SAND2019-12375J

Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders

Kookjin Lee[a,*], Kevin T. Carlberg[a]

[a]*Sandia National Laboratories*

Abstract

Nearly all model-reduction techniques project the governing equations onto a linear subspace of the original state space. Such subspaces are typically computed using methods such as balanced truncation, rational interpolation, the reduced-basis method, and (balanced) proper orthogonal decomposition (POD). Unfortunately, restricting the state to evolve in a linear subspace imposes a fundamental limitation to the accuracy of the resulting reduced-order model (ROM). In particular, linear-subspace ROMs can be expected to produce low-dimensional models with high accuracy only if the problem admits a fast decaying Kolmogorov n-width (e.g., diffusion-dominated problems). Unfortunately, many problems of interest exhibit a slowly decaying Kolmogorov n-width (e.g., advection-dominated problems). To address this, we propose a novel framework for projecting dynamical systems onto nonlinear manifolds using minimum-residual formulations at the time-continuous and time-discrete levels; the former leads to *manifold Galerkin* projection, while the latter leads to *manifold least-squares Petrov–Galerkin (LSPG)* projection. We perform analyses that provide insight into the relationship between these proposed approaches and classical linear-subspace reduced-order models; we also derive *a posteriori* discrete-time error bounds for the proposed approaches. In addition, we propose a computationally practical approach for computing the nonlinear manifold, which is based on convolutional autoencoders from deep learning. Finally, we demonstrate the ability of the method to significantly outperform even the optimal linear-subspace ROM on benchmark advection-dominated problems, thereby demonstrating the method's ability to overcome the intrinsic n-width limitations of linear subspaces.

Keywords: model reduction, deep learning, autoencoders, machine learning, nonlinear manifolds, optimal projection

1. Introduction

Physics-based modeling and simulation has become indispensable across many applications in engineering and science, ranging from aircraft design to monitoring national critical infrastructure. However, as simulation is playing an increasingly important role in scientific discovery, decision making, and design, greater demands are being placed on model fidelity. Achieving high fidelity often necessitates including fine spatiotemporal resolution in computational models of the system of interest; this can lead to very large-scale models whose simulations consume months on thousands of computing cores. This computational burden precludes the integration of such high-fidelity models in important scenarios that are *real time* or *many query* in nature, as these scenarios require the (parameterized) computational model to be simulated very rapidly (e.g., model predictive control) or thousands of times (e.g., uncertainty propagation).

Projection-based reduced-order models (ROMs) provide one approach for overcoming this computational burden. These techniques comprise two stages: an *offline* stage and an *online* stage. During the offline stage,

*7011 East Ave, MS 9159, Livermore, CA 94550.
Email addresses: koolee@sandia.gov (Kookjin Lee), ktcarlb@sandia.gov (Kevin T. Carlberg)
URL: sandia.gov/~ktcarlb (Kevin T. Carlberg)

these methods perform computationally expensive training tasks (e.g., simulating the high-fidelity model at several points in the parameter space) to compute a representative low-dimensional 'trial' subspace for the system state. Next, during the inexpensive online stage, these methods rapidly compute approximate solutions for different points in the parameter space via projection: they compute solutions in the low-dimensional trial subspace by enforcing the high-fidelity-model residual to be orthogonal to a low-dimensional test subspace of the same dimension.

As suggested above, nearly all projection-based model-reduction approaches employ *linear trial subspaces*. This includes the reduced-basis technique [64, 69] and proper orthogonal decomposition (POD) [39, 16] for parameterized stationary problems; balanced truncation [56], rational interpolation [5, 34], and Craig–Bampton model reduction [21] for linear time invariant (LTI) systems; and Galerkin projection [39], least-squares Petrov–Galerkin projection [14], and other Petrov–Galerkin projections [80] with (balanced) POD [39, 46, 80, 68] for nonlinear dynamical systems.

The Kolmogorov n-width [63] provides one way to quantify the optimal linear trial subspace; it is defined as

$$d_n(\mathcal{M}) := \inf_{\mathcal{S}_n} \sup_{f \in \mathcal{M}} \inf_{g \in \mathcal{S}_n} \|f - g\|,$$

where the first infimum is taken over all n-dimensional subspaces of the state space, and $\mathcal{M}$ denotes the manifold of solutions over all time and parameters. Assuming the dynamical system has a unique trajectory for each parameter instance, the *intrinsic solution-manifold dimensionality* is (at most) equal to the number of parameters plus one (time). For problems that exhibit a fast decaying Kolmogorov n-width (e.g., diffusion-dominated problems), employing a linear trial subspace is theoretically justifiable [4, 7] and has enjoyed many successful demonstrations. Unfortunately, many computational problems exhibit slowly decaying Kolmogorov n-width (e.g., advection-dominated problems). In such cases, the use of low-dimensional linear trial subspaces often produces inaccurate results; the ROM dimensionality must be significantly increased to yield acceptable accuracy [60]. Indeed, the Kolmogorov n-width with n equal to the intrinsic solution-manifold dimensionality is often quite large for such problems.

Several approaches have been pursued to address this n-width limitation of linear trial subspaces. One set of approaches transforms the trial basis to improve its approximation properties for advection-dominated problems. Such methods include separating transport dynamics via 'freezing' [59], applying a coordinate transformation to the trial basis [42, 58, 11], shifting the POD basis [65], transforming the physical domain of the snapshots [79, 78], constructing the trial basis on a Lagrangian formulation of the governing equations [55], and using Lax pairs of the Schrödinger operator to construct a time-evolving trial basis [29]. Other approaches pursue the use of multiple linear subspaces instead of employing a single global linear trial subspace; these local subspaces can be tailored to different regions of the time domain [26, 24], physical domain [73], or state space [3, 62]; Ref [2] employs local trial subspaces in time and parameter spaces and applies ℓ^1-norm minimizations of the residual. Ref. [12] aims to overcome the limitations of using a linear trial subspace by providing online-adaptive h-refinement mechanism that constructs a hierarchical sequence of linear subspaces that converges to the original state space. However, all of these methods attempt to construct, manually transform, or refine a linear basis to be locally accurate; they do not consider nonlinear trial manifolds of more general structure. Further, many of these approaches rely on substantial additional knowledge about the problem, such as the particular advection phenomena governing basis shifting.

This work aims to address the fundamental n-width deficiency of linear trial subspaces. However, in contrast to the above methods, we pursue an approach that is both *more general* (i.e., it should not be limited to piecewise linear manifolds or mode transformations) and only requires the *same snapshot data* as typical POD-based approaches (e.g., it should require no special knowledge about any particular advection phenomena). To accomplish this objective, we propose an approach that (1) performs optimal projection of dynamical systems onto arbitrary nonlinear trial manifolds (during the online stage), and (2) computes this nonlinear trial manifold from snapshot data alone (during the offline stage).

For the first component, we perform optimal projection onto arbitrary (continuously differentiable) nonlinear trial manifolds by applying minimum-residual formulations at the time-continuous (ODE) and time-discrete (OΔE) levels. The time-continuous formulation leads to *manifold Galerkin* projection, which can be interpreted as performing orthogonal projection of the velocity onto the tangent space of the trial

manifold. The time-discrete formulation leads to *manifold least-squares Petrov–Galerkin (LSPG)* projection, which can also be straightforwardly extended to stationary (i.e., steady-state) problems. We also perform analyses that illustrate the relationship between these manifold ROMs and classical linear-subspace ROMs. Manifold Galerkin and manifold LSPG projection require the trial manifold to be characterized as a (generally nonlinear) mapping from the low-dimensional reduced state to the high-dimensional state; the mapping from the high-dimensional state to the low-dimensional state is not required.

The second component aims to compute a nonlinear trial manifold from snapshot data alone. Many machine-learning methods exist to perform nonlinear dimensionality reduction. However, many of these methods do not provide the required mapping from the low-dimensional embedding to the high-dimensional input; examples include Isomap [75], locally linear embedding (LLE) [67], Hessian eigenmaps [25], spectral embedding [6], and t-SNE [51]. Methods that *do* provide this required mapping include self-organizing maps [45], generative topographic mapping [8], kernel principal component analysis (PCA) [72], Gaussian process latent variable model [47], diffeomorphic dimensionality reduction [77], and autoencoders [37]. In principle, manifold Galerkin and manifold LSPG projection could be applied with manifolds constructed by any of the methods in the latter category. However, this study restricts focus to autoencoders—more specifically deep convolutional autoencoders—due to their expressiveness and scalability, as well as the availability of high-performance software tools for their construction.

Autoencoders (also known as auto-associators [23]) comprise a specific type of feedforward neural network that aim to learn the identity mapping: they attempt to copy the input to an accurate approximation of itself. Learning the identity mapping is not a particularly useful task unless, however, it associates with a dimensionality-reduction procedure comprising data compression and subsequent recovery. This is precisely what autoencoders accomplish by employing a neural-network architecture consisting of two parts: an *encoder* that provides a nonlinear mapping from the high-dimensional input to a low-dimensional embedding, and a *decoder* that provides a nonlinear mapping from the low-dimensional embedding to an approximation of the high-dimensional input. Convolutional autoencoders are a specific type of autoencoder that employ convolutional layers, which have been shown to be effective for extracting representative features in images [53]. Inspired by the analogy between images and spatially distributed dynamical-system states (e.g., when the dynamical system corresponds to the spatial discretization of a partial-differential-equations model), we propose a specific deep convolutional autoencoder architecture tailored to dynamical systems with states that are spatially distributed. Critically, training this autoencoder requires only the same snapshot data as POD; no additional problem-specific information is needed.

In summary, new contributions of this work include:

1. Manifold Galerkin (Section 3.2) and manifold LSPG (Section 3.3) projection techniques, which project the dynamical-system model onto arbitrary continuously-differentiable manifolds. We equip these methods with
   (a) the ability to exactly satisfy the initial condition (Remark 3.1), and
   (b) quasi-Newton solvers (Section 3.4) to solve the system of algebraic equations arising from implicit time integration.
2. Analysis (Section 4), which includes
   (a) demonstrating that employing an affine trial manifold recovers classical linear-subspace Galerkin and LSPG projection (Proposition 4.1),
   (b) sufficient conditions for commutativity of time discretization and manifold Galerkin projection (Theorem 4.1),
   (c) conditions under which manifold Galerkin and manifold LSPG projection are equivalent (Theorem 4.2), and
   (d) *a posteriori* discrete-time error bounds for both the manifold Galerkin and manifold LSPG projection methods (Theorem 4.3).
3. A novel convolutional autoencoder architecture tailored to spatially distributed dynamical-system states (Section 5.2) with accompanying offline training algorithm that requires only the same snapshot data as POD (Section 6).

4. Numerical experiments on advection-dominated benchmark problems (Section 7). These experiments illustrate the ability of the method to outperform even the projection of the solution onto the optimal linear subspace; further, the proposed method is close to achieving the optimal performance of any nonlinear-manifold method. This demonstrates the method's ability to overcome the intrinsic n-width limitations of linear trial subspaces.

We note that the methodology is applicable to both linear and nonlinear dynamical systems.

To the best of our knowledge, Refs. [35, 43] comprise the only attempts to incorporate an autoencoder within a projection-based ROM. These methods seek solutions in the nonlinear trial manifold provided by an autoencoder; however, these methods reduce the number of equations by applying the encoder to the velocity. Unfortunately, as we discuss in Remark 3.5, this approach is *kinematically inconsistent*, as the velocity resides in the tangent space to the manifold, not the manifold itself. Thus, encoding the velocity can produce significant approximation errors. Instead, the proposed manifold Galerkin and manifold LSPG projection methods produce approximations that associate with minimum-residual formulations and adhere to the kinematics imposed by the trial manifold.

Relatedly, Ref. [33] proposes a general framework for projection of dynamical systems onto nonlinear manifolds. However, the proposed method constructs a piecewise linear trial manifold by generating local linear subspaces and concatenating those subspaces. Then, the method projects the residual of the governing equations onto a nonlinear test manifold that is also piecewise linear; this is referred to as a 'piecewise linear projection function'. Thus, the approach is limited to piecewise-linear manifolds, and the resulting approximation does not associate with any optimality property.

We also note that autoencoders have been applied to various non-intrusive model-reduction methods that are purely data driven in nature and are not based on a projection process. Examples include Ref. [54], which constructs a nonlinear model of wall turbulence using an autoencoder; Ref. [31], which applies an autoencoder to compress the state, followed by a recurrent neural network (RNN) [70] to learn the dynamics; Refs. [74, 61, 50, 57], which apply autoencoders to learn approximate invariant subspaces of the Koopman operator; and Ref. [18], which applies hierarchical dimensionality reduction comprising autoencoders and PCA followed by dynamics learning to recover missing CFD data.

The paper is organized as follows. Section 2 describes the full-order model, which corresponds to a parameterized system of (linear or nonlinear) ordinary differential equations. Section 3 describes model reduction on nonlinear manifolds, including the mathematical characterization of the nonlinear trial manifold (Section 3.1), manifold Galerkin projection (Section 3.2), manifold LSPG projection (Section 3.3), and associated quasi-Newton methods to solve the system of algebraic equations arising at each time instance in the case of implicit time integration (Section 3.4). Section 4 provides the aforementioned analysis results. Section 5 describes a practical approach for constructing the nonlinear trial manifold using deep convolutional autoencoders, including a brief description of autoencoders (Section 5.1), the proposed autoencoder architecture applicable to spatially distributed states (Section 5.2), and the way in which the proposed autoencoder can be used to satisfy the initial condition (Section 5.3). When the manifold Galerkin and manifold LSPG ROMs employ this choice of decoder, we refer to them as *Deep Galerkin* and *Deep LSPG* ROMs, respectively. Section 6 describes offline training, which entails snapshot-based data collection (Section 6.1), data standardization (Section 6.2), and autoencoder training (Section 6.3); Algorithm 1 summarizes the offline training stage. Section 7 assesses the performance of the proposed Deep Galerkin and Deep LSPG ROMs compared to (linear-subspace) POD–Galerkin and POD–LSPG ROMs on two advection-dominated benchmark problems. Finally, Section 8 concludes the paper.

2. Full-order model

This work considers the full-order model (FOM) to correspond to a dynamical system expressed as a parameterized system of ordinary differential equations (ODEs)

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, t; \boldsymbol{\mu}), \qquad \boldsymbol{x}(0; \boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu}), \tag{2.1}$$

where $t \in [0,T]$ denotes time with final time $T \in \mathbb{R}_+$, and $\boldsymbol{x} : [0,T] \times \mathcal{D} \to \mathbb{R}^N$ denotes the time-dependent, parameterized state implicitly defined as the solution to problem (2.1) given the parameters $\boldsymbol{\mu} \in \mathcal{D}$. Here, $\mathcal{D} \subseteq \mathbb{R}^{n_\mu}$ denotes the parameter space, $\boldsymbol{x}^0 : \mathcal{D} \to \mathbb{R}^N$ denotes the parameterized initial condition, and $\boldsymbol{f} : \mathbb{R}^N \times [0,T] \times \mathcal{D} \to \mathbb{R}^N$ denotes the velocity, which may be linear or nonlinear in its first argument. We denote differentiation of a variable x with respect to time by $\dot{x}$. Such dynamical systems may arise from the semidiscretization of a partial-differential-equations (PDE) model, for example. We refer to Eq. (2.1) as the FOM ODE.

Numerically solving the FOM ODE (2.1) requires application of a time-discretization method. For simplicity, this work restricts attention to linear multistep methods; see Ref. [13] for analysis of both (linear-subspace) Galerkin and LSPG reduced-order models with Runge–Kutta schemes. A linear k-step method applied to numerically solve the FOM ODE (2.1) leads to solving the system of algebraic equations

$$\boldsymbol{r}^n(\boldsymbol{x}^n; \boldsymbol{\mu}) = \boldsymbol{0}, \quad n = 1, \dots, N_t, \tag{2.2}$$

where the time-discrete residual $\boldsymbol{r}^n : \mathbb{R}^N \times \mathcal{D} \to \mathbb{R}^N$ is defined as

$$\boldsymbol{r}^n : (\boldsymbol{\xi}; \boldsymbol{\nu}) \mapsto \alpha_0 \boldsymbol{\xi} - \Delta t \beta_0 \boldsymbol{f}(\boldsymbol{\xi}, t^n; \boldsymbol{\nu}) + \sum_{j=1}^{k} \alpha_j \boldsymbol{x}^{n-j} - \Delta t \sum_{j=1}^{k} \beta_j \boldsymbol{f}(\boldsymbol{x}^{n-j}, t^{n-j}; \boldsymbol{\nu}). \tag{2.3}$$

Here, $\Delta t \in \mathbb{R}_+$ denotes the time step, $\boldsymbol{x}^k$ denotes the numerical approximation to $\boldsymbol{x}(k\Delta t; \boldsymbol{\mu})$, and the coefficients α_j and β_j, $j = 0, \dots, k$ with $\sum_{j=0}^{k} \alpha_j = 0$ define a particular linear multistep scheme. These methods are implicit if $\beta_0 \neq 0$. For notational simplicity, we have assumed a uniform time step Δt and a fixed number of steps k for each time instance; the extensions to nonuniform grids and a non-constant value of k are straightforward. We refer to Eq. (2.2) as the FOM OΔE.

Remark 2.1 (Intrinsic solution-manifold dimensionality)**.** Assuming the initial value problem (2.2) has a unique solution for each parameter instance $\boldsymbol{\mu} \in \mathcal{D}$, the intrinsic dimensionality of the solution manifold $\{\boldsymbol{x}(t; \boldsymbol{\mu}) \mid t \in [0,T],\ \boldsymbol{\mu} \in \mathcal{D}\}$ is (at most) $p^\star = n_\mu + 1$, as the mapping $(t; \boldsymbol{\mu}) \mapsto \boldsymbol{x}$ is unique in this case. This provides a practical lower bound on the dimension of a nonlinear trial manifold for exactly representing the dynamical-system state.

3. Model reduction on nonlinear manifolds

This section proposes two classes of residual-minimizing ROMs on nonlinear manifolds. The first minimizes the (time-continuous) FOM ODE residual and is analogous to classical Galerkin projection, while the second minimizes the (time-discrete) FOM OΔE residual and is analogous to least-squares Petrov–Galerkin (LSPG) projection [14, 17, 13]. Section 3.1 introduces the notion of a nonlinear trial manifold, Section 3.2 describes the manifold Galerkin ROM resulting from time-continuous residual minimization, and Section 3.3 describes the manifold LSPG ROM resulting from time-discrete residual minimization.

3.1. Nonlinear trial manifold

We begin by prescribing the subset of the original state space $\mathbb{R}^N$ on which the ROM techniques will seek approximate solutions. Rather than introducing a classical affine *trial subspace* for this purpose, we instead introduce a continuously differentiable nonlinear *trial manifold*. In particular, we seek approximate solutions $\tilde{\boldsymbol{x}} \approx \boldsymbol{x}$ of the form

$$\tilde{\boldsymbol{x}}(t; \boldsymbol{\mu}) = \boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}(t; \boldsymbol{\mu})), \tag{3.1}$$

where $\tilde{\boldsymbol{x}} : \mathbb{R}_+ \times \mathcal{D} \to \boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \mathcal{S}$ and $\mathcal{S} := \{\boldsymbol{g}(\hat{\boldsymbol{\xi}}) \mid \hat{\boldsymbol{\xi}} \in \mathbb{R}^p\}$ denotes the nonlinear trial manifold from the extrinsic view. Here, $\boldsymbol{x}_{\text{ref}} : \mathcal{D} \to \mathbb{R}^N$ denotes a parameterized reference state and $\boldsymbol{g} : \hat{\boldsymbol{\xi}} \mapsto \boldsymbol{g}(\hat{\boldsymbol{\xi}})$ with $\boldsymbol{g} : \mathbb{R}^p \to \mathbb{R}^N$ and $p \leq N$ denotes the parameterization function, which we refer to in this work as the *decoder*,

which comprises a nonlinear mapping from low-dimensional generalized coordinates $\hat{\boldsymbol{x}} : \mathbb{R}_+ \times \mathcal{D} \to \mathbb{R}^p$ to the high-dimensional state approximation. From application of the chain rule, the approximated velocity is then

$$\dot{\tilde{\boldsymbol{x}}}(t;\boldsymbol{\mu}) = \boldsymbol{J}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu}))\dot{\hat{\boldsymbol{x}}}(t;\boldsymbol{\mu}), \tag{3.2}$$

where $\boldsymbol{J} : \hat{\boldsymbol{\xi}} \mapsto \frac{d\boldsymbol{g}}{d\hat{\boldsymbol{\xi}}}(\hat{\boldsymbol{\xi}})$ with $\boldsymbol{J} : \mathbb{R}^p \to \mathbb{R}^{N\times p}$ denotes the Jacobian of the decoder. This Jacobian defines the tangent space to the manifold

$$T_{\hat{\boldsymbol{x}}}\mathcal{S} \equiv \mathrm{Ran}(\boldsymbol{J}(\hat{\boldsymbol{x}})), \tag{3.3}$$

where $\mathrm{Ran}(\boldsymbol{A})$ denotes the range of matrix $\boldsymbol{A}$. From Remark 2.1, we observe that the nonlinear trial manifold dimension must be greater than or equal to the intrinsic solution-manifold dimension, i.e., $p \geq p^\star$, to be able to exactly represent the state.

Remark 3.1 (Initial-condition satisfaction)**.** Satisfaction of the initial conditions requires the initial generalized coordinates $\hat{\boldsymbol{x}}(0;\boldsymbol{\mu}) = \hat{\boldsymbol{x}}^0(\boldsymbol{\mu})$ to satisfy $\boldsymbol{g}(\hat{\boldsymbol{x}}^0(\boldsymbol{\mu})) = \boldsymbol{x}^0(\boldsymbol{\mu})$. This can be achieved for any choice of $\hat{\boldsymbol{x}}^0(\boldsymbol{\mu})$ provided the reference state is set to

$$\boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu}) - \boldsymbol{g}(\hat{\boldsymbol{x}}^0(\boldsymbol{\mu})). \tag{3.4}$$

However, doing so must ensure that the decoder can accurately represent deviations from this reference state (3.4). In the context of the proposed autoencoder-based trial manifold, Section 5.3 describes a strategy for computing the initial generalized coordinates $\hat{\boldsymbol{x}}^0(\boldsymbol{\mu})$ and defining training data for manifold construction such that the decoder accomplishes this.

Remark 3.2 (Linear trial subspace)**.** The classical linear (or more precisely *affine*) trial subspace corresponds to a specific case of a nonlinear trial manifold; this occurs when the decoder is linear. In this case, the decoder can be expressed as $\boldsymbol{g} : \hat{\boldsymbol{\xi}} \mapsto \boldsymbol{\Phi}\hat{\boldsymbol{\xi}}$, where $\boldsymbol{\Phi} \in \mathbb{R}^{N\times p}_\star$ denotes trial-basis matrix and $\mathbb{R}^{n\times m}_\star$ denotes the set of full-column-rank $n \times m$ matrices (the non-compact Stiefel manifold). In this case, the approximated state and velocity can be expressed as

$$\tilde{\boldsymbol{x}}(t;\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu}) + \boldsymbol{\Phi}\hat{\boldsymbol{x}}(t;\boldsymbol{\mu}) \quad \text{and} \quad \dot{\tilde{\boldsymbol{x}}}(t;\boldsymbol{\mu}) = \boldsymbol{\Phi}\dot{\hat{\boldsymbol{x}}}(t;\boldsymbol{\mu}), \tag{3.5}$$

respectively, such that the trial manifold is affine, i.e., $\mathcal{S} = \mathrm{Ran}(\boldsymbol{\Phi})$, and the decoder Jacobian is the constant matrix $\boldsymbol{J}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})) = \boldsymbol{J} = \boldsymbol{\Phi}$. Note that this approach can enforce the initial condition by setting the initial generalized coordinates $\hat{\boldsymbol{x}}^0(\boldsymbol{\mu})$ to zero and subsequently setting $\boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu})$. Common choices for computing the reduced-basis matrix $\boldsymbol{\Phi}$ when the velocity is linear in its first argument include balanced truncation [56], rational interpolation [5, 34], the reduced-basis method [64, 69], Rayleigh–Ritz eigenvectors [21]; common choices when the velocity is nonlinear include POD [38] and balanced POD [46, 80, 68].

3.2. Manifold Galerkin ROM: time-continuous residual minimization

We now derive the ROM corresponding to time-continuous residual minimization. To do so, define the FOM ODE residual as

$$\boldsymbol{r} : (\boldsymbol{v}, \boldsymbol{\xi}, \tau; \boldsymbol{\nu}) \mapsto \boldsymbol{v} - \boldsymbol{f}(\boldsymbol{\xi}, \tau; \boldsymbol{\nu})$$

with $\boldsymbol{r} : \mathbb{R}^N \times \mathbb{R}^N \times \mathbb{R}_+ \times \mathcal{D} \to \mathbb{R}^N$. It can be easily verified that the FOM ODE (2.1) is equivalent to

$$\boldsymbol{r}\left(\dot{\boldsymbol{x}}, \boldsymbol{x}, t; \boldsymbol{\mu}\right) = \boldsymbol{0}, \qquad \boldsymbol{x}(0;\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu}). \tag{3.6}$$

To derive the time-continuous residual-minimizing ROM, we substitute the approximated state $\boldsymbol{x} \leftarrow \tilde{\boldsymbol{x}}$ defined in Eq. (3.1) and velocity $\dot{\boldsymbol{x}} \leftarrow \dot{\tilde{\boldsymbol{x}}}$ defined in Eq. (3.2) into the FOM ODE (3.6) and minimize the ℓ^2-norm of the residual, i.e.,

$$\dot{\hat{\boldsymbol{x}}}(t;\boldsymbol{\mu}) = \operatorname*{arg\,min}_{\hat{\boldsymbol{v}}\in\mathbb{R}^p} \left\|\boldsymbol{r}\left(\boldsymbol{J}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu}))\hat{\boldsymbol{v}}, \boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})), t; \boldsymbol{\mu}\right)\right\|_2^2 \tag{3.7}$$

with initial condition $\hat{\boldsymbol{x}}(0;\boldsymbol{\mu}) = \hat{\boldsymbol{x}}^0(\boldsymbol{\mu})$. If the Jacobian $\boldsymbol{J}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu}))$ has full column rank, then Problem (3.7) is convex and has a unique solution that leads to the *manifold Galerkin* ODE

$$\dot{\hat{\boldsymbol{x}}}(t;\boldsymbol{\mu}) = \boldsymbol{J}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu}))^+ \boldsymbol{f}\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})), t; \boldsymbol{\mu}\right), \quad \hat{\boldsymbol{x}}(0;\boldsymbol{\mu}) = \hat{\boldsymbol{x}}^0(\boldsymbol{\mu}), \tag{3.8}$$

where the superscript $+$ denotes the Moore–Penrose pseudoinverse.

