Data-driven models of nonautonomous systems

Hannah Lu[a,b,1], Daniel M. Tartakovsky[c,*]

[a] *Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*
[b] *Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*
[c] *Department of Energy Science and Engineering, Stanford University, Stanford, CA 94305, USA*

Abstract

Nonautonomous dynamical systems are characterized by time-dependent inputs, which complicates the discovery of predictive models describing the spatiotemporal evolution of the state variables of quantities of interest from their temporal snapshots. When dynamic mode decomposition (DMD) is used to infer a linear model, this difficulty manifests itself in the need to approximate the time-dependent Koopman operators. Our approach is to approximate the original nonautonomous system with a modified system derived via a local parameterization of the time-dependent inputs. The modified system comprises a sequence of local parametric systems, which are subsequently approximated by a parametric surrogate model using the DRIPS (dimension reduction and interpolation in parameter space) framework. The offline step of DRIPS relies on DMD to build a linear surrogate model, endowed with reduced-order bases for the observables mapped from training data. The online step interpolates on suitable manifolds to construct a sequence of iterative parametric surrogate models; the target/test parameter points on these manifolds are specified by a local parameterization of the test time-dependent inputs. We use numerical experimentation to demonstrate the robustness of our method and compare its performance with that of deep neural networks.

Keywords: data-driven learning; nonautonomous systems; manifold interpolations

1. Introduction

Rapid advances in modern machine learning algorithms and the increasing availability of observational data have spurred the innovation in data-driven learning of complex systems. Many regression techniques rely on a library of plausible spatial and/or temporal derivatives of a state variable to build up a (potentially nonlinear) governing equation from observations. Examples include sparse identification of nonlinear dynamical systems [e.g., 1, 2, 3], Gaussian-process regression [4], dynamic mode decomposition (DMD) with library learning [5, 6, 7], and deep neural networks (DNNs) [8, 9, 10]. The performance of such techniques depends on the library completeness and the selection of the degree of sparsification (thresholding).

An alternative strategy is to avoid the need for a library altogether by constructing a surrogate (aka reduced-order) model from (observational and/or synthetic) data. In particular, DMD enables one to construct an optimal linear model for the unknown system [11] and to learn the unknown dynamics of chosen observables, rather than of the system state itself [12]. The latter task is accomplished by utilizing the Koopman operator theory [13] to construct linear models on the observable space, instead of seeking for nonlinear models on the state space [14]. Likewise, DNNs have been deployed as nonlinear surrogates for ODEs [10, 15] and PDEs [16, 17].

*Corresponding author
Email address: tartakovsky@stanford.edu (Daniel M. Tartakovsky)
[1] email: hannahlu@mit.edu

Many, if not most, of such techniques are designed for autonomous systems, i.e., problems whose inputs are constant in time so that their parameter space is finite after a suitable spatial discretization. The input's time-dependence, the defining feature of nonautonomous systems, complicates the learning process because the parameter space becomes infinite and because it is hard to differentiate the data features driven by internal dynamics from their counterparts attributable to external factors (e.g., temporal variability of boundary functions). Attempts to learn differential models of nonautonomous systems from data in the context of control and optimization [18, 19] assume the Koopman operator to have a fixed time-independent structure, relegating time-dependence to forcing/control terms.

More general treatments of nonautonomous dynamical systems include the Koopman operators with time-dependent eigenfuctions, eigenvalues and modes [20], multi-resolution DMD with multiple time scale decomposition [21], spatiotemporal pattern extraction [22], and delay-coordinate maps [23]. These methods apply to nonautonomous systems of a certain kind (e.g., periodic or quasi-periodic) and/or require special analytic tools (e.g., rescaling) and particular knowledge about the system properties. Meanwhile, accurate and efficient computation of the nonautonomous Koopman operator spectrum continues to face significant challenges. For example, online DMD and weighted DMD [24], developed to recover approximations of the time-dependent Koopman eigenvalues and eigenfunctions, require large amount of data to capture the variations in the Koopman operator in each time window; and the deployment of DMD with state observables introduces an intrinsic error [25].

Instead of approximating the time-dependent Koopman operator, our approach is to transform the task of model discovery for a nonautonomous system into the task of learning a locally parameterized system. This transformation was used in [26] to recover unknown nonautonomous dynamical systems via deep neural networks. The local parameterization of the external time-dependent inputs is defined over a set of discrete time instances and conducted using a chosen local basis over time. We use the DRIPS (dimension reduction and interpolation in parameter space) framework [12] to discover the resulting piecewise local parametric systems in each time interval. Once the local surrogate model is constructed from training data, predictions for different initial conditions and/or time-dependent external inputs are made by iterative computation of the interpolated parametric surrogate models over each discrete time instance. The numerical experiments reported below demonstrate that our approach requires significantly less training data than its DNN-based alternative [26]. Moreover, while the DNN's accuracy is insensitive to the time step (frequency) of training-data collection, the accuracy of our method increases as the time step is decreased; this property contributes to the explainability of our method, which DNNs lack.

We start by formulating the problem of interest in section 2. In section 3, we present our methodology, which combines the transformation induced by local parameterization with DRIPS to learn the modified system. In section 4, we test this framework on the numerical examples from [26], which include linear and nonlinear ordinary differential equations (ODEs) and a partial differential equation (PDE). The comparison with DNN demonstrates the efficiency and robustness of our method. Section 5 provides a summary of the main conclusions drawn from this study.

2. Problem Formulation

We consider a multi-physics system described by N_s state variables $\{s_1, \ldots, s_{N_s}\} \equiv \mathbf{s}(\mathbf{x}, t)$, which vary in space, $\mathbf{x} \in \mathcal{D}$, and time, $t \in [0, T]$, throughout the simulation domain $\mathcal{D}$ during the simulation time interval $[0, T]$. The system dynamics is modeled, with high degree of fidelity, by coupled PDEs

$$\frac{\partial s_i}{\partial t} = \phi_i(\mathbf{s}; \gamma(\mathbf{x}, t)), \quad (\mathbf{x}, t) \in \mathcal{D} \times (0, T], \tag{2.1a}$$

which are subject to appropriate boundary conditions and initials conditions

$$s_i(0) = s_i^0, \qquad i = 1, \ldots, N_s. \tag{2.1b}$$

Here, ϕ_i are (linear/nonlinear) differential operators that contain spatial derivatives, and $\gamma(\mathbf{x}, t)$ represents known time-dependent inputs (coefficients of the differential operators and boundary functions). When

solved numerically, the spatial domain $\mathcal{D}$ is discretized into $N_{\rm el}$ nodes, leading to the discretized state variable $\mathbf{S}(t)$ of (high) dimension $N_S = N_s \times N_{\rm el}$, which satisfies nonautonomous ODEs (2.2)

$$\begin{cases} \dfrac{\mathrm{d}\mathbf{S}}{\mathrm{d}t} = \mathbf{\Phi}(\mathbf{S}, \Gamma(t)), \\ \mathbf{S}(0) = \mathbf{S}_0, \end{cases} \tag{2.2}$$

where $\Gamma(t)$ is a vector-valued time-dependent input resulting from the spatial discretization of $\gamma(\mathbf{x}, t)$.

Suppose now that nonautonomous model (2.2) is unknown and one is given instead a set of $N_{\rm obs}$ temporal snapshots of the state variables $\mathbf{S}$ collected at times $t_1, \ldots, t_{N_{\rm obs}}$. Our goal is to construct from these data a numerical surrogate model, which approximates (2.2). More precisely, if the true state variable $\mathbf{S}$ satisfies (2.2), we seek its accurate approximation $\hat{\mathbf{S}}$ for any initial condition $\mathbf{S}_0$ and any input $\Gamma(t)$ within a finite time horizon $T > 0$, i.e.,

$$\hat{\mathbf{S}}(t_k; \mathbf{S}_0, \Gamma(t_k)) \approx \mathbf{S}(t_k; \mathbf{S}_0, \Gamma(t_k)), \qquad k = 1, \ldots, N_T, \quad 0 = t_0 < \cdots < t_{N_T} = T. \tag{2.3}$$

In what follows, we take $\Gamma(t)$ to be a scalar function for illustration purpose. The method can readily handle vector-valued time-dependent inputs, component by component. Likewise, to simplify the presentation and without loss of generality, we consider a uniform time discretization $t_k = k\Delta t \in [0, T]$ with $k = 0, \ldots, N_T$.

3. Methodology

Our DMD-based framework for learning nonautonomous systems (2.2) from data consists of two major steps. The first is to decompose the dynamical system into a modified system comprising a sequence of local systems by parameterizing the external input $\Gamma(t)$ locally in time. A similar parameterization is used in [26] to recover unknown nonautonomous dynamical systems via deep neural networks. The second is to learn the local parametric systems via DRIPS [12].

3.1. Local parameterization and modified system

A discrete-time representation of (2.2) is

$$\begin{aligned} \mathbf{S}(t_{k+1}) &= \mathbf{\Phi}_{\Delta t}(\mathbf{S}(t_k), \Gamma(t_k)) \\ &:= \mathbf{S}(t_k) + \int_{t_k}^{t_k + \Delta t} \mathbf{\Phi}(\mathbf{S}(\tau), \Gamma(\tau)) \mathrm{d}\tau, \\ &= \mathbf{S}(t_k) + \int_0^{\Delta t} \mathbf{\Phi}(\mathbf{S}(t_k + \tau), \Gamma(t_k + \tau)) \mathrm{d}\tau, \end{aligned} \tag{3.1}$$

where $\mathbf{S}(t_k) \in \Omega_{\mathbf{S}} \subset \mathbb{R}^{N_S}$ for $t_k \in [0, T]$. In each time interval $[t_k, t_{k+1}]$ with $k = 0, \ldots, N_T - 1$, we use a *local* parameterization of a given input function $\Gamma(t)$ in the form [26]

$$\tilde{\Gamma}_k(\tau; \mathbf{p}_k) := \sum_{j=1}^{N_{\rm par}} p_k^j b_j(\tau) \approx \Gamma(t_k + \tau), \qquad \tau \in [0, \Delta t], \tag{3.2}$$

where $b_j(\tau)$ with $j = 1, \ldots, N_{\rm par}$ is a set of $N_{\rm par}$ prescribed analytical basis functions, and

$$\mathbf{p}_k = (p_k^1, \cdots, p_k^{N_{\rm par}}) \in \Omega_{\mathbf{p}} \subset \mathbb{R}^{N_{\rm par}} \tag{3.3}$$

are the basis coefficients parameterizing the local input $\Gamma(t)$ in $[t_k, t_{k+1}]$. Examples of local parameterization of a given input $\Gamma(t)$ include interpolating polynomials and Taylor polynomials (Section 3.1 in [26]). Then, a *global* parameterization of $\Gamma(t)$ is constructed as

$$\tilde{\Gamma}(t; \mathbf{p}) = \sum_{k=0}^{N_T - 1} \tilde{\Gamma}_k(t - t_k; \mathbf{p}_k) \mathbb{I}_{[t_k, t_{k+1}]}(t), \qquad \mathbf{p} := \{\mathbf{p}_k\}_{k=0}^{N_T - 1} \in \mathbb{R}^{N_T \times N_{\rm par}}, \tag{3.4a}$$

where $\mathbf{p}$ is a global parameter set for $\tilde{\Gamma}(t)$, and $\mathbb{I}_{[a,b]}$ is the indicator function

$$\mathbb{I}_{[a,b]}(t) = \begin{cases} 1 & \text{if } t \in [a,b], \\ 0 & \text{otherwise.} \end{cases} \tag{3.4b}$$

