SAND2019-11191R

Deep Conservation: A latent dynamics model for exact satisfaction of physical conservation laws

Kookjin Lee
Sandia National Laboratories
Livermore, CA
koolee@sandia.gov

Kevin T. Carlberg
Facebook Reality Labs
Redmond, WA
kevintcarlberg@gmail.com

Abstract

This work proposes an approach for latent dynamics learning that exactly enforces physical conservation laws. The method comprises two steps. First, we compute a low-dimensional embedding of the high-dimensional dynamical-system state using deep convolutional autoencoders. This defines a low-dimensional nonlinear manifold on which the state is subsequently enforced to evolve. Second, we define a latent dynamics model that associates with a constrained optimization problem. Specifically, the objective function is defined as the sum of squares of conservation-law violations over control volumes in a finite-volume discretization of the problem; nonlinear equality constraints explicitly enforce conservation over prescribed subdomains of the problem. The resulting dynamics model—which can be considered as a projection-based reduced-order model—ensures that the time-evolution of the latent state exactly satisfies conservation laws over the prescribed subdomains. In contrast to existing methods for latent dynamics learning, this is the only method that both employs a nonlinear embedding and computes dynamics for the latent state that guarantee the satisfaction of prescribed physical properties. Numerical experiments on a benchmark advection problem illustrate the method's ability to significantly reduce the dimensionality while enforcing physical conservation.

1 Introduction

Learning a latent dynamics model for complex real-world physical processes is extremely valuable, as it provides a mechanism for modeling the dynamics of physical systems and can provide a rapid simulation tool for time-critical applications such as control and design optimization. Two main ingredients are required to learn a latent dynamics model: (1) an *embedding*, which provides a mapping between high-dimensional dynamical-system state and low-dimensional latent variables, and (2) a *dynamics model*, which provides a prediction of the time evolution of the latent variables in the latent space.

There are two primary classes of methods for learning a latent dynamics model. The first class comprises *data-driven dynamics learning*, which aims to learn both the embedding and the dynamics model in a purely data-driven manner that requires only measurements of the state/velocity. As such, this class of methods does not require explicit knowledge of the system of ordinary differential equations (ODEs) governing the high-dimensional dynamical system. These methods typically learn a nonlinear embedding (e.g., via autoencoders [29, 32, 26, 27]), and—inspired by Koopman operator theory—learn a dynamics model that is constrained to be linear. In a control [24] or reinforcement-learning context [4], the embedding and dynamics models can be learned simultaneously from observations of the state, but most approaches restrict the dynamics to be locally linear [14, 19, 33, 2].

Preprint. Under review.

The second class of methods corresponds to *projection-based dynamics learning* (often referred to as "model reduction"), which learns the embedding in a data-driven manner, but computes the dynamics model via a projection process executed on the governing system of ODEs (which must be explicitly known). As opposed to the data-driven dynamics learning, projection-based methods almost always employ a linear embedding, which is typically defined by principal component analysis (or "proper orthogonal decomposition" [17]) performed on measurements of the state. The projection process that produces the latent dynamics model requires substituting the linear embedding in the governing ODEs and enforcing orthogonality of the resulting residual to a low-dimensional linear subspace [3].

Each approach suffers from particular shortcomings. Because data-driven dynamics learning lacks explicit knowledge of the governing ODEs—and thus predicts latent dynamics separately from any computational-physics code—these methods risk severe violation of physical principles underpinning the dynamical system. On the other hand, projection-based dynamics-learning methods heavily rely on linear embeddings and, thus, exhibit limited dimensionality reduction compared with what is achievable with nonlinear embeddings [28]. Recently, this limitation has been resolved by employing a nonlinear embedding (learned by deep convolutional autoenoders) and projecting the governing ODEs onto the resulting low-dimensional nonlinear manifold [23]. Further, many projection-based dynamics learning methods perform projection using a minimum-residual formulation [7] that does not preclude the violation of important physical properties such as conservation. To mitigate this issue, recent work has proposed a projection technique that explicitly enforces conservation over subdomains by adopting a constrained least-squares formulation to define the projection [8].

In this study, we consider problems characterized by physical conservation laws, and propose Deep Conservation: a projection-based dynamics learning method that combines the advantages of Refs. [23] and [8], as the method (1) learns a nonlinear embedding via deep convolutional autoencoders, and (2) defines a dynamics model via projection process that explicitly enforces conservation over subdomains. The method assumes explicit knowledge of the governing ODEs in a particular form, namely, through a finite-volume discretization of the governing conservation laws.

2 Full-order model

2.1 Physical conservation laws

This work considers parameterized systems of *physical conservation laws*. In integral form, the governing equations correspond to

$$\frac{d}{dt}\int_{\omega} w_i(\vec{x},t;\boldsymbol{\mu})\,\mathrm{d}\vec{x} + \int_{\gamma} \boldsymbol{g}_i(\vec{x},t;\boldsymbol{\mu})\cdot\boldsymbol{n}(\vec{x})\,\mathrm{d}\vec{s}(\vec{x}) = \int_{\omega} s_i(\vec{x},t;\boldsymbol{\mu})\,\mathrm{d}\vec{x}, \quad i\in\mathbb{N}(n_u),\ \forall\omega\subseteq\Omega, \tag{1}$$

which is solved in time domain $t\in[0,T]$ given an initial condition denoted by $w_i^0:\Omega\times\mathcal{D}\to\mathbb{R}$ such that $w_i(\vec{x},0;\boldsymbol{\mu}) = w_i^0(\vec{x};\boldsymbol{\mu})$, $i\in\mathbb{N}(n_u)$, where $\mathbb{N}(a) := \{1,\dots,a\}$ such that $\boldsymbol{A}\equiv[\boldsymbol{a}_1\ \cdots\ \boldsymbol{a}_n]$. Here, ω denotes any subset of the spatial domain of interest $\Omega\subset\mathbb{R}^d$ with $d\leq 3$; $\gamma := \partial\omega$ denotes the boundary of the subset ω, while $\Gamma := \partial\Omega$ denotes the boundary of the domain Ω; $\mathrm{d}\vec{s}(\vec{x})$ denotes integration with respect to the boundary; and $w_i:\Omega\times[0,T]\times\mathcal{D}\to\mathbb{R}$, $\boldsymbol{g}_i:\Omega\times[0,T]\times\mathcal{D}\to\mathbb{R}^d$, and $s_i:\Omega\times[0,T]\times\mathcal{D}\to\mathbb{R}$, $i\in\mathbb{N}(n_u)$ denote the ith conserved variable (per unit volume), the flux associated with the ith conserved variable (per unit area per unit time), and the source associated with the ith conserved variable (per unit volume per unit time). The parameters $\boldsymbol{\mu}\in\mathcal{D}$ characterize physical properties of the governing equations, where $\mathcal{D}\subset\mathbb{R}^{n_\mu}$ denotes the parameter space. Finally, $\boldsymbol{n}:\gamma\to\mathbb{R}^d$ denotes the outward unit normal to ω. We emphasize that equations (1) describe conservation of *any* set of variables w_i, $i\in\mathbb{N}(n_u)$, given their respective flux $\boldsymbol{g}_i$ and source s_i functions.

