Journal of Computational Physics 451 (2022) 110841

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder

Youngkyu Kim [a,*], Youngsoo Choi [b], David Widemann [c], Tarek Zohdi [a]

[a] Mechanical Engineering, University of California, Berkeley, CA 94720, United States of America
[b] Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94550, United States of America
[c] Computational Engineering Division, Lawrence Livermore National Laboratory, Livermore, CA 94550, United States of America

ARTICLE INFO

Article history:
Received 25 September 2020
Accepted 7 November 2021
Available online 12 November 2021

Keywords:
Nonlinear manifold solution representation
Physics-informed neural network
Reduced order model
Nonlinear dynamical system
Hyper-reduction

ABSTRACT

Traditional linear subspace reduced order models (LS-ROMs) are able to accelerate physical simulations in which the intrinsic solution space falls into a subspace with a small dimension, i.e., the solution space has a small Kolmogorov n-width. However, for physical phenomena not of this type, e.g., any advection-dominated flow phenomena such as in traffic flow, atmospheric flows, and air flow over vehicles, a low-dimensional linear subspace poorly approximates the solution. To address cases such as these, we have developed a fast and accurate physics-informed neural network ROM, namely nonlinear manifold ROM (NM-ROM), which can better approximate high-fidelity model solutions with a smaller latent space dimension than the LS-ROMs. Our method takes advantage of the existing numerical methods that are used to solve the corresponding full order models. The efficiency is achieved by developing a hyper-reduction technique in the context of the NM-ROM. Numerical results show that neural networks can learn a more efficient latent space representation on advection-dominated data from 1D and 2D Burgers' equations. A speedup of up to 2.6 for 1D Burgers' and a speedup of 11.7 for 2D Burgers' equations are achieved with an appropriate treatment of the nonlinear terms through a hyper-reduction technique. Finally, a posteriori error bounds for the NM-ROMs are derived that take account of the hyper-reduced operators.

1. Introduction

Physical simulations are influencing developments in science, engineering, and technology more rapidly than ever before. However, high-fidelity, forward physical simulations are computationally expensive and, thus, make intractable any decision-making applications, such as design optimization, inverse problems, optimal controls, and uncertainty quantification, for which many forward simulations are required to explore the parameter space in the outer loop.

To compensate for the computational expense issue, the projection-based reduced order models (ROMs) take advantage of both the known governing equation and the data. ROMs generate the solution data from the corresponding physical simulations and then compress the data to find an intrinsic solution subspace, which is represented by a linear combination of basis vectors, i.e., LS-ROMs. This condensed solution representation is plugged back into the (semi-)discretized governing equation to reduce the number of unknowns, resulting in an over-determined system, i.e., more equations than unknowns.

* Corresponding author.
E-mail addresses: youngkyu_kim@berkeley.edu (Y. Kim), choi15@llnl.gov (Y. Choi), widemann1@llnl.gov (D. Widemann), zohdi@berkeley.edu (T. Zohdi).

https://doi.org/10.1016/j.jcp.2021.110841
0021-9991/

Note that the full governing equations are used to constrain the LS-ROM through this substitution. Therefore, this can be considered as a physics-informed surrogate model. Additionally, the existing numerical methods for the corresponding full order model (FOM) are utilized in the LS-ROM solution process. Therefore, the LS-ROM fully respects the original discretization of the governing equations that describe/approximate the underlying physical laws, unlike black-box approaches.

The LS-ROM approach has been successfully applied to many problems and applications, including, but not limited to, rocket nozzle shape design [2], flutter avoidance wing shape optimization [16], topology optimization of wind turbine blades [20], porous media flow/reservoir simulations [29,35,74], computational electro-cardiology [73], inverse problems [28], shallow water equations [76,66], computing electromyography [51], spatio-temporal dynamics of predator—prey systems [23], and acoustic wave-driven microfluidic biochips [3]. A survey paper for the projection-based LS-ROM techniques can be found in [5].

In spite of its successes, the linear subspace solution representation suffers from not being able to represent certain physical simulation solutions with a small basis dimension, such as advection-dominated or sharp gradient solutions. This is because LS-ROMs work only for physical problems in which the intrinsic solution space falls into a subspace with a small dimension, i.e., the solution space has a small Kolmogorov n-width. Unfortunately, even though problems that are advection-dominated or have sharp gradient solutions are important, they do not have small Kolmogorov n-width. Such physical simulations include, but are not limited to, the hyperbolic equations with high Reynolds number, the Boltzmann transport equations, and the traffic flow simulations.

Therefore, there have been many attempts to build efficient ROMs for the advection-dominated or sharp gradient problems. The attempts can be divided mainly into two categories: the first one is to enhance the solution representability of the linear subspace by introducing some special treatments and adaptive schemes and the second one is to replace the linear subspace solution representation with the nonlinear manifold.

The effort of enhancing the solution representability of the linear subspace includes the artificial viscosity, the Petrov–Galerkin projection applied to the computational fluid dynamics problems [13,12,19], the residual discrete empirical interpolation approach to handle the Navier—Stokes equations with a large Reynolds number [72], and the space–time ROM [17,18,67] where the temporal as well as spatial dimensions were reduced to maximize the compressibility even with the advection-dominated problems. A dictionary-based model reduction method was developed in [1] where ℓ_1 minimization is used to project onto the reduced linear subspace. A fail-safe h-adaptive algorithm was developed in [9] where the reduced linear subspace basis vectors are broken algebraically to enrich the solution subspace. The shifted proper orthogonal decomposition (POD) was introduced to address the issue that arises from the advection-dominated problems [60] where a transport operator is incorporated within the POD process. The drawback with this approach is that the speed of the transport operator must be known a priori. In a similar spirit, the transport reversal was introduced in [61], which was inspired by the template fitting [38]. The windowed least-squares Petrov–Galerkin model reduction for dynamical systems with implicit time integrators is introduced in [53], which can overcome the challenges arising from the advection-dominated problems by representing only a small time window with a local ROM. In order to capture the sharp gradient accurately, many approaches use localization strategies. The examples of such methods include the online adaptive bases and sampling approach in [56] and [21]. Transformed snapshot interpolation method was also developed in [70] to capture a sharp gradient in the solution, by introducing a new transform discretization near singularities.

Even though all the approaches mentioned above do show some remedies of overcoming the challenges that arise from the advection-dominated problems, the solution representability of the linear subspace is still limited in a sense that the treatments introduced in the methods above are problem-specific and require some a priori knowledge, such as advection direction. In order to maximize the representability and make the methodology as general as possible, it seems unavoidable to transition from the linear subspace to a nonlinear manifold solution representation.

There are many works available in the current literature that looked into the nonlinear manifold solution representation in physical simulations. Many of them treat the weights and biases of a neural network (NN) to be unknowns in the solution process. For example, Lagaris, et al., used a single output NN as an argument for trial functions and minimized the partial/ordinary differential equation (PDE/ODE) residual norm [41], where the weights of the NNs are used as optimization variables. Dissanayake and Phan-Thien used the universal approximator of NNs as a solution representation for solving PDEs. They also used the weights of the NNs as parameters as in the work by Lagaris, et al. [24]. A similar method was also applied to a plasma equilibrium solver [68]. Meade and Fernandez used hard limit transfer functions for linear ordinary differential equations [50]. However, these approaches can introduce too many unknowns because all the weights and biases need to be found during the PDE/ODE solution process.

Recently, similar attempts have been made to incorporate physical laws into NN-based surrogate models —- so called physics-informed surrogate models, where the weights and biases of the NN are determined in the training phase. Such models include, but are not limited to, attempts to mimic temporal evolution by incorporating a time integrator in a loss function [59,15,36,47,4] and to represent the solution with a trained NN [59,77,6], the deep Galerkin method [65], approximating spatial gradient functions with a multilayer feedforward NN [30], DeepONet [48], DeepXDE [49], fractional physics-informed NNs (fPINNs) [52], PINNs with uncertainty quantification [75], and Deep Ritz method of minimizing the energy functional with trial functions of NNs [69,31]. However, inclusions of NNs in the governing equations of the underlying physical laws, such as those above, do not take advantage of the existing numerical methods for high-fidelity physical simulations.

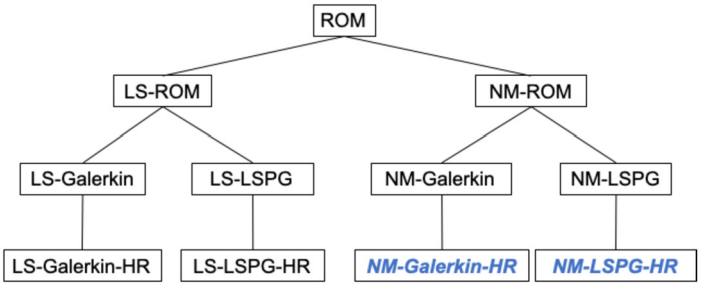


Fig. 1. The figure shows the hierarchy of several ROMs. If the governing equation is nonlinear, then a hyper-reduction is required to achieve both accuracy and speed-up with respect to the corresponding FOM. This paper contributes to the development of NM-LSPG-HR and NM-Galerkin-HR that achieve both speedup and accuracy with the NM-ROM. Throughout the paper, we will compare the performance of LS-ROMs and NM-ROMs.

Recently, a neural network-based ROM is developed in [43], where the weights and biases are determined in the training phase and the existing numerical methods are utilized in their models. The same technique is extended to preserve the conserved quantities in the physical conservation laws [42]. However, their approaches do not achieve any speed-up with respect to the corresponding FOM because the nonlinear terms that still scale with the FOM size need to be updated every time step or Newton step.

Two interesting papers were written by Rim, et al., recently. First of all, manifold approximations via transported subspaces in [62] introduced a nonlinear solution representation by explicitly composing global transport dynamics with locally linear approximations of the solution manifolds. However, their approach is only applicable to 1D problem for now. The other work by Rim, et al., is the depth separation for reduced deep networks in nonlinear model reduction [63], where they applied a compression technique on weight matrices and bias vectors to achieve the reduced deep networks.

We present a fast and accurate physics-informed neural network ROM with a nonlinear manifold solution representation, i.e., the nonlinear manifold ROM (NM-ROM). We train a shallow masked autoencoder with solution data from the corresponding FOM simulations and use the decoder as the nonlinear manifold solution representation. Our NM-ROM is different from the aforementioned physics-informed neural networks in that we take advantage of the existing numerical methods of solving PDE/ODEs in our approach. Furthermore, our NM-ROM is different from the neural network-based ROM of [43] in a sense that we use a shallow masked autoencoder, while they used a deep convolutional autoencoder. The choice of the shallow masked NN over the deep convolutional NN is determined by the efficiency of the hyper-reduction technique we have developed.

1.1. Nomenclature

We use the following nomenclature/abbreviation for various ROMs throughout the paper:

- FOM: full order model
- LS-ROM: linear subspace reduced order model
- LS-Galerkin: linear subspace Galerkin
- LS-LSPG: linear subspace least-squares Petrov–Galerkin
- LS-Galerkin-HR: linear subspace Galerkin hyper-reduction
- LS-LSPG-HR: linear subspace least-squares Petrov–Galerkin hyper-reduction
- NM-ROM: nonlinear manifold reduced order model
- NM-Galerkin: nonlinear manifold Galerkin
- NM-LSPG: nonlinear manifold least-squares Petrov–Galerkin
- NM-Galerkin-HR: nonlinear manifold Galerkin hyper-reduction
- NM-LSPG-HR: nonlinear manifold least-squares Petrov–Galerkin hyper-reduction

These ROMs form a hierarchy that is depicted in Fig. 1.

1.2. Organization of the paper

We organize the subsequent sections by starting to discuss some background materials in Section 2, where the FOM is stated in Section 2.1 and two LS-ROMs, i.e., LS-Galerkin and LS-LSPG, are described in Sections 2.2.2 and 2.2.3, respectively. Our NM-ROM is introduced in Section 3, where the nonlinear manifold solution representation is explained in Section 3.1. The shallow masked autoencoder that is used for the solution representation is described in Section 3.2. The NM-Galerkin is explained in Section 3.3 and the NM-LSPG is described in Section 3.4. The hyper-reduction technique that enables the NM-ROM to achieve a speed-up is elaborated in Section 4. The error analysis is presented in Section 5. Finally, the performance of our NM-ROM is demonstrated in two numerical experiments in Section 6. Finally, the paper is concluded with summary and discussion in Section 7.

2. Background

2.1. Full order model

A parameterized nonlinear dynamical system is considered, characterized by a system of nonlinear ordinary differential equations (ODEs), which can be considered as a resultant system from semi-discretization of Partial Differential Equations (PDEs) in space domains

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t; \boldsymbol{\mu}), \qquad \boldsymbol{x}(0; \boldsymbol{\mu}) = \boldsymbol{x}_0(\boldsymbol{\mu}), \tag{2.1}$$

where $t \in [0, T]$ denotes time with the final time $T \in \mathbb{R}_+$, and $\boldsymbol{x}(t; \boldsymbol{\mu})$ denotes the time-dependent, parameterized state implicitly defined as the solution to problem (2.1) with $\boldsymbol{x} : [0, T] \times \mathcal{D} \to \mathbb{R}^{N_s}$. Further, $\boldsymbol{f} : \mathbb{R}^{N_s} \times [0, T] \times \mathcal{D} \to \mathbb{R}^{N_s}$ with $(\boldsymbol{w}, \tau; \boldsymbol{v}) \mapsto \boldsymbol{f}(\boldsymbol{w}, \tau; \boldsymbol{v})$ denotes the velocity of $\boldsymbol{x}$, which we assume to be nonlinear in at least its first argument. The initial state is denoted by $\boldsymbol{x}_0 : \mathcal{D} \to \mathbb{R}^{N_s}$, and $\boldsymbol{\mu} \in \mathcal{D}$ denotes parameters in the domain $\mathcal{D} \subseteq \mathbb{R}^{n_\mu}$.

A uniform time discretization is assumed throughout the paper, characterized by time step $\Delta t \in \mathbb{R}_+$ and time instances $t^n = t^{n-1} + \Delta t$ for $n \in \mathbb{N}(N_t)$ with $t^0 = 0$, $N_t \in \mathbb{N}$, and $\mathbb{N}(N) := \{1, \ldots, N\}$. To avoid notational clutter, we introduce the following time discretization-related notations: $\boldsymbol{x}_n := \boldsymbol{x}(t^n; \boldsymbol{\mu})$, $\tilde{\boldsymbol{x}}_n := \tilde{\boldsymbol{x}}(t^n; \boldsymbol{\mu})$, $\hat{\boldsymbol{x}}_n := \hat{\boldsymbol{x}}(t^n; \boldsymbol{\mu})$, and $\boldsymbol{f}_n := \boldsymbol{f}(\boldsymbol{x}(t^n; \boldsymbol{\mu}), t^n; \boldsymbol{\mu})$, where $\tilde{\boldsymbol{x}}$, $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{x}}$ are defined in Section 2.2.

Implicit time integrators are considered as time discretization methods. To illustrate this, we mainly consider the backward Euler time integrator for an implicit scheme. Several other time integrators are shown in Appendix A.

The implicit Backward Euler (BE) method numerically solves Eq. (2.1), by solving the following nonlinear system of equations for $\boldsymbol{x}_n$ at n-th time step:

$$\boldsymbol{x}_n - \boldsymbol{x}_{n-1} = \Delta t \boldsymbol{f}_n. \tag{2.2}$$

Eq. (2.2) implies the following subspace inclusion:

$$\text{span}\{\boldsymbol{f}_n\} \subseteq \text{span}\{\boldsymbol{x}_{n-1}, \boldsymbol{x}_n\}.$$

By induction, we conclude the following subspace inclusion relation:

$$\text{span}\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_{N_t}\} \subseteq \text{span}\{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{N_t}\},$$

which shows that the span of nonlinear term snapshots is included in the span of solution snapshots. The residual function with the backward Euler time integrator is defined as

$$\boldsymbol{r}^n_{\text{BE}}(\boldsymbol{x}_n; \boldsymbol{x}_{n-1}, \boldsymbol{\mu}) := \boldsymbol{x}_n - \boldsymbol{x}_{n-1} - \Delta t \boldsymbol{f}_n. \tag{2.3}$$

2.2. Linear subspace reduced order model (LS-ROM)

Many projection-based reduced order models with linear subspace solution representation can be considered for nonlinear dynamical systems. We consider Galerkin and least-squares Petrov-Galerkin projection methods, which are the most relevant to our proposed method, i.e., NM-ROM.

2.2.1. Linear subspace solution representation

The linear subspace reduced order model approach applies spatial projection using a subspace $\mathcal{S} := \text{span}\{\boldsymbol{\phi}_i\}_{i=1}^{n_s} \subseteq \mathbb{R}^{N_s}$ with $\dim(\mathcal{S}) = n_s \ll N_s$. Using this subspace, it approximates the solution as $\boldsymbol{x} \approx \tilde{\boldsymbol{x}} \in \boldsymbol{x}_{ref} + \mathcal{S}$ (i.e., in a trial subspace) or equivalently

$$\boldsymbol{x} \approx \tilde{\boldsymbol{x}} = \boldsymbol{x}_{ref} + \boldsymbol{\Phi}\hat{\boldsymbol{x}} \tag{2.4}$$

and the time derivative of the solution as

$$\frac{d\boldsymbol{x}}{dt} \approx \frac{d\tilde{\boldsymbol{x}}}{dt} = \boldsymbol{\Phi}\frac{d\hat{\boldsymbol{x}}}{dt} \tag{2.5}$$

where $\boldsymbol{x}_{ref} \in \mathbb{R}^{N_s}$ denotes a reference solution and $\boldsymbol{\Phi} := [\boldsymbol{\phi}_1 \cdots \boldsymbol{\phi}_{n_s}] \in \mathbb{R}^{N_s \times n_s}$ denotes a basis matrix and $\hat{\boldsymbol{x}} \in \mathbb{R}^{n_s}$ denotes the generalized coordinates. The initial condition for the generalized coordinate, $\hat{\boldsymbol{x}}_0 \in \mathbb{R}^{n_s}$, is given by $\hat{\boldsymbol{x}}_0 = \boldsymbol{\Phi}^T\left(\boldsymbol{x}_0 - \boldsymbol{x}_{ref}\right)$.

For constructing $\boldsymbol{\Phi}$, Proper Orthogonal Decomposition (POD) is commonly used. POD [7] obtains $\boldsymbol{\Phi}$ from a truncated Singular Value Decomposition (SVD) approximation to a FOM solution snapshot matrix. It is related to principal component analysis in statistical analysis [34] and Karhunen–Loève expansion [46] in stochastic analysis. POD forms a solution snapshot matrix, $\boldsymbol{X} := \left[\boldsymbol{x}_0^{\boldsymbol{\mu}_1} - \boldsymbol{x}_{ref} \quad \cdots \quad \boldsymbol{x}_{N_t}^{\boldsymbol{\mu}_{n_\mu}} - \boldsymbol{x}_{ref}\right] \in \mathbb{R}^{N_s \times n_\mu(N_t+1)}$, where $\boldsymbol{x}_n^{\boldsymbol{\mu}_k}$ is a solution state at n-th time step with parameter $\boldsymbol{\mu}_k$ for $n \in \mathbb{N}(N_t)$ and $k \in \mathbb{N}(n_\mu)$. Then, POD computes its thin SVD:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T,$$

where $\boldsymbol{U} \in \mathbb{R}^{N_s \times n_\mu(N_t+1)}$ and $\boldsymbol{V} \in \mathbb{R}^{n_\mu(N_t+1)\times n_\mu(N_t+1)}$ are orthogonal matrices and $\boldsymbol{\Sigma} \in \mathbb{R}^{n_\mu(N_t+1)\times n_\mu(N_t+1)}$ is a diagonal matrix with singular values on its diagonals. Then POD chooses the leading n_s columns of $\boldsymbol{U}$ to set $\boldsymbol{\Phi}$ (i.e., $\boldsymbol{\Phi} = \begin{bmatrix}\boldsymbol{u}_1 & \cdots & \boldsymbol{u}_{n_s}\end{bmatrix}$, where $\boldsymbol{u}_k$ is k-th column vector of $\boldsymbol{U}$). The POD basis minimizes $\|\boldsymbol{X} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{X}\|_F^2$ over all $\boldsymbol{\Phi} \in \mathbb{R}^{N_s\times n_s}$ with orthonormal columns, where $\|\boldsymbol{A}\|_F$ denotes the Frobenius norm of a matrix $\boldsymbol{A} \in \mathbb{R}^{I\times J}$, defined as $\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}^2}$ with a_{ij} being an (i, j)-th element of $\boldsymbol{A}$. Since the objective function does not change if $\boldsymbol{\Phi}$ is post-multiplied by an arbitrary $n_s \times n_s$ orthogonal matrix, the POD procedure seeks the optimal n_s—dimensional subspace that captures the snapshots in the least-squares sense. For more details on POD, we refer to [33,40].

