Journal of Computational Physics 395 (2019) 105–124

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Recovering missing CFD data for high-order discretizations using deep neural networks and dynamics learning

Kevin T. Carlberg [a,*,2,1], Antony Jameson [c,4], Mykel J. Kochenderfer [b,3], Jeremy Morton [b,3], Liqian Peng [a,2], Freddie D. Witherden [b,3]

[a] Sandia National Laboratories, United States of America
[b] Stanford University, United States of America
[c] Texas A&M University, United States of America

ARTICLE INFO

Article history:
Received 1 December 2018
Received in revised form 24 May 2019
Accepted 27 May 2019
Available online 12 June 2019

Keywords:
CFD
High-order schemes
Deep learning
Autoencoders
Dynamics learning
Machine learning

ABSTRACT

Data I/O poses a significant bottleneck in large-scale CFD simulations; thus, practitioners would like to significantly reduce the number of times the solution is saved to disk, yet retain the ability to recover any field quantity (at any time instance) *a posteriori*. The objective of this work is therefore to accurately recover missing CFD data *a posteriori* at any time instance, given that the solution has been written to disk at only a relatively small number of time instances. We consider in particular high-order discretizations (e.g., discontinuous Galerkin), as such techniques are becoming increasingly popular for the simulation of highly separated flows. To satisfy this objective, this work proposes a methodology consisting of two stages: 1) dimensionality reduction and 2) dynamics learning. For dimensionality reduction, we propose a novel hierarchical approach. First, the method reduces the number of degrees of freedom within each element of the high-order discretization by applying autoencoders from deep learning. Second, the methodology applies principal component analysis to compress the global vector of encodings. This leads to a low-dimensional state, which associates with a nonlinear embedding of the original CFD data. For dynamics learning, we propose to apply regression techniques (e.g., kernel methods) to learn the discrete-time velocity characterizing the time evolution of this low-dimensional state. A numerical example on a large-scale CFD example characterized by nearly 13 million degrees of freedom illustrates the suitability of the proposed method in an industrial setting.

* Corresponding author.
E-mail addresses: ktcarlb@sandia.gov (K.T. Carlberg), antony.jameson@tamu.edu (A. Jameson), mykel@stanford.edu (M.J. Kochenderfer), jmorton2@stanford.edu (J. Morton), pengliqian@gmail.com (L. Peng), fdw@stanford.edu (F.D. Witherden).
URLs: http://sandia.gov/~ktcarlb (K.T. Carlberg), http://engineering.tamu.edu/aerospace/profiles/jameson-antony.html (A. Jameson), http://mykel.kochenderfer.com (M.J. Kochenderfer), http://freddie.witherden.org (F.D. Witherden).

[1] 7011 East Ave, MS 9159, Livermore, CA 94550. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

[2] Authors are listed in alphabetical order.

[3] Durand Building, 496 Lomita Mall, Stanford University, Stanford, CA 94305-3035.

[4] 701 H.R. Bright Bldg, College Station, TX 77843-3141.

https://doi.org/10.1016/j.jcp.2019.05.041
0021-9991/

1. Introduction

Industrial practitioners of Computational Fluid Dynamics (CFD) often desire high-fidelity scale-resolving simulations of transient compressible flows within the vicinity of complex geometries. For example, to improve the design of next-generation aircraft, one must simulate—at Reynolds numbers in the range 10^4–10^7 and Mach numbers in the range 0.1–1.0—highly separated flow over deployed spoilers/air-brakes; separated flow within serpentine intake ducts; and flow over entire vehicle configurations at off-design conditions. In order to perform these simulations, it is necessary to solve the unsteady compressible Navier–Stokes equations with a high level of fidelity.

Theoretical studies and numerical experiments have shown high-order accurate discontinuous spectral element methods (DSEM) to be particularly well suited for these types of simulations [52,49,35]. Indeed, we remark that some of the largest CFD simulations to date—involving hundreds of billions of degrees of freedom—have been performed using these methods [51]. DSEMs work by first decomposing the domain into a set of conforming elements and then representing the solution inside of each element by a polynomial. However, in contrast to classical finite element methods, the solution is permitted to be discontinuous across element boundaries. Examples of DSEMs include discontinuous Galerkin (DG) methods [38,10, 15], spectral difference (SD) schemes [26,29,43], and flux reconstruction (FR) schemes [18].

On account of the large number of spatial degrees of freedom (arising from the large number of discretization elements and high-order polynomials within each element) and small time steps that are often required for accuracy, such simulations have the potential to generate petabytes of solution data; this is well beyond what can be handled by current I/O and storage sub-systems. Therefore, it is not practical to write out a sequence of finely spaced solution snapshots to disk for offline analysis and post-processing. Instead, the practitioner must instrument the simulation in advance. Such instrumentation typically includes the online accumulation of time averages of various volumetric expressions, integrating forces on boundaries, and the regular sampling of the flow field at specific points in the volume. However, doing this effectively requires a degree of *a priori* knowledge about the dynamics and evolution of the system, which negates many of the exploratory advantages inherent to simulation.

A common strategy for reducing the storage overhead associated with CFD simulations is to compress the data before writing to disk [20,39,47]. Although these approaches can substantially reduce file sizes, they do not necessarily reduce the overall number of time instances at which a file must be written. In this paper, we instead consider the problem of accurately recovering the full CFD solution at any time instance *a posteriori*, given that it has been written at only a relatively small number of time instances. Several methods have been proposed to reconstruct missing simulation data, especially in the context of CFD. However, most of these methods either aim to reconstruct missing spatial data (which is not the focus of this work) [7,55] or are not amenable to scenarios where no data has been written at some time instances [2,48,12].

To address this work's objective, we propose a novel two-stage strategy that leverages recent innovations in machine learning: 1) dimensionality reduction and 2) dynamics learning. *The first stage is dimensionality reduction*. We propose a novel hierarchical approach to dimensionality reduction that leverages the structure of high-order discretizations. It comprises *local compression* using autoencoders from deep learning followed by *global compression* using principal component analysis (PCA). This stage computes an accurate, low-dimensional, nonlinear embedding of the original data while remaining computationally tractable for large-scale data.

For local compression, we reduce the large (e.g., $\sim 10^2$) number of degrees of freedom per element in the mesh via autoencoders; this step leverages the structure of high-order discretizations. An autoencoder is a (typically deep) neural network that performs dimensionality reduction by learning a nonlinear mapping from high-dimensional feature space to a lower-dimensional encoding [16]. A standard autoencoder consists of two components: an *encoder* network that performs the mapping between input data and the encodings and a *decoder* network that attempts to reconstruct the original input from the encodings. The autoencoder and its variants have previously been used extensively for discovering low-dimensional representations of image data [50,25,45]. However, rather than targeting image data, we use autoencoders for local compression of CFD data. Recent work has explored using autoencoder architectures for discovering low-dimensional representations of fluid flow, primarily for the purposes of simulation and flow control [54,34,30,37,31]. In these studies, *entire solution states* could be treated as neural network inputs (i.e., features) due to the relatively small number of degrees of freedom in the systems being simulated.

Unfortunately, memory constraints prohibit training autoencoders on full solution states for many large-scale CFD simulations. Furthermore, due to the large number of parameters that must be optimized, neural networks typically require many training examples; this work assumes that the solution has been written at only a relatively small number of time instances. The proposed method avoids these issues: the features correspond to the degrees of freedom within a single element (and so pose no memory issue), and the number of training examples is the number of time instances at which the solution has been written *multiplied by* the number of elements in the mesh (which may be very large, e.g., $\sim 10^6$).

Global compression is necessary because even if autoencoders significantly reduce the number of degrees of freedom in each element, the number of elements in the mesh may still be large. We therefore apply classical principal component analysis (with some modifications to handle large-scale data) to reduce the dimensionality of the vector of local encodings across the entire spatial domain. Although PCA identifies a linear subspace, PCA in combination with autoencoders yields a nonlinear embedding of the global CFD solution. In the past, PCA has been widely applied to global data in the context of CFD, but often under the name proper orthogonal decomposition (POD) [17,8]. However, to our knowledge, PCA has not yet been applied to a vector of autoencoder codes.

The second stage is dynamics learning. Because this work assumes that the solution has been written to disk at only a relatively small subset of the desired time instances, we must devise an approach for reconstructing the solution at the missing time instances. For this purpose, we propose learning the dynamics. In particular, we assume that there exists a Markovian discrete-time dynamical system that describes the time-evolution of the low-dimensional latent state (obtained after applying both local and global compression) on the time grid of interest; of course, such a dynamical system may not exist due to the closure problem. Next, we aim to approximate the discrete-time velocity characterizing this hypothesized dynamical system. We regress the (possibly nonlinear) mapping from the low-dimensional latent state at the current time instance to the low-dimensional latent state at the next time instance. For this purpose, we investigate the viability of a wide range of regression techniques within a machine-learning framework for model selection, including support vector regression [42], random forests [5], boosted decision trees [13], k-nearest neighbors [1], the vectorial kernel orthogonal greedy algorithm (VKOGA) [58,59], sparse identification of nonlinear dynamics (SINDy) [6], and dynamic mode decomposition (DMD) [41].

Dynamics learning is an active area of research, and many approaches have been proposed for learning dynamics in multiple fields. However, none has been applied for the purpose of reconstructing missing simulation data, and few of these methods are applicable to the kind of data we consider: very large-scale states, and relatively few time instances at which the state has been recorded. For example, the field of data-driven dynamical systems aims to learn (typically continuous-time) dynamics from observations of the state or velocity. Many of these methods enforce linear dynamics: DMD [41] enforces linear dynamics for the full-system state, while methods based on Koopman theory enforce linear dynamics on a finite number of system observables, which can be specified either manually [57,56,22] or learned using autoencoders [44,34,30,31]. Other works have enabled nonlinear dynamics, but a rich library of candidate basis functions must be manually specified [6], and model selection using validation data has been limited. In contrast to these methods, our work aims to model nonlinear dynamics with a large candidate set of functional forms for the velocity, and subsequently performs model selection using validation data. We also emphasize that the aforementioned autoencoder approaches are not applicable to the large-scale data sets we consider, as the entire state is treated as an input to the autoencoder.

Relatedly, the field of state representation learning [28,3] aims to learn simultaneously both an underlying low-dimensional latent space and discrete-time dynamics on this latent space from a set of high-dimensional observations (e.g., raw pixels from a camera), typically in a control or reinforcement-learning context. The dynamics are typically constrained to be (locally) linear to facilitate control [11,53,21]. In contrast to these approaches, we are not interested in control and thus need not restrict the dynamics to be linear. Finally, these methods would encounter difficulties when applied to the data sets we consider (i.e., a high-dimensional observation space, but data at relatively few time instances), as simultaneously learning an autoencoder and dynamics on an observation space of dimension $\sim 10^7$ requires a substantial amount of computation and data.

Another contribution of this work is the application of the proposed methodology to a large-scale industrial CFD application characterized by a high-order discretization with 320 degrees of freedom per element and 40 584 elements, yielding nearly 13×10^6 degrees of freedom. These numerical experiments demonstrate the viability of the proposed method, as it significantly reduces the dimension of the solution to only 500 (for a compression ratio of 26 000 : 1), yet accurately recovers CFD data, as it incurs sub-1% relative errors in the global solution and drag over all desired 8 600 time instances. The numerical experiments show that the VKOGA algorithm yields the best performance for dynamics learning, due both to its low regression test error and its boundedness. In contrast, while SINDy yielded low test regression error, it yielded an unstable dynamical system to the unboundedness of the polynomial basis functions it employs.

The remainder of the paper is organized as follows. Section 2 provides the problem formulation. Section 3 describes the two-stage dimensionality-reduction process; here, Section 3.1 describes local compression using autoencoders, and Section 3.2 describes global compression using PCA. Section 4 describes dynamics learning, wherein Section 4.1 precisely describes the regression setting, Section 4.2 describes the required training data, and Section 4.3 provides a description of all considered regression models. Section 4.4 provides some analysis related to boundedness of the learned discrete-time velocity, which has implications on the stability of the resulting dynamical-system model. Section 5 reports the numerical experiments. Finally, Section 6 concludes the paper.

2. Problem formulation: recovering a state sequence for high-order discretizations

The objective of this work is to recover a sequence of states $\{\boldsymbol{w}^n\}_{n=0}^{N_t} \subset \mathbb{R}^{N_{\boldsymbol{w}}}$ given only the initial state $\boldsymbol{w}^0 \in \mathbb{R}^{N_{\boldsymbol{w}}}$ and a subset of the elements of the sequence $\{\boldsymbol{w}^n\}_{n\in\mathbb{T}_{\text{sample}}}$ with $\mathbb{T}_{\text{sample}} \subset \{0,\dots,N_t\}$. This scenario often arises in CFD applications, where the practitioner is interested in having access to the fluid solution at many time instances, but it is prohibitively expensive to write the solution to disk for all time instances of interest. In this case, the state corresponds to the vector of fluid variables over the spatial domain, and the sequence corresponds to the value of these degrees of freedom at time instances of interest (which need not correspond to the time steps taken by the time integrator).

This work aims to satisfy this objective in the context of high-order spatial discretizations (e.g., discontinuous Galerkin) of partial differential equations (PDEs), which also often arise in CFD applications. Such discretizations are characterized by N_{el} elements, each of which has $N_{\boldsymbol{w},\text{el}} \gg 1$ local degrees of freedom. Mathematically, we can characterize the state as the vectorization of local states as

$$\boldsymbol{w} := \text{vec}(\boldsymbol{w}_1,\dots,\boldsymbol{w}_{N_{\text{el}}}), \tag{2.1}$$

where $\boldsymbol{w}_i \in \mathbb{R}^{N_{\boldsymbol{w},\mathrm{el}}}$, $i = 1, \ldots, N_{\mathrm{el}}$ denotes the local state associated with the ith element, and $\boldsymbol{w} \in \mathbb{R}^{N_{\boldsymbol{w}}}$ denotes the global state with $N_{\boldsymbol{w}} = N_{\mathrm{el}} N_{\boldsymbol{w},\mathrm{el}}$.