2.2 Finite-volume discretization

To discretize the governing equations (1), we apply the finite-volume method [25, 11], as it divides the spatial domain into many control volumes and explicitly enforces conservation (Eq. (1)) over prescribed control volumes. In particular, we assume that the spatial domain Ω has been partitioned into a mesh $\mathcal{M}$, of $N_\Omega\in\mathbb{N}$ non-overlapping (closed, connected) control volumes $\Omega_i\subseteq\Omega$, $i\in\mathbb{N}(N_\Omega)$. We define the mesh as $\mathcal{M} := \{\Omega_i\}_{i=1}^{N_\Omega}$, and denote the boundary of the ith control volume by $\Gamma_i := \partial\Omega_i$. The ith control-volume boundary is partitioned into a set of faces denoted by $\mathcal{E}_i$

such that $\Gamma_i = \{\vec{x} \,|\, \vec{x} \in e,\ \forall e \in \mathcal{E}_i,\ i \in \mathbb{N}(|\mathcal{E}_i|)\}$. Then the full set of N_e faces within the mesh is $\mathcal{E} \equiv \{e_i\}_{i=1}^{N_e} := \cup_{i=1}^{N_\Omega} \mathcal{E}_i$. Enforcing conservation (1) on each control volume in the mesh yields

$$\frac{d}{dt}\int_{\Omega_j} w_i(\vec{x},t;\boldsymbol{\mu})\,\mathrm{d}\vec{x} + \int_{\Gamma_j} \boldsymbol{g}_i(\vec{x},t;\boldsymbol{\mu})\cdot\boldsymbol{n}_j(\vec{x})\,\mathrm{d}\vec{s}(\vec{x}) = \int_{\Omega_j} s_i(\vec{x},t;\boldsymbol{\mu})\,\mathrm{d}\vec{x}, \quad i \in \mathbb{N}(n_u),\ j \in \mathbb{N}(N_\Omega), \tag{2}$$

where $\boldsymbol{n}_j : \Gamma_j \to \mathbb{R}^d$ denotes the unit normal to control volume Ω_j. Finite-volume schemes complete the spatial discretization by forming a state vector $\boldsymbol{x} : [0,T] \times \mathcal{D} \to \mathbb{R}^N$ with $N = N_\Omega n_u$ such that

$$x_{\mathcal{I}(i,j)}(t;\boldsymbol{\mu}) = \frac{1}{|\Omega_j|}\int_{\Omega_j} w_i(\vec{x},t;\boldsymbol{\mu})\,\mathrm{d}\vec{x}, \quad i \in \mathbb{N}(n_u),\ j \in \mathbb{N}(N_\Omega), \tag{3}$$

where $\mathcal{I} : \mathbb{N}(n_u) \times \mathbb{N}(N_\Omega) \to \mathbb{N}(N)$ denotes a mapping from conservation-law index and control-volume index to degree of freedom, and a velocity vector $\boldsymbol{f} : (\boldsymbol{w},\tau;\boldsymbol{\nu}) \mapsto \boldsymbol{f}^g(\boldsymbol{w},\tau;\boldsymbol{\nu}) + \boldsymbol{f}^s(\boldsymbol{w},\tau;\boldsymbol{\nu})$ with $\boldsymbol{f}^g, \boldsymbol{f}^s : \mathbb{R}^N \times [0,T] \times \mathcal{D} \to \mathbb{R}^N$ whose elements consist of

$$\begin{aligned} f^g_{\mathcal{I}(i,j)}(\boldsymbol{x},t;\boldsymbol{\mu}) &= -\frac{1}{|\Omega_j|}\int_{\Gamma_j} \boldsymbol{g}_i^{\mathrm{FV}}(\boldsymbol{x};\vec{x},t;\boldsymbol{\mu})\cdot\boldsymbol{n}_j(\vec{x})\,\mathrm{d}\vec{s}(\vec{x}) \\ f^s_{\mathcal{I}(i,j)}(\boldsymbol{x},t;\boldsymbol{\mu}) &= \frac{1}{|\Omega_j|}\int_{\Omega_j} s_i^{\mathrm{FV}}(\boldsymbol{x};\vec{x},t;\boldsymbol{\mu})\,\mathrm{d}\vec{x} \end{aligned} \tag{4}$$

for $i \in \mathbb{N}(n_u)$, $j \in \mathbb{N}(N_\Omega)$. Here, $\boldsymbol{g}_i^{\mathrm{FV}} : \mathbb{R}^N \times \Omega \times [0,T] \times \mathcal{D} \to \mathbb{R}^d$ and $s_i^{\mathrm{FV}} : \mathbb{R}^N \times \Omega \times [0,T] \times \mathcal{D} \to \mathbb{R}$, $i \in \mathbb{N}(n_u)$ denote the approximated flux and source, respectively, associated with the ith conserved variable. Substituting $\int_{\Omega_j} w_i(\vec{x},t;\boldsymbol{\mu})\,\mathrm{d}\vec{x} \leftarrow |\Omega_j| x_{\mathcal{I}(i,j)}(t;\boldsymbol{\mu})$, $\boldsymbol{g}_i \leftarrow \boldsymbol{g}_i^{\mathrm{FV}}$, and $s_i \leftarrow s_i^{\mathrm{FV}}$ in Eq. (2) and dividing by $|\Omega_j|$ yields

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},t;\boldsymbol{\mu}), \qquad \boldsymbol{x}(0;\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu}), \tag{5}$$

where $x^0_{\mathcal{I}(i,j)}(\boldsymbol{\mu}) := \frac{1}{|\Omega_i|}\int_{\Omega_j} w_i^0(\vec{x};\boldsymbol{\mu})\,\mathrm{d}\vec{x}$ denotes the parameterized initial condition. This is a parameterized system of nonlinear ordinary differential equations (ODEs) characterizing an initial value problem, which we consider to be our full-order model (FOM). We thus refer to Eq. (5) as the FOM ODE.

Numerically solving the FOM ODE (5) requires application of a time-discretization method. For simplicity, this work restricts attention to linear multistep methods. A linear k-step method applied to numerically solve the FOM ODE (5) leads to solving the system of algebraic equations

$$\boldsymbol{r}^n(\boldsymbol{x}^n;\boldsymbol{\mu}) = \boldsymbol{0}, \quad n = 1,\ldots,N_t, \tag{6}$$

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

$$\boldsymbol{r}^n : (\boldsymbol{\xi};\boldsymbol{\nu}) \mapsto \alpha_0\boldsymbol{\xi} - \Delta t\beta_0\boldsymbol{f}(\boldsymbol{\xi},t^n;\boldsymbol{\nu}) + \sum_{j=1}^{k}\alpha_j\boldsymbol{x}^{n-j} - \Delta t\sum_{j=1}^{k}\beta_j\boldsymbol{f}(\boldsymbol{x}^{n-j},t^{n-j};\boldsymbol{\nu}). \tag{7}$$

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, $j = 0,\ldots,k$ with $\sum_{j=0}^{k}\alpha_j = 0$ define a particular linear multistep scheme. These methods are implicit if $\beta_0 \neq 0$. We refer to Eq. (6) as the FOM OΔE.

