Computer Methods in Applied Mechanics and Engineering 419 (2024) 116657

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

Active-learning-driven surrogate modeling for efficient simulation of parametric nonlinear systems

Harshit Kapadia *, Lihong Feng, Peter Benner

Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany

ARTICLE INFO

Dataset link: https://gitlab.mpi-magdeburg.mpg.de/kapadia/active-learning-surrogate-modeling, https://doi.org/10.5281/zenodo.10237734

Keywords:
Active learning
Error estimation
Data-driven surrogate modeling
Non-intrusive model order reduction
Shallow neural networks
Parametric dynamical systems

ABSTRACT

When repeated evaluations for varying parameter configurations of a high-fidelity physical model are required, surrogate modeling techniques based on model order reduction are desirable. In absence of the governing equations describing the dynamics, we need to construct the parametric reduced-order surrogate model in a non-intrusive fashion. In this setting, the usual residual-based error estimate for optimal parameter sampling associated with the reduced basis method is not directly available. Our work provides a non-intrusive error-estimator-based optimality criterion to efficiently populate the parameter snapshots, thereby, enabling us to effectively construct a parametric surrogate model. We consider parameter-specific proper orthogonal decomposition subspaces and propose an active-learning-driven surrogate model using kernel-based shallow neural networks (KSNNs), abbreviated as ActLearn-POD-KSNN surrogate model. The center location for each kernel, along with center-dependent kernel widths, can be learned for the KSNN by using an alternating dual-staged iterative training procedure. To demonstrate the efficiency of our proposed ideas, we present numerical experiments using four physical models, including incompressible Navier–Stokes equations. The ActLearn-POD-KSNN surrogate model efficiently predicts the solution at new parameter locations, even for a setting with multiple interacting shock profiles and a fluid flow scenario with Hopf bifurcation. We also provide an investigation of the surrogate's performance when the available data is noisy.

1. Introduction

In scenarios where computing the parametric full-order model (FOM) becomes computationally expensive, reduced-order modeling techniques provide beneficial alternatives. In recent years, there has been significant interest in developing non-intrusive model order reduction (MOR) approaches or data-driven surrogate models as they do not require access to first principle models. As a result, non-intrusive MOR is flexible for constructing reduced-order models (ROMs) of dynamical systems that are simulated using a black-box software or systems with limited access to the governing equations. Many of the non-intrusive MOR methods are based on machine learning: some use Gaussian process regression (GPR) [1–3], and radial basis function (RBF) interpolation [4–7], which can be interpreted as a shallow neural network, while many others use deep network architectures [8–18].

When non-intrusive MOR methods are employed, a substantial amount of training data, i.e., spatial solution field over parameter and time domain, is typically required to obtain a reasonable approximation of the underlying physics. Generating such a vast amount of training data is computationally expensive, since it is obtained by repeated evaluations of the FOM. Alternately, if the solution data is collected from experimental measurements, conducting repeated experiments for a vast pool of parametric configurations could become practically infeasible. To alleviate this situation, we propose a surrogate modeling framework for

* Corresponding author.
E-mail addresses: kapadia@mpi-magdeburg.mpg.de (H. Kapadia), feng@mpi-magdeburg.mpg.de (L. Feng), benner@mpi-magdeburg.mpg.de (P. Benner).

https://doi.org/10.1016/j.cma.2023.116657
Received 2 June 2023; Received in revised form 25 October 2023; Accepted 20 November 2023
Available online 6 December 2023

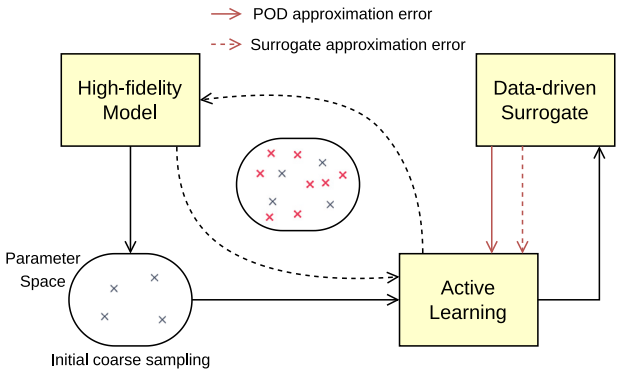


Fig. 1.1. Overview of the active-learning-driven surrogate modeling paradigm.