To achieve the stated objective, we adopt a two-stage process:

1. *Dimensionality reduction*. We apply hierarchical dimensionality reduction to the state $\boldsymbol{w}$ in two steps. First, we apply autoencoders to reduce the local dimensionality, i.e., the dimensionality of the local states $\boldsymbol{w}_i$, $i = 1, \ldots, N_{\mathrm{el}}$ (Section 3.1). Second, we apply principal component analysis (PCA) to globally reduce the dimensionality of the set of autoencoder-compressed local states (Section 3.2). This results in a low-dimensional state $\boldsymbol{x} = \mathbf{r}(\boldsymbol{w}) \in \mathbb{R}^{N_x}$ with $N_x \ll N_{\boldsymbol{w}}$ from which the full state can be approximated as $\boldsymbol{w} \approx \mathbf{p}(\boldsymbol{x}) \in \mathbb{R}^{N_{\boldsymbol{w}}}$. Here, $\mathbf{r} : \mathbb{R}^{N_{\boldsymbol{w}}} \to \mathbb{R}^{N_x}$ is the restriction operator associated with dimensionality reduction, and $\mathbf{p} : \mathbb{R}^{N_x} \to \mathbb{R}^{N_{\boldsymbol{w}}}$ is the prolongation operator.
2. *Dynamics learning*. We regress the discrete-time dynamics of the low-dimensional state $\boldsymbol{x}$ (Section 4) under the assumption that the sequence of low-dimensional states $\{\boldsymbol{x}^n\}_{n=0}^{N_t} \subset \mathbb{R}^{N_x}$ can be recovered from the Markovian dynamical system

$$\boldsymbol{x}^{n+1} = \boldsymbol{f}(\boldsymbol{x}^n), \quad n = 0, \ldots, N_t - 1 \tag{2.2}$$

with $\boldsymbol{x}^0 = \mathbf{r}(\boldsymbol{w}^0)$ for some unknown velocity $\boldsymbol{f}$. Then, we construct a regression approximation to the discrete-time velocity $\tilde{\boldsymbol{f}} \approx \boldsymbol{f}$. This allows the desired sequence $\{\boldsymbol{w}^n\}_{n=0}^{N_t}$ to be approximated as $\{\tilde{\boldsymbol{w}}^n\}_{n=0}^{N_t}$, where $\tilde{\boldsymbol{w}}^0 = \boldsymbol{w}^0$ and $\tilde{\boldsymbol{w}}^n = \mathbf{p}(\tilde{\boldsymbol{x}}^n)$, $n = 1, \ldots, N_t$. Here, the approximated low-dimensional state evolves according to the approximated dynamics

$$\tilde{\boldsymbol{x}}^{n+1} = \tilde{\boldsymbol{f}}(\tilde{\boldsymbol{x}}^n), \quad n = 0, \ldots, N_t - 1 \tag{2.3}$$

with $\tilde{\boldsymbol{x}}^0 = \mathbf{r}(\boldsymbol{w}^0)$.

We note that while stage 2 could be executed without stage 1, learning discrete-time dynamics of a low-dimensional state requires substantially less data and computational effort than doing so for a high-dimensional state.

3. Hierarchical dimensionality reduction for high-order discretizations

This section describes the first stage of the proposed methodology: dimensionality reduction. Section 3.1 describes local compression using autoencoders, and Section 3.2 describes global compression using PCA.

3.1. Local compression using autoencoders

The first step of the proposed approach is to reduce the dimensionality of the local states. To achieve this, we make the observation that the total number of available training examples is equal to the number of samples $|\mathbb{T}_{\mathrm{sample}}|$ multiplied by the number of elements N_{el}, where $|\cdot|$ denotes the cardinality of a set. For fine spatial discretizations, N_{el} is large, which implies that we may have access to a large amount of training data, even for a single simulation with $|\mathbb{T}_{\mathrm{sample}}| \sim 100$. Due to this fact, we apply autoencoders, as they often enable very accurate nonlinear embeddings, yet require a large amount of training data.

In the present context, autoencoders aim to find two mappings: the *encoder* $\phi : \mathbb{R}^{N_{\boldsymbol{w},\mathrm{el}}} \to \mathbb{R}^{N_{\hat{\boldsymbol{w}},\mathrm{el}}}$ that maps a local state $\boldsymbol{w}_i \in \mathbb{R}^{N_{\boldsymbol{w},\mathrm{el}}}$ to lower-dimensional representation $\hat{\boldsymbol{w}}_i \in \mathbb{R}^{N_{\hat{\boldsymbol{w}},\mathrm{el}}}$; i.e. $\phi : \boldsymbol{w}_i \mapsto \hat{\boldsymbol{w}}_i$, $i = 1, \ldots, N_{\mathrm{el}}$; and the *decoder* $\psi : \mathbb{R}^{N_{\hat{\boldsymbol{w}},\mathrm{el}}} \to \mathbb{R}^{N_{\boldsymbol{w},\mathrm{el}}}$. Here, $N_{\hat{\boldsymbol{w}},\mathrm{el}} (\ll N_{\boldsymbol{w},\mathrm{el}})$ denotes the reduced dimension of the local state vector or *code*. Its value dictates the tradeoff between compression and reconstruction accuracy.

The encoder ϕ takes the form of a feed-forward neural network with N_{layers} layers constructed such that

$$\phi(\boldsymbol{w}_i; \boldsymbol{\theta}) = \mathbf{h}_{N_{\mathrm{layers}}}(\cdot; \boldsymbol{\Theta}_{N_{\mathrm{layers}}}) \circ \mathbf{h}_{N_{\mathrm{layers}}-1}(\cdot; \boldsymbol{\Theta}_{N_{\mathrm{layers}}-1}) \circ \cdots \circ \mathbf{h}_1(\boldsymbol{w}_i; \boldsymbol{\Theta}_1), \tag{3.1}$$

where $\mathbf{h}_i(\cdot; \boldsymbol{\Theta}_i) : \mathbb{R}^{p_{i-1}} \to \mathbb{R}^{p_i}$, $i = 1, \ldots, N_{\mathrm{layers}}$ denotes the function applied at layer i of the neural network; $\boldsymbol{\Theta}_i$, $i = 1, \ldots, N_{\mathrm{layers}}$ denote the weights employed at layer i; p_i denotes the dimensionality of the output at layer i; and $\boldsymbol{\theta} \equiv (\boldsymbol{\Theta}_1, \ldots, \boldsymbol{\Theta}_{N_{\mathrm{layers}}})$. The input is of dimension $p_0 = N_{\boldsymbol{w},\mathrm{el}}$ and the final (output) layer ($i = N_{\mathrm{layers}}$) produces the code $\hat{\boldsymbol{w}}_i = \phi(\boldsymbol{w}_i)$ such that $p_{N_{\mathrm{layers}}} = N_{\hat{\boldsymbol{w}},\mathrm{el}}$. An activation function h is applied in layers 1 to N_{layers} to some function of the weights and the outputs from the previous layer $\mathbf{y}_{i-1} \in \mathbb{R}^{p_{i-1}}$ such that

$$\mathbf{h}_i(\mathbf{y}_{i-1}; \boldsymbol{\Theta}_i) = h(g(\boldsymbol{\Theta}_i, \mathbf{y}_{i-1})), \tag{3.2}$$

where $g(\boldsymbol{\Theta}_i, \mathbf{y}_{i-1}) = \boldsymbol{\Theta}_i[1; \mathbf{y}_{i-1}]$ with $\boldsymbol{\Theta}_i$ a real-valued matrix for a traditional multilayer perceptron (MLP), while g corresponds to a convolution operator for a convolutional neural network with $\boldsymbol{\Theta}_i$ providing the convolutional-filter weights. Note that $\mathbf{y}_0 := \boldsymbol{w}_i$. The activation function is applied element-wise to the vector argument; common choices include the rectified linear unit (ReLU), the logistic sigmoid, and the hyperbolic tangent.

Analogously, the decoder ψ also takes the form of a feed-forward artificial neural network with $\bar{N}_{\mathrm{layers}}$ such that

$$\psi(\boldsymbol{w}_i; \bar{\boldsymbol{\theta}}) = \bar{\mathbf{h}}_{\bar{N}_{\mathrm{layers}}}(\cdot; \bar{\boldsymbol{\Theta}}_{\bar{N}_{\mathrm{layers}}}) \circ \bar{\mathbf{h}}_{\bar{N}_{\mathrm{layers}}-1}(\cdot; \bar{\boldsymbol{\Theta}}_{\bar{N}_{\mathrm{layers}}-1}) \circ \cdots \circ \bar{\mathbf{h}}_1(\boldsymbol{w}_i; \bar{\boldsymbol{\Theta}}_1), \tag{3.3}$$

with $\bar{\mathbf{h}}_i(\cdot;\bar{\boldsymbol{\Theta}}_i):\mathbb{R}^{\bar{p}_{i-1}}\to\mathbb{R}^{\bar{p}_i}$, $i=1,\dots,\bar{N}_{\text{layers}}$, and $\bar{\boldsymbol{\theta}}\equiv[\text{vec}(\bar{\boldsymbol{\Theta}}_1);\ \dots\ ;\text{vec}(\bar{\boldsymbol{\Theta}}_{\bar{N}_{\text{layers}}})]$. The input is of dimension $\bar{p}_0=N_{\hat{\boldsymbol{w}},\text{el}}$ and the final (output) layer ($i=\bar{N}_{\text{layers}}$) produces the high-dimensional representation $\tilde{\boldsymbol{w}}_i=\psi(\hat{\boldsymbol{w}}_i)$ such that $\tilde{\boldsymbol{w}}_i\approx\boldsymbol{w}_i$ and $\bar{p}_{\bar{N}_{\text{layers}}}=N_{\boldsymbol{w},\text{el}}$. As before,

$$\bar{\mathbf{h}}_i(\mathbf{y}_{i-1};\bar{\boldsymbol{\Theta}}_i)=h(g(\bar{\boldsymbol{\Theta}}_i,\mathbf{y}_{i-1})), \tag{3.4}$$

where $\mathbf{y}_{i-1}\in\mathbb{R}^{\bar{p}_{i-1}}$ is the output from the previous layer, and $\mathbf{y}_0:=\hat{\boldsymbol{w}}_i$.

To train the autoencoder, we use data corresponding to the local states at a subset of time instances $\mathbb{T}_{\text{autoencoder}}\subseteq\mathbb{T}_{\text{sample}}$, i.e., the training data corresponds to

$$\{\boldsymbol{w}_i^n\}_{i=1,\dots,N_{\text{el}}}^{n\in\mathbb{T}_{\text{autoencoder}}}.$$

We then compute the weights $(\boldsymbol{\theta}^\star,\bar{\boldsymbol{\theta}}^\star)$ by solving the minimization problem

$$\underset{(\boldsymbol{\theta},\bar{\boldsymbol{\theta}})}{\text{minimize}}\sum_{n\in\mathbb{T}_{\text{autoencoder}}}\sum_{i=1}^{N_{\text{el}}}\|\boldsymbol{w}_i^n-\psi(\cdot;\bar{\boldsymbol{\theta}})\circ\phi(\boldsymbol{w}_i^n;\boldsymbol{\theta})\|_2^2+\Omega(\boldsymbol{\theta},\bar{\boldsymbol{\theta}}), \tag{3.5}$$

where Ω denotes a regularization function, e.g., ridge, lasso, elastic-net [61]. We note that this optimization problem is often solved approximately to improve generalization behavior; for instance, optimization iterations are often terminated when the error on an independent validation set begins to increase; see Ref. [4] for a review of training deep neural networks. The encoder and decoder are then set to $\phi=\phi(\cdot;\boldsymbol{\theta}^\star)$ and $\psi=\psi(\cdot;\bar{\boldsymbol{\theta}}^\star)$, respectively.

After applying the encoder, the global reduced state takes the form

$$\hat{\boldsymbol{w}}:=\text{vec}(\hat{\boldsymbol{w}}_1,\dots,\hat{\boldsymbol{w}}_{N_{\text{el}}})\in\mathbb{R}^{N_{\text{el}}N_{\hat{\boldsymbol{w}},\text{el}}}, \tag{3.6}$$

where $\hat{\boldsymbol{w}}_i=\phi(\boldsymbol{w}_i)$, $i=1,\dots,N_{\text{el}}$. Note that we can also write $\hat{\boldsymbol{w}}=\phi_{\text{global}}(\boldsymbol{w})$, where

$$\begin{aligned}\phi_{\text{global}}:\boldsymbol{w}&\mapsto\text{vec}(\phi(\boldsymbol{w}_1),\dots,\phi(\boldsymbol{w}_{N_{\text{el}}}))\\&:\mathbb{R}^{N_{\boldsymbol{w}}}\to\mathbb{R}^{N_{\text{el}}N_{\hat{\boldsymbol{w}},\text{el}}}.\end{aligned} \tag{3.7}$$

The dimension of the global reduced state $\hat{\boldsymbol{w}}$ is $N_{\text{el}}N_{\hat{\boldsymbol{w}},\text{el}}$; while this is significantly smaller than the dimension $N_{\boldsymbol{w}}=N_{\text{el}}N_{\boldsymbol{w},\text{el}}$ of the full state $\boldsymbol{w}$, it may still be large if the number of elements N_{el} is large, as is the case for fine spatial discretizations. To address this, Section 3.2 describes how PCA can be applied to reduce the dimensionality of the global reduced state $\hat{\boldsymbol{w}}$.