2.2.2. Linear subspace Galerkin projection

We derive LS-Galerkin using time continuous residual minimization. First, we rewrite FOM ODE Eq. (2.1) as

$$\boldsymbol{r}(\dot{\boldsymbol{x}}, \boldsymbol{x}, t; \boldsymbol{\mu}) := \dot{\boldsymbol{x}} - \boldsymbol{f}(\boldsymbol{x}, t; \boldsymbol{\mu}) = 0, \quad \boldsymbol{x}(0; \boldsymbol{\mu}) = \boldsymbol{x}_0(\boldsymbol{\mu}) \tag{2.6}$$

where $\boldsymbol{r} : \mathbb{R}^{N_s} \times \mathbb{R}^{N_s} \times \mathbb{R}_+ \times \mathcal{D} \to \mathbb{R}^{N_s}$ with $(\dot{\boldsymbol{w}}, \boldsymbol{w}, \tau; \boldsymbol{\nu}) \mapsto \boldsymbol{r}(\dot{\boldsymbol{w}}, \boldsymbol{w}, \tau; \boldsymbol{\nu})$ denotes the time continuous residual. Here, we denote $\dot{(\cdot)}$ as time derivative of $(\cdot)$ for notational simplicity. Replacing $\boldsymbol{x}$ with $\tilde{\boldsymbol{x}}$ given by Eq. (2.4) and $\dot{\boldsymbol{x}}$ with $\dot{\tilde{\boldsymbol{x}}}$ given by Eq. (2.5) leads to the following residual function with the reduced number of unknowns

$$\tilde{\boldsymbol{r}}(\dot{\hat{\boldsymbol{x}}}, \hat{\boldsymbol{x}}, t; \boldsymbol{\mu}) := \boldsymbol{r}(\boldsymbol{\Phi}\dot{\hat{\boldsymbol{x}}}, \boldsymbol{x}_{ref} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}, t; \boldsymbol{\mu}),$$

where $\tilde{\boldsymbol{r}} : \mathbb{R}^{n_s} \times \mathbb{R}^{n_s} \times \mathbb{R}_+ \times \mathcal{D} \to \mathbb{R}^{N_s}$ with $(\dot{\hat{\boldsymbol{w}}}, \hat{\boldsymbol{w}}, \tau; \boldsymbol{\nu}) \mapsto \tilde{\boldsymbol{r}}(\dot{\hat{\boldsymbol{w}}}, \hat{\boldsymbol{w}}, \tau; \boldsymbol{\nu})$ denotes the time continuous residual. Note that $\tilde{\boldsymbol{r}}(\dot{\hat{\boldsymbol{x}}}, \hat{\boldsymbol{x}}, t; \boldsymbol{\mu}) = \boldsymbol{0}$ is an over-determined system. Therefore, it is likely that no solution exists. To close the system, we minimize the squared norm of the residual vector function:

$$\dot{\hat{\boldsymbol{x}}} = \underset{\hat{\boldsymbol{v}}\in\mathbb{R}^{n_s}}{\operatorname{argmin}} \|\tilde{\boldsymbol{r}}(\hat{\boldsymbol{v}}, \hat{\boldsymbol{x}}, t; \boldsymbol{\mu})\|_2^2 \tag{2.7}$$

with $\hat{\boldsymbol{x}}(0; \boldsymbol{\mu}) = \hat{\boldsymbol{x}}_0(\boldsymbol{\mu}) = \boldsymbol{\Phi}^T\left(\boldsymbol{x}_0(\boldsymbol{\mu}) - \boldsymbol{x}_{ref}\right)$. The solution to Eq. (2.7) leads to the LS-Galerkin

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{\Phi}^T\boldsymbol{f}(\boldsymbol{x}_{ref} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}, t; \boldsymbol{\mu}), \quad \hat{\boldsymbol{x}}(0; \boldsymbol{\mu}) = \hat{\boldsymbol{x}}_0(\boldsymbol{\mu}). \tag{2.8}$$

Applying a time integrator to Eq. (2.8) leads to a fully discretized reduced system, denoted as the reduced OΔE. Note that the reduced OΔE has n_s unknowns and n_s equations. If an implicit time integrator is applied, a Newton–type method can be applied to solve for unknown generalized coordinates each time step. If an explicit time integrator is applied, time marching updates will solve the system. However, we cannot expect any speed-up because the size of the nonlinear term and its Jacobian, which need to be updated for every Newton step, scales with the FOM size. In order to handle this issue, the hyper-reduction will be applied (see Section 4.2.1).

2.2.3. Linear subspace least-squares Petrov–Galerkin projection

The Least-Squares Petrov–Galerkin (LSPG) method projects a fully discretized solution space onto a trial subspace. That is, it discretizes Eq. (2.1) in time domain and replaces $\boldsymbol{x}_n$ with $\tilde{\boldsymbol{x}}_n := \boldsymbol{x}_{ref} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}_n$ for $n \in \mathbb{N}(N_t)$ in residual functions defined in Section 2.1 and Appendix A. Here, we consider only implicit time integrators because the LSPG projection is equivalent to the Galerkin projection when an explicit time integrator is used as shown in Section 5.1 in [10]. The residual functions for implicit time integrators are defined in (2.3), (A.1), and (A.2) for various time integrators. For example, the residual function with the backward Euler time integrator[1] after the trial subspace projection becomes

$$\begin{aligned}\tilde{\boldsymbol{r}}^n_{\mathrm{BE}}(\hat{\boldsymbol{x}}_n; \hat{\boldsymbol{x}}_{n-1}, \boldsymbol{\mu}) &:= \boldsymbol{r}^n_{\mathrm{BE}}(\boldsymbol{x}_{ref} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}_n; \boldsymbol{x}_{ref} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}_{n-1}, \boldsymbol{\mu}) \\ &= \boldsymbol{\Phi}(\hat{\boldsymbol{x}}_n - \hat{\boldsymbol{x}}_{n-1}) - \Delta t\,\boldsymbol{f}(\boldsymbol{x}_{ref} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}_n, t_n; \boldsymbol{\mu}).\end{aligned} \tag{2.9}$$

The basis matrix $\boldsymbol{\Phi}$ can be found by the POD as in the Galerkin approach. Note that Eq. (2.9) is an over-determined system. To close the system and solve for the unknown generalized coordinates, $\hat{\boldsymbol{x}}_n$, the LSPG takes the squared norm of the residual vector function and minimize it at every time step:

$$\hat{\boldsymbol{x}}_n = \underset{\hat{\boldsymbol{v}}\in\mathbb{R}^{n_s}}{\operatorname{argmin}} \quad \frac{1}{2}\left\|\tilde{\boldsymbol{r}}^n_{\mathrm{BE}}(\hat{\boldsymbol{v}}; \hat{\boldsymbol{x}}_{n-1}, \boldsymbol{\mu})\right\|_2^2. \tag{2.10}$$

The Gauss–Newton method with the starting point $\hat{\boldsymbol{x}}_{n-1}$ is applied to solve the minimization problem (2.10) in LSPG. However, as in the Galerkin approach, a hyper-reduction, which will be discussed in Section 4.2.2, is required for a speed-up due to the presence of the nonlinear residual vector function that scales with the full order model size.

[1] Although the backward Euler time integrator is used extensively in the paper for illustrative purposes, many other time integrators introduced in Appendix A can be applied to all the ROM methods dealt in the paper in a straightforward way.

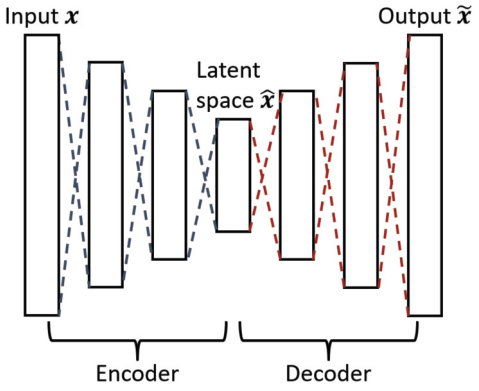


Fig. 2. General description of an autoencoder: $\boldsymbol{x}$ being encoded to a latent vector, $\hat{\boldsymbol{x}}$, by the encoder and decoded by the decoder, to $\tilde{\boldsymbol{x}}$. The mean square error between $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$ is minimized to update neural network weights and bias.

3. Nonlinear manifold reduced order model (NM-ROM)

A projection-based reduced order model with nonlinear manifold solution representation is introduced in this section. The ROM formulation with nonlinear manifold solution representation is introduced in Section 3.1. Section 3.2 describes how we construct the neural network that is used as a nonlinear manifold solution representation. As in the LS-ROMs of Section 2.2, Galerkin and least-squares Petrov–Galerkin projections will be applied in Sections 3.3 and 3.4. Finally, the hyper-reduction for the NM-ROM is described in Section 4.

3.1. Nonlinear manifold solution representation

The NM-ROM applies solution representation using a nonlinear manifold $\mathcal{S} := \{\boldsymbol{g}(\hat{\boldsymbol{v}}) \,|\, \hat{\boldsymbol{v}} \in \mathbb{R}^{n_s}\}$, where $\boldsymbol{g} : \mathbb{R}^{n_s} \to \mathbb{R}^{N_s}$ with $n_s \ll N_s$ denotes a nonlinear function that maps a latent space of dimension n_s to the full order model space of dimension, N_s. That is, the NM-ROM approximates the solution in a trial manifold as

$$\boldsymbol{x} \approx \tilde{\boldsymbol{x}} = \boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}) \tag{3.1}$$

and the time derivative of the solution as

$$\frac{d\boldsymbol{x}}{dt} \approx \frac{d\tilde{\boldsymbol{x}}}{dt} = \boldsymbol{J}_g(\hat{\boldsymbol{x}}) \frac{d\hat{\boldsymbol{x}}}{dt} \tag{3.2}$$

where $\hat{\boldsymbol{x}} \in \mathbb{R}^{n_s}$ denotes the generalized coordinates. The initial condition for the generalized coordinate, $\hat{\boldsymbol{x}}_0 \in \mathbb{R}^{n_s}$, is given by $\hat{\boldsymbol{x}}_0 = \boldsymbol{h}(\boldsymbol{x}_0 - \boldsymbol{x}_{ref})$, where $\boldsymbol{h} \approx \boldsymbol{g}^{-1}$ (i.e., $\boldsymbol{x} - \boldsymbol{x}_{ref} \approx \boldsymbol{g}(\boldsymbol{h}(\boldsymbol{x} - \boldsymbol{x}_{ref}))$). The details about the nonlinear functions, $\boldsymbol{h}$ and $\boldsymbol{g}$, are presented in Section 3.2.

3.2. Shallow masked autoencoder

In this section, we present the approach for constructing a nonlinear manifold. Here, we use an autoencoder, $\boldsymbol{A}$, in the form of a feedforward neural network, that is trained to reconstruct its input. The autoencoder architecture is composed of an encoder, $\boldsymbol{E}$ and a decoder, $\boldsymbol{D}$. The encoder maps a high dimensional input, $\boldsymbol{x} \in \mathbb{R}^{N_s}$ to a low-dimensional latent vector, $\hat{\boldsymbol{x}} \in \mathbb{R}^{n_s}$, i.e., $\boldsymbol{E}(\boldsymbol{x}) = \hat{\boldsymbol{x}}$, and the decoder then maps the latent vector to $\tilde{\boldsymbol{x}} \in \mathbb{R}^{N_s}$, i.e., $\boldsymbol{D}(\hat{\boldsymbol{x}}) = \tilde{\boldsymbol{x}}$, where $n_s \ll N_s$ (see Fig. 2). Therefore, we have

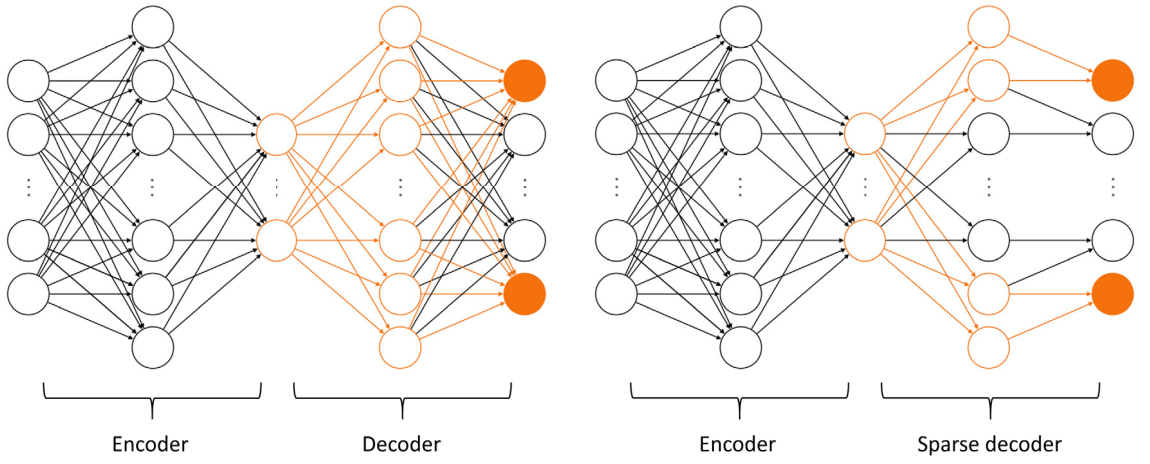
$$\boldsymbol{x} \approx \tilde{\boldsymbol{x}} = \boldsymbol{A}(\boldsymbol{x}) = \boldsymbol{D}(\boldsymbol{E}(\boldsymbol{x})).$$

The main idea behind an autoencoder is that it forces the model to learn salient features by compressing the input into a low-dimensional space and then reconstructing the input.

The universal approximation theorem [22,57], proves that functions of the form,

$$v_k = \sum_{j=1}^{N_2} w_{jk2}\sigma\left(\sum_{i=1}^{N_1} w_{ij1}u_i + \theta_j\right) \quad \text{for } k \in \mathbb{N}(N_3), \tag{3.3}$$

where $w_{ij1}, w_{jk2} \in \mathbb{R}$ are weights, $\theta_j \in \mathbb{R}$ is a bias, σ is a non-polynomial activation function, u_i is an input and v_k is an output, can approximate any continuous, real-valued function arbitrarily well. Eq. (3.3) is a simple, single hidden layer neural network with a non-polynomial activation function. Its input dimension is N_1, width of the hidden layer is N_2, and



(a) Without masking (b) With masking

Fig. 3. Three layer autoencoder architecture: (a) unmasked and (b) masked shallow neural network. Nodes and edges in orange color represent active path that stems from the sampled outputs that are marked as the orange disks. Note that the masked shallow neural network has a sparser structure than the unmasked one.

output dimension is N_3. We construct two single hidden layer neural networks, one is the encoder, $\boldsymbol{E}$, and the other is the decoder, $\boldsymbol{D}$. For non-polynomial activation functions, a sigmoidal function given by

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

or a swish function given by

$$\sigma(x) = \frac{x}{1+\exp(-x)}$$

are used. We use a non-deep neural network for the decoder because the decoder and its Jacobian are computed many times during the ROM computation. In order for this computation to be on par with POD methods, it is necessary to limit the depth of the decoder network. The dimension of the encoder input and the decoder output is N_s and the dimension of the encoder $\boldsymbol{E}$ output and the decoder $\boldsymbol{D}$ input is n_s. The width of the encoder $\boldsymbol{E}$ and decoder $\boldsymbol{D}$ are hyper-parameters. The first layers of the encoder $\boldsymbol{E}$ and decoder $\boldsymbol{D}$ are fully-connected layers, where the nonlinear activation functions are applied and the last layer of the encoder $\boldsymbol{E}$ is fully-connected layer with no activation functions. The last layer of the decoder $\boldsymbol{D}$ is either fully-connected layer or sparsely-connected layer with no activation functions. These network architectures are shown in Fig. 3.

Then, combining the encoder and the decoder yields the autoencoder which can be trained to learn the identity mapping in an unsupervised manner because the desired output is the input. During the training phase, the error measured by

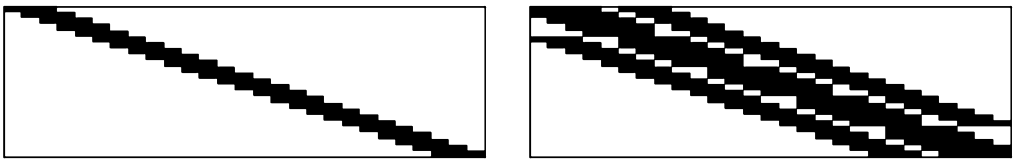
$$\|\boldsymbol{X} - \tilde{\boldsymbol{X}}\|_F^2,$$

where $\boldsymbol{X}$ is solution snapshot matrix and $\tilde{\boldsymbol{X}}$ is a reconstructed solution snapshot matrix, is minimized by optimizing learnable parameters (i.e., weights and bias) in the two networks. The error is back-propagated through the networks and the gradient with respect to the learnable parameters are computed by using the chain rule [64,54,71]. Then, the parameters are updated in the steepest descent direction with respect to the gradient. Here, ADAM [37], a variant of stochastic gradient descent (SGD), is used to approximate the gradient with a few data samples to make training process faster. Stochastic gradient noise helps the neural network avoiding over-fitting [8]. Furthermore, graphics processing units (GPUs) are utilized to parallelize the autonencoder's training by simultaneously approximating multiple snapshots [58]. In practice, a dataset is usually normalized before the training process. Here, we normalize the dataset (i.e., solution snapshots) in the following way:

$$\boldsymbol{x}_{normal} = \boldsymbol{x}_{scale} \odot \left(\boldsymbol{x} - \boldsymbol{x}_{ref}\right)$$

where $\boldsymbol{x}$ is a column vector of the dataset matrix $\boldsymbol{X}$ and $\odot$ denotes the element-wise product. $\boldsymbol{x}_{scale}$ and $\boldsymbol{x}_{ref}$ are directly computed from the dataset along each feature direction such that $\boldsymbol{x}_{normal}$ ranges either $[-1, 1]$ or $[0, 1]$.

After data normalization, an autoencoder can be trained to learn the identity mapping with the normalized dataset. Now, a normalized encoder maps from a high dimensional normalized input $\boldsymbol{x}_{normal} \in \mathbb{R}^{N_s}$ to a low dimensional latent vector $\hat{\boldsymbol{x}} \in \mathbb{R}^{n_s}$ in the form:



(a) Mask matrix for 1D Burgers equation (b) Mask matrix for 2D Burgers equation

Fig. 4. Mask matrix. Note that the mask matrices have the analogical structure to the ones of Mass matrix that arises from a numerical discretization, such as the finite element or difference method, with 1D or 2D diffusion equations.

$$\hat{\boldsymbol{x}} = \mathbf{en}\left(\boldsymbol{x}_{normal}\right)$$

and a normalized decoder maps from the low dimensional latent vector $\hat{\boldsymbol{x}} \in \mathbb{R}^{n_s}$ to a reconstructed normalized input $\tilde{\boldsymbol{x}}_{normal} \in \mathbb{R}^{N_s}$ in the form:

$$\tilde{\boldsymbol{x}}_{normal} = \mathbf{de}\left(\hat{\boldsymbol{x}}\right).$$

Next, the encoder $\boldsymbol{E}$ and the decoder $\boldsymbol{D}$ can be written by

$$\boldsymbol{E}\left(\boldsymbol{x}\right) = \mathbf{en}\left(\boldsymbol{x}_{scale} \odot \left(\boldsymbol{x} - \boldsymbol{x}_{ref}\right)\right)$$
$$\boldsymbol{D}\left(\hat{\boldsymbol{x}}\right) = \boldsymbol{x}_{ref} + \mathbf{de}\left(\hat{\boldsymbol{x}}\right) \oslash \boldsymbol{x}_{scale}$$

where $\odot$ and $\oslash$ denote the element-wise product and division, respectively. Moreover, the row-wise product of $\boldsymbol{x}_{scale}$ and the first layer weight matrix of **en** yields the scaled encoder $\boldsymbol{h}$. Likewise, the row-wise division of $\boldsymbol{x}_{scale}$ and the last layer weight matrix of **de** gives us the scaled decoder $\boldsymbol{g}$. Finally, the encoder $\boldsymbol{E}$ and the decoder $\boldsymbol{D}$ are given by

$$\boldsymbol{E}\left(\boldsymbol{x}\right) = \boldsymbol{h}\left(\boldsymbol{x} - \boldsymbol{x}_{ref}\right)$$
$$\boldsymbol{D}\left(\hat{\boldsymbol{x}}\right) = \boldsymbol{x}_{ref} + \boldsymbol{g}\left(\hat{\boldsymbol{x}}\right).$$

We set the decoder $\boldsymbol{D}\left(\hat{\boldsymbol{x}}\right) = \boldsymbol{x}_{ref} + \boldsymbol{g}\left(\hat{\boldsymbol{x}}\right)$ as the nonlinear manifold solution representation discussed in Section 3.