We also consider a modified representation of the true (unknown) system (2.2),

$$\begin{cases} \dfrac{\mathrm{d}\tilde{\mathbf{S}}}{\mathrm{d}t} = \boldsymbol{\Phi}(\tilde{\mathbf{S}}, \tilde{\Gamma}(t;\mathbf{p})), \\ \tilde{\mathbf{S}}(0) = \mathbf{S}_0, \end{cases} \tag{3.5}$$

where $\tilde{\Gamma}(t;\mathbf{p})$ is the globally parameterized input defined in (3.4). When the system input $\Gamma(t)$ is already known or given in a parametric form, i.e., when $\tilde{\Gamma}(t) = \Gamma(t)$, the modified system (3.5) is equivalent to the original system (2.2). When the parameterized process $\tilde{\Gamma}(t)$ needs to be numerically constructed, the modified system (3.5) becomes an approximation to the true system (2.2). The approximation accuracy depends on the accuracy of $\tilde{\Gamma}(t) \approx \Gamma(t)$.

According to Lemma 3.1 in [26], there exists a function $\tilde{\boldsymbol{\Phi}}_{\Delta t} : \Omega_{\mathbf{S}} \times \Omega_{\mathbf{p}} \to \Omega_{\mathbf{S}}$, which depends on $\boldsymbol{\Phi}$, such that, for any time interval $[t_k, t_{k+1}]$, the solution of (3.5) satisfies

$$\tilde{\mathbf{S}}(t_{k+1}) = \tilde{\boldsymbol{\Phi}}_{\Delta t}(\tilde{\mathbf{S}}(t_k), \mathbf{p}_k), \qquad k = 0, \ldots, N_T - 1, \tag{3.6}$$

where $\mathbf{p}_k$ is the local parameter set in (3.3) for the locally parameterized input $\tilde{\Gamma}_k(t)$ in (3.2).

3.2. Discovery of the modified system via DRIPS

The function $\tilde{\boldsymbol{\Phi}}_{\Delta t}$ in (3.6) governs the evolution of the solution to the modified system (3.5) and is the target function to learn. The challenge posed by nonautonomous systems is now shifted to the task of learning the parametric system (3.6) in any time interval $[t_k, t_{k+1}]$. This task falls into the category of problems where DRIPS framework applies.

3.2.1. Training and testing datasets

Consider a set of N_{sam} samples of the model input, $\{\Gamma^{(1)}(t), \ldots, \Gamma^{(N_{\text{sam}})}(t)\}$. Each sample is evaluated at discrete times $0 = t_0 < t_1 < \ldots t_k < \cdots < t_{N_T} = T$ with $\Delta t = t_{k+1} - t_k$, $k = 0, \ldots, N_T - 1$. For the ith sample, we arrange these inputs and system responses as

$$\{\mathbf{S}^{(i)}(t_k), \Gamma^{(i)}(t_k)\} \equiv \text{input} \quad \Rightarrow \quad \{\mathbf{S}^{(i)}(t_{k+1})\} \equiv \text{output}$$

pairs, and treat them as representative of the true discrete-time dynamical system (3.1) in the time interval $[t_k, t_{k+1}]$, i.e.,

$$\mathbf{S}^{(i)}(t_{k+1}) = \boldsymbol{\Phi}_{\Delta t}(\mathbf{S}^{(i)}(t_k), \Gamma^{(i)}(t_k)), \qquad k = 0, \ldots N_T - 1, \quad i = 1, \ldots, N_{\text{sam}}. \tag{3.7}$$

The local parameterization of $\Gamma^{(i)}(t_k)$ gives $\tilde{\Gamma}_k^{(i)}(\tau; \mathbf{p}_k^{(i)})$, where $\tau \in [0, \Delta t]$ and $\mathbf{p}_k^{(i)}$ is the parameter set for the local parameterization of the input in the form of (3.2). Along the ith sample trajectory and during the kth time interval, a local dataset is

$$\mathcal{S}_{\text{train}}^{(k,i)} = \{\mathbf{S}^{(i)}(t_k), \mathbf{p}_k^{(i)} \ ; \ \mathbf{S}^{(i)}(t_{k+1})\}. \tag{3.8}$$

These input/output pairs satisfy *approximately* the modified system (3.6),

$$\mathbf{S}^{(i)}(t_{k+1}) \approx \tilde{\boldsymbol{\Phi}}_{\Delta t}(\mathbf{S}^{(i)}(t_k), \mathbf{p}_k^{(i)}), \qquad k = 0, \ldots N_T - 1, \quad i = 1, \ldots, N_{\text{sam}}. \tag{3.9}$$

We assemble these data into the training dataset

$$\mathcal{S}_{\text{train}} = \bigcup_{k,i} \mathcal{S}_{\text{train}}^{(k,i)}, \qquad k = 0, \ldots N_T - 1, \qquad i = 1, \ldots, N_{\text{sam}}, \tag{3.10}$$

for the full simulation-time horizon $t \in [0, T]$. As we observe from (3.9), time variable does not play an explicit role in the learning process. So to better learn the map $\tilde{\mathbf{\Phi}}_{\Delta t} : \Omega_{\mathbf{S}} \times \Omega_{\mathbf{p}} \to \Omega_{\mathbf{S}}$ in practice, a preferred dataset can be generated from sampling over the space $\Omega_{\mathbf{S}} \times \Omega_{\mathbf{p}}$ as follows

$$\mathcal{S}_{\text{train}} = \bigcup_{m,j} \mathcal{S}_{\text{train}}^{(m,j)}, \qquad j = 1, \dots, N_{\text{sam}}^{\mathbf{p}}, \qquad m = 0, \dots N_{\text{sam}}^{\mathbf{S}}, \tag{3.11a}$$

where

$$\mathcal{S}_{\text{train}}^{(m,j)} = \{\mathbf{S}^{(m,j)}(0), \mathbf{p}^{(j)} \; ; \; \mathbf{S}^{(m,j)}(\Delta t)\}, \tag{3.11b}$$

and the input/output pairs satisfy approximately

$$\mathbf{S}^{(m,j)}(\Delta t) \approx \tilde{\mathbf{\Phi}}_{\Delta t}(\mathbf{S}^{(m,j)}(0), \mathbf{p}^{(j)}), \qquad j = 1, \dots, N_{\text{sam}}^{\mathbf{p}}, \qquad m = 0, \dots N_{\text{sam}}^{\mathbf{S}}. \tag{3.12}$$

The sampling strategy in $\Omega_{\mathbf{S}}$ and $\Omega_{\mathbf{p}}$ will be specified in section 4 for each problems.

Our goal is to build, from the dataset $\mathcal{S}_{\text{train}}$, a surrogate model of the full (unknown) system (3.6). This surrogate should yield a low-cost prediction of the system-state dynamics, $\{\mathbf{S}(t_0), \dots, \mathbf{S}(t_{N_{T^*}})\}$, for an arbitrary input $\Gamma^*(t)$ not seen during the training, $\Gamma^*(t) \notin \{\Gamma^{(i)}(t)\}_{i=1}^{N_{\text{sam}}}$. Another goal is to use this surrogate to make predictions over the time horizon $[0, T^*]$ that is larger than the one used in training, $[0, T]$.

Using local parameterization (3.2) to approximate $\Gamma^*(t_k)$ with $\tilde{\Gamma}_k^*(\tau; \mathbf{p}_k^*)$, we obtain testing dataset

$$\mathcal{S}_{\text{test}} = \{\tilde{\mathbf{S}}(t_0; \mathbf{p}_0^*), \dots, \tilde{\mathbf{S}}(t_{N_{T^*}}; \mathbf{p}_{N_{T^*}}^*)\}. \tag{3.13}$$

We use DRIPS [12] to estimate $\mathcal{S}_{\text{test}}$, i.e., to learn the target function $\tilde{\mathbf{\Phi}}_{\Delta t}$ and the modified system (3.6). This approach consists of the offline and online steps.

3.2.2. Offline Step: DMD-Based Surrogates

The dynamics of $\mathbf{S}$, i.e., the functional form of $\mathbf{\Phi}$ in (2.2) or $\mathbf{\Phi}_{\Delta t}$ in (3.1) is unknown, as is $\tilde{\mathbf{\Phi}}_{\Delta t}$ in the modified system (3.6). Hence, we replace the unknown discrete system (3.6) with its linear surrogate constructed from the dataset in (3.11). The latter task is facilitated by the Koopman operator theory, which allows one to handle the potential nonlinearity in the unknown dynamics $\mathbf{\Phi}$ and $\tilde{\mathbf{\Phi}}_{\Delta t}$:

Definition 3.1 (Koopman operator [27])**.** *For nonlinear dynamic system* (2.2)*, the Koopman operator* $\mathcal{K}^{\Gamma(t)}$ *is an infinite-dimensional linear operator that acts on all observable functions* $g : \mathcal{M} \to \mathbb{R}$ *so that*

$$\mathcal{K}^{\Gamma(t)} g(\mathbf{S}(t)) = g(\mathbf{\Phi}(\mathbf{S}(t), \Gamma(t))). \tag{3.14}$$

For discrete dynamic system (3.6)*, the discrete-time Koopman operator* $\mathcal{K}_{\Delta t}^{\mathbf{p}_k}$ *is*

$$\mathcal{K}_{\Delta t}^{\mathbf{p}_k} g(\mathbf{S}(t_k; \mathbf{p}_k)) = g(\tilde{\mathbf{\Phi}}_{\Delta t}(\mathbf{S}(t_k; \mathbf{p}_k), \mathbf{p}_k)) = g(\mathbf{S}(t_{k+1}; \mathbf{p}_k)). \tag{3.15}$$

The Koopman operator transforms the finite-dimensional nonlinear problem (3.6) in the state space into the infinite-dimensional linear problem (3.15) in the observable space. Since $\mathcal{K}_{\Delta t}^{\mathbf{p}_k}$ is an infinite-dimensional linear operator, it has infinitely many eigenvalues $\{\lambda_i(\mathbf{p}_k)\}_{i=1}^{\infty}$ and eigenfunctions $\{\phi_i(\mathbf{p}_k)\}_{i=1}^{\infty}$. In practice, one has to make do with a finite number of the eigenvalues and eigenfunctions. The following assumption is essential to both a finite-dimensional approximation and the choice of observables.