3.2. Global compression using PCA

Because the dimensionality of the global reduced state $\hat{\boldsymbol{w}}=\phi_{\text{global}}(\boldsymbol{w})\in\mathbb{R}^{N_{\text{el}}N_{\hat{\boldsymbol{w}},\text{el}}}$ may still be large, we proceed by applying dimensionality reduction to this global quantity. However, in this case we have many fewer training examples than in the case of local compression. This arises from the fact that local compression employs a training set whose cardinality is the product of the number of training time instances and the number of elements (which is often large). The size of the global-compression training set is simply the number of training time instances. Thus, for global compression, we use PCA, a dimensionality reduction method that does not rely on access to a large amount of data. Furthermore, because we are usually considering large-scale data, it may be computationally costly to compute global principal components to represent the global reduced state vector $\hat{\boldsymbol{w}}$; thus, we consider piecewise PCA for this purpose.

To achieve this, we first define a set of training instances $\mathbb{T}_{\text{PCA}}\equiv\{n_{\text{PCA},j}\}_j\subseteq\mathbb{T}_{\text{sample}}$ and associated global reduced states $\{\hat{\boldsymbol{w}}^n\}^{n\in\mathbb{T}_{\text{PCA}}}$ that will be employed to compute the principal components. Then, we decompose the global reduced state into $n_{\hat{\boldsymbol{w}}}$ components $\hat{\mathbf{w}}_i$, $i=1,\dots,n_{\hat{\boldsymbol{w}}}$ such that $\hat{\boldsymbol{w}}\equiv[\hat{\mathbf{w}}_1^T\ \cdots\ \hat{\mathbf{w}}_{n_{\hat{\boldsymbol{w}}}}^T]^T$. Next, we compute the singular value decompositions

$$[(\hat{\mathbf{w}}_i^{n_{\text{PCA},1}}-\bar{\mathbf{w}}_i)\ \cdots\ (\hat{\mathbf{w}}_i^{n_{\text{PCA},|\mathbb{T}_{\text{PCA}}|}}-\bar{\mathbf{w}}_i)]=\boldsymbol{U}_i\boldsymbol{\Sigma}_i\boldsymbol{V}_i^T,\quad i=1,\dots,n_{\hat{\boldsymbol{w}}}, \tag{3.8}$$

where

$$\bar{\mathbf{w}}_i:=\frac{1}{|\mathbb{T}_{\text{PCA}}|}\sum_{n\in\mathbb{T}_{\text{PCA}}}\hat{\mathbf{w}}_i^n,\quad i=1,\dots,n_{\hat{\boldsymbol{w}}} \tag{3.9}$$

denote the sample means, and $\boldsymbol{U}_i\equiv[\boldsymbol{u}_{i,1}\ \cdots\ \boldsymbol{u}_{i,|\mathbb{T}_{\text{PCA}}|}]$, $\boldsymbol{\Sigma}_i=\text{diag}(\sigma_{i,j})_{j=1}^{|\mathbb{T}_{\text{PCA}}|}$ with

$$\sigma_{i,1}\geq\cdots\sigma_{i,|\mathbb{T}_{\text{PCA}}|}\geq 0. \tag{3.10}$$

Subsequently, PCA truncates the left singular vectors to obtain the basis matrix

$$\boldsymbol{\Phi}_i=\begin{bmatrix}\boldsymbol{u}_{i,1}&\cdots&\boldsymbol{u}_{i,N_{x,i}}\end{bmatrix}, \tag{3.11}$$

where $N_{x,i}$ can be set according to a statistical energy criterion, for example. We note that the resulting basis matrix satisfies the minimization problem

$$\mathrm{Ran}(\boldsymbol{\Phi}_i) = \underset{\mathcal{S},\, \dim(\mathcal{S})=N_{x,i}}{\arg\min} \sum_{j=1}^{|\mathbb{T}_{\mathrm{PCA}}|} \|(\boldsymbol{I} - \boldsymbol{P}_{\mathcal{S}})(\hat{\mathbf{w}}_i^{n_{\mathrm{PCA},j}} - \bar{\mathbf{w}}_i)\|_2^2, \tag{3.12}$$

where $\boldsymbol{P}_{\mathcal{S}}$ denotes the orthogonal projector (in the Euclidean norm) onto the subspace $\mathcal{S}$; and minimization is taken over all subspaces of dimension $N_{x,i}$ (i.e., the Grassmannian).

We then compute the singular value decomposition

$$[\mathrm{diag}(\boldsymbol{\Phi}_i)]^T[(\hat{\boldsymbol{w}}^{n_{\mathrm{PCA},1}} - \bar{\boldsymbol{w}}) \;\cdots\; (\hat{\boldsymbol{w}}^{n_{\mathrm{PCA},|\mathbb{T}_{\mathrm{PCA}}|}} - \bar{\boldsymbol{w}})] = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T, \tag{3.13}$$

with $\boldsymbol{U} \equiv [\boldsymbol{u}_1 \;\cdots\; \boldsymbol{u}_{|\mathbb{T}_{\mathrm{PCA}}|}]$ and

$$\bar{\boldsymbol{w}} := \frac{1}{|\mathbb{T}_{\mathrm{PCA}}|} \sum_{n\in\mathbb{T}_{\mathrm{PCA}}} \hat{\boldsymbol{w}}^n. \tag{3.14}$$

We define the global basis matrix $\boldsymbol{\Phi} := \mathrm{diag}(\boldsymbol{\Phi}_i)\begin{bmatrix}\boldsymbol{u}_1 & \cdots & \boldsymbol{u}_{N_x}\end{bmatrix} \in \mathbb{R}^{N_{\mathrm{el}}N_{\hat{\boldsymbol{w}},\mathrm{el}}\times N_x}$ with $N_x \leq \sum_i N_{x,i} \ll N_{\mathrm{el}}N_{\hat{\boldsymbol{w}},\mathrm{el}} \ll N_{\mathrm{el}}N_{\boldsymbol{w},\mathrm{el}}$ determined from an energy criterion, for example.

Now, we can define the reduced global state:

$$\boldsymbol{x}^n = \mathbf{r}(\boldsymbol{w}^n), \quad n = 0, \ldots, N_t, \tag{3.15}$$

where $\mathbf{r} : \boldsymbol{w} \mapsto \boldsymbol{\Phi}^T(\phi_{\mathrm{global}}(\boldsymbol{w}) - \bar{\mathbf{w}}_i)$. The mapping from reduced state to the approximated global state is then $\mathbf{p} : \boldsymbol{x} \mapsto \psi_{\mathrm{global}}(\boldsymbol{\Phi}\boldsymbol{x} + \bar{\mathbf{w}}_i)$, where

$$\begin{aligned}\psi_{\mathrm{global}} : \hat{\boldsymbol{w}} &\mapsto \mathrm{vec}(\psi(\hat{\boldsymbol{w}}_1), \ldots, \psi(\hat{\boldsymbol{w}}_{N_{\mathrm{el}}})) \\ &: \mathbb{R}^{N_{\mathrm{el}}N_{\hat{\boldsymbol{w}},\mathrm{el}}} \to \mathbb{R}^{N_{\boldsymbol{w}}}.\end{aligned} \tag{3.16}$$

4. Dynamics learning using regression

This section describes dynamics learning. In particular, we assume that the sequence of low-dimensional states $\{\boldsymbol{x}^n\}_{n=0}^{N_t} \subset \mathbb{R}^{N_x}$ with $\boldsymbol{x}^n$ defined in Eq. (3.15) can be recovered from the Markovian discrete-time dynamical system

$$\boldsymbol{x}^{n+1} = \boldsymbol{f}(\boldsymbol{x}^n), \quad n = 0, \ldots, N_t - 1 \tag{4.1}$$

for some (unknown) discrete-time velocity $\boldsymbol{f} : \mathbb{R}^{N_x} \to \mathbb{R}^{N_x}$. Then, if such a velocity exists and can be computed, the entire sequence of low-dimensional states $\{\boldsymbol{x}^n\}_{n=0}^{N_t} \subset \mathbb{R}^{N_x}$ can be recovered from Eq. (4.1) by specifying the initial state $\boldsymbol{x}^0$ only; this allows an approximation to the sequence of states to be computed as $\{\boldsymbol{w}^n\}_{n=1}^{N_t} \approx \{\mathbf{p}(\boldsymbol{x}^n)\}_{n=1}^{N_t}$ (note that $\boldsymbol{w}^0$ is given).

Because we do not have direct access to this discrete-time velocity $\boldsymbol{f}$, we must generate an approximation $\tilde{\boldsymbol{f}} \approx \boldsymbol{f}$ with $\tilde{\boldsymbol{f}} : \mathbb{R}^{N_x} \to \mathbb{R}^{N_x}$. This work constructs this approximation by regressing each component of the velocity. Once the approximated velocity $\tilde{\boldsymbol{f}}$ is constructed, the sequence of low-dimensional states $\{\boldsymbol{x}^n\}_{n=0}^{N_t} \subset \mathbb{R}^{N_x}$ can be approximated as $\{\tilde{\boldsymbol{x}}^n\}_{n=0}^{N_t} \subset \mathbb{R}^{N_x}$, where $\tilde{\boldsymbol{x}}^0 = \boldsymbol{x}^0$ and $\tilde{\boldsymbol{x}}^n$, $n = 1, \ldots, N_t$ satisfies the approximated discrete-time dynamics

$$\tilde{\boldsymbol{x}}^{n+1} = \tilde{\boldsymbol{f}}(\tilde{\boldsymbol{x}}^n), \quad n = 0, \ldots, N_t - 1. \tag{4.2}$$

Then, the desired sequence of states $\{\boldsymbol{w}^n\}_{n=0}^{N_t}$ can be approximated as $\{\tilde{\boldsymbol{w}}^n\}_{n=0}^{N_t}$, where $\tilde{\boldsymbol{w}}^0 = \boldsymbol{w}^0$ and $\tilde{\boldsymbol{w}}^n = \mathbf{p}(\tilde{\boldsymbol{x}}^n)$, $n = 1, \ldots, N_t$.

4.1. Regression setting

To cast this as a regression problem, we consider the ith equation of (4.1):

$$x_i^{n+1} = f_i(\boldsymbol{x}^n), \quad n = 0, \ldots, N_t - 1, \quad i = 1, \ldots, N_x, \tag{4.3}$$

where $\boldsymbol{x} \equiv [x_1 \;\cdots\; x_{N_x}]^T$ and $\boldsymbol{f} \equiv [f_1 \;\cdots\; f_{N_x}]^T$. This expression shows that the ith component of the velocity f_i can be interpreted as a function that maps the reduced state at the current time step $\boldsymbol{x}^n$ to the ith element of the state at the next time step x_i^{n+1} for $n = 0, \ldots, N_t - 1$.

Based on this observation, we construct independent regression models $\tilde{f}_i \approx f_i$, $i = 1, \ldots, N_x$ whose *features* (i.e., regression-model inputs) correspond to the state at the current time step $\boldsymbol{x}^n$, and whose *response* (i.e., regression-model output) corresponds to the ith element of the state at the next time step x_i^{n+1} for $n = 0, \ldots, N_t - 1$. The approximated velocity is then $\tilde{\boldsymbol{f}} \equiv [\tilde{f}_1 \;\cdots\; \tilde{f}_{N_x}]^T$. Section 4.2 describes the training data used to construct the regression models, and Section 4.3 describes the proposed regression models.

4.2. Training data

Based on the regression setting described in Section 4.1, it is clear that the appropriate training data for constructing the ith regression model $\tilde{f}_i$ corresponds to response–feature pairs

$$\mathcal{T}_{\text{train,i}} := \{(x_i^{n+1}, \boldsymbol{x}^n)\}_{n\in\mathbb{T}_{\text{reg}}},$$

where $\mathbb{T}_{\text{reg}} \equiv \{n_{\text{reg},j}\}_j \subset \mathbb{T}_{\text{sample}}$ satisfies $n_{\text{reg},j}+1 \in \mathbb{T}_{\text{sample}}$, $j = 1,\dots,n_{\text{train}}$, where $n_{\text{train}} := |\mathbb{T}_{\text{reg}}| = |\mathcal{T}_{\text{train,i}}|$, $i = 1,\dots,N_x$ such that sequential states (required for accessing both the features and the response) are accessible from the available sample set $\mathbb{T}_{\text{sample}}$.

4.3. Regression models

In this work, we consider a wide range of types of models for constructing regression models $\tilde{f}_i$, $i = 1,\dots,N_x$. We denote a generic regression model as $\tilde{f}$ and its training data as

$$\mathcal{T}_{\text{train}} \equiv \{(\bar{y}^i, \bar{\boldsymbol{x}}^i)\}_{i=1}^{n_{\text{train}}} \tag{4.4}$$

For all candidate models, we employ validation for model selection; this process chooses values of the hyperparameters characterizing each model.