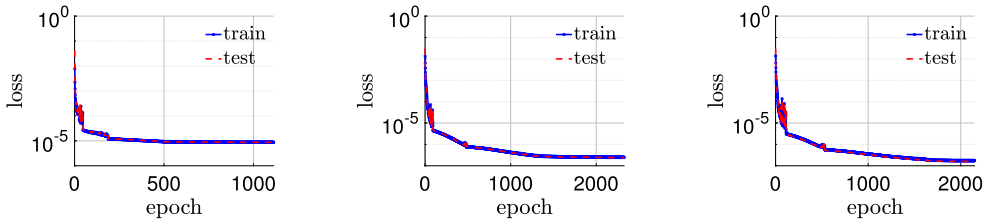
The scaled decoder $\boldsymbol{g}$ can be written in the form

$$\boldsymbol{g}\left(\hat{\boldsymbol{x}}\right) = \boldsymbol{W}_2\boldsymbol{\sigma}\left(\boldsymbol{W}_1\hat{\boldsymbol{x}} + \boldsymbol{b}_1\right)$$

where $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are weight matrices, $\boldsymbol{b}_1$ is a bias vector, and $\boldsymbol{\sigma}$ is an element-wise activation function. The decoder can have more than two hidden layers (i.e., deep network). However, we use the single layer decoder (i.e., shallow network) because the Jacobian computation of the multiple hidden layer decoder involves multiple matrix−matrix multiplications. The output layer of the decoder $\boldsymbol{g}$ is fully-connected as depicted in Fig. 3 (a) (i.e., $\boldsymbol{W}_2$ is a dense matrix), which means all nodes in the previous layer are required to compute even one element of the output vector. We apply a sparsity mask on the output layer of the decoder. Then, sampling a subset of the output vector doesn't need all nodes in the previous layer as depicted in Fig. 3 (b). Thus, more speed-up can be achieved by a hyper-reduction technique that is described in Section 4. For example, the orange color nodes in Fig. 3 show the required nodes when the first and the last elements of the output are selected, which are represented as solid orange disks. To create a sparsely connected layer, we use a mask matrix $\boldsymbol{S}$ which contains either zero or one as shown in Fig. 4. By element-wise product $\boldsymbol{S} \odot \boldsymbol{W}_2$, a sparse weight matrix is obtained. The mask matrix $\boldsymbol{S}$ is constructed to reflect local connectivity as in the Laplacian operator approximated by the central difference scheme in Finite Difference Method. The autoencoder composed of the encoder and the sparse decoder is trained by using custom pruning in PyTorch [55] pruning module.

In the autoencoder, the number of learnable parameters (i.e., weights and bias) is determined by the number of nodes in the hidden layers in the encoder and the decoder, dimension of latent vector, and the sparsity in the mask matrix. The sparsity is determined by how many nodes in the hidden layer are used to compute one element of the output and how many nodes in the hidden layer are shared for neighboring elements of the output. To generate a mask matrix for 1D problem, we use two variables b and δb, where b denotes the number of nodes in the hidden layer to compute one output element (width of the block in each row in Fig. 4(a)) and δb denotes the amount by which the block shifts. For example, at the ith row, $j \in \{(i-1)db, (i-1)db+1, \cdots, (i-1)db+b\}$th column is one and the others are zero. For a mask of the 2D problem, we create a building matrix in the same way as the mask matrix for 1D problem. Then, we add all rows neighboring ith row (e.g., 5-point stencil for 2D and 7-point stencil for 3D) to ith row and change nonzero values to one. Note that the mask matrix for 2D problem as in Fig. 4(b) looks similar to 2D finite difference Laplacian operator.

There is no way to determine these hyper-parameters *a priori*. If the number of learnable parameters is not enough, the decoder is not able to represent the nonlinear manifold well. On the other hand, too many learnable parameters may result in over-fitting, so the decoder is not able to generalize well, which means the trained decoder can't be used for problems whose data is unseen, i.e., the predictive case. To avoid over-fitting, there are two options to consider. In the first option,



(a) 1D Burgers equation (b) 2D Burgers equation, u (c) 2D Burgers equation, v

Fig. 5. Loss history of decoders for various problems; all three figures show good agreement between train and test loss history, which is a sign for good balance between overfitting and accuracy.

one first divides the data into two sets, i.e., train and test sets. Then, the autoencoder is trained using the train set only and is tested for the generalization ability using the test set. If the mean squared error on the test and train sets are very different, the over-fitting occurs and we should reduce the number of learnable parameters [39].

The second option of avoiding the overfitting is to use Akaike's information criteria (AIC) which is given by

$$\text{AIC} = ln(e) + 2\frac{N_w}{N}$$

where $e = \frac{\|\boldsymbol{X}-\tilde{\boldsymbol{X}}\|_F^2}{2N}$, N_w is the total number of learnable parameters, and N is the number of elements in the data set matrix (i.e., $\boldsymbol{X}$). If one minimizes only the first term of AIC, then an over-fit network will be obtained. On the other hand, if one minimizes only the second term of AIC, i.e., $N_w = 0$, then the network will not fit the training distribution. Therefore, the minimum of AIC helps train a model that is not over-fit and generalizes well. [45,39]. However, finding the minimum of AIC requires a lot more training processes than the first option above. Because of randomness in training, e.g., the random initialization of weights and bias in neural networks and SGD optimization method, e will be different for every training process even with the same N_w and the dataset matrix. Therefore, AIC needs to be averaged over several training for each N_w to find the minimum of AIC.

Because of the practicality of the first option of avoiding the overfitting over the second option, we use the first option in our numerical experiments. For example, as shown in Fig. 5, the mean squares error on the test and train data sets are very close. This implies that the trained autoencoder is not over-fit.

3.3. Nonlinear manifold Galerkin projection

We derive NM-Galerkin using time continuous residual minimization. Replacing $\boldsymbol{x}$ with $\tilde{\boldsymbol{x}}$ given by Eq. (3.1) and $\dot{\boldsymbol{x}}$ with $\dot{\tilde{\boldsymbol{x}}}$ given by Eq. (3.2) in Eq. (2.6) leads to the following residual function with the reduced number of unknowns

$$\tilde{\boldsymbol{r}}(\dot{\hat{\boldsymbol{x}}}, \hat{\boldsymbol{x}}, t; \boldsymbol{\mu}) := \boldsymbol{r}(\boldsymbol{J}_g(\hat{\boldsymbol{x}})\dot{\hat{\boldsymbol{x}}}, \boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}), t; \boldsymbol{\mu}). \tag{3.4}$$

Note that Eq. (3.4) is an over-determined system. Therefore, it is likely that no solution exists. To close the system, we minimize the squared norm of the residual vector function:

$$\dot{\hat{\boldsymbol{x}}} = \underset{\hat{\boldsymbol{v}}\in\mathbb{R}^{n_s}}{\text{argmin}} \|\tilde{\boldsymbol{r}}(\hat{\boldsymbol{v}}, \hat{\boldsymbol{x}}, t; \boldsymbol{\mu})\|_2^2 \tag{3.5}$$

with $\hat{\boldsymbol{x}}(0; \boldsymbol{\mu}) = \hat{\boldsymbol{x}}_0(\boldsymbol{\mu}) = \boldsymbol{h}\left(\boldsymbol{x}_0(\boldsymbol{\mu}) - \boldsymbol{x}_{ref}\right)$. The solution to Eq. (3.5) leads to the NM-Galerkin

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{J}_g(\hat{\boldsymbol{x}})^\dagger \boldsymbol{f}(\boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}), t; \boldsymbol{\mu}), \quad \hat{\boldsymbol{x}}(0; \boldsymbol{\mu}) = \hat{\boldsymbol{x}}_0(\boldsymbol{\mu}) \tag{3.6}$$

where the Moore–Penrose inverse of a matrix $\boldsymbol{A} \in \mathbb{R}^{N_s\times n_s}$ with full column rank is defined as $\boldsymbol{A}^\dagger := (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$.

Applying a time integrator to Eq. (3.6) leads to a fully discretized reduced system, denoted as the reduced OΔE. Note that the reduced OΔE has n_s unknowns and n_s equations. If an implicit time integrator is applied, a Newton–type method can be applied to solve for unknown generalized coordinates each time step. If an explicit time integrator is applied, time marching updates will solve the system. However, we cannot expect any speed-up because the size of the nonlinear terms and their Jacobians, which need to be updated for every Newton step, scales with the FOM size. In order to handle this issue, the hyper-reduction will be applied (see Section 4.3.1).

3.4. Nonlinear manifold least-squares Petrov–Galerkin projection

Alternatively, the nonlinear manifold least-squares Petrov–Galerkin (NM-LSPG) approach projects a fully discretized solution space onto a trial manifold. That is, it discretizes Eq. (2.1) in time domain and replaces $\boldsymbol{x}_n$ with $\tilde{\boldsymbol{x}}_n := \boldsymbol{x}_{ref} + \boldsymbol{g}\left(\hat{\boldsymbol{x}}_n\right)$ for $n \in \mathbb{N}(N_t)$ in residual functions defined in Section 2.1 and Appendix A. Here, we consider only implicit time integrators for

simplicity. See Ref. [44] for other types of time integrators. The residual functions for several implicit time integrators are defined in (2.3), (A.1), and (A.2). For example, the residual function with the backward Euler time integrator[1] after the trial manifold projection becomes

$$\begin{aligned}\tilde{\mathbf{r}}^n_{\mathrm{BE}}(\hat{\boldsymbol{x}}_n;\hat{\boldsymbol{x}}_{n-1},\boldsymbol{\mu}) &:= \boldsymbol{r}^n_{\mathrm{BE}}(\boldsymbol{x}_{ref}+\boldsymbol{g}\left(\hat{\boldsymbol{x}}_n\right);\boldsymbol{x}_{ref}+\boldsymbol{g}\left(\hat{\boldsymbol{x}}_{n-1}\right),\boldsymbol{\mu})\\ &= \boldsymbol{g}\left(\hat{\boldsymbol{x}}_n\right)-\boldsymbol{g}\left(\hat{\boldsymbol{x}}_{n-1}\right)-\Delta t\,\boldsymbol{f}(\boldsymbol{x}_{ref}+\boldsymbol{g}\left(\hat{\boldsymbol{x}}_n\right),t_n;\boldsymbol{\mu}).\end{aligned} \tag{3.7}$$

The nonlinear manifold $\boldsymbol{g}$ can be found by training the autoencoder as described in Section 3.2. Note that Eq. (3.7) is an over-determined system. Therefore, it is likely that no solution exists. To close the system and solve for the unknown generalized coordinates, $\hat{\boldsymbol{x}}_n$, the NM-LSPG takes the squared norm of the residual vector function and minimizes it at every time step:

$$\hat{\boldsymbol{x}}_n = \underset{\hat{\boldsymbol{v}}\in\mathbb{R}^{n_s}}{\operatorname{argmin}} \quad \frac{1}{2}\left\|\tilde{\mathbf{r}}^n_{\mathrm{BE}}(\hat{\boldsymbol{v}};\hat{\boldsymbol{x}}_{n-1},\boldsymbol{\mu})\right\|_2^2. \tag{3.8}$$

The Gauss–Newton method with the starting point $\hat{\boldsymbol{x}}_{n-1}$ is applied to solve the minimization problem (3.8). However, as in the Galerkin approach, a hyper-reduction which will be discussed in Section 4.3.2 is required for a speed-up due to the presence of the nonlinear residual vector function that scales with the full order model size. More specifically, $\boldsymbol{g}\left(\hat{\boldsymbol{x}}_n\right)$, $\boldsymbol{f}(\boldsymbol{x}_{ref}+\boldsymbol{g}\left(\hat{\boldsymbol{x}}_n\right),t;\boldsymbol{\mu})$, and their Jacobians are needed to be updated whenever $\hat{\boldsymbol{x}}_n$ changes if the backward Euler time integrator is used.

4. Hyper-reduction

As mentioned in Section 2.2 and 3, we cannot expect speed-up even though the dimension of unknowns in ROMs is small, i.e., $n_s \ll N_s$, because the nonlinear term still scales with the full order model size. To overcome this issue, there are several hyper-reduction techniques available, e.g., [14,25,26,13,19] for LS-ROMs. These hyper-reduction techniques share a common feature and it plays an important role in the development of the hyper-reduction technique in the NM-ROMs, so we will go over one of the hyper-reduction technique that is commonly used in the LS-ROMs.

4.1. Nonlinear residual approximation

We follow the DEIM-SNS and GNAT-SNS approaches introduced in [19] where the solution snapshots, whose span includes a span of nonlinear term snapshots, are taken to build a nonlinear term basis. Then, it selects a subset of each nonlinear term basis vector to either interpolate or data-fit in a least-squares sense. In this way, it reduces the computational complexity of updating nonlinear terms in an iterative solver for nonlinear problems.

In more details, the GNAT-SNS method approximates the nonlinear residual term with gappy POD [27] as

$$\tilde{\mathbf{r}} \approx \boldsymbol{\Phi}_r \hat{\boldsymbol{r}}, \tag{4.1}$$

where $\boldsymbol{\Phi}_r := [\boldsymbol{\phi}_{r,1},\ldots,\boldsymbol{\phi}_{r,n_r}] \in \mathbb{R}^{N_s\times n_r}$, $n_s \le n_r \ll N_s$, denotes the residual basis matrix and $\hat{\boldsymbol{r}} \in \mathbb{R}^{n_r}$ denotes the generalized coordinates of the nonlinear residual term. Here, $\tilde{\mathbf{r}}$ represents a residual vector function, e.g., the backward Euler residual, $\tilde{\mathbf{r}}^n_{BE}$, defined in Eq. (2.9). The GNAT-SNS method uses the SVD of the FOM solution snapshot matrix to construct $\boldsymbol{\Phi}_r$, which reduces computational cost by avoiding another POD to a nonlinear residual term snapshots. The hyper-reduction method solves the following least-squares problem to obtain the generalized coordinates $\hat{\boldsymbol{r}}$:

$$\hat{\boldsymbol{r}} := \underset{\hat{\boldsymbol{v}}\in\mathbb{R}^{n_r}}{\operatorname{argmin}} \quad \frac{1}{2}\left\|\boldsymbol{Z}^T(\tilde{\mathbf{r}}-\boldsymbol{\Phi}_r\hat{\boldsymbol{v}})\right\|_2^2, \tag{4.2}$$

where $\boldsymbol{Z}^T := [\boldsymbol{e}_{p_1},\ldots,\boldsymbol{e}_{p_{n_z}}]^T \in \mathbb{R}^{n_z\times N_s}$, $n_s \le n_r \le n_z \ll N_s$, is the sampling matrix and $\boldsymbol{e}_{p_i}$ is the p_ith column of the identity matrix $\boldsymbol{I}_{N_s} \in \mathbb{R}^{N_s\times N_s}$. The solution to Eq. (4.2) is given as

$$\hat{\boldsymbol{r}} = (\boldsymbol{Z}^T\boldsymbol{\Phi}_r)^\dagger \boldsymbol{Z}^T\tilde{\mathbf{r}},$$

where the Moore–Penrose inverse of a matrix $\boldsymbol{A} \in \mathbb{R}^{n_z\times n_r}$ with full column rank is defined as $\boldsymbol{A}^\dagger := (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$. Therefore, Eq. (4.1) becomes

$$\tilde{\mathbf{r}} \approx \mathcal{P}\tilde{\mathbf{r}}, \tag{4.3}$$

where $\mathcal{P} := \boldsymbol{\Phi}_r(\boldsymbol{Z}^T\boldsymbol{\Phi}_r)^\dagger\boldsymbol{Z}^T$ is the oblique projection matrix. The projection matrix has a pseudo-inverse instead of the inverse because it allows the oversampling, i.e., $n_r < n_z$. The hyper-reduction method does not construct the sampling matrix $\boldsymbol{Z}$. *Instead, it maintains the sampling indices* $\{p_1,\ldots,p_{n_f}\}$ *and corresponding rows of* $\boldsymbol{\Phi}_r$ *and* $\tilde{\mathbf{r}}$. This enables hyper-reduced ROMs to achieve a speed-up when it is applied to nonlinear problems.

The sampling indices (i.e., $\boldsymbol{Z}$) can be determined by Algorithm 3 of [13] for computational fluid dynamics problems and Algorithm 5 of [11] for other problems. These two algorithms take greedy procedure to minimize the error in the gappy reconstruction of the POD basis vectors $\boldsymbol{\Phi}_r$. These sampling algorithms for the hyper-reduction method allow oversampling (i.e., $n_z > n_r$), resulting in solving least-squares problems in the greedy procedure. These selection algorithms can be viewed as the extension of Algorithm 1 in [14] (i.e., a row pivoted LU decomposition) to the oversampling case. The nonlinear residual term projection error associated with these sampling algorithms is presented in Appendix D of [13]. That is,

$$\|\tilde{\mathbf{r}} - \mathcal{P}\tilde{\mathbf{r}}\|_2 \leq \|\boldsymbol{R}^{-1}\|_2 \|\tilde{\mathbf{r}} - \boldsymbol{\Phi}_r \boldsymbol{\Phi}_r^T \tilde{\mathbf{r}}\|_2$$

where $\boldsymbol{R}$ is the triangular factor from the QR factorization of $\boldsymbol{Z}^T\boldsymbol{\Phi}_r$ (i.e., $\boldsymbol{Z}^T\boldsymbol{\Phi}_r = \boldsymbol{Q}\,\boldsymbol{R}$). For more details, please refer to [19].[2]

4.2. Hyper-reduction for LS-ROM

We present formulations of LS-Galerkin-HR and LS-LSPG-HR. For numerical examples, LS-LSPG-HR is only implemented.

4.2.1. LS-Galerkin-HR

We denote the hyper-reduced linear subspace Galerkin as LS-Galerkin-HR. The LS-Galerkin-HR method approximates the nonlinear residual term with the gappy POD procedure as in Section 4.1. Therefore, the LS-Galerkin-HR method replaces the residual in (2.7) with $\mathcal{P}\tilde{\mathbf{r}}(\hat{\boldsymbol{v}}, \hat{\boldsymbol{x}}, t; \boldsymbol{\mu})$ given by Eq. (4.3). Thus, it minimizes the following least-squares problem:

$$\dot{\hat{\boldsymbol{x}}} = \underset{\hat{\boldsymbol{v}} \in \mathbb{R}^{n_s}}{\operatorname{argmin}} \|(\boldsymbol{Z}^T\boldsymbol{\Phi}_r)^\dagger \boldsymbol{Z}^T \tilde{\mathbf{r}}(\hat{\boldsymbol{v}}, \hat{\boldsymbol{x}}, t; \boldsymbol{\mu})\|_2^2 \tag{4.4}$$

with $\hat{\boldsymbol{x}}(0; \boldsymbol{\mu}) = \hat{\boldsymbol{x}}_0(\boldsymbol{\mu})$. The solution to Eq. (4.4) leads to the following reduced ODE:

$$\dot{\hat{\boldsymbol{x}}} = ((\boldsymbol{Z}^T\boldsymbol{\Phi}_r)^\dagger \boldsymbol{Z}^T\boldsymbol{\Phi})^\dagger ((\boldsymbol{Z}^T\boldsymbol{\Phi}_r)^T \boldsymbol{Z}^T\boldsymbol{\Phi}_r)^{-1} (\boldsymbol{Z}^T\boldsymbol{\Phi}_r)^T \boldsymbol{Z}^T \boldsymbol{f}(\boldsymbol{x}_{ref} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}, t; \boldsymbol{\mu}), \quad \hat{\boldsymbol{x}}(0; \boldsymbol{\mu}) = \hat{\boldsymbol{x}}_0(\boldsymbol{\mu}). \tag{4.5}$$

Applying a time integrator to Eq. (4.5) leads to a fully discretized reduced system, denoted as the reduced OΔE. Note that the reduced OΔE has n_s unknowns and n_s equations. If an implicit time integrator is applied, a Newton–type method can be applied to solve for unknown generalized coordinates each time step. If an explicit time integrator is applied, time marching updates can be applied.

Note that the operator $((\boldsymbol{Z}^T\boldsymbol{\Phi}_r)^\dagger \boldsymbol{Z}^T\boldsymbol{\Phi})^\dagger ((\boldsymbol{Z}^T\boldsymbol{\Phi}_r)^T \boldsymbol{Z}^T\boldsymbol{\Phi}_r)^{-1} (\boldsymbol{Z}^T\boldsymbol{\Phi}_r)^T$ can be pre-computed once for all. We avoid constructing the sampling matrix $\boldsymbol{Z}$. For example, the operator $\boldsymbol{Z}^T\boldsymbol{\Phi}_r$ can be computed simply by extracting only the selected rows of $\boldsymbol{\Phi}_r$. For the term, $\boldsymbol{Z}^T\boldsymbol{f}$, only the nonlinear term elements that are selected by the sampling matrix need to be computed. This implies that we have to keep track of the rows of $\boldsymbol{\Phi}$ that are needed to compute the selected nonlinear term elements, which is usually a larger set than the rows that are selected solely by the sampling matrix, i.e., $\boldsymbol{Z}^T\boldsymbol{\Phi}$, as in the 5-point stencil or 7-point stencil in the finite difference method.