We note that although the state evolves on the nonlinear manifold $\mathcal{S}$, the generalized coordinates evolve in the Euclidean space $\mathbb{R}^p$, which facilitates time integration of the manifold Galerkin ODE (3.8). Indeed, this can be accomplished using any time integrator. If a linear multistep scheme is employed, then the manifold Galerkin OΔE corresponds to

$$\hat{\boldsymbol{r}}_G^n(\hat{\boldsymbol{x}}^n;\boldsymbol{\mu}) = \boldsymbol{0}, \tag{3.9}$$

where the manifold Galerkin OΔE residual is

$$\begin{aligned}\hat{\boldsymbol{r}}_G^n : (\hat{\boldsymbol{\xi}};\boldsymbol{\nu}) \mapsto & \alpha_0\hat{\boldsymbol{\xi}} - \Delta t\beta_0 \boldsymbol{J}(\hat{\boldsymbol{\xi}})^+ \boldsymbol{f}(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{\xi}}), t^n; \boldsymbol{\nu}) + \\ & \sum_{j=1}^k \alpha_j \hat{\boldsymbol{x}}^{n-j} - \Delta t \sum_{j=1}^k \beta_j \boldsymbol{J}(\hat{\boldsymbol{x}}^{n-j})^+ \boldsymbol{f}(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}), t^{n-j}; \boldsymbol{\nu}).\end{aligned} \tag{3.10}$$

We note that the manifold Galerkin OΔE is nonlinear if either the velocity is nonlinear in its first argument or if the trial manifold is nonlinear.

Remark 3.3 (Manifold Galerkin projection as orthogonal projection onto the tangent space)**.** Eq. (3.7) can be written equivalently as

$$\dot{\hat{\boldsymbol{x}}}(t;\boldsymbol{\mu}) = \underset{\boldsymbol{v} \in T_{\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})}\mathcal{S}}{\arg\min} \left\| \boldsymbol{v} - \boldsymbol{f}\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})), t; \boldsymbol{\mu}\right) \right\|_2^2, \tag{3.11}$$

where the tangent space is defined in Eq. (3.3). The solution to this minimization problem is $\dot{\hat{\boldsymbol{x}}}(t;\boldsymbol{\mu}) = \boldsymbol{J}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu}))\boldsymbol{J}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu}))^+ \boldsymbol{f}\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})), t; \boldsymbol{\mu}\right)$. Thus, Manifold Galerkin projection can be interpreted as computing the orthogonal (i.e., ℓ^2-optimal) projection of the velocity onto the tangent space $T_{\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})}\mathcal{S}$ of the trial manifold.

Remark 3.4 (Weighted ℓ^2-norms for manifold Galerkin projection)**.** We note that the minimization problem (3.7) could be posed in other norms. For example, if we consider residual minimization in the weighted ℓ^2-norm $\|\cdot\|_{\boldsymbol{\Theta}}$ satisfying $\|\boldsymbol{x}\|_{\boldsymbol{\Theta}} \equiv (\boldsymbol{x},\boldsymbol{x})_{\boldsymbol{\Theta}}$ with $(\boldsymbol{x},\boldsymbol{y})_{\boldsymbol{\Theta}} \equiv \sqrt{\boldsymbol{x}^T\boldsymbol{\Theta}\boldsymbol{y}}$ and $\boldsymbol{\Theta} \equiv \boldsymbol{\Theta}^{T/2}\boldsymbol{\Theta}^{1/2}$, then Problem (3.7) becomes

$$\dot{\hat{\boldsymbol{x}}}(t;\boldsymbol{\mu}) = \underset{\hat{\boldsymbol{v}} \in \mathbb{R}^p}{\arg\min} \left\| \boldsymbol{r}\left(\boldsymbol{J}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu}))\hat{\boldsymbol{v}}, \boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})), t; \boldsymbol{\mu}\right) \right\|_{\boldsymbol{\Theta}}^2, \tag{3.12}$$

leading to the manifold Galerkin ODE

$$\dot{\hat{\boldsymbol{x}}}(t;\boldsymbol{\mu}) = [\boldsymbol{\Theta}^{1/2}\boldsymbol{J}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu}))]^+ \boldsymbol{\Theta}^{1/2} \boldsymbol{f}\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})), t; \boldsymbol{\mu}\right), \quad \hat{\boldsymbol{x}}(0;\boldsymbol{\mu}) = \hat{\boldsymbol{x}}^0(\boldsymbol{\mu}), \tag{3.13}$$

and manifold Galerkin OΔE residual

$$\hat{\boldsymbol{r}}_G^n : (\hat{\boldsymbol{\xi}};\boldsymbol{\nu}) \mapsto \alpha_0\hat{\boldsymbol{\xi}} - \Delta t\beta_0 [\boldsymbol{\Theta}^{1/2}\boldsymbol{J}(\hat{\boldsymbol{\xi}})]^+ \boldsymbol{\Theta}^{1/2} \boldsymbol{f}(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{\xi}}), t^n; \boldsymbol{\nu}) + \tag{3.14}$$

$$\sum_{j=1}^k \alpha_j \hat{\boldsymbol{x}}^{n-j} - \Delta t \sum_{j=1}^k \beta_j [\boldsymbol{\Theta}^{1/2}\boldsymbol{J}(\hat{\boldsymbol{x}}^{n-j})]^+ \boldsymbol{\Theta}^{1/2} \boldsymbol{f}(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}), t^{n-j}; \boldsymbol{\nu}). \tag{3.15}$$

For notational simplicity, this work restricts attention to the ℓ^2-norm; future work will consider such weighted norms, e.g., to enable hyper-reduction.

Remark 3.5 (Alternative Galerkin projection)**.** Refs. [35, 43] provide an alternative way to perform Galerkin projection on nonlinear manifolds constructed from autoencoders. In contrast to the proposed manifold Galerkin ROM (and manifold LSPG ROM introduced in Section 3.3 below), these methods additionally

require the existence of an *encoder* $\bar{\boldsymbol{g}} : \mathbb{R}^N \to \mathbb{R}^p$ that satisfies $\boldsymbol{g} \circ \bar{\boldsymbol{g}} : \boldsymbol{x} \mapsto \tilde{\boldsymbol{x}}$ with $\tilde{\boldsymbol{x}} \approx \boldsymbol{x}$, which is trained on state-snapshot data. These methods then form the low-dimensional system of ODEs

$$\dot{\hat{\boldsymbol{x}}}(t;\boldsymbol{\mu}) = \bar{\boldsymbol{g}}\left(\boldsymbol{f}\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})), t; \boldsymbol{\mu}\right)\right), \quad \hat{\boldsymbol{x}}(0;\boldsymbol{\mu}) = \bar{\boldsymbol{g}}(\boldsymbol{x}^0(\boldsymbol{\mu})). \tag{3.16}$$

One problem with this approach is that it implicitly assumes that the velocity $\boldsymbol{f}$ can be represented on the manifold used to represent the state. This is *kinematically inconsistent*, as the velocity resides in the tangent space to this manifold as derived in Eq. (3.2), i.e., $\dot{\hat{\boldsymbol{x}}}(t;\boldsymbol{\mu}) \in T_{\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})}\mathcal{S}$; the tangent space and the manifold coincide if and only if the trial manifold associates with a linear trial subspace. Thus, encoding the velocity is likely to produce a poor approximation of the reduced-state velocity $\dot{\hat{\boldsymbol{x}}}(t;\boldsymbol{\mu})$. Instead, the proposed manifold Galerkin ROM performs orthogonal projection of the velocity onto the tangent space (see Remark 3.3). We also note the approaches proposed in Refs. [35, 43] compute the initial state by encoding the initial condition, which is not guaranteed to satisfy the initial conditions as discussed in Remark 3.1.

3.3. Manifold LSPG ROM: time-discrete residual minimization

Analogously to the previous section, we now derive the ROM corresponding to time-discrete residual minimization. To do so, we simply substitute the approximated state $\boldsymbol{x} \leftarrow \tilde{\boldsymbol{x}}$ defined in Eq. (3.1) into the FOM OΔE (2.2) and minimize the ℓ^2-norm of the residual, i.e.,

$$\hat{\boldsymbol{x}}^n(\boldsymbol{\mu}) = \underset{\hat{\boldsymbol{\xi}} \in \mathbb{R}^p}{\arg\min} \left\| \boldsymbol{r}^n\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{\xi}}); \boldsymbol{\mu}\right)\right\|_2^2, \tag{3.17}$$

which is solved sequentially for $n = 1, \ldots, N_t$ with initial condition $\hat{\boldsymbol{x}}(0;\boldsymbol{\mu}) = \hat{\boldsymbol{x}}^0(\boldsymbol{\mu})$. Necessary optimality conditions for Problem (3.17) correspond to stationarity of the objective function, i.e., the solution $\hat{\boldsymbol{x}}^n$ satisfies the manifold LSPG OΔE

$$\boldsymbol{\Psi}^n(\hat{\boldsymbol{x}}^n;\boldsymbol{\mu})^T \boldsymbol{r}^n\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}^n); \boldsymbol{\mu}\right) = \boldsymbol{0}, \tag{3.18}$$

where the test basis matrix $\boldsymbol{\Psi}^n : \mathbb{R}^p \times \mathcal{D} \to \mathbb{R}^{N \times p}$ is defined as

$$\boldsymbol{\Psi}^n : (\hat{\boldsymbol{\xi}}; \boldsymbol{\nu}) \mapsto \frac{\partial \boldsymbol{r}^n}{\partial \boldsymbol{\xi}}(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{\xi}}); \boldsymbol{\nu})\boldsymbol{J}(\hat{\boldsymbol{\xi}}) = \left(\alpha_0 \boldsymbol{I} - \Delta t \beta_0 \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\xi}}\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{\xi}}), t^n; \boldsymbol{\nu}\right)\right)\boldsymbol{J}(\hat{\boldsymbol{\xi}}). \tag{3.19}$$

As with the manifold Galerkin OΔE, the manifold LSPG OΔE is nonlinear if either the velocity is nonlinear in its first argument or if the trial manifold is nonlinear.

Remark 3.6 (Weighted ℓ^2-norms for manifold LSPG projection)**.** As discussed in Remark 3.4, the minimization problem in (3.17) could be posed in other norms. If the weighted ℓ^2-norm $\|\cdot\|_{\boldsymbol{\Theta}}$ is employed, then Problem (3.17) becomes

$$\hat{\boldsymbol{x}}^n(\boldsymbol{\mu}) = \underset{\hat{\boldsymbol{\xi}} \in \mathbb{R}^p}{\arg\min} \left\| \boldsymbol{r}^n\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{\xi}}); \boldsymbol{\mu}\right)\right\|_{\boldsymbol{\Theta}}^2, \tag{3.20}$$

and the test basis matrix in the manifold LSPG OΔE(3.18) becomes

$$\boldsymbol{\Psi}^n : (\hat{\boldsymbol{\xi}}; \boldsymbol{\nu}) \mapsto \boldsymbol{\Theta}\frac{\partial \boldsymbol{r}^n}{\partial \boldsymbol{\xi}}(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{\xi}}); \boldsymbol{\nu})\boldsymbol{J}(\hat{\boldsymbol{\xi}}).$$

As with manifold Galerkin projection, this work restricts attention to the ℓ^2-norm for notational simplicity.

Remark 3.7 (Application to stationary problems)**.** Manifold LSPG projection can also be applied to stationary (i.e., steady-state) problems. Stationary problems are characterized by computing $\boldsymbol{x}(\boldsymbol{\mu})$ as the implicit solution to Eq. (2.1) with $\dot{\boldsymbol{x}} = \boldsymbol{0}$, i.e.,

$$\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\mu}) = \boldsymbol{0}, \tag{3.21}$$

where the (time-independent) velocity is equivalent to the stationary-problem residual. Substituting the approximated state $\boldsymbol{x}(\boldsymbol{\mu}) \leftarrow \tilde{\boldsymbol{x}}(\boldsymbol{\mu}) = \boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}(\boldsymbol{\mu}))$ into Eq. (3.21) and subsequently minimizing the ℓ^2-norm of the residual yields

$$\hat{\boldsymbol{x}}(\boldsymbol{\mu}) = \underset{\hat{\boldsymbol{\xi}} \in \mathbb{R}^p}{\arg\min} \left\| \boldsymbol{f}\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{\xi}}; \boldsymbol{\mu})\right) \right\|_2^2, \tag{3.22}$$

which has the same nonlinear-least-squares form as manifold LSPG projection applied to the dynamical system model (3.17). In this case, necessary optimality conditions for Problem (3.22) correspond to stationarity of the objective function, i.e., the solution $\hat{\boldsymbol{x}}(\boldsymbol{\mu})$ satisfies the system of algebraic equations

$$\boldsymbol{\Psi}(\hat{\boldsymbol{x}}(\boldsymbol{\mu}); \boldsymbol{\mu})^T \boldsymbol{f}\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}(\boldsymbol{\mu})); \boldsymbol{\mu}\right) = \boldsymbol{0}, \tag{3.23}$$

where the test basis matrix $\boldsymbol{\Psi} : \mathbb{R}^p \times \mathcal{D} \to \mathbb{R}^{N \times p}$ is defined as

$$\boldsymbol{\Psi} : (\hat{\boldsymbol{\xi}}; \boldsymbol{\nu}) \mapsto \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\xi}}\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{\xi}}); \boldsymbol{\nu}\right) \boldsymbol{J}(\hat{\boldsymbol{\xi}}).$$

Note that we do not consider a manifold Galerkin projection applied to stationary problems, as the manifold Galerkin ROM was derived by minimizing the time-continuous residual, and a Galerkin-like projection of the form $\boldsymbol{J}(\hat{\boldsymbol{x}}(\boldsymbol{\mu}))^T \boldsymbol{f}\left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}(\boldsymbol{\mu})); \boldsymbol{\mu}\right) = \boldsymbol{0}$ does not generally associate with any optimality property.

3.4. Quasi-Newton methods for implicit integrators

When an implicit time integrator is employed such that $\beta_0 \neq 0$ and the trial manifold is nonlinear, then solving the manifold Galerkin OΔE (3.10) and manifold LSPG OΔE (3.18) using Newton's method is challenging, as the residual Jacobians involve high-order derivatives of the decoder. For this purpose, we propose quasi-Newton methods that approximate these Jacobians while retaining convergence of the nonlinear solver to the solution of the OΔEs under certain conditions.

Manifold Galerkin. We first consider the manifold Galerkin case. If an implicit integrator is employed (i.e., $\beta_0 \neq 0$), then the manifold Galerkin OΔE (3.9) corresponds to a system of algebraic equations. Applying Newton's method to solve this system can be challenging for nonlinear decoders, as the Jacobian of the residual requires computing a third-order tensor of second derivatives in this case. Indeed, the ith column of the Jacobian is

$$\frac{\partial \hat{\boldsymbol{r}}_G^n}{\partial \hat{\xi}_i} : (\hat{\boldsymbol{w}}; \boldsymbol{\nu}) \mapsto \alpha_0 \hat{\boldsymbol{e}}_i - \Delta t \beta_0 \left(\frac{\partial \boldsymbol{J}^+}{\partial \hat{\xi}_i}(\hat{\boldsymbol{w}})\, \boldsymbol{f}(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{w}}), t^n; \boldsymbol{\nu}) + \boldsymbol{J}(\hat{\boldsymbol{w}})^+ \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\xi}}(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{w}}), t^n; \boldsymbol{\nu}) \boldsymbol{J}(\hat{\boldsymbol{w}}) \hat{\boldsymbol{e}}_i \right)$$

for $i = 1, \ldots, p$, where $\hat{\boldsymbol{e}}_i \in \{0, 1\}^p$ denotes the ith canonical unit vector. The gradient of the pseudo-inverse of the decoder Jacobian, which appears in the second term of the right-hand side, can be computed from the gradients of this Jacobian as

$$\begin{aligned} \frac{\partial \boldsymbol{J}^+}{\partial \hat{\xi}_i}(\hat{\boldsymbol{w}}) = &- \boldsymbol{J}(\hat{\boldsymbol{w}})^+ \frac{\partial \boldsymbol{J}}{\partial \hat{\xi}_i}(\hat{\boldsymbol{w}})\, \boldsymbol{J}(\hat{\boldsymbol{w}})^+ + \boldsymbol{J}(\hat{\boldsymbol{w}})^+ \boldsymbol{J}(\hat{\boldsymbol{w}})^{+\,T} \frac{\partial \boldsymbol{J}^T}{\partial \hat{\xi}_i}(\hat{\boldsymbol{w}})\, (\boldsymbol{I} - \boldsymbol{J}(\hat{\boldsymbol{w}}) \boldsymbol{J}(\hat{\boldsymbol{w}})^+) \\ &+ (\boldsymbol{I} - \boldsymbol{J}(\hat{\boldsymbol{w}})^+ \boldsymbol{J}(\hat{\boldsymbol{w}})) \frac{\partial \boldsymbol{J}^T}{\partial \hat{\xi}_i}(\hat{\boldsymbol{w}})\, \boldsymbol{J}(\hat{\boldsymbol{w}})^{+\,T} \boldsymbol{J}(\hat{\boldsymbol{w}})^+ \end{aligned}$$

The primary difficulty of computing this term is the requirement of computing second derivatives of the form $\partial \boldsymbol{J} / \partial \hat{\xi}_i = \partial^2 \boldsymbol{g} / \partial \hat{\boldsymbol{\xi}} \partial \hat{\xi}_i$. For this reason, we propose to approximate the residual Jacobian as $\tilde{\boldsymbol{J}}_{\hat{r}_G}^n \approx \partial \hat{\boldsymbol{r}}_G / \partial \hat{\boldsymbol{\xi}}$ by neglecting this term such that

$$\tilde{\boldsymbol{J}}_{\hat{r}_G}^n : (\hat{\boldsymbol{w}}; \boldsymbol{\nu}) \mapsto \alpha_0 \boldsymbol{I} - \Delta t \beta_0 \boldsymbol{J}(\hat{\boldsymbol{w}})^+ \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\xi}}(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{w}}), t^n; \boldsymbol{\nu}) \boldsymbol{J}(\hat{\boldsymbol{w}})$$

The resulting quasi-Newton method to solve the manifold Galerkin OΔE (3.9) takes the form

$$\begin{aligned}\tilde{\boldsymbol{J}}^n_{\hat{r}_G}(\hat{\boldsymbol{x}}^{n(k)};\boldsymbol{\mu})\boldsymbol{p}^{n(k)} &= -\hat{\boldsymbol{r}}_G(\hat{\boldsymbol{x}}^{n(k)};\boldsymbol{\mu})\\ \hat{\boldsymbol{x}}^{n(k+1)} &= \hat{\boldsymbol{x}}^{n(k)} + \alpha^{n(k)}\boldsymbol{p}^{n(k)},\end{aligned} \tag{3.24}$$

for $k = 0, \dots, K$. Here, $\hat{\boldsymbol{x}}^{n(0)}$ is the initial guess (often taken to be $\hat{\boldsymbol{x}}^{n-1}$ or a data-driven extrapolation [19]), and $\alpha^{n(k)} \in \mathbb{R}$ is the step length. Convergence of the quasi-Newton iterations (3.24) to a root of $\hat{\boldsymbol{r}}_G(\cdot;\boldsymbol{\mu})$ is ensured if

$$\hat{\boldsymbol{r}}_G(\hat{\boldsymbol{x}}^{n(k)};\boldsymbol{\mu})^T \tilde{\boldsymbol{J}}^n_{\hat{r}_G}(\hat{\boldsymbol{x}}^{n(k)};\boldsymbol{\mu})^T \frac{\partial \hat{\boldsymbol{r}}^n_G}{\partial \hat{\boldsymbol{\xi}}}(\hat{\boldsymbol{x}}^{n(k)};\boldsymbol{\mu})\hat{\boldsymbol{r}}_G(\hat{\boldsymbol{x}}^{n(k)};\boldsymbol{\mu}) > 0, \quad k = 0, \dots, K$$

and $\alpha^{n(k)} \in \mathbb{R}$, $k = 0, \dots, K$ is computed to satisfy the strong Wolfe conditions, as this ensures that the search direction $\boldsymbol{p}^{n(k)}$ is a descent direction of the function $\|\hat{\boldsymbol{r}}_G(\hat{\boldsymbol{\xi}};\boldsymbol{\nu})\|_2^2$ evaluated at $(\hat{\boldsymbol{\xi}};\boldsymbol{\nu}) = (\hat{\boldsymbol{x}}^{n(k)};\boldsymbol{\mu})$ for $k = 0, \dots, K$ [52].

Manifold LSPG. We now consider the manifold LSPG case. As in previous works that considered applying LSPG on affine trial subspaces [14, 17, 13], we propose to solve the nonlinear least-squares problem (3.17) using the Gauss–Newton method, which leads to the iterations

$$\begin{aligned}\boldsymbol{\Psi}^n(\hat{\boldsymbol{x}}^{n(k)};\boldsymbol{\mu})^T\boldsymbol{\Psi}^n(\hat{\boldsymbol{x}}^{n(k)};\boldsymbol{\mu})\boldsymbol{p}^{n(k)} &= -\boldsymbol{\Psi}^n(\hat{\boldsymbol{x}}^{n(k)};\boldsymbol{\mu})^T\boldsymbol{r}^n\left(\boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\mu}) + \boldsymbol{g}(\hat{\boldsymbol{x}}^{n(k)});\boldsymbol{\mu}\right)\\ \hat{\boldsymbol{x}}^{n(k+1)} &= \hat{\boldsymbol{x}}^{n(k)} + \alpha^{n(k)}\boldsymbol{p}^{n(k)},\end{aligned} \tag{3.25}$$

for $k = 0, \dots, K$ with $\hat{\boldsymbol{x}}^{n(0)}$ a provided initial guess, and $\alpha^{n(k)} \in \mathbb{R}$ is a step length chosen to satisfy the strong Wolfe conditions, which ensure global convergence to a local minimum of the manifold LSPG objective function in (3.17) that satisfies the stationarity conditions associated with the LSPG OΔE (3.18).

To show the connection between the Gauss–Newton method and the quasi-Newton approach proposed for the manifold Galerkin method, we define the discrete residual associated with manifold LSPG projection as

$$\hat{\boldsymbol{r}}^n_L : (\hat{\boldsymbol{\xi}};\boldsymbol{\nu}) \mapsto \boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}};\boldsymbol{\nu})^T\boldsymbol{r}^n\left(\boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{\xi}});\boldsymbol{\nu}\right) \tag{3.26}$$

such that the manifold LSPG OΔE (3.18) can be expressed simply as the system of algebraic equations

$$\hat{\boldsymbol{r}}^n_L(\hat{\boldsymbol{x}}^n;\boldsymbol{\mu}) = \boldsymbol{0}. \tag{3.27}$$

Solving Eq. (3.27) using Newton's method requires computing the residual Jacobian, whose ith column is

$$\begin{aligned}\frac{\partial \hat{\boldsymbol{r}}^n_L}{\partial \hat{\xi}_i} : (\hat{\boldsymbol{w}};\boldsymbol{\nu}) &\mapsto \frac{\partial \boldsymbol{\Psi}^n}{\partial \hat{\xi}_i}(\hat{\boldsymbol{w}};\boldsymbol{\nu})^T\boldsymbol{r}^n\left(\boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{w}});\boldsymbol{\nu}\right) + \boldsymbol{\Psi}^n(\hat{\boldsymbol{w}};\boldsymbol{\nu})^T\boldsymbol{\Psi}^n(\hat{\boldsymbol{w}};\boldsymbol{\nu})\hat{\boldsymbol{e}}_i\\ &= \frac{\partial \boldsymbol{J}}{\partial \hat{\xi}_i}(\hat{\boldsymbol{w}})^T\frac{\partial \boldsymbol{r}^n}{\partial \boldsymbol{\xi}}\left(\boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{w}});\boldsymbol{\nu}\right)^T\boldsymbol{r}^n\left(\boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\nu}) + \boldsymbol{g}(\hat{\boldsymbol{w}});\boldsymbol{\nu}\right) + \boldsymbol{\Psi}^n(\hat{\boldsymbol{w}};\boldsymbol{\nu})^T\boldsymbol{\Psi}^n(\hat{\boldsymbol{w}};\boldsymbol{\nu})\hat{\boldsymbol{e}}_i.\end{aligned}$$

Thus, it is clear that the Gauss–Newton iterations (3.25) are equivalent to employing a quasi-Newton method to solve the manifold LSPG OΔE (3.27) with an approximated residual Jacobian $\tilde{\boldsymbol{J}}^n_{\hat{r}_L} \approx \partial \hat{\boldsymbol{r}}^n_L / \partial \hat{\boldsymbol{\xi}}$ defined as $\tilde{\boldsymbol{J}}^n_{\hat{r}_L} : (\hat{\boldsymbol{w}};\boldsymbol{\nu}) \mapsto \boldsymbol{\Psi}^n(\hat{\boldsymbol{w}};\boldsymbol{\nu})^T\boldsymbol{\Psi}^n(\hat{\boldsymbol{w}};\boldsymbol{\nu})$, which is obtained from neglecting the term containing second derivatives of the decoder, namely $\partial \boldsymbol{J}/\partial \hat{\xi}_i = \partial^2 \boldsymbol{g}/\partial \hat{\boldsymbol{\xi}}\partial \hat{\xi}_i$.

4. Analysis

We now perform analysis of the proposed manifold Galerkin and manifold LSPG projection methods. For notational simplicity, this section omits the dependence of operators on parameters $\boldsymbol{\mu}$.

Proposition 4.1 (Affine trial manifold recovers classical Galerkin and LSPG)**.** If the trial manifold is affine, then manifold LSPG projection is equivalent to classical linear-subspace LSPG projection. If additionally the decoder mapping associates with an orthgonal matrix, then manifold Galerkin projection is equivalent to classical linear-subspace Galerkin projection.

