling through stochastic differential equations,” arXiv:2011.13456 (2020).

[22] C. Villani, *Optimal Transport: Old and New* (Springer, 2009), Vol. 338.

[23] J. H. Friedman and W. Stuetzle, “Projection pursuit regression,” J. Am. Stat. Assoc. **76**(376), 817–823 (1981).

[24] J.-D. Benamou, T. O. Gallouët, and F.-X. Vialard, “Second-order models for optimal transport and cubic splines on the Wasserstein space,” Found. Comput. Math. **19**, 1113–1143 (2019).

[25] S. Chewi, J. Clancy, T. L. Gouic, P. Rigollet, G. Stepaniants, and A. J. Stromme, “Fast and smooth interpolation on Wasserstein space,” in *International Conference on Artificial Intelligence and Statistics* (PMLR, 2021), pp. 3061–3069.

[26] A. Tong, J. Huang, G. Wolf, D. van Dijk, and S. Krishnaswamy, “Trajectorynet: A dynamic optimal transport network for modeling cellular dynamics,” in *International Conference on Machine Learning* (PMLR, 2020), pp. 9526–9536.

[27] C. Meng, Y. Ke, J. Zhang, M. Zhang, W. Zhong, and P. Ma, “Large-scale optimal transport map estimation using projection pursuit,” *Advances in Neural Information Processing Systems* (Curran Associates, 2019), Vol. 32.

[28] R. Dennis Cook, “SAVE: A method for dimension reduction and graphics in regression,” Commun. Stat. Theory Methods **29**(9–10), 2109–2121 (2000).

[29] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, “Neural spline flows,” *Advances in Neural Information Processing Systems* (Curran Associates, 2019), Vol. 32.

[30] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *International Conference on Machine Learning* (PMLR, 2017), pp. 214–223.

[31] L. Rout, A. Korotin, and E. Burnaev, “Generative modeling with optimal transport maps,” arXiv:2110.02999 (2021).

[32] G. Peyré and M. Cuturi, “Computational optimal transport: With applications to data science,” Found. Trends Mach. Learn. **11**(5–6), 355–607 (2019).

[33] T. Odland (2018). “tommyod/KDEpy: Kernel Density Estimation in Python,” Version v0.9.10. Zenodo. https://doi.org/10.5281/zenodo.2392268

[34] D. W. Scott, “Multivariate density estimation and visualization,” in *Handbook of Computational Statistics: Concepts and Methods* (Springer Berlin, Heidelberg, 2012), pp. 549–569.

[35] Z. I. Botev, J. F. Grotowski, and D. P. Kroese, “Kernel density estimation via diffusion,” Ann. Stat. **38**(5), 2916–2957 (2010).

[36] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios (2020). “nflows: Normalizing flows in PyTorch,” version v0.14. Zenodo. https://doi.org/10.5281/zenodo.4296287

[37] G. A. Pavliotis, *Stochastic Processes and Applications* (Springer New York, New York, 2014).

[38] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, “A kernel two-sample test,” J. Mach. Learn. Res. **13**(25), 723–773 (2012).

[39] K. Fukumizu, A. Gretton, G. Lanckriet, B. Schölkopf, and B. K. Sriperumbudur, “Kernel choice and classifiability for RKHS embeddings of probability distributions,” in *Advances in Neural Information Processing Systems*, edited by Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta (Curran Associates, Inc., 2009), Vol. 22.

[40] Y. Katz, K. Tunstrøm, C. Ioannou, C. Huepe, and I. Couzin (2021). “Fish Schooling Data Subset,” Dataset. Oregon State University. https://doi.org/10.7267/zk51vq07c

[41] K. Moon (2018). “Embryoid Body data for PHATE,” Mendeley Data, V1, https://doi.org/10.17632/v6n743h5ng.1