2.3 Computational barrier: time-critical problems

Many problems in science and engineering are *time critical* in nature, meaning that the solution to the FOM OΔE (6) must be computed within a specified computational budget (e.g., less than 0.1 core–hours) for arbitrary parameter instances $\boldsymbol{\mu} \in \mathcal{D}$. When the full-order model is truly high fidelity, the computational mesh $\mathcal{M}$ often becomes very fine, which can yield an extremely large state-space dimension N (e.g., $N \sim 10^7$). This introduces a *de facto* computational barrier: the full-order model is too computationally expensive to solve within the prescribed computational budget. Such cases demand a method for *approximately* solving the full-order model while retaining high levels of accuracy. We now present a method that (1) computes a nonlinear embedding of the state using deep convolutional autoencoders, and (2) computes a dynamics model for the resulting latent state that exactly satisfies the physical conservation laws over *subdomains* comprising unions control volumes of the mesh.

3 Nonlinear embedding: deep convolutional autoencoders

3.1 Deep convolutional autoencoders

Autoencoders [10, 15] consist of two parts: an encoder $\boldsymbol{h}_{\text{enc}}(\cdot;\boldsymbol{\theta}_{\text{enc}}) : \mathbb{R}^N \to \mathbb{R}^p$ and a decoder $\boldsymbol{h}_{\text{dec}}(\cdot;\boldsymbol{\theta}_{\text{dec}}) : \mathbb{R}^p \to \mathbb{R}^N$ with latent-state dimension $p \ll N$ such that the autoencoder is

$$\boldsymbol{h} : (\boldsymbol{x};\boldsymbol{\theta}_{\text{enc}},\boldsymbol{\theta}_{\text{dec}}) \mapsto \boldsymbol{h}_{\text{dec}}(\cdot;\boldsymbol{\theta}_{\text{dec}}) \circ \boldsymbol{h}_{\text{enc}}(\boldsymbol{x};\boldsymbol{\theta}_{\text{enc}}),$$

where $\boldsymbol{\theta}_{\text{enc}}$ and $\boldsymbol{\theta}_{\text{dec}}$ denote parameters associated with the encoder and decoder, respectively.

Because we are considering finite-volume discretizations of conservation laws, the state elements $x_{\mathcal{I}(i,j)}$, $j \in \mathbb{N}(N_\Omega)$ correspond to the value of the ith conserved variable w_i distributed across the N_Ω control volumes characterizing the mesh $\mathcal{M}$. As such, we can interpret the state $\boldsymbol{x} \in \mathbb{R}^N$ as representing the distribution of spatially distributed data with n_u channels. This is precisely the format required by convolutional neural networks, which often generalize well to unseen test data [22, 21] because they exploit local connectivity, employ parameter sharing, and exhibit translation equivariance [13, 22]. Thus, we leverage the connection between conservation laws and image data, and employ convolutional autoencoders.

3.2 Offline training

The first step of offline training is snapshot-based data collection. This requires solving the FOM OΔE (6) 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

$$\mathbf{X} := \left[\mathbf{X}(\boldsymbol{\mu}^1_{\text{train}}) \;\cdots\; \mathbf{X}(\boldsymbol{\mu}^{n_{\text{train}}}_{\text{train}})\right] \in \mathbb{R}^{N \times n_{\text{snap}}} \tag{8}$$

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

To improve numerical stability of the gradient-based optimization for training, the first layer of the proposed autoencoder applies an affine scaling operator $\boldsymbol{S}$ to ensure that all elements of the training data lie between zero and one [18, 31]. Then the autoencoder reformats the input vector into a tensor compatible with convolutional layers; the last layer applies the inverse of this scaling operator $\boldsymbol{S}^{-1}$ and subsequently reformats the data into a vector. Appendix A provides details related to the scaling and the inverse scaling operators.

Given the autoencoder architecture $\boldsymbol{h}(\boldsymbol{x};\boldsymbol{\theta}_{\text{enc}},\boldsymbol{\theta}_{\text{dec}})$, we compute parameter values $(\boldsymbol{\theta}^\star_{\text{enc}},\boldsymbol{\theta}^\star_{\text{dec}})$ by (approximately) solving

$$\underset{\boldsymbol{\theta}_{\text{enc}},\boldsymbol{\theta}_{\text{dec}}}{\text{minimize}} \sum_{i=1}^{n_{\text{snap}}} \|\mathbf{x}^i - \boldsymbol{h}(\mathbf{x}^i;\boldsymbol{\theta}_{\text{enc}},\boldsymbol{\theta}_{\text{dec}})\|_2^2$$

using stochastic gradient descent (SGD) with minibatching and early stopping [5].

3.3 Nonlinear embedding

Given the trained autoencoder $\boldsymbol{h} : (\boldsymbol{x};\boldsymbol{\theta}^\star_{\text{enc}},\boldsymbol{\theta}^\star_{\text{dec}}) \mapsto \boldsymbol{h}_{\text{dec}}(\cdot;\boldsymbol{\theta}^\star_{\text{dec}}) \circ \boldsymbol{h}_{\text{enc}}(\boldsymbol{x};\boldsymbol{\theta}^\star_{\text{enc}})$, we construct a nonlinear embedding by defining a low-dimensional nonlinear "trial manifold" on which we will restrict the approximated state to evolve. In particular, we define this manifold from the extrinsic view as $\mathcal{S} := \{\boldsymbol{d}(\hat{\boldsymbol{\xi}}) \,|\, \hat{\boldsymbol{\xi}} \in \mathbb{R}^p\}$, where the parameterization function is defined from the trained decoder as $\boldsymbol{d} : \hat{\boldsymbol{\xi}} \mapsto \boldsymbol{h}_{\text{dec}}(\hat{\boldsymbol{\xi}};\boldsymbol{\theta}^\star_{\text{dec}})$ with $\boldsymbol{d} : \mathbb{R}^p \to \mathbb{R}^N$. We subsequently approximate the state as $\boldsymbol{x}(t;\boldsymbol{\mu}) \approx \tilde{\boldsymbol{x}}(t;\boldsymbol{\mu}) \in \boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \mathcal{S}$, where $\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu}) - \boldsymbol{h}_{\text{enc}}(\boldsymbol{x}^0(\boldsymbol{\mu});\boldsymbol{\theta}^\star_{\text{enc}})$ is the reference state. This approximation can be expressed algebraically as

$$\tilde{\boldsymbol{x}}(t;\boldsymbol{\mu}) = \boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{d}(\hat{\boldsymbol{x}}(t;\boldsymbol{\mu})), \tag{9}$$

which elucidates the mapping between the latent state $\hat{\boldsymbol{x}} : \mathbb{R}_+ \times \mathcal{D} \to \mathbb{R}^p$ and the approximated state $\tilde{\boldsymbol{x}} : \mathbb{R}_+ \times \mathcal{D} \to \mathbb{R}^N$.