4.3.1. Support vector regression

Support vector regression (SVR) [42] employs a model

$$\tilde{f}(\boldsymbol{x};\boldsymbol{\theta}) = \langle \boldsymbol{\theta}, \boldsymbol{\psi}(\boldsymbol{x})\rangle + b, \tag{4.5}$$

where $\boldsymbol{\psi} : \mathbb{R}^{N_x} \to \mathcal{H}$, $\boldsymbol{\theta} \in \mathcal{H}$, and $\mathcal{H}$ is a (potentially unknown) feature space equipped with inner product $\langle\cdot,\cdot\rangle$. SVR aims to compute a 'flat' function (i.e., $\langle\boldsymbol{\theta},\boldsymbol{\theta}\rangle$ small) that penalizes prediction errors that exceed a specified threshold ϵ (i.e., soft margin loss). SVR uses slack variables $\boldsymbol{\xi}$ and $\boldsymbol{\xi}^\star$ to address deviations exceeding epsilon margin (ϵ) and employs the box constraint (C) to penalize these deviations, leading to the primal formula [9]

$$\begin{aligned}
&\underset{\bar{\boldsymbol{\theta}},b,\boldsymbol{\xi},\boldsymbol{\xi}^\star}{\text{minimize}}\ \frac{1}{2}\langle\bar{\boldsymbol{\theta}},\bar{\boldsymbol{\theta}}\rangle + C\sum_{i=1}^{n_{\text{train}}}(\xi_i+\xi_i^\star),\\
&\text{subject to } \bar{y}^i - \langle\bar{\boldsymbol{\theta}},\boldsymbol{\psi}(\bar{\boldsymbol{x}}^i)\rangle - b \le \epsilon+\xi_i, \quad i=1,\dots,n_{\text{train}},\\
&\qquad\qquad \langle\bar{\boldsymbol{\theta}},\boldsymbol{\psi}(\bar{\boldsymbol{x}}^i)\rangle + b - \bar{y}^i \le \epsilon+\xi_i^\star, \quad i=1,\dots,n_{\text{train}},\\
&\qquad\qquad \boldsymbol{\xi},\ \boldsymbol{\xi}^\star \ge \mathbf{0},
\end{aligned} \tag{4.6}$$

whose corresponding dual problem is

$$\begin{aligned}
&\underset{\boldsymbol{\alpha},\boldsymbol{\alpha}^\star}{\text{minimize}}\ \frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^\star)^T\mathbf{Q}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^\star) + \epsilon\mathbf{1}^T(\boldsymbol{\alpha}+\boldsymbol{\alpha}^\star) - [\bar{y}^1\ \cdots\ \bar{y}^{n_{\text{train}}}]^T(\boldsymbol{\alpha}-\boldsymbol{\alpha}^\star),\\
&\text{subject to } \mathbf{1}^T(\boldsymbol{\alpha}-\boldsymbol{\alpha}^\star) = 0,\\
&\qquad\qquad 0 \le \alpha_i,\ \alpha_i^\star \le C, \quad i=1,\dots,n_{\text{train}}.
\end{aligned}$$

Here, $Q_{ij} := K(\bar{\boldsymbol{x}}^i,\bar{\boldsymbol{x}}^j) := \langle\boldsymbol{\psi}(\bar{\boldsymbol{x}}^i),\boldsymbol{\psi}(\bar{\boldsymbol{x}}^j)\rangle$ and $\boldsymbol{\theta} = \sum_{i=1}^{n_{\text{train}}}(\alpha_i-\alpha_i^\star)\boldsymbol{\psi}(\bar{\boldsymbol{x}}^i)$. The resulting model (4.5) can be equivalently expressed as

$$\tilde{f}(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{i=1}^{n_{\text{train}}}\left(\alpha_i-\alpha_i^\star\right)K(\bar{\boldsymbol{x}}^i,\boldsymbol{x}) + b. \tag{4.7}$$

There exist many kernel functions $K(\bar{\boldsymbol{x}}^i,\bar{\boldsymbol{x}}^j)$ that correspond to an inner product in a feature space $\mathcal{H}$. We consider both a Gaussian radial basis function (RBF) kernel $K(\bar{\boldsymbol{x}}^i,\bar{\boldsymbol{x}}^j) = \exp(-\gamma\|\bar{\boldsymbol{x}}^i-\bar{\boldsymbol{x}}^j\|^2)$ and polynomial kernel $K(\bar{\boldsymbol{x}}^i,\bar{\boldsymbol{x}}^j) = (1+[\bar{\boldsymbol{x}}^i]^T\bar{\boldsymbol{x}}^j)^q$. When the RBF kernel is used, we refer to the method as SVRrbf; when using the polynomial kernel with $q=2$ or 3, we refer to the method SVR2 and SVR3, respectively. In the numerical experiments, we apply a box constraint $C=1$ and select the sensitivity margin ϵ using a validation set.

4.3.2. Random forests

Random forests (RF) [5] use decision trees constructed by decomposing feature space (in this case $\mathbb{R}^{N_x}$) along canonical directions in a way that sequentially minimizes the mean-squared prediction error over the training data. The prediction generated by a decision tree corresponds to the average value of the response over the training data that reside within the same feature-space region as the prediction point.

Decision trees are low bias and high variance; thus, random forests employ a variance-reduction method, namely bootstrap aggregating (i.e., bagging), to reduce the prediction variance. Here, N_{tree} different data sets are generated by sampling the original training set $\mathcal{T}_{\text{train}}$ with replacement. The method then constructs a decision tree from each of these training sets, yielding N_{tree} regression functions $\tilde{f}^i$, $i = 1, \dots, N_{\text{tree}}$. The ultimate regression function corresponds to the average prediction across the ensemble:

$$\tilde{f}(\boldsymbol{x}) = \frac{1}{N_{\text{tree}}} \sum_{1=1}^{N_{\text{tree}}} \tilde{f}^i(\boldsymbol{x}). \tag{4.8}$$

To decorrelate the decision trees and further reduce prediction variance, random forests introduce an additional source of randomness. When training each tree, a random subset of $N_{\text{split}} \ll N_x$ features is considered when performing the feature-space split at each node in the tree.

The hyper-parameter we consider for this approach corresponds to the number of trees in the ensemble N_{tree}. We set $N_{\text{split}} = N_x/3$ for splitting during training.

4.3.3. Boosted decision trees

Boosted decision trees [13] combines weak learners (decision trees) into a single strong learner in an iterative fashion. The algorithm adds one tree at each stage. Let F_m be the aggregate model at stage m, where $m = 1, \dots, N_{\text{tree}}$. The gradient boosting algorithm improves upon F_m by constructing a new model that adds a weak learner h_m to yield a more accurate model $F_{m+1}(\boldsymbol{x}) = F_m(\boldsymbol{x}) + \alpha_m h_m(\boldsymbol{x})$, where α_m is a constant.

The numerical experiments employ LSBoost, where the goal is to minimize the mean squared error $\sum_{i=1}^{n_{\text{train}}} |\tilde{f}(\bar{\boldsymbol{x}}^i) - \bar{y}^i|^2$. Decision stumps are applied as weak learners $h_m(\boldsymbol{x})$. Initially, $F_0 = h_0(\boldsymbol{x})$. At iteration m (for $m = 1, \cdots, N_{\text{tree}}$), compute the residual $r_i = \bar{y}^i - F_{m-1}(\bar{\boldsymbol{x}}^i)$ $(i = 1, \dots, n_{\text{train}})$ and solve the optimization problem

$$(\alpha_m, h_m) = \arg\min_{\alpha,h} \sum_{i=1}^{n_{\text{train}}} |r_i - \alpha h(\bar{\boldsymbol{x}}^i)|^2.$$

The final result is given by $\tilde{f}(\boldsymbol{x}) = F_{N_{\text{tree}}}(\boldsymbol{x})$.

The hyper-parameter we consider for this approach corresponds to the number of trees N_{tree} in the ensemble.

4.3.4. k-nearest neighbors

The k-nearest neighbors (k-NN) method [1] produces predictions corresponding to a weighted average of the responses associated with the k-nearest training points in feature space, i.e.,

$$\tilde{f}(\boldsymbol{x}) = \sum_{i \in \mathcal{I}(k)} \tau(\bar{\boldsymbol{x}}^i, \boldsymbol{x}) \bar{\boldsymbol{x}}^i.$$

Here, $\mathcal{I}(k) \subseteq \{1, \dots, n_{\text{train}}\}$ with $|\mathcal{I}(k)| = k (\leq n_{\text{train}})$ satisfies $\|\boldsymbol{x} - \bar{\boldsymbol{x}}^i\|_2 \leq \|\boldsymbol{x} - \bar{\boldsymbol{x}}^j\|_2$ for all $i \in \mathcal{I}(k)$, $j \in \{1, \dots, n_{\text{train}}\} \setminus \mathcal{I}(k)$. In the numerical experiments, we consider only uniform weights $\tau(\bar{\boldsymbol{x}}^j, \boldsymbol{x}) := \frac{1}{k}$.

The hyper-parameter we consider for this method corresponds to the number of nearest neighbors k.

4.3.5. Vectorial kernel orthogonal greedy algorithm (VKOGA)

In support vector machines, kernel-based interpolation is applied to scalar-valued functions (as in Eq. (4.7)). If each individual regression model $\tilde{f}_i$, $i = 1, \dots, N_x$ requires n_{train} sets of kernels to construct its approximation, the resulting regression model for the vector of responses $\tilde{\boldsymbol{f}}$ would require $N_x n_{\text{train}}$ independent kernels. This is expensive for both training and evaluation when $N_x \gg 1$. To reduce the overall number of kernels, one can impose the restriction that a common subspace be used for every component of the vector-valued response. In particular, the vectorial kernel orthogonal greedy algorithm (VKOGA) [58,59] yields a regression model of the form

$$\tilde{\boldsymbol{f}}(\boldsymbol{x}) = \sum_{i=1}^{n_{\text{VKOGA}}} \boldsymbol{\alpha}_i K(\boldsymbol{x}_i, \boldsymbol{x}), \tag{4.9}$$

where $K : \mathbb{R}^{N_x \times N_x} \to \mathbb{R}$ denotes a kernel function, $\bar{\boldsymbol{x}}^i \in \mathbb{R}^{N_x}$, $i = 1, \dots, n_{\text{VKOGA}}$ denotes the kernel centers and $\boldsymbol{\alpha}_i \in \mathbb{R}^{N_x}$, $i = 1, \dots, n_{\text{VKOGA}}$ denote vector-valued basis functions.

VKOGA first computes kernel functions $K(\bar{\boldsymbol{x}}^i, \cdot)$ by a greedy algorithm. The greedy algorithm determines kernel centers from $\Omega = \{\bar{\boldsymbol{x}}^1, \ldots, \bar{\boldsymbol{x}}^{n_{\text{train}}}\}$. Initially, let $\Omega_0 = \emptyset$. At stage m, choose

$$\boldsymbol{x}_m := \underset{\boldsymbol{x} \in \Omega \setminus \Omega_{m-1}}{\text{argmax}} \; |\langle \boldsymbol{f}, \boldsymbol{\phi}_x^{m-1} \rangle|,$$

where $\boldsymbol{\phi}_x^{m-1}$ denotes the orthogonal remainer of $K(\boldsymbol{x}, \cdot)$ with respect to the reproducing kernel Hilbert space spanned by $\{K(\boldsymbol{x}_1, \cdot), \ldots, K(\boldsymbol{x}_{m-1}, \cdot)\}$. Then $\Omega_m = \Omega_{m-1} \cup \{\boldsymbol{x}_m\}$. After the kernel centers have been computed, the basis functions $\boldsymbol{\alpha}_i$, $i = 1, \ldots, n_{\text{VKOGA}}$ are determined by a least-squares approximation to best fit training data.

In the numerical experiments, we apply Gaussian RBF kernel $K(\bar{\boldsymbol{x}}^i, \bar{\boldsymbol{x}}^j) = \exp(-\gamma \|\bar{\boldsymbol{x}}^i - \bar{\boldsymbol{x}}^j\|^2)$. The hyper-parameter we consider for this method corresponds to the number of kernel functions n_{VKOGA}.

4.3.6. Sparse identification of nonlinear dynamics (SINDy)

The sparse identification of nonlinear dynamics (SINDy) method [6] is an application of linear regression to learning dynamics. Although SINDy was originally devised for continuous-time dynamics, it can be easily extended to the discrete-time case. In this context, it constructs a model

$$\tilde{f}(\boldsymbol{x}; \boldsymbol{\theta}) = \sum_{i=1}^{n_{\text{SINDy}}} g_i(\boldsymbol{x}) \theta_i \tag{4.10}$$

where $g_i : \mathbb{R}^{N_x} \to \mathbb{R}$, $i = 1, \ldots, n_{\text{SINDy}}$ denote a "library" of prescribed basis functions. The least absolute shrinkage and selection operator (LASSO) [46] is used to determine a sparse set of coefficients $(\theta_1, \ldots, \theta_{n_{\text{SINDy}}})$.

Hyperparameters in this approach consist of the selected basis functions. For simplicity, we consider only linear and quadratic functions in the library, i.e., $g_i \in \{x_1, \ldots, x_{N_x}, x_1 x_1, x_1 x_2, \ldots, x_{N_x} x_{N_x}\}$.

4.3.7. Dynamic mode decomposition (DMD)

Dynamic mode decomposition (DMD) [41] computes the approximated velocity $\tilde{\boldsymbol{f}}$ as the linear operator

$$\tilde{\boldsymbol{f}} : \boldsymbol{x} \mapsto [\boldsymbol{x}^{n_{\text{reg},2}} \; \cdots \; \boldsymbol{x}^{n_{\text{reg},n_{\text{train}}+1}}][\boldsymbol{x}^{n_{\text{reg},1}} \; \cdots \; \boldsymbol{x}^{n_{\text{reg},n_{\text{train}}}}]^+ \boldsymbol{x} \tag{4.11}$$

where the superscript $+$ denotes the Moore–Penrose pseudoinverse. Critically, note that this approach ensures that $\boldsymbol{x}^{n+1} = \tilde{\boldsymbol{f}}(\boldsymbol{x}^n)$, $n \in \mathbb{T}_{\text{reg}}$ if $N_x \geq n_{\text{train}}$.

4.4. Boundedness

This section provides analysis related to the stability of the approximated discrete-time dynamical system (4.2) when the different regression methods described in Section 4.3 are employed to generate the approximated velocity $\tilde{\boldsymbol{f}}$.