parametric nonlinear dynamical systems that actively generates solution snapshots at new parameter locations by querying the FOM only when necessary, thereby obtaining a set of optimal training data corresponding to important parameter values.

Fig. 1.1 provides an overview of our proposed active-learning-driven surrogate modeling paradigm. An initial coarse sampling of the parameter space is first considered, and the corresponding high-fidelity solution snapshots are generated and stored for a particular discrete-time trajectory. With this initial set of snapshots, a data-driven surrogate model is trained. By designing an appropriate optimality criterion that helps us to pick new important parameter locations, we can actively improve the accuracy of the surrogate model. Such an optimality criterion can be designed by using the error caused in the surrogate approximation. In this work, we employ a new strategy to design an error-estimator-based optimality criterion—constructing an error estimator from the parameter-specific POD-based solution approximations, where POD stands for proper orthogonal decomposition.

The active learning framework is similar in spirit to the greedy procedure in the reduced basis method (RBM) [19,20], developed for parametric dynamical systems. In RBM, an a posteriori error estimator is constructed by using the governing equations which then drives the greedy algorithm for constructing the reduced-order model. To reduce the offline time of the RBM when a large training set of parameter samples are required, authors in [21] propose an RBF-interpolation-based surrogate for the error estimator. This reduces the numerous ROM evaluations required for the error estimator construction, while enabling sufficient exploration of the parameter space for the RBM. However, all the greedy-based RBM methods are intrusive and need to access the partial differential equations (PDEs) of the FOM. Intrusive DL-MOR methods and relevant error estimation are proposed in [22–25], but they also require the discretized governing equations of the PDEs to be known.

In a non-intrusive setting, one cannot access the governing equations required for building the error estimator. Moreover, efficient error estimation for non-intrusive MOR is still rarely discussed in the literature. In [26], GPR is used to learn the error of the RBF-ROM from [4]. The learned error is the RBF-ROM's approximation error, which is a long vector whose dimension is same as the FOM dimension. In fact, the error estimator is obtained via two ROMs: RBF-ROM for the FOM, and GPR-based ROM for the error. Consequently, the learning process needs to be implemented for each element of the error vector, i.e., one Gaussian process (GP) error model is learned for each element of the error vector. Authors in [27] construct error models for data-driven surrogates of parametric systems by employing various regression techniques, and provide a detailed comparison between them. These error models are used to learn the error in a particular quantity of interest corresponding to each time step, which can then be used as corrections for the surrogate model's prediction of the quantity of interest.

In this work, we propose a new non-intrusive error estimator, estimating the POD-approximate state-vector error. Furthermore, we also propose a reliable optimality criterion based on this error estimator, which can enable dynamic updates of the training snapshots, which enables active learning. We do not construct a series of error models for the whole error vector, which would be expensive, as is done in [26]. Instead, only a single error model is learned. The proposed error estimator is computationally much cheaper.

We focus on building an error estimator that outputs an estimation of the errors corresponding to each discrete time instance at new out-of-training parameter locations. Later, we look at the temporal maximum of the predicted error values, referred to as the error-estimator-based optimality criterion, to pick new samples and drive the training set adaptivity. Moreover, the error estimator is rebuilt iteratively, following each update of the snapshot data, as well as an adaptation of the POD bases. This process iteratively improves the accuracy of the proposed error estimator, as well as the data-driven surrogate model. In contrast, the attempt in existing literature for non-intrusive MOR is to only build an error indicator once, by using a static set of error training data [26,27].