4.2.2. LS-LSPG-HR

We denote the hyper-reduced linear subspace LSPG as LS-LSPG-HR. The LS-LSPG-HR method approximates the nonlinear residual term with the gappy POD procedure as in Section 4.1. Therefore, the LS-LSPG-HR method replaces the residual in (2.10) with $\mathcal{P}\tilde{\mathbf{r}}(\hat{\boldsymbol{v}}, \hat{\boldsymbol{x}}, t; \boldsymbol{\mu})$ given by Eq. (4.3). Thus, it minimizes the following least-squares problem:

$$\hat{\boldsymbol{x}}_n = \underset{\hat{\boldsymbol{v}} \in \mathbb{R}^{n_s}}{\operatorname{argmin}} \quad \frac{1}{2} \left\| (\boldsymbol{Z}^T\boldsymbol{\Phi}_r)^\dagger \boldsymbol{Z}^T \tilde{\mathbf{r}}^n_{\mathrm{BE}}(\hat{\boldsymbol{v}}; \hat{\boldsymbol{x}}_{n-1}, \boldsymbol{\mu}) \right\|_2^2,$$

with $\hat{\boldsymbol{x}}(0; \boldsymbol{\mu}) = \hat{\boldsymbol{x}}_0(\boldsymbol{\mu})$. Note that the pseudo-inverse $(\boldsymbol{Z}^T\boldsymbol{\Phi}_r)^\dagger$ can be pre-computed once for all. Due to the definition of $\tilde{\mathbf{r}}^n_{\mathrm{BE}}$ in Eq. (2.9), the sampling matrix $\boldsymbol{Z}$ needs to be applied to the following terms: $\boldsymbol{\Phi}(\hat{\boldsymbol{x}}_n - \hat{\boldsymbol{x}}_{n-1})$ and $\boldsymbol{f}(\boldsymbol{x}_{ref} + \boldsymbol{\Phi}\hat{\boldsymbol{x}}_n, t; \boldsymbol{\mu})$ at every time step. The first term $\boldsymbol{Z}^T\boldsymbol{\Phi}$ can be precomputed by extracting the selected rows of the basis matrix. For the second term, only the nonlinear term elements that are selected by the sampling matrix need to be computed. This implies that we have to keep track of the rows of $\boldsymbol{\Phi}$ that are needed to compute the selected nonlinear term elements, which is usually a larger set than the rows that are selected solely by the sampling matrix, i.e., $\boldsymbol{Z}^T\boldsymbol{\Phi}$, as in the 5-point stencil or 7-point stencil in the finite difference method.

4.3. Hyper-reduction for NM-ROM

There are two layers of nonlinear terms in the NM-ROM: (i) the nonlinear term in the original governing equations, i.e., $\boldsymbol{f}$ in Eq. (2.1), and (ii) the decoder, which is nonlinear function of the generalized coordinates, i.e., $\boldsymbol{g}$ in Eq. (3.1) and appears in the definition of residuals both for Galerkin and Petrov–Galerkin cases. The first layer nonlinear term can be treated in

[2] In this paper, GNAT-SNS in [19] is re-named as LS-LSPG-HR to emphasize the difference between the LS-ROMs and NM-ROMs.

the same way as the LS-ROMs (see Sections 4.2.1 and 4.2.2). Now, it is the second layer nonlinear term that requires a special attention. For example, the Jacobian of the decoder needs to be evaluated at every solver iteration. Because the cost of computing the Jacobian scales with the number of learnable parameters in the decoder, we cannot expect much speed-up. As we did in the hyper-reduction process of the LS-ROMs, we have to avoid computing all the entries of the decoder or its Jacobian because they scale with the full order model size. This will be achieved by constructing a subnet that computes only the relevant outputs, which is discussed in Section 4.4. First, we state the hyper-reduced NM-ROMs, i.e., the NM-Galerkin-HR in Section 4.3.1 and the NM-LSPG-HR in Section 4.3.2. At last, the flop count estimate comparison between non-hyper-reduced and hyper-reduced models is shown at the end of Section 4.4 and their derivations are shown in Appendix B.

4.3.1. NM-Galerkin-HR

Now, we apply the hyper-reduction to the NM-Galerkin method. We denote the hyper-reduced nonlinear manifold Galerkin as NM-Galerkin-HR. The NM-Galerkin-HR method approximates the nonlinear residual term with the gappy POD procedure as in Section 4.1. Therefore, the NM-Galerkin-HR method replaces the residual in (3.5) with $\mathcal{P}\tilde{\mathbf{r}}(\hat{\boldsymbol{v}}, \hat{\boldsymbol{x}}, t; \boldsymbol{\mu})$ given by Eq. (4.3). Thus, it minimizes the following least-squares problem:

$$\dot{\hat{\boldsymbol{x}}} = \underset{\hat{\boldsymbol{v}} \in \mathbb{R}^{n_s}}{\operatorname{argmin}} \|(\boldsymbol{Z}^T \boldsymbol{\Phi}_r)^\dagger \boldsymbol{Z}^T \tilde{\mathbf{r}}(\hat{\boldsymbol{v}}, \hat{\boldsymbol{x}}, t; \boldsymbol{\mu})\|_2^2 \tag{4.6}$$

with $\hat{\boldsymbol{x}}(0; \boldsymbol{\mu}) = \hat{\boldsymbol{x}}_0(\boldsymbol{\mu})$. The solution to Eq. (4.6) leads to the NM-Galerkin-HR

$$\dot{\hat{\boldsymbol{x}}} = ((\boldsymbol{Z}^T \boldsymbol{\Phi}_r)^\dagger \boldsymbol{Z}^T \boldsymbol{J}_g(\hat{\boldsymbol{x}}))^\dagger (\boldsymbol{Z}^T \boldsymbol{\Phi}_r)^\dagger \boldsymbol{Z}^T \boldsymbol{f}(\boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}), t; \boldsymbol{\mu}), \quad \hat{\boldsymbol{x}}(0; \boldsymbol{\mu}) = \hat{\boldsymbol{x}}_0(\boldsymbol{\mu}). \tag{4.7}$$

Applying a time integrator to Eq. (4.7) leads to a fully discretized reduced system, denoted as the reduced OΔE. Note that the reduced OΔE has n_s unknowns and n_s equations. If an implicit time integrator is applied, a Newton–type method can be applied to solve for unknown generalized coordinates each time step. If an explicit time integrator is applied, time marching updates will solve the system.

Note that the pseudo inverse, $(\boldsymbol{Z}^T \boldsymbol{\Phi}_r)^\dagger$, can be pre-computed once for all by extracting only the selected rows of $\boldsymbol{\Phi}_r$. However, the term, $\boldsymbol{Z}^T \boldsymbol{J}_g(\hat{\boldsymbol{x}})$, cannot be precomputed because $\boldsymbol{J}_g$ needs to be updated every time $\hat{\boldsymbol{x}}$ is updated. Fortunately, we need to compute only the selected rows of $\boldsymbol{J}_g$ by the sampling matrix $\boldsymbol{Z}$. Similarly, for the term, $\boldsymbol{Z}^T \boldsymbol{f}$, only the nonlinear term elements that are selected by the sampling matrix need to be computed. This implies that we have to keep track of the outputs of $\boldsymbol{g}$ that are needed to compute the selected nonlinear term elements, which is usually a larger set than the outputs that are selected solely by the sampling matrix, i.e., $\boldsymbol{Z}^T \boldsymbol{g}$, as in the 5-point stencil or 7-point stencil in the finite difference method.

4.3.2. NM-LSPG-HR

We apply the hyper-reduction to the NM-LSPG method discussed in Section 3.4. The hyper-reduction procedure for the nonlinear residual function after the trial manifold projection is the same as the one in Section 4.1, i.e., we replace the residual defined in (3.7) with $\mathcal{P}\tilde{\mathbf{r}}^n_{BE}$ and plug it into the minimization problem in Eq. (3.8). Then, the minimization problem becomes

$$\hat{\boldsymbol{x}}_n = \underset{\hat{\boldsymbol{v}} \in \mathbb{R}^{n_s}}{\operatorname{argmin}} \quad \frac{1}{2} \left\| (\boldsymbol{Z}^T \boldsymbol{\Phi}_r)^\dagger \boldsymbol{Z}^T \tilde{\mathbf{r}}^n_{\mathrm{BE}}(\hat{\boldsymbol{v}}; \hat{\boldsymbol{x}}_{n-1}, \boldsymbol{\mu}) \right\|_2^2.$$

Note that the pseudo-inverse $(\boldsymbol{Z}^T \boldsymbol{\Phi}_r)^\dagger$ can be pre-computed once for all. Due to the definition of $\tilde{\mathbf{r}}^n_{\mathrm{BE}}$ in Eq. (3.7), the sampling matrix $\boldsymbol{Z}$ needs to be applied the following two terms: $\boldsymbol{g}(\hat{\boldsymbol{x}}_n) - \boldsymbol{g}(\hat{\boldsymbol{x}}_{n-1})$ and $\boldsymbol{f}(\boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}_n), t; \boldsymbol{\mu})$ at every time step. The first term, $\boldsymbol{Z}^T(\boldsymbol{g}(\hat{\boldsymbol{x}}_n) - \boldsymbol{g}(\hat{\boldsymbol{x}}_{n-1}))$, requires to compute only the selected outputs of the decoder. Furthermore, for the second term, only the nonlinear term elements that are selected by the sampling matrix need to be computed. This implies that we have to keep track of the outputs of $\boldsymbol{g}$ that are needed to compute the selected nonlinear term elements by the sampling matrix, which is usually a larger set than the outputs that are selected solely by the sampling matrix, i.e., $\boldsymbol{Z}^T \boldsymbol{g}$, as in the 5-point stencil or 7-point stencil in the finite difference method. Therefore, we build a subnet that computes only the outputs of the decoder that is required to compute the elements of the nonlinear term, $\boldsymbol{f}$. Then, with the same subnet, the outputs required for the first term, $\boldsymbol{Z}^T(\boldsymbol{g}(\hat{\boldsymbol{x}}_n) - \boldsymbol{g}(\hat{\boldsymbol{x}}_{n-1}))$, can be extracted from the same subnet. The construction of the subnet is explained in Section 4.4.1.

4.4. Efficient hyper-reduction decoder computation

In the NM-LSPG-HR method, the residual is evaluated at the sampling points given by the hyper-reduction. We use "sample points" and "hyper-reduction indices" interchangeably throughout the paper. Evaluating the decoder and its Jacobian can be done efficiently by restricting the computation to the active paths of the outputs required to compute the selected residual elements. For example, active paths of the sparse decoder are shown in orange color in Fig. 3(b). The costs of computing the decoder and its Jacobian scale piecewise-linearly with the number of sample points because the slopes

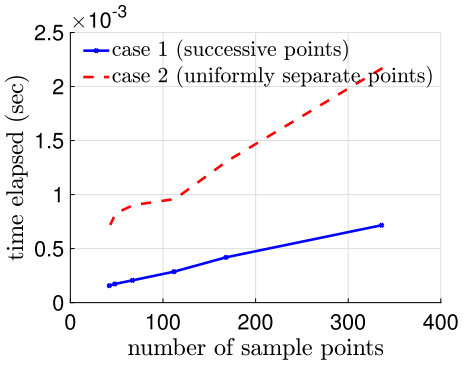


Fig. 6. Illustration of the computational time elapsed for the evaluation of decoder and its Jacobian vs the number of sample points from 2D Burgers' equation in Section 6.2. The total number of points is 3364.

of the costs of computing the decoder and its Jacobian vs the number of sample points are different depending on how many nodes in hidden layer are shared for each sample point (see Fig. 6). The distribution of the hyper-reduction indices determines the number of overlapping nodes in hidden layer of decoder. The more the overlapping nodes in hidden layer imply the more efficient computation of the hyper-reduced decoder. If successive points are selected, overlapping of nodes in hidden layer are maximized. If the selected points are uniformly apart, then the overlapping of nodes in hidden layer is minimized. In the case of random distribution, if the number of selected points is small, the possibility of overlapping is low. Our required outputs to compute the selected residual elements after the hyper-reduction are neither successive nor uniformly separated. Thus, the cost of computing the decoder and its Jacobian would be between case 1 (successive points) and case 2 (uniformly separated points) in Fig. 6. By restricting our computation to active paths, we only compute along the subnet of the decoder network that is needed for our required outputs.

4.4.1. Construction of a subnet

To determine the sparse decoder's active paths for given hyper-reduction indices together with additional indices required to compute the hyper-reduced residual (i.e., the neighbor indices that are used to approximate the derivatives at the sample point), denoted as $\mathcal{H}$, we follow the steps below:

Step 1: Set nonlinear activation functions to be identity functions.

Step 2: Replace nonzero elements of the weight matrices, $\boldsymbol{W}_1, \boldsymbol{W}_2 \odot \boldsymbol{S}$ and the bias vector, $\boldsymbol{b}_1$ with one and then denote each of them as $\tilde{\boldsymbol{W}}_1$, $\tilde{\boldsymbol{W}}_2$, and $\tilde{\boldsymbol{b}}_1$, respectively. Zero elements of $\tilde{\boldsymbol{W}}_1$, $\tilde{\boldsymbol{W}}_2$, and $\tilde{\boldsymbol{b}}_1$ represent non-connected edges between layers.

Step 3: A new decoder model, $\tilde{\boldsymbol{g}}(\boldsymbol{y}_0)$, is created in the form

$$\tilde{\boldsymbol{g}}(\boldsymbol{y}_0) = \tilde{\boldsymbol{W}}_2(\tilde{\boldsymbol{W}}_1\boldsymbol{y}_0 + \tilde{\boldsymbol{b}}_1)$$

or for each layer, we can write

$$\boldsymbol{y}_1 = \tilde{\boldsymbol{W}}_1\boldsymbol{y}_0 + \tilde{\boldsymbol{b}}_1$$
$$\boldsymbol{y}_2 = \tilde{\boldsymbol{W}}_2\boldsymbol{y}_1.$$

Step 4: Set $\boldsymbol{y}_0 = (1, \cdots, 1)^T \in \mathbb{R}^{n_s}$ as an input. By construction, $\tilde{\boldsymbol{g}}(\boldsymbol{y}_0)$ must be all positive.

Step 5: Define the target vector as $\boldsymbol{y}_* = \boldsymbol{y}_2 - \mathbf{e} \in \mathbb{R}^{N_s}$, where ith component of the error vector, $\mathbf{e}$, is defined as $e_i = \delta_{ij}$, $j \in \mathcal{H}$. Then the loss function, L, is defined as

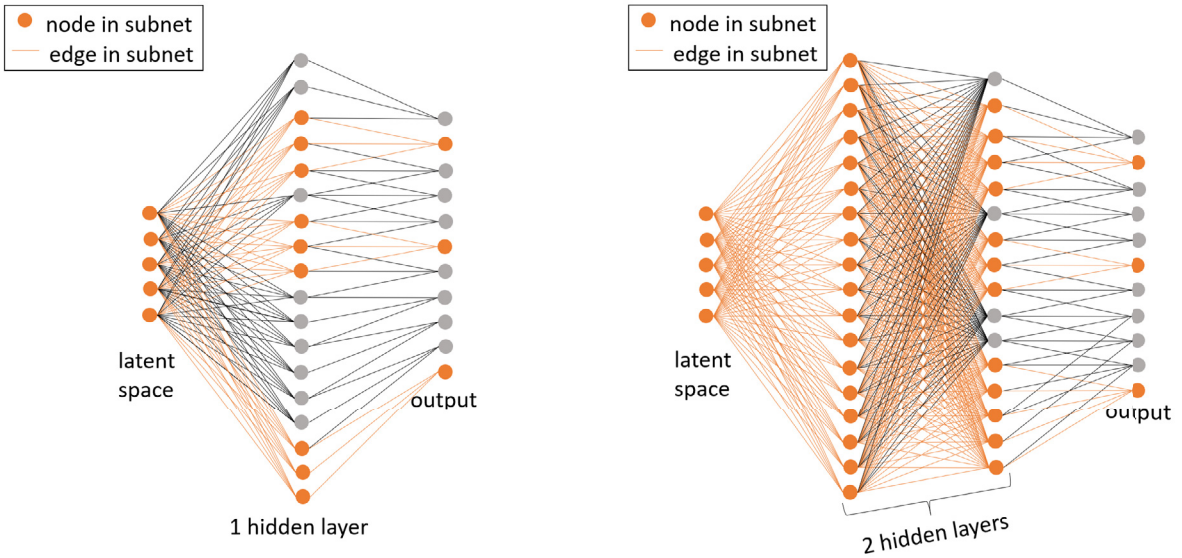
$$L = \frac{1}{2}\|\boldsymbol{y}_2 - \boldsymbol{y}_*\|_2^2$$

and $\frac{\partial L}{\partial \boldsymbol{y}_2}$ is given by

$$\frac{\partial L}{\partial \boldsymbol{y}_2} = \boldsymbol{y}_2 - \boldsymbol{y}_* = \mathbf{e}.$$

Step 6: Compute $\nabla_{\tilde{\boldsymbol{W}}_2} L$, $\nabla_{\tilde{\boldsymbol{W}}_1} L$, and $\nabla_{\tilde{\boldsymbol{b}}_1} L$ using the chain rule

$$\nabla_{\tilde{\boldsymbol{W}}_2} L = \frac{\partial L}{\partial \tilde{\boldsymbol{W}}_2} = \frac{\partial L}{\partial \boldsymbol{y}_2}\frac{\partial \boldsymbol{y}_2}{\partial \tilde{\boldsymbol{W}}_2} = \left(\frac{\partial L}{\partial \boldsymbol{y}_2}\boldsymbol{y}_1^T\right) \odot s(\tilde{\boldsymbol{W}}_2)$$



(a) Shallow network (b) Deep network

Fig. 7. Illustration of the effect on the sparsity of the active path for shallow network vs deep network. The shallow network provides a sparser network than the deep network in the subnet. Therefore, the shallow network is expected to achieve a higher speed-up than the deep network.

$$\nabla_{\tilde{\boldsymbol{W}}_1} L = \frac{\partial L}{\partial \tilde{\boldsymbol{W}}_1} = \frac{\partial L}{\partial \boldsymbol{y}_1}\frac{\partial \boldsymbol{y}_1}{\partial \tilde{\boldsymbol{W}}_1} = \frac{\partial L}{\partial \boldsymbol{y}_2}\frac{\partial \boldsymbol{y}_2}{\partial \boldsymbol{y}_1}\frac{\partial \boldsymbol{y}_1}{\partial \tilde{\boldsymbol{W}}_1} = \left(\tilde{\boldsymbol{W}}_2^T \frac{\partial L}{\partial \boldsymbol{y}_2}\boldsymbol{y}_0^T\right) \odot s(\tilde{\boldsymbol{W}}_1)$$

$$\nabla_{\tilde{\boldsymbol{b}}_1} L = \frac{\partial L}{\partial \tilde{\boldsymbol{b}}_1} = \frac{\partial L}{\partial \boldsymbol{y}_1}\frac{\partial \boldsymbol{y}_1}{\partial \tilde{\boldsymbol{b}}_1} = \frac{\partial L}{\partial \boldsymbol{y}_2}\frac{\partial \boldsymbol{y}_2}{\partial \boldsymbol{y}_1}\frac{\partial \boldsymbol{y}_1}{\partial \tilde{\boldsymbol{b}}_1} = \left(\tilde{\boldsymbol{W}}_2^T \frac{\partial L}{\partial \boldsymbol{y}_2}\right) \odot s(\tilde{\boldsymbol{b}}_1)$$

where

$$s(x) := \begin{cases} 0 & if \quad x \le 0 \\ 1 & otherwise \end{cases}$$

is the element-wise function. Here, we make derivatives of L with respect to non-connected edges (i.e., zero elements of $\tilde{\boldsymbol{W}}_2$, $\tilde{\boldsymbol{W}}_1$, and $\tilde{\boldsymbol{b}}_1$) zeros by element-wise multiplication with $s(\tilde{\boldsymbol{W}}_2)$, $s(\tilde{\boldsymbol{W}}_1)$, and $s(\tilde{\boldsymbol{b}}_1)$ because we do not consider non-connected edges as variables.

Step 7: Using the fact that the weights and bias that are not in the active paths do not contribute to computing L, we deduce that the derivatives of L with respect to such weights and bias are zero. On the other hand, the derivatives of L with respect to the weights and bias that are in the support of indices in $\mathcal{H}$ must be strictly positive because the special structure of $\tilde{\boldsymbol{g}}$ (i.e., the same structure as the sparse decoder, $\boldsymbol{g}$, except all the nonzero weights and bias are one and the nonlinear activation functions are identity functions), choosing the all-ones vector as input vector, and defining the target vector as above should induce the positive gradient to reduce the L. Thus, active path weights and bias are obtained by

$$\begin{aligned} \boldsymbol{W}_2^a &= (\boldsymbol{W}_2 \odot \boldsymbol{S}) \odot s(\nabla_{\tilde{\boldsymbol{W}}_2} L) \\ \boldsymbol{W}_1^a &= \boldsymbol{W}_1 \odot s(\nabla_{\tilde{\boldsymbol{W}}_1} L) \\ \boldsymbol{b}_1^a &= \boldsymbol{b}_1 \odot s(\nabla_{\tilde{\boldsymbol{b}}_1} L). \end{aligned}$$

Step 8: Removing zero rows and zero columns of the active path weights and bias, $\boldsymbol{W}_2^a$, $\boldsymbol{W}_1^a$, and $\boldsymbol{b}_1^a$ yields the subnet weights and bias, which are denoted as $\boldsymbol{W}_2^{sn}$, $\boldsymbol{W}_1^{sn}$, and $\boldsymbol{b}_1^{sn}$, respectively. Then the subnet, $\boldsymbol{g}^{sn}$ is given by

$$\boldsymbol{g}^{sn}(\hat{\boldsymbol{x}}) = \boldsymbol{W}_2^{sn}\sigma(\boldsymbol{W}_1^{sn}\hat{\boldsymbol{x}} + \boldsymbol{b}_1^{sn}).$$

This subnet strategy works for neural networks of arbitrary depth. However, we have illustrated it in the context of the neural network with one hidden layer. It is because that is what we use to achieve enough speed up. Please see Fig. 7 for an argument of a shallow over a deep network.