Remark 3.1 (Linear embedding via proper orthogonal decomposition) *Classical methods for projection-based dynamics learning employ proper orthogonal decomposition (POD) [17]—which is closely related to principal component analysis—to construct a linear embedding. Using the above notation, POD computes the singular value decomposition of the snapshot matrix* $\mathbf{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}$

and sets a "trial basis matrix" $\boldsymbol{\Phi} \in \mathbb{R}^{N\times p}$ to be equal to the first p columns of $\boldsymbol{U}$. Then, these methods define low-dimensional linear "trial subspace" such that the state is approximated as $\boldsymbol{x}(t;\boldsymbol{\mu}) \approx \tilde{\boldsymbol{x}}(t;\boldsymbol{\mu}) \in \boldsymbol{x}^0(\boldsymbol{\mu}) + \mathrm{Ran}(\boldsymbol{\Phi})$, which is equivalent to the approximation in Eq. (9) with $\boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu})$ and a linear parameterization function $\boldsymbol{d} : \hat{\boldsymbol{\xi}} \mapsto \boldsymbol{\Phi}\hat{\boldsymbol{\xi}}$.

4 Dynamics model: conservation-enforcing projection

We now describe the proposed projection-based dynamics model, starting with deep least-squares Petrov–Galerkin (LSPG) projection (proposed in Ref. [23]) in Section 4.1, and proceeding with the proposed Deep Conservation projection in Section 4.2.

4.1 Deep LSPG projection

To construct a dynamics model for the approximated state $\tilde{\boldsymbol{x}}$, the Deep LSPG method [23] simply substitutes $\boldsymbol{x} \leftarrow \tilde{\boldsymbol{x}}$ defined in Eq. (9) into the FOM OΔE (6) and minimizes the ℓ^2-norm of the residual, i.e.,

$$\hat{\boldsymbol{x}}^n(\boldsymbol{\mu}) = \underset{\hat{\boldsymbol{\xi}}\in\mathbb{R}^p}{\arg\min} \left\| \boldsymbol{r}^n\left(\boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\mu}) + \boldsymbol{d}(\hat{\boldsymbol{\xi}});\boldsymbol{\mu}\right)\right\|_2^2, \tag{10}$$

which is solved sequentially for $n = 1,\dots,N_t$ with initial latent state $\hat{\boldsymbol{x}}^0(\boldsymbol{\mu}) = \boldsymbol{h}_{\mathrm{enc}}(\boldsymbol{x}^0(\boldsymbol{\mu});\boldsymbol{\theta}^\star_{\mathrm{enc}})$.[1] Eq. (10) defines the (discrete-time) dynamics model for the latent state associated with Deep LSPG projetion. The nonlinear least-squares problem (10) can be solved using, for example, the Gauss–Newton method, which leads to the iterations

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

for $k = 0,\dots,K$. Here, $\hat{\boldsymbol{x}}^{n(0)}$ is the initial guess (often taken to be $\hat{\boldsymbol{x}}^{n-1}$); $\alpha^{n(k)} \in \mathbb{R}$ is a step length chosen to satisfy the strong Wolfe conditions for global convergence; and $\boldsymbol{\Psi}^n : \mathbb{R}^p \times \mathcal{D} \to \mathbb{R}^{N\times p}$ is

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

with $\boldsymbol{J} : \hat{\boldsymbol{\xi}} \mapsto \frac{d\boldsymbol{d}}{d\hat{\boldsymbol{\xi}}}(\hat{\boldsymbol{\xi}})$ and $\boldsymbol{J} : \mathbb{R}^p \to \mathbb{R}^{N\times p}$ the Jacobian of the decoder.

Remark 4.1 (POD–LSPG projection) *POD–LSPG projection [6, 7] employs the same residual-minimization projection (10), but with reference state $\boldsymbol{x}_{\mathrm{ref}}(\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu})$ and linear parameterization function $\boldsymbol{d} : \hat{\boldsymbol{\xi}} \mapsto \boldsymbol{\Phi}\hat{\boldsymbol{\xi}}$ as described in Remark 3.1.*

4.2 Deep Conservation projection

We now derive the proposed Deep Conservation projection, which effectively combines Deep LSPG projection [23] just described with conservative LSPG projection [8], which was developed for linear embeddings only.

To begin, we decompose the mesh $\mathcal{M}$ into subdomains, each of which comprises the union of control volumes. That is, we define a decomposed mesh $\bar{\mathcal{M}}$ of $N_{\bar{\Omega}} (\leq N_\Omega)$ subdomains $\bar{\Omega}_i = \cup_{j\in\mathcal{K}\subseteq\mathbb{N}(N_\Omega)}\Omega_j$, $i \in \mathbb{N}(N_{\bar{\Omega}})$ with $\bar{\mathcal{M}} := \{\bar{\Omega}_i\}_{i=1}^{N_{\bar{\Omega}}}$. Denoting the boundary of the ith subdomain by $\bar{\Gamma}_i := \partial\bar{\Omega}_i$, we have $\bar{\Gamma}_i = \{\vec{x}\,|\,\vec{x}\in e,\ \forall e \in \bar{\mathcal{E}}_i,\ i \in \mathbb{N}(|\bar{\mathcal{E}}_i|)\} \subseteq \cup_{j=1}^{N_\Omega}\Gamma_j, i \in \mathbb{N}(N_{\bar{\Omega}})$ with $\bar{\mathcal{E}}_i \subseteq \mathcal{E}$ representing the set of faces belonging to the ith subdomain. We denote the full set of faces within the decomposed mesh by $\bar{\mathcal{E}} := \cup_{i=1}^{N_{\bar{\Omega}}}\bar{\mathcal{E}}_i \subseteq \mathcal{E}$. Note that the global domain can be considered by employing $\bar{\mathcal{M}} = \bar{\mathcal{M}}_{\mathrm{global}}$, which is characterized by $N_{\bar{\Omega}} = 1$ subdomain that corresponds to the global domain.