Assumption 3.1. *Let* $\mathbf{y}(t_k; \mathbf{p}_k)$ *denote an* $N \times 1$ *vector of observables,*

$$\mathbf{y}(t_k; \mathbf{p}_k) = \mathbf{g}(\mathbf{S}(t_k; \mathbf{p}_k)) = \begin{bmatrix} g_1(\mathbf{S}(t_k; \mathbf{p}_k)) \\ \vdots \\ g_N(\mathbf{S}(t_k; \mathbf{p}_k)) \end{bmatrix}, \tag{3.16}$$

where $g_j : \mathcal{M} \to \mathbb{R}$ *is an observable function with* $j = 1, \dots, N$*. If the chosen observables* $\mathbf{g}$ *are restricted to an invariant subspace spanned by eigenfunctions of the Koopman operator* $\mathcal{K}_{\Delta t}^{\mathbf{p}_k}$*, then they induce a finite-dimensional linear operator* $\mathbf{K}(\mathbf{p}_k)$ *that advances these eigen-observable functions on this subspace [28].*

Assumption 3.1 enables one to deploy the DMD Algorithm 1 to approximate the N-dimensional linear operator $\mathbf{K}(\mathbf{p}_k)$ and its low-dimensional approximation $\mathbf{K}_r(\mathbf{p}_k)$ of rank r. At each parameter point $\mathbf{p}^{(j)}$ with $j = 1, \ldots, N_{\text{sam}}^{\mathbf{P}}$, discrete system (3.12) on the state space is approximated by its N-dimensional linear surrogate

$$\mathbf{y}^{(m)}(\Delta t; \mathbf{p}^{(j)}) = \mathbf{K}(\mathbf{p}^{(j)})\mathbf{y}^{(m)}(0; \mathbf{p}^{(j)}), \quad m = 0, \ldots N_{\text{sam}}^{\mathbf{S}} \tag{3.17}$$

on the observable space. The two spaces are connected by the observable function $\mathbf{g}$ and its inverse $\mathbf{g}^{-1}$. Algorithm 1 induces the ROM for (3.17),

$$\mathbf{y}_r^{(m)}(\Delta t; \mathbf{p}^{(j)}) = \mathbf{K}_r(\mathbf{p}^{(j)})\mathbf{y}_r^{(m)}(0; \mathbf{p}^{(j)}), \tag{3.18}$$

where $\mathbf{y}_r^{(m)}(\mathbf{p}^{(j)})$ is the reduced-order observable vector of dimension r. In a reduced-order basis (ROB) $\mathbf{V}(\mathbf{p}^{(j)})$, these are expressed as

$$\mathbf{y}^{(m)}(\mathbf{p}^{(j)}) = \mathbf{V}(\mathbf{p}^{(j)})\mathbf{y}_r^{(m)}(\mathbf{p}^{(j)}) \quad \text{and} \quad \mathbf{K}_r(\mathbf{p}^{(j)}) = \mathbf{V}(\mathbf{p}^{(j)})^\top \mathbf{K}(\mathbf{p}^{(j)})\mathbf{V}(\mathbf{p}^{(j)}). \tag{3.19}$$

Next, Algorithm 1 is executed on the dataset (3.11).

Algorithm 1: DMD on observable space [27] for parameter point $\mathbf{p}^{(j)}$, $j = 1, \ldots, N_{\text{sam}}^{\mathbf{P}}$.

Input: $\bigcup_m \mathcal{S}_{\text{train}}^{(m,j)}$ in (3.11b), observable function $\mathbf{g}$

1. Create data matrices of the observables

$$\mathbf{Y}_1(\mathbf{p}^{(j)}) = \begin{bmatrix} | & & | \\ \mathbf{g}(\mathbf{S}^{(1)}(0; \mathbf{p}^{(j)})) & \ldots & \mathbf{g}(\mathbf{S}^{(N_{\text{sam}}^{\mathbf{S}})}(0; \mathbf{p}^{(j)})) \\ | & & | \end{bmatrix},$$

$$\mathbf{Y}_2(\mathbf{p}^{(m)}) = \begin{bmatrix} | & & | \\ \mathbf{g}(\mathbf{S}^{(1)}(\Delta t; \mathbf{p}^{(j)})) & \ldots & \mathbf{g}(\mathbf{S}^{(N_{\text{sam}}^{\mathbf{S}})}(\Delta t; \mathbf{p}^{(j)})) \\ | & & | \end{bmatrix}.$$

2. Apply SVD $\mathbf{Y}_1(\mathbf{p}^{(j)}) \approx \mathbf{V}(\mathbf{p}^{(j)})\mathbf{\Sigma}(\mathbf{p}^{(j)})\mathbf{Z}(\mathbf{p}^{(j)})^\top$, where

$$\mathbf{V}(\mathbf{p}^{(j)}) \in \mathbb{R}^{N \times r}, \qquad \mathbf{\Sigma}(\mathbf{p}^{(j)}) \in \mathbb{R}^{r \times r}, \qquad \mathbf{Z}(\mathbf{p}^{(j)}) \in \mathbb{R}^{r \times N_{\text{sam}}^{\mathbf{S}}},$$

   and r is the truncation rank chosen by certain criteria and kept the same for all $j = 1, \ldots, N_{\text{sam}}^{\mathbf{P}}$.

3. Compute

$$\mathbf{K}_r(\mathbf{p}^{(j)}) = \mathbf{V}(\mathbf{p}^{(j)})^\top \mathbf{Y}_2(\mathbf{p}^{(j)})\mathbf{Z}(\mathbf{p}^{(j)})\mathbf{\Sigma}(\mathbf{p}^{(j)})^{-1}$$

   as an $r \times r$ low-rank approximation of $\mathbf{K}(\mathbf{p}^{(j)})$.

4. For $j_1, j_2 = 1, \ldots, N_{\text{MC}}$, compute

$$\mathbf{P}^{(j_1,j_2)} = \mathbf{V}(\mathbf{p}^{(j_1)})^\top \mathbf{V}(\mathbf{p}^{(j_2)}).$$

Output: $\mathbf{V}(\mathbf{p}^{(j)})$, $\mathbf{K}_r(\mathbf{p}^{(j)})$ and $\mathbf{P}^{(j_1,j_2)}$.

Remark 3.1. The offline construction of DMD surrogates allows one to precompute $\mathbf{P}^{(j_1,j_2)}$ for the later online step. This step consumes a majority of the computational time in the entire framework, when the training data (3.11) come from the high-fidelity computation. Yet, the output of this step can be precomputed and stored efficiently; the output storage is $(N \cdot r + r \cdot r + r \cdot r \cdot (N_{\text{sam}}^{\mathbf{S}} + 1)/2) \cdot N_{\text{sam}}^{\mathbf{S}}$.

Remark 3.2. A theorem in [29] relates the DMD theory to the Koopman spectral analysis under specific conditions on the observables and collected data. This theorem indicates that a judicious selection of the observables is critical to the success of the Koopman method. Given the lack of a principled way to

select observables without expert knowledge of a dynamical system, machine learning techniques have been deployed to identify relevant terms in the dynamics from data, which then guides the selection of the observables [30, 31]. In our numerical examples, we rely on knowledge of the underlying physics to select the observables, as was done in [e.g., 5, 32, 33, 34, 35], or use sparse identification of the observables from a small dictionary of possible candidates.

3.2.3. Online Step: Interpolation in reduced space

For an input $\Gamma^*(t) \notin \{\Gamma^{(i)}(t)\}_{i=1}^{N_{\text{sam}}}$, the goal is to compute $\{\mathbf{S}(t_0), \ldots, \mathbf{S}(t_{N_{T^*}})\}$, at a low cost without recourse to (2.2). Using the same local parameterization as in (3.2), we seek to approximate the test dataset (3.13) via the the parametric reduced-order model (PROM)

$$\mathbf{y}_r(t_{k+1}; \mathbf{p}_k^*) = \mathbf{K}_r(\mathbf{p}_k^*)\mathbf{y}_r(t_k; \mathbf{p}_k^*). \tag{3.20}$$

Subsequently, the observables $\mathbf{y}$ and state variable $\mathbf{S}$ are estimated as

$$\mathbf{y}(t_k; \mathbf{p}_k^*) = \mathbf{V}(\mathbf{p}_k^*)\mathbf{y}_r(t_k; \mathbf{p}_k^*), \qquad \mathbf{S}(t_k; \mathbf{p}_k^*) = \mathbf{g}^{-1}(\mathbf{y}(t_k; \mathbf{p}_k^*)). \tag{3.21}$$

Therefore, the online task comprises the computation of three quantities, $\mathbf{V}(\mathbf{p}_k^*)$, $\mathbf{K}_r(\mathbf{p}_k^*)$, and $\mathbf{y}_r(t_k; \mathbf{p}_k^*)$. In general, $\mathbf{p}_k^* \notin \{\mathbf{p}^{(j)}\}_{j=1}^{N_{\text{sam}}^{\mathbf{P}}}$ for $k = 0, \ldots, N_{T^*}$.

3.2.3.1 ROB Interpolation

We rely on interpolation on the Grassman manifold [36] to compute the ROB $\mathbf{V}(\mathbf{p}_k^*)$ from the dataset $\{\mathbf{V}(\mathbf{p}^{(1)}), \ldots, \mathbf{V}(\mathbf{p}^{(N_{\text{sam}}^{\mathbf{P}})})\}$. This interpolation approach is briefly reviewed below.

Definition 3.2. *Grassmann manifold $\mathcal{G}(r, N)$ is a set of all subspaces in $\mathbb{R}^N$ of dimension r.*

Definition 3.3. *Orthogonal Stiefel manifold $\mathcal{ST}(r, N)$ is a set of orthogonal ROB matrices in $\mathbb{R}^{r \times N}$.*

The ROB $\mathbf{V}(\mathbf{p}^{(j)}) \in \mathbb{R}^{N \times r}$, with $j = 1, \ldots, N_{\text{sam}}^{\mathbf{P}}$ and $r \leq N$, is a full-rank column matrix, whose columns provide a basis of subspace $\mathcal{S}_j$ of dimension r in $\mathbb{R}^N$. While an associated ROM is not uniquely defined by the ROB, it is uniquely defined by the subspace $\mathcal{S}_j$. Therefore, the correct entity to interpolate is the subspace $\mathcal{S}_j$, rather than the ROB $\mathbf{V}(\mathbf{p}^{(j)})$. Hence, the goal is to compute $\mathcal{S}_{k,*} = \text{range}(\mathbf{V}(\mathbf{p}_k^*))$ by interpolating between $\{\mathcal{S}_j\}_{j=1}^{N_{\text{sam}}^{\mathbf{P}}}$ with access to the ROB $\mathbf{V}(\mathbf{p}_k^*)$.