Active learning is also proposed in [3,5,7,28]. The method in [5] proposes a greedy non-intrusive method that selects the parameters iteratively according to a proposed indicator. However, the indicator has nothing to do with the true error of the approximate solution. The method in [7] proposes RBF interpolation for predicting the reduced state vector in future time instances. Greedy algorithms are proposed to adaptively select the snapshots of the reduced state vector that are called projected snapshots. Moreover, the projected snapshots are greedily selected according to a residual and a power function, rather than by error estimation of the approximate solution. The projected snapshots are selected from precomputed solution snapshots at a given set of time instances that need extensive offline computations. The method applies only to non-parametric time-dependent cases. In [3], GPR

is proposed to learn the reduced state vector as a function of parameters. Active learning using deviation of GP as an indicator to iteratively enrich the training data (snapshots) that are then used for retraining GPR. Again, the deviation of GP cannot predict the error of the approximate solution computed from the proposed method there. Only steady-state problems are addressed in [3], and extension of the method to time-dependent problems is not straightforward. Similarly, in the most recent work [28], an error estimator based GPR is proposed to perform active learning by using single-time step snapshots of the parametric system states. Their method works for time-dependent problems, but its performance for models with mixed—convective and diffusive—effects is unclear.

While the proposed active learning framework can in principle be applied to any POD-based surrogate model, in this work, we apply it to a new surrogate model called POD-KSNN, which employs parameter-specific POD subspaces and a kernel-based shallow neural network (KSNN) to perform interpolation/regression. This approach follows from the RBF-ROM in [4], where the snapshots at a new parameter sample are learned via RBF interpolation, and the reduced basis for the solution space at a new parameter is obtained via singular value decomposition (SVD) of the approximate snapshot matrix corresponding to the new parameter. However, in this work, we exploit the interpretation of RBF interpolation as a neural network with a single hidden layer equipped with RBF kernels as its activation functions. As opposed to standard RBF interpolation in [4], with uniform kernel widths and fixed center locations, we equip the KSNN with trainable center locations and center-specific locally-adaptive kernel widths by proposing an alternating dual-staged iterative training (ADSIT) procedure. This leads to improved performance over the standard RBF interpolation. Furthermore, it enables us to perform not only interpolation but also regression. Moreover, instead of requiring to construct individual RBF interpolants to interpolate between each entry of the parametric snapshot matrices in the parameter domain, we efficiently train a single KSNN in the parameter domain, i.e., μ-KSNN. Furthermore, for predictions in the time domain, we similarly train an additional KSNN (t-KSNN) over the POD modal coefficients.

The proposed active-learning-driven-POD-KSNN (ActLearn-POD-KSNN) surrogate iteratively detects locations in the parameter domain where the variation between solution features is high, and queries the FOM solver in those regions to generate new training snapshots. The POD subspace enrichment during the active learning procedure results in a varying number of POD bases between each of the parameter-specific subspaces, corresponding to different levels of energy (information) retention in each subspace. This enables us to adaptively choose an appropriate energy criterion for creating POD subspaces at newly queried parameter samples in the online phase such that the subspaces are expressive enough to provide a solution approximation up to a desired accuracy. The proposed active learning framework builds a surrogate model in an efficient fashion—by limiting the generation of the expensive FOM snapshots to an optimal set of parameter samples which still provide a sufficient exploration of the parameter space.

The shallow architecture of KSNN, along with the ADSIT procedure, makes it possible to adaptively create and deploy the network upon a frequently updating dataset. Such a situation arises in the active learning process due to the iterative construction of the non-intrusive error estimator. As a result, during active learning, we employ a KSNN for building the error estimator and refer to it as AL-KSNN.

The non-intrusive error-estimator-based optimality criterion is built from the error arising in a parameter-specific POD-approximation of the solution states. To the best of our knowledge, this is in contrast to all the previously proposed frameworks for active learning. As a result, our optimality criteria allows us to actively learn important solution snapshots at new parameter locations, without the need to repeatedly re-evaluate and retrain the entire non-intrusive ROM. This further reduces the computational burden. During the online phase, we do not need to evaluate the high-fidelity model, but simply query the actively learned non-intrusive ROM.