Enforcing conservation (1) on each subdomain in the decomposed mesh yields

$$\frac{d}{dt}\int_{\bar{\Omega}_j} w_i(\vec{x},t;\boldsymbol{\mu})\,\mathrm{d}\vec{x} + \int_{\bar{\Gamma}_j}\boldsymbol{g}_i(\vec{x},t;\boldsymbol{\mu})\cdot\bar{\boldsymbol{n}}_j(\vec{x})\,\mathrm{d}\vec{s}(\vec{x}) = \int_{\bar{\Omega}_j} s_i(\vec{x},t;\boldsymbol{\mu})\,\mathrm{d}\vec{x}, \quad i \in \mathbb{N}(n_u),\ j \in \mathbb{N}(N_{\bar{\Omega}}), \tag{11}$$

[1] This initial latent state, combined with the definition of the reference state, ensures $\tilde{\boldsymbol{x}}^0(\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu})$.

where $\bar{\boldsymbol{n}}_j : \bar{\Gamma}_j \to \mathbb{R}^d$ denotes the unit normal to subdomain $\bar{\Omega}_j$. We propose applying the same finite-volume discretization employed to discretize the control-volume conservation equations (2) to the subdomain conservation equations (11). To accomplish this, we introduce a "decomposed" state vector $\bar{\boldsymbol{x}} : \mathbb{R}^N \times [0,T] \times \mathcal{D} \to \mathbb{R}^{\bar{N}}$ with $\bar{N} = N_{\bar{\Omega}} n_u$ and elements

$$\bar{x}_{\bar{\mathcal{I}}(i,j)}(\boldsymbol{x}, t; \boldsymbol{\mu}) = \frac{1}{|\bar{\Omega}_j|} \int_{\bar{\Omega}_j} w_i(\vec{x}, t; \boldsymbol{\mu}) \, \mathrm{d}\vec{x}, \quad i \in \mathbb{N}(n_u),\ j \in \mathbb{N}(N_{\bar{\Omega}}), \tag{12}$$

where $\bar{\mathcal{I}} : \mathbb{N}(n_u) \times \mathbb{N}(N_{\bar{\Omega}}) \to \mathbb{N}(\bar{N})$ denotes a mapping from conservation-law index and subdomain index to decomposed degree of freedom. The decomposed state vector can be computed from the state vector $\boldsymbol{x}$ as

$$\bar{\boldsymbol{x}}(\boldsymbol{x}) = \bar{\boldsymbol{C}} \boldsymbol{x}$$

where $\bar{\boldsymbol{C}} \in \mathbb{R}_+^{\bar{N} \times N}$ has elements $\bar{\boldsymbol{C}}_{\bar{\mathcal{I}}_{(i,j)}, \mathcal{I}_{(l,k)}} = \frac{|\Omega_k|}{|\bar{\Omega}_j|} \delta_{il} I(\Omega_k \subseteq \bar{\Omega}_j)$, where I is the indicator function, which evaluates to one if its argument is true, and zero if its argument is false.

Similarly, the velocity associated with the finite-volume scheme applied to subdomain conservation can be expressed as

$$\bar{\boldsymbol{f}}(\boldsymbol{x}, t; \boldsymbol{\mu}) = \bar{\boldsymbol{C}} \boldsymbol{f}(\boldsymbol{x}, t; \boldsymbol{\mu}), \tag{13}$$

such that subdomain conservation can be expressed as

$$\bar{\boldsymbol{C}} \dot{\boldsymbol{x}} = \bar{\boldsymbol{C}} \boldsymbol{f}(\boldsymbol{x}, t; \boldsymbol{\mu}). \tag{14}$$

Applying a linear multistep scheme to discretize (14) in time yields

$$\bar{\boldsymbol{C}} \boldsymbol{r}^n(\boldsymbol{x}^n; \boldsymbol{\mu}) = \boldsymbol{0}. \tag{15}$$

For the explanation on the derivation of Eqs. (13)–(15), we refer readers Appendix B.

Remark 4.2 (Lack of conservation for Deep LSPG) *We note that the Deep LSPG dynamics model (10) in general violates the conservation laws underlying the dynamical system of interest. This occurs because the objective function in (10) is generally nonzero at the solution, and thus conservation condition (15) is not generally satisfied for any decomposed mesh $\bar{\mathcal{M}}$. This lack of conservation can lead to spurious generation or dissipation of physical quantities that should be conserved in principle (e.g., mass, momentum, energy).*

We aim to remedy this primary shortcoming of Deep LSPG with the proposed Deep Conservation projection method. In particular, we define the Deep Conservation dynamics model by equipping the nonlinear least-squares problem (10) with *nonlinear equality constraints* corresponding to Eq. (15), which has the effect of explicitly enforcing conservation over the decomposed mesh $\bar{\mathcal{M}}$. In particular, the Deep Conservation dynamics model computes latent states $\hat{\boldsymbol{x}}^n(\boldsymbol{\mu})$, $n = 1, \ldots, N_t$ that satisfy

$$\begin{aligned} &\underset{\hat{\boldsymbol{\xi}} \in \mathbb{R}^p}{\text{minimize}} \left\| \boldsymbol{r}^n \left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{d}(\hat{\boldsymbol{\xi}}; \boldsymbol{\mu}) \right) \right\|_2^2 \\ &\text{subject to } \bar{\boldsymbol{C}} \boldsymbol{r}^n \left(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{d}(\hat{\boldsymbol{\xi}}; \boldsymbol{\mu}) \right) = \boldsymbol{0}. \end{aligned} \tag{16}$$

To solve the problem (16) at each time instance, we follow the approach considered in [8] and apply sequential quadratic programming with the Gauss–Newton Hessian approximation, which leads to iterations

$$\begin{aligned} &\begin{bmatrix} \boldsymbol{\Psi}^n(\hat{\boldsymbol{x}}^{n(k)}; \boldsymbol{\mu})^T \boldsymbol{\Psi}^n(\hat{\boldsymbol{x}}^{n(k)}; \boldsymbol{\mu}) & \boldsymbol{\Psi}^n(\hat{\boldsymbol{x}}^{n(k)}; \boldsymbol{\mu})^T \bar{\boldsymbol{C}}^T \\ \bar{\boldsymbol{C}} \boldsymbol{\Psi}^n(\hat{\boldsymbol{x}}^{n(k)}; \boldsymbol{\mu}) & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \delta \hat{\boldsymbol{x}}^{n(k)} \\ \delta \boldsymbol{\lambda}^{n(k)} \end{bmatrix} \\ &\qquad = - \begin{bmatrix} \boldsymbol{\Psi}^n(\hat{\boldsymbol{x}}^{n(k)}; \boldsymbol{\mu})^T \boldsymbol{r}^n(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{d}(\hat{\boldsymbol{x}}^{n(k)}); \boldsymbol{\mu}) \\ \bar{\boldsymbol{C}} \boldsymbol{r}^n(\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) + \boldsymbol{d}(\hat{\boldsymbol{x}}^{n(k)}); \boldsymbol{\mu})) \end{bmatrix} \\ &\begin{bmatrix} \hat{\boldsymbol{x}}^{n(k+1)} \\ \boldsymbol{\lambda}^{n(k+1)} \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{x}}^{n(k)} \\ \boldsymbol{\lambda}^{n(k)} \end{bmatrix} + \eta^{n(k)} \begin{bmatrix} \delta \hat{\boldsymbol{x}}^{n(k)} \\ \delta \boldsymbol{\lambda}^{n(k)} \end{bmatrix} \end{aligned}$$

for $k = 0, \ldots, K$, where $\boldsymbol{\lambda}^{n(k)} \in \mathbb{R}^{\bar{N}}$ denotes Lagrange multipliers at time instance n and iteration k and $\eta^{n(k)} \in \mathbb{R}$ is the step length that can be chosen, e.g., to satisfy the strong Wolfe conditions to ensure global convergence to a local solution of (16).