The subspaces $\mathcal{S}_1, \ldots, \mathcal{S}_{N_{\text{sam}}^{\mathbf{P}}}$ belong to the Grassmann manifold $\mathcal{G}(r, N)$ [37, 38]. Each r-dimensional subspace $\tilde{\mathcal{S}}$ of $\mathbb{R}^N$ is a point in $\mathcal{G}(r, N)$ and is nonuniquely represented by a matrix $\tilde{\mathbf{V}} \in \mathbb{R}^{N \times r}$, whose columns span the subspace $\tilde{\mathcal{S}}$. The matrix $\tilde{\mathbf{V}}$ is chosen among those whose columns form a set of orthonormal vectors in $\mathbb{R}^N$ and belong to the orthogonal Stiefel manifold $\mathcal{ST}(r, N)$ [37, 39]. There exists a projection map [37] from $\mathcal{ST}(r, N)$ onto $\mathcal{G}(r, N)$, as each matrix in $\mathcal{ST}(r, N)$ spans an r-dimensional subspace of $\mathbb{R}^N$ and different matrices can span the same subspace. At each point $\tilde{\mathcal{S}}$ of the manifold $\mathcal{G}(r, N)$, there exists a tangent space $\mathcal{T}_{\tilde{\mathcal{S}}}$ [37, 39] of the same dimension [39]. Each point in this space is represented by a matrix $\tilde{\mathbf{M}} \in \mathbb{R}^{N \times r}$. Since $\mathcal{T}_{\tilde{\mathcal{S}}}$ is a vector space, usual interpolation is allowed for the matrices representing its points. Let $\mathbf{M}^j = m_{\mathbf{V}}(\mathbf{V}(\mathbf{p}^{(j)}))$, where $m_{\mathbf{V}}$ denotes the map from the matrix manifolds $\mathcal{G}(r, N)$ onto the tangent space $\mathcal{T}_{\tilde{\mathcal{S}}}$. This suggests a strategy for computing $\mathbf{M}^{k,*}$ via usual interpolation between $\{\mathbf{M}^j\}_{j=1}^{N_{\text{sam}}^{\mathbf{P}}}$ and then evaluating $\mathbf{V}(\mathbf{p}_k^*)$ through the inverse map $m_{\mathbf{V}}^{-1}(\mathbf{M}^{k,*})$.

The map $m_{\mathbf{V}}$ is chosen to be logarithmic, which maps the Grassmann manifold onto its tangent space, and $m_{\mathbf{V}}^{-1}$ is chosen to be an exponential mapping, which maps the tangent space onto the Grassmann manifold itself. This choice borrows concepts of geodesic path on a Grassmann manifold from differential geometry [37, 38]. This strategy, discussed in detail in [36], is implemented in Algorithm 2.

Remark 3.3. A choice of the interpolation method $\mathcal{P}$ depends on the dimension of the parameter space, N_{par}. If $N_{\text{par}} = 1$, a univariate (typically, a Lagrange type) interpolation method is chosen. Otherwise, a multivariate interpolation scheme [e.g., 40, 41] is chosen.

Remark 3.4. Since the logarithmic map $\mathcal{L}^{S_{j_0}}$ is defined in a neighborhood of $\mathcal{S}_{j_0} \in \mathcal{G}(r, N)$, the method is expected to be insensitive to a choice of the reference point $\mathcal{S}_{j_0}$ in step 1 of Algorithm 2. This is confirmed in numerical experiments [36].

Algorithm 2: Interpolation of ROBs [36]

Input: $\{\mathbf{V}(\mathbf{p}^{(j)})\}_{j=1}^{N_{\text{sam}}^{\mathbf{P}}}$, $\{\mathbf{P}^{(j_1,j_2)}\}_{j_1,j_2=1}^{N_{\text{sam}}^{\mathbf{P}}}$, $\{\mathbf{p}^{(j)}\}_{j=1}^{N_{\text{sam}}^{\mathbf{P}}}$ and target parameter point $\{\mathbf{p}_k^*\}_{k=0}^{N_{T^*}}$

1. Denote $\mathcal{S}_j = \text{range}(\mathbf{V}(\mathbf{p}^{(j)}))$, for $j = 1, \dots N_{\text{sam}}^{\mathbf{P}}$. A point $\mathcal{S}_{j_0}$ with $j_0 \in \{1, \dots N_{\text{sam}}^{\mathbf{P}}\}$ of the manifold is chosen as a reference and origin point for interpolation.
2. Select points $\mathcal{S}_j$ with $j \in \mathcal{I}_{j_0} \subset \{1, \dots, N_{\text{sam}}^{\mathbf{P}}\}$, which lie in a sufficiently small neighborhood of $\mathcal{S}_{j_0}$, and use the logarithm map $\mathcal{L}^{\mathcal{S}_{j_0}}$ to map $\{\mathcal{S}_j\}_{j\in\mathcal{I}_{j_0}}$ onto matrices $\{\mathbf{M}^j\}_{j\in\mathcal{I}_{j_0}}$ representing the corresponding points of $\mathcal{T}_{\mathcal{S}_{j_0}}$. This is computed as
$$\begin{aligned} &(\mathbf{I} - \mathbf{V}(\mathbf{p}^{(j_0)})\mathbf{V}(\mathbf{p}^{(j_0)})^\top)\mathbf{V}(\mathbf{p}^{(j)})(\mathbf{P}^{(j_0,j)})^{-1} = \mathbf{U}_j\mathbf{\Omega}_j\mathbf{W}_j^\top, \quad \text{(thin SVD)} \\ &\mathbf{M}^j = \mathbf{U}_j \tan^{-1}(\mathbf{\Omega}_j)\mathbf{W}_j^\top. \end{aligned} \tag{3.22}$$
3. Compute $\mathbf{M}^{k,*}$ by interpolating $\{\mathbf{M}^j\}_{j\in\mathcal{I}_{j_0}}$ entry by entry,
$$M_{il}^{k,*} = \mathcal{P}(\mathbf{p}_k^*; \{M_{il}^j, \mathbf{p}^{(j)}\}_{j\in\mathcal{I}_{j_0}}), \quad i = 1, \dots, N, \quad l = 1, \dots, r. \tag{3.23}$$
4. Use the exponential map $\mathcal{E}^{\mathcal{S}_{j_0}}$ to map the matrix $\mathbf{M}^{k,*}$, representing a point of $\mathcal{T}_{\mathcal{S}_{j_0}}$, onto the desired subspace $\mathcal{S}_{k,*}$ on $\mathcal{G}(r, N)$ spanned by the ROB $\mathbf{V}(\mathbf{p}_k^*)$. This is computed as
$$\begin{aligned} &\mathbf{M}^{k,*} = \mathbf{U}_{k,*} \tan^{-1}(\mathbf{\Omega}_{k,*})\mathbf{W}_{k,*}^\top, \quad \text{(thin SVD)} \\ &\mathbf{V}(\mathbf{p}_k^*) = \mathbf{V}(\mathbf{p}^{(m_0)})\mathbf{W}_{k,*}\cos(\mathbf{\Omega}_{k,*}) + \mathbf{U}_{k,*}\sin(\mathbf{\Omega}_{k,*}). \end{aligned} \tag{3.24}$$

Output: $\{\mathbf{V}(\mathbf{p}_k^*)\}_{k=1}^{N_{T^*}}$

3.2.3.2 PROM Interpolation

The reduced-order operator $\mathbf{K}_r(\mathbf{p}_k^*)$ in (3.20) is computed via interpolation on the matrix manifold between the ROMs $\{\mathbf{K}_r(\mathbf{p}^{(1)}), \dots, \mathbf{K}_r(\mathbf{p}^{(N_{\text{sam}}^{\mathbf{P}})})\}$. This is done in two steps [42]:

Algorithm 3: Step A of the PROM interpolation [36]

Input: $\{\mathbf{K}_r(\mathbf{p}^{(1)}), \dots, \mathbf{K}_r(\mathbf{p}^{(N_{\text{sam}}^{\mathbf{P}})})\}$, $\{\mathbf{P}^{(j_1,j_2)}\}_{j_1,j_2=1}^{N_{\text{sam}}^{\mathbf{P}}}$, reference configuration choice j_0
For $j \in \{1, \dots N_{\text{sam}}^{\mathbf{P}}\} \setminus \{j_0\}$

- Compute $\mathbf{P}^{(j,j_0)} = \mathbf{U}_{j,j_0}\mathbf{\Sigma}_{j,j_0}\mathbf{Z}_{j,j_0}^\top$ (SVD),
- Compute $\mathbf{S}_j = \mathbf{U}_{j,j_0}\mathbf{Z}_{j,j_0}^\top$,
- Transform $\tilde{\mathbf{K}}_r(\mathbf{p}^{(j)}) = \mathbf{S}_j^\top\mathbf{K}_r(\mathbf{p}^{(j)})\mathbf{S}_j$

End
Output: $\{\tilde{\mathbf{K}}_r(\mathbf{p}^{(1)}), \dots, \tilde{\mathbf{K}}_r(\mathbf{p}^{(N_{\text{sam}}^{\mathbf{P}})})\}$

- Step A). Since any ROM can be endowed with alternative ROBs, the resulting ROMs may have been computed in different generalized coordinates systems. The validity of an interpolation may depend crucially on a choice of the representative element within each equivalent class. Given the precomputed ROMs $\{\mathbf{K}_r(\mathbf{p}^{(1)}), \dots, \mathbf{K}_r(\mathbf{p}^{(N_{\text{sam}}^{\mathbf{P}})})\}$, a set of congruence transformations is determined so that a representative element of the equivalent ROBs for each precomputed ROM is chosen to assign the precomputed ROMs to consistent sets of generalized coordinates. The consistency is

enforced by solving the orthogonal Procrustes problems [43],

$$\min_{\mathbf{S}_j, \mathbf{S}_j^\top \mathbf{S}_j = \mathbf{I}_r} \|\mathbf{V}(\mathbf{p}^{(j)})^\top \mathbf{S}_j - \mathbf{V}(\mathbf{p}^{(j_0)})\|_F, \qquad \forall j = 1, \ldots, N_{\text{sam}}^{\mathbf{p}}, \tag{3.25}$$

where $j_0 \in \{1, \ldots N_{\text{sam}}^{\mathbf{p}}\}$ is chosen as a reference configuration. The representative element is identified by solving the above problem analytically. This procedure is summarized in Algorithm 3.