First, we note that a function $f : X \to \mathbb{R}^{N_x}$ is *bounded* if the set of its values is bounded. In other words, there exists a real number M such that $\|f(x)\| \leq M$ for all $x \in X$. An important special case is a bounded sequence, where X is taken to be the set $\mathbb{N}$ of natural numbers. Thus a sequence

$$(\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots)$$

is bounded if there exists a real number M such that $\|\boldsymbol{x}(t)\| \leq M$ for every natural number t. With $\tilde{\boldsymbol{x}}^{n+1} = \tilde{\boldsymbol{f}}(\tilde{\boldsymbol{x}}^n)$, boundedness of $\tilde{\boldsymbol{f}} : \mathbb{R}^{N_x} \to \mathbb{R}^{N_x}$ yields boundedness of the sequence $(\tilde{\boldsymbol{x}}^0, \tilde{\boldsymbol{x}}^1, \tilde{\boldsymbol{x}}^2, \ldots)$.

Lemma 4.1. *Let $\mathbb{I}$ be an index set. With $\rho_i : \mathbb{R}^{N_x} \to \mathbb{R}$, $\{\rho_i\}_{i \in \mathbb{I}}$ is a set of scalar basis functions. Suppose $\tilde{\boldsymbol{f}} \in \text{span}(\{\rho_i\}_{i \in \mathbb{I}})$, i.e., there exist $\boldsymbol{\alpha}_i \in \mathbb{R}$, $i = 1, \ldots, n$ (with n finite) such that*

$$\tilde{\boldsymbol{f}}(\boldsymbol{x}) = \sum_{i=1}^{n} \boldsymbol{\alpha}_i \rho_i(\boldsymbol{x}).$$

If each basis function ρ_i is bounded, then $\tilde{\boldsymbol{f}}$ is bounded.

Proof. Let $\alpha = \max_{i=1}^n \|\boldsymbol{\alpha}_i\|$. Since $\rho_i(\cdot)$ is bounded, $\|\rho_i(\boldsymbol{x})\| \leq M_i$ for all $\boldsymbol{x} \in \mathbb{R}^{N_x}$. Let $M = \max_{i=1}^n M_i$. Thus,

$$\|\tilde{\boldsymbol{f}}(\boldsymbol{x})\| = \left\| \sum_{i=1}^{n} \boldsymbol{\alpha}_i \rho_i(\boldsymbol{x}) \right\| \leq \sum_{i=1}^{n} \|\boldsymbol{\alpha}_i \rho_i(\boldsymbol{x})\| \leq \sum_{i=1}^{n} \|\boldsymbol{\alpha}_i\| \cdot \|\rho_i(\boldsymbol{x})\| \leq n\alpha M.$$

The last expression provides an upper bound for $\|\tilde{\boldsymbol{f}}(\boldsymbol{x})\|$. □

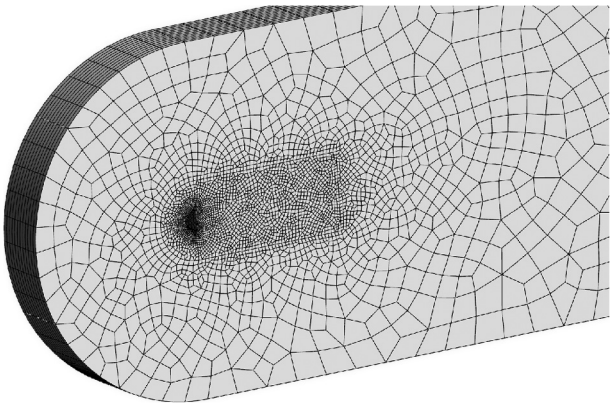


Fig. 1. The half-cylinder mesh characterized by 40 584 elements.

Corollary 4.1.1. *If the approximated velocity $\tilde{\boldsymbol{f}}$ is constructed using SVRrbf (as described in Section 4.3.1) or VKOGA (as described in Section 4.3.5), then the approximated velocity is bounded.*

Proof. Since a Gaussian kernel satisfies $K(\bar{\boldsymbol{x}}^i, \boldsymbol{x}) = \exp(-\gamma \|\bar{\boldsymbol{x}}^i - \boldsymbol{x}\|^2) \leq 1$ for all $\boldsymbol{x} \in \mathbb{R}^{N_x}$, it is bounded. By the previous lemma, with $\rho_i(\boldsymbol{x}) = K(\bar{\boldsymbol{x}}^i, \boldsymbol{x})$, the mapping $\tilde{\boldsymbol{f}}$ is also bounded. □

With the previous lemma, as long as the basis function is bounded, then the function $\tilde{\boldsymbol{f}}$ is also bounded. Thus, k-NN, random forest, and boosting can also yield a bounded sequence $(\boldsymbol{x}(0), \boldsymbol{x}(1), \boldsymbol{x}(2), \ldots)$. In contrast, basis functions associated with DMD are linear, basis functions associated with SINDy are polynomial, and basis functions associated with SVR with polynomial kernels are $K(\bar{\boldsymbol{x}}^i, \bar{\boldsymbol{x}}^j) = (1 + [\bar{\boldsymbol{x}}^i]^T \bar{\boldsymbol{x}}^j)^q$, none of which are bounded.

5. Numerical experiments

To assess the proposed methodology, we consider a large-scale CFD problem with a high-order discontinuous Galerkin discretization.

5.1. Test case and discretization

Our chosen test case is the flow over an extruded half-cylinder at Reynolds number $\mathrm{Re} = 350$ and an effectively incompressible Mach number of $M = 0.2$. This case contains several complex flow features, including separated shear layers, turbulent transition, and a fully turbulent wake. When compared with the circular cylinder, the dynamics of the half-cylinder are noteworthy because the point of separation is essentially fixed. Flow over such a cylinder has been the subject of several numerical and experimental studies [33,27,40].

In the present study, the cylinder is taken to have a diameter D along its major axis. The domain is taken to be $[-9D, 25D]$, $[-9D, 9D]$, and $[0, \pi D]$ in the stream-, cross-, and span-wise directions, respectively. The cylinder is positioned such that the back surface is centered at $(0, 0, 0)$. The stream-wise and cross-wise dimensions are comparable to those used in studies of the circular cylinder [36,60]. The domain is periodic in the span-wise direction. We remark here that, at this Reynolds number, the dynamics are strongly influenced by the span-wise extent of the domain. Indeed, for extrusions less than $\sim \pi D/2$, the wake is coherent. We apply a no-slip adiabatic wall boundary condition at the surface of the cylinder and Riemann invariant boundary conditions at the far-field. To non-dimensionalize the system, we take the cylinder diameter to be $D = 1$, the free-stream density to be one, and the free-stream pressure to be one. The free-stream velocity is thus $v_\infty = \sqrt{\gamma} M \simeq 0.234$, where $\gamma = 1.4$ is the ratio of specific heats.

We mesh the domain using quadratically curved hexahedral elements. In total the mesh has $N_{\mathrm{el}} = 40\,584$ elements. An overview of the mesh can be seen in Fig. 1. We solve the compressible Navier–Stokes equations on this mesh with viscosity fixed such that the Reynolds number based on the diameter of the cylinder is $\mathrm{Re} = 350$. For the spatial discretization, we employ a high-order nodal DG scheme with third-order solution polynomials and Gauss–Legendre solution points. Due to the third-order solution polynomials, the local state within each element contains the density, x-momentum, y-momentum, z-momentum, and energy at 64 points, leading to $N_{\boldsymbol{w},\mathrm{el}} = 5 \times 64 = 320$. Thus, the resulting dimension of the global state is $N_{\boldsymbol{w}} \approx 13 \times 10^6$. This implies that saving a given solution snapshot to disk consumes $\sim$100 MiB of storage using double precision arithmetic. We calculate inviscid fluxes between elements using a Rusanov-type Riemann solver, and we take the Prandtl number to be $\mathrm{Pr} = 0.72$. For time integration, we employ the explicit five-stage fourth-order RK45[2R+] scheme of Carpenter and Kennedy [23], which manages the local temporal error by utilizing a PI type controller to adaptively modify the time step.

We start the simulation cold with a constant initial condition corresponding to the free-stream at $t = 0$ and terminate the simulation at $t = 600$. This corresponds to approximately 71 stream-wise passes over the half-cylinder. Our time grid of interest comprises $t \in \{170.05 + 0.05i\}_{i=1}^{N_t}$ with $N_t = 8\,600$ total time instances. Thus, the desired sequence of states $\{\boldsymbol{w}^n\}_{n=0}^{N_t} \subset \mathbb{R}^{N_{\boldsymbol{w}}}$ corresponds to the CFD solution at these points in time.

5.2. Dimensionality reduction

As described in Section 3, the first stage of our proposed methodology is dimensionality reduction, which proceeds in two steps: local compression using autoencoders, and global compression using PCA.

5.2.1. Local compression using autoencoders

Because convolutional neural networks were originally devised for image data, they require the local degrees of freedom (i.e., the five conserved variables) to be equispaced within each element. However, within the simulation, these variables are defined on the tensor product of a 4-node Gauss–Legendre quadrature rule in order to minimize aliasing errors when projecting the non-linear flux into the polynomial space of the solution. Thus, before applying the autoencoder, we transform the solution by interpolating it from these Gauss–Legendre points to a $4 \times 4 \times 4$ grid of equispaced points. The dimension of the network inputs is therefore $4 \times 4 \times 4 \times 5$, where the first three dimensions correspond to spatial dimensions and the last dimension corresponds to the flow quantities. We do not account for the fact that the mesh elements can vary in size, and hence the spacing between points is not uniform across all training examples. Since solution values at both sets of points define the same polynomial interpolant, the solution inside of each element is unchanged by this transformation.

For the autoencoder, we apply convolutional layers with ResNet skip connections, as they have previously been shown to achieve state-of-the-art performance on image classification and related tasks [14]. Between convolutional layers, we apply batch normalization [19] and ReLU activations [32]. The encoder network has $N_{\text{layers}} = 5$, where each layer contains parameters associated with a set of three-dimensional convolutional filters $\boldsymbol{\Theta}_i \in \mathbb{R}^{3\times3\times3\times f_i}$, and f_i denotes the number of filters contained in layer i. There are 512, 128, 32, 16, and 8 filters in the five layers of the encoder. This architecture is representative of a standard autoencoder [16], where the dimensionality of the transformed input is initially large to allow for adequate feature extraction, and is subsequently decreased gradually through the remaining hidden layers of the encoder. While we found that this autoencoder architecture allowed for a high level of compression and reconstruction accuracy, it is possible that other autoencoder architectures may achieve satisfactory performance. In all layers, the size of the final output dimension is equal to f_i. In the first four layers of the encoder, the convolutions are performed with a stride of one, meaning that the first three dimensions of the layer output $\mathbf{h}_i(\mathbf{y}_{i-1}; \boldsymbol{\Theta}_i)$ are equal in size to the first three dimensions of the layer input $\mathbf{y}_{i-1}$. In the final layer of the encoder, the convolutions are performed with a stride of two, meaning that the first three dimensions of the layer output $\mathbf{h}_5(\mathbf{y}_4; \boldsymbol{\Theta}_5)$ are half the size of the first three dimensions of the layer input $\mathbf{y}_4$. Given all of these transformations, we have $p_0 = 320$, $p_1 = 32\,768$, $p_2 = 8\,192$, $p_3 = 2\,048$, $p_4 = 1\,024$, and $p_5 = 64$.

The output of the final convolutional layer is reshaped into a vector and subsequently mapped to encoding $\hat{\boldsymbol{w}}_i$ with an affine transformation. We set the reduced dimension of the local state vector (i.e., the code dimensionality) to $N_{\hat{\boldsymbol{w}},\text{el}} = 24$. We then use an affine transformation and reshaping operation to create an input to the decoder that is the same size as the output of the encoder. We construct the decoder network to invert all operations performed by the encoder in order to obtain reconstructed solutions; this implies that $\bar{N}_{\text{layers}} = 5$, $\bar{p}_0 = 64$, $\bar{p}_1 = 1\,024$, $\bar{p}_2 = 2\,048$, $\bar{p}_3 = 8\,192$, $\bar{p}_4 = 32\,768$, and $\bar{p}_5 = 320$. Fig. 2 provides an illustration of the network architecture.

The training data for the autoencoder consists of the 90 randomly selected values of the set of time instances $\{86i\}_{i=1}^{100}$ such that $|\mathbb{T}_{\text{autoencoder}}| = 90$; the remaining 10 points of the set comprise validation data that is used to check for overfitting. To solve the minimization problem (3.5), we employ stochastic gradient descent with the Adam optimizer [24]. We apply L_2 regularization to the network weights

$$\Omega(\boldsymbol{\theta}, \bar{\boldsymbol{\theta}}) = \lambda \left(\|\boldsymbol{\theta}\|_2^2 + \|\bar{\boldsymbol{\theta}}\|_2^2 \right), \tag{5.1}$$

where λ is a hyperparameter that controls the level of regularization. When an increase in the validation error is observed across training epochs, the learning rate is cut in half. Training is terminated once the loss on the validation set fails to decrease across three training epochs, which typically occurs after the learning rate has been decayed many times.

5.2.2. Global compression using PCA

We apply PCA for global compression as described in Section 3.2 using training time instances[5] of $\mathbb{T}_{\text{PCA}} = \mathbb{T}_{\text{sample}}$. We decompose the global reduced state vector $\hat{\boldsymbol{w}}$ into $n_{\hat{\mathbf{w}}} = 38$ components, each of which is characterized by 24 codes and 1 068 cells such that $\hat{\mathbf{w}}_i \in \mathbb{R}^{25\,632}$, $i = 1, \ldots, 38$. We truncate the component principal components such that $N_{x,i} = 500$, $i = 1, \ldots, 38$.

[5] Note that PCA is robust with respect to missing data points; thus, we expect the principal components computed from this set to be close to those computed on a smaller training set.