Regarding our newly proposed adaptive POD subspace technique, some existing work that adaptively constructs POD subspaces focuses on various other aspects. For instance, [29] proposes an intrusive nonlinear ROM that uses POD and the discrete empirical interpolation method (DEIM). They aim to adapt the DEIM basis in the online stage by sampling the nonlinear function at some key locations. In a similar fashion, [30] proposes an adaptive sampling approach to select key components at which the FOM should be queried and subsequently update the DEIM basis. They focus on constructing efficient ROMs for transport-dominated problems. However, [29,30] deals with intrusive MOR, where the governing equations are accessible. This is in stark contrast to the setting in our work. Authors in [31] propose shifted POD by enabling time-dependent shifts of the snapshot matrix, rendering a physically intuitive basis for multi-transport phenomena. As part of their multi-shift and reduce approach, they iteratively update the POD modes with the objective to determine the underlying transport velocities and separate them. However, their approach is specifically designed to address multiple transport phenomena and primarily deals with non-parametric snapshot data. Furthermore, all the aforementioned works focus on adapting a global bases set, whereas, our focus is towards building localized parameter-specific adaptive bases sets.

The remaining article is organized as follows. Section 2 introduces the general setting for parametric nonlinear dynamical systems. This is followed by presenting the non-intrusive error-estimator-based optimality criterion and the active learning framework for POD-based surrogate models. Next, in Section 3, we introduce the kernel-based shallow neural network (KSNN) and the newly proposed alternating dual-staged iterative training procedure (ADSIT). Later, the POD-based data-driven surrogate model using KSNNs, i.e., POD-KSNN, is outlined. Section 4 summarizes the novel ActLearn-POD-KSNN surrogate model by detailing its complete algorithm and providing an analysis of the computational complexity. Here, we also investigate the performance of ActLearn-POD-KSNN surrogate when the available solution data is corrupted by noise. Then, we provide detailed numerical experiments for various models with mixed—convective and diffusive—physical phenomena in Section 5. At the end, we draw some conclusions in Section 6.

2. Active learning for POD-based data-driven surrogates

We can represent a full-order nonlinear dynamical system arising from the spatial discretization of a parametric partial differential equation as

$$\frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u}, t; \boldsymbol{\mu}), \quad \mathbf{u}(0) = \mathbf{u}_0(\boldsymbol{\mu}), \quad t \in [0, T], \tag{2.1}$$

where $T \in \mathbb{R}^+$ denotes the final time; $\mathbf{u} \equiv \mathbf{u}(t, \boldsymbol{\mu})$ with $\mathbf{u} : [0, T] \times \mathcal{D} \to \mathbb{R}^N$ denotes the solution; $\mathbf{u}_0 : \mathcal{D} \to \mathbb{R}^N$ denotes the parametrized initial condition; $\boldsymbol{\mu} \in \mathcal{D} \subseteq \mathbb{R}^{N_\mu}$ denotes the parameters; and $\mathbf{f} : \mathbb{R}^N \times [0, T] \times \mathcal{D} \to \mathbb{R}^N$ denotes a nonlinear function.

To create an accurate reduced-order surrogate model, one typically requires a substantial amount of training data at quite a few parameter samples. Our aim is to be efficient and choose a set of optimal training samples, from a vast pool of parameter samples. However, there is no trivial notion of optimality. We address this by proposing a non-intrusive error-estimator-based optimality criterion. This is further used to actively generate the training or snapshot data and leverage the most out of a data-driven surrogate.