Remark 3.5. An optimal choice of the reference configuration, j_0, if it exists, remains an open problem.

Algorithm 4: Step B of the PROM interpolation [36]

Input: $\{\tilde{\mathbf{K}}_r(\mathbf{p}^{(1)}), \ldots, \tilde{\mathbf{K}}_r(\mathbf{p}^{(N_{\text{sam}}^{\mathbf{p}})})\}$, reference configuration choice j_0

1. **For** $m \in \{1, \ldots N_{\text{sam}}^{\mathbf{p}}\} \setminus \{j_0\}$
   - Compute $\mathbf{M}^j = \mathcal{L}^{\tilde{\mathbf{K}}_r(\mathbf{p}^{(j_0)})}(\tilde{\mathbf{K}}_r(\mathbf{p}^{(j)}))$

   End
2. Compute $\mathbf{M}^{k,*}$ by interpolating $\{\mathbf{M}^j\}_{j \in \mathcal{I}_{j_0}}$ entry by entry, as in (3.23)
3. Compute $\mathbf{K}_r(\mathbf{p}_k^*) = \mathcal{E}^{\tilde{\mathbf{K}}_r(\mathbf{p}^{(j_0)})}(\mathbf{M}^{k,*})$

Output: $\{\mathbf{K}_r(\mathbf{p}_k^*)\}_{k=1}^{N_{T^*}}$

- Step B). The transformed ROMs $\{\tilde{\mathbf{K}}_r(\mathbf{p}^{(1)}), \ldots, \tilde{\mathbf{K}}_r(\mathbf{p}^{(N_{\text{sam}}^{\mathbf{p}})})\}$ are interpolated to compute ROMs $\{\mathbf{K}_r(\mathbf{p}_k^*)\}_{k=1}^{N_{T^*}}$. Similar to the ROB interpolation above, this interpolation must be performed on a specific manifold containing both $\{\tilde{\mathbf{K}}_r(\mathbf{p}^{(1)}), \ldots, \tilde{\mathbf{K}}_r(\mathbf{p}^{(N_{\text{sam}}^{\mathbf{p}})})\}$ and $\{\mathbf{K}_r(\mathbf{p}_k^*)\}_{k=1}^{N_{T^*}}$, so that the distinctive properties (e.g., orthogonality, nonsingularity) are preserved. The main idea again is to first map all the precomputed matrices onto the tangent space to the matrix manifold of interest at a chosen reference point using the logarithm mapping, then interpolate the mapped data in this linear vector space, and finally map the interpolated result back onto the manifold of interest using the associated exponential map. This is done in Algorithm 4.

  Remark 3.6. The log map $\mathcal{L}$ and exp map $\mathcal{E}$ in Algorithm 4 denote the matrix logarithm and exponential respectively. The specific expressions of different matrix manifolds of interest are listed in Table 4.1 of [42].

3.2.3.3 Computation of the Solution

With $\{\mathbf{K}_r(\mathbf{p}_k^*)\}_{k=0}^{N_{T^*}}$ and $\{\mathbf{V}(\mathbf{p}_k^*)\}_{k=0}^{N_{T^*}}$ computed, we use (3.20) and (3.21) to obtain the solution.

3.2.4. Algorithm Summary

The proposed framework is summarized in Algorithm 5.

Remark 3.7. The sampling strategy for $\{\mathbf{p}^{(1)}, \ldots, \mathbf{p}^{(N_{\text{sam}}^{\mathbf{p}})}\}$ in the parameter space plays a key role in the accuracy of the subspace approximation. The so-called "curse of dimensionality"—a phenomenon in which the number of required training samples N_{MC} grows exponentially with the number of parameters, N_{par}—is a well-known challenge. In general, uniform sampling is used for $N_{\text{par}} \leq 5$ and moderately computationally intensive HFMs; latin hypercube sampling is used for $N_{\text{par}} > 5$ and moderately computationally intensive HFMs; and adaptive, goal-oriented, greedy sampling is used for $N_{\text{par}} > 5$ and highly computationally intensive HFMs. We limit our numerical experiments to $N_{\text{par}} = 3$ for simplicity, leaving the challenge posed by the curse of dimensionality for future studies.

Algorithm 5: Learning nonautonomous system via DMD.

Offline Step:

For $m = 1, \ldots, N_{\text{sam}}^{\mathbf{P}}$,

Get the training data (3.11),

Input: $\bigcup_m \mathcal{S}_{\text{train}}^{(m,j)}$ and $\mathbf{g} \xrightarrow{\text{Algorithm 1}}$ Output: $\mathbf{V}(\mathbf{p}^{(j)}), \mathbf{K}_r(\mathbf{p}^{(j)})$ and $\mathbf{P}^{(j_1,j_2)}$

End

Online Step:

- Interpolation of ROBs:

$$\text{Input: } \{\mathbf{V}(\mathbf{p}^{(j)})\}_{m=1}^{N_{\text{sam}}^{\mathbf{P}}},\ \{\mathbf{P}^{(j_1,j_2)}\}_{j_1,j_2=1}^{N_{\text{sam}}^{\mathbf{P}}},\ \{\mathbf{p}^{(j)}\}_{j=1}^{N_{\text{sam}}^{\mathbf{P}}},\ \{\mathbf{p}_k^*\}_{k=0}^{N_{T^*}} \xrightarrow{\text{Algorithm 2}} \text{Output: } \{\mathbf{V}(\mathbf{p}_k^*)\}_{k=1}^{N_{T^*}}$$

- Interpolation of PROMs:

$$\text{Input: } \{\mathbf{K}_r(\mathbf{p}^{(j)})\}_{j=1}^{N_{\text{sam}}^{\mathbf{P}}},\ \{\mathbf{P}^{(j_1,j_2)}\}_{j_1,j_2=1}^{N_{\text{sam}}^{\mathbf{P}}},\ \text{reference choice } j_0 \xrightarrow{\text{Algorithms 3 \& 4}} \text{Output: } \mathbf{K}_r(\mathbf{p}_k^*)$$

- DMD reconstruction:

$$\text{Input: } \mathbf{K}_r(\mathbf{p}_k^*),\ \mathbf{V}(\mathbf{p}_k^*),\ \mathbf{y}(t=0;\mathbf{p}_0^*),\ \mathbf{g} \xrightarrow{(3.20),(3.21)} \text{Output: } \mathbf{S}_{\text{DMD}}(t_k;\mathbf{p}_k^*)$$

4. Numerical Experiments

We test our framework on every numerical example from [26]. Synthetic data generated from known dynamical systems with known time-dependent inputs are used to validate our methodology. The training data are generated by solving the known system with a high-resolution numerical scheme in Matlab, e.g., `ode45`, which is based on an explicit fourth-order Runge-Kutta formula. To facilitate comparison with [26], we deploy their local parameterization via interpolating polynomials over equally spaced points. Although not shown here, we found other parameterization strategies to produce similar results.

For convenience, we generate the training dataset in the form of (3.11). Rather than uniformly sampling the parameters (3.3), as was done in [26], we take the parameter values on the Cartesian grid of the N_{par}-dimensional parameter space with 3 points (two endpoints and one middle point) in each dimension, so that $N_{\text{sam}}^{\mathbf{P}} = 3^{N_{\text{par}}}$. For each training subset $\mathcal{S}_{\text{train}}^{(m,j)}$, the first data entry (the initial state $S^{(m)}(0)$) is randomly sampled from a domain $\Omega_{\mathbf{S}}$ using uniform distribution. The second data entry is produced by solving the underlying reference dynamical system with time step Δt and subject to a parameterized input in the form of (3.2). Unless specified otherwise, all numerical examples use $\Delta t = 0.1$. Therefore, the total amount of training data is $N_{\text{sam}}^{\mathbf{S}} \times 3^{N_{\text{par}}}$ pairs in the form of (3.11b). This number, specified in each numerical example below, is much smaller than what is needed to train a neural network, $\mathcal{O}(10^5)$, in [26].

The sampling domain $\Omega_{\mathbf{S}}$ and parameter domain $\Omega_{\mathbf{p}}$ are determined from prior knowledge of the underlying unknown equations to ensure that the range of the target trajectories is covered by $\Omega_{\mathbf{S}}$ for parameters in $\Omega_{\mathbf{p}}$ so that the assumptions in the theoretical analysis of [26] are satisfied. The same $\Omega_{\mathbf{S}}$ and $\Omega_{\mathbf{p}}$ are adopted from [26] and are specified separately for each example.

Our learning algorithm 5 is then applied to the training dataset. The observable $\mathbf{g}$ is determined by the underlying system, as specified in each numerical example. A choice of the observables may not be unique and optimal. The task of optimal construction of the observables is beyond the scope of this paper; instead we employ several standard tricks in DMD studies to construct reasonable observables. Once the learned model is constructed, we predict the system state iteratively, solving the model with new

initial conditions and new inputs. The prediction results are then compared with the reference solution obtained by solving the exact system with the same new inputs.

4.1. Linear ODE with variable coefficients

Consider a linear ODE

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\alpha(t)S + \beta(t), \quad t > 0; \qquad S(0) = S_0. \tag{4.1}$$

The time-dependent inputs $\alpha(t)$ and $\beta(t)$ are locally parameterized with second-degree polynomials. As a result, the local parameter set (3.3) $\mathbf{p}_k \in \mathbb{R}^{N_{\mathrm{par}}}$ with $N_{\mathrm{par}} = 3+3 = 6$. The training data are generated from $N^{\mathbf{P}}_{\mathrm{sam}} = 3^6$ parameter points on the Cartesian grid of the parameter space $I_{\mathbf{p}} = [-5, 5]^6$. For each parameter point $\mathbf{p}^{(j)}$, $N^{\mathbf{S}}_{\mathrm{sam}} = 2$ pairs of data in the form of (3.11b) are generated with the initial state $S^{(m)}(0)$ randomly drawn from state variable space $\Omega_S = [-2, 2]$. The total number of data pairs is 1458.