Remark 4.1. To count flops of NM-LSPG and NM-LSPG-HR, let m, f, z, and b denote FOM size, ROM size, the size of subnet output, and the number of nodes in the hidden layer to compute one output element of the sparse decoder, respectively. Then, the flop counts of NM-LSPG is $\mathcal{O}(mbf)$ and the flop counts of NM-LSPG-HR is $\mathcal{O}(zbf) + \mathcal{O}(fz^2)$. Thus, if z is small e.g., $z^2 < m$, speed-up can be achieved. For full details, see Appendix B.

5. Error analysis

We present error analysis of the NM-LSPG-HR method. The error analysis is based on [43] and we added an oblique projection matrix for a hyper-reduction method. *A posteriori* discrete-time error bounds for NM-Galerkin and NM-LSPG without any hyper-reduction are derived in [43]. Here, we apply a linear multi-step method described in Appendix A.

Theorem 5.1. *Let $\boldsymbol{Z} \in \mathbb{R}^{N_s \times n_z}$ with $n_z \ll N_s$ denote a sampling matrix, $\mathcal{P} \in \mathbb{R}^{N_s \times N_s}$ be an oblique projection matrix used in NM-Galerkin-HR and NM-LSPG-HR, i.e., $\mathcal{P} = \mathcal{P}_{\text{NM-Galerkin-HR}}$ for NM-Galerkin-HR and $\mathcal{P} = \mathcal{P}_{\text{NM-LSPG-HR}}$ for NM-LSPG-HR, and $\tilde{\mathbf{r}}^n \in \mathbb{R}^{N_s}$ denote the nonlinear residual term, which is defined by replacing $\boldsymbol{x}_n$ with $\tilde{\boldsymbol{x}}_n := \boldsymbol{x}_{ref} + \boldsymbol{g}\left(\hat{\boldsymbol{x}}_n\right)$ for $n \in \mathbb{N}(N_t)$ in residual functions defined in Section 2.1 and Appendix A, e.g., the residual with the backward Euler time integrator is defined in Sections 3.4 and 4.3.2. Then, if the velocity $\boldsymbol{f}$ is Lipschitz continuous with the Lipschitz constant L and the time step size Δt is sufficiently small such that $\Delta t < \frac{\gamma_1|\alpha_0|}{\gamma_2|\beta_0|L}$, we have the following error bound*

$$\begin{aligned}\|\boldsymbol{x}_n - \boldsymbol{x}_{ref} - \boldsymbol{g}(\hat{\boldsymbol{x}}_n)\|_2 \leq & \frac{1}{\|\mathcal{P}\|_2(\gamma_1 - \gamma_2\frac{|\beta_0|\Delta tL}{|\alpha_0|})|\alpha_0|}\left\|\mathcal{P}\tilde{\mathbf{r}}^n(\boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}_n))\right\|_2 \\ & + \sum_{j=1}^{k}\frac{|\alpha_j| + |\beta_j|\Delta tL}{(\gamma_1 - \gamma_2\frac{|\beta_0|\Delta tL}{|\alpha_0|})|\alpha_0|}\left\|\boldsymbol{x}_{n-j} - \boldsymbol{x}_{ref} - \boldsymbol{g}(\hat{\boldsymbol{x}}_{n-j})\right\|_2\end{aligned} \tag{5.1}$$

for NM-Galerkin-HR and

$$\begin{aligned}\|\boldsymbol{x}_n - \boldsymbol{x}_{ref} - \boldsymbol{g}(\hat{\boldsymbol{x}}_n)\|_2 \leq & \frac{1}{\|\mathcal{P}\|_2(\gamma_1|\alpha_0| - \gamma_2|\beta_0|\Delta tL)}\min_{\hat{\boldsymbol{v}}\in\mathbb{R}^{n_s}}\left\|\mathcal{P}\tilde{\mathbf{r}}^n(\hat{\boldsymbol{v}}; \hat{\boldsymbol{x}}_{n-1}, \cdots, \hat{\boldsymbol{x}}_{n-k}, \boldsymbol{\mu})\right\|_2 \\ & + \sum_{j=1}^{k}\frac{|\alpha_j| + |\beta_j|\Delta tL}{(\gamma_1|\alpha_0| - \gamma_2|\beta_0|\Delta tL)}\left\|\boldsymbol{x}_{n-j} - \boldsymbol{x}_{ref} - \boldsymbol{g}(\hat{\boldsymbol{x}}_{n-j})\right\|_2\end{aligned} \tag{5.2}$$

for NM-LSPG-HR, where coefficients $\alpha_j,\ \beta_j \in \mathbb{R},\ j = 0, \cdots, k$ define a particular linear multi-step scheme and $\gamma_1, \gamma_2 \in \mathbb{R}$ are $0 < \gamma_1, \gamma_2 \leq 1$.

Proof. We have

$$\mathbf{r}^n(\boldsymbol{x}_n) = \sum_{j=0}^{k}\alpha_j\boldsymbol{x}_{n-j} - \Delta t\sum_{j=0}^{k}\beta_j\boldsymbol{f}(\boldsymbol{x}_{n-j}) = 0, \tag{5.3}$$

$$\mathcal{P}\tilde{\mathbf{r}}^n(\boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}_n)) = \mathcal{P}\left(\sum_{j=0}^{k}\alpha_j\left(\boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}_{n-j})\right) - \Delta t\sum_{j=0}^{k}\beta_j\boldsymbol{f}\left(\boldsymbol{x}_{ref} + \boldsymbol{g}\left(\hat{\boldsymbol{x}}_{n-j}\right)\right)\right) \tag{5.4}$$

where $\boldsymbol{x}_n \in \mathbb{R}^{N_s}$ denotes FOM solution and $\boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}_n),\ \hat{\boldsymbol{x}}_n \in \mathbb{R}^{n_s}$ is approximate solution.

Subtracting Eq. (5.3) from Eq. (5.4) gives

$$\begin{aligned}-\mathcal{P}\tilde{\mathbf{r}}^n\left(\boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}_n)\right) = \mathcal{P}\Bigg(& \alpha_0\left(\boldsymbol{x}_n - \boldsymbol{x}_{ref} - \boldsymbol{g}(\hat{\boldsymbol{x}}_n)\right) - \Delta t\beta_0\left(\boldsymbol{f}(\boldsymbol{x}_n) - \boldsymbol{f}(\boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}_n))\right) \\ & + \sum_{j=1}^{k}\alpha_j\left(\boldsymbol{x}_{n-j} - \boldsymbol{x}_{ref} - \boldsymbol{g}(\hat{\boldsymbol{x}}_{n-j})\right) - \Delta t\sum_{j=1}^{k}\beta_j\left(\boldsymbol{f}\left(\boldsymbol{x}_{n-j}\right) - \boldsymbol{f}\left(\boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}_{n-j})\right)\right)\Bigg).\end{aligned}$$

We can re-write this in the following form

$$\underbrace{\mathcal{P}\left(\boldsymbol{x}_n - \boldsymbol{x}_{ref} - \boldsymbol{g}(\hat{\boldsymbol{x}}_n) - \frac{\beta_0\Delta t}{\alpha_0}\left(\boldsymbol{f}(\boldsymbol{x}_n) - \boldsymbol{f}(\boldsymbol{x}_{ref} + \boldsymbol{g}(\hat{\boldsymbol{x}}_n))\right)\right)}_{\text{I}} =$$

$$\underbrace{-\frac{1}{\alpha_0}\mathcal{P}\tilde{\mathbf{r}}^n(\boldsymbol{x}_{ref}+\boldsymbol{g}(\hat{\boldsymbol{x}}_n))+\mathcal{P}\left(-\frac{1}{\alpha_0}\sum_{j=1}^{k}\alpha_j\left(\boldsymbol{x}_{n-j}-\boldsymbol{x}_{ref}-\boldsymbol{g}(\hat{\boldsymbol{x}}_{n-j})\right)+\frac{\Delta t}{\alpha_0}\sum_{j=1}^{k}\beta_j\left(\boldsymbol{f}\left(\boldsymbol{x}_{n-j}\right)-\boldsymbol{f}\left(\boldsymbol{x}_{ref}+\boldsymbol{g}(\hat{\boldsymbol{x}}_{n-j})\right)\right)\right)}_{\mathrm{II}}$$

Applying the reverse triangle inequality gives

$$\|\mathrm{I}\|_2 \geq \left| \|\mathcal{P}(\boldsymbol{x}_n-\boldsymbol{x}_{ref}-\boldsymbol{g}(\hat{\boldsymbol{x}}_n))\|_2 - \left\|\frac{\beta_0\Delta t}{\alpha_0}\mathcal{P}(\boldsymbol{f}(\boldsymbol{x}_n)-\boldsymbol{f}(\boldsymbol{x}_{ref}+\boldsymbol{g}(\hat{\boldsymbol{x}}_n)))\right\|_2 \right|.$$

Now, we use relationships

$$\|\mathcal{P}(\boldsymbol{x}_n-\boldsymbol{x}_{ref}-\boldsymbol{g}(\hat{\boldsymbol{x}}_n))\|_2 = \gamma_1\|\mathcal{P}\|_2\|\boldsymbol{x}_n-\boldsymbol{x}_{ref}-\boldsymbol{g}(\hat{\boldsymbol{x}}_n)\|_2$$

and

$$\|\mathcal{P}(\boldsymbol{f}(\boldsymbol{x}_n)-\boldsymbol{f}(\boldsymbol{x}_{ref}+\boldsymbol{g}(\hat{\boldsymbol{x}}_n)))\|_2 = \gamma_2\|\mathcal{P}\|_2\|\boldsymbol{f}(\boldsymbol{x}_n)-\boldsymbol{f}(\boldsymbol{x}_{ref}+\boldsymbol{g}(\hat{\boldsymbol{x}}_n))\|_2$$

where $0<\gamma_1\leq 1$ and $0<\gamma_2\leq 1$. Then, we have

$$\begin{aligned}\|\mathrm{I}\|_2 &\geq \left|\gamma_1\|\mathcal{P}\|_2\|\boldsymbol{x}_n-\boldsymbol{x}_{ref}-\boldsymbol{g}(\hat{\boldsymbol{x}}_n)\|_2-\gamma_2\|\mathcal{P}\|_2\left\|\frac{\beta_0\Delta t}{\alpha_0}(\boldsymbol{f}(\boldsymbol{x}_n)-\boldsymbol{f}(\boldsymbol{x}_{ref}+\boldsymbol{g}(\hat{\boldsymbol{x}}_n)))\right\|_2\right| \\ &= \|\mathcal{P}\|_2\left|\gamma_1\|\boldsymbol{x}_n-\boldsymbol{x}_{ref}-\boldsymbol{g}(\hat{\boldsymbol{x}}_n)\|_2-\gamma_2\left\|\frac{\beta_0\Delta t}{\alpha_0}(\boldsymbol{f}(\boldsymbol{x}_n)-\boldsymbol{f}(\boldsymbol{x}_{ref}+\boldsymbol{g}(\hat{\boldsymbol{x}}_n)))\right\|_2\right|.\end{aligned}$$

If f is Lipschitz continuous with L and Δt is sufficiently small such that $\Delta t<\frac{\gamma_1|\alpha_0|}{\gamma_2|\beta_0|L}$, we have

$$\|\mathrm{I}\|_2 \geq \|\mathcal{P}\|_2\left(\gamma_1-\gamma_2\frac{|\beta_0|\Delta t L}{|\alpha_0|}\right)\|\boldsymbol{x}_n-\boldsymbol{x}_{ref}-\boldsymbol{g}(\hat{\boldsymbol{x}}_n)\|_2. \tag{5.5}$$

With triangle inequality and Lipschitz continuity of f, we have

$$\|\mathrm{II}\|_2 \leq \frac{1}{|\alpha_0|}\|\mathcal{P}\tilde{\mathbf{r}}^n(\boldsymbol{x}_{ref}+\boldsymbol{g}(\hat{\boldsymbol{x}}_n))\|_2+\|\mathcal{P}\|_2\frac{1}{|\alpha_0|}\sum_{j=1}^{k}\left((|\alpha_j|+|\beta_j|\Delta t L)\|\boldsymbol{x}_{n-j}-\boldsymbol{x}_{ref}-\boldsymbol{g}(\hat{\boldsymbol{x}}_{n-j})\|_2\right). \tag{5.6}$$

Combining Eq. (5.5) and (5.6) yields

$$\begin{aligned}\|\boldsymbol{x}_n-\boldsymbol{x}_{ref}-\boldsymbol{g}(\hat{\boldsymbol{x}}_n)\|_2 \leq{}& \frac{1}{\|\mathcal{P}\|_2(\gamma_1-\gamma_2\frac{|\beta_0|\Delta t L}{|\alpha_0|})|\alpha_0|}\left\|\mathcal{P}\tilde{\mathbf{r}}^n(\boldsymbol{x}_{ref}+\boldsymbol{g}(\hat{\boldsymbol{x}}_n))\right\|_2 \\ &+\sum_{j=1}^{k}\frac{|\alpha_j|+|\beta_j|\Delta t L}{(\gamma_1-\gamma_2\frac{|\beta_0|\Delta t L}{|\alpha_0|})|\alpha_0|}\left\|\boldsymbol{x}_{n-j}-\boldsymbol{x}_{ref}-\boldsymbol{g}(\hat{\boldsymbol{x}}_{n-j})\right\|_2.\end{aligned} \tag{5.7}$$

The error bound for NM-Galerkin-HR Eq. (5.1) is proved. Furthermore, noting that NM-LSPG-HR solution $\hat{\boldsymbol{x}}_n$ minimizes the term $\|\mathcal{P}\tilde{\mathbf{r}}^n(\boldsymbol{x}_{ref}+\boldsymbol{g}(\hat{\boldsymbol{x}}_n))\|_2$ in Eq. (5.7) proves the error bound for NM-LSPG-HR Eq. (5.2). □

From the error bound for NM-LSPG-HR, we know that the NM-LSPG-HR solutions satisfy sequential minimization of the error bound.

6. Numerical results

We demonstrate the accuracy and speed-up of the nonlinear manifold reduced order model for two advection-dominated problems: (i) a parameterized 1D inviscid Burgers equation in Section 6.1 and (ii) a parameterized 2D viscous Burgers equation with a large Reynolds number (i.e., the advection-dominated case) in Section 6.2. The ROMs are trained with solution snapshot associated with train parameters in a parameter space and are used to predict the solution of the parameter that is not included in the train parameters. We refer this to the predictive case. The accuracy of ROM solution $\tilde{\boldsymbol{x}}(\cdot;\boldsymbol{\mu})$ is assessed from its maximum relative error:

$$\text{maximum relative error} = \max_{n\in\mathbb{N}(N_t)}\left(\frac{\|\tilde{\boldsymbol{x}}(t^n;\boldsymbol{\mu})-\boldsymbol{x}(t^n;\boldsymbol{\mu})\|_2}{\|\boldsymbol{x}(t^n;\boldsymbol{\mu})\|_2}\right)$$

where $\boldsymbol{x}$ is the corresponding FOM solution. We also introduce the projection errors for the lower bounds of LS-ROMs and NM-ROMs maximum relative errors:

$$\text{projection error} = \sqrt{\sum_{n=1}^{N_t} \| \left(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T\right) \left(\boldsymbol{x}(t^n;\boldsymbol{\mu}) - \boldsymbol{x}_{ref}(\boldsymbol{\mu})\right) \|_2^2} \Big/ \sqrt{\sum_{n=1}^{N_t} \|\boldsymbol{x}(t^n;\boldsymbol{\mu})\|_2^2} \tag{6.1}$$

for linear subspace projection and

$$\text{projection error} = \sqrt{\sum_{n=1}^{N_t} \| \left(\boldsymbol{x}(t^n;\boldsymbol{\mu}) - \boldsymbol{x}_{ref}(\boldsymbol{\mu})\right) - \boldsymbol{g} \circ \boldsymbol{h}\left(\boldsymbol{x}(t^n;\boldsymbol{\mu}) - \boldsymbol{x}_{ref}(\boldsymbol{\mu})\right) \|_2^2} \Big/ \sqrt{\sum_{n=1}^{N_t} \|\boldsymbol{x}(t^n;\boldsymbol{\mu})\|_2^2} \tag{6.2}$$

for nonlinear manifold projection, where $\boldsymbol{\Phi}$ denotes a POD basis matrix, and the scaled decoder $\boldsymbol{g}$ and the scaled encoder $\boldsymbol{h}$ are a nonlinear manifold and its approximate inverse function that are obtained from an autoencoder, respectively. The computational cost is measured in terms of the CPU wall time. Specifically, timing is obtained by performing calculations on an Intel(R) Xeon(R) CPU E5-2637 v3 @ 3.50 GHz and DDR4 Memory @ 1866 MT/s. The autoencoders are trained on a NVIDIA Quadro M6000 GPU with 3072 NVIDIA CUDA Cores and 12 GB GDDR5 GPU Memory using PyTorch [55] which is the open source machine learning frame work.

6.1. 1D inviscid Burgers' equation

We consider a parameterized 1D inviscid Burgers' equation

$$\begin{aligned} &\frac{\partial u(x,t;\mu)}{\partial t} + u(x,t;\mu)\frac{\partial u(x,t;\mu)}{\partial x} = 0, \\ &\qquad x \in \Omega = [0,2] \\ &\qquad t \in [0,T], \end{aligned} \tag{6.3}$$

where $u \in \mathbb{R}$ denotes a scalar-valued time dependent state variable with the periodic boundary condition

$$u(2,t;\mu) = u(0,t;\mu)$$

and the initial condition

$$u(x,0;\mu) = \begin{cases} 1 + \frac{\mu}{2}\left(\sin\left(2\pi x - \frac{\pi}{2}\right) + 1\right) & \text{if } 0 \le x \le 1 \\ 1 & \text{otherwise} \end{cases}$$

where $\mu \in \mathcal{D} = [0.9, 1.1]$ is a parameter. Discretizing the space domain Ω into $n_x - 1$ uniform mesh gives n_x grid points $x_i = (i-1)\Delta x$ where $i \in \{1, 2, \cdots, n_x\}$ and $\Delta x = \frac{2}{n_x - 1}$. We denote the discrete solutions on grid points as $u_i(t;\mu) = u(x_i, t;\mu)$, where $i \in \mathbb{N}(n_x)$. Then, the backward difference scheme $\frac{\partial u}{\partial x} \approx \frac{u_i - u_{i-1}}{\Delta x}$ yields the semi-discretized equation which is written by

$$\frac{d\boldsymbol{U}}{dt} = \boldsymbol{f}(\boldsymbol{U})$$

where $\boldsymbol{U} = (u_1, u_2, \cdots, u_{n_x-1})^T$ and $\boldsymbol{f} : \mathbb{R}^{n_x-1} \to \mathbb{R}^{n_x-1}$ is in the form

$$\boldsymbol{f}(\boldsymbol{U}) = -\frac{1}{\Delta x}\left(\boldsymbol{M}\boldsymbol{U} \odot \boldsymbol{U} + \boldsymbol{B}\boldsymbol{U}\right)$$

where

$$\boldsymbol{M} = \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}_{(n_x-1)\times(n_x-1)}, \quad \boldsymbol{B} = \begin{bmatrix} u_{n_x-1} & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix}_{(n_x-1)\times(n_x-1)},$$

with $\odot$ denoting element-wise multiplication.

For a time integrator, we use the backward Euler scheme with time step size $\Delta t = \frac{T}{n_t}$, where T is final time and n_t is the number of time steps. We set $T = 0.5$, $n_x = 1001$, and $n_t = 500$.

For the training process, we collect solution snapshots associated with the parameter $\mu \in \mathcal{D}_{train} = \{0.9, 1.1\}$ such that $n_{\text{train}} = 2$ at which the FOM is solved. Then, the number of train data points is $n_{\text{train}} \cdot (n_t + 1) = 1002$ and 10% of the train data are used for validation purpose. We employ the Adam optimizer [37] for SGD with initial learning rate 0.001 which decreases by a factor of 10 when a training loss stagnates for 10 successive training epochs. We set the number of nodes in the hidden layer of the encoder, $M_1 = 2000$, and the number of nodes in the hidden layer of the decoder, $M_2 = 12024$. The weights and bias of the autoencoder are initialized via Kaiming initialization [32]. The size of the batch is 20 and the

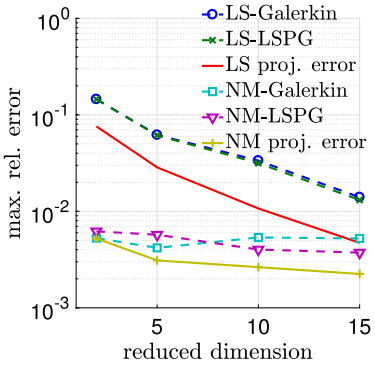


Fig. 8. 1D Burgers' equation. Relative errors vs reduced dimensions.