2.1. Non-intrusive error-estimator-based optimality criterion

To assess the quality of any POD-based data-driven surrogate's solution, we require a way to estimate the error in its approximate solution, in comparison with the full-order (or high-fidelity) solution. There are two types of errors induced while constructing and deploying the POD-based surrogate: the error caused due to restricting the solution corresponding to each parameter sample in an (active) linear subspace obtained via POD, and the amalgamation of errors arising from the chosen interpolation or regression technique. We assume that the total error $\mathcal{E} \in \mathbb{R}^N$ of the data-driven surrogate in the spatial domain for some parameter $\boldsymbol{\mu}$ corresponding to a time instance t can be represented as

$$\mathcal{E}(t, \boldsymbol{\mu}) = \mathcal{E}_{POD}(t, \boldsymbol{\mu}) + \mathcal{E}_\star(t, \boldsymbol{\mu}), \tag{2.2}$$

where $\mathcal{E}_{POD} \in \mathbb{R}^N$ represents the POD projection error, and $\mathcal{E}_\star \in \mathbb{R}^N$ represents the total interpolation or regression error when the surrogate model is evaluated.

Consider the norm of the total error in the spatial domain,

$$\|\mathcal{E}(t, \boldsymbol{\mu})\| = \|\mathcal{E}_{POD}(t, \boldsymbol{\mu}) + \mathcal{E}_\star(t, \boldsymbol{\mu})\|. \tag{2.3}$$

Upon application of the triangle inequality, we obtain

$$\|\mathcal{E}(t, \boldsymbol{\mu})\| \leq \|\mathcal{E}_{POD}(t, \boldsymbol{\mu})\| + \|\mathcal{E}_\star(t, \boldsymbol{\mu})\|, \tag{2.4}$$

$$\epsilon(t, \boldsymbol{\mu}) \leq \epsilon_{POD}(t, \boldsymbol{\mu}) + \epsilon_\star(t, \boldsymbol{\mu}), \tag{2.5}$$

where $\epsilon := \|\mathcal{E}\|$, $\epsilon_{POD} := \|\mathcal{E}_{POD}\|$, and $\epsilon_\star := \|\mathcal{E}_\star\|$ represent the respective spatial norms.

In a situation when the error caused due to the interpolation or regression procedure, i.e., $\epsilon_\star$, is smaller in magnitude than the error contribution from ϵ_{POD}, ϵ_{POD} can be approximately used as a bound for the total error ϵ. As a result, we can construct an estimator for the error ϵ caused in the surrogate solution by interpolating between the errors ϵ_{POD} corresponding to all the training parameter samples. Afterward, this error estimator can be queried at new parameter samples, providing an approximation for ϵ at any out-of-training parameters. By employing such an error estimate, we later devise an optimality criterion which lets us drive the active learning procedure. The key benefit is that we can carry out active learning without repeatedly retraining and evaluating the surrogate model in the offline phase.

Let us now dive into the details of the error estimator construction. Consider that we have the solution snapshots coming from Eq. (2.1), along discrete time trajectories $\{t_0, t_1, \ldots, t_{N_t}\}$ with $t_0 = 0$ and $t_{N_t} = T$. The snapshots can be collected in matrices $U(\boldsymbol{\mu}_i)$ sized $N \times (N_t + 1)$ corresponding to each parameter sample $\boldsymbol{\mu}_i$ for $i \in \{1, \ldots, m\}$,

$$U(\boldsymbol{\mu}_i) = [\ \mathbf{u}(t_0, \boldsymbol{\mu}_i) \mid \mathbf{u}(t_1, \boldsymbol{\mu}_i) \mid \ \ldots \ \mid \mathbf{u}(t_{N_t}, \boldsymbol{\mu}_i)\ ]. \tag{2.6}$$