Since S is a scalar, we chose the observable $\mathbf{g}$ to form an augmented data matrix $\mathbf{y}(t_k; \mathbf{p}_k) = \mathbf{g}(S(t_k; \mathbf{p}_k)) = [S(t_k; \mathbf{p}_k), S(t_k; \mathbf{p}_k)^2]^\top$. (Alternatively, one can construct the shift-stacked data matrices [29].) Once the DRIPS surrogate is trained, we use it to predict the system behavior for the model parameters $\alpha(t) = 1 + \sin(4t)$ and $\beta(t) = \cos(t^2/1000)$ and the initial condition $S_0 = 2$ not used in the training (Fig. 1). The surrogate's prediction of the system state $S(t)$ over the time horizon $T = 100$ is visually indistinguishable from the numerical solution of (4.1), having the modified relative L_2 error $\mathcal{E}(t) = \|S_{\mathrm{DRIPS}}(t) - S_{\mathrm{num}}(t)\|_2/(\|S_{\mathrm{num}}(t)\|_2 + \varepsilon)$ on the order of $10^{-2} - 10^{-4}$. A small number $\varepsilon = 10^{-2}$ is used to prevent blow-up near $\|S_{\mathrm{num}}(t)\|_2 = 0$.

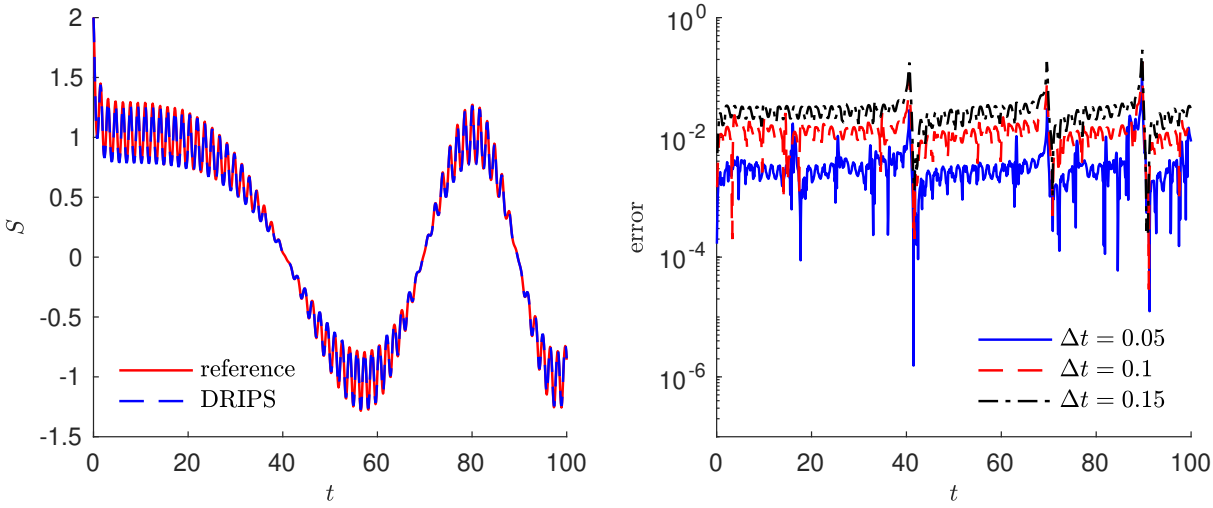

Figure 1: (a) DRIPS prediction and numerical solution of (4.1) with parameters $\alpha(t) = \sin(4t)+1$ and $\beta(t) = \cos(t^2/1000)$, for the initial condition $S_0 = 2$. (b) Prediction error of the DRIPS surrogate, $\mathcal{E}(t) = \|S_{\mathrm{DRIPS}}(t) - S_{\mathrm{num}}(t)\|_2/(\|S_{\mathrm{num}}(t)\|_2 + \varepsilon)$ with $\varepsilon = 10^{-2}$, for different sampling frequency, $1/\Delta t$, used to collect training data.

The prediction error of DRIPS, $e(t)$, decreases with the time frequency, $1/\Delta t$, at which the training data are collected (Fig. 1b). This is in contrast with the DNN-based prediction, whose error of $\mathcal{O}(10^{-2})$ is relatively insensitive to the choice of Δt (Figure 4.1 in [26]). The prediction error of both surrogates has two sources. The first is the approximation error of the parameterization, i.e., the difference between the true non-autonomous system (2.2) and the modified system (3.5). The second is the error of the surrogate, i.e., the difference between the modified system (3.5) and the surrogate model, (3.21) in the case of DRIPS. These sources happen to be well balanced in DNN, so the overall error stays the same order for all choices of Δt. In DRIPS, the second source is more dominant and, thus, the overall error is sensitive to the choice of Δt. That is because the accuracy of DMD surrogates is known to be highly dependent on the sampling frequency $1/\Delta t$.

4.2. Predator-prey model with control

Consider state variables $S_1(t)$ and $S_2(t)$, whose dynamics satisfies a Lotka-Volterra predator-prey model with a time-dependent input $u(t)$,

$$\begin{aligned} \frac{\mathrm{d}S_1}{\mathrm{d}t} &= S_1 - S_1 S_2 + u(t), \\ \frac{\mathrm{d}S_2}{\mathrm{d}t} &= -S_2 + S_1 S_2. \end{aligned} \tag{4.2}$$

The local parameterization of $u(t)$ is conducted using quadratic polynomials, i.e., $\mathbf{p} \in \mathbb{R}^3$. We set $\Omega_{\mathbf{p}} = [0,5]^3$ and the state variable space $\Omega_{\mathbf{S}} = [0,5]^2$. Therefore, $N_{\mathrm{sam}}^{\mathbf{p}} = 27$ parameter points on the Cartesian grid of $\Omega_{\mathbf{p}}$ are selected to generate the training data with $N_{\mathrm{sam}}^{\mathbf{S}} = 9$ pairs. The total number of training data pairs is 243. The observable is constructed from $S_1(t_k;\mathbf{p}_k)$ and $S_2(t_k;\mathbf{p}_k)$ as

$$\mathbf{y}(t_k;\mathbf{p}_k) = \mathbf{g}(\mathbf{S}(t_k;\mathbf{p}_k)) = [S_1, S_2, S_1^2, S_1S_2, S_2^2, S_1^3, S_1^2S_2, S_1S_2^2, S_2^3]^\top.$$

The trained DRIPS surrogate is tested on the external control $u(t) = 2 + \sin(t/3) + \cos(t)$, for the initial condition $\mathbf{S}_0 = [3,2]$ for time up to $T = 100$. The DRIPS prediction of $S_1(t)$ is accurate; its L_2 error, relative to the numerical solution of (4.2), is on the order of 10^{-3} (Fig. 2). Although not shown here, similar results hold for $S_2(t)$.

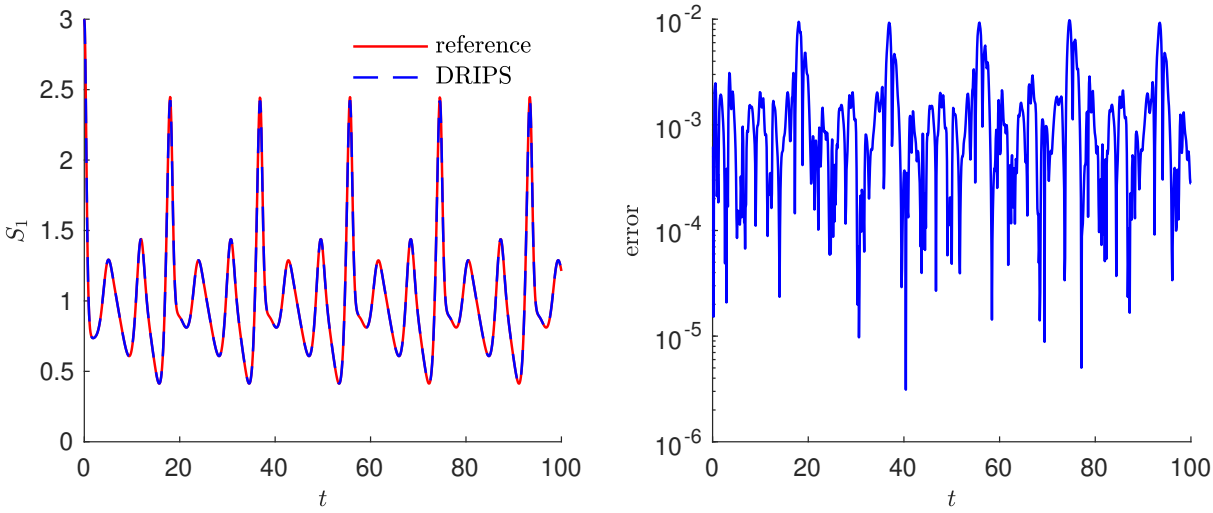

Figure 2: (a) DRIPS prediction and numerical solution of (4.2) with control $u(t) = 2+\sin(t/3)+\cos(t)$, for the initial condition $S_1(0) = 3$ and $S_2(0) = 2$. (b) Prediction error of the DRIPS surrogate, $\mathcal{E}(t) = \|S_{1,\mathrm{DRIPS}}(t) - S_{1,\mathrm{num}}(t)\|/\|S_{1,\mathrm{num}}(t)\|$, for sampling frequency, $1/\Delta t = 10$, used to collect training data.

4.3. Forced oscillator

Consider state variables $S_1(t)$ and $S_2(t)$, whose dynamics satisfies a forced-oscillator model,

$$\begin{aligned} \frac{\mathrm{d}S_1}{\mathrm{d}t} &= S_2, \\ \frac{\mathrm{d}S_2}{\mathrm{d}t} &= -\nu(t)S_1 - S_2 + f(t), \end{aligned} \tag{4.3}$$

with the time-dependent the damping coefficient $\nu(t)$ and forcing $f(t)$. Local parameterization for the inputs is conducted using quadratic polynomials and $N_{\mathrm{sam}}^{\mathbf{p}} = 3^6$ parameter points on the Cartesian grid of $\Omega_{\mathbf{p}} = [-1,1]^6$ are selected for training. For each parameter point, initial conditions $\mathbf{S}(0) = \mathbf{S}_0$ are

sampled randomly from state variable space $\Omega_{\mathbf{S}} = [-3,3]^2$ and $N^{\mathbf{S}}_{\text{sam}} = 3$ pairs of input-output data pairs are collected. The total number of training-data pairs is 2187. The observable is constructed from $S_1(t_k;\mathbf{p}_k)$ and $S_2(t_k;\mathbf{p}_k)$ as

$$\mathbf{y}(t_k;\mathbf{p}_k) = \mathbf{g}(\mathbf{S}(t_k;\mathbf{p}_k)) = [S_1, S_2]^\top.$$