Remark 4.3 (Conservative LSPG projection) *Conservative LSPG projection [8] employs the same formulation (16), but with reference state $\boldsymbol{x}_{\text{ref}}(\boldsymbol{\mu}) = \boldsymbol{x}^0(\boldsymbol{\mu})$ and linear parameterization function $\boldsymbol{d} : \hat{\boldsymbol{\xi}} \mapsto \boldsymbol{\Phi} \hat{\boldsymbol{\xi}}$ as described in Remark 3.1.*

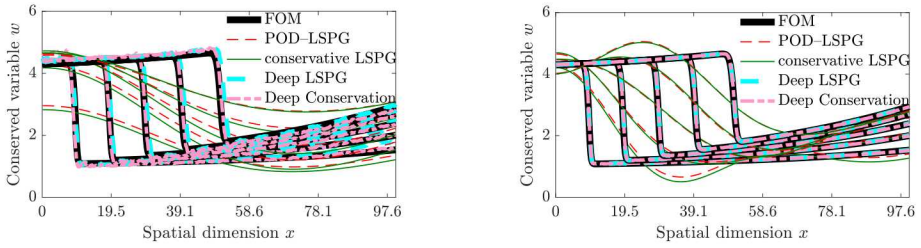
5 Numerical experiments

This section assesses the performance of (1) the proposed Deep Conservation projection, (2) Deep LSPG projection, which also employs a nonlinear embedding but does not enforce conservation, (3) POD–LSPG projection, which employs a linear embedding and does not enforce conservation, and (4) conservative LSPG projection, which employs a linear embedding but enforces conservation. We consider a parameterized Burgers' equation, as it is a benchmark advection-dominated problem. We employ the numerical PDE tools and projection functionality provided by `pyMORTestbed` [34], and we construct the autoencoder using `TensorFlow` 1.11.0 [1].

The Deep Conservation and Deep LSPG methods employ a 10-layer convolutional autoencoder. The encoder $\boldsymbol{h}_{\text{enc}}$ consists of 5 layers with 4 convolutional layers, followed by 1 fully-connected layer. The decoder $\boldsymbol{h}_{\text{dec}}$ consists of 1 fully-connected layer, followed by 4 transposed-convolution layers. The latent state of the autoencoder is of dimension p, which will vary during the experiments to define different latent-state dimensions. The length of the convolutional kernels in the encoder and the decoder is 25; the numbers of kernel filters in each convolutional and transposed-convolutional layer are $\{8, 16, 32, 64\}$ and $\{32, 16, 8, 1\}$; the stride is configured as $\{2, 4, 4, 4\}$ and $\{4, 4, 4, 2\}$; the "SAME" padding strategy is used; and no pooling is used.For the nonlinear activation functions, we use exponential linear units (ELU) [9], and no activation function in the output layer.

For the model problem, we consider a parameterized inviscid Burgers' equation [30], where the system is governed by a conservation law of the form (1) with $n_u = 1$, $d = 1$, $\Omega = [0, 100]$, $\boldsymbol{g}(x, t; \boldsymbol{\mu}) = \frac{w(x,t;\boldsymbol{\mu})^2}{2}$, $s(x, t; \boldsymbol{\mu}) = 0.02e^{\mu_2 x}$ with initial and boundary conditions $w(0, t; \boldsymbol{\mu}) = \mu_1, \forall t \in [0, T]$, $w(x, 0) = 1, \quad \forall x \in [0, 100]$. There are $n_\mu = 2$ parameters (i.e., $\boldsymbol{\mu} = (\mu_1, \mu_2)$) with the parameter domain $\mathcal{D} = [4.25, 5.5] \times [0.015, 0.03]$, and the final time is set to $T = 35$. We apply Godunov's scheme [16], which is a finite-volume scheme, with $N_\Omega = 512$ control volumes, which results in a system of ODEs of the form (5) with $N = 512$ spatial degrees of freedom. For time discretization, we use the backward-Euler scheme (i.e., $k = 1$, $\alpha_0 = \beta_0 = 1$, $\alpha_1 = -1$, and $\beta_1 = 0$ in Eq. (7)). 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. After collecting the snapshots, we split the snapshot matrix (8) into a training set and a validation set; the fraction of snapshots to use for validation is 10%. Then we compute optimal parameters $(\boldsymbol{\theta}^\star_{\text{enc}}, \boldsymbol{\theta}^\star_{\text{dec}})$ using Adam optimizer [20] with an initial uniform learning rate $\eta = 10^{-4}$ and initial parameters $\boldsymbol{\theta}^{(0)}$ are computed via Xavier initialization [12]. The number of minibatches determined by a fixed batch size of 20; a maximum number of epochs is $n_{\text{epoch}} = 1000$; and early-stopping is enforced if the loss on the validation set fails to decrease over 100 epochs. For the online stage, we consider an online-parameter instance $\boldsymbol{\mu}^1_{\text{test}} = (4.3, 0.021)$, which are not included in $\mathcal{D}_{\text{train}}$. The FOM solution for the given parameter instance is illustrated in Appendix C.



(a) Latent-space dimension, $p = 2$

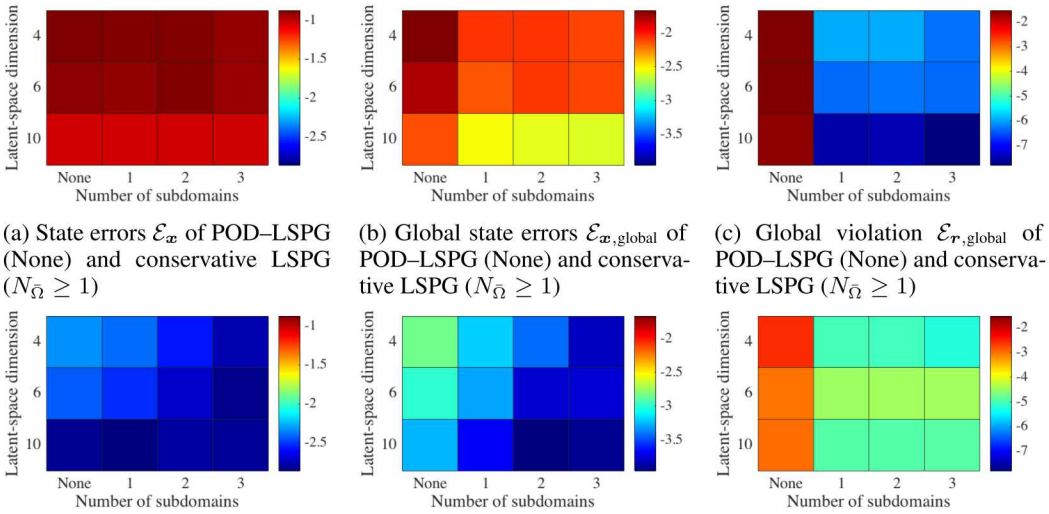
(b) Latent-space dimension, $p = 4$

Figure 1: Online solutions at time instances $t = \{3.5, 7.0, 10.5, 14, 17.5\}$ computed by the FOM, POD–LSPG, conservative LSPG, Deep LSPG and Deep Conservation. All conservative methods employ $N_{\bar{\Omega}} = 1$ subdomains.