The snapshot matrices can be factorized using the singular value decomposition (SVD),

$$U(\boldsymbol{\mu}_i) = L(\boldsymbol{\mu}_i)\Sigma(\boldsymbol{\mu}_i)K^\top(\boldsymbol{\mu}_i), \tag{2.7}$$

where $L \in \mathbb{R}^{N \times N}$, $\Sigma \in \mathbb{R}^{N \times (N_t+1)}$, and $K \in \mathbb{R}^{(N_t+1) \times (N_t+1)}$. The parameter-specific POD subspaces $\Phi(\boldsymbol{\mu}_i) \in \mathbb{R}^{N \times r_i}$ can then be written as

$$\Phi(\boldsymbol{\mu}_i) = [\phi_1(\boldsymbol{\mu}_i) \mid \phi_2(\boldsymbol{\mu}_i) \mid \ \ldots \ \mid \phi_{r_i}(\boldsymbol{\mu}_i)], \tag{2.8}$$

where $\phi_k(\boldsymbol{\mu}_i) := \mathbf{l}_k(\boldsymbol{\mu}_i)$ with $\mathbf{l}_k(\boldsymbol{\mu}_i)$ being the kth column of the left singular value matrix $L(\boldsymbol{\mu}_i)$, and $r_i \leq \min\{N, (N_t + 1)\}$ denotes the level of POD subspace truncation corresponding to each parameter $\boldsymbol{\mu}_i$. Now, let us consider a POD approximation of the solution snapshots,

$$U_{POD}(\boldsymbol{\mu}_i) = \Phi(\boldsymbol{\mu}_i)A(\boldsymbol{\mu}_i), \tag{2.9}$$

where $A(\boldsymbol{\mu}_i) \in \mathbb{R}^{r_i \times (N_t+1)}$. Each entry a_{kj} of $A(\mu_i)$ is defined as $a_{kj} := \phi_k^\top(\boldsymbol{\mu}_i)\mathbf{u}(t_j, \boldsymbol{\mu}_i)$ with $k = 1, \ldots, r_i$ and $j = 1, \ldots, (N_t + 1)$.

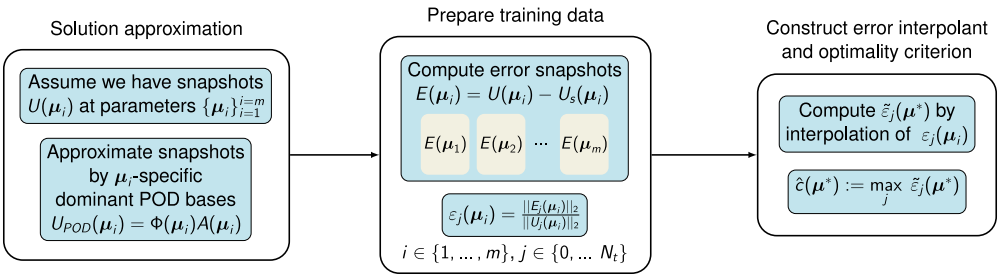


Fig. 2.1. Procedure to non-intrusively estimate the error at new parameter locations.

The error of the POD approximate solution at the parameters $\{\mu_1, \mu_2, \dots, \mu_m\}$ is given by

$$E(\mu_i) = U(\mu_i) - U_{POD}(\mu_i), \ i = 1, \dots, m. \tag{2.10}$$

We avoid interpolating $E(\mu_i)$ entry-wise, as opposed to [26], since the computational cost for doing that is equivalent to interpolating the high-dimensional snapshots. Our motivation is to alleviate this computational burden, but at the same time, have an informative indication for the error. This is accomplished by taking the norm of the solution error $E_j(\mu_i) \in \mathbb{R}^N$, i.e., the jth column of $E(\mu_i)$, at each time instance t_j. In our experiments, we have tested using l^2, l^1, and l^∞ norms. We noticed similar qualitative results for all the norms. Depending on the problem setting, one can be preferred over the other, for instance, physical systems prone to advective effects might benefit from l^1 norm usage. As a generic choice, we use the l^2 norm for our discussion because the considered test problems in Section 5 have mixed diffusive and convective effects.