Proof. If the trial manifold is affine, then the decoder can be expressed as the linear mapping

$$\boldsymbol{g} : \hat{\boldsymbol{\xi}} \mapsto \boldsymbol{\Phi}\hat{\boldsymbol{\xi}} \tag{4.1}$$

with $\boldsymbol{\Phi} \in \mathbb{R}^{N\times p}$.
Manifold Galerkin. Substituting the linear decoder (4.1) into the manifold Galerkin ODE (3.8) yields

$$\dot{\hat{\boldsymbol{x}}}(t) = \boldsymbol{\Phi}^{+}\boldsymbol{f}\left(\boldsymbol{x}_{\text{ref}} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}(t), t\right), \quad \hat{\boldsymbol{x}}(0) = \boldsymbol{0}. \tag{4.2}$$

Similarly, substituting the linear decoder (4.1) into the manifold Galerkin OΔE Eq. (3.9) yields the same system, but with the residual defined as

$$\hat{\boldsymbol{r}}^n_G : \hat{\boldsymbol{\xi}} \mapsto \boldsymbol{\Phi}^{+}\boldsymbol{r}^n(\boldsymbol{x}_{\text{ref}} + \boldsymbol{\Phi}\hat{\boldsymbol{\xi}}^n). \tag{4.3}$$

If additionally the trial-basis matrix is orthogonal, i.e., then $\boldsymbol{\Phi}^{+} = \boldsymbol{\Phi}^T$ and the ODE (4.2) is equivalent to the classical Galerkin ODE

$$\dot{\hat{\boldsymbol{x}}}(t) = \boldsymbol{\Phi}^{T}\boldsymbol{f}\left(\boldsymbol{x}_{\text{ref}} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}(t), t\right), \quad \hat{\boldsymbol{x}}(0) = \boldsymbol{0}, \tag{4.4}$$

while the OΔE residual (4.3) is equivalent to the classical Galerkin OΔE residual

$$\hat{\boldsymbol{r}}^n_G : \hat{\boldsymbol{\xi}} \mapsto \boldsymbol{\Phi}^{T}\boldsymbol{r}^n(\boldsymbol{x}_{\text{ref}} + \boldsymbol{\Phi}\hat{\boldsymbol{\xi}}^n). \tag{4.5}$$

Manifold LSPG. Substituting the linear decoder (4.1) into the manifold LSPG OΔE (3.18) yields the classical LSPG OΔE

$$\boldsymbol{\Psi}^n(\hat{\boldsymbol{x}}^n)^T\boldsymbol{r}^n\left(\boldsymbol{x}_{\text{ref}} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}^n\right) = \boldsymbol{0} \tag{4.6}$$

with classical LSPG test basis

$$\boldsymbol{\Psi}^n : \hat{\boldsymbol{\xi}} \mapsto \left(\alpha_0\boldsymbol{I} - \Delta t\beta_0\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\xi}}\left(\boldsymbol{x}_{\text{ref}} + \boldsymbol{\Phi}\hat{\boldsymbol{\xi}}, t^n\right)\right)\boldsymbol{\Phi}.$$

□

The manifold Galerkin OΔE (3.9)–(3.10) was derived using a project-then-discretize approach, as we formulated the residual-minimization problem (3.7) at the time-continuous level, and subsequently discretized the equivalent ODE (3.8) in time. We now derive conditions under which an equivalent model could have been derived using a discretize-then-project approach, i.e., by substituting the approximated state $\boldsymbol{x} \leftarrow \tilde{\boldsymbol{x}}$ defined in Eq. (3.1) into the FOM OΔE (2.2) and subsequently premultiplying the overdetermined system by a test basis.

Theorem 4.1 (Commutativity of time discretization and manifold Galerkin projection)**.** For manifold Galerkin projection, time discretization and projection are commutative if either (1) the trial manifold corresponds to an affine subspace, or (2) the nonlinear trial manifold is twice continuously differentiable; $\|\hat{\boldsymbol{x}}^{n-j} - \hat{\boldsymbol{x}}^n\| = O(\Delta t)$ for all $n \in \{1, \dots, N_t\}$, $j \in \{1, \dots, k\}$; and the limit $\Delta t \to 0$ is taken.

Proof. The discrete residual characterizing the manifold Galerkin OΔE (3.9), which was derived using a project-then-discretize approach, is defined as $\hat{\boldsymbol{r}}^n_G$ in Eq. (3.10).
Substituting the approximated state $\boldsymbol{x} \leftarrow \tilde{\boldsymbol{x}}$ defined in Eq. (3.1) into the FOM OΔE (2.2) yields $\boldsymbol{r}^n(\boldsymbol{x}_{\text{ref}} + \boldsymbol{g}(\hat{\boldsymbol{x}}^n)) = \boldsymbol{0}$. This is an overdetermined system of equations and thus may not have a solution. As such, we perform projection by enforcing orthgonality of this residual to a test basis $\boldsymbol{\Psi}_G : \mathbb{R}^p \to \mathbb{R}^{N\times p}$ such that the discretize-then-project manifold Galerkin OΔE corresponds to $\hat{\boldsymbol{r}}^n_{G,\text{disc}}(\hat{\boldsymbol{x}}^n) = \boldsymbol{0}$ with

$$\hat{\boldsymbol{r}}^n_{G,\text{disc}} : \hat{\boldsymbol{\xi}} \mapsto \boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{r}^n(\boldsymbol{x}_{\text{ref}} + \boldsymbol{g}(\hat{\boldsymbol{\xi}})). \tag{4.7}$$

The discretize-then-project manifold Galerkin ROM is equivalent to the proposed manifold Galerkin ROM if and only if there exists a full-rank matrix $\boldsymbol{A}_G^n : \mathbb{R}^p \to \mathbb{R}_\star^{p\times p}$ such that

$$\hat{\boldsymbol{r}}_G^n(\hat{\boldsymbol{\xi}}) = \boldsymbol{A}_G^n(\hat{\boldsymbol{\xi}})\hat{\boldsymbol{r}}_{G,\mathrm{disc}}^n(\hat{\boldsymbol{\xi}}). \tag{4.8}$$

Case 1. We now consider the general (non-asymptotic) case. Eq. (4.8) holds for any sequence of approximate solutions $\hat{\boldsymbol{x}}^n$, $n = 1, \ldots, N_t$ if and only if each term in this expansion matches, i.e.,

$$\begin{aligned}
\hat{\boldsymbol{\xi}} &= \boldsymbol{A}_G^n(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{g}(\hat{\boldsymbol{\xi}}) \\
\boldsymbol{J}(\hat{\boldsymbol{\xi}})^+\boldsymbol{f}(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{g}(\hat{\boldsymbol{\xi}}), t^n) &= \boldsymbol{A}_G^n(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{f}(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{g}(\hat{\boldsymbol{\xi}}), t^n) \\
\hat{\boldsymbol{x}}^{n-j} &= \boldsymbol{A}_G^n(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}) \\
\boldsymbol{J}(\hat{\boldsymbol{x}}^{n-j})^+\boldsymbol{f}(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}), t^{n-j}) &= \boldsymbol{A}_G^n(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{f}(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}), t^{n-j}),
\end{aligned} \tag{4.9}$$

where we have used $\sum_{j=0}^k \alpha_j = 0$. We first consider the first and third conditions of (4.9). Because the inverse of a nonlinear operator—if it exists—must also be nonlinear, a necessary condition for these two requirements is that the trial manifold corresponds is affine such that the decoder satisfies (4.1). Substituting Eq. (4.1) into the first and third of the conditions of (4.9) yields

$$\begin{aligned}
\hat{\boldsymbol{\xi}} &= \boldsymbol{A}_G^n(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{\Phi}\hat{\boldsymbol{\xi}} \\
\hat{\boldsymbol{x}}^{n-j} &= \boldsymbol{A}_G^n(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{\Phi}\hat{\boldsymbol{x}}^{n-j}
\end{aligned} \tag{4.10}$$

for $j = 1, \ldots, k$. A necessary and sufficient conditions for these to hold is

$$\boldsymbol{A}_G^n(\hat{\boldsymbol{\xi}}) = [\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{\Phi}]^{-1}. \tag{4.11}$$

We now consider the second and fourth conditions of (4.9). Substituting Eqs. (4.1) and (4.11) into these conditions yields

$$\begin{aligned}
\boldsymbol{\Phi}^+\boldsymbol{f}(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{\Phi}\hat{\boldsymbol{\xi}}, t^n) &= [\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{\Phi}]^{-1}\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{f}(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{\Phi}\hat{\boldsymbol{\xi}}, t^n) \\
\boldsymbol{\Phi}^+\boldsymbol{f}(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}^{n-j}, t^{n-j}) &= [\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{\Phi}]^{-1}\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{f}(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}^{n-j}, t^{n-j})
\end{aligned} \tag{4.12}$$

for $j = 1, \ldots, k$. Noting that $\boldsymbol{\Phi}^+ = [\boldsymbol{\Phi}^T\boldsymbol{\Phi}]^{-1}\boldsymbol{\Phi}^T$, a necessary and sufficient condition for (4.12) to hold is $\boldsymbol{\Phi}^+ = [\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{\Phi}]^{-1}\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T$. Because $\boldsymbol{\Phi}$ has no dependence on the generalized state, the test basis must also be independent of the generalized state, which yields

$$\boldsymbol{\Phi}^+ = [\boldsymbol{\Psi}_G^T\boldsymbol{\Phi}]^{-1}\boldsymbol{\Psi}_G^T. \tag{4.13}$$

Thus, without using any asymptotic arguments, the proposed manifold Galerkin ROM could have been derived using a discretize-then-project approach if and only if the test basis $\boldsymbol{\Psi}_G$ applied to the overdetermined FOM OΔE acting on the trial manifold satisfies Eq. (4.13). Examples of test bases that satisfy this condition include $\boldsymbol{\Psi}_G = [\boldsymbol{\Phi}^+]^T$ and $\boldsymbol{\Psi}_G = \gamma\boldsymbol{\Phi}$ for some nonzero scalar $\gamma \in \mathbb{R}$.

Case 2. We now consider asymptotic arguments. Assuming the nonlinear trial manifold is twice continuously differentiable, and $\|\hat{\boldsymbol{x}}^{n-j} - \hat{\boldsymbol{x}}^n\| = O(\Delta t)$, then for $\hat{\boldsymbol{\xi}}$ in a neighborhood of $\hat{\boldsymbol{x}}^n$ such that $\|\hat{\boldsymbol{\xi}} - \hat{\boldsymbol{x}}^n\| = O(\Delta t)$, we have

$$\begin{aligned}
\boldsymbol{J}(\hat{\boldsymbol{x}}^{n-j})^+ &= \boldsymbol{J}(\hat{\boldsymbol{x}}^n)^+ + O(\Delta t) \\
\boldsymbol{J}(\hat{\boldsymbol{\xi}})^+ &= \boldsymbol{J}(\hat{\boldsymbol{x}}^n)^+ + O(\Delta t) \\
\boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}) &= \boldsymbol{g}(\hat{\boldsymbol{x}}^n) + \boldsymbol{J}(\hat{\boldsymbol{x}}^n)[\hat{\boldsymbol{x}}^{n-j} - \hat{\boldsymbol{x}}^n] + O(\Delta t) \\
\boldsymbol{g}(\hat{\boldsymbol{\xi}}) &= \boldsymbol{g}(\hat{\boldsymbol{x}}^n) + \boldsymbol{J}(\hat{\boldsymbol{x}}^n)[\hat{\boldsymbol{\xi}} - \hat{\boldsymbol{x}}^n] + O(\Delta t).
\end{aligned} \tag{4.14}$$

Substituting these expressions into the definition of the discretize-then-project manifold Galerkin residual $\hat{\boldsymbol{r}}^n_{G,\text{disc}}$ and enforcing equivalence of each term in the matching conditions (4.8) and taking the limit $\Delta t \to 0$ yields

$$\begin{aligned}
\hat{\boldsymbol{\xi}} =& \boldsymbol{A}^n_G(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{J}(\hat{\boldsymbol{x}}^n)\hat{\boldsymbol{\xi}} \\
\boldsymbol{J}(\hat{\boldsymbol{x}}^n)^+\boldsymbol{f}(\boldsymbol{x}_{\text{ref}}+\boldsymbol{g}(\hat{\boldsymbol{\xi}}),t^n) =& \boldsymbol{A}^n_G(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{f}(\boldsymbol{x}_{\text{ref}}+\boldsymbol{g}(\hat{\boldsymbol{\xi}}),t^n) \\
\hat{\boldsymbol{x}}^{n-j} =& \boldsymbol{A}^n_G(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{J}(\hat{\boldsymbol{x}}^n)\hat{\boldsymbol{x}}^{n-j} \\
\boldsymbol{J}(\hat{\boldsymbol{x}}^n)^+\boldsymbol{f}(\boldsymbol{x}_{\text{ref}}+\boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}),t^{n-j}) =& \boldsymbol{A}^n_G(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{f}(\boldsymbol{x}_{\text{ref}}+\boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}),t^{n-j}),
\end{aligned} \tag{4.15}$$

where we have used $\sum_{j=0}^k \alpha_j = 0$. A necessary and sufficient condition for the first and third conditions to hold is

$$\boldsymbol{A}^n_G(\hat{\boldsymbol{\xi}}) = [\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{J}(\hat{\boldsymbol{x}}^n)]^{-1}. \tag{4.16}$$

Then, a necessary and sufficient condition for the second and fourth conditions to hold is

$$\boldsymbol{J}(\hat{\boldsymbol{x}}^n)^+ = [\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T\boldsymbol{J}(\hat{\boldsymbol{x}}^n)]^{-1}\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}})^T. \tag{4.17}$$

Examples of test bases that satisfy Eq. (4.17) include $\boldsymbol{\Psi}_G(\hat{\boldsymbol{\xi}}) = [\boldsymbol{J}(\hat{\boldsymbol{x}}^n)^+]^T$ and $\boldsymbol{\Psi}_G = \gamma\boldsymbol{J}(\hat{\boldsymbol{x}}^n)$ for some nonzero scalar $\gamma \in \mathbb{R}$. □

Theorem 4.1 shows that manifold Galerkin can be derived using a discretize-then-project approach for nonlinear trial manifolds. This shows that previous analysis [13, Theorem 3.4] related to commutativity of (residual-minimizing) Galerkin projection and time discretization extends to the case of nonlinear trial manifolds.

We now show that the limiting equivalence result reported in [13, Section 5] between Galerkin and LSPG projection for linear subspaces also extends to nonlinear trial manifolds.

Theorem 4.2 (Limiting equivalence)**.** Manifold Galerkin and manifold LSPG are equivalent if either (1) the trial manifold corresponds to an affine subspace and either an explicit scheme is employed or the limit $\Delta t \to 0$ is taken, or (2) the nonlinear manifold is twice continuously differentiable; $\|\hat{\boldsymbol{x}}^{n-j} - \hat{\boldsymbol{x}}^n\| = O(\Delta t)$ for all $n \in \{1,\dots,N_t\}$, $j \in \{1,\dots,k\}$; and the limit $\Delta t \to 0$ is taken.

Proof. The proof follows similar steps to those applied in the proof of Theorem 4.1. The discrete residual $\hat{\boldsymbol{r}}^n_L$ defined in (3.27), which characterizes the manifold LSPG OΔE (3.18), can be written as

$$\begin{aligned}
\hat{\boldsymbol{r}}^n_L : \hat{\boldsymbol{\xi}} \mapsto& \alpha_0\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T\boldsymbol{g}(\hat{\boldsymbol{\xi}}) - \Delta t\beta_0\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T\boldsymbol{f}(\boldsymbol{x}_{\text{ref}}+\boldsymbol{g}(\hat{\boldsymbol{\xi}}),t^n) + \\
&\sum_{j=1}^k \alpha_j\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T\boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}) - \Delta t\sum_{j=1}^k \beta_j\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T\boldsymbol{f}(\boldsymbol{x}_{\text{ref}}+\boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}),t^{n-j})
\end{aligned} \tag{4.18}$$

where we have used $\sum_{i=0}^k \alpha_i = 0$. Manifold LSPG projection is equivalent to manifold Galerkin projection if and only if the OΔEs characterizing the two methods are equivalent, which is equivalent to requiring the existence of a full-rank matrix $\boldsymbol{A}^n : \mathbb{R}^p \to \mathbb{R}^{p\times p}_\star$ such that

$$\hat{\boldsymbol{r}}^n_G(\hat{\boldsymbol{\xi}}) = \boldsymbol{A}^n(\hat{\boldsymbol{\xi}})\hat{\boldsymbol{r}}^n_L(\hat{\boldsymbol{\xi}}). \tag{4.19}$$

Case 1. We now consider the general (non-asymptotic) case. Eq. (4.19) holds for any sequence of approximate solutions $\hat{\boldsymbol{x}}^n$, $n = 1,\dots,N_t$ if and only if each term in this expansion matches, i.e.,

$$\begin{aligned}
\hat{\boldsymbol{\xi}} =& \boldsymbol{A}^n(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T\boldsymbol{g}(\hat{\boldsymbol{\xi}}) \\
\boldsymbol{J}(\hat{\boldsymbol{\xi}})^+\boldsymbol{f}(\boldsymbol{x}_{\text{ref}}+\boldsymbol{g}(\hat{\boldsymbol{\xi}}),t^n) =& \boldsymbol{A}^n(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T\boldsymbol{f}(\boldsymbol{x}_{\text{ref}}+\boldsymbol{g}(\hat{\boldsymbol{\xi}}),t^n) \\
\hat{\boldsymbol{x}}^{n-j} =& \boldsymbol{A}^n(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T\boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}) \\
\boldsymbol{J}(\hat{\boldsymbol{x}}^{n-j})^+\boldsymbol{f}(\boldsymbol{x}_{\text{ref}}+\boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}),t^{n-j}) =& \boldsymbol{A}^n(\hat{\boldsymbol{\xi}})\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T\boldsymbol{f}(\boldsymbol{x}_{\text{ref}}+\boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}),t^{n-j})
\end{aligned} \tag{4.20}$$

for $j = 1, \ldots, k$. Comparing Conditions (4.20) with (4.9), we see that the requirements are the same as in Theorem 4.1, but with $\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}}; \boldsymbol{\nu})$ replacing $\boldsymbol{\Psi}_G$ and $\boldsymbol{A}^n$ replacing $\boldsymbol{A}_G^n$. Thus, we arrive at the same result: necessary and sufficient conditions for equivalence are that the trial manifold corresponds to an affine subspace such that the decoder satisfies (4.1) and the test basis $\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})$ satisfies

$$\boldsymbol{\Phi}^+ = [\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T \boldsymbol{\Phi}]^{-1} \boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T. \tag{4.21}$$

Note that necessary conditions for Eq. (4.21) to hold are $\mathrm{Ran}(\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})) = \mathrm{Ran}(\boldsymbol{\Phi})$. Examples of test basis that satisfy this condition include $\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}}) = [\boldsymbol{\Phi}^+]^T$ and $\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}}) = \gamma \boldsymbol{\Phi}$ for some nonzero scalar $\gamma \in \mathbb{R}$. However, unlike in Theorem 4.1, we are not free to choose the test basis; rather, it is defined as

$$\boldsymbol{\Psi}^n : \hat{\boldsymbol{\xi}} \mapsto \left(\alpha_0 \boldsymbol{I} - \Delta t \beta_0 \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\xi}} \left(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{\Phi} \hat{\boldsymbol{\xi}}, t^n \right) \right) \boldsymbol{\Phi}. \tag{4.22}$$

For the necessary condition $\mathrm{Ran}(\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})) = \mathrm{Ran}(\boldsymbol{\Phi})$ to hold, the second term in Eq. (4.22) must be zero. This occurs if and only if either (1) an explicit scheme is employed such that $\beta_0 = 0$, or (2) the limit $\Delta t \to 0$ is taken. In both cases, $\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}}) = \gamma \boldsymbol{\Phi}$ holds with $\gamma = \alpha_0$.

Case 2. We now consider asymptotic arguments. Assuming the nonlinear trial manifold is twice continuously differentiable, and $\|\hat{\boldsymbol{x}}^{n-j} - \hat{\boldsymbol{x}}^n\| = O(\Delta t)$, then for $\hat{\boldsymbol{\xi}}$ in a neighborhood of $\hat{\boldsymbol{x}}^n$ such that $\|\hat{\boldsymbol{\xi}} - \hat{\boldsymbol{x}}^n\| = O(\Delta t)$, we obtain the expressions (4.14). Substituting these expressions into the definition of the $\hat{\boldsymbol{r}}_L^n$ in Eq. (4.18), enforcing equivalence of each term in the matching conditions (4.19), and taking the limit $\Delta t \to 0$ yields

$$\begin{aligned}
\hat{\boldsymbol{\xi}} &= \boldsymbol{A}^n(\hat{\boldsymbol{\xi}}) \boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T \boldsymbol{J}(\hat{\boldsymbol{x}}^n) \hat{\boldsymbol{\xi}} \\
\boldsymbol{J}(\hat{\boldsymbol{x}}^n)^+ \boldsymbol{f}(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{g}(\hat{\boldsymbol{\xi}}), t^n) &= \boldsymbol{A}^n(\hat{\boldsymbol{\xi}}) \boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T \boldsymbol{f}(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{g}(\hat{\boldsymbol{\xi}}), t^n) \\
\hat{\boldsymbol{x}}^{n-j} &= \boldsymbol{A}^n(\hat{\boldsymbol{\xi}}) \boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T \boldsymbol{J}(\hat{\boldsymbol{x}}^n) \hat{\boldsymbol{x}}^{n-j} \\
\boldsymbol{J}(\hat{\boldsymbol{x}}^n)^+ \boldsymbol{f}(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}), t^{n-j}) &= \boldsymbol{A}^n(\hat{\boldsymbol{\xi}}) \boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T \boldsymbol{f}(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{g}(\hat{\boldsymbol{x}}^{n-j}), t^{n-j}),
\end{aligned} \tag{4.23}$$

where we have used $\sum_{j=0}^k \alpha_j = 0$. A necessary and sufficient condition for the first and third conditions to hold is

$$\boldsymbol{A}^n(\hat{\boldsymbol{\xi}}) = [\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T \boldsymbol{J}(\hat{\boldsymbol{x}}^n)]^{-1}. \tag{4.24}$$

Then, a necessary and sufficient condition for the second and fourth conditions to hold is

$$\boldsymbol{J}(\hat{\boldsymbol{x}}^n)^+ = [\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T \boldsymbol{J}(\hat{\boldsymbol{x}}^n)]^{-1} \boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})^T. \tag{4.25}$$

As in Case 1 above, a necessary condition for (4.25) to hold is that $\mathrm{Ran}(\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}})) = \mathrm{Ran}(\boldsymbol{J}(\hat{\boldsymbol{x}}^n))$. Due to the definition of the test basis, this requires the second term on the right-hand side of Eq. (3.19) to be zero. This is already satisfied by the stated assumption that the limit $\Delta t \to 0$ is taken, in which case $\boldsymbol{\Psi}^n(\hat{\boldsymbol{\xi}}) = \gamma \boldsymbol{J}(\hat{\boldsymbol{\xi}})$ holds with $\gamma = \alpha_0$. It can be easily verified that this test basis satisfies the condition (4.25). □

We now derive *a posteriori* local discrete-time error bounds for both the manifold Galerkin and manifold LSPG projection methods in the context of linear multistep methods, and demonstrate that manifold LSPG projection sequentially minimizes the error bound in time. For notational simplicity, we suppress the second argument of the velocity $\boldsymbol{f}$, as the time superscript appearing in the second argument always matches that of the first argument.

Theorem 4.3 (Error bound)**.** If the velocity $\boldsymbol{f}$ is Lipschitz continuous, i.e., there exists a constant $\kappa > 0$ such that $\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{y})\|_2 \leq \kappa \|\boldsymbol{x} - \boldsymbol{y}\|_2$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$, and the time step is sufficiently small such that $\Delta t < |\alpha_0|/|\beta_0| \, \kappa$, then

$$\|\boldsymbol{x}^n - \boldsymbol{x}_{\mathrm{ref}} - \boldsymbol{g}(\hat{\boldsymbol{x}}_G^n)\|_2 \leq \frac{1}{h} \|\tilde{\boldsymbol{r}}^n \left(\boldsymbol{x}_{\mathrm{ref}} + \boldsymbol{g}(\hat{\boldsymbol{x}}_G^n) \right) \|_2 + \sum_{j=1}^k \gamma_j \|\boldsymbol{x}^{n-j} - \boldsymbol{x}_{\mathrm{ref}} - \boldsymbol{g}(\hat{\boldsymbol{x}}_G^{n-j})\|_2 \tag{4.26}$$

$$\|\boldsymbol{x}^n - \boldsymbol{x}_{\mathrm{ref}} - \boldsymbol{g}(\hat{\boldsymbol{x}}_P^n)\|_2 \leq \frac{1}{h} \min_{\boldsymbol{\xi} \in \boldsymbol{x}_{\mathrm{ref}} + \mathcal{S}} \|\tilde{\boldsymbol{r}}^n \left(\boldsymbol{\xi} \right) \|_2 + \sum_{j=1}^k \gamma_j \|\boldsymbol{x}^{n-j} - \boldsymbol{x}_{\mathrm{ref}} - \boldsymbol{g}(\hat{\boldsymbol{x}}_P^{n-j})\|_2, \tag{4.27}$$

where $\hat{\boldsymbol{x}}_G^n$ denotes the manifold Galerkin solution satisfying (3.9) and $\hat{\boldsymbol{x}}_P^n$ denotes the manifold LSPG solution satisfying (3.17). Here, $h := |\alpha_0| - |\beta_0|\,\kappa\Delta t$ and $\gamma_j := (|\alpha_j| + |\beta_j|\,\kappa\Delta t)\,/h$.