Figure 1 reports solutions at five different time instances computed using FOM and all of the considered projection methods. All projection methods employ the same latent-state dimension of $p = 2$ and $p = 4$. These results clearly demonstrate that the projection-based methods using nonlinear embeddings yield significantly lower error than methods using the classical linear embeddings.

Moreover, Figure 1 demonstrates that the accuracy of nonlinear embedding solutions is significantly improved as the latent dimension is increased from $p = 2$ (Figure 1a) to $p = 4$ (Figure 1b). As the problem is characterized by three factors (t, μ_1, μ_2), the intrinsic solution-manifold dimension is (at most) 3. Thus, an autoencoder with the latent dimension greater than or equal to $p = 3$ will be able to reconstruct the original input data given sufficient capacity.

Now, we quantitatively assess the accuracy of of the approximated state $\tilde{\boldsymbol{x}}$ computed using all considered projection methods with the following metrics: 1) the mean-squared state-space error, $\mathcal{E}_{\boldsymbol{x}} := \sqrt{\sum_{n=1}^{N_t} \|\boldsymbol{x}^n(\boldsymbol{\mu}) - \tilde{\boldsymbol{x}}^n(\boldsymbol{\mu})\|_2^2} \Big/ \sqrt{\sum_{n=1}^{N_t} \|\boldsymbol{x}^n(\boldsymbol{\mu})\|_2^2}$, 2) the mean-squared error in the globally conserved variables, $\mathcal{E}_{\boldsymbol{x},\text{global}} := \sqrt{\sum_{n=1}^{N_t} \|\bar{\boldsymbol{C}}_{\text{global}}\boldsymbol{x}^n(\boldsymbol{\mu}) - \bar{\boldsymbol{C}}_{\text{global}}\tilde{\boldsymbol{x}}^n(\boldsymbol{\mu})\|_2^2} \Big/ \sqrt{\sum_{n=1}^{N_t} \|\bar{\boldsymbol{C}}_{\text{global}}\boldsymbol{x}^n(\boldsymbol{\mu})\|_2^2}$, and 3) the mean-squared violation in global conservation, $\mathcal{E}_{\boldsymbol{r},\text{global}} := \sqrt{\sum_{n=1}^{N_t} \|\bar{\boldsymbol{C}}_{\text{global}}\boldsymbol{r}^n(\tilde{\boldsymbol{x}}^n(\boldsymbol{\mu}); \boldsymbol{\mu})\|_2^2}$, where $\bar{\boldsymbol{C}}_{\text{global}} \in \mathbb{R}_+^{n_u \times N}$ is the operator $\bar{\boldsymbol{C}}$ associated with the global conservation $\bar{\mathcal{M}}_{\text{global}} := \{\Omega\}$. Figure 2 reports these quantities for the considered methods. These results illustrate that the best performance is obtained through the combination of a nonlinear embedding and conservation enforcement as provided by the proposed Deep Conservation method. In all error metrics, the proposed Deep Conservation outperforms the Deep LSPG projection. As numerically demonstrated in [8], minimizing the residual with the conservation constraint leads to smaller errors in states for both linear and nonlinear embeddings (Figures 2a and 2d). In particular, Deep Conservation reduces the global conservation violation $\mathcal{E}_{\boldsymbol{r},\text{global}}$ by more than an order of magnitude relative to Deep LSPG.



(a) State errors $\mathcal{E}_{\boldsymbol{x}}$ of POD–LSPG (None) and conservative LSPG ($N_{\bar{\Omega}} \geq 1$)

(b) Global state errors $\mathcal{E}_{\boldsymbol{x},\text{global}}$ of POD–LSPG (None) and conservative LSPG ($N_{\bar{\Omega}} \geq 1$)

(c) Global violation $\mathcal{E}_{\boldsymbol{r},\text{global}}$ of POD–LSPG (None) and conservative LSPG ($N_{\bar{\Omega}} \geq 1$)

(d) State errors $\mathcal{E}_{\boldsymbol{x}}$ of Deep LSPG (None) and Deep conservation ($N_{\bar{\Omega}} \geq 1$)

(e) Global state errors $\mathcal{E}_{\boldsymbol{x},\text{global}}$ of Deep LSPG (None) and Deep conservation ($N_{\bar{\Omega}} \geq 1$)

(f) Global violation $\mathcal{E}_{\boldsymbol{r},\text{global}}$ of Deep LSPG (None) and Deep conservation ($N_{\bar{\Omega}} \geq 1$)

Figure 2: Error metrics for varying latent-space dimensions p (vertical axis) and for varying numbers of subdomains $N_{\bar{\Omega}}$ (horizontal axis).

6 Conclusion

This work has proposed Deep Conservation: a novel projection-based dynamics learning technique that learns a nonlinear embedding using deep convolutional autoencoders, and computes a dynamics model via a projection process that enforces physical conservation laws. The dynamics model associates with a nonlinear least-squares problem with nonlinear equality constraints, and the method requires the availability of a finite-volume discretization of the original dynamical system, which is used to define the objective function and constraints. Numerical experiments on an advection-dominated benchmark problem demonstrated that Deep Conservation both achieves significantly higher accuracy compared with classical projection-based methods, and guarantees the time evolution of the latent state satisfies prescribed conservation laws.

Acknowledgments

This work was sponsored by Sandia's Advanced Simulation and Computing (ASC) Advanced Machine Learning (AML) Project/Task #212291/6.01. 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. 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.

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. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] E. Banijamali, R. Shu, M. Ghavamzadeh, H. Bui, and A. Ghodsi. Robust locally-linear controllable embedding. In *International Conference on Artificial Intelligence and Statistics*, 2018.

[3] P. Benner, S. Gugercin, and K. Willcox. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM Review*, 57(4):483–531, 2015.

[4] W. Böhmer, J. T. Springenberg, J. Boedecker, M. Riedmiller, and K. Obermayer. Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations. *KI-Künstliche Intelligenz*, 29(4): 353–362, 2015.

[5] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[6] 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(2):155–181, 2011.

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

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

[9] 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.