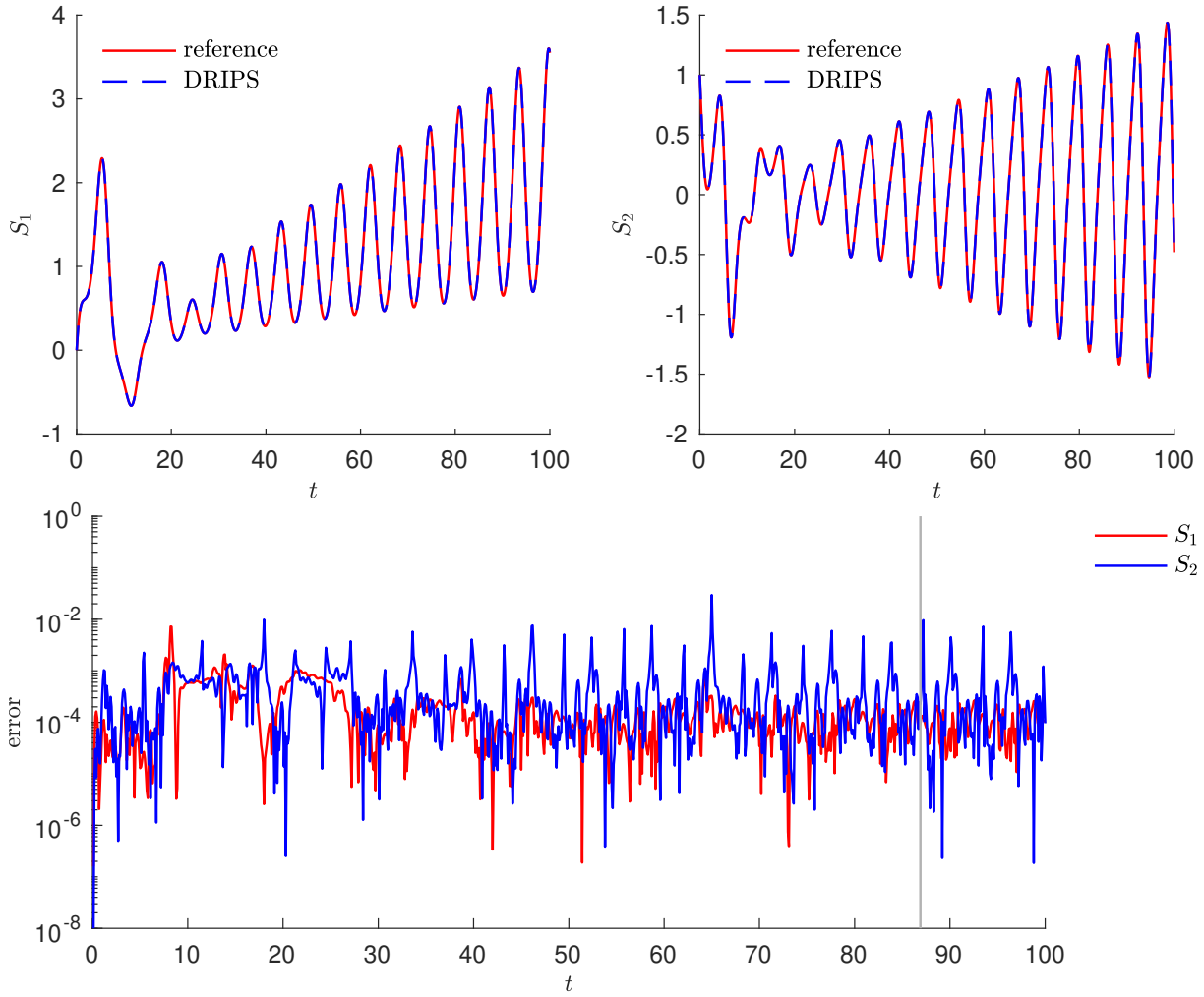

Figure 3: Top: DRIPS prediction and numerical solution of (4.3) with inputs $\nu(t) = \cos(t)$ and $f(t) = t/200$, for the initial condition $S_1(0) = 0$ and $S_2 = 1$; Bottom: Prediction error of the DRIPS surrogate, $\mathcal{E}(t) = \|S_{\text{DRIPS}}(t) - S_{\text{num}}(t)\|_2/(\|S_{\text{DRIPS}}(t)\|_2 + \varepsilon)$ with $\varepsilon = 10^{-2}$, for sampling frequency, $1/\Delta t = 10$, used to collect training data. The gray vertical line indicates the time point beyond which the values of test data are out of the training domain $\Omega_{\mathbf{S}} = [-3,3]$.

The trained DRIPS surrogate is tested on the oscillator with $\nu(t) = \cos(t)$ and $f(t) = t/200$, for the initial condition $\mathbf{S}_0 = [0,1]^\top$ for time up to $T = 100$. In this experiment, we also verify the extrapolation power of the DRIP surrogate: the training data cover the time interval $0 \le t \le 87$, while the dynamics of $S_1(t)$ and $S_2(t)$ is predicted over the time horizon $0 \le t \le 100$. The DRIPS-based prediction of $S_1(t)$ and $S_2(t)$ is visually indistinguishable from their counterparts obtained via numerical solution of (4.3) (Fig. 3), even though the values of $S_1(t)$ at times $t > 87$ are out of the training domain $\Omega_{\mathbf{S}} = [-3,3]$. The relative errors of the DRIPS predictions of $S_1(t)$ and $S_2(t)$ are on the order of $10^{-6} - 10^{-2}$ and stay the same order in the extrapolation regime $t > 87$.

4.4. Heat conduction with source

Consider a state variable $S(x,t)$ whose spatiotemporal evolution is governed by a heat-equation equation with a source term $q(x,t)$,

$$\frac{\partial S}{\partial t} = \frac{\partial S}{\partial x^2} + q(x,t), \qquad x \in [0,1], \quad t > 0. \tag{4.4a}$$

This PDE is subject to the initial and boundary conditions

$$S(x,0) = S_0(x), \qquad S(0,t) = S(1,t) = 0. \tag{4.4b}$$

We set

$$q(x,t) = \alpha(t) \exp\left[-\frac{(x-\mu)^2}{\sigma^2}\right], \tag{4.4c}$$

where $\alpha(t) > 0$ is a time-dependent amplitude, and parameters μ and σ determine its spatial profile. Boundary-value problem (4.4) is solved numerically on a grid comprising $N_S = 20$ equally-spaced grid points spanning the domain $[0,1]$. The discretized state variable $S(x,t)$ is arranged into a vector

$$\mathbf{S}(t) = [S(x_1,t), \ldots, S(x_{N_S},t)]^\top. \tag{4.5}$$

Local parameterization of the input $\alpha(t)$ is conducted using fourth-order polynomials. More specifically, $N^{\mathbf{P}}_{\text{sam}} = 3^7$ parameter points are selected from the Cartesian grid of the local parameterization space $\Omega_{\mathbf{P}} = \Omega_\alpha \times \Omega_\mu \times \Omega_\sigma = [0,1]^5 \times [0,3] \times [0.05, 0.5]$, and $N^{\mathbf{S}}_{\text{sam}} = 25$ training data pairs are generated by randomly sampling from state variable space $\Omega_{\mathbf{S}} = [0,1]^{N_S}$ and doing one-time step simulation. The resulting training dataset comprises 54675 data pairs.

We test the predictive ability of the DRIPS surrogate on a source term with a sawtooth discontinuous function $\alpha = t - \lfloor t \rfloor$ (not in training data set), for $\mu = 1$ and $\sigma = 0.5$. Figure 4 demonstrates that the numerical solution of (4.4) and the prediction of its DRIPS surrogate are visually indistinguishable. The relative errors of DRIPS predictions are on the order of $10^{-3} - 10^{-1}$.

5. Conclusion

We presented a DRIPS (dimension reduction and interpolation for parametric systems) framework for learning a model of unknown nonautonomous dynamical systems from temporal snapshots of quantities of interest. To circumvent the numerical difficulties of computing the spectrum of the nonautonomous Koopman operator, the nonautonomous system is transformed into a family of modified systems over a set of discrete time instances. The modified system, induced by a local parameterization of the external time-dependent inputs over each time instance, is learned via DRIPS. The interpolation of the surrogate models in the parameter space allows one to conduct system predictions for other external time-dependent inputs by computing a parametric ROM of the new modified system iteratively over each time instance.

Our DRIPS framework has several advantages over its competitors. First, unlike strategies based on an approximation of the spectrum of the time-dependent Koopman operator, DRIPS is applicable to general nonautonomous systems without any special requirements on special structures (e.g., periodic/quasi periodic). Second, compared to other data-driven learning method like DNN, our method achieves comparably satisfactory accuracy using much less training data. The efficiency and robustness of DRIPS are demonstrated on various numerical examples of ODEs and PDEs. Future work includes applications of this framework to more complex large systems and rigorous analysis of the surrogate modeling errors.

Acknowledgement

This research was funded in part by the Air Force Office of Scientific Research under grant FA9550-21-1-0381, by the National Science Foundation under award 2100927, by the Office of Advanced Scientific Computing Research (ASCR) within the Department of Energy Office of Science under award number DE-SC0023163, and by the Strategic Environmental Research and Development Program (SERDP) of the Department of Defense under award RC22-3278.