Proof. We begin by defining linear multistep residual operators associated with the FOM sequence of solutions $\{\boldsymbol{x}^j\}_{j=1}^n$ and an (arbitrary) sequence of approximate solutions $\{\tilde{\boldsymbol{x}}^j\}_{j=1}^n$, i.e.,

$$\boldsymbol{r}_\star^n : \boldsymbol{\xi} \mapsto \alpha_0\boldsymbol{\xi} - \Delta t\beta_0\boldsymbol{f}(\boldsymbol{\xi}) + \sum_{j=1}^k \alpha_j\boldsymbol{x}^{n-j} - \Delta t\sum_{j=1}^k \beta_j\boldsymbol{f}(\boldsymbol{x}^{n-j}) \tag{4.28}$$

$$\tilde{\boldsymbol{r}}^n : \boldsymbol{\xi} \mapsto \alpha_0\boldsymbol{\xi} - \Delta t\beta_0\boldsymbol{f}(\boldsymbol{\xi}) + \sum_{j=1}^k \alpha_j\tilde{\boldsymbol{x}}^{n-j} - \Delta t\sum_{j=1}^k \beta_j\boldsymbol{f}(\tilde{\boldsymbol{x}}^{n-j}). \tag{4.29}$$

We note that $\tilde{\boldsymbol{x}}^j = \boldsymbol{x}_{\text{ref}} + \boldsymbol{g}(\hat{\boldsymbol{x}}_G^j)$, $j = 1, \ldots, n$ in the case of manifold Galerkin projection, and $\tilde{\boldsymbol{x}}^j = \boldsymbol{x}_{\text{ref}} + \boldsymbol{g}(\hat{\boldsymbol{x}}_P^j)$, $j = 1, \ldots, n$ in the case of manifold LSPG projection. Subtracting $\tilde{\boldsymbol{r}}^n\left(\tilde{\boldsymbol{x}}^n\right)$ from $\boldsymbol{r}_\star^n\left(\boldsymbol{x}^n\right)$ and noting from Eq. (2.2) that $\boldsymbol{r}_\star^n\left(\boldsymbol{x}^n\right) = \boldsymbol{0}$ yields

$$-\tilde{\boldsymbol{r}}^n\left(\tilde{\boldsymbol{x}}^n\right) = \alpha_0\left(\boldsymbol{x}^n - \tilde{\boldsymbol{x}}^n\right) - \Delta t\beta_0\left(\boldsymbol{f}(\boldsymbol{x}^n) - \boldsymbol{f}(\tilde{\boldsymbol{x}}^n)\right) + \sum_{j=1}^k \alpha_j\left(\boldsymbol{x}^{n-j} - \tilde{\boldsymbol{x}}^{n-j}\right) - \Delta t\sum_{j=1}^k \beta_j\left(\boldsymbol{f}(\boldsymbol{x}^{n-j}) - \boldsymbol{f}(\tilde{\boldsymbol{x}}^{n-j})\right).$$

Rearranging this expression gives

$$\begin{aligned}&\underbrace{\boldsymbol{x}^n - \tilde{\boldsymbol{x}}^n - \frac{\Delta t\beta_0}{\alpha_0}\left(\boldsymbol{f}(\boldsymbol{x}^n) - \boldsymbol{f}(\tilde{\boldsymbol{x}}^n)\right)}_{\text{(I)}} = \\ &\qquad \underbrace{-\frac{1}{\alpha_0}\tilde{\boldsymbol{r}}^n\left(\tilde{\boldsymbol{x}}^n\right) - \frac{1}{\alpha_0}\sum_{j=1}^k \alpha_j\left(\boldsymbol{x}^{n-j} - \tilde{\boldsymbol{x}}^{n-j}\right) + \frac{\Delta t}{\alpha_0}\sum_{j=1}^k \beta_j\left(\boldsymbol{f}(\boldsymbol{x}^{n-j}) - \boldsymbol{f}(\tilde{\boldsymbol{x}}^{n-j})\right)}_{\text{(II)}}.\end{aligned} \tag{4.30}$$

We proceed by bounding $\|\text{(I)}\|_2$ from below, and $\|\text{(II)}\|_2$ from above. To bound $\|\text{(I)}\|_2$ from below, we apply the reverse triangle to obtain

$$\|\text{(I)}\|_2 \geq \left| \|\boldsymbol{x}^n - \tilde{\boldsymbol{x}}^n\|_2 - \| \underbrace{\frac{\Delta t\beta_0}{\alpha_0}\left(\boldsymbol{f}(\boldsymbol{x}^n) - \boldsymbol{f}(\tilde{\boldsymbol{x}}^n)\right)}_{\text{(I.a)}} \|_2 \right|. \tag{4.31}$$

We now use Lipschitz continuity of the velocity $\boldsymbol{f}$ to bound $\|\text{(I.a)}\|_2$ from above as

$$\|\text{(I.a)}\|_2 \leq \frac{\Delta t\,|\beta_0|\,\kappa}{|\alpha_0|}\|\boldsymbol{x}^n - \tilde{\boldsymbol{x}}^n\|_2. \tag{4.32}$$

If time-step-restriction condition $\Delta t < |\alpha_0|/|\beta_0|\,\kappa$ holds, then we can combine inequalities (4.31) and (4.32) as

$$\|\text{(I)}\|_2 \geq \frac{h}{|\alpha_0|}\|\boldsymbol{x}^n - \tilde{\boldsymbol{x}}^n\|_2. \tag{4.33}$$

To bound $\|\text{(II)}\|_2$ in Eq. (4.30) from above, we apply the triangle inequality and employ Lipschitz continuity of the velocity $\boldsymbol{f}$, which gives

$$\|\text{(II)}\|_2 \leq \frac{1}{|\alpha_0|}\|\tilde{\boldsymbol{r}}^n\left(\tilde{\boldsymbol{x}}^n\right)\|_2 + \frac{1}{|\alpha_0|}\sum_{j=1}^k \left(|\alpha_j| + |\beta_j|\,\kappa\Delta t\right)\|\boldsymbol{x}^{n-j} - \tilde{\boldsymbol{x}}^{n-j}\|_2. \tag{4.34}$$

Combining Eq. (4.30) with inequalities (4.33) and (4.34) gives

$$\|\boldsymbol{x}^n - \tilde{\boldsymbol{x}}^n\|_2 \leq \frac{1}{h}\|\tilde{\boldsymbol{r}}^n\left(\tilde{\boldsymbol{x}}^n\right)\|_2 + \sum_{j=1}^{k} \gamma_j \|\boldsymbol{x}^{n-j} - \tilde{\boldsymbol{x}}^{n-j}\|_2. \tag{4.35}$$

The manifold-Galerkin error bound (4.26) results from substituting $\tilde{\boldsymbol{x}}^j = \boldsymbol{x}_{\text{ref}} + \boldsymbol{g}(\hat{\boldsymbol{x}}_G^j)$, $j = 1, \ldots, n$ in (4.35), while the manifold-LSPG error bound (4.27) results from substituting $\tilde{\boldsymbol{x}}^j = \boldsymbol{x}_{\text{ref}} + \boldsymbol{g}(\hat{\boldsymbol{x}}_P^j)$, $j = 1, \ldots, n$ in (4.35) and noting that the manifold LSPG solution $\boldsymbol{x}_{\text{ref}} + \boldsymbol{g}(\hat{\boldsymbol{x}}_P^j)$ satisifies the time-discrete residual-minimization property (3.17). □

This result illustrates that the time-discrete residual minimization property of manifold LSPG projection allows its approximation to sequentially minimize the error bound, as the first term on the right-hand side of bound (4.27) corresponds to the new contribution to the error bound at time instance t^n, while the second term on the right-hand side comprises the recursive term in the bound.

5. Nonlinear trial manifold based on deep convolutional autoencoders

This section describes the approach we propose for constructing the decoder $\boldsymbol{g} : \mathbb{R}^p \to \mathbb{R}^N$ that defines the nonlinear trial manifold. As described in the introduction, any nonlinear-manifold learning method equipped with a continuously differentiable mapping from the generalized coordinates (i.e., the latent state) to an approximation of the state (i.e., the data) is compatible with the manifold Galerkin and manifold LSPG methods proposed in Section 3. Here, we pursue deep convolutional autoencoders for this purpose, as they are very expressive and scalable; there is also high-performance software available for their construction. When the nonlinear trial manifold is constructed using the proposed deep convolutional autoencoder, we refer to the Manifold Galerkin ROM as the *Deep Galerkin* ROM and the Manifold LSPG ROM as the *Deep LSPG* ROM.[1]

Section 5.1 describes the mathematical structure of autoencoders, which provides a neural-network model for the decoder $\boldsymbol{g}(\cdot; \boldsymbol{\theta}) : \mathbb{R}^p \to \mathbb{R}^N$, where $\boldsymbol{\theta}$ denotes the neural-network parameters to be computed during training; Section 5.2 describes the proposed autoencoder architecture, which is tailored to spatially distributed dynamical-system states; Section 5.3 describes how we propose to satisfy the initial condition using the proposed autoencoder in line with Remark 3.1.

5.1. Autoencoder

An autoencoder is a type of feedforward neural network that aims to learn the identity mapping, i.e., $\boldsymbol{h} : \boldsymbol{x} \mapsto \tilde{\boldsymbol{x}}$ with $\tilde{\boldsymbol{x}} \approx \boldsymbol{x}$ and $\boldsymbol{h} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$. Autoencoders achieve this using an architecture composed of two parts: the encoder $\boldsymbol{h}_{\text{enc}} : \boldsymbol{x} \mapsto \hat{\boldsymbol{x}}$ with $\boldsymbol{h}_{\text{enc}} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_{\hat{x}}}$ and $n_{\hat{x}} \ll n_x$, which maps a high-dimensional vector $\boldsymbol{x}$ to a low-dimensional *code* $\hat{\boldsymbol{x}}$; and the decoder $\boldsymbol{h}_{\text{dec}} : \hat{\boldsymbol{x}} \mapsto \tilde{\boldsymbol{x}}$ with $\boldsymbol{h}_{\text{dec}} : \mathbb{R}^{n_{\hat{x}}} \to \mathbb{R}^{n_x}$, which maps the code $\hat{\boldsymbol{x}}$ to an approximation of the original high-dimensional vector $\tilde{\boldsymbol{x}}$. Thus, the resulting autoencoder takes the form

$$\boldsymbol{h} : \boldsymbol{x} \mapsto \boldsymbol{h}_{\text{dec}} \circ \boldsymbol{h}_{\text{enc}}(\boldsymbol{x}).$$

If $\boldsymbol{h}(\boldsymbol{x}) \approx \boldsymbol{x}$ over a data set $\boldsymbol{x} \in \{\boldsymbol{x}^{(i)}\}_i$, then the low-dimensional codes $\boldsymbol{h}_{\text{enc}}(\boldsymbol{x}^{(i)})$ contain sufficient information to recover accurate approximations of the data $\tilde{\boldsymbol{x}}^{(i)} \approx \boldsymbol{x}^{(i)}$ via the application of the decoder $\boldsymbol{h}_{\text{dec}}$. In this work, we propose to employ the autoencoder decoder $\boldsymbol{h}_{\text{dec}}$ for the decoder $\boldsymbol{g}$ used to define the nonlinear trial manifold introduced in Section 3.1.

[1] This is analogous to the naming convention employed for linear-subspace ROMs, as linear-subspace Galerkin projection with POD is referred to as a POD–Galerkin ROM.

In feedforward networks, each network layer typically corresponds to a vector or tensor, whose values are computed by applying an affine transformation to the previous layer followed by a nonlinear activation function. An encoder with n_{L} layers takes the form

$$\boldsymbol{h}_{\mathrm{enc}} : (\boldsymbol{x}; \boldsymbol{\theta}_{\mathrm{enc}}) \mapsto \boldsymbol{h}_{n_{\mathrm{L}}}(\cdot; \boldsymbol{\Theta}_{n_{\mathrm{L}}}) \circ \boldsymbol{h}_{n_{\mathrm{L}}-1}(\cdot; \boldsymbol{\Theta}_{n_{\mathrm{L}}-1}) \circ \cdots \circ \boldsymbol{h}_1(\boldsymbol{x}; \boldsymbol{\Theta}_1),$$

where $\boldsymbol{h}_i(\cdot; \boldsymbol{\Theta}_i) : \mathbb{R}^{p_{i-1}} \to \mathbb{R}^{p_i}$, $i = 1, \ldots, n_{\mathrm{L}}$ denotes the function applied at layer i of the neural network; $\boldsymbol{\Theta}_i$, $i = 1, \ldots, n_{\mathrm{L}}$ denotes the weights and the biases employed at layer i with $\boldsymbol{\theta}_{\mathrm{enc}} \equiv (\boldsymbol{\Theta}_1, \ldots, \boldsymbol{\Theta}_{n_{\mathrm{L}}})$; and p_i, $i = 1, \ldots, n_{\mathrm{L}}$ denotes the dimensionality of the output at layer i. The input has dimension $p_0 = n_x$ and the final layer produces a code with dimension $p_{n_{\mathrm{L}}} = n_{\hat{x}}$. The nonlinear activation function is applied in layers 1 to n_{L} to a function of the weights, biases, and the outputs from the previous layer such that

$$\boldsymbol{h}_i : (\boldsymbol{x}; \boldsymbol{\Theta}_i) \mapsto \boldsymbol{\phi}_i(h_i(\boldsymbol{\Theta}_i, \boldsymbol{x})),$$

where $\boldsymbol{\phi}_i$ is an element-wise nonlinear activation function. For fully-connected layers as in the traditional multilayer perceptron (MLP), $h_i(\boldsymbol{\Theta}_i, \boldsymbol{x}) = \boldsymbol{\Theta}_i[1, \boldsymbol{x}^T]^T$ with $\boldsymbol{\Theta}_i \in \mathbb{R}^{p_i \times (p_{i-1}+1)}$ a real-valued matrix. For convolutional layers, h_i corresponds to a convolution operator with $\boldsymbol{\Theta}_i$ providing the convolutional-filter weights.

A decoder with $\bar{n}_{\mathrm{L}}$ layers also corresponds to a feedforward network; it takes the form

$$\boldsymbol{h}_{\mathrm{dec}} : (\hat{\boldsymbol{x}}; \boldsymbol{\theta}_{\mathrm{dec}}) \mapsto \bar{\boldsymbol{h}}_{\bar{n}_{\mathrm{L}}}(\cdot; \bar{\boldsymbol{\Theta}}_{\bar{n}_{\mathrm{L}}}) \circ \bar{\boldsymbol{h}}_{\bar{n}_{\mathrm{L}}-1}(\cdot; \bar{\boldsymbol{\Theta}}_{\bar{n}_{\mathrm{L}}-1}) \circ \cdots \circ \bar{\boldsymbol{h}}_1(\hat{\boldsymbol{x}}; \bar{\boldsymbol{\Theta}}_1)$$

with $\bar{\boldsymbol{h}}_i(\cdot; \bar{\boldsymbol{\Theta}}_i) : \mathbb{R}^{\bar{p}_{i-1}} \to \mathbb{R}^{\bar{p}_i}$, $i = 1, \ldots, \bar{n}_{\mathrm{L}}$, and $\boldsymbol{\theta}_{\mathrm{dec}} \equiv (\bar{\boldsymbol{\Theta}}_1, \ldots, \bar{\boldsymbol{\Theta}}_{\bar{n}_{\mathrm{L}}})$. The dimension of the input is $\bar{p}_0 = n_{\hat{x}}$ and the dimension of the final layer (i.e., the output layer) is $\bar{p}_{\bar{n}_{\mathrm{L}}} = n_x$. Again, the nonlinear activation is applied to the output of the previous layer such that

$$\bar{\boldsymbol{h}}_i : (\boldsymbol{x}; \bar{\boldsymbol{\Theta}}_i) \mapsto \bar{\boldsymbol{\phi}}_i(\bar{h}_i(\bar{\boldsymbol{\Theta}}_i, \boldsymbol{x})).$$

As in the encoder, $\bar{h}_i(\bar{\boldsymbol{\Theta}}_i, \boldsymbol{x}) = \bar{\boldsymbol{\Theta}}_i[1, \boldsymbol{x}^T]^T$ with $\bar{\boldsymbol{\Theta}}_i \in \mathbb{R}^{\bar{p}_i \times (\bar{p}_{i-1}+1)}$ a real-valued matrix for fully-connected layers, while $\bar{h}_i$ corresponds to a convolution operator with $\bar{h}_i$ providing transposed convolutional-filter weights for convolutional layers.

Because MLP autoencoders are fully connected, the number of parameters (corresponding to the edge weights and biases) can be extremely large when the number of inputs n_x is large; in such scenarios, these models typically require a large amount of training data. As this work aims to enable model reduction for large-scale dynamical systems, directly applying an MLP autoencoder to the state such that $n_x = N$ is not practical in many scenarios. To address this, alternative neural-network architectures have been devised that make use of *parameter sharing* to reduce the total number of parameters in the model, and thus the amount of data needed for training. In the context of dynamical systems characterized by a state that can be represented as spatially distributed data, we propose to apply *convolutional autoencoders*. Such methods are applicable to multi-channel spatially distributed input data and employ parameter sharing such that they can be trained with less data. Further, such models tend to generalize well to unseen test data [48, 49] because they exploit three key properties of natural signals: local connectivity, parameter sharing, and equivariance to translation [32, 48].

5.2. Deep convolutional autoencoders for spatially distributed states

Many dynamical systems are characterized by a state that can be represented as spatially distributed data, e.g., spatially discretized partial-differential-equations models. In such cases, there is a *restriction operator* $\boldsymbol{R}$ that maps the state to a tensor representing spatially distributed data, i.e.,

$$\boldsymbol{R} : \mathbb{R}^N \to \mathbb{R}^{n_1 \times \cdots \times n_d \times n_{\mathrm{chan}}}, \tag{5.1}$$

where n_i denotes the number of discrete points in spatial dimension i; $d \in \{1, 2, 3\}$ denotes the spatial dimension, and n_{chan} denotes the number of *channels*. For images, typically $n_{\mathrm{chan}} = 3$ (i.e., red, green, and blue). For dynamical system models the number of channels n_{chan} is equal to the number of state variables

defined at a given spatial location; for example, n_{chan} is equal to the number of conserved variables when the dynamical system arises from the spatial discretization of a conservation law. We also write the associated *prolongation operator*, which aims to provide the inverse mapping such that

$$\boldsymbol{P} : \mathbb{R}^{n_1 \times \cdots \times n_d \times n_{\text{chan}}} \to \mathbb{R}^N. \tag{5.2}$$

For coarse discretizations of the spatial domain, it is possible to ensure $\boldsymbol{R} \circ \boldsymbol{P}(\cdot)$ corresponds to the identity map; for fine discretizations, it is possible to ensure $\boldsymbol{P} \circ \boldsymbol{R}(\cdot)$ corresponds to the identity map; for cases where underlying grid provides an isomorphic representation of the state, it is possible to achieve both [18].

After reformatting the state from a vector to a tensor by applying the restriction operator $\boldsymbol{R}$, we apply an invertible affine scaling operator $\boldsymbol{S} : \mathbb{R}^{n_1 \times \cdots \times n_d \times n_{\text{chan}}} \to \mathbb{R}^{n_1 \times \cdots \times n_d \times n_{\text{chan}}}$ for data-standardization purposes. Section 6.2 defines the specific elements of this operator, which are computed from the training data.

Figure 1 depicts the architecture of the proposed deep convolutional autoencoder. Before the first convolutional layer, the autoencoder applies the restriction operator $\boldsymbol{R}$ and scaling operator $\boldsymbol{S}$, while after the last convolutional layer, the autoencoder applies the inverse scaling operator $\boldsymbol{S}^{-1}$ and prolongation operator $\boldsymbol{P}$. We note that the restriction and prolongation operators are (deterministic) operations that simply reshape the state into an appropriate format for the autoencoder, while the scaling operator is defined explicitly from the range of the training data; thus, these quantities are not subject to training. We consider the combination of convolutional layers (gray boxes) and fully-connected layers (blue rectangles). Appendix B provides a description of convolutional layers, including hyperparameters and parameters that are subject to optimization. The encoder network is composed of restriction and scaling, followed by a sequence of n_{conv} convolutional layers and n_{full} fully-connected layers. The decoder network is composed of a sequence of n_{full} fully-connected layers and n_{conv} transposed-convolutional layers with no nonlinear activation in the last layer, followed by inverse scaling and prolongation. As a result, the dimension of the input to the first convolutional layer is $p_0 = n_{\text{chan}} \prod_{i=1}^{d} n_i$. The dimension $p_{n_{\text{conv}}}$ of the output of encoder layer n_{conv} (i.e., the final convolutional layer) is determined by the hyperparameters defining the kernels used in the convolutional layers (i.e., depth, stride, zero-padding). Similarly, the dimension $\bar{p}_{n_{\text{full}}}$ of the output of decoder layer n_{full} (i.e., the final fully connected layer) is determined by the hyperparameters defining the kernels in the subsequent convolutional layers. The dimension of the output of the final decoder layer is $\bar{p}_{n_{\text{L}}} = p_0 = n_{\text{chan}} \prod_{i=1}^{d} n_i$.

A particular instance of the network architecture can be defined by specifying the number of convolutional layers n_{conv}, the number of fully-connected layers n_{full}, the number of units in each layer, and the types of nonlinear activations. Such architecture-related parameters are typically considered hyperparameters, as they are not subject to optimization during training.

We propose to set the decoder for the proposed manifold ROMs to the decoder arising from the deep convolutional autoencoder architecture described in Figure 1, i.e., $\boldsymbol{g} = \boldsymbol{h}_{\text{dec}}$. When the Manifold Galerkin and Manifold LSPG ROMs employ this choice of decoder, we refer to them as *Deep Galerkin* and *Deep LSPG* ROMs, respectively.

5.3. Initial condition satisfaction

As discussed in Remark 3.1, the initial generalized coordinates $\hat{\boldsymbol{x}}^0(\boldsymbol{\mu})$ can be ensured to satisfy the initial conditions by employing a reference state defined by Eq. (3.4); however, this implies that the decoder must be able to accurately represent deviations from this reference state. To accomplish this for the proposed autoencoder, we propose (1) to train the autoencoder with snapshot data centered on the initial condition along with the zero vector (as described in Section 6.1), and (2) to set the initial generalized coordinates to an encoding of zero, i.e., $\hat{\boldsymbol{x}}^0(\boldsymbol{\mu}) = \boldsymbol{h}_{\text{enc}}(\boldsymbol{0})$ for all $\boldsymbol{\mu} \in \mathcal{D}$. Then, setting the reference state according to Eq. (3.4) leads to a reference state of $\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu}) - \boldsymbol{g}(\boldsymbol{h}_{\text{enc}}(\boldsymbol{0}))$. Further, if $\boldsymbol{h}_{\text{dec}}(\boldsymbol{h}_{\text{enc}}(\boldsymbol{0})) \approx \boldsymbol{0}$, as is encouraged by including the zero vector in training, then the reference state comprises a perturbation of the initial condition, and the decoder $\boldsymbol{h}_{\text{dec}}$ must only represent deviations from this perturbation. This is consistent with the training of the autoencoder, as the snapshots have been centered on the initial condition.

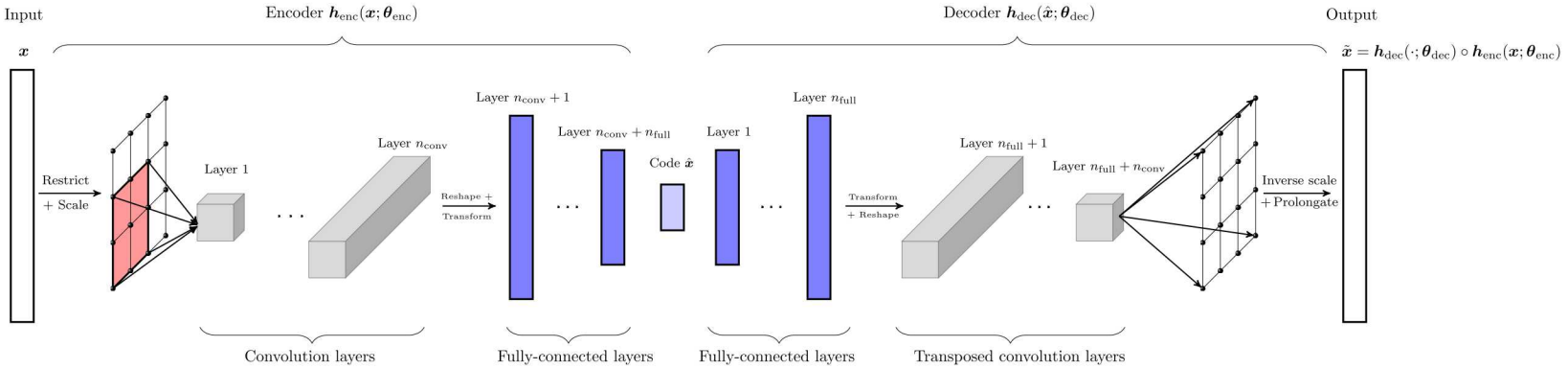


Figure 1: Network architecture of the deep convolutional autoencoder, which takes a state of dynamical systems as an input and produces an approximated state as an output. The encoder extracts low-dimensional code $\hat{\boldsymbol{x}}$ from the discrete representation of the state $\boldsymbol{x}$ by applying the restriction operator (Eq. (5.1)) and a set of convolutional layers (gray boxes), followed by a set of fully-connected layers (blue rectangles). The decoder approximately reconstructs the high-dimensional vector $\boldsymbol{h}(\boldsymbol{x};\boldsymbol{\theta})$ by performing the inverse operations of the encoder, applying fully-connected layers (blue rectangles), followed by the transposed-convolutional layers (gray boxes), then applying the prolongation operator (Eq. (5.2)) to the resulting quantity.

6. Offline training

This section describes the offline training process used to train the deep convolutional autoencoder proposed in Section 5.2. The approach employs precisely the same snapshot data used by POD. Section 6.1 describes the (snapshot-based) data collection procedure, which is identical to that employed by POD. Section 6.2 describes how the data are scaled to improve numerical stability of autoencoder training. Section 6.3 summarizes the gradient-based-optimization approach employed for training the autoencoder (i.e., computing the optimal parameters $\boldsymbol{\theta}^\star$) given the scaled training data. Algorithm 1 provides a summary of offline training.