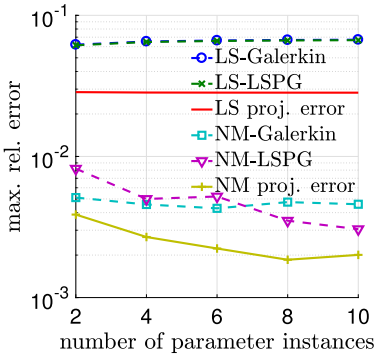


Fig. 9. 1D Burgers' equation. Relative errors vs the number of parameter instances. We use $\mathcal{D}_{train} = \{0.9, 1.1\}$ for $n_{\text{train}} = 2$, $\mathcal{D}_{train} = \{0.9, 0.95, 1.05, 1, 1\}$ for $n_{\text{train}} = 4$, $\mathcal{D}_{train} = \{0.9, 0.9 + \frac{1}{30}, 0.9 + \frac{2}{30}, 1 + \frac{1}{30}, 1 + \frac{2}{30}, 1.1\}$ for $n_{\text{train}} = 6$, $\mathcal{D}_{train} = \{0.9, 0.925, 0.95, 0.975, 1.025, 1.05, 1.075, 1.1\}$ for $n_{\text{train}} = 8$, and $\mathcal{D}_{train} = \{0.9, 0.92, 0.94, 0.96, 0.98, 1.02, 1.04, 1.06, 1.08, 1.1\}$ for $n_{\text{train}} = 10$.

maximum number of epochs is 10, 000. The training process is stopped if the loss on the validation dataset stagnates for 200 epochs.

After the training is done, the NM-ROMs and LS-ROMs solve the Eq. (6.3) with the target parameter $\mu = 1$ which is not included in the train dataset for training the autoencoder and the linear subspace. Fig. 8 shows the relative error versus the reduced dimension n_s. It also shows the projection errors for LS-ROMs and NM-ROMs, which are defined in (6.1) and (6.2). These are the lower bounds for LS-ROMs and NM-ROMs, respectively. As expected the relative errors for the NM-ROMs are lower than the ones for the LS-ROMs. We even observe that the relative errors of NM-ROMs are even lower than the lower bounds of LS-ROMs.

To see the trends regarding the number of training parameter instances, we increase the number of parameters starting from $n_{\text{train}} = 2$ with the fixed reduced dimension $n_s = 5$ to achieve less than 1% maximum relative error for NM-ROMs. In Fig. 9, we observe that $n_{\text{train}} = 2$ is enough.

LS-ROMs with $n_s = 5$ are able to achieve speed-up, but their accuracies are not as good as NM-ROMs. For example, LS-ROMs are about 5 to 6 times faster than FOM on average and have a maximum relative error of 6%. NM-ROMs solve the problem with less than the maximum relative error of 1%. For LS-ROMs, a hyper-reduction improves speed-up (e.g., 9 to 10 times faster than FOM) but accuracy doesn't get better. On the other hand, NM-ROMs without a hyper-reduction do not achieve any speed-up with respect to the corresponding FOM simulation. For example, the FOM simulation takes 1.30 seconds, while the NM-Galerkin and NM-LSPG with $n_s = 5$ takes 1.67 and 1.35 seconds, respectively. Therefore, the hyper-reduction is essential to achieve a speed-up with a reasonable accuracy for the NM-ROMs. Now, we compute the maximum relative error and wall-clock time for the hyper-reduced ROMs, i.e., NM-LSPG-HR and LS-LSPG-HR, by varying the number of residual basis and residual samples with the fixed number of training parameter instances $n_{\text{train}} = 2$ and the reduced dimension $n_s = 5$ and show the results in Table 1. Although the LS-LSPG-HR can achieve a better speed-up than the NM-LSPG-HR, the relative error of the LS-LSPG-HR is too large, e.g., the relative errors of around 6%. On the other hand, the NM-LSPG-HR achieves much better accuracy, i.e., a relative error of around 1%, than the LS-LSPG-HR with a speedup of higher than 2.

Fig. 10 shows solutions in both space and time domain of FOM, NM-LSPG-HR, and LS-LSPG-HR with the reduced dimension being $n_s = 5$. For NM-LSPG-HR, 31 residual basis and 47 residual samples are used and for LS-LSPG-HR, 30 residual basis and 47 residual samples are used. In fact, the NM-LSPG-HR is able to achieve an accuracy as good as the NM-LSPG for some combinations of the small number of residual basis and residual samples.

Table 1
The top 6 maximum relative errors and wall-clock times at different numbers of residual basis and samples which range from 30 to 50.

	NM-LSPG-HR						LS-LSPG-HR					
Residual basis	31	33	36	32	40	32	30	30	30	31	41	41
Residual samples	47	49	40	47	42	46	47	48	49	49	49	48
Max. rel. error (%)	1.03	1.07	1.18	1.23	1.23	1.25	6.07	6.08	6.08	6.09	6.11	6.11
Wall-clock time (sec)	0.63	0.51	0.49	0.50	0.51	0.50	0.14	0.13	0.13	0.23	0.14	0.13
Speed-up	2.07	2.53	2.63	2.62	2.56	2.62	9.29	9.80	9.71	5.65	9.63	9.82

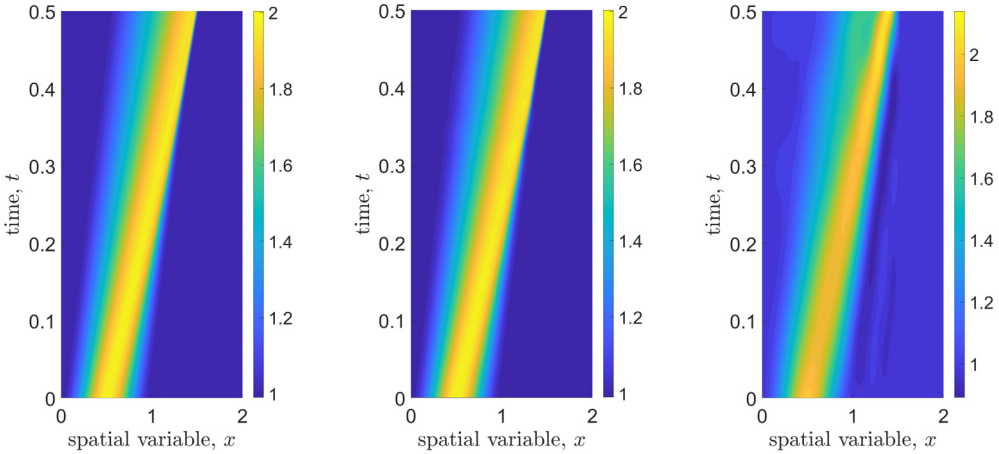


Fig. 10. A space–time solution instances of FOM and ROMs for 1D Burgers' equation.

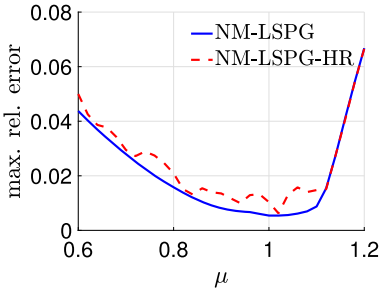


Fig. 11. The comparison of the NM-LSPG-HR and NM-LSPG on the maximum relative error vs μ.

We look into the numerical tests to see the generalization capability of the NM-LSPG and NM-LSPG-HR, i.e., the robustness of the NM-LSPG and NM-LSPG-HR outside of the trained domain. The training sample point set, $\mu \in \mathcal{D}_{train} = \{0.9, 1.1\}$, is used to train a NM-LSPG-HR. Then the trained NM-LSPG-HR model is used to predict the following parameter points, $\mu \in \mathcal{D}_{test} = \{\mu | \mu = 0.6 + 0.02i, i = 0, 1, \cdots, 30\}$. The residual basis dimension and the number of residual samples for each test case are given in Table 2. Fig. 11 shows the maximum relative error over the test range of the parameter points. Note that the NM-LSPG and NM-LSPG-HR are the most accurate within the range of the training points, i.e., $[0.9, 1.1]$. As the parameter points go beyond the training parameter domain, the accuracy of the NM-LSPG and NM-LSPG-HR start to deteriorate gradually. This implies that the NM-LSPG and NM-LSPG-HR have a trust region. Its trust region should be determined by an application. For example, if the application is okay with the maximum relative error of 10%, then the trust region of this particular NM-LSPG-HR is $[0.6, 1.2]$. However, if the application requires a higher accuracy, e.g., the maximum relative error of 2%, then the trust region of the NM-LSPG-HR is $[0.82, 1.12]$. Note that the average speed-up of the NM-LSPG-HR for all the test cases is 2.72 (see Table 2).

6.2. 2D Burgers' equation

We now consider a parameterized 2D viscous Burgers' equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{6.4}$$

Table 2
The residual basis dimension and the number of residual samples for each test parameter μ. The wall-clock time and the speed-up of the NM-LSPG-HR with respect to the corresponding FOM are also reported.

μ	Residual basis	Residual samples	Wall-clock time (sec)	Speed-up
0.60	46	48	0.54	2.41
0.62	37	39	0.46	2.83
0.64	37	39	0.47	2.77
0.66	44	46	0.52	2.50
0.68	42	44	0.48	2.71
0.70	42	44	0.48	2.71
0.72	42	44	0.48	2.71
0.74	42	44	0.48	2.71
0.76	43	45	0.53	2.45
0.78	36	45	0.46	2.83
0.80	38	47	0.47	2.77
0.82	38	47	0.47	2.77
0.84	38	47	0.47	2.77
0.86	38	47	0.47	2.77
0.88	37	46	0.46	2.83
0.90	33	33	0.45	2.89
0.92	33	33	0.46	2.83
0.94	33	33	0.46	2.83
0.96	33	33	0.45	2.89
0.98	31	47	0.45	2.89
1.00	31	47	0.45	2.89
1.02	33	49	0.48	2.71
1.04	31	48	0.46	2.83
1.06	30	48	0.46	2.83
1.08	33	39	0.48	2.71
1.10	33	40	0.48	2.71
1.12	33	42	0.48	2.71
1.14	44	49	0.54	2.41
1.16	31	48	0.48	2.71
1.18	31	48	0.47	2.77
1.20	44	48	0.57	2.28

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$
$$(x, y) \in \Omega = [0, 1] \times [0, 1]$$
$$t \in [0, 2],$$

with the boundary condition

$$u(x, y, t; \mu) = v(x, y, t; \mu) = 0 \quad \text{on} \quad \Gamma = \{(x, y) | x \in \{0, 1\}, y \in \{0, 1\}\}$$

and the initial condition

$$u(x, y, 0; \mu) = \begin{cases} \mu \sin(2\pi x) \cdot \sin(2\pi y) & \text{if } (x, y) \in [0, 0.5] \times [0, 0.5] \\ 0 & \text{otherwise} \end{cases}$$
$$v(x, y, 0; \mu) = \begin{cases} \mu \sin(2\pi x) \cdot \sin(2\pi y) & \text{if } (x, y) \in [0, 0.5] \times [0, 0.5] \\ 0 & \text{otherwise} \end{cases}$$

where $\mu \in \mathcal{D} = [0.9, 1.1]$ is a parameter and $u(x, y, t; \mu)$ and $v(x, y, t; \mu)$ denote the x and y directional velocities, respectively, with $u : \Omega \times [0, 2] \times \mathcal{D} \to \mathbb{R}$ and $v : \Omega \times [0, 2] \times \mathcal{D} \to \mathbb{R}$ defined as the solutions to Eq. (6.4), and Re is a Reynolds number which is set $Re = 10000$. In the case of $Re = 10000$ (an advection-dominated case), the FOM solution snapshot shows slowly decaying singular values compared to the case of $Re = 100$ as shown in Fig. 12 and we observe that a sharp gradient, i.e., a shock, appears in Figs. 15(a) and 15(d).

Discretizing the space domain Ω into $n_x - 1$ and $n_y - 1$ uniform meshes in x and y directions, respectively, gives $n_x \times n_y$ grid points (x_i, y_j). x_i is given by $x_i = (i-1)\Delta x$ where $i \in \{1, 2, \cdots, n_x\}$ and $\Delta x = \frac{1}{n_x - 1}$ and y_j is given by $y_j = (j-1)\Delta y$ where $j \in \{1, 2, \cdots, n_y\}$ and $\Delta y = \frac{1}{n_y - 1}$. We denote the discrete solutions on grid points as $u_{i,j}(t; \mu) = u(x_i, y_j, t; \mu)$ and

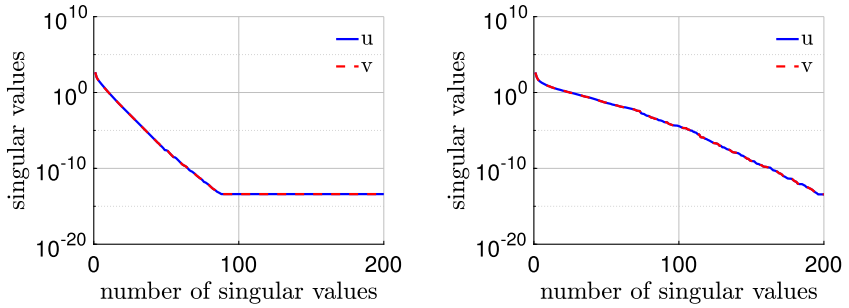


Fig. 12. The effect of Reynolds number on the singular value decay.

$v_{i,j}(t;\mu) = v(x_i, y_j, t;\mu)$, where $i \in \mathbb{N}(n_x)$ and $j \in \mathbb{N}(n_y)$. Let $n_{xy} = (n_x - 2) \times (n_y - 2)$. Then, the backward difference scheme for the first spatial derivative terms

$$\frac{\partial(\cdot)}{\partial x} \approx \frac{(\cdot)_{i,j} - (\cdot)_{i-1,j}}{\Delta x},$$
$$\frac{\partial(\cdot)}{\partial y} \approx \frac{(\cdot)_{i,j} - (\cdot)_{i,j-1}}{\Delta y}$$

and the central difference scheme for the second spatial derivative terms

$$\frac{\partial^2(\cdot)}{\partial x^2} \approx \frac{(\cdot)_{i+1,j} - 2(\cdot)_{i,j} + (\cdot)_{i-1,j}}{\Delta x^2},$$
$$\frac{\partial^2(\cdot)}{\partial y^2} \approx \frac{(\cdot)_{i,j+1} - 2(\cdot)_{i,j} + (\cdot)_{i,j-1}}{\Delta y^2}$$

yield the semi-discretized equation which is written by

$$\frac{d\boldsymbol{U}}{dt} = \boldsymbol{f}_u(\boldsymbol{U}, \boldsymbol{V}),$$
$$\frac{d\boldsymbol{V}}{dt} = \boldsymbol{f}_v(\boldsymbol{U}, \boldsymbol{V})$$

where $\boldsymbol{U} = (u_{2,2}, u_{3,2}, \cdots, u_{n_x-2,2}, u_{2,3}, u_{3,3}, \cdots, u_{n_x-2,3}, \cdots u_{2,n_y-2}, u_{3,n_y-2}, \cdots, u_{n_x-2,n_y-2})^T$ and $\boldsymbol{V} = (v_{2,2}, v_{3,2}, \cdots, v_{n_x-2,2}, v_{2,3}, v_{3,3}, \cdots, v_{n_x-2,3}, \cdots v_{2,n_y-2}, v_{3,n_y-2}, \cdots, v_{n_x-2,n_y-2})^T$ with superscript T standing for the transpose operation and $\boldsymbol{f}_u : \mathbb{R}^{n_{xy}} \times \mathbb{R}^{n_{xy}} \to \mathbb{R}^{n_{xy}}$ and $\boldsymbol{f}_v : \mathbb{R}^{n_{xy}} \times \mathbb{R}^{n_{xy}} \to \mathbb{R}^{n_{xy}}$ are in the form

$$\begin{aligned}\boldsymbol{f}_u(\boldsymbol{U}, \boldsymbol{V}) = &-\frac{1}{\Delta x}\boldsymbol{U} \odot (\boldsymbol{M}\boldsymbol{U} - \boldsymbol{b}_{ux1}) - \frac{1}{\Delta y}\boldsymbol{V} \odot \left(\boldsymbol{N}\boldsymbol{U} - \boldsymbol{b}_{uy1}\right) \\ &+ \frac{1}{Re\Delta x^2}\left(\boldsymbol{D}_x\boldsymbol{U} + \boldsymbol{b}_{ux2}\right) + \frac{1}{Re\Delta y^2}\left(\boldsymbol{D}_y\boldsymbol{U} + \boldsymbol{b}_{uy2}\right) \\ \boldsymbol{f}_v(\boldsymbol{U}, \boldsymbol{V}) = &-\frac{1}{\Delta x}\boldsymbol{U} \odot (\boldsymbol{M}\boldsymbol{V} - \boldsymbol{b}_{vx1}) - \frac{1}{\Delta y}\boldsymbol{V} \odot \left(\boldsymbol{N}\boldsymbol{V} - \boldsymbol{b}_{vy1}\right) \\ &+ \frac{1}{Re\Delta x^2}\left(\boldsymbol{D}_x\boldsymbol{V} + \boldsymbol{b}_{vx2}\right) + \frac{1}{Re\Delta y^2}\left(\boldsymbol{D}_y\boldsymbol{V} + \boldsymbol{b}_{vy2}\right)\end{aligned}$$

where

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_b & & \\ & \ddots & \\ & & \boldsymbol{M}_b \end{bmatrix}_{n_{xy}\times n_{xy}}, \quad \boldsymbol{M}_b = \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}_{(n_x-2)\times(n_x-2)},$$

$$\boldsymbol{N} = \begin{bmatrix} \boldsymbol{N}_b & & & \\ -\boldsymbol{N}_b & \boldsymbol{N}_b & & \\ & \ddots & \ddots & \\ & & -\boldsymbol{N}_b & \boldsymbol{N}_b \end{bmatrix}_{n_{xy}\times n_{xy}}, \quad \boldsymbol{N}_b = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}_{(n_x-2)\times(n_x-2)},$$

$$\boldsymbol{b}_{ux1} = \left(\left(u_{1,2}, u_{1,3}, \cdots, u_{1,n_y-1} \right)_{1\times(n_y-2)} \otimes (1, 0, \cdots, 0)_{1\times(n_x-2)} \right)^T,$$

$$\boldsymbol{b}_{uy1} = \left((1, 0, \cdots, 0)_{1\times(n_y-2)} \otimes \left(u_{2,1}, u_{3,1}, \cdots, u_{n_x-1,1} \right)_{1\times(n_x-2)} \right)^T,$$

$$\boldsymbol{b}_{vx1} = \left(\left(v_{1,2}, v_{1,3}, \cdots, v_{1,n_y-1} \right)_{1\times(n_y-2)} \otimes (1, 0, \cdots, 0)_{1\times(n_x-2)} \right)^T,$$

$$\boldsymbol{b}_{vy1} = \left((1, 0, \cdots, 0)_{1\times(n_y-2)} \otimes \left(v_{2,1}, v_{3,1}, \cdots, v_{n_x-1,1} \right)_{1\times(n_x-2)} \right)^T,$$

$$\boldsymbol{D}_x = \begin{bmatrix} \boldsymbol{D}_{xb} & & \\ & \ddots & \\ & & \boldsymbol{D}_{xb} \end{bmatrix}_{n_{xy}\times n_{xy}}, \quad \boldsymbol{D}_{xb} = \begin{bmatrix} -2 & 1 & \\ 1 & \ddots & 1 \\ & 1 & -2 \end{bmatrix}_{(n_x-2)\times(n_x-2)},$$

$$\boldsymbol{D}_y = \begin{bmatrix} -2\boldsymbol{D}_{yb} & \boldsymbol{D}_{yb} & \\ \boldsymbol{D}_{yb} & \ddots & \boldsymbol{D}_{yb} \\ & \boldsymbol{D}_{yb} & -2\boldsymbol{D}_{yb} \end{bmatrix}_{n_{xy}\times n_{xy}}, \quad \boldsymbol{D}_{yb} = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}_{(n_x-2)\times(n_x-2)},$$

$$\boldsymbol{b}_{ux2} = (u_{1,2}, 0, \cdots, 0, u_{n_x,2}, u_{1,3}, 0, \cdots, 0, u_{n_x,3}, \cdots, u_{1,n_y-1}, 0, \cdots, 0, u_{n_x,n_y-1})^T,$$

$$\boldsymbol{b}_{uy2} = (u_{2,1}, u_{3,1}, \cdots, u_{n_x-1,1}, 0, \cdots, 0, u_{2,n_y}, u_{3,n_y}, \cdots, u_{n_x-1,n_y})^T,$$

$$\boldsymbol{b}_{vx2} = (v_{1,2}, 0, \cdots, 0, v_{n_x,2}, v_{1,3}, 0, \cdots, 0, v_{n_x,3}, \cdots, v_{1,n_y-1}, 0, \cdots, 0, v_{n_x,n_y-1})^T,$$

$$\boldsymbol{b}_{vy2} = (v_{2,1}, v_{3,1}, \cdots, v_{n_x-1,1}, 0, \cdots, 0, v_{2,n_y}, v_{3,n_y}, \cdots, v_{n_x-1,n_y})^T$$

with $\odot$ and $\otimes$ denoting the element-wise multiplication and the Kronecker product, respectively.