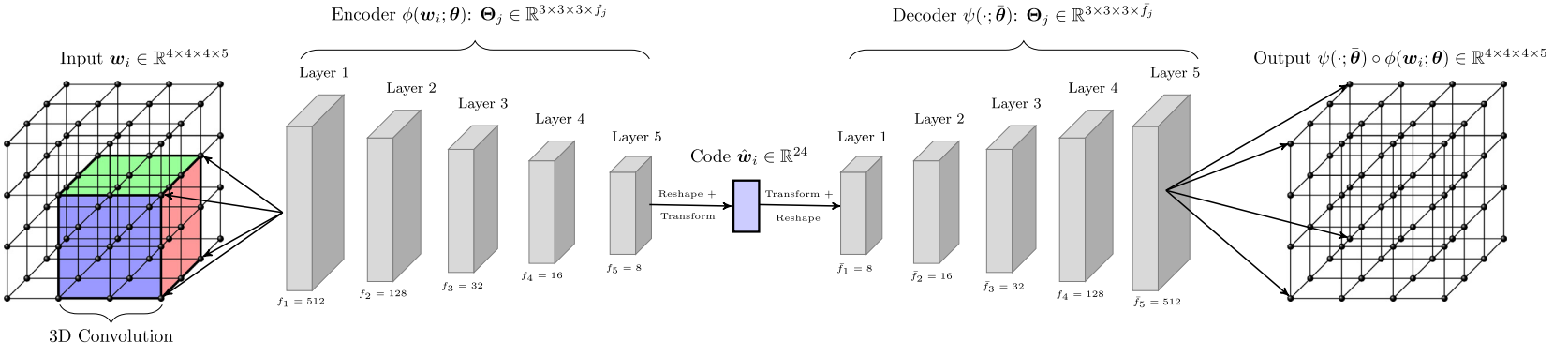


Fig. 2. Illustration of the network architecture. In the encoder, the input is transformed by a series of convolutional layers with a decreasing number of filters at each layer. The code is obtained by reshaping the output of the final encoder layer into a vector and performing an affine transformation. The decoder outputs a reconstruction of the original input w_i by performing the inverse of all operations performed by the encoder.

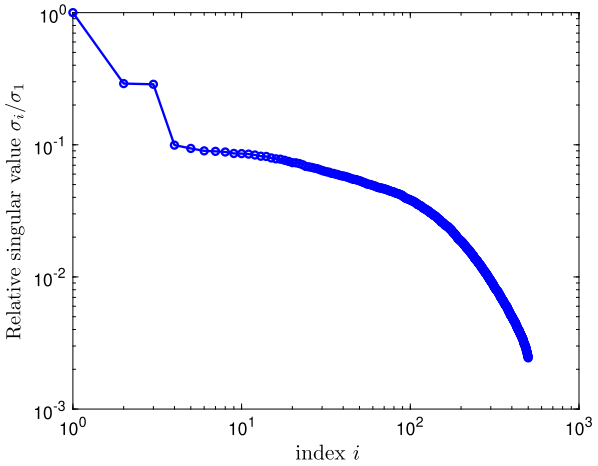


Fig. 3. Decay of the singular values. Numerical computation shows that a basis dimension of $N_x = 100$ captures approximately 95% of the statistical energy in the data, and a basis dimension of $N_x = 500$ captures approximately 98% of the statistical energy.

Fig. 3 reports the decay of the first 500 (of 8 600) singular values (i.e., the first 500 diagonal elements of $\mathbf{\Sigma}$ in Eq. (3.13)). In this case, the first 100 principal components capture approximately 95% of the statistical energy (as measured by the sum of squares of all 8 600 singular values) in the data, while the first 500 principal components capture approximately 98% of the statistical energy. We first set $N_x = 100$.

5.3. Dynamics learning

We now apply the approach described in Section 4 for dynamics learning. We employ a maximum set of training time instances $\mathbb{T}_{\text{reg}}$ with $|\mathbb{T}_{\text{reg}}| = 6\,450$. Before applying regression, we standardize the features by applying an affine transformation (i.e., we subtract the sample mean and divide by the sample standard deviation).

5.3.1. Hyperparameter selection using validation

We now present validation results for the regression methods with tunable hyperparameters, i.e., SVR2, SVR3, SVRrbf, random forest, boosting, k-NN, and VKOGA. We set the validation set to be the 2 150 time instances not included in the maximum training set.

Fig. 4 reports these results, wherein the models are trained using the maximum training set $\mathbb{T}_{\text{reg}}$. Here, the relative mean-squared error (MSE) over a set of time indices $\mathbb{T} \subseteq \{1, \ldots, N_t\}$ is defined as

$$\frac{\sum_{n \in \mathbb{T}} \| \tilde{\boldsymbol{f}}(\boldsymbol{x}^n) - \boldsymbol{x}^{n+1} \|_2}{\sum_{n \in \mathbb{T}} \| \boldsymbol{x}^{n+1} \|_2}. \tag{5.2}$$

Based on these results, we select the following hyperparameters: $\epsilon = 10^{-3}$ for SVR2, SVR3, and SVRrbf; $N_{\text{tree}} = 30$ for random forest; $N_{\text{tree}} = 100$ for boosted decision trees; $k = 6$ for k-nearest neighbors; and 200 kernel functions for VKOGA.

5.3.2. Training and testing error

We now analyze the training and testing errors of the various regression methods, where the hyperparameters have been fixed according to the values selected using validation in Section 5.3.1. Fig. 5 reports these results. Here, three different training sets are considered to construct the regression models: one with the maximum training set $\mathbb{T}_{\text{reg}}$ with $|\mathbb{T}_{\text{reg}}| = 6\,450$ (yellow bars), one with 75% of the maximum training data selected at random (red bars), and one with 50% of the maximum training data selected at random (blue bars). The reported errors correspond to the relative MSE as defined in Eq. (5.2).

First, note that the training and testing errors are extremely similar; this suggests that the training data are indeed representative of the testing data, indicating that we have not overfit our models.

Second, note that even with the smallest training set characterized by only 50% of the maximum number of considered points training set (i.e., 3 225 time instances), the training and test errors are quite similar to those achieved with the maximum training set. Indeed, the only regression methods where doubling the training set yields improvements are random forest, and k-NN. Even in these cases, the additional training data provide modest gains. In the case of SVR2, including additional training data actually *degrades* both the training and testing errors. This likely occurs because the SVR2 model is trained using the hinge loss (see Eq. (4.6)), which is different from the relative MSE loss reported here. The hinge loss does not penalize small errors, which do influence the relative MSE.

Third, note that the smallest errors (relative MSE on the test set $< 1 \times 10^{-3}$) are achieved by SVRrbf, VKOGA, and SINDy. We expect these techniques to perform best when attempting to recover the desired sequence of states by simulating the

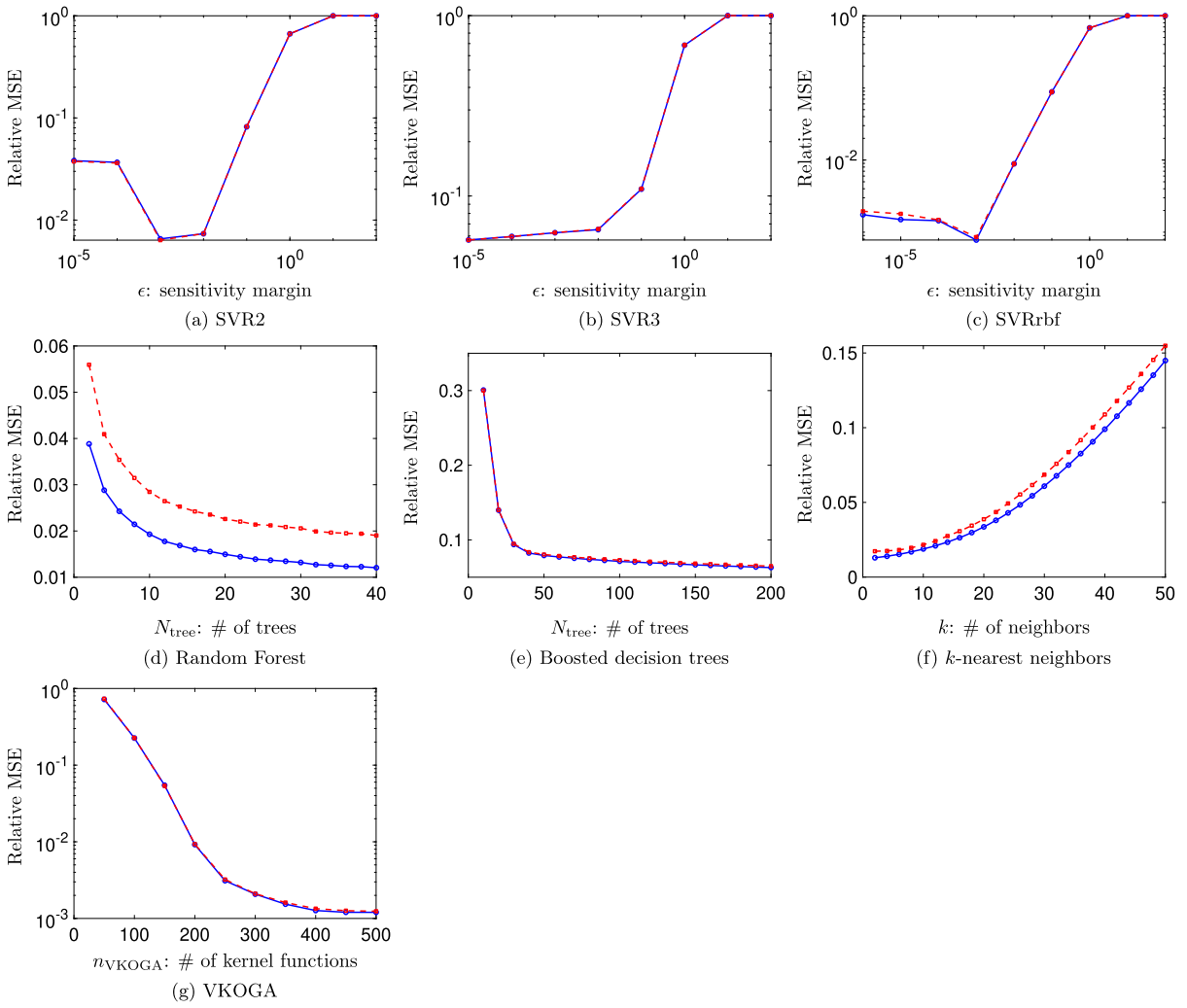


Fig. 4. Hyperparameter selection for different regression methods. Reported errors correspond to the relative MSE values on the training set (blue) and the validation set (red). The training set corresponds to the maximum training set $\mathbb{T}_{\text{reg}}$, and the validation set corresponds to the remaining 2 150 time instances. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

approximated discrete-time dynamics (4.2). However, as discussed in Section 4.4, only VKOGA and SVRrbf are bounded; SINDy admits unbounded dynamics.

5.3.3. Approximated dynamics

We now turn to the final stage of the method: computing the sequence of low-dimensional approximated states $\{\tilde{\boldsymbol{w}}^n\}_{n=0}^{N_t}$ according to approximated discrete-time dynamics (4.2), and subsequently computing the approximation to the desired sequence of states, i.e., $\{\tilde{\boldsymbol{w}}^n\}_{n=0}^{N_t}$, where $\tilde{\boldsymbol{w}}^0 = \boldsymbol{w}^0$ and $\tilde{\boldsymbol{w}}^n = \mathbf{p}(\tilde{\boldsymbol{x}}^n)$, $n = 1, \ldots, N_t$.

Fig. 6 reports these results. For each regression model (constructed with the maximum training set $\mathbb{T}_{\text{reg}}$), this figure reports the time-instantaneous relative error $\|\tilde{\boldsymbol{x}}^n - \boldsymbol{x}^n\|_2 / \|\boldsymbol{x}^n\|_2$ over all time instances, i.e., for $n = 0, \ldots, N_t$. These results show that both VKOGA and SVRrbf yield accurate responses over time, as the time-instantaneous relative error is less than 5% and 3% for these models, respectively, at all time instances. These results are sensible, as their training and testing errors were also small, as reported in Fig. 5. Somewhat surprisingly, note that SINDy—despite yielding small training and testing errors in Fig. 5—produces an unstable response when integrated into the approximated discrete-time dynamics (4.2). This can be explained from the boundedness analysis in Section 4.4: the polynomial basis functions employed with the SINDy method are unbounded. Thus, this approach does not ensure a bounded sequence of approximated states, and—in this case—produces unstable discrete-time dynamics as a result. This highlights that the stability properties of the ultimate dynamical system should play an important role in deciding on the appropriate regression model; low training and testing errors are not sufficient for an accurate model in this context.

We emphasize the promise of these results: the original sequence of data—wherein each vector has dimension $N_{\boldsymbol{w}} \approx 13 \times 10^6$—has been accurately approximated as dynamical system of dimension $N_x = 100$, which corresponds to a compression

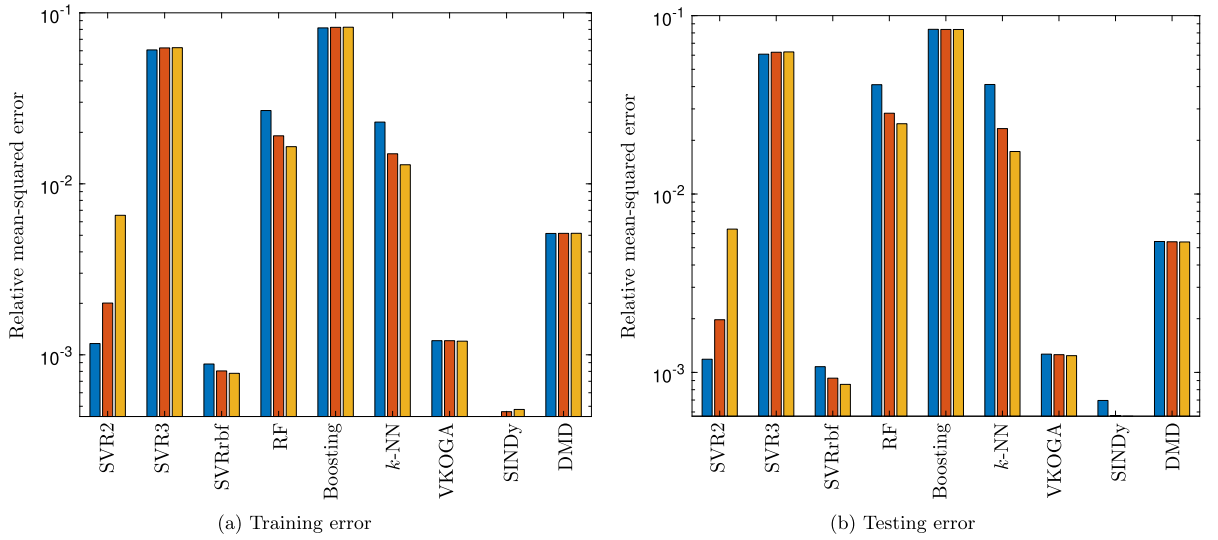


Fig. 5. Training and testing relative MSE errors for all regression methods. We consider models constructed using three training sets: one with the maximum training set $\mathbb{T}_{\text{reg}}$ with $|\mathbb{T}_{\text{reg}}| = 6450$ (yellow bars), one with 75% of the maximum training data selected at random (red bars), and one with 50% of the maximum training data selected at random (blue bars).