Algorithm 1 Offline training

Input: Training-parameter instances $\mathcal{D}_{\text{train}}$; restriction operator $\boldsymbol{R}$; prolongation operator $\boldsymbol{P}$; autoencoder architecture; SGD hyperparameters (adaptive learning-rate strategy; initial parameters $\boldsymbol{\theta}^{(0)}$; number of minibatches n_{batch}; maximum number of epochs n_{epoch}; early-stopping criterion)

Output: Encoder $\boldsymbol{h}_{\text{enc}}$; decoder $\boldsymbol{g}$

1: Solve the FOM OΔE (2.2) for $\boldsymbol{\mu} \in \mathcal{D}_{\text{train}}$ and form the snapshot matrix $\boldsymbol{W}$ (Eq. (6.1)).
2: Compute the scaling operator $\boldsymbol{S}$ (Eq. (6.4)), which completes the definition of the autoencoder $\boldsymbol{h}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{h}_{\text{dec}}(\cdot;\boldsymbol{\theta}_{\text{dec}}) \circ \boldsymbol{h}_{\text{enc}}(\boldsymbol{x};\boldsymbol{\theta}_{\text{enc}})$.
3: Train the autoencoder by executing Algorithm 2 in Appendix A with inputs corresponding to the snapshot matrix $\boldsymbol{W}$; the autoencoder $\boldsymbol{h}(\boldsymbol{x};\boldsymbol{\theta})$; and SGD hyperparameters. This returns the encoder $\boldsymbol{h}_{\text{enc}}$ and decoder $\boldsymbol{g}$.

6.1. Snapshot-based data collection

The first step of offline training is snapshot-based data collection. This requires solving the FOM OΔE (2.2) for training-parameter instances $\boldsymbol{\mu} \in \mathcal{D}_{\text{train}} \equiv \{\boldsymbol{\mu}^i_{\text{train}}\}_{i=1}^{n_{\text{train}}} \subset \mathcal{D}$ and assembling the snapshot matrix

$$\boldsymbol{W} := \left[\boldsymbol{W}(\boldsymbol{\mu}^1_{\text{train}}) \;\cdots\; \boldsymbol{W}(\boldsymbol{\mu}^{n_{\text{train}}}_{\text{train}})\right] \in \mathbb{R}^{N \times n_{\text{snap}}} \tag{6.1}$$

with $n_{\text{snap}} := N_t n_{\text{train}}$ and $\boldsymbol{W}(\boldsymbol{\mu}) := [\boldsymbol{x}^1(\boldsymbol{\mu}) - \boldsymbol{x}^0(\boldsymbol{\mu}) \;\cdots\; \boldsymbol{x}^{N_t}(\boldsymbol{\mu}) - \boldsymbol{x}^0(\boldsymbol{\mu})]$.

Remark 6.1 (Proper orthogonal decomposition)**.** POD employs the snapshot matrix $\boldsymbol{W}$ to compute a trial basis matrix $\boldsymbol{\Phi}$ used to define an affine trial subspace $\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \text{Ran}(\boldsymbol{\Phi})$. To do so, POD computes the singular value decomposition (SVD) and sets the trial basis matrix to be equal to the first p left singular vectors, i.e.,

$$\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T, \qquad \boldsymbol{\phi}_i = \boldsymbol{u}_i, \;\; i = 1, \ldots, p. \tag{6.2}$$

The resulting POD trial basis matrix $\boldsymbol{\Phi}$ satisfies the minimization problem

$$\underset{\boldsymbol{\Phi} \in V_k(\mathbb{R}^N)}{\text{minimize}} \sum_{i=1}^{n_{\text{snap}}} \|\boldsymbol{w}^i - \boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{w}^i\|_2^2, \tag{6.3}$$

where $\boldsymbol{w}^i$ denotes the ith column of $\boldsymbol{W}$ and $V_k(\mathbb{R}^n)$ and denotes the set of orthogonal $k \times n$ matrices (the compact Stiefel manifold). The solution is unique up to a rotation. This technique is equivalent (up to the data-centering process) to principal component analysis [40], an unsupervised machine-learning method for linear dimensionality reduction.

6.2. Data standardization

As described in Section 5.2, the first layer of the proposed autoencoder applies a restriction operator $\boldsymbol{R}$, which reformats the input vector into a tensor compatible with convolutional layers, followed by an affine scaling operator $\boldsymbol{S}$; the last layer applies the inverse of this scaling operator $\boldsymbol{S}^{-1}$ and subsequently applies the prolongation operator $\boldsymbol{P}$ to reformat the data into a vector. We now define the scaling operator from the training data to ensure that all elements of the training data lie between zero and one. This scaling improves numerical stability of the gradient-based optimization for training [41, 71], which will be described in Section

6.3. We adopt a standard scaling procedure also followed, e.g., by Ref. [31]. Namely, defining the restriction of the ith snapshot as $\mathcal{W}^i := \boldsymbol{R}(\boldsymbol{w}^i) \in \mathbb{R}^{n_1 \times \cdots \times n_d \times n_{\text{chan}}}$, we set the elements of the scaling operator $\boldsymbol{S}$ to

$$s_{i_1 \cdots i_d j} : x \mapsto \frac{x - \mathcal{W}_j^{\min}}{\mathcal{W}_j^{\max} - \mathcal{W}_j^{\min}} \tag{6.4}$$

with

$$\mathcal{W}_j^{\max} := \max_{k \in \{1,\ldots,n_{\text{snap}}\},\, i_1 \in \{1,\ldots,n_1\},\ldots,\, i_d \in \{1,\ldots,n_d\}} \mathcal{W}^k_{i_1 \cdots i_d j}, \quad \text{and}$$
$$\mathcal{W}_j^{\min} := \min_{k \in \{1,\ldots,n_{\text{snap}}\},\, i_1 \in \{1,\ldots,n_1\},\ldots,\, i_d \in \{1,\ldots,n_d\}} \mathcal{W}^k_{i_1 \cdots i_d j}.$$

6.3. Autoencoder training

Once the autoencoder architecture has been defined (including the restriction, prolongation and scaling operators), it takes the form $\boldsymbol{h}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{h}_{\text{dec}}(;\boldsymbol{\theta}_{\text{dec}}) \circ \boldsymbol{h}_{\text{enc}}(\boldsymbol{x};\boldsymbol{\theta}_{\text{enc}})$, where the undefined parameters $\boldsymbol{\theta}$ correspond to convolutional-filter weights and weights and biases for the fully connected layers. We compute optimal values of these parameters $\boldsymbol{\theta}^\star$ using a standard approach from deep learning: stochastic gradient descent (SGD) with minibatching and early stopping [9]. Appendix A provides additional details, where Algorithm 2 provides the training algorithm.

7. Numerical experiments

This section assesses the performance of the proposed Deep Galerkin and Deep LSPG ROMs, which employ nonlinear trial manifolds, compared to POD–Galerkin and POD–LSPG ROMs, which employ affine trial subspaces. We consider two advection-dominated benchmark problems: 1D Burgers' equation and a chemically reacting flow. We employ the numerical PDE tools and ROM functionality provided by `pyMORTestbed` [81], and we construct the autoencoder using `TensorFlow` [1].

For both benchmark problems, the Deep Galerkin and Deep LSPG ROMs employ a 10-layer convolutional autoencoder corresponding to the architecture depicted in Figure 1. The encoder $\boldsymbol{h}_{\text{enc}}$ consists of $n_{\text{L}} = 5$ layers with $n_{\text{conv}} = 4$ convolutional layers, followed by $n_{\text{full}} = 1$ fully-connected layer. The decoder $\boldsymbol{h}_{\text{dec}}$ consists of $n_{\text{full}} = 1$ fully-connected layer, followed by $n_{\text{conv}} = 4$ transposed-convolution layers. The latent code of the autoencoder is of dimension p (i.e., $p_{n_{\text{L}}} = \bar{p}_0 = p$), which will vary during the experiments to define different reduced-state dimensions. Table 1 specifies attributes of the kernels used in convolutional and transposed-convolutional layers.

For the nonlinear activation functions $\boldsymbol{\phi}_i$, $i = 1, \ldots, n_{\text{L}}$ and $\bar{\boldsymbol{\phi}}_i$, $i = 1, \ldots, \bar{n}_{\text{L}} - 1$, we use exponential linear units (ELU) [20], which is defined as

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{cases} \boldsymbol{x} & \text{if } \boldsymbol{x} \geq 0 \\ \exp(\boldsymbol{x}) - 1 & \text{otherwise} \end{cases},$$

and an identity activation function in the output at layer $\bar{n}_{\text{L}}$ in decoder (as is common practice). This choice of activation ensures that the resulting decoder $\boldsymbol{g}$ is continuously differentiable everywhere, and is twice continuously differentiable almost everywhere. We employ the ℓ^2-loss function defined in Eq. (A.2) in the minimization problem (A.1), which is equivalent to the loss function minimized by POD (see Remark 6.1). We apply the Adam optimizer [44], which is compatible with the stochastic gradient descent (SGD) algorithm reported in Algorithm 2; here, the adaptive learning rate strategy computes rates for different parameters using estimates of first and second moments of the gradients.

Using the same snapshots as that to train the autoencoder, we also compute a POD basis $\boldsymbol{\Phi}$ following the steps discussed in Remark 6.1.

We compare the performance of four ROMs: 1) POD–Galerkin: linear-subspace Galerkin projection (Eq. (4.4)) with the POD basis defining $\boldsymbol{\Phi}$, 2) POD-LSPG: linear-subspace LSPG projection (Eq. (4.6)) with the POD basis defining $\boldsymbol{\Phi}$, 3) Deep Galerkin: manifold Galerkin projection (Eq. (3.8)) with the deep convolutional

Table 1: Parameters of the autoencoder architecture described in Figure 1 applied to both benchmark problems. The encoder consists of $n_{\mathrm{L}} = 5$ layers with $n_{\mathrm{conv}} = 4$ convolutional layers, followed by $n_{\mathrm{full}} = 1$ fully-connected layers. The decoder consists of $n_{\mathrm{full}} = 1$ fully-connected layers, followed by $n_{\mathrm{conv}} = 4$ transposed-convolution layers. For the parameterized 1D Burgers' equation, we employ 1D convolutional kernel filters with the kernel length 25 at every convolutional layer and transposed-convolutional layer, and apply length-2 stride ($s = 2$) encoder layer 1 and the decoder layer 5, and length-4 stride ($s = 4$) for other convolutional layers and transposed-convolutional layers. For the chemically reacting flow, we employ 5×5 convolutional kernel filters at every convolutional layer and transposed-convolutional layer, and apply length-2 stride ($s = 2$). For the definition of stride, we refer to Appendix B. For zero-padding, we use half padding [28, 32]. With these settings, $p_{n_{\mathrm{conv}}} = \bar{p}_{n_{\mathrm{full}}} = 128$ for the 1D Burgers' equation and $p_{n_{\mathrm{conv}}} = \bar{p}_{n_{\mathrm{full}}} = 512$ for the chemically reacting flow.

Encoder network

Convolution layers	
Layer	Number of filters
1	8
2	16
3	32
4	64

Fully-connected layers		
Layer	Input dimension	Output dimension
5	$p_{n_{\mathrm{conv}}}$	p

Decoder network

Fully-connected layers		
Layer	Input dimension	Output dimension
1	p	$\bar{p}_{n_{\mathrm{full}}}$

Transposed-convolutional layers	
Layer	Number of filters
2	64
3	32
4	16
5	1

decoder defining $\boldsymbol{g}$, and 4) Deep LSPG: manifold LSPG projection (Eq. (3.18)) with the deep convolutional decoder defining $\boldsymbol{g}$. All ROMs enforce the initial condition by setting the reference state according to Remark 3.1; this implies $\boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu})$ for the linear-subspace ROMs.

To solve the OΔEs arising at each time instance, we apply Newton's method for POD–Galerkin, the Gauss–Newton method for POD–LSPG, and the quasi-Newton methods proposed in Section 3.4 for the Deep Galerkin and Deep LSPG. We terminate the (quasi)-Newton iterations when the residual norm drops below 10^{-6} of its initial guess at that time instance; the initial guess corresponds to the solution at the previous time instance.

To assess the ROM accuracy, we compute the relative ℓ^2-norm of the state error

$$\text{relative error} = \sqrt{\sum_{n=1}^{N_t} \|\boldsymbol{x}^n(\boldsymbol{\mu}) - \tilde{\boldsymbol{x}}^n(\boldsymbol{\mu})\|_2^2} \Bigg/ \sqrt{\sum_{n=1}^{N_t} \|\boldsymbol{x}^n(\boldsymbol{\mu})\|_2^2} \,. \tag{7.1}$$

We also include the projection error of the solution

$$\text{projection error} = \sqrt{\sum_{n=1}^{N_t} \|(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T)(\boldsymbol{x}^n(\boldsymbol{\mu}) - \boldsymbol{x}^0(\boldsymbol{\mu}))]\|_2^2} \Bigg/ \sqrt{\sum_{n=1}^{N_t} \|\boldsymbol{x}^n(\boldsymbol{\mu})\|_2^2} \,, \tag{7.2}$$

onto both (1) the POD basis, in which case $\boldsymbol{\Phi}$ is the POD basis employed by POD–Galerkin and POD–LSPG, and (2) the optimal basis, in which case $\boldsymbol{\Phi} = \boldsymbol{\Phi}^\star$, which consists of the first p left singular vectors of the snapshot matrix collected at the online-parameter instance $\boldsymbol{W}(\boldsymbol{\mu})$. We refer to the former metric as the *POD projection error*, as it provides a lower bound for the POD–Galerkin and POD–LSPG relative errors; we refer to the latter metric as the *optimal projection error*, as it provides an ℓ^2-norm counterpart to the Kolmogorov p-width. Finally, we compute the projection of the FOM solution on the trial manifold

$$\tilde{\boldsymbol{x}}_\star^n(\boldsymbol{\mu}) = \underset{\boldsymbol{w} \in \boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\mu}) + \mathcal{S}}{\arg\min} \|\boldsymbol{x}^n - \boldsymbol{w}\|_2 \tag{7.3}$$

and compute its relative error by employing $\tilde{\boldsymbol{x}}^n \leftarrow \tilde{\boldsymbol{x}}_\star^n$ in Eq. (7.1); we refer to this as the *manifold projection error*.

7.1. 1D Burgers' equation

We first consider a parameterized inviscid Burgers' equation [66], as it comprises a very simple benchmark problem for which linear subspaces are ill suited due to its slowly decaying Kolmogorov n-width. The governing system of partial differential equations (with initial and boundary conditions) is

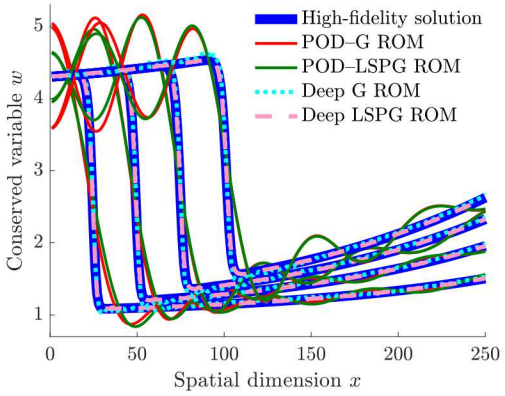
$$\begin{aligned} \frac{\partial w(x,t;\boldsymbol{\mu})}{\partial t} + \frac{\partial f(w(x,t;\boldsymbol{\mu}))}{\partial x} &= 0.02e^{\mu_2 x}, \quad \forall x \in [0,100],\ \forall t \in [0,T] \\ w(0,t;\boldsymbol{\mu}) &= \mu_1, \quad \forall t \in [0,T] \\ w(x,0) &= 1, \quad \forall x \in [0,100], \end{aligned} \tag{7.4}$$

where the flux is $f(w) = 0.5w^2$ and there are $n_\mu = 2$ parameters; thus, the intrinsic solution-manifold dimension is $p^\star = 3$ (see Remark 2.1). We set the parameter domain to $\mathcal{D} = [4.25, 5.5] \times [0.015, 0.03]$ and the final time to $T = 35$. We apply Godunov's scheme with 256 control volumes to spatially discretize Eq. (7.4), which results in a system of parameterized ODEs of the form (2.1) with $N = 256$ spatial degrees of freedom and initial condition $\boldsymbol{x}^0(\boldsymbol{\mu}) = \boldsymbol{x}^0 = \mathbf{1}$. For time discretization, we use the backward-Euler scheme, which corresponds to a linear multistep scheme with $k = 1$, $\alpha_0 = \beta_0 = 1$, $\alpha_1 = -1$, and $\beta_1 = 0$ in Eq. (2.3). We consider a uniform time step $\Delta t = 0.07$, resulting in $N_t = 500$ time instances.

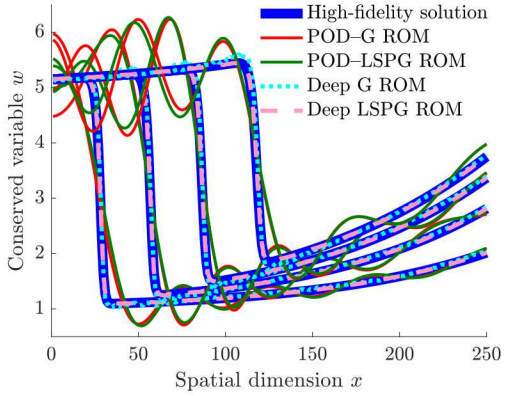
For offline training, we set the training-parameter instances to $\mathcal{D}_{\text{train}} = \{(4.25 + (1.25/9)i,\ 0.015 + (0.015/7)j)\}_{i=0,\dots,9;\ j=0,\dots 7}$, resulting in $n_{\text{train}} = 80$ training-parameter instances. The restriction operator $\boldsymbol{R}$ and prolongation operator $\boldsymbol{P}$ correspond to reshaping operators (without interpolation); the restriction operator reshapes the state vector into a tensor corresponding to the finite-volume grid such that $d = 1$, $n_1 = 256$, and $n_{\text{chan}} = 1$ in definitions (5.1) and (5.2). Then, we apply Algorithm 1 with inputs specified above and the following SGD hyperparameters: the fraction of snapshots to use for validation $\omega = 0.1$; Adam optimizer learning-rate strategy with an initial uniform learning rate $\eta = 10^{-4}$; initial parameters $\boldsymbol{\theta}^{(0)}$ computed via Xavier initialization [30] for weights and zero for biases; the number of minibatches determined by a fixed batch size of $m^{\mathcal{I}(i)} = 20$, $i = 1, \dots, n_{\text{batch}}$; a maximum number of epochs $n_{\text{epoch}} = 1000$; and early-stopping enforced if the loss on the validation set fails to decrease over 100 epochs. For the online stage, we consider two online-parameter instances $\boldsymbol{\mu}^1_{\text{test}} = (4.3, 0.021)$, and $\boldsymbol{\mu}^2_{\text{test}} = (5.15, 0.0285)$, which are not included in $\mathcal{D}_{\text{train}}$.

Figure 2 reports solutions at four different time indices computed by using FOM OΔE and the four considered ROMs. All ROMs employ the same reduced dimension of $p = 10$ in Figure 2 (left) and $p = 20$ in Figure 2 (right). These results demonstrate that nonlinear-manifold ROMs Deep Galerkin and Deep LSPG produce extremely accurate solutions, while the linear-subspace ROMs—constructed using the same training data—exhibit significant errors. This is due to the fundamental ill-suitedness of linear trial subspaces to advection-dominated problems.

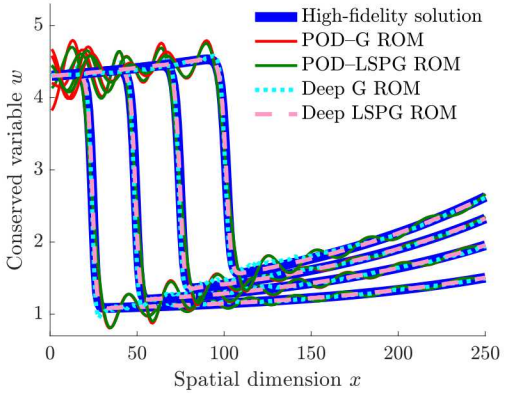
Figure 3 reports the convergence of the relative error as a function of reduced dimension p. These results illustrate the promise of employing nonlinear trial manifolds. First, we note that employing a linear trial subspace immediately introduces significant errors: the projection error onto the optimal basis of dimension $p = p^\star = 3$ is over 10%; the optimal nonlinear trial manifold (corresponding to the solution manifold) of the same dimension yields zero error. Even with a reduced dimension $p = 50$, the relative projection error onto the optimal basis has not yet reached 0.1%. Second, we note that with a reduced dimension of only $p = 5 = p^\star + 2$, the Deep LSPG ROM realizes relative errors near 0.1%, while linear-subspace ROMs (including the projection error with the optimal basis) exhibit relative errors near 10%. Thus, the proposed convolutional autoencoder is very close to achieving the optimal performance of any nonlinear trial manifold; the dimension it requires to realize sub-0.1% errors is only two larger than the intrinsic solution-manifold dimension $p^\star = 3$. We also observe that the POD–Galerkin and POD–LSPG ROMs are also nearly able to achieve optimal performance for linear-subspace ROMs, as their relative errors are close to the projection error onto the optimal basis; unfortunately, this error remains quite large relative to the Deep Galerkin and Deep LSPG ROMs. This highlights that the proposed nonlinear-manifold ROMs are able to overcome the fundamental limitations of linear-subspace ROMs on problems exhibiting a slowly decaying Kolmogorov n-width. We also observe that the POD basis is very close to the optimal basis, which implies that the training data are sufficient to accurately represent the online solution.



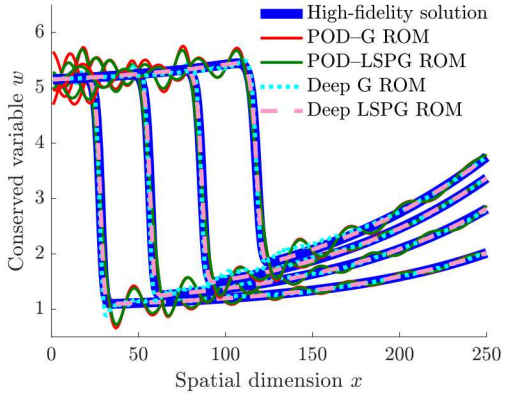
(a) Online-parameter instance $\boldsymbol{\mu}^1_{\text{test}} = (4.3, 0.021)$ with $p = 10$



(b) Online-parameter instance $\boldsymbol{\mu}^2_{\text{test}} = (5.15, 0.0285)$ with $p = 10$



(c) Online-parameter instance $\boldsymbol{\mu}^1_{\text{test}} = (4.3, 0.021)$ with $p = 20$

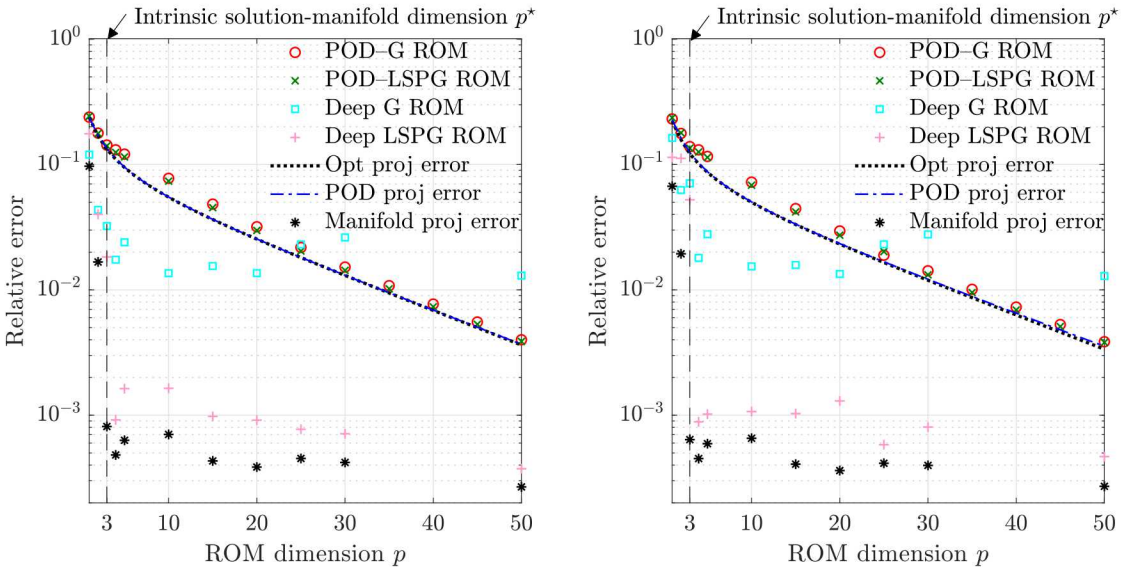


(d) Online-parameter instance $\boldsymbol{\mu}^2_{\text{test}} = (5.15, 0.0285)$ with $p = 20$

Figure 2: *1D Burgers' equation.* Online solutions at four different time instances $t = \{3.5, 7.0, 10.5, 14\}$ computed by solving the FOM (High-fidelity solution, solid blue line), POD–ROMs with Galerkin projection (POD–G ROM, solid red line) and LSPG projection (POD–LSPG ROM, solid green line), and the nonlinear-manifold ROMs equipped with the deep convolutional decoder associated with time-continuous residual minimization (Deep G ROM, dotted cyan line) and with time-discrete residual minimization (Deep LSPG ROM, dashed magenta line). All ROMs employ a reduced dimension of $p = 10$ (left) and $p = 20$ (right).

Additionally, we note that Deep LSPG outperforms Deep Galerkin for this problem, likely due to the fact that the residual-minimization problem is defined over a finite time step rather than time-instantaneously; similar results have been shown in the case of linear trial subspaces, e.g., in Refs. [13].