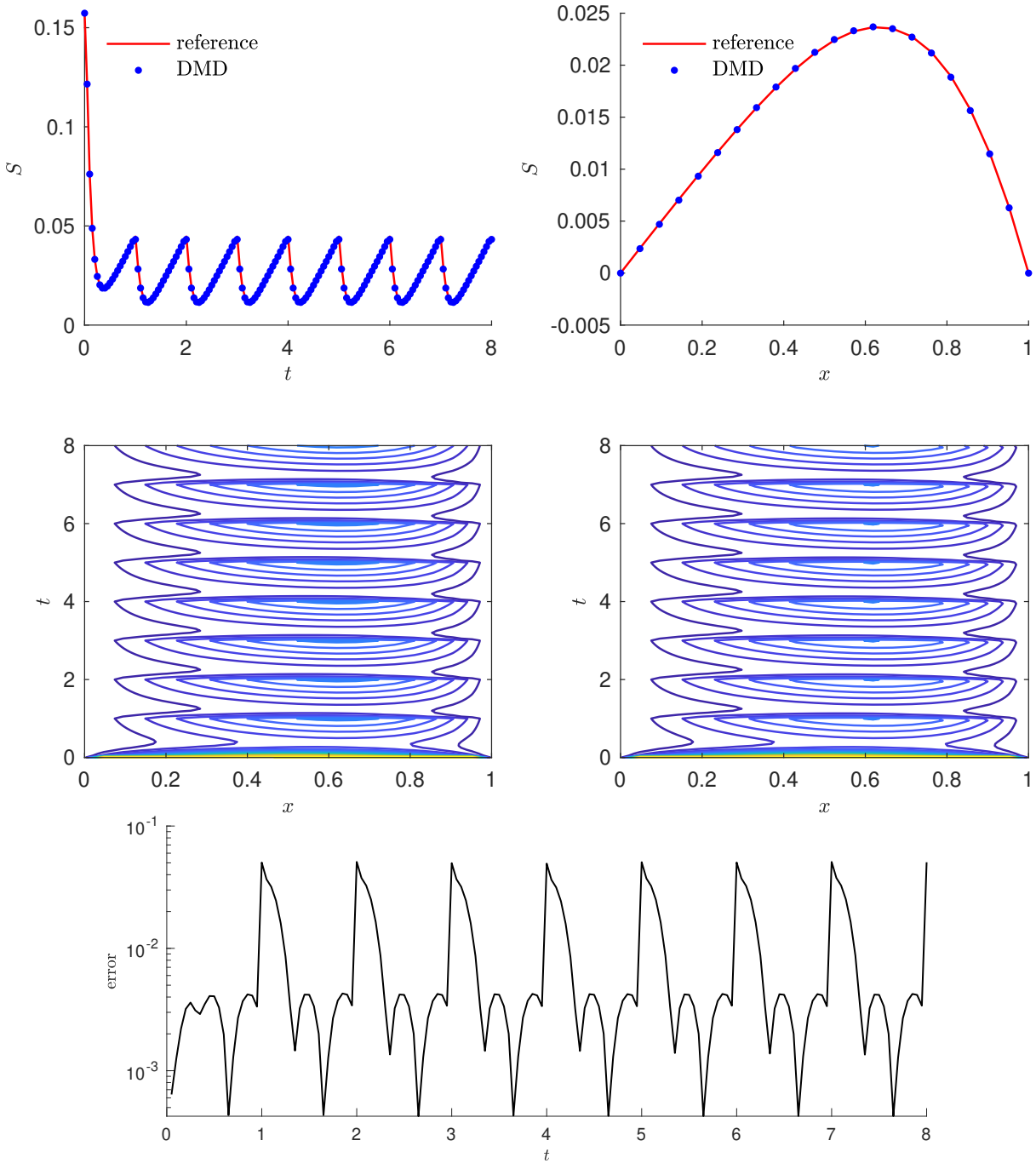

Figure 4: DMD prediction of (4.4) with inputs $\alpha(t) = t - \lfloor t \rfloor$, $\mu = 1$ and $\sigma = 0.5$. (a). Solution evolution at $x = 5$; (b). Solution profile at $t = 2$; (c). Reference solution contours over time; (d). DMD prediction contours over time;(e). Prediction error of the DRIPS surrogate, $\mathcal{E}(t) = \|S_{\text{DRIPS}}(t) - S_{\text{num}}(t)\|_2/(\|S_{\text{num}}(t)\|_2 + \varepsilon)$ with $\varepsilon = 10^{-2}$, for sampling frequency, $1/\Delta t = 20$, used to collect training data.

References

[1] S. L. Brunton, J. L. Proctor, J. N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proceedings of the National Academy of Sciences 113 (15) (2016) 3932–3937.

[2] H. Schaeffer, Learning partial differential equations via data discovery and sparse optimization, Proceedings of the Royal Society A 473 (2197) (2017) 20160446.

[3] J. Bakarji, D. M. Tartakovsky, Data-driven discovery of coarse-grained equations, Journal of Computational Physics 434 (2021) 110219.

[4] M. Raissi, P. Perdikaris, G. E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, Journal of Computational Physics 348 (2017) 683–693.

[5] Q. Li, F. Dietrich, E. M. Bollt, I. G. Kevrekidis, Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator, Chaos 27 (10) (2017) 103111.

[6] M. Korda, I. Mezić, On convergence of extended dynamic mode decomposition to the koopman operator, Journal of Nonlinear Science 28 (2) (2018) 687–710.

[7] M. O. Williams, I. G. Kevrekidis, C. W. Rowley, A data–driven approximation of the Koopman operator: Extending dynamic mode decomposition, Journal of Nonlinear Science 25 (6) (2015) 1307–1346.

[8] R. Rico-Martinez, I. G. Kevrekidis, Continuous time modeling of nonlinear systems: A neural network-based approach, in: IEEE International Conference on Neural Networks, IEEE, 1993, pp. 1522–1525.

[9] M. Raissi, Deep hidden physics models: Deep learning of nonlinear partial differential equations, The Journal of Machine Learning Research 19 (1) (2018) 932–955.

[10] S. H. Rudy, J. N. Kutz, S. L. Brunton, Deep learning of dynamics and signal-noise decomposition with time-stepping constraints, Journal of Computational Physics 396 (2019) 483–506.

[11] P. J. Schmid, Dynamic mode decomposition of numerical and experimental data, Journal of Fluid Mechanics 656 (2010) 5–28.

[12] H. Lu, D. M. Tartakovsky, DRIPS: A framework for dimension reduction and interpolation in parameter space, J. Comput. Phys. 493 (2023) 112455. doi:10.1016/j.jcp.2023.112455.

[13] B. O. Koopman, Hamiltonian systems and transformation in Hilbert space, Proceedings of the National Academy of Sciences 17 (5) (1931) 315–318.

[14] S. L. Brunton, B. W. Brunton, J. L. Proctor, E. Kaiser, J. N. Kutz, Chaos as an intermittently forced linear system, Nature Communications 8 (1) (2017) 1–9.

[15] T. Qin, K. Wu, D. Xiu, Data driven governing equations approximation using deep neural networks, Journal of Computational Physics 395 (2019) 620–635.

[16] Z. Long, Y. Lu, B. Dong, PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network, Journal of Computational Physics 399 (2019) 108925.

[17] K. Wu, D. Xiu, Data-driven deep learning of partial differential equations in modal space, Journal of Computational Physics 408 (2020) 109307.

[18] J. L. Proctor, S. L. Brunton, J. N. Kutz, Dynamic mode decomposition with control, SIAM Journal on Applied Dynamical Systems 15 (1) (2016) 142–161.

[19] J. L. Proctor, S. L. Brunton, J. N. Kutz, Generalizing Koopman theory to allow for inputs and control, SIAM Journal on Applied Dynamical Systems 17 (1) (2018) 909–930.

[20] I. Mezic, A. Surana, Koopman mode decomposition for periodic/quasi-periodic time dependence, IFAC-PapersOnLine 49 (18) (2016) 690–697.

[21] J. N. Kutz, X. Fu, S. L. Brunton, Multiresolution dynamic mode decomposition, SIAM Journal on Applied Dynamical Systems 15 (2) (2016) 713–735.

[22] D. Giannakis, Data-driven spectral decomposition and forecasting of ergodic dynamical systems, Applied and Computational Harmonic Analysis 47 (2) (2019) 338–396.

[23] S. Das, D. Giannakis, Delay-coordinate maps and the spectra of Koopman operators, Journal of Statistical Physics 175 (6) (2019) 1107–1145.

[24] H. Zhang, C. W. Rowley, E. A. Deem, L. N. Cattafesta, Online dynamic mode decomposition for time-varying systems, SIAM Journal on Applied Dynamical Systems 18 (3) (2019) 1586–1609.

[25] S. Macesic, N. Crnjaric-Zic, I. Mezic, Koopman operator family spectrum for nonautonomous systems, SIAM Journal on Applied Dynamical Systems 17 (4) (2018) 2478–2515.

[26] T. Qin, Z. Chen, J. D. Jakeman, D. Xiu, Data-driven learning of nonautonomous systems, SIAM Journal on Scientific Computing 43 (3) (2021) A1607–A1624.

[27] J. N. Kutz, S. L. Brunton, B. W. Brunton, J. L. Proctor, Dynamic mode decomposition: data-driven modeling of complex systems, SIAM, 2016.

[28] S. L. Brunton, B. W. Brunton, J. L. Proctor, J. N. Kutz, Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control, PloS One 11 (2) (2016) e0150171.

[29] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, J. N. Kutz, On dynamic mode decomposition: Theory and applications, Journal of Computational Dynamics 1 (2) (2014) 391–421. doi:10.3934/jcd.2014.1.391.

[30] M. Schmidt, H. Lipson, Distilling free-form natural laws from experimental data, Science 324 (5923) (2009) 81–85.

[31] W.-X. Wang, R. Yang, Y.-C. Lai, V. Kovanis, C. Grebogi, Predicting catastrophes in nonlinear dynamical systems by compressive sensing, Physical Review Letters 106 (15) (2011) 154101.

[32] H. Lu, D. M. Tartakovsky, Lagrangian dynamic mode decomposition for construction of reduced-order models of advection-dominated phenomena, J. Comput. Phys. 407 (2020) 109229. doi:10.1016/j.jcp.2020.109229.

[33] H. Lu, D. M. Tartakovsky, Prediction accuracy of dynamic mode decomposition, SIAM J. Sci. Comput. 42 (3) (2020) A1639–A1662. doi:10.1137/19M1259948.

[34] H. Lu, D. M. Tartakovsky, Dynamic mode decomposition for construction of reduced-order models of hyperbolic problems with shocks, J. Mach. Learn. Model. Comput. 2 (1) (2021) 1–29. doi:10.1615/JMachLearnModelComput.2021036132.

[35] H. Lu, D. M. Tartakovsky, Extended dynamic mode decomposition for inhomogeneous problems, J. Comput. Phys. 444 (2021) 110550. doi:10.1016/j.jcp.2021.110550.

[36] D. Amsallem, C. Farhat, Interpolation method for adapting reduced-order models and application to aeroelasticity, AIAA Journal 46 (7) (2008) 1803–1813.

[37] P.-A. Absil, R. Mahony, R. Sepulchre, Riemannian geometry of Grassmann manifolds with a view on algorithmic computation, Acta Applicandae Mathematica 80 (2) (2004) 199–220.

[38] W. M. Boothby, W. M. Boothby, An introduction to differentiable manifolds and Riemannian geometry, Revised, Vol. 120, Gulf Professional Publishing, 2003.

[39] A. Edelman, T. A. Arias, S. T. Smith, The geometry of algorithms with orthogonality constraints, SIAM Journal on Matrix Analysis and Applications 20 (2) (1998) 303–353.

[40] H. Späth, One dimensional spline interpolation algorithms, AK Peters/CRC Press, 1995.

[41] C. De Boor, A. Ron, Computational aspects of polynomial interpolation in several variables, Mathematics of Computation 58 (198) (1992) 705–727.

[42] D. Amsallem, C. Farhat, An online method for interpolating linear parametric reduced-order models, SIAM Journal on Scientific Computing 33 (5) (2011) 2169–2198.

[43] C. F. Van Loan, G. Golub, Matrix Computations, Johns Hopkins Studies in Mathematical Sciences, Johns Hopkins University Press, 1996.