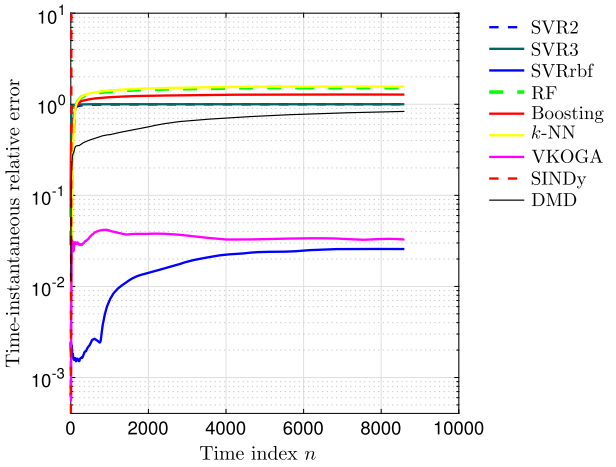


Fig. 6. The time-instantaneous relative error $\|\tilde{\boldsymbol{x}}^n - \boldsymbol{x}^n\|_2 / \|\boldsymbol{x}^n\|_2$, $n = 1, \ldots, N_t$ over time for each considered regression model. Regression models are trained using the maximum training set $\mathbb{T}_{\text{reg}}$. The low-dimensional state dimension is $N_x = 100$. Note that both SVRrbf and VKOGA yield accurate responses, which are consistent with their small training and testing errors. In contrast, SINDy produces an unstable response despite its small training and testing errors; this is due to the fact that it yields an unbounded approximated velocity.

ratio of 130 000 : 1. This implies that—with access only to the initial state $\boldsymbol{w}^n$, the proposed method can recover the desired sequence of states to within 5% accuracy in the low-dimensional state with knowledge of only the restriction operator $\mathbf{r}$ and approximated velocity $\tilde{\boldsymbol{f}}$.

We now consider increasing the dimension of the low-dimensional state to gauge the effect of accuracy on this parameter. To achieve this, after computing the principal components as described in Section 5.2.2, we now preserve the first $N_x = 500$ principal components, yielding a compression ratio of 26 000 : 1 wherein we have captured 98% of the statistical energy. Rather than repeat the experiment for all considered regression methods, we now only consider the VKOGA method, as it (along with SVRrbf) yielded the best results in the case of $N_x = 100$. Fig. 7 reports these results; comparing this figure with Fig. 6 shows that increasing the dimension of the low-dimensional state from $N_x = 100$ to $N_x = 500$ (and thus increasing the captured statistical energy in the global PCA step from 95% to 98%) has reduced the relative error by nearly an order of magnitude. The response associated with $N_x = 500$ now exhibits time-instantaneous relative errors below 0.7% over all time.

The above results present the prediction error over time in the reduced state. Of greater interest is the prediction error relative to the global state $\boldsymbol{w}^n$ once the reduced state values are passed through the prolongation operator $\mathbf{p}$. The prediction error across all time instances for the approximated solutions $\tilde{\boldsymbol{w}}^n$ can be found in Fig. 8. To put these results in context, we compare against the error obtained by passing each global state through the encoder and decoder without performing the

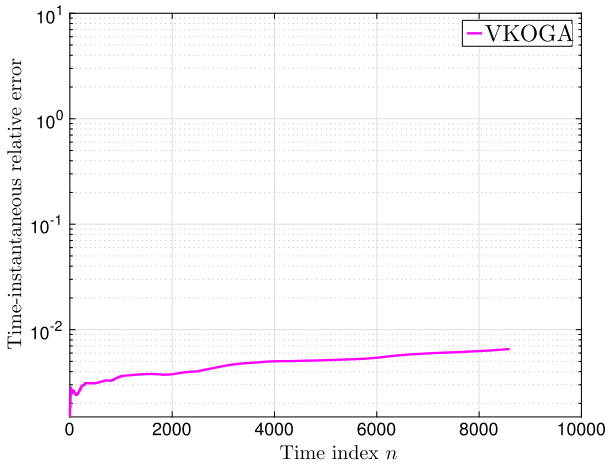


Fig. 7. The time-instantaneous relative error $\|\tilde{\boldsymbol{x}}^n - \boldsymbol{x}^n\|_2/\|\boldsymbol{x}^n\|_2$, $n = 1, \ldots, N_t$ for VKOGA trained using the maximum training set $\mathbb{T}_{\text{reg}}$. We now employ a low-dimensional state dimension of $N_x = 500$. Note that the errors have decreased by roughly one order of magnitude relative to the case of $N_x = 100$ (compare with Fig. 6).

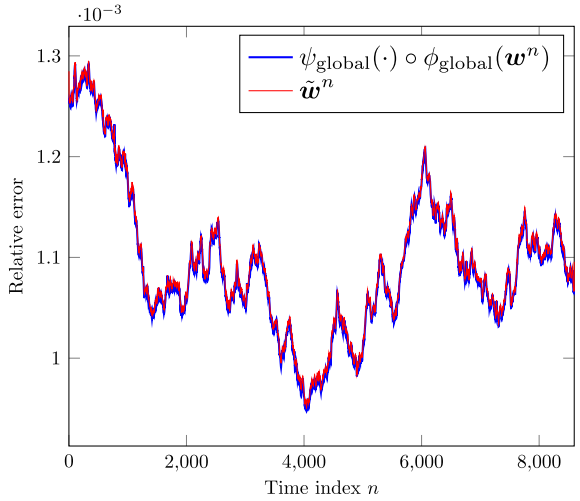


Fig. 8. The time-instantaneous relative error for the autoencoder alone $\|\boldsymbol{w}^n - \psi_{\text{global}}(\cdot) \circ \phi_{\text{global}}(\boldsymbol{w}^n)\|_1/\|\boldsymbol{w}^n\|_1$, $n = 1, \ldots, N_t$, as well as for the approximated solutions $\|\boldsymbol{w}^n - \tilde{\boldsymbol{w}}^n\|_1/\|\boldsymbol{w}^n\|_1$, $n = 1, \ldots, N_t$. The L_1 norm is used because numerical roundoff errors cause spikes in time instances when $\|\boldsymbol{w}^n\|_2$ is small.

PCA and dynamics-learning procedures. We note that the error introduced by compressing and reconstructing the global states through the autoencoder introduces very little error, with a relative error of less than 0.13% across all time instances. More critically, we can see that the relative error for the approximated states is nearly identical to the error from the autoencoder reconstructions, meaning that very little error is introduced by the PCA and dynamics-learning steps.

Beyond solely considering the global error in reconstructions, we are also interested in whether local properties of the fluid flow are preserved during the various stages of dimensionality reduction, dynamics propagation, and reconstruction. One manner of determining whether flow properties are preserved is to consider the aggregate lift and drag forces that act on the surface of the half-cylinder over time. By extracting pressure values on the surface of the cylinder, we can resolve the distribution of forces acting on the cylinder in the downstream and cross-stream directions. Integrating over the surface of the half-cylinder, we obtain the lift and drag coefficients for the cylinder. The left column of Fig. 9 reports these lift and drag forces plotted across all time instances for the CFD solutions $\boldsymbol{w}^n$, autoencoder reconstructions $\psi_{\text{global}}(\cdot) \circ \phi_{\text{global}}(\boldsymbol{w}^n)$, and approximated solutions $\tilde{\boldsymbol{w}}^n$ for $n = 1, \ldots, N_t$. The right column of Fig. 9 reports the error in lift and drag forces associated with the reconstructed solutions generated by the autoencoder and the approximated solutions. The absolute error is provided for lift values rather than the relative error due to some lift values being close to zero. From these results, we first note that the error in lift and drag introduced by encoding and decoding solutions with the autoencoder is quite low: the absolute error is less than 1×10^{-3} across all time steps for lift values and the relative error is less than 0.1% for drag values. Furthermore, we note that the principal component analysis and dynamics-learning procedures introduce very little additional error in these quantities beyond what is introduced by the autoencoder.

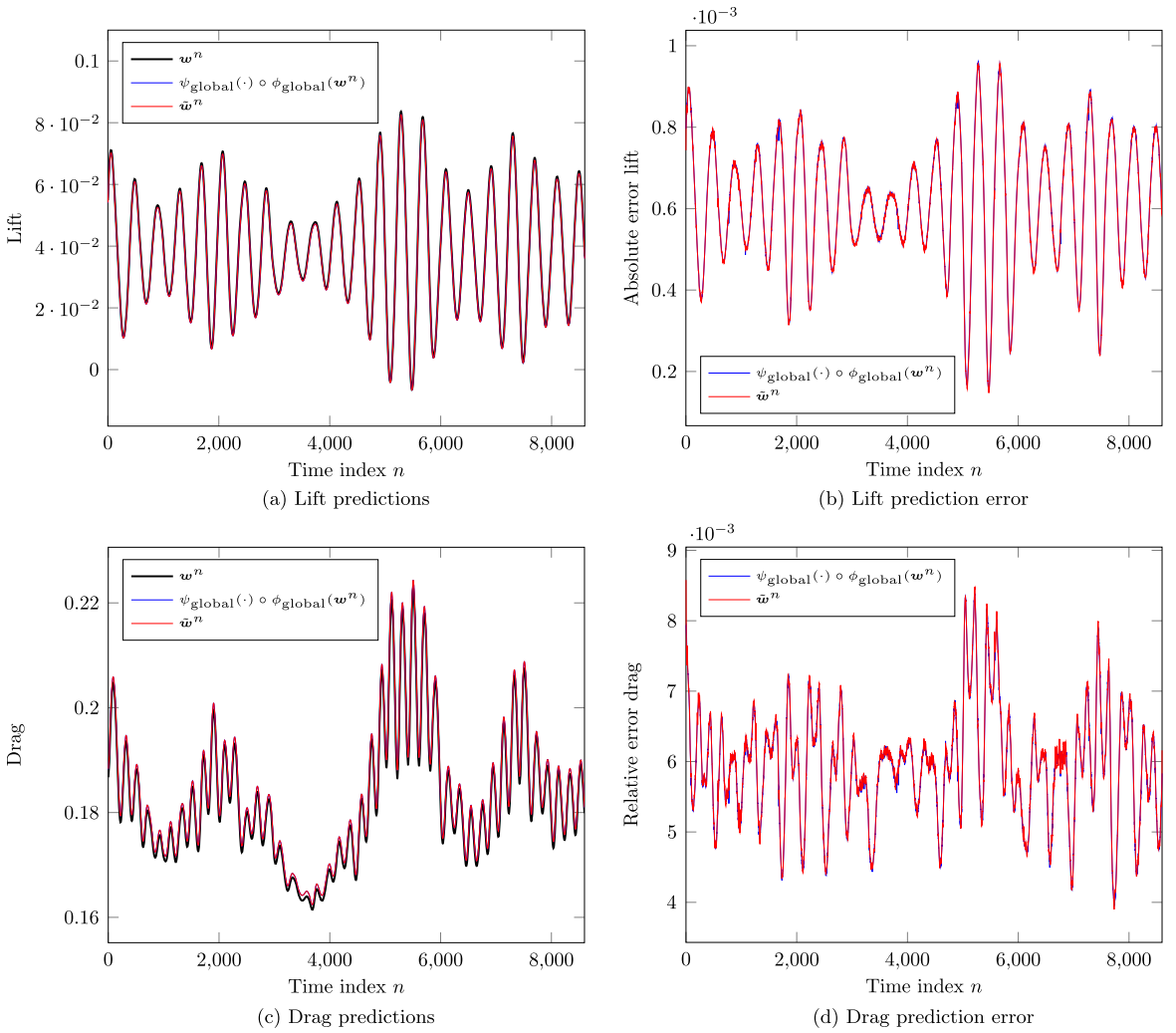


Fig. 9. Lift and drag forces on the surface of the cylinder across all time instances. The left column shows lift and drag predictions, while the right column presents the error in lift and drag predictions. Absolute error is used rather than relative error to avoid numerical issues when lift values are close to zero.

We now provide several visualizations that lend qualitative insight into the performance of the method. Fig. 10 compares instantaneous iso-surfaces of Q-criterion colored by velocity magnitude for the CFD solution, the autoencoder reconstruction, and the approximated solution. The figure shows that for all considered cases, the autoencoder reconstruction retains all of the key vortical structures of the CFD solution, albeit with some noise in the iso-surfaces. We also see that there is no distinguishable difference between the autoencoder reconstruction and the approximated solution. This further supports our assertion that the principal component analysis and dynamics-learning procedures introduce very little additional error.