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

[11] R. Eymard, T. Gallouët, and R. Herbin. Finite volume methods. *Handbook of numerical analysis*, 7: 713–1018, 2000.

[12] 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*, pages 249–256, 2010.

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

[14] R. Goroshin, M. F. Mathieu, and Y. LeCun. Learning to linearize under uncertainty. In *Advances in Neural Information Processing Systems*, pages 1234–1242, 2015.

[15] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[16] C. Hirsch. *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics*. Elsevier, 2007.

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

[18] 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. URL `https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

[19] M. Karl, M. Soelch, J. Bayer, and P. van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. In *International Conference on Learning Representations*, 2017.

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

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[23] K. Lee and K. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *arXiv preprint arXiv:1812.08373*, 2018.

[24] T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, and D. Filliat. State representation learning for control: An overview. *Neural Networks*, 2018.

[25] R. J. LeVeque. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[26] B. Lusch, J. N. Kutz, and S. L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):4950, 2018.

[27] J. Morton, A. Jameson, M. J. Kochenderfer, and F. Witherden. Deep dynamical modeling and control of unsteady fluid flows. In *Advances in Neural Information Processing Systems*, pages 9258–9268, 2018.

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

[29] S. E. Otto and C. W. Rowley. Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems*, 18(1):558–593, 2019.

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

[31] 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.

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

[33] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pages 2746–2754, 2015.

[34] 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(5):1111–1135, 2015.

A Scaling and inverse scaling operators

Given the reshaped ith snapshot as $\mathcal{X}^i \in \mathbb{R}^{n_1 \times \cdots \times n_d \times n_{\text{chan}}}$, 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, we set the elements of the scaling operator $\boldsymbol{S}$ to

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

with

$$\begin{aligned} \mathcal{X}_j^{\max} &:= \max_{k \in \mathbb{N}(n_{\text{snap}}),\, i_1 \in \mathbb{N}(n_1),\, \ldots\, i_d \in \mathbb{N}(n_d)} \mathcal{X}^k_{i_1 \cdots i_d j} \\ \mathcal{X}_j^{\min} &:= \min_{k \in \mathbb{N}(n_{\text{snap}}),\, i_1 \in \mathbb{N}(n_1),\, \ldots\, i_d \in \mathbb{N}(n_d)} \mathcal{X}^k_{i_1 \cdots i_d j}, \end{aligned}$$

where $\mathbb{N}(n) := \{1, \ldots, n\}$. Inverse scaling operator applies an inverse action of the scaling operator.

B Derivations of Equations (13)–(15)

The velocity vector $\bar{\boldsymbol{f}}(\boldsymbol{x}, t; \boldsymbol{\mu})$ on the decomposed mesh $\bar{\mathcal{M}}$, described in Section 4.2, can be obtained by enforcing conservation (1) on each subdomain as in Eq (11) such that $\bar{\boldsymbol{f}} : (\boldsymbol{w}, \tau; \boldsymbol{\nu}) \mapsto \bar{\boldsymbol{f}}^g(\boldsymbol{w}, \tau; \boldsymbol{\nu}) + \bar{\boldsymbol{f}}^s(\boldsymbol{w}, \tau; \boldsymbol{\nu})$ with $\bar{\boldsymbol{f}}^g, \bar{\boldsymbol{f}}^s : \mathbb{R}^N \times [0, T] \times \mathcal{D} \to \mathbb{R}^{\bar{N}}$, whose elements consist of

$$\begin{aligned} \bar{f}^g_{\mathcal{I}(i,j)}(\boldsymbol{x}, t; \boldsymbol{\mu}) &= -\frac{1}{|\bar{\Omega}_j|} \int_{\bar{\Gamma}_j} \boldsymbol{g}_i^{\text{FV}}(\boldsymbol{x}; \vec{x}, t; \boldsymbol{\mu}) \cdot \boldsymbol{n}_j(\vec{x})\, \mathrm{d}\vec{s}(\vec{x}) \\ \bar{f}^s_{\mathcal{I}(i,j)}(\boldsymbol{x}, t; \boldsymbol{\mu}) &= \frac{1}{|\bar{\Omega}_j|} \int_{\bar{\Omega}_j} s_i^{\text{FV}}(\boldsymbol{x}; \vec{x}, t; \boldsymbol{\mu})\, \mathrm{d}\vec{x} \end{aligned} \tag{17}$$

for $i \in \mathbb{N}(n_u),\ j \in \mathbb{N}(N_{\bar{\Omega}})$. Using the same matrix $\bar{\boldsymbol{C}} \in \mathbb{R}_+^{\bar{N} \times N}$ in Section 4.2, $\bar{\boldsymbol{f}}^g$ and $\bar{\boldsymbol{f}}^s$ can be written in terms of $\boldsymbol{f}^g$ and $\boldsymbol{f}^s$ such that

$$\bar{\boldsymbol{f}}^g(\boldsymbol{x}, t; \boldsymbol{\mu}) = \bar{\boldsymbol{C}} \boldsymbol{f}^g(\boldsymbol{x}, t; \boldsymbol{\mu}), \quad \bar{\boldsymbol{f}}^s(\boldsymbol{x}, t; \boldsymbol{\mu}) = \bar{\boldsymbol{C}} \boldsymbol{f}^s(\boldsymbol{x}, t; \boldsymbol{\mu}),$$

and, thus,

$$\bar{\boldsymbol{f}}(\boldsymbol{x}, t; \boldsymbol{\mu}) = \bar{\boldsymbol{C}} \boldsymbol{f}(\boldsymbol{x}, t; \boldsymbol{\mu}).$$

For theoretical aspects of this decomposition, we refer readers to Ref. [8, Section 4.1].

C Solutions of a parameterized inviscid Burgers' equation

The full-order model (FOM) solution of the parameterized inviscid Burgers' equation with $\boldsymbol{\mu}^1_{\text{test}} = (4.3, 0.021)$ can be obtained by applying the finite-volume discretization (Sec 2.2) and the time-discretization method (Eqs. (6)–(7)) with the experimental setting described in Section 5. In Figure 3, the solution snapshots at time indices $t = \{3.5, 7.0, 10.5, 14, 17.5\}$ demonstrate that the location of shocks, where the discontinuities exist, move from left to right as time evolves.

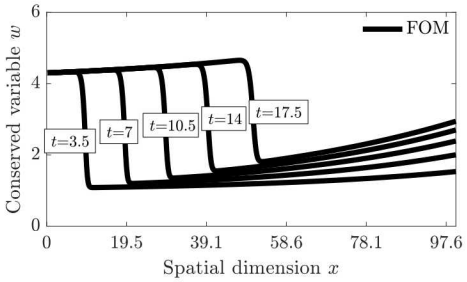


Figure 3: FOM solutions of the parameterized inviscid Burgers' equation with $\boldsymbol{\mu}^1_{\text{test}} = (4.3, 0.021)$ at time instances $t = \{3.5, 7.0, 10.5, 14, 17.5\}$.