Finally, to illustrate the dependence of the proposed methods on the amount of training data, we vary the number of training-parameter instances in the set $n_{\text{train}} \in \{5, 10, 20, 40, 80, 120\}$, and we use the resulting $n_{\text{snap}} = N_t n_{\text{train}}$ snapshots to train both the autoencoder for the Deep Galerkin and Deep LSPG ROMs and to compute the POD basis for the POD–Galerkin and POD–LSPG ROMs. We fix the reduced dimension to $p = 5$. For offline training, we define the maximum number of epochs and the early-stopping strategy in a manner that the Adam optimizer employs the same number of gradient computations to train each model. Figure 4 reports the relative error of the four ROMs for the diferent number of training-parameter instances. This figure demonstrates that the proposed Deep Galerkin and Deep LSPG ROMs yield accurate results—around 2% relative error for Deep Galerkin and less than 1% relative error for Deep LSPG—with only $n_{\text{train}} = 5$ parameter instances. This highlights that—for this example—the proposed methods do not



(a) Online-parameter instance 1, $\boldsymbol{\mu}^1_{\text{test}} = (4.3, 0.021)$ (b) Online-parameter instance 2, $\boldsymbol{\mu}^2_{\text{test}} = (5.15, 0.0285)$

Figure 3: *1D Burgers' equation.* Relative errors of the ROM solutions (Eq. (7.1)) at online-parameter instances $\boldsymbol{\mu}^1_{\text{test}}$ and $\boldsymbol{\mu}^2_{\text{test}}$ for varying dimensions of the ROMs. The figure shows the relative errors of POD–ROMs with Galerkin projection (POD–G ROM, red circle) and LSPG projection (POD–LSPG ROM, green cross), and the nonlinear-manifold ROMs equipped with the decoder with time-continuous optimality (Deep G ROM, cyan square) and time-discrete optimality (Deep LSPG ROM, magenta plus sign). The figure also reports the projection error (Eq. (7.2)) of the solution onto the POD basis (POD proj error, dashed blue line) and the optimal basis (Opt proj error, dotted black line), and reports the manifold projection error of the solution (Eq. (7.3), Manifold proj error, black asterisk). The vertical dashed line indicates the intrinsic solution-manifold dimension $p^\star$ (see Remark 2.1).

require an excessive amount of training data relative to standard POD–based ROMs. This lack of demand for a large amount of data is likely due to the use of convolutional layers in the autoencoder, which effectively employ parameter sharing to reduce significantly the total number of parameters in the autoencoder, and thus the amount of data needed for training.

7.2. Chemically reacting flow

We now consider a model of the reaction of a premixed H_2-air flame at constant uniform pressure [10]. The evolution of the flame is modeled by the nonlinear convection–diffusion–reaction equation

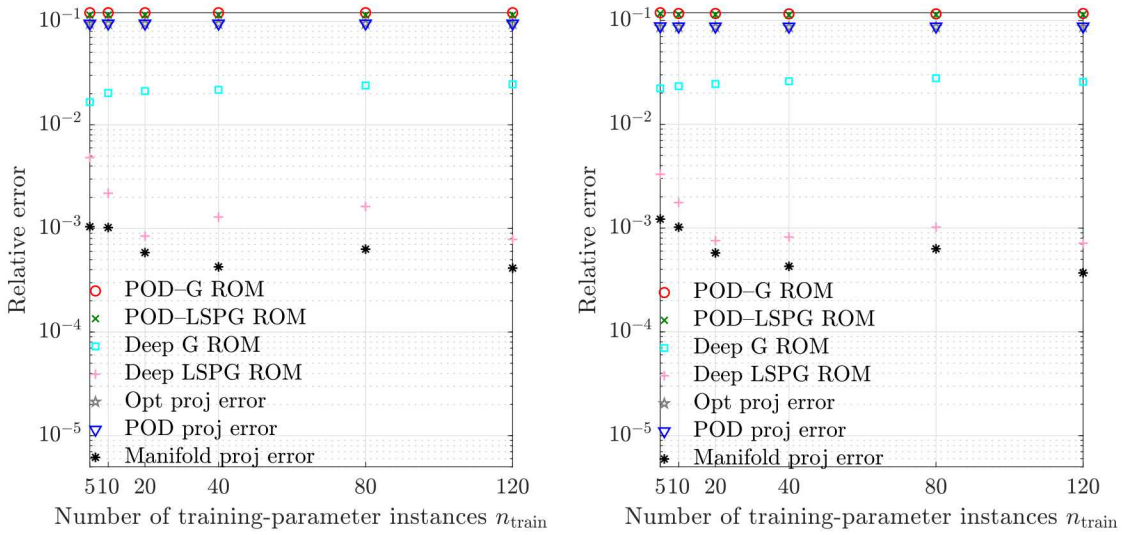
$$\frac{\partial \boldsymbol{w}(\vec{x},t;\boldsymbol{\mu})}{\partial t} = \nabla \cdot (\kappa \nabla \boldsymbol{w}(\vec{x},t;\boldsymbol{\mu})) - \boldsymbol{v} \cdot \nabla \boldsymbol{w}(\vec{x},t;\boldsymbol{\mu}) + \boldsymbol{q}(\boldsymbol{w}(\vec{x},t;\boldsymbol{\mu});\boldsymbol{\mu}) \qquad \text{in } \Omega \times \mathcal{D} \times [0,T], \tag{7.5}$$

where ∇ denotes the gradient with respect to physical space, κ denotes the molecular diffusivity, $\boldsymbol{v}$ denotes the velocity field, and

$$\boldsymbol{w}(\vec{x},t;\boldsymbol{\mu}) \equiv [w_T(\vec{x},t;\boldsymbol{\mu}), w_{\mathrm{H_2}}(\vec{x},t;\boldsymbol{\mu}), w_{\mathrm{O_2}}(\vec{x},t;\boldsymbol{\mu})), w_{\mathrm{H_2O}}((\vec{x},t;\boldsymbol{\mu}))]^T \in \mathbb{R}^4$$

denotes the thermo-chemical composition vector consisting of the temperature $w_T(\vec{x},t;\boldsymbol{\mu})$ and the mass fractions of chemical species H_2, O_2, and H_2O, i.e., $w_i(\vec{x},t;\boldsymbol{\mu})$ for $i \in \{\mathrm{H_2}, \mathrm{O_2}, \mathrm{H_2O}\}$. The nonlinear reaction source term $\boldsymbol{q}(\boldsymbol{w}(\vec{x},t;\boldsymbol{\mu});\boldsymbol{\mu}) \equiv [q_T(\boldsymbol{w};\boldsymbol{\mu}), q_{\mathrm{H_2}}(\boldsymbol{w};\boldsymbol{\mu}), q_{\mathrm{O_2}}(\boldsymbol{w};\boldsymbol{\mu}), q_{\mathrm{H_2O}}(\boldsymbol{w};\boldsymbol{\mu})]^T$ is of Arrhenius type and is defined as

$$\begin{aligned} q_T(\boldsymbol{w};\boldsymbol{\mu}) &= Q q_{\mathrm{H_2O}}(\boldsymbol{w};\boldsymbol{\mu}) \\ q_i(\boldsymbol{w};\boldsymbol{\mu}) &= -\nu_i \left(\frac{W_i}{\rho}\right) \left(\frac{\rho w_{\mathrm{H_2}}}{W_{\mathrm{H_2}}}\right)^{\nu_{\mathrm{H_2}}} \left(\frac{\rho w_{\mathrm{O_2}}}{W_{\mathrm{O_2}}}\right)^{\nu_{\mathrm{O_2}}} A \exp\left(-\frac{E}{R w_T}\right), \quad i \in \{\mathrm{H_2}, \mathrm{O_2}, \mathrm{H_2O}\}, \end{aligned}$$



(a) Online-parameter instance $\boldsymbol{\mu}^1_{\text{test}} = (4.3, 0.021)$ with $p = 5$

(b) Online-parameter instance $\boldsymbol{\mu}^2_{\text{test}} = (5.15, 0.0285)$ with $p = 5$

Figure 4: *1D Burgers' equation: varying amount of training data.* Relative errors of the ROM solutions (Eq. (7.1)) at online-parameter instances $\boldsymbol{\mu}^1_{\text{test}}$ and $\boldsymbol{\mu}^2_{\text{test}}$ for a varying the number training-parameter instances $n_{\text{train}} \in \{5, 10, 20, 40, 80, 120\}$. The figure shows the relative errors of POD–ROMs with Galerkin projection (POD–G ROM, red circle) and LSPG projection (POD–LSPG ROM, green cross), and the nonlinear-manifold ROMs equipped with the decoder with time-continuous optimality (Deep G ROM, cyan square) and time-discrete optimality (Deep LSPG ROM, magenta plus sign). The figure also reports the projection error (Eq. (7.2)) of the solution onto the POD basis (POD proj error, blue triangle) and the optimal basis (Opt proj error, gray pentagram), and reports the manifold projection error of the solution (Eq. (7.3), Manifold proj error, black asterisk).

where $(\nu_{\text{H}_2}, \nu_{\text{O}_2}, \nu_{\text{H}_2\text{O}}) = (2, 1, -2)$ denote stoichiometric coefficients, $(W_{\text{H}_2}, W_{\text{O}_2}, W_{\text{H}_2\text{O}}) = (2.016, 31.9, 18)$ denote molecular weights with units g·mol^{-1}, $\rho = 1.39 \times 10^{-3}$ g $\cdot$ cm^{-3} denotes the density mixture, $R = 8.314$ J $\cdot$ mol$^{-1}\cdot$ K^{-1} denotes the universal gas constant, and $Q = 9800$K denotes the heat of the reaction. The $n_\mu = 2$ parameters correspond to $\boldsymbol{\mu} = (A, E)$, which are the pre-exponential factor A and the activation energy E; we set the corresponding parameter domain to $\mathcal{D} = [2.3375 \times 10^{12}, 6.5 \times 10^{12}] \times [5.625 \times 10^3, 9 \times 10^3]$. Thus, the intrinsic solution-manifold dimension is $p^\star = 3$ (see Remark 2.1). We set the molecular diffusivity to $\kappa = 2$ cm^2·s^{-1}, and the velocity field set to be constant and divergence-free with $\boldsymbol{v} = [50 \text{ cm} \cdot \text{s}^{-1},\ 0]^T$. We set the final time to $T = 0.06$ s.

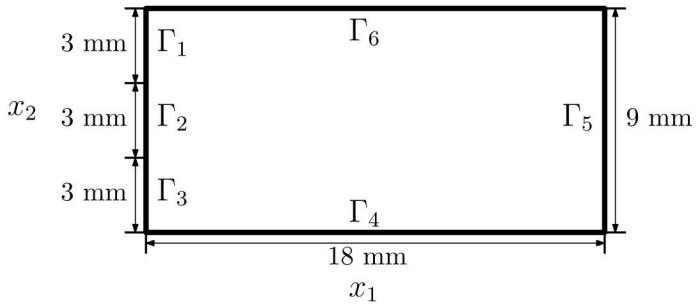


Figure 5: *Chemically reacting flow.* The geometry of the spatial domain Ω.

Figure 5 reports the geometry of the spatial domain. On the inflow boundary Γ_2, we impose Dirichlet boundary conditions $w_{\text{H}_2} = 0.0282$, $w_{\text{O}_2} = 0.2259$ and $w_{\text{H}_2\text{O}} = 0$ for the chemical-species mass fractions and $w_T = 950$K for the temperature. On boundaries Γ_1 and Γ_3, we impose homogeneous Dirichlet boundary

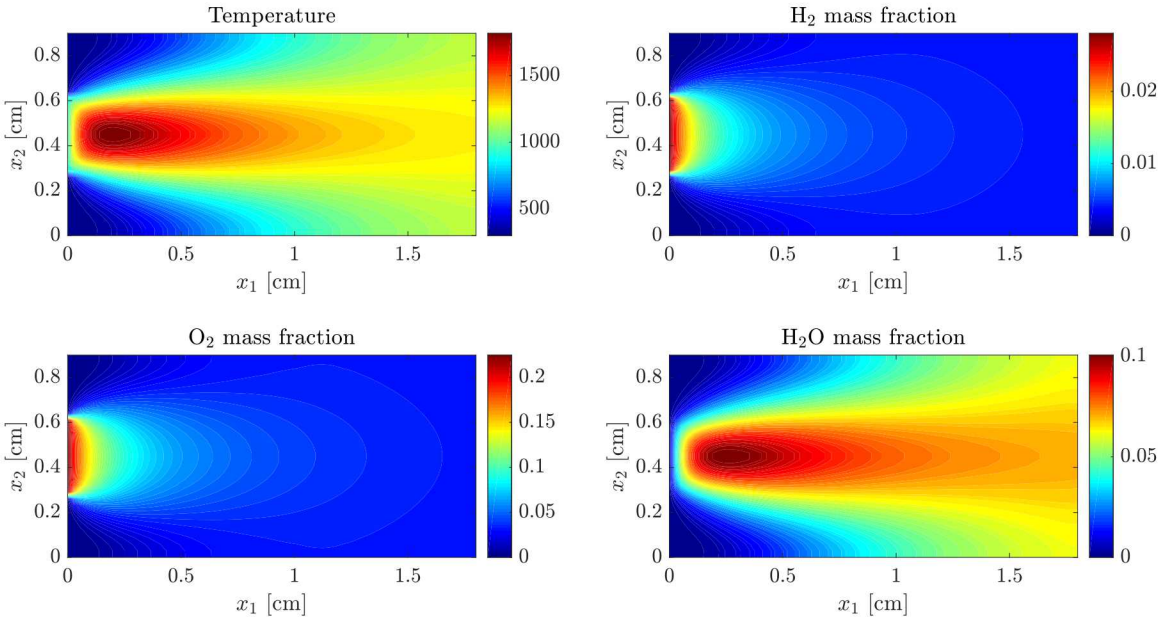


Figure 6: *Chemically reacting flow.* FOM solutions of the temperature, mass fractions of H_2, O_2, and H_2O at online-parameter instance $\boldsymbol{\mu}^1_{\text{test}} = (2.5 \times 10^{12},\ 5.85 \times 10^3)$ at $t = 0.06$.

conditions for the chemical-species mass fractions, and we set the temperature $w_T = 300$ K. On Γ_4, Γ_5, and Γ_6, we impose homogeneous Neumann conditions on the temperature and mass fractions. We consider a uniform initial condition corresponding to a domain that is empty of chemical species such that $w_{\text{H}_2} = w_{\text{O}_2} = w_{\text{H}_2\text{O}} = 0$ and we set the temperature to $w_T = 300$ K.

We employ a finite-difference method with 65 grid points in the horizontal direction and 32 grid points in the vertical direction to spatially discretize Eq. (7.5), which results in a system of parameterized ODEs of the form (2.1) with $N = 8192$ degrees of freedom.

For time discretization, we employ the second order backward difference scheme (BDF2), which corresponds to a linear multistep scheme with $k = 2$, $\alpha_0 = 1$, $\alpha_1 = -\frac{4}{3}$, $\alpha_2 = \frac{1}{3}$, $\beta_0 = \frac{2}{3}$, and $\beta_1 = \beta_2 = 0$ in Eq. (2.3). We consider a uniform time step $\Delta t = 10^{-4}$, resulting in $N_t = 600$ time instances.

For offline training, we set the training-parameter instances to $\mathcal{D}_{\text{train}} = \{(2.3375 \times 10^{12} + (3.2725 \times 10^{12}/7)i,\ 5.625 \times 10^3 + (3.375 \times 10^3/7)j)\}_{i=0,\ldots,7;\ j=0,\ldots7}$, resulting in $n_{\text{train}} = 64$ training-parameter instances. The restriction operator $\boldsymbol{R}$ and prolongation operator $\boldsymbol{P}$ correspond to reshaping operators (without interpolation); the restriction operator reshapes the state vector into a tensor corresponding to the finite-difference grid such that $d = 2$, $n_1 = 64$, $n_2 = 32$, and $n_{\text{chan}} = 4$ in definitions (5.1) and (5.2). We emphasize that each of the 4 unknown variables is considered a different channel for the input data. Then, we apply Algorithm 1 with inputs specified above and the following SGD hyperparameters: the fraction of snapshots to use for validation $\omega = 0.1$; Adam optimizer learning-rate strategy with an initial uniform learning rate $\eta = 10^{-4}$; initial parameters parameters $\boldsymbol{\theta}^{(0)}$ computed via He initialization [36] for weights and zero for biases; number of minibatches determined by a fixed batch size of $m^{\mathcal{I}(i)} = 20$, $i = 1, \ldots, n_{\text{batch}}$; a maximum number of epochs $n_{\text{epoch}} = 5000$; and early-stopping enforced if the loss on the validation set fails to decrease over 500 epochs. For the online stage, we consider parameter instances $\boldsymbol{\mu}^1_{\text{test}} = (2.5 \times 10^{12},\ 5.85 \times 10^3)$ and $\boldsymbol{\mu}^2_{\text{test}} = (3.2 \times 10^{12},\ 7.25 \times 10^3)$ that are not included in $\mathcal{D}_{\text{train}}$.

Figure 6 reports the FOM solution for online-parameter instance $\boldsymbol{\mu}^1_{\text{test}}$ and final time $t = T = 0.06$ s. Figures 7 and 8 report relative errors of the temperature solutions and the H_2 mass fraction solutions at the final time $t = T$ computed using all considered ROMs with the same reduced dimension $p = 3$. Again, we observe that the proposed nonlinear-manifold ROMs produce significantly lower errors as compared with the linear-subspace ROMs. Figure 9 reports the convergence of the relative error as a function of reduced

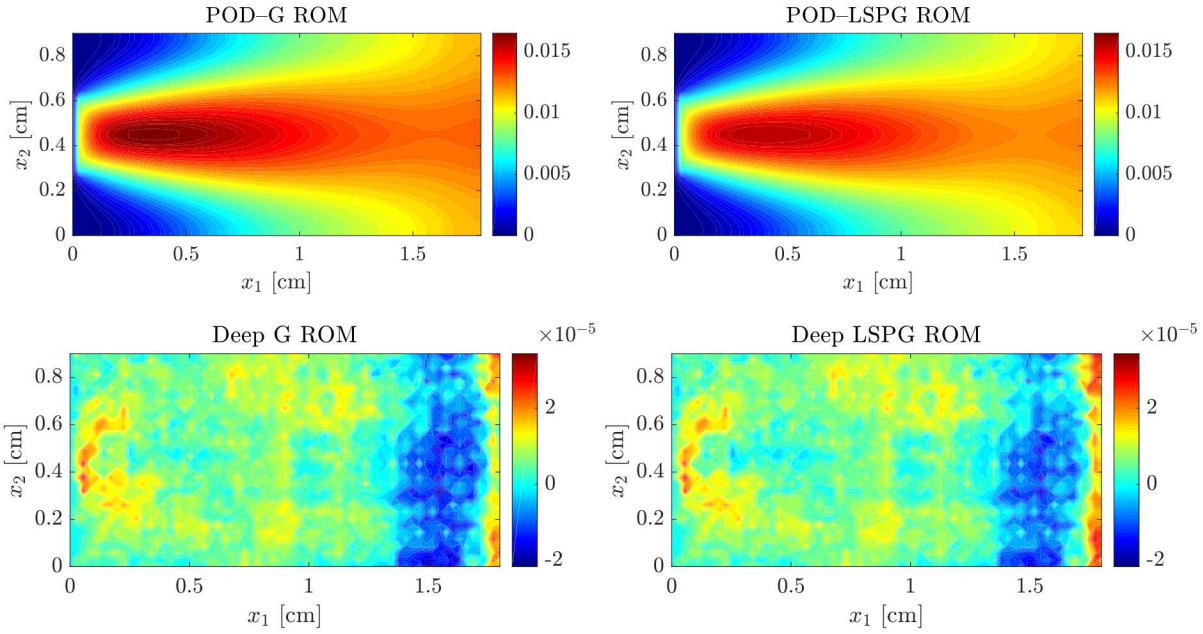


Figure 7: *Chemically reacting flow.* Relative errors of the temperature solution computed using ROMs at online-parameter instance $\boldsymbol{\mu}^1_{\text{test}} = (2.5 \times 10^{12}, 5.85 \times 10^3)$ at $t = 0.06$. The dimensions of the generalized coordinates in both linear-subspace ROMs (top row) and nonlinear-manifold ROMs (bottom row) are the same $p = 3$. The colormaps on the each row are in the same scale.

dimension p along with the projection errors Eq. 7.2 onto the POD basis and the optimal basis. This figure again shows that the proposed manifold ROMs significantly outperform the linear-subspace ROMs, as well as the projection error onto both the POD and optimal bases. First, we observe that employing a linear trial subspace again introduces significant errors, as the projection error onto the optimal basis of dimension $p = p^\star = 3$ is around 5%; the optimal nonlinear trial manifold (corresponding to the solution manifold) of the same dimension yields zero error. Second, we note that for a reduced dimension of only $p = p^\star = 3$, both the Deep Galerkin and Deep LSPG ROMs yield relative errors of less than 0.1%, while the projection errors onto the POD and optimal bases exceed 5%, and the linear-subspace ROMs yield relative errors in excess of 40%. Thus, the proposed convolutional autoencoder nearly achieves optimal performance, as the dimension it requires to realize sub-0.1% errors is exactly equal to the intrinsic solution-manifold dimension $p^\star = 3$.

We also observe that there is a significant gap between the performance of the linear-subspace ROMs and the POD projection error; this gap is attributable to the closure problem. We also observe that—in contrast with the previous example—there is a non-trivial gap between the POD projection error and the optimal-basis projection error, which suggests that the online solution is less well represented by the training data than in the previous case.

Additionally, we observe that for a reduced dimension of $p = 10$, the projection error onto the optimal basis begins to become smaller than the relative error of the proposed Deep Galerkin and Deep LSPG ROMs; however this occurs for (already small) errors less than 0.1%. The error saturation of the Deep Galerkin and Deep LSPG ROMs is likely due to the fact that the online solution cannot be perfectly represented using the training data, as illustrated by the gap between the projection errors associated with POD and the optimal basis.

We vary the number of training-parameter instances in the set $n_{\text{train}} \in \{4, 8, 16, 32, 64\}$, and we use the resulting $n_{\text{snap}} = N_t n_{\text{train}}$ snapshots to train both the autoencoder for the Deep Galerkin and Deep LSPG ROMs and to compute the POD basis for the POD–Galerkin and POD–LSPG ROMs. We fix the reduced dimension to $p = 5$. Figure 10 reports the relative error of the four ROMs for these different amounts of training data. This figure demonstrates that the proposed Deep Galerkin and Deep LSPG ROMs yield

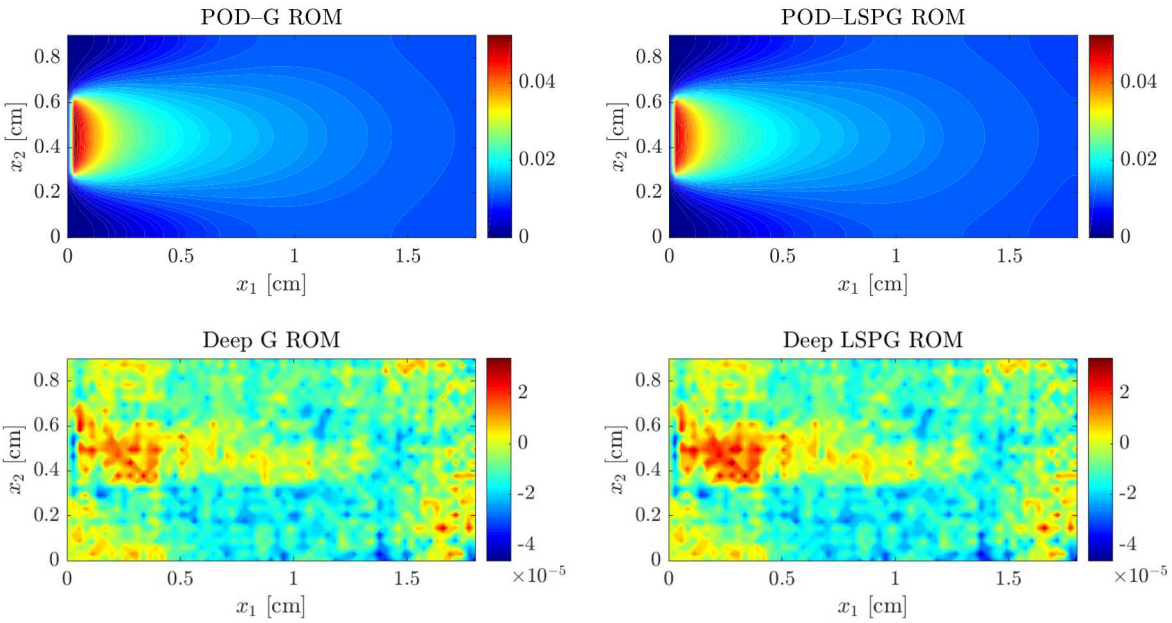


Figure 8: *Chemically reacting flow.* Relative errors of the mass fraction of H_2 computed using ROMs at online-parameter instance $\boldsymbol{\mu}^1_{\text{test}} = (2.5 \times 10^{12}, 5.85 \times 10^3)$ at $t = 0.06$. The dimensions of the generalized coordinates in both linear-subspace ROMs (top row) and nonlinear-manifold ROMs (bottom row) are the same $p = 3$. The color maps on the each row are in the same scale.

accurate results—less than 1% relative error for both Deep Galerkin and Deep LSPG—with only $n_{\text{train}} = 4$ parameter instances. These results again show that the proposed methods do not appear to require an excessive amount of training data relative to standard POD–based ROMs.

8. Conclusion

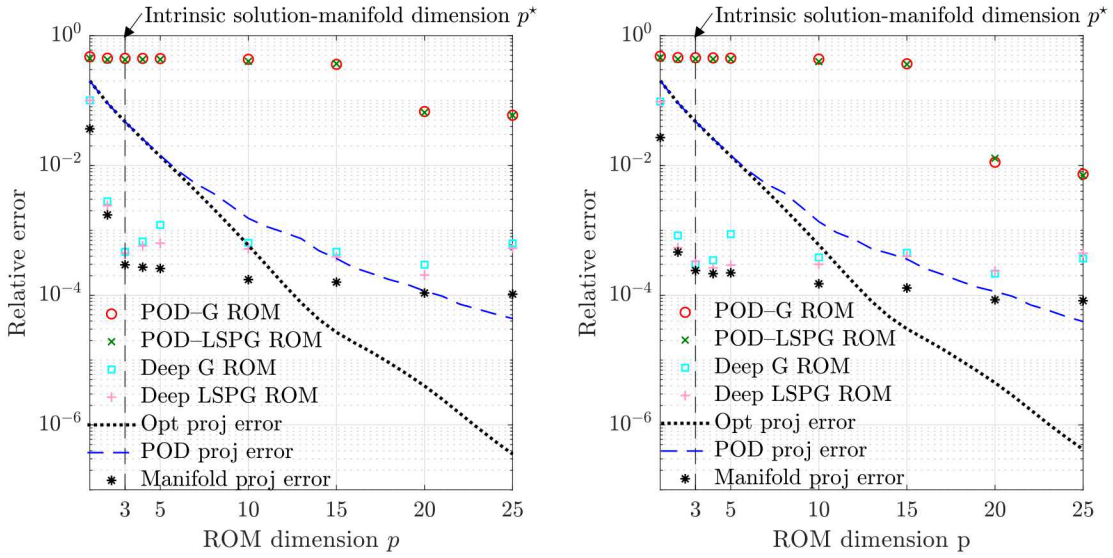
This work has proposed novel manifold Galerkin and manifold LSPG projection techniques, which project dynamical-system models onto arbitrary continuously-differentiable nonlinear manifolds. We demonstrated how these methods can exactly satisfy the initial condition, and provided quasi-Newton solvers for implicit time integration.