We denote the relative error by,

$$\varepsilon_j(\mu_i) = \frac{\|E_j(\mu_i)\|_2}{\|U_j(\mu_i)\|_2}, \tag{2.11}$$

where $U_j(\mu_i)$ denotes the jth column (corresponding to t_j) of the snapshot matrix for μ_i. This can be written compactly as a vector with entries corresponding to each time instance,

$$\boldsymbol{\varepsilon}(\mu_i) = [\ \varepsilon_0(\mu_i), \varepsilon_1(\mu_i), \dots, \varepsilon_{N_t}(\mu_i)\ ]^\top. \tag{2.12}$$

Further, we define the maximal relative error in time at each parameter sample μ_i as

$$\hat{\varepsilon}(\mu_i) := \|\boldsymbol{\varepsilon}(\mu_i)\|_\infty = \max\left(\varepsilon_0(\mu_i), \varepsilon_1(\mu_i), \dots, \varepsilon_{N_t}(\mu_i)\right). \tag{2.13}$$

This quantity will play an essential role in the procedure undertaken to adaptively enrich the parameter-specific POD subspaces $\Phi(\mu_i)$. A more detailed discussion regarding this is included in Section 2.2.

The relative error values $\boldsymbol{\varepsilon}(\mu_i)$ at all parameter samples μ_i for $i \in \{1, \dots, m\}$ can be used as training data to construct an interpolant or regressor $\tilde{\boldsymbol{\varepsilon}}(\boldsymbol{\mu})$ in the parameter space. Later, the learned relative error at time t_j for any new parameter value μ^* can be written compactly as a vector with entries corresponding to each time instance,

$$\tilde{\boldsymbol{\varepsilon}}(\mu^*) = [\ \tilde{\varepsilon}_0(\mu^*), \tilde{\varepsilon}_1(\mu^*), \dots, \tilde{\varepsilon}_{N_t}(\mu^*)\ ]^\top. \tag{2.14}$$

The final error-estimator-based optimality criterion is taken to be the maximal learned relative error in time, given by

$$\hat{c}(\mu^*) := \|\tilde{\boldsymbol{\varepsilon}}(\mu^*)\|_\infty = \max(|\tilde{\varepsilon}_0(\mu^*)|, |\tilde{\varepsilon}_1(\mu^*)|, \dots, |\tilde{\varepsilon}_{N_t}(\mu^*)|). \tag{2.15}$$

The entire procedure to compute the non-intrusive error estimator, and the optimality criterion based on it, is summarized in Fig. 2.1.

Remark 2.1 (***Strategy to interpolate the relative error values*** $\boldsymbol{\varepsilon}(\mu_i)$ ***in the parameter domain***)**.** We use a KSNN, as introduced in Section 3.1, to interpolate between the relative error values $\boldsymbol{\varepsilon}(\mu_i) \in \mathbb{R}^{(N_t+1)}$ where $i \in \{1, \dots, m\}$. This choice is primarily due to its fast training procedure, allowing us to repeatedly reconstruct accurate interpolants $\tilde{\boldsymbol{\varepsilon}}(\cdot)$ in an efficient fashion, as the amount of available data m changes. The need for their repeated reconstructions has to do with the frequent updates of the error-estimator-based optimality criterion $\hat{c}(\cdot)$, instrumental in driving the active learning algorithm. For a detailed discussion regarding this, please refer to Section 2.2.

For the sake of completeness, while using a KSNN, the interpolated relative error at time t_j for any new parameter value μ^* becomes

$$\tilde{\boldsymbol{\varepsilon}}(\mu^*) = \begin{bmatrix} \sum_{i=1}^m w_i^{(0)} \phi_i(\|\mu^* - \mu_i\|; \epsilon_i) \\ \sum_{i=1}^m w_i^{(1)} \phi_i(\|\mu^* - \mu_i\|; \epsilon_i) \\ \vdots \\ \sum_{i=1}^m w_i^{(N_t+1)} \phi_i(\|\mu^* - \mu_i\|; \epsilon_i) \end{bmatrix}. \tag{2.16}$$

For exact details about each term, please refer Section 3.1. As this KSNN-based interpolant for the relative error values is repeatedly constructed during the active learning loop, we shortly refer to it as AL-KSNN.