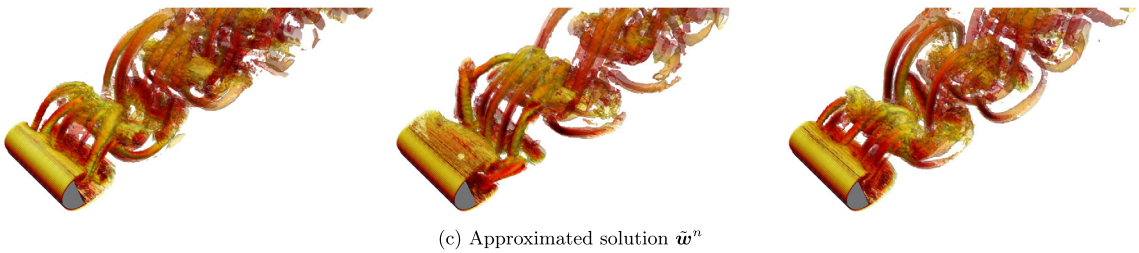
6. Conclusions

This article presented a novel methodology for recovering missing CFD data given that the solution has been saved to disk at a relatively small number of time steps.

The first stage of the methodology—hierarchical dimensionality reduction—comprises a novel two-step technique tailored for high-order discretizations. In the first step, we apply autoencoders for local compression; this step computes a low-dimensional, nonlinear embedding of the degrees of freedom within each element of the high-order discretization. In the numerical experiments, we employed a convolutional neural network for this purpose. Results showed that the autoencoder reduced the dimensionality of the element-local state from $N_{\boldsymbol{w},\text{el}} = 320$ to $N_{\hat{\boldsymbol{w}},\text{el}} = 24$ without incurring significant errors (see Figs. 9 and 10). In the second dimensionality-reduction step, we apply principal component analysis to compress the global vector of encodings. Results showed that this second step was able to reduce the number of global degrees of freedom from $N_{\hat{\boldsymbol{w}},\text{el}} N_{\text{el}} = 24 \times 40\,584 \sim 10^6$ to only 500, constituting a compression ratio of $26\,000 : 1$, while retaining very high levels of accuracy.

(a) CFD solution $\boldsymbol{w}^n$

(b) Autoencoder reconstruction $\psi_{\text{global}}(\cdot) \circ \phi_{\text{global}}(\boldsymbol{w}^n)$



(c) Approximated solution $\tilde{\boldsymbol{w}}^n$

Fig. 10. Instantaneous iso-surfaces of Q-criterion colored by velocity magnitude at $n = 1\,000, 4\,000$ and $7\,000$.

The second stage of the methodology—dynamics learning—applied regression methods from machine learning to learn the discrete-time dynamics of the low-dimensional state. We considered a wide range of regression methods for this purpose, and found that support vector regression with a radial-basis-function kernel (SVRrbf) and the vectorial kernel orthogonal greedy algorithm (VKOGA) yielded the best performance (see Fig. 6). Although the sparse identification of nonlinear dynamics (SINDy) yielded low regression errors on an independent test set (see Fig. 5), it did not produce an accurate solution approximation due to its unboundedness; indeed, the resulting trajectory was unstable (see Fig. 6).

Ultimately, applying the proposed methodology with VKOGA and a low-dimensional state dimension of 500 satisfied the objective of this work, as it enabled the original CFD data to be reconstructed with extremely high levels of accuracy, yielding low errors in the state, lift, and drag, as well as solution fields that match well visually (see Figs. 8, 9, and 10).

Future work involves introducing parameterization into the problem setting such that the reconstruction task can be performed across multiple CFD simulations characterized by different parameters, e.g., different boundary conditions, geometric shapes, and operating conditions.

Acknowledgements

K. Carlberg's and L. Peng's research was sponsored by Sandia's Advanced Simulation and Computing (ASC) Verification and Validation (V&V) Project/Task #65755/002.01.19. F. D. Witherden's and A. Jameson's research was supported by the Air Force Office of Scientific Research via grant FA9550-14-1-0186. J. Morton's research was supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-114747.

References

[1] N.S. Altman, An introduction to kernel and nearest-neighbor nonparametric regression, Am. Stat. 46 (1992) 175–185.
[2] J.-M. Beckers, M. Rixen, EOF calculations and data filling from incomplete oceanographic datasets, J. Atmos. Ocean. Technol. 20 (2003) 1839–1856.
[3] W. Böhmer, J.T. Springenberg, J. Boedecker, M. Riedmiller, K. Obermayer, Autonomous learning of state representations for control: an emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations, KI-Künstl. Intell. 29 (2015) 353–362.
[4] L. Bottou, F.E. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, SIAM Rev. 60 (2018) 223–311.
[5] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32.
[6] S.L. Brunton, J.L. Proctor, J.N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proc. Natl. Acad. Sci. (2016) 201517384.

[7] T. Bui-Thanh, M. Damodaran, K. Willcox, Aerodynamic data reconstruction and inverse design using proper orthogonal decomposition, AIAA J. 42 (2004) 1505–1516.
[8] K. Carlberg, M. Barone, H. Antil, Galerkin v. least-squares Petrov–Galerkin projection in nonlinear model reduction, J. Comput. Phys. 330 (2017) 693–734.
[9] C. Chung Chang, C. Jen Lin, LIBSVM: a library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (2001).
[10] B. Cockburn, C.-W. Shu, The Runge-Kutta local projection P1-discontinuous-Galerkin finite element method for scalar conservation laws, ESAIM: Math. Model. Numer. Anal. 25 (1991) 337–361.
[11] R. Goroshin, M.F. Mathieu, Y. LeCun, Learning to linearize under uncertainty, in: C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems 28, Curran Associates, Inc., 2015, pp. 1234–1242.
[12] H. Gunes, S. Sirisup, G.E. Karniadakis, Gappy data: to krig or not to krig?, J. Comput. Phys. 212 (2006) 358–382.
[13] T. Hastie, R. Tibshirani, J.H. Friedman, The Elements of Statistical Learning, Springer, 2009.
[14] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, 2016.
[15] J.S. Hesthaven, T. Warburton, Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications, vol. 54, Springer Verlag, New York, 2008.
[16] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (2006) 504–507.
[17] P. Holmes, J. Lumley, G. Berkooz, Turbulence, Coherent Structures, Dynamical Systems and Symmetry, Cambridge University Press, 1996.
[18] H.T. Huynh, A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods, in: 18th AIAA Computational Fluid Dynamics Conference, 2007, p. 4079.
[19] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, arXiv preprint, arXiv:1502.03167, 2015.
[20] H. Kang, D. Lee, D. Lee, A study on CFD data compression using hybrid supercompact wavelets, KSME Int. J. 17 (2003) 1784–1792.
[21] M. Karl, M. Soelch, J. Bayer, P. van der Smagt, Deep variational Bayes filters: unsupervised learning of state space models from raw data, arXiv preprint, arXiv:1605.06432, 2016.
[22] Y. Kawahara, Dynamic mode decomposition with reproducing kernels for Koopman spectral analysis, in: Advances in Neural Information Processing Systems, 2016, pp. 911–919.
[23] C.A. Kennedy, M.H. Carpenter, R.M. Lewis, Low-storage, explicit Runge–Kutta schemes for the compressible Navier–Stokes equations, Appl. Numer. Math. 35 (1999) 177–219.
[24] D. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv preprint, arXiv:1412.6980, 2014.
[25] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, arXiv preprint, arXiv:1312.6114, 2013.
[26] D.A. Kopriva, A staggered-grid multidomain spectral method for the compressible Navier–Stokes equations, J. Comput. Phys. 143 (1998) 125–158.
[27] S. Kumarasamy, J.B. Barlow, Computation of unsteady flow over a half-cylinder close to a moving wall, J. Wind Eng. Ind. Aerodyn. 69 (1997) 239–248.
[28] T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, D. Filliat, State representation learning for control: an overview, Neural Netw. (2018).
[29] Y. Liu, M. Vinokur, Z. Wang, Spectral difference method for unstructured grids I: basic formulation, J. Comput. Phys. 216 (2006) 780–801.
[30] B. Lusch, J.N. Kutz, S.L. Brunton, Deep learning for universal linear embeddings of nonlinear dynamics, arXiv preprint, arXiv:1712.09707, 2017.
[31] J. Morton, F.D. Witherden, A. Jameson, M.J. Kochenderfer, Deep dynamical modeling and control of unsteady fluid flows, 2018.
[32] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proceedings of the 27th International Conference on Machine Learning, ICML-10, 2010, pp. 807–814.
[33] Y. Nakamura, Vortex shedding from bluff bodies with splitter plates, J. Fluids Struct. 10 (1996) 147–158.
[34] S.E. Otto, C.W. Rowley, Linearly-recurrent autoencoder networks for learning dynamics, arXiv preprint, arXiv:1712.01378, 2017.
[35] J. Park, F. Witherden, P. Vincent, High-Order Implicit Large-Eddy Simulations of Flow Over a NACA0021 Aerofoil, AIAA Journal, 2017, pp. 2186–2197.
[36] P. Parnaudeau, J. Carlier, D. Heitz, E. Lamballais, Experimental and numerical studies of the flow over a circular cylinder at Reynolds number 3900, Phys. Fluids 20 (2008) 085101.
[37] S.C. Puligilla, B. Jayaraman, Deep multilayer convolution frameworks for data-driven learning of fluid flow dynamics, in: 2018 Fluid Dynamics Conference, 2018, p. 3091.
[38] W.H. Reed, T.R. Hill, Triangular Mesh Methods for the Neutron Transport Equation, Technical Report LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.
[39] R. Sakai, D. Sasaki, K. Nakahashi, Parallel implementation of large-scale CFD data compression toward aeroacoustic analysis, Comput. Fluids 80 (2013) 116–127.
[40] A. Santa Cruz, L. David, J. Pecheux, A. Texier, Characterization by proper-orthogonal-decomposition of the passive controlled wake flow downstream of a half cylinder, Exp. Fluids 39 (2005) 730–742.
[41] P.J. Schmid, Dynamic mode decomposition of numerical and experimental data, J. Fluid Mech. 656 (2010) 5–28.
[42] A.J. Smola, B. Schölkopf, A tutorial on support vector regression, Stat. Comput. 14 (2004) 199–222.
[43] Y. Sun, Z.J. Wang, Y. Liu, High-order multidomain spectral difference method for the Navier–Stokes equations on unstructured hexahedral grids, Commun. Comput. Phys. 2 (2007) 310–333.
[44] N. Takeishi, Y. Kawahara, T. Yairi, Learning Koopman invariant subspaces for dynamic mode decomposition, in: Advances in Neural Information Processing Systems, 2017, pp. 1130–1140.
[45] L. Theis, W. Shi, A. Cunningham, F. Huszár, Lossy image compression with compressive autoencoders, in: International Conference on Learning Representations, 2017.
[46] R. Tibshirani, Regression shrinkage and selection via the lasso, J. R. Stat. Soc., Ser. B, Methodol. (1996) 267–288.
[47] A. Trott, R. Moorhead, J. McGinley, Wavelets applied to loseless compression and progressive transmission of floating point data in 3-d curvilinear grids, in: Proceedings of the 7th Conference on Visualization'96, IEEE Computer Society Press, 1996, p. 385.
[48] D. Venturi, G.E. Karniadakis, Gappy data and reconstruction procedures for flow past a cylinder, J. Fluid Mech. 519 (2004) 315–336.
[49] B.C. Vermeire, F.D. Witherden, P.E. Vincent, On the utility of GPU accelerated high-order methods for unsteady flow simulations: a comparison with industry-standard tools, J. Comput. Phys. 334 (2017) 497–521.
[50] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion 11 (2010) 3371–3408.
[51] P. Vincent, F. Witherden, B. Vermeire, J.S. Park, A. Iyer, Towards green aviation with Python at petascale, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE Press, 2016, p. 1.
[52] P.E. Vincent, A. Jameson, Facilitating the adoption of unstructured high-order methods amongst a wider community of fluid dynamicists, Math. Model. Nat. Phenom. 6 (2011) 97–140.
[53] M. Watter, J. Springenberg, J. Boedecker, M. Riedmiller, Embed to control: a locally linear latent dynamics model for control from raw images, in: Advances in Neural Information Processing Systems, 2015, pp. 2746–2754.
[54] S. Wiewel, M. Becher, N. Thuerey, Latent-space physics: towards learning the temporal evolution of fluid flow, arXiv preprint, arXiv:1802.10123, 2018.
[55] K. Willcox, Unsteady flow sensing and estimation via the gappy proper orthogonal decomposition, Comput. Fluids 35 (2006) 208–226.
[56] M.O. Williams, I.G. Kevrekidis, C.W. Rowley, A data–driven approximation of the Koopman operator: extending dynamic mode decomposition, J. Nonlinear Sci. 25 (2015) 1307–1346.

[57] M.O. Williams, C.W. Rowley, I.G. Kevrekidis, A kernel-based approach to data-driven Koopman spectral analysis, arXiv preprint, arXiv:1411.2260, 2014.
[58] D. Wirtz, B. Haasdonk, A vectorial kernel orthogonal greedy algorithm, Dolomites Res. Notes Approx. 6 (2013).
[59] D. Wirtz, N. Karajan, B. Haasdonk, Surrogate modeling of multiscale models using kernel methods, Int. J. Numer. Methods Eng. 101 (2015) 1–28.
[60] F.D. Witherden, B.C. Vermeire, P.E. Vincent, Heterogeneous computing on mixed unstructured grids with PyFR, Comput. Fluids 120 (2015) 173–186.
[61] H. Zou, T. Hastie, Regularization and variable selection via the elastic net, J. R. Stat. Soc., Ser. B, Stat. Methodol. 67 (2005) 301–320.