We performed analyses that demonstrated that employing an affine trial manifold recovers classical linear-subspace Galerkin and LSPG projection. We also derived sufficient conditions for commutativity of time discretization and manifold Galerkin projection, as well as conditions under which manifold Galerkin and manifold LSPG projection are equivalent. In addition, we derived *a posteriori* time-discrete error bounds for the proposed methods.

We also proposed a practical strategy for computing a representative low-dimensional nonlinear trial manifold that employs a specific convolutional autoencoder tailored to spatially distributed dynamical-system states. When the Manifold Galerkin and Manifold LSPG ROMs employ this choice of decoder, we refer to them as Deep Galerkin and Deep LSPG ROMs, respectively.

Finally, numerical experiments demonstrated the ability of the method to produce substantially lower errors for low-dimensional reduced states than even the projection onto the optimal basis. Indeed, the proposed methodology is nearly able to achieve optimal performance for a nonlinear-manifold method; in the first case, the reduced dimension required to achieve sub-0.1% errors is only two larger than the intrinsic solution-manifold dimension; in the second case, the reduced dimension required to achieve such errors is exactly equal to the intrinsic solution-manifold dimension.

We note that one drawback of the method in its current form (relative to classical linear-subspace methods) is that it incurs a costlier training process, as training a deep convolutional autoencoder is significantly more



(a) Online-parameter instance 1, $\boldsymbol{\mu}^1_{\text{test}} = \left(2.5 \times 10^{12}, 5.85 \times 10^3\right)$

(b) Online-parameter instance 2, $\boldsymbol{\mu}^2_{\text{test}} = \left(3.2 \times 10^{12}, 7.25 \times 10^3\right)$

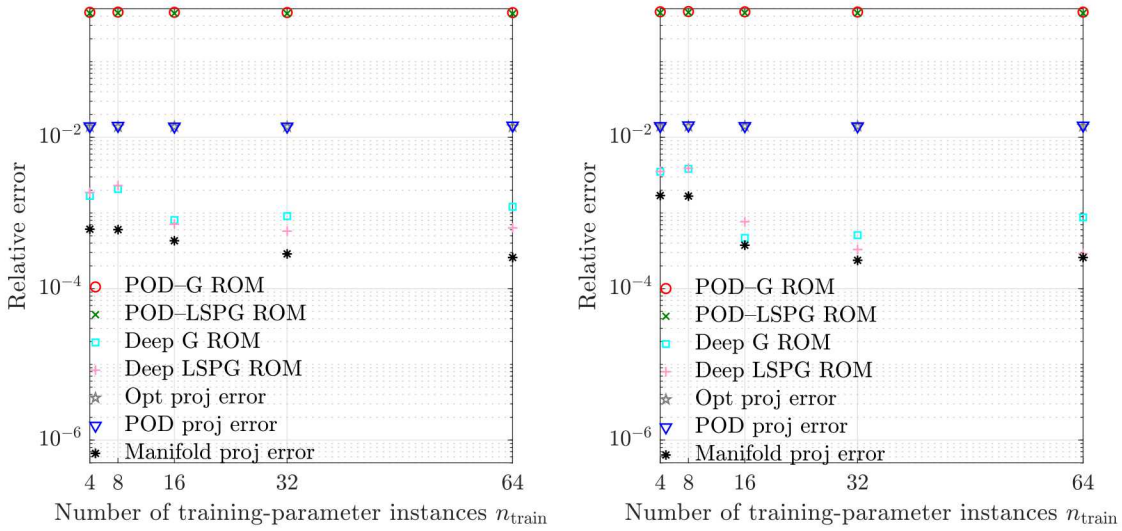
Figure 9: *Chemically reacting flow.* Relative errors of the ROM solutions (Eq. (7.1)) at online-parameter instances $\boldsymbol{\mu}^1_{\text{test}} = (2.5 \times 10^{12},\ 5.85 \times 10^3)$ and $\boldsymbol{\mu}^2_{\text{test}} = \left(3.2 \times 10^{12}, 7.25 \times 10^3\right)$ for varying ROM dimension. The figure shows the relative errors of POD–ROMs with Galerkin projection (POD–G ROM, red circle) and LSPG projection (POD–LSPG ROM, green cross), and the nonlinear-manifold ROMs equipped with the decoder with time-continuous optimality (Deep G ROM, cyan square) and time-discrete optimality (Deep LSPG ROM, magenta plus sign). The figure also reports the projection error (Eq. (7.2)) of the solution onto the POD basis (POD proj error, dashed blue line) and the optimal basis (Opt proj error, dotted black line), and reports the manifold projection error of the solution (Eq. (7.3), Manifold proj error, black asterisk). The vertical dashed line indicates the intrinsic solution-manifold dimension $p^\star$ (see Remark 2.1).

computationally expensive than simply computing the left singular vectors of a snapshot matrix. In addition, the proposed method is characterized by significantly more hyperparameters—which relate to the autoencoder architecture—than classical methods.

Future work involves integrating the proposed manifold Galerkin and manifold LSPG projection methods within a full hyper-reduction framework to realize computational cost savings; this investigation will leverage sparse norms as discussed in Remarks 3.4 and 3.6, and will also consider specific instances of the proposed autoencoder architecture that enable computationally efficient hyper-reduction. Additional future works include developing autoencoder architectures that can capture solution features across multiple scales (as is relevant for multiscale problems), investigating graph convolutional layers for application to problems discretized on nonuniform or irregular grids, and enforcing constraints that ensure the resulting solution satisfies important physical properties (e.g., conservation [15], monotonicity preservation [2, 22]). In addition, we aim to implement the proposed techniques in a production-level simulation code and demonstrate the methods on truly large-scale dynamical-system models.

Acknowledgments

The authors gratefully acknowledge Matthew Zahr for graciously providing the `pyMORTestbed` code that was modified to obtain the numerical results, as well as Jeremy Morton for useful discussions on the application of convolutional neural networks to simulation data. The authors also thank Patrick Blonigan and Eric Parish for providing useful feedback. This work was sponsored by Sandia's Advanced Simulation and Computing (ASC) Verification and Validation (V&V) Project/Task #103723/05.30.02. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.



(a) Online-parameter instance 1, $\boldsymbol{\mu}^1_{\text{test}} = \left(2.5 \times 10^{12}, 5.85 \times 10^3\right)$ with $p = 5$

(b) Online-parameter instance 2, $\boldsymbol{\mu}^2_{\text{test}} = \left(3.2 \times 10^{12}, 7.25 \times 10^3\right)$ with $p = 5$

Figure 10: *Chemically reacting flow: varying amount of training data.* Relative errors of the ROM solutions (Eq. (7.1)) at online-parameter instances $\boldsymbol{\mu}^1_{\text{test}} = (2.5 \times 10^{12}, 5.85 \times 10^3)$ and $\boldsymbol{\mu}^2_{\text{test}} = \left(3.2 \times 10^{12}, 7.25 \times 10^3\right)$ for a varying the number training-parameter instances $n_{\text{train}} \in \{4, 8, 16, 32, 64\}$. The figure shows the relative errors of POD–ROMs with Galerkin projection (POD–G ROM, red circle) and LSPG projection (POD–LSPG ROM, green cross), and the nonlinear-manifold ROMs equipped with the decoder with time-continuous optimality (Deep G ROM, cyan square) and time-discrete optimality (Deep LSPG ROM, magenta plus sign). The figure also reports the projection error (Eq. (7.2)) of the solution onto the POD basis (POD proj error, blue triangle) and the optimal basis (Opt proj error, gray pentagram), and reports the manifold projection error of the solution (Eq. (7.3), Manifold proj error, black asterisk).

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.

Appendix A. Stochastic gradient descent for autoencoder training

This section briefly summarizes stochastic gradient descent (SGD) with minibatching and early stopping, which is used to compute the parameters $\boldsymbol{\theta}^\star$ of the autoencoder.

We begin by randomly shuffling the snapshot matrix $\boldsymbol{W}$ (defined in Eq. (6.1)) into neural-network training snapshots $\boldsymbol{W}_{\text{train}} \in \mathbb{R}^{N \times m}$ and validation snapshots $\boldsymbol{W}_{\text{val}} \in \mathbb{R}^{N \times \bar{m}}$, i.e.,

$$[\boldsymbol{W}_{\text{train}} \ \boldsymbol{W}_{\text{val}}] = \boldsymbol{W}\boldsymbol{P}$$

where $\boldsymbol{P} \in \{0,1\}^{n_{\text{snap}} \times n_{\text{snap}}}$ denotes a random permutation matrix and $m + \bar{m} = n_{\text{snap}}$, typically with $\bar{m} \approx 0.1m$.

Given the training data $\boldsymbol{W}_{\text{train}}$, we compute the optimal parameters $\boldsymbol{\theta}^\star$ by (approximately) solving the optimization problem

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \ \boldsymbol{J}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{\text{data}}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\boldsymbol{x}^i_{\text{train}}, \boldsymbol{\theta}), \tag{A.1}$$

which minimizes the empirical risk over the training data. Here, $\hat{p}_{\text{data}}$ denotes the empirical distribution associated with the training data $\boldsymbol{W}_{\text{train}}$ and the loss function $\mathcal{L}$ provides a measure of discrepancy between

training snapshot $\boldsymbol{x}^i_{\text{train}}$ and its reconstruction $\boldsymbol{h}(\boldsymbol{x}^i_{\text{train}};\boldsymbol{\theta})$; for example, the ℓ^2-loss function is often employed such that

$$\mathcal{L} : (\boldsymbol{x}, \boldsymbol{\theta}) \mapsto \|\boldsymbol{x} - \boldsymbol{h}(\boldsymbol{x};\boldsymbol{\theta})\|_2^2, \tag{A.2}$$

in which case the loss function is equivalent to that employed for POD (compare with Problem (6.3)). Note that autoencoder training is categorized as an *unsupervised learning* (or semi-supervised learning) problem, as there is no target response variable other than recovery of the original input data.

We apply SGD to (approximately) solve optimization problem (A.1), which leads to parameter updates at the ith iteration of the form

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \widetilde{\nabla \boldsymbol{J}}^{(i)}(\boldsymbol{\theta}^{(i)}). \tag{A.3}$$

Here, $\eta \in \mathbb{R}$ denotes the step length or *learning rate* and $\widetilde{\nabla \boldsymbol{J}}^{(i)} (\approx \nabla \boldsymbol{J})$ denotes a gradient approximation at optimization iteration i. The gradient approximation corresponds to the sample mean of the gradient over a *minibatch* $\boldsymbol{W}_{\text{mini},\mathcal{I}(i)} \in \mathbb{R}^{N \times m^{\mathcal{I}(i)}}$, where

$$[\boldsymbol{W}_{\text{mini},1} \ \cdots \ \boldsymbol{W}_{\text{mini},n_{\text{batch}}}] = \boldsymbol{W}_{\text{train}} \boldsymbol{P}_{\text{train}},$$

where $\mathcal{I} : i \mapsto ((i-1) \mod n_{\text{batch}}) + 1$ provides the mapping from optimization iteration i to batch index, and $\boldsymbol{P}_{\text{train}} \in \{0,1\}^{N \times m}$ denotes a random permutation matrix. That is, the gradient approximation at optimization iteration i is

$$\widetilde{\nabla \boldsymbol{J}}^{(i)}(\boldsymbol{\theta}^{(i)}) = \frac{1}{m^{\mathcal{I}(i)}} \sum_{j=1}^{m^{\mathcal{I}(i)}} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x}^j_{\text{mini},\mathcal{I}(i)}, \boldsymbol{\theta}). \tag{A.4}$$

For small batch sizes, this gradient approximation not only significantly reduces the per-iteration cost of the optimization algorithm, it can also improve generalization performance, and—for early iterations and for minibatch size 1—leads to the same sublinear rate of convergence of the *expected risk* as the empirical risk [9]. In practice, each gradient contribution $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x}^j_{\text{mini},\mathcal{I}(i)}, \boldsymbol{\theta})$ is computed from the chain rule via automatic differentiation, which is referred to in deep learning as *backpropagation* [70].

Some optimization methods employ adaptive learning rates that are tailored for each parameter, which comprises a modification of the parameter update (A.3) to

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \boldsymbol{\eta}^{(i)} \odot \widetilde{\nabla \boldsymbol{J}}^{(i)}(\boldsymbol{\theta}^{(i)}) \tag{A.5}$$

where $\boldsymbol{\eta}^{(i)}$ denotes a vector of learning rates and $\odot$ denotes the (element-wise) Hadamard product. Examples include AdaGrad [27], RMSProp [76], and Adam.

Although we have presented the specific case of non-adaptive minibatches (which we employ in the numerical experiments), it is also possible to employ adaptive batch sizes; often, the batch size increases with iteration count to produce a lower-variance gradient estimate as a local minimum is approached [9].

Rather than terminating iterations when a local minimum of the objective function $\boldsymbol{J}$ is reached, we instead employ *early stopping*, which is a widely used form of regularization that has been shown to improve generalization performance in many applications [32, 9]. Here, we terminate iterations when the loss function on the validation snapshots

$$\frac{1}{\bar{m}} \sum_{i=1}^{\bar{m}} \mathcal{L}(\boldsymbol{x}^i_{\text{val}}, \boldsymbol{\theta})$$

does not decrease for a certain number of *epochs*, where an epoch is equivalent to n_{batch} iterations, i.e., a single pass through the training data. Early stopping effectively treats the number of optimization iterations as a hyperparameter, as allowing a large number of iterations can be associated with a higher-capacity model.

Algorithm 2 describes autoencoder training using SGD with minibatching and early stopping. Note that the adaptive learning rate strategy, initial parameters $\boldsymbol{\theta}^{(0)}$, number of minibatches n_{batch}, maximum number of epochs n_{epoch}, and early-stopping criterion are all SGD hyperparameters that comprise inputs to the

algorithm. Line 6 of Algorithm 2 applies the adaptive learning rate strategy. For example, AdaGrad scales the learning rate to be inversely proportional to the square root of the sum of all the historical squared values of the gradient [27]; Adam updates the learning rate based on estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients [44]. Alternatively, the learning rate can be kept to a single constant for all parameters over the entire training procedure, which simplifies the parameter update procedure in Line (7) of Algorithm 2 to the typical SGD update (A.3). See Ref. [9] for a review of training deep neural networks.

Algorithm 2 Stochastic gradient descent with minibatching and early stopping

Input: Snapshot matrix $\boldsymbol{W}$; autoencoder $\boldsymbol{h}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{h}_{\text{dec}}(\cdot;\boldsymbol{\theta}_{\text{dec}}) \circ \boldsymbol{h}_{\text{enc}}(\boldsymbol{x};\boldsymbol{\theta}_{\text{enc}})$; SGD hyperparameters (fraction of snapshots to employ for validation $\omega \in [0,1]$; adaptive learning rate strategy; initial parameters $\boldsymbol{\theta}^{(0)}$; number of minibatches n_{batch}; maximum number of epochs n_{epoch}; early-stopping criterion)
Output: Encoder $\boldsymbol{h}_{\text{enc}}(\cdot) = \boldsymbol{h}_{\text{enc}}(\cdot;\boldsymbol{\theta}^{\star}_{\text{enc}})$; decoder $\boldsymbol{g}(\cdot) = \boldsymbol{h}_{\text{dec}}(\cdot;\boldsymbol{\theta}^{\star}_{\text{dec}})$;

1: Randomly shuffle the snapshot matrix into neural-network training and validation snapshots $[\boldsymbol{W}_{\text{train}}\ \boldsymbol{W}_{\text{val}}] = \boldsymbol{W}\boldsymbol{P}$ with $\boldsymbol{W}_{\text{val}} \in \mathbb{R}^{N\times\bar{m}}$ with $\bar{m} = \omega n_{\text{snap}}$.
2: Randomly shuffle the training snapshots into minibatches $[\boldsymbol{W}_{\text{mini},1}\ \cdots\ \boldsymbol{W}_{\text{mini},n_{\text{batch}}}] = \boldsymbol{W}_{\text{train}}\boldsymbol{P}_{\text{train}}$.
3: Initialize optimization iterations $i \leftarrow 0$
4: **for** $j = 1,\ldots,n_{\text{epoch}}$ **do**
5: **for** $k = 1,\ldots,n_{\text{batch}}$ **do**
6: Update learning rate $\boldsymbol{\eta}^{(i)}$ based on adaptive learning rate strategy
7: Perform parameter update $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \boldsymbol{\eta}^{(i)} \odot \widetilde{\nabla \boldsymbol{J}}^{(i)}(\boldsymbol{\theta}^{(i)})$
8: $i \leftarrow i+1$
9: Terminate if early-stopping criterion is satisfied on the loss $\frac{1}{\bar{m}}\sum_{i=1}^{\bar{m}} \mathcal{L}(\boldsymbol{x}^i_{\text{val}}, \boldsymbol{\theta}^{(i)})$
10: Set $\boldsymbol{\theta}^{\star} \leftarrow \boldsymbol{\theta}^{(i)}$

Appendix B. Basics of convolutional layers

This section provides notations and basic operations performed in convolutional layers. See [28] for more details of the convolution arithmetic for deep learning.

Units in convolutional layers are organized as *feature maps* $\mathcal{H}$, where each unit is connected to the local patches of the feature maps of the previous layer through a discrete convolution defined by a set of *kernels*, which is denoted by *filter bank* $\mathcal{W}$, followed by a nonlinear activation and a pooling opertion. The feature map at layer l can be considered as a 3-dimensional tensor $\mathcal{H}^l \in \mathbb{R}^{n^l_{\text{chan}}\times n^l_1 \times n^l_2}$ with element $\mathcal{H}^l_{i,j,k}$ representing a unit within channel i at row j and column k, and the filter banks at layer l can be considered as a 4-dimensional tensor $\mathcal{W}^l \in \mathbb{R}^{n^l_{\text{filter}}\times n^{l-1}_{\text{chan}}\times k^l_1 \times k^l_2}$ with element $\mathcal{W}^l_{i,j,m,n}$ connecting between a unit in channel i of the output and a unit in channel j of the input, with an offset of m rows and n columns between the output unit and the input unit. The number of filters in the filter bank is denoted by n_{filter}, and the kernel length is characterized by k_1 and k_2. Convolving a feature map $\mathcal{H}^{l-1}$ with a filter bank $\mathcal{W}^l$ can be written as

$$\mathcal{H}^l_{i,j,k} = \phi_l\left(\sum_{r,m,n} \mathcal{H}^{l-1}_{r,(j-1)\times s+m,(k-1)\times s+n}\mathcal{W}^l_{i,r,m,n} + \mathcal{B}^l_{i,j,k}\right),$$

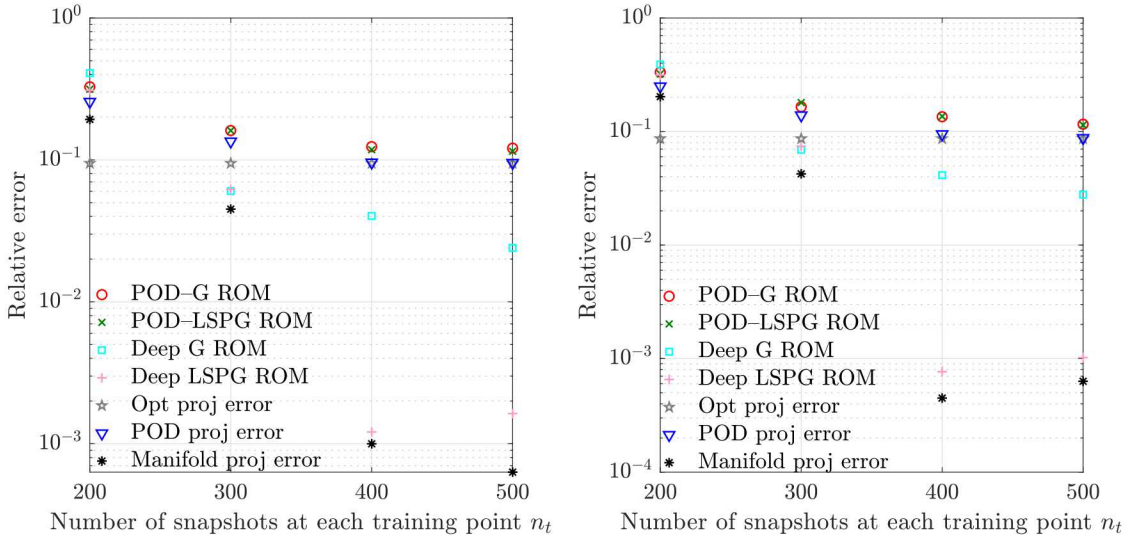
for all valid i, where $\mathcal{B}^l$ is a tensor indicating bias. Here, s denote the *stride*, which determines downsampling rate of each convolution; only every s elements is sampled in each direction in the output. By having $s > 1$, the dimension of the next feature map can be reduced by factor of s in each direction. The filter banks $\mathcal{W}$ and the biases $\mathcal{B}$ are learnable parameters, whereas the kernel length $[k_1, k_2]$ and the number of filters n_{filter}, and the stride s are the hyperparameters. After the nonlinearity, a pooling function is typically applied to the output, extracting a statistical summary of the neighboring units of the output at certain locations.

Appendix C. Additional numerical experiments: extrapolation and interpolation in time

We now perform additional investigations that assess the ability of the proposed Deep Galerkin and Deep LSPG ROMs to both extrapolate (Appendix C.1) and interpolate (Appendix C.2) in time. We also assess the effect of early stopping on the performance of these methods (Appendix C.3). All numerical experiments in this section are performed on the 1D Burgers' equation described in Section 7.1 with the same setup except when otherwise specified.

Appendix C.1. Time extrapolation

To assess time extrapolation, we collect snapshots associated with the first $n_t (\leq N_t = 500)$ time instances for each training-parameter instance and use the resulting $n_{\text{snap}} = n_t n_{\text{train}}$ snapshots to train the autoencoder for the Deep Galerkin and Deep LSPG ROMs and to compute the POD basis for the POD–Galerkin and POD–LSPG ROMs. Figure C.11 reports the relative error for each of these ROMs of dimension $p = 5$ at the online-parameter instances $\boldsymbol{\mu}^1_{\text{test}}$ and $\boldsymbol{\mu}^2_{\text{test}}$ for the number of collected snapshots varying in the set $n_t \in \{200, 300, 400, 500\}$. These results show that the proposed Deep Galerkin and Deep LSPG ROMs yield more accurate results than the POD–Galerkin and POD–LSPG ROMs for $n_t \geq 300$, with Deep LSPG yielding relative errors around 0.1% for $n_t \geq 400$. Thus, we conclude that—for this example—the proposed Deep Galerkin and Deep LSPG generally yield superior time-extrapolation results than the POD–Galerkin and POD–LSPG ROMs.



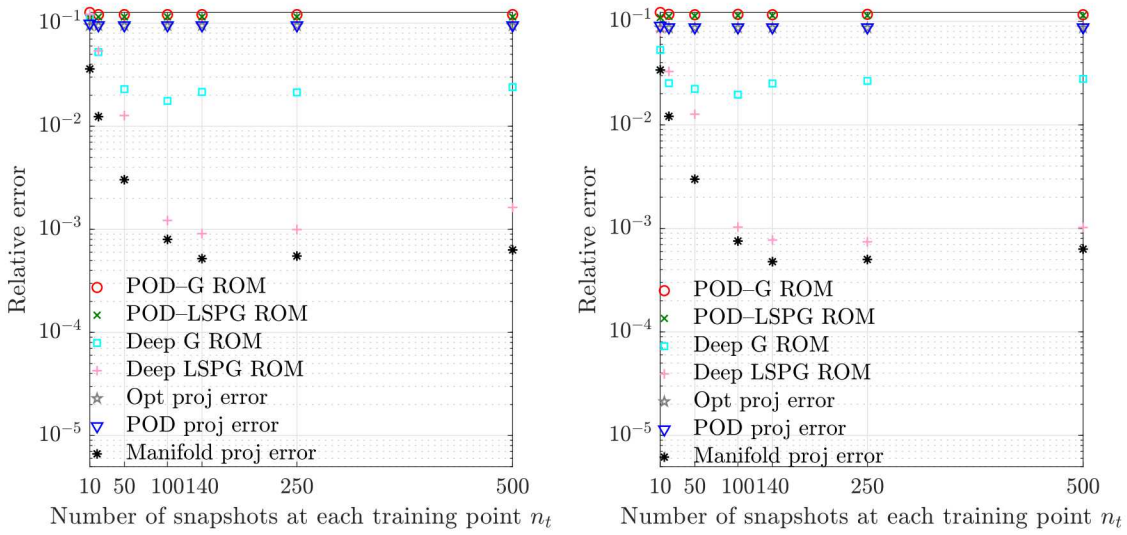
(a) Online-parameter instance $\boldsymbol{\mu}^1_{\text{test}} = (4.3, 0.021)$ with $p = 5$

(b) Online-parameter instance $\boldsymbol{\mu}^2_{\text{test}} = (5.15, 0.0285)$ with $p = 5$

Figure C.11: *1D Burgers' equation: time extrapolation.* Relative errors of the ROM solutions (Eq. (7.1)) at online-parameter instances $\boldsymbol{\mu}^1_{\text{test}}$ and $\boldsymbol{\mu}^2_{\text{test}}$ for a varying number of snapshots $n_t \in \{200, 300, 400, 500\}$ collected at each training-parameter instance. In this numerical experiment, these snapshots correspond to the state collected at the first n_t time instances at each training-parameters instance. The figure shows the relative errors of POD–ROMs with Galerkin projection (POD–G ROM, red circle) and LSPG projection (POD–LSPG ROM, green cross), and the nonlinear-manifold ROMs equipped with the decoder with time-continuous optimality (Deep G ROM, cyan square) and time-discrete optimality (Deep LSPG ROM, magenta plus sign). The figure also reports the projection error (Eq. (7.2)) of the solution onto the POD basis (POD proj error, blue triangle) and the optimal basis (Opt proj error, gray pentagram), and reports the manifold projection error of the solution (Eq. (7.3), Manifold proj error, black asterisk).