For a time integrator, we use the backward Euler scheme with time step size $\Delta t = \frac{2}{n_t}$, where n_t is the number of time steps. We set $n_x = 60$, $n_y = 60$, and $n_t = 1500$.

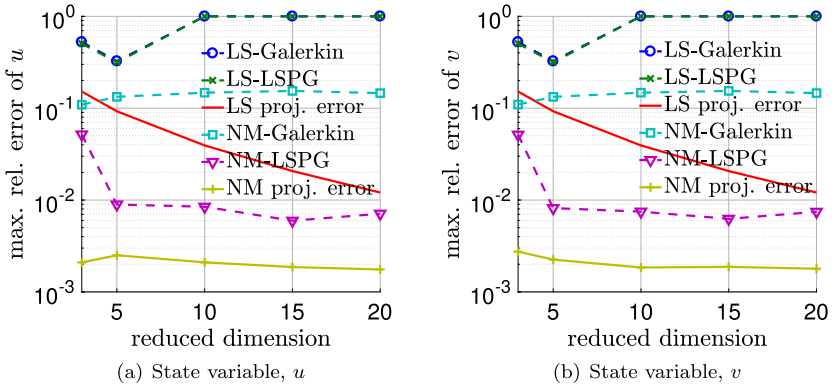
For the training process, we collect solution snapshots associated with the parameter $\mu \in \mathcal{D}_{train} = \{0.9, 0.95, 1.05, 1.1\}$ such that $n_{\text{train}} = 4$ at which the FOM is solved. Then, the number of train data points is $n_{\text{train}} \cdot (n_t + 1) = 6004$ and 10% of the train data are used for validation purpose. We employ the Adam optimizer [37] with the SGD and the initial learning rate of 0.001, which decreases by a factor of 10 when a training loss stagnates for 10 successive training epochs. Here, we have two autoencoders. One for $\boldsymbol{U}$ vector and the other for $\boldsymbol{V}$ vector. The reason why we have such two autoencoders is that we can use less memory for training process compared to one autoencoder for $(\boldsymbol{U}^T, \boldsymbol{V}^T)^T$ vector and train both of them at the same time. We set the number of nodes in hidden layer in the encoder, $M_1 = 6728$, and the number of nodes in hidden layer in the decoder, $M_2 = 33730$. The weights and bias of the autoencoder are initialized via Kaiming initialization [32] as in the first numerical example. The size of the batch is 240 and the maximum number of epochs is 10,000. The training process is stopped if the loss on the validation dataset stagnates for 200 epochs.

After the training is done, the NM-ROMs and LS-ROMs solve the Eq. (6.4) with the target parameter $\mu = 1$, which is not included in the train dataset for training the autoencoder and the linear subspace. Fig. 13 shows the relative error versus the reduced dimension n_s for both NM-ROMs and LS-ROMs. It also shows the projection errors for LS-ROMs and NM-ROMs, which are defined in (6.1) and (6.2). These are the lower bounds for LS-ROMs and NM-ROMs, respectively. As expected the relative errors for the NM-ROMs are lower than the ones for the LS-ROMs. We even observe that the relative errors of NM-LSPG are even lower than the lower bounds of LS-ROMs. One notable observation is that NM-Galerkin is not able to achieve a good accuracy, while the NM-LSPG does. Another observation is that LS-ROMs struggle more for this problem than the 1D invisid Burgers' equations, e.g., some LS-ROMs fail to converge.

To see the trends regarding the number of training parameter instances, we increase the number of parameters starting from $n_{\text{train}} = 2$ with the fixed reduced dimension $n_s = 5$ to achieve less than 1% maximum relative error for NM-ROMs. In Fig. 14, we observe that $n_{\text{train}} = 4$ is enough.

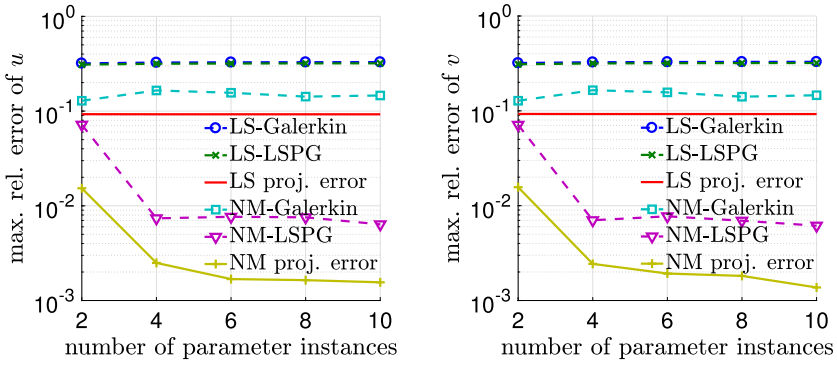
Both NM-Galerkin and LS-ROMs without a hyper-reduction do not achieve any speed-up with respect to the corresponding FOM simulation. For example, the FOM simulation takes 140.67 seconds, while the NM-Galerkin, NM-LSPG, LS-Galerkin, and LS-LSPG with $n_s = 5$ takes 143.41, 78.22, 519.12 and 2193.70 seconds, respectively. Although NM-LSPG is able to achieve a speed-up, it is not considerable. Therefore, the hyper-reduction is essential to achieve a considerable speed-up with a reasonable accuracy.

We compute the maximum relative error by choosing the larger of the two errors between the maximum relative error of u and the maximum relative error of v. We vary the number of residual basis and residual samples, with the fixed number of training parameter instances $n_{\text{train}} = 4$ and the reduced dimension $n_s = 5$, and measure the wall-clock time. The results are shown in Table 3. Although the LS-LSPG-HR can achieve better speedup than the NM-LSPG-HR, the relative error of the LS-LSPG-HR is too large to be reasonable, e.g., the relative errors of around 37%. On the other hand, the NM-LSPG-HR achieves much better accuracy, i.e., a relative error of around 1%, than the LS-LSPG-HR with a good speedup, i.e., a speedup of higher than 11.



(a) State variable, u (b) State variable, v

Fig. 13. 2D Burgers' equation. Relative errors vs reduced dimensions. A maximum relative error that is 1 means the ROM failed to solve the problem.



(a) State variable, u (b) State variable, v

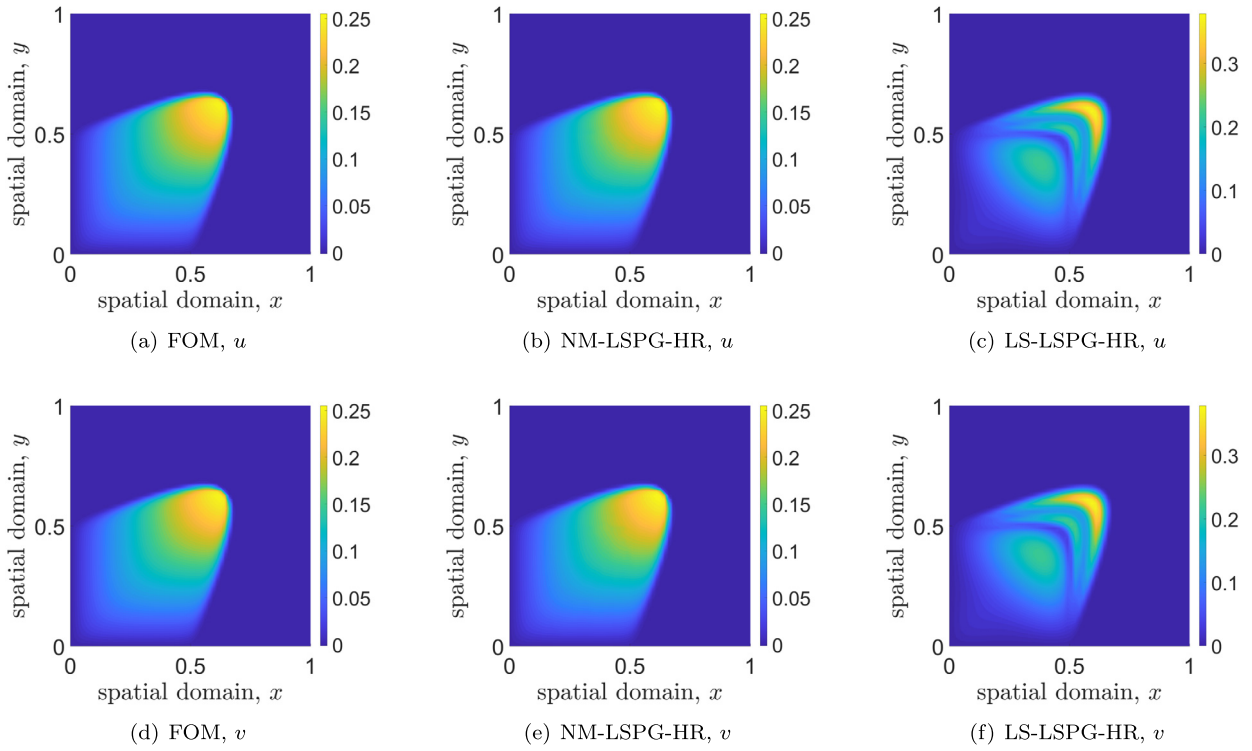
Fig. 14. 2D Burgers' equation. Relative errors vs the number of parameter instances. The following parameter sets are used: $\mathcal{D}_{train} = \{0.9, 1.1\}$ for $n_{\text{train}} = 2$, $\mathcal{D}_{train} = \{0.9, 0.95, 1.05, 1.\}$ for $n_{\text{train}} = 4$, $\mathcal{D}_{train} = \{0.9, 0.9 + \frac{1}{30}, 0.9 + \frac{2}{30}, 1 + \frac{1}{30}, 1 + \frac{2}{30}, 1.1\}$ for $n_{\text{train}} = 6$, $\mathcal{D}_{train} = \{0.9, 0.925, 0.95, 0.975, 1.025, 1.05, 1.075, 1.1\}$ for $n_{\text{train}} = 8$, and $\mathcal{D}_{train} = \{0.9, 0.92, 0.94, 0.96, 0.98, 1.02, 1.04, 1.06, 1.08, 1.1\}$ for $n_{\text{train}} = 10$.

Table 3
The top 6 maximum relative errors and wall-clock times at different numbers of residual basis and samples which range from 40 to 60.

	NM-LSPG-HR						LS-LSPG-HR					
Residual basis	55	56	51	53	54	44	59	53	53	53	53	53
Residual samples	58	59	54	56	57	47	59	58	59	56	55	53
Max. rel. error (%)	0.93	0.94	0.95	0.97	0.97	0.98	34.38	37.73	37.84	37.95	37.96	37.97
Wall-clock time (sec)	12.15	12.35	12.09	12.14	12.29	12.01	5.26	5.02	4.86	5.05	4.75	7.18
Speed-up	11.58	11.39	11.63	11.58	11.44	11.71	26.76	28.02	28.95	27.83	29.61	19.58

Fig. 15 shows solutions at the last time step of FOM, NM-LSPG-HR, and LS-LSPG-HR with the reduced dimension being $n_s = 5$. For NM-LSPG-HR, 55 residual basis dimension and 58 residual samples are used and for LS-LSPG-HR, 59 residual basis dimension and 59 residual samples are used. Both FOM and NM-LSPG-HR show good agreement in their solutions, while the LS-LSPG-HR is not able to achieve a good accuracy. In fact, the NM-LSPG-HR is able to achieve an accuracy as good as the NM-LSPG for some combinations of the small number of residual basis and residual samples as in Section 6.1.

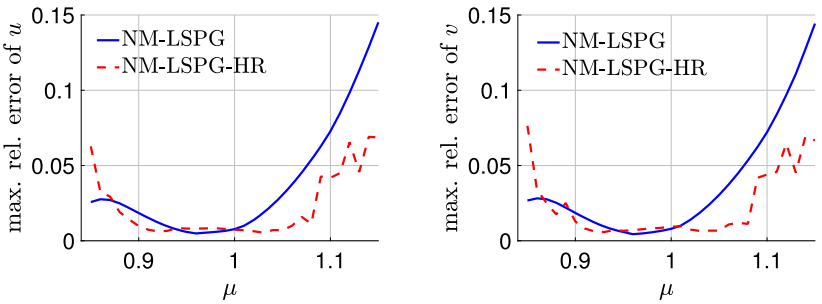
We look into the numerical tests to see the generalization capability of the NM-LSPG and NM-LSPG-HR, i.e., the robustness of the NM-LSPG and NM-LSPG-HR outside of the trained domain. The training sample point set, $\mu \in \mathcal{D}_{train} = \{0.9, 0.95, 1.05, 1.1\}$, is used to train a NM-LSPG-HR. Then the trained NM-LSPG-HR model is used to predict the following parameter points, $\mu \in \mathcal{D}_{test} = \{\mu | \mu = 0.85 + 0.01i, i = 0, 1, \cdots, 30\}$. The residual basis dimension and the number of residual samples for each test case are given in Table 4. Fig. 16 shows the maximum relative error over the test range of the parameter points. Note that the NM-LSPG and NM-LSPG-HR are the most accurate within the range of the training points, i.e., $[0.9, 1.1]$. As the parameter points go beyond the training parameter domain, the accuracy of the NM-LSPG and NM-LSPG-HR start to deteriorate gradually. This implies that the NM-LSPG and NM-LSPG-HR have a trust region. Its trust region should be determined by an application. For example, if the application is okay with the maximum relative error of 10%, then the trust region of this particular NM-LSPG-HR is $[0.85, 1.15]$. However, if the application requires a higher



(a) FOM, u (b) NM-LSPG-HR, u (c) LS-LSPG-HR, u

(d) FOM, v (e) NM-LSPG-HR, v (f) LS-LSPG-HR, v

Fig. 15. Solution snapshots of FOM, NM-LSPG-HR, and LS-LSPG-HR at $t = 2$.



(a) State variable, u (b) State variable, v

Fig. 16. The comparison of the NM-LSPG-HR and NM-LSPG on the maximum relative error vs μ.

accuracy, e.g., the maximum relative error of 2%, then the trust region of the NM-LSPG-HR is $[0.87, 1.08]$. Note that the average speed-up of the NM-LSPG-HR for all the test cases is 10.61 (see Table 4).

7. Discussion & conclusion

In this work, we have successfully developed an accurate and efficient nonlinear manifold based reduced order model. We demonstrated that the linear subspace based reduced order model is not able to represent advection-dominated or sharp gradient solutions of 1D inviscid Burgers' equation and 2D viscous Burgers' equation with a high Reynolds number. However, our new approach, NM-LSPG-HR, solves such problems accurately and efficiently. For 1D case, the NM-LSPG-HR method achieves a good accuracy i.e., the maximum relative error of around 1% with the speed-up of higher than 2. For 2D case, the NM-LSPG-HR method was able to achieve even better accuracy, i.e., the maximum relative error of less than 1%, with even better speed-up of around 12 than the 1D case. We also presented *a posteriori* error bounds for NM-Galerkin-HR and NM-LSPG-HR. The speed-up of the NM-LSPG-HR is achieved by choosing the sparse shallow decoder as the nonlinear manifold and applying the efficient hyper-reduction computation, which can be done by constructing a subnet. Furthermore, we expect more speed-up as the FOM size increases because the difference in the computational cost between the FOM and NM-LSPG-HR increases as shown in Fig. 17.

Table 4
The residual basis dimension and the number of residual samples for each test parameter μ. The wall-clock time and the speed-up of the NM-LSPG-HR with respect to the corresponding FOM are also reported.

μ	Residual basis	Residual samples	Wall-clock time (sec)	Speed-up
0.85	47	59	13.65	10.31
0.86	50	50	13.19	10.66
0.87	45	45	12.61	11.16
0.88	49	50	12.69	11.08
0.89	52	52	13.41	10.49
0.90	53	57	13.35	10.54
0.91	59	59	13.60	10.34
0.92	55	58	13.41	10.49
0.93	51	54	13.17	10.68
0.94	54	57	13.32	10.56
0.95	55	58	13.52	10.40
0.96	55	58	13.54	10.39
0.97	54	57	13.39	10.51
0.98	52	55	13.20	10.66
0.99	52	55	13.18	10.67
1.00	55	58	13.38	10.51
1.01	46	49	12.80	10.99
1.02	50	53	13.35	10.54
1.03	50	53	13.40	10.50
1.04	52	53	13.40	10.50
1.05	46	58	13.21	10.65
1.06	54	57	13.58	10.36
1.07	45	57	13.20	10.66
1.08	45	57	13.23	10.63
1.09	43	55	13.27	10.60
1.10	44	48	13.31	10.57
1.11	40	43	12.79	11.00
1.12	48	59	13.66	10.30
1.13	42	51	13.25	10.62
1.14	46	49	13.10	10.74
1.15	40	50	13.11	10.73

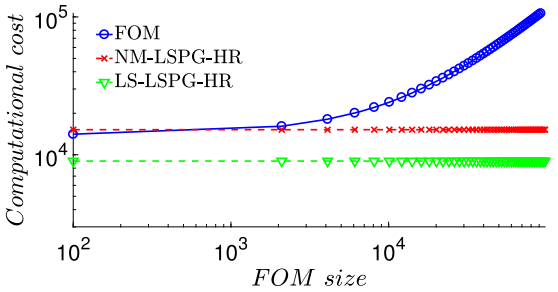


Fig. 17. Computational cost vs FOM size. The figure shows that the higher the speed up will be achieved, the larger the underlying FOM problem is. The graph is generated based on the computational cost analysis done in Appendix B.

Compared with the deep neural networks for computer vision and natural language processing applications, our neural networks are shallow with a small number of parameters. However, these networks were able to capture the variation in our 1D and 2D Burgers' simulations. A main future work for transferring this work to more complex simulations, will be to find the right balance between a shallow network that is large enough to capture the data variance and yet small enough to run faster than the FOM. Another future work will be to find an efficient way of determining the proper size of the residual basis and the number of sample points *a priori*. To find the optimal size of residual basis and the number of sample points for hyper-reduced ROMs, we relied on test results. This issue is not just for NM-LSPG-HR but also for LS-LSPG-HR.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was performed at Lawrence Livermore National Laboratory and was supported by the LDRD program (project 20-FS-007). Youngkyu was also supported for this work through generous funding from DTRA. Lawrence Livermore National

Laboratory is operated by Lawrence Livermore National Security, LLC, for the U.S. Department of Energy, National Nuclear Security Administration under Contract DE-AC52-07NA27344 and LLNL-JRNL-814844.

Appendix A. Time integrators

A.1. The linear multistep methods

Applying a linear k-step method to numerically solve Eq. (2.1) yields an OΔE characterized by the following system of nonlinear algebraic residual function that needs to be solved for the numerical solution $\boldsymbol{x}_n \in \mathbb{R}^{N_s}$ at each time instance:

$$\begin{aligned}\boldsymbol{r}^n(\boldsymbol{x}_n; \boldsymbol{x}_{n-1} \ldots, \boldsymbol{x}_{n-k(t^n)}, \boldsymbol{\mu}) &:= \sum_{j=0}^{k(t^n)} \alpha_j^n \boldsymbol{x}_{n-j} - \Delta t \sum_{j=0}^{k(t^n)} \beta_j^n \boldsymbol{f}_{n-j} \\ &= \boldsymbol{0}, \quad n \in \mathbb{N}(N_t),\end{aligned}$$

where coefficients $\alpha_j^n, \beta_j^n \in \mathbb{R}$, $j = 0, \ldots, k(t^n)$ define a particular linear multistep scheme. It is necessary for consistency to have $\alpha_0^n \neq 0$ and $\sum_{j=0}^{k(t^n)} \alpha_j^n = 0$. Here, $k(t^n)(\leq n)$ denotes the number of steps used by the linear multistep method at time instance n. The linear multistep methods include the one-step Euler methods, the implicit Adams–Moulton methods, the explicit Adams–Bashforth methods, and the Backward Differentiation Formulas (BDFs).