Appendix C.2. Time interpolation

To assess time interpolation, we now consider collecting n_t snapshots that are equispaced in the time interval at each training-parameter instance, with the final snapshot collected at the final time instance t^{N_t}. As in the case of time extrapolation, we then use the resulting $n_{\text{snap}} = n_t n_{\text{train}}$ snapshots to train both the autoencoder for the Deep Galerkin and Deep LSPG ROMs and to compute the POD basis for the POD–Galerkin and POD–LSPG ROMs. Figure C.12 reports the relative error for each of these ROMs of dimension $p = 5$ at the online-parameter instances $\boldsymbol{\mu}^1_{\text{test}}$ and $\boldsymbol{\mu}^2_{\text{test}}$ for the number of collected snapshots varying in the set $n_t \in \{10, 20, 50, 100, 140, 250, 500\}$. First, these results show that the POD–Galerkin and POD–LSPG ROMs are almost entirely insensitive to the number of snapshots collected within the time interval for $n_t \geq 10$. Similarly, the proposed Deep Galerkin and Deep LSPG ROMs are insensitive to the number of snapshots for $n_t \geq 100$. For $n_t \in \{10, 20, 50\}$, the Deep Galerkin and Deep LSPG ROMs yield larger errors than for $n_t \geq 100$; however, these errors are still smaller than the errors produced by the POD–Galerkin and POD–LSPG ROMs. Thus, we conclude that—for this example—the proposed methods do not exhibit accuracy degradation when only 20% of the snapshots (collected equally spaced in time) are used to train the autoencoder.



(a) Online-parameter instance $\boldsymbol{\mu}^1_{\text{test}} = (4.3, 0.021)$ with $p = 5$

(b) Online-parameter instance $\boldsymbol{\mu}^2_{\text{test}} = (5.15, 0.0285)$ with $p = 5$

Figure C.12: *1D Burgers' equation: time interpolation.* Relative errors of the ROM solutions (Eq. (7.1)) at online-parameter instances $\boldsymbol{\mu}^1_{\text{test}}$ and $\boldsymbol{\mu}^2_{\text{test}}$ for a varying number of snapshots $n_t \in \{10, 20, 50, 100, 140, 250, 500\}$ collected at each training-parameter instance. In this numerical experiment, these snapshots correspond to the state collected at n_t equally-spaced time instances at each training-parameters instance, with the final snapshot being collected at the final time t^{N_t}. The figure shows the relative errors of POD–ROMs with Galerkin projection (POD–G ROM, red circle) and LSPG projection (POD–LSPG ROM, green cross), and the nonlinear-manifold ROMs equipped with the decoder with time-continuous optimality (Deep G ROM, cyan square) and time-discrete optimality (Deep LSPG ROM, magenta plus sign). The figure also reports the projection error (Eq. (7.2)) of the solution onto the POD basis (POD proj error, blue triangle) and the optimal basis (Opt proj error, gray pentagram), and reports the manifold projection error of the solution (Eq. (7.3), Manifold proj error, black asterisk).

Appendix C.3. Early-stopping study

We now assess the effect of early stopping—a common approach in training deep neural networks for improving generalization performance—on the performance of the proposed Deep Galerkin and Deep LSPG methods. To do so, we set the reduced dimension to $p = 30$ and train the autoencoder both with and without the early stopping strategy. To train without early stopping, we set the maximum number of

epochs to $n_{\text{epoch}} = 1000$, save the network parameters at each epoch, and select the parameters (out of all $n_{\text{epoch}} = 1000$ candidates) that yield the smallest value of the loss function $\mathcal{L}$ (defined in Eq. A.2) on the validation set. Table C.2 reports the relative errors obtained by the proposed ROMs—as well as the optimal projection error onto the manifold—at the online-parameter instances. In this case, the early-stopping strategy terminated iterations after 506 epochs, while the strategy outlined above selected the network parameters arising at the 944th epoch. These results show that adopting the 'no early stopping' strategy can slightly improve performance over the early-stopping stratgy. We hypothesize this to be the case because—for this example—the training data are quite informative of the prediction task; thus, attempting to prevent overfitting essentially amounts to underfitting.

Table C.2: *1D Burgers' equation.* Relative errors of the ROM solutions (Eq. (7.1)) at online-parameter instances $\boldsymbol{\mu}^1_{\text{test}}$ and $\boldsymbol{\mu}^2_{\text{test}}$. In this experiment, the offline training is performed both with and without the early stopping strategy.

	$\boldsymbol{\mu}^1_{\text{test}} = (4.3, 0.021)$		$\boldsymbol{\mu}^2_{\text{test}} = (5.15, 0.0285)$	
Early stopping	with	without	with	without
Manifold LSPG	7.12×10^{-4}	6.78×10^{-4}	8.05×10^{-4}	4.26×10^{-4}
Manifold Galerkin	2.62×10^{-2}	1.15×10^{-2}	2.77×10^{-2}	1.20×10^{-2}
Optimal projection error	4.20×10^{-4}	2.87×10^{-4}	3.98×10^{-4}	2.81×10^{-4}

Appendix D. Computational costs

Let us consider the ith layer of the autoencoder,

$$\boldsymbol{h}_i : (\boldsymbol{x}; \boldsymbol{\Theta}_i) \mapsto \boldsymbol{\phi}_i(h_i(\boldsymbol{\Theta}_i, \boldsymbol{x})).$$

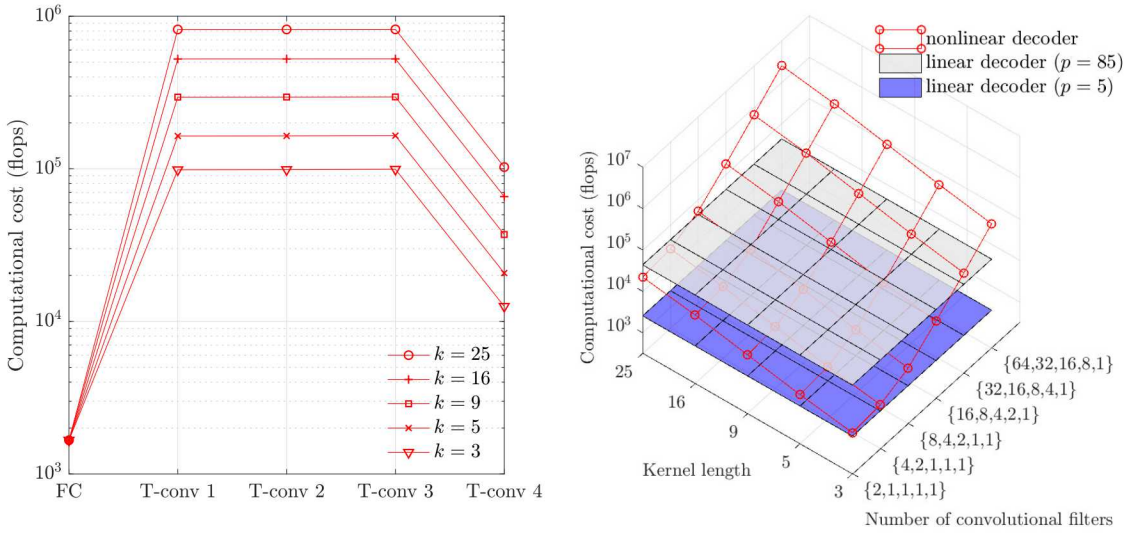
Fully-connected layers: If the ith layer is a fully-connected layer, then $\boldsymbol{\Theta}_i \in \mathbb{R}^{p_i \times (p_{i-1}+1)}$ is a real-valued matrix and the operation $\boldsymbol{h}_i$ comprises a matrix–vector product $h_i(\boldsymbol{\Theta}_i, \boldsymbol{x}) = \boldsymbol{\Theta}_i[1, \boldsymbol{x}^T]^T$, followed by an element-wise nonlinear activation $\boldsymbol{\phi}_i$. Thus, evaluating $\boldsymbol{\phi}_i(h_i(\boldsymbol{\Theta}_i, \boldsymbol{x}))$ incurs $2p_i(p_{i-1}+1) + c_{\boldsymbol{\phi}} p_i$ floating-point operations (flops), where $c_{\boldsymbol{\phi}}$ denotes the number of flops incurred by applying the nonlinear activation to a scalar.

Convolutional layers: If the ith layer is a (transposed-)convolutional layer, the operation $\boldsymbol{h}_i$ comprises discrete convolutions, followed by an element-wise nonlinear activation $\boldsymbol{\phi}_i$. The input and the output of the ith convolutional layer are 3-way tensors $\mathcal{H}^{i-1} \in \mathbb{R}^{n^{i-1}_{\text{chan}} \times n^{i-1}_1 \times n^{i-1}_2}$, and $\mathcal{H}^i \in \mathbb{R}^{n^i_{\text{chan}} \times n^i_1 \times n^i_2}$ (see Appendix B). Each element of the output tensor is computed by applying convolutional filters to a certain local region at every channel of the input tensor, where a single application of the discrete convolution with filter size $k^i_1 \times k^i_2$ incurs $2k^i_1 k^i_2 n^{i-1}_{\text{chan}}$ flops. That is, computing $p_i (= n^i_{\text{chan}} n^i_1 n^i_2)$ elements with convolutional filters of size $k^i_1 \times k^i_2$ requires $2p_i k^i_1 k^i_2 n^{i-1}_{\text{chan}}$ flops. Thus, evaluating $\boldsymbol{\phi}_i(h_i(\boldsymbol{\Theta}_i, \boldsymbol{x}))$ incurs $2p_i k^i_1 k^i_2 n^{i-1}_{\text{chan}} + c_{\boldsymbol{\phi}} p_i$ flops and, again, $c_{\boldsymbol{\phi}}$ denotes the number of flops incurred by applying the nonlinear activation to a scalar.

Restrictor, prolongator, scaling and inverse scaling operator: In this study, the restrictor and the prolongator are restricted to only change the shapes of the snapshot matrix into/from a tensor representing spatially distributed data and, thus, they do not incur floating-point operations. The scaling and the inverse scaling operators are applied to each FOM solution snapshot and incur $2N$ flops.

Figure D.13 reports the estimated computational costs of the decoder with the latent code of dimension $p = 5$ for varying choices of the hyperparameters. On the left, the figure shows the computational costs required at each layer of the decoder for varying convolutional filter sizes $k = \{25, 16, 9, 5, 3\}$ with the number of the input channels and the number of convolutional filters specified as $\{64, 32, 16, 8, 1\}$, where the first

element is the number of the input channels and the last four elements are the number of convolutional filters, and with strides $\{4, 4, 4, 2\}$. On the right, the figure shows the total computational costs for varying convolutional filter sizes $k = \{25, 16, 9, 5, 3\}$ and different numbers of input channels and convolutional filters $\{\{64, 32, 16, 8, 1\}, \{32, 16, 8, 4, 1\}, \{16, 8, 4, 2, 1\}, \{8, 4, 2, 1, 1\}, \{4, 2, 1, 1, 1\}, \{2, 1, 1, 1, 1\}\}$ and with strides $\{4, 4, 4, 2\}$. The gray plane is the computational cost for the linear decoder (e.g., POD) with the reduced dimension $p = 85$, which produces approximated solutions with the accuracy comparable to the accuracy of the manifold ROM solutions with the autoencoder configured as described in Table 1 with the reduced dimension $p = 5$. The figure also reports the computational cost for the linear decoder (e.g., POD) with the latent dimension $p = 5$ (the blue plane).



(a) Estimated computational costs of the decoder at each layer

(b) Estimated total computational costs of linear and nonlinear decoder

Figure D.13: *1D Burgers' equation: estimated computational costs of the decoder.* The computational costs required at a fully-connected layer (FC) and transposed-convolutional layers (T-conv) of the decoder with the latent code of dimension $p = 5$ for varying convolution filter sizes $k = \{25, 16, 9, 5, 3\}$ with the number of convolutional filters $\{64, 32, 16, 8, 1\}$ (left) and the total computational costs for varying convolutional filter sizes $k = \{25, 16, 9, 5, 3\}$ and varying numbers of convolutional filters $\{\{64, 32, 16, 8, 1\}, \{32, 16, 8, 4, 1\}, \{16, 8, 4, 2, 1\}, \{8, 4, 2, 1, 1\}, \{4, 2, 1, 1, 1\}, \{2, 1, 1, 1, 1\}\}$ (right). The gray and blue planes are the computational costs for the linear decoder (e.g., POD) with the reduced dimension $p = 85$ and $p = 5$, respectively.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.

[2] R. Abgrall and R. Crisovan, *Model reduction using L^1-norm minimization as an application to nonlinear hyperbolic problems*, International Journal for Numerical Methods in Fluids, 87 (2018), pp. 628–651.

[3] D. Amsallem, M. J. Zahr, and C. Farhat, *Nonlinear model order reduction based on local reduced-order bases*, International Journal for Numerical Methods in Engineering, 92 (2012), pp. 891–916.

[4] M. Bachmayr and A. Cohen, *Kolmogorov widths and low-rank approximations of parametric elliptic PDEs*, Mathematics of Computation, 86 (2017), pp. 701–724.

[5] U. Baur, C. Beattie, P. Benner, and S. Gugercin, *Interpolatory projection methods for parameterized model reduction*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2489–2518.

[6] M. Belkin and P. Niyogi, *Laplacian eigenmaps for dimensionality reduction and data representation*, Neural Computation, 15 (2003), pp. 1373–1396.

[7] P. Binev, A. Cohen, W. Dahmen, R. DeVore, G. Petrova, and P. Wojtaszczyk, *Convergence rates for greedy algorithms in reduced basis methods*, SIAM Journal on Mathematical Analysis, 43 (2011), pp. 1457–1472.

[8] C. M. Bishop, M. Svensén, and C. K. Williams, *GTM: A principled alternative to the self-organizing map*, in Advances in Neural Information Processing Systems, 1997, pp. 354–360.

[9] L. Bottou, F. E. Curtis, and J. Nocedal, *Optimization methods for large-scale machine learning*, SIAM Review, 60 (2018), pp. 223–311.

[10] M. Buffoni and K. Willcox, *Projection-based model reduction for reacting flows*, in 40th Fluid Dynamics Conference and Exhibit, AIAA Paper, 2010, p. 5008.

[11] N. Cagniart, Y. Maday, and B. Stamm, *Model order reduction for problems with large convection effects*, in Contributions to Partial Differential Equations and Applications, Springer, 2019, pp. 131–150.

[12] K. Carlberg, *Adaptive h-refinement for reduced-order models*, International Journal for Numerical Methods in Engineering, 102 (2015), pp. 1192–1210.

[13] K. Carlberg, M. Barone, and H. Antil, *Galerkin v. least-squares Petrov–Galerkin projection in nonlinear model reduction*, Journal of Computational Physics, 330 (2017), pp. 693–734.

[14] K. Carlberg, C. Bou-Mosleh, and C. Farhat, *Efficient non-linear model reduction via a least-squares Petrov–Galerkin projection and compressive tensor approximations*, International Journal for Numerical Methods in Engineering, 86 (2011), pp. 155–181.

[15] K. Carlberg, Y. Choi, and S. Sargsyan, *Conservative model reduction for finite-volume models*, Journal of Computational Physics, 371 (2018), pp. 280–314.

[16] K. Carlberg and C. Farhat, *A low-cost, goal-oriented 'compact proper orthogonal decomposition' basis for model reduction of static systems*, International Journal for Numerical Methods in Engineering, 86 (2011), pp. 381–402.

[17] K. Carlberg, C. Farhat, J. Cortial, and D. Amsallem, *The GNAT method for nonlinear model reduction: effective implementation and application to computational fluid dynamics and turbulent flows*, Journal of Computational Physics, 242 (2013), pp. 623–647.

[18] K. Carlberg, A. Jameson, M. J. Kochenderfer, J. Morton, L. Peng, and F. D. Witherden, *Recovering missing CFD data for high-order discretizations using deep neural networks and dynamics learning*, arXiv preprint arXiv:1812.01177, (2018).

[19] K. Carlberg, J. Ray, and B. van Bloemen Waanders, *Decreasing the temporal complexity for nonlinear, implicit reduced-order models by forecasting*, Computer Methods in Applied Mechanics and Engineering, 289 (2015), pp. 79–103.

[20] D. Clevert, T. Unterthiner, and S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (ELUs)*, in the 4th International Conference on Learning Representations, 2016.

[21] R. Craig and M. Bampton, *Coupling of substructures for dynamic analyses*, AIAA Journal, 6 (1968), pp. 1313–1319.

[22] R. Crisovan, D. Torlo, R. Abgrall, and S. Tokareva, *Model order reduction for parametrized nonlinear hyperbolic problems as an application to uncertainty quantification*, Journal of Computational and Applied Mathematics, 348 (2019), pp. 466–489.

[23] D. DeMers and G. W. Cottrell, *Non-linear dimensionality reduction*, in Advances in Neural Information Processing Systems, 1993, pp. 580–587.

[24] M. Dihlmann, M. Drohmann, and B. Haasdonk, *Model reduction of parametrized evolution problems using the reduced basis method with adaptive time-partitioning*, in the International Conference on Adaptive Modeling and Simulation, 2011, pp. 156–167.

[25] D. L. Donoho and C. Grimes, *Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data*, Proceedings of the National Academy of Sciences, 100 (2003), pp. 5591–5596.

[26] M. Drohmann, B. Haasdonk, and M. Ohlberger, *Adaptive reduced basis methods for nonlinear convection–diffusion equations*, in Finite Volumes for Complex Applications VI Problems & Perspectives, Springer, 2011, pp. 369–377.

[27] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, Journal of Machine Learning Research, 12 (2011), pp. 2121–2159.

[28] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning*, arXiv preprint arXiv:1603.07285, (2016).

[29] J.-F. Gerbeau and D. Lombardi, *Approximated Lax pairs for the reduced order integration of nonlinear evolution equations*, Journal of Computational Physics, 265 (2014), pp. 246–269.

[30] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.

[31] F. J. Gonzalez and M. Balajewicz, *Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems*, arXiv preprint arXiv:1808.01346, (2018).

[32] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1, MIT press Cambridge, 2016.

[33] C. Gu, *Model order reduction of nonlinear dynamical systems*, PhD thesis, UC Berkeley, 2011.

[34] S. Gugercin, A. C. Antoulas, and C. Beattie, *H_2 model reduction for large-scale linear dynamical systems*, SIAM Journal on Matrix Analysis and Applications, 30 (2008), pp. 609–638.

[35] D. Hartman and L. K. Mestha, *A deep learning framework for model reduction of dynamical systems*, in Control Technology and Applications (CCTA), 2017 IEEE Conference on, IEEE, 2017, pp. 1917–1922.

[36] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026–1034.

[37] G. E. Hinton and R. R. Salakhutdinov, *Reducing the dimensionality of data with neural networks*, Science, 313 (2006), pp. 504–507.

[38] P. Holmes, J. Lumley, and G. Berkooz, *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*, Cambridge University Press, 1996.

[39] P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley, *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*, Cambridge University Press, 2012.

[40] H. Hotelling, *Analysis of a complex of statistical variables into principal components.*, Journal of Educational Psychology, 24 (1933), p. 417.

[41] C. W. Hsu, C. C. Chang, and C. J. Lin, *A practical guide to support vector classification.* Available at: https://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.pdf, 2003.

[42] A. Iollo and D. Lombardi, *Advection modes by optimal mass transfer*, Physical Review E, 89 (2014), p. 022923.

[43] K. Kashima, *Nonlinear model reduction by deep autoencoder of noise response data*, in Decision and Control (CDC), 2016 IEEE 55th Conference on, IEEE, 2016, pp. 5750–5755.

[44] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in the 3rd International Conference on Learning Representations, 2015.

[45] T. Kohonen, *Self-organized formation of topologically correct feature maps*, Biological cybernetics, 43 (1982), pp. 59–69.

[46] S. Lall, J. E. Marsden, and S. Glavaški, *A subspace approach to balanced truncation for model reduction of nonlinear control systems*, International Journal of Robust and Nonlinear Control, 12 (2002), pp. 519–535.

[47] N. D. Lawrence, *Gaussian process latent variable models for visualisation of high dimensional data*, in Advances in Neural Information Processing Systems, 2004, pp. 329–336.

[48] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), p. 436.

[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[50] B. Lusch, J. N. Kutz, and S. L. Brunton, *Deep learning for universal linear embeddings of nonlinear dynamics*, arXiv preprint arXiv:1712.09707, (2017).

[51] L. v. d. Maaten and G. Hinton, *Visualizing data using t-SNE*, Journal of Machine Learning Research, 9 (2008), pp. 2579–2605.

[52] J. M. Martinez, *Practical quasi-Newton methods for solving nonlinear systems*, Journal of Computational and Applied Mathematics, 124 (2000), pp. 97–121.

[53] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, *Stacked convolutional auto-encoders for hierarchical feature extraction*, in International Conference on Artificial Neural Networks, Springer, 2011, pp. 52–59.

[54] M. Milano and P. Koumoutsakos, *Neural network modeling for near wall turbulent flow*, Journal of Computational Physics, 182 (2002), pp. 1–26.

[55] R. Mojgani and M. Balajewicz, *Lagrangian basis method for dimensionality reduction of convection dominated nonlinear flows*, arXiv preprint arXiv:1701.04343, (2017).

[56] B. Moore, *Principal component analysis in linear systems: Controllability, observability, and model reduction*, IEEE Transactions on Automatic Control, 26 (1981), pp. 17–32.

[57] J. Morton, F. D. Witherden, A. Jameson, and M. J. Kochenderfer, *Deep dynamical modeling and control of unsteady fluid flows*, arXiv preprint arXiv:1805.07472, (2018).

[58] N. J. Nair and M. Balajewicz, *Transported snapshot model order reduction approach for parametric, steady-state fluid flows containing parameter dependent shocks*, arXiv preprint arXiv:1712.09144, (2017).

[59] M. Ohlberger and S. Rave, *Nonlinear reduced basis approximation of parameterized evolution equations via the method of freezing*, Comptes Rendus Mathematique, 351 (2013), pp. 901–906.

[60] M. Ohlberger and S. Rave, *Reduced basis methods: Success, limitations and future challenges*, in Proceedings of ALGORITMY, Slovak University of Technology, 2016, pp. 1–12.

[61] S. E. Otto and C. W. Rowley, *Linearly-recurrent autoencoder networks for learning dynamics*, arXiv preprint arXiv:1712.01378, (2017).

[62] B. Peherstorfer and K. Willcox, *Online adaptive model reduction for nonlinear systems via low-rank updates*, SIAM Journal on Scientific Computing, 37 (2015), pp. A2123–A2150.

[63] A. Pinkus, *n-Widths in Approximation Theory*, vol. 7, Springer Science & Business Media, 2012.

[64] C. Prud'Homme, D. V. Rovas, K. Veroy, L. Machiels, Y. Maday, A. T. Patera, and G. Turinici, *Reliable real-time solution of parametrized partial differential equations: Reduced-basis output bound methods*, Journal of Fluids Engineering, 124 (2002), pp. 70–80.

[65] J. Reiss, P. Schulze, J. Sesterhenn, and V. Mehrmann, *The shifted proper orthogonal decomposition: A mode decomposition for multiple transport phenomena*, SIAM Journal on Scientific Computing, 40 (2018), pp. A1322–A1344.

[66] M. J. Rewieński, *A trajectory piecewise-linear approach to model order reduction of nonlinear dynamical systems*, PhD thesis, Massachusetts Institute of Technology, 2003.

[67] S. T. Roweis and L. K. Saul, *Nonlinear dimensionality reduction by locally linear embedding*, Science, 290 (2000), pp. 2323–2326.

[68] C. Rowley, *Model reduction for fluids, using balanced proper orthogonal decomposition*, International Journal of Bifurcation and Chaos, 15 (2005), pp. 997–1013.

[69] G. Rozza, D. B. P. Huynh, and A. T. Patera, *Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations*, Archives of Computational Methods in Engineering, 15 (2007), p. 1.

[70] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, Nature, 323 (1986), pp. 533–536.

[71] W. S. Sarle, *Neural network FAQ, periodic posting to the usenet newsgroup comp. ai. neural-nets.* Available at: ftp://ftp. sas. com/pub/neural/FAQ. html, 1997.

[72] B. Schölkopf, A. Smola, and K.-R. Müller, *Nonlinear component analysis as a kernel eigenvalue problem*, Neural Computation, 10 (1998), pp. 1299–1319.

[73] T. Taddei, S. Perotto, and A. Quarteroni, *Reduced basis techniques for nonlinear conservation laws*, ESAIM: Mathematical Modelling and Numerical Analysis, 49 (2015), pp. 787–814.

[74] N. Takeishi, Y. Kawahara, and T. Yairi, *Learning Koopman invariant subspaces for dynamic mode decomposition*, in Advances in Neural Information Processing Systems, 2017, pp. 1130–1140.

[75] J. B. Tenenbaum, V. De Silva, and J. C. Langford, *A global geometric framework for nonlinear dimensionality reduction*, Science, 290 (2000), pp. 2319–2323.

[76] T. Tieleman and G. Hinton, *Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning, 2012.

[77] C. Walder and B. Schölkopf, *Diffeomorphic dimensionality reduction*, in Advances in Neural Information Processing Systems, 2009, pp. 1713–1720.

[78] G. Welper, *h and hp-adaptive interpolation by transformed snapshots for parametric and stochastic hyperbolic PDEs*, arXiv preprint arXiv:1710.11481, (2017).

[79] G. Welper, *Interpolation of functions with parameter dependent jumps by transformed snapshots*, SIAM Journal on Scientific Computing, 39 (2017), pp. A1225–A1250.

[80] K. Willcox and J. Peraire, *Balanced model reduction via the proper orthogonal decomposition*, AIAA Journal, 40 (2002), pp. 2323–2330.

[81] M. J. Zahr and C. Farhat, *Progressive construction of a parametric reduced-order model for PDE-constrained optimization*, International Journal for Numerical Methods in Engineering, 102 (2015), pp. 1111–1135.