The second order Adams–Bashforth (AB) method numerically solves Eq. (2.1), by solving the following nonlinear system of equations for $\boldsymbol{x}_n$ at n-th time step:

$$\boldsymbol{x}_n - \boldsymbol{x}_{n-1} = \Delta t \left(\frac{3}{2} \boldsymbol{f}_{n-1} - \frac{1}{2} \boldsymbol{f}_{n-2} \right).$$

The residual function of the second AB method is defined as

$$\boldsymbol{r}^n_{\mathrm{AB}}(\boldsymbol{x}_n; \boldsymbol{x}_{n-1}, \boldsymbol{\mu}) := \boldsymbol{x}_n - \boldsymbol{x}_{n-1} - \Delta t \left(\frac{3}{2} \boldsymbol{f}_{n-1} - \frac{1}{2} \boldsymbol{f}_{n-2} \right).$$

The second order Adams–Moulton (AM) method numerically solves Eq. (2.1), by solving the following nonlinear system of equations for $\boldsymbol{x}_n$ at n-th time step:

$$\boldsymbol{x}_n - \boldsymbol{x}_{n-1} = \frac{1}{2} \Delta t (\boldsymbol{f}_n + \boldsymbol{f}_{n-1}).$$

The residual function of the second AM method is defined as

$$\boldsymbol{r}^n_{\mathrm{AM}}(\boldsymbol{x}_n; \boldsymbol{x}_{n-1}, \boldsymbol{\mu}) := \boldsymbol{x}_n - \boldsymbol{x}_{n-1} - \Delta t \frac{1}{2} (\boldsymbol{f}_n + \boldsymbol{f}_{n-1}). \tag{A.1}$$

The 2-step BDF numerically solves Eq. (2.1), by solving the following nonlinear system of equations for $\boldsymbol{x}_n$ at n-th time step:

$$\boldsymbol{x}_n - \frac{4}{3} \boldsymbol{x}_{n-1} + \frac{1}{3} \boldsymbol{x}_{n-2} = \frac{2}{3} \Delta t \boldsymbol{f}_n.$$

The residual function of the two-step BDF method is defined as

$$\boldsymbol{r}^n_{\mathrm{BDF}}(\boldsymbol{x}_n; \boldsymbol{x}_{n-1}, \boldsymbol{x}_{n-2}, \boldsymbol{\mu}) := \boldsymbol{x}_n - \frac{4}{3} \boldsymbol{x}_{n-1} + \frac{1}{3} \boldsymbol{x}_{n-2} - \frac{2}{3} \Delta t \boldsymbol{f}_n. \tag{A.2}$$

A.2. The midpoint Runge–Kutta method

The midpoint method, a 2-stage Runge–Kutta method, takes the following two stages to advance at n-th time step of Eq. (2.1):

$$\begin{aligned}\boldsymbol{x}_{n-\frac{1}{2}} &= \boldsymbol{x}_{n-1} + \frac{\Delta t}{2} \boldsymbol{f}_{n-1} \\ \boldsymbol{x}_n &= \boldsymbol{x}_{n-1} + \Delta t \boldsymbol{f}_{n-\frac{1}{2}}.\end{aligned}$$

Appendix B. Computational costs

Let's denote the size of FOM and ROM as m and f, respectively. Because of mathematical models that require local information, we need not only the indices selected from the hyper-reduction, but also their neighbors. We denote the total number of indices as z and assume $z \ll m$, $z^2 < m$ and $z > f$ (e.g., $z = 10f$). For simplicity, we assume the mask matrix for the sparse decoder has the same structure as the mask matrix for 1D Burgers equation as depicted in Section 3.2. To generate the mask matrix, two variables b and δb are used, where b denotes the number of nodes in the hidden layer to compute single output element and δb denotes the amount by which the block of b nodes shifts. Then, the number of nodes in the hidden layer can be computed as $M_2 = b + (m-1)\delta b$.

B.1. Computational costs of NM-LSPG

Since the decoder is a single hidden layer neural network, the cost of the decoder and its Jacobian evaluation is $\mathcal{O}(mb) + \mathcal{O}(M_2 f)$ and $\mathcal{O}(fM_2) + \mathcal{O}(mbf)$, respectively. Computing residual, $\tilde{r}$, includes only element-wise vector calculation, resulting in $\mathcal{O}(m)$. Jacobian of the residual, $\tilde{J}$, can be computed using row-wise multiplication of matrix and vector because of local connectivity of mathematical model (e.g., discrete 1D and 2D Burgers equation) in $\mathcal{O}(fm)$. For the Gauss–Newton method, we need to construct $\hat{r} = \tilde{J}^T\tilde{r}$ and $\hat{J} = \tilde{J}^T\tilde{J}$, which requires $\mathcal{O}(fm)$ and $\mathcal{O}(f^2m)$, respectively. It also takes $\mathcal{O}(f^2)$ to compute each update, $\delta u = -\hat{J}^{-1}\hat{r}$, iteratively. Assuming the number of Gauss–Newton iterations is in the same order for the given governing equation, we can factor out the number of iterations. Thus, the total computational costs of NM-LSPG for each time step is $\mathcal{O}(fM_2) + \mathcal{O}(mbf) + \mathcal{O}(f^2m)$. With the assumption of $M_2 \approx m\delta b$, $f < b$, and $\delta b < b$, we have $\mathcal{O}(mbf)$.

B.2. Computational costs of NM-LSPG-HR

The size of the weight matrix connecting the hidden layer and the output layer is reduced to z by βM_2, where $\beta = \frac{z}{m}$ for the best case (z successive points are selected) and $\beta = 1$ for the worst case (z uniformly separate points are selected). Note that when z is small, it is possible to have $\beta < 1$ even for the worst case. Then, replacing m with z and M_2 with βM_2 in the decoder and its Jacobian evaluation gives us $\mathcal{O}(zb) + \mathcal{O}(\beta M_2 f)$ and $\mathcal{O}(f\beta M_2) + \mathcal{O}(zbf)$, respectively. Costs of computing residual, $\tilde{r}_{HR} = Z^T\tilde{r}$ and its Jacobian, $\tilde{J}_{HR} = Z^T\tilde{J}$ for NM-LSPG-HR are $\mathcal{O}(z)$ and $\mathcal{O}(fz)$, respectively because the sampling matrix Z^T selects z elements of the residual and z rows of its Jacobian without constructing the sampling matrix. For the Gauss–Newton method, we need to construct $\hat{r} = \tilde{J}_{HR}^T\mathcal{P}\tilde{r}_{HR}$ and $\hat{J} = \tilde{J}_{HR}^T\mathcal{P}\tilde{J}_{HR}$, where $\mathcal{P}$ is the pre-computed $z \times z$ matrix, which require $\mathcal{O}(fz) + \mathcal{O}(z^2)$ and $\mathcal{O}(fz^2) + \mathcal{O}(f^2z)$, respectively. It also takes $\mathcal{O}(f^2)$ to compute each update, $\delta u = -\hat{J}^{-1}\hat{r}$, iteratively. Assuming the number of Gauss–Newton iterations is in the same order for the given governing equation, we can factor out the number of iterations. Thus, the total computational costs of NM-LSPG-HR for each time step is $\mathcal{O}(f\beta M_2) + \mathcal{O}(zbf) + \mathcal{O}(fz^2)$. With the assumption of $M_2 \approx \delta bm$, we have $\mathcal{O}(f\beta\delta bm) + \mathcal{O}(zbf) + \mathcal{O}(fz^2)$. For the best case, $\beta = \frac{z}{m}$, the computational costs is $\mathcal{O}(fz\delta b) + \mathcal{O}(zbf) + \mathcal{O}(fz^2)$. Assuming $\delta b < b$, we have $\mathcal{O}(zbf) + \mathcal{O}(fz^2)$. For the worst case, $\beta = 1$, we have $\mathcal{O}(f\delta bm) + \mathcal{O}(zbf) + \mathcal{O}(fz^2)$.

B.3. Computational costs of LS-LSPG

The decoder $\boldsymbol{g}(\hat{\boldsymbol{x}})$ and its Jacobian $J_g(\hat{\boldsymbol{x}})$ are replaced with $\boldsymbol{\Phi}\hat{\boldsymbol{x}}$ and $\boldsymbol{\Phi}$, respectively. Thus, the cost of $\boldsymbol{\Phi}\hat{\boldsymbol{x}}$ is $\mathcal{O}(mf)$ and the cost of its Jacobian evaluation is zero. The costs of computing residual and its Jacobian are the same as for NM-LSPG. Also, the costs of the Gauss–Newton method is the same as in Section B.1. Assuming the number of Gauss–Newton iterations is in the same order for the given governing equation, we can factor out the number of iterations. Thus, the total computational costs of LS-LSPG for each time step is $\mathcal{O}(f^2m)$.

B.4. Computational costs of LS-LSPG-HR

For LS-LSPG-HR, we construct reduced model with the size of basis matrix $\boldsymbol{\Phi}_{HR}$ being z by f, where $\boldsymbol{\Phi}_{HR} := \boldsymbol{Z}^T\boldsymbol{\Phi}$. Thus, the costs of $\boldsymbol{\Phi}_{HR}\hat{\boldsymbol{x}}$ is $\mathcal{O}(zf)$ and the cost of its Jacobian evaluation is zero. The costs of computing residual and its Jacobian are the same as for NM-LSPG-HR. Also, the costs of the Gauss–Newton method is the same as in Section B.2. Assuming the number of Gauss–Newton iterations is in the same order for the given governing equation, we can factor out the number of iterations. Thus, the total computational costs of LS-LSPG-HR for each time step is $\mathcal{O}(f^2z) + \mathcal{O}(fz^2)$.

References

[1] Rémi Abgrall, David Amsallem, Roxana Crisovan, Robust model reduction by l^1-norm minimization and approximation via dictionaries: application to nonlinear hyperbolic problems, Adv. Model. Simul. Eng. Sci. 3 (1) (2016) 1–16.

[2] David Amsallem, Matthew Zahr, Youngsoo Choi, Charbel Farhat, Design optimization using hyper-reduced-order models, Struct. Multidiscip. Optim. 51 (4) (2015) 919–940.

[3] Harbir Antil, Matthias Heinkenschloss, Ronald H.W. Hoppe, Christopher Linsenmann, Achim Wixforth, Reduced order modeling based shape optimization of surface acoustic wave driven microfluidic biochips, Math. Comput. Simul. 82 (10) (2012) 1986–2003.

[4] Christian Beck, E. Weinan, Arnulf Jentzen, Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations, J. Nonlinear Sci. 29 (4) (2019) 1563–1619.
[5] Peter Benner, Serkan Gugercin, Karen Willcox, A survey of projection-based model reduction methods for parametric dynamical systems, SIAM Rev. 57 (4) (2015) 483–531.
[6] Jens Berg, Kaj Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41.
[7] Gal Berkooz, Philip Holmes, John L. Lumley, The proper orthogonal decomposition in the analysis of turbulent flows, Annu. Rev. Fluid Mech. 25 (1) (1993) 539–575.
[8] Léon Bottou, Olivier Bousquet, The tradeoffs of large scale learning, in: Advances in Neural Information Processing Systems, 2008, pp. 161–168.
[9] Kevin Carlberg, Adaptive h-refinement for reduced-order models, Int. J. Numer. Methods Eng. 102 (5) (2015) 1192–1210.
[10] Kevin Carlberg, Matthew Barone, Harbir Antil, Galerkin v. least-squares Petrov–Galerkin projection in nonlinear model reduction, J. Comput. Phys. 330 (2017) 693–734.
[11] Kevin Carlberg, Charbel Bou-Mosleh, Charbel Farhat, Efficient non-linear model reduction via a least-squares Petrov–Galerkin projection and compressive tensor approximations, Int. J. Numer. Methods Eng. 86 (2) (2011) 155–181.
[12] Kevin Carlberg, Youngsoo Choi, Syuzanna Sargsyan, Conservative model reduction for finite-volume models, J. Comput. Phys. 371 (2018) 280–314.
[13] Kevin Carlberg, Charbel Farhat, Julien Cortial, David Amsallem, The gnat method for nonlinear model reduction: effective implementation and application to computational fluid dynamics and turbulent flows, J. Comput. Phys. 242 (2013) 623–647.
[14] Saifon Chaturantabut, Danny C. Sorensen, Nonlinear model reduction via discrete empirical interpolation, SIAM J. Sci. Comput. 32 (5) (2010) 2737–2764.
[15] Ricky T.Q. Chen, Yulia Rubanova, Jesse Bettencourt, David K. Duvenaud, Neural ordinary differential equations, in: Advances in Neural Information Processing Systems, 2018, pp. 6571–6583.
[16] Youngsoo Choi, Gabriele Boncoraglio, Spenser Anderson, David Amsallem, Charbel Farhat, Gradient-based constrained optimization using a database of linear reduced-order models, J. Comput. Phys. (2020) 109787.
[17] Youngsoo Choi, Peter Brown, Bill Arrighi, Robert Anderson, Kevin Huynh, Space–time reduced order model for large-scale linear dynamical systems with application to Boltzmann transport problems, J. Comput. Phys. (2020) P109845.
[18] Youngsoo Choi, Kevin Carlberg, Space–time least-squares Petrov–Galerkin projection for nonlinear model reduction, SIAM J. Sci. Comput. 41 (1) (2019) A26–A58.
[19] Youngsoo Choi, Deshawn Coombs, Robert Anderson, Sns: a solution-based nonlinear subspace method for time-dependent model order reduction, SIAM J. Sci. Comput. 42 (2) (2020) A1116–A1146.
[20] Youngsoo Choi, Geoffrey Oxberry, Daniel White, Trenton Kirchdoerfer, Accelerating design optimization using reduced order models, arXiv preprint, arXiv:1909.11320, 2019.
[21] P.G. Constantine, G. Iaccarino, Reduced order models for parameterized hyperbolic conservations laws with shock reconstruction, in: Center for Turbulence Research Annual Brief, 2012.
[22] George Cybenko, Math. Control Signals Syst. 2 (1989) 303.
[23] Gabriel Dimitriu, Ionel M. Navon, Răzvan Ştefănescu, Application of pod-deim approach for dimension reduction of a diffusive predator-prey system with Allee effect, in: International Conference on Large-Scale Scientific Computing, Springer, 2013, pp. 373–381.
[24] M.W.M.G. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (3) (1994) 195–201.
[25] Zlatko Drmac, Serkan Gugercin, A new selection operator for the discrete empirical interpolation method—improved a priori error bound and extensions, SIAM J. Sci. Comput. 38 (2) (2016) A631–A648.
[26] Zlatko Drmac, Arvind Krishna Saibaba, The discrete empirical interpolation method: canonical structure and formulation in weighted inner product spaces, SIAM J. Matrix Anal. Appl. 39 (3) (2018) 1152–1180.
[27] Richard Everson, Lawrence Sirovich, Karhunen–Loeve procedure for gappy data, JOSA A 12 (8) (1995) 1657–1664.
[28] Hongfei Fu, Hong Wang, Zhu Wang, Pod/deim reduced-order modeling of time-fractional partial differential equations with applications in parameter identification, J. Sci. Comput. 74 (1) (2018) 220–243.
[29] Mohamadreza Ghasemi, Eduardo Gildin, Localized model reduction in porous media flow, IFAC-PapersOnLine 48 (6) (2015) 242–247.
[30] Jiequn Han, Arnulf Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (34) (2018) 8505–8510.
[31] Juncai He, Lin Li, Jinchao Xu, Chunyue Zheng, Relu deep neural networks and linear finite elements, arXiv preprint, arXiv:1807.03973, 2018.
[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian 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.
[33] Michael Hinze, Stefan Volkwein, Proper orthogonal decomposition surrogate models for nonlinear dynamical systems: error estimates and suboptimal control, in: Dimension Reduction of Large-Scale Systems, Springer, 2005, pp. 261–306.
[34] Harold Hotelling, Analysis of a complex of statistical variables into principal components, J. Educ. Psychol. 24 (6) (1933) 417.
[35] Rui Jiang, Louis J. Durlofsky, Implementation and detailed assessment of a gnat reduced-order model for subsurface flow simulation, J. Comput. Phys. 379 (2019) 192–213.
[36] Yuehaw Khoo, Jianfeng Lu, Lexing Ying, Solving parametric pde problems with artificial neural networks, arXiv preprint, arXiv:1707.03351, 2017.
[37] Diederik P. Kingma, Jimmy Ba, Adam: A Method for Stochastic Optimization, 2014.
[38] Michael Kirby, Dieter Armbruster, Reconstructing phase space from pde simulations, Z. Angew. Math. Phys. 43 (6) (1992) 999–1022.
[39] Mark A. Kramer, Nonlinear principal component analysis using autoassociative neural networks, AIChE J. 37 (2) (1991) 233–243.
[40] Karl Kunisch, Stefan Volkwein, Galerkin proper orthogonal decomposition methods for a general equation in fluid dynamics, SIAM J. Numer. Anal. 40 (2) (2002) 492–515.
[41] Isaac E. Lagaris, Aristidis Likas, Dimitrios I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.
[42] Kookjin Lee, Kevin Carlberg, Deep conservation: a latent dynamics model for exact satisfaction of physical conservation laws, arXiv preprint, arXiv: 1909.09754, 2019.
[43] Kookjin Lee, Kevin T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, J. Comput. Phys. 404 (2020) 108973.
[44] Randall J. LeVeque, Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems, SIAM, 2007.
[45] L. Ljung, System Identification: Theory for the User. Prentice Hall Information and System Sciences Series, Prentice Hall PTR, 1999.
[46] Michel Loeve, Probability Theory, D. Van Nostrand, New York, 1955.
[47] Zichao Long, Yiping Lu, Xianzhong Ma, Bin Dong Pde-net, Learning pdes from data, in: International Conference on Machine Learning, 2018, pp. 3208–3216.
[48] Lu Lu, Pengzhan Jin, George Em Karniadakis, Deeponet: learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators, arXiv preprint, arXiv:1910.03193, 2019.

[49] Lu Lu, Xuhui Meng, Zhiping Mao, George E. Karniadakis, Deepxde: a deep learning library for solving differential equations, arXiv preprint, arXiv: 1907.04502, 2019.
[50] Andrew J. Meade Jr, Alvaro A. Fernandez, The numerical solution of linear ordinary differential equations by feedforward neural networks, Math. Comput. Model. 19 (12) (1994) 1–25.
[51] M. Mordhorst, Timm Strecker, D. Wirtz, Thomas Heidlauf, Oliver Röhrle, Pod-deim reduction of computational emg models, J. Comput. Sci. 19 (2017) 86–96.
[52] Guofei Pang, Lu Lu, George Em Karniadakis, fpinns: fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (4) (2019) A2603–A2626.
[53] Eric J. Parish, Kevin T. Carlberg, Windowed least-squares model reduction for dynamical systems, arXiv preprint, arXiv:1910.11388, 2019.
[54] David B. Parker, Learnins logic, Technical Report, 1985.
[55] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al., Pytorch: an imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, 2019, pp. 8026–8037.
[56] Benjamin Peherstorfer, Model reduction for transport-dominated problems via online adaptive bases and adaptive sampling, arXiv preprint, arXiv: 1812.02094, 2018.
[57] Allan Pinkus, Approximation theory of the mlp model in neural networks, Acta Numer. 8 (1) (1999) 143–195.
[58] Rajat Raina, Anand Madhavan, Andrew Y. Ng, Large-scale deep unsupervised learning using graphics processors, in: Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 873–880.
[59] Maziar Raissi, Paris Perdikaris, George E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[60] Julius Reiss, Philipp Schulze, Jörn Sesterhenn, Volker Mehrmann, The shifted proper orthogonal decomposition: a mode decomposition for multiple transport phenomena, SIAM J. Sci. Comput. 40 (3) (2018) A1322–A1344.
[61] Donsub Rim, Scott Moe, Randall J. LeVeque, Transport reversal for model reduction of hyperbolic partial differential equations, SIAM/ASA J. Uncertain. Quantificat. 6 (1) (2018) 118–150.
[62] Donsub Rim, Benjamin Peherstorfer, Kyle T. Mandli, Manifold approximations via transported subspaces: model reduction for transport-dominated problems, arXiv preprint, arXiv:1912.13024, 2019.
[63] Donsub Rim, Luca Venturi, Joan Bruna, Benjamin Peherstorfer, Depth separation for reduced deep networks in nonlinear model reduction: distilling shock waves in nonlinear hyperbolic problems, arXiv preprint, arXiv:2007.13977, 2020.
[64] David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536.
[65] Justin Sirignano, Konstantinos Spiliopoulos, Dgm: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[66] R. Ştefănescu, Ionel Michael Navon, Pod/deim nonlinear model order reduction of an adi implicit shallow water equations model, J. Comput. Phys. 237 (2013) 95–114.
[67] Tommaso Taddei, Lei Zhang, Space-time registration-based model reduction of parameterized one-dimensional hyperbolic pdes, arXiv preprint, arXiv: 2004.06693, 2020.
[68] B.Ph. van Milligen, V. Tribaldos, J.A. Jiménez, Neural network differential equation and plasma equilibrium solver, Phys. Rev. Lett. 75 (20) (1995) 3594.
[69] E. Weinan, Bing Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (1) (2018) 1–12.
[70] G. Welper, Transformed snapshot interpolation with high resolution transforms, SIAM J. Sci. Comput. 42 (4) (2020) A2037–A2061.
[71] Paul Werbos, Beyond regression: new tools for prediction and analysis in the behavioral sciences, Ph. D. dissertation, Harvard University, 1974.
[72] Dunhui Xiao, Fangxin Fang, Andrew G. Buchan, Christopher C. Pain, Ionel Michael Navon, Juan Du, G. Hu, Non-linear model reduction for the Navier–Stokes equations using residual deim method, J. Comput. Phys. 263 (2014) 1–18.
[73] Huanhuan Yang, Alessandro Veneziani, Efficient estimation of cardiac conductivities via pod-deim model order reduction, Appl. Numer. Math. 115 (2017) 180–199.
[74] Yanfang Yang, Mohammadreza Ghasemi, Eduardo Gildin, Yalchin Efendiev, Victor Calo, et al., Fast multiscale reservoir simulations with pod-deim model reduction, SPE J. 21 (06) (2016) 2–141.
[75] Dongkun Zhang, Lu Lu, Ling Guo, George Em Karniadakis, Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems, J. Comput. Phys. 397 (2019) 108850.
[76] Pengfei Zhao, Cai Liu, Xuan Feng, Pod-deim based model order reduction for the spherical shallow water equations with Turkel-Zwas finite difference discretization, J. Appl. Math. 2014 (2014).
[77] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, Paris Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81.