2.2. Active learning framework

The intention of the active learning procedure is to enrich the snapshot data in a fashion that is most beneficial for the reduced-order surrogate model. In other words, each enrichment of the training snapshots lead to an optimal or near-optimal improvement of the approximate dynamics. The motivation is similar to the greedy procedure used for the reduced basis method [19,20]. However, in our setting we do not have access to the first principle models, so we cannot leverage the equations to decide the choice of new parameter samples for efficient training of the non-intrusive ROM. Instead, we utilize the non-intrusive error estimator in Eq. (2.14) and the optimality criterion in Eq. (2.15) to enable active learning.

We initialize the parameter set P with a coarse sampling of the parameter space $\mathcal{D}$. Additionally, a second set P^* is prepared which holds all the candidate parameter values that could be included in set P as the active learning progresses. Consider coarse initial sampling set P, i.e., $P = \{\boldsymbol{\mu}_i | i \in I\}$ with index set $I = \{1, \dots, p\}$. And the candidate set P^* is composed of a fine sampling in $(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2)$, given by

$$P^* = \{\bar{\boldsymbol{\mu}}_1, \bar{\boldsymbol{\mu}}_2, \dots, \bar{\boldsymbol{\mu}}_q\} \text{ with } \bar{\boldsymbol{\mu}}_j \neq \boldsymbol{\mu}_i;\ i = 1, \dots, p;\ j = 1, \dots, q. \tag{2.17}$$

The high-fidelity solution is computed for all the parameters in set P. This is followed by performing a POD approximation Eq. (2.9) for the snapshot matrix at each $\boldsymbol{\mu}_i$ in P. The level of POD space truncation r_i for each $\boldsymbol{\mu}_i \in P$ is obtained by maintaining a constant energy criterion $\eta(\boldsymbol{\mu}_i)$ defined at $\boldsymbol{\mu}_i$ as

$$\eta(\boldsymbol{\mu}_i) = 1 - \frac{\sum_{k=1}^{k=r_i} (\sigma_k^{(i)})^2}{\sum_{k=1}^{k=s_i} (\sigma_k^{(i)})^2}. \tag{2.18}$$

Here, $\sigma_k^{(i)}$ with $k = 1, \dots, s_i$, are the nonzero singular values obtained from SVD of the snapshot matrix $U(\boldsymbol{\mu}_i)$.

Using the high-fidelity and POD approximate solutions, error snapshots for the parameters in set P are computed,

$$E^{(1)}(P) := \{E(\boldsymbol{\mu}_1), \dots, E(\boldsymbol{\mu}_p)\}, \tag{2.19}$$

where $E(\boldsymbol{\mu}_i)$ is given by Eq. (2.10) and the superscript in $E^{(1)}$ denotes the first iteration of the active learning loop, i.e. $iter = 1$. The error-estimator-based optimality criterion is computed for the parameters in the candidate set P^*,

$$\hat{c}^{(1)}(P^*) := \{\hat{c}(\bar{\boldsymbol{\mu}}_1), \hat{c}(\bar{\boldsymbol{\mu}}_2), \dots, \hat{c}(\bar{\boldsymbol{\mu}}_q)\}, \tag{2.20}$$

where $\hat{c}(\cdot)$ is computed by Eq. (2.15).

The parameter sample corresponding to the maximal value of the optimality criterion is chosen ($iter = 1$),

$$\bar{\boldsymbol{\mu}}^{(iter)} = \underset{\bar{\boldsymbol{\mu}} \in P^*}{\operatorname{argmax}}\, \hat{c}(\bar{\boldsymbol{\mu}}). \tag{2.21}$$

Then we pick the parameter corresponding to the maximal $\hat{\varepsilon}(\boldsymbol{\mu}_i)$ Eq. (2.13) among all parameters $\boldsymbol{\mu}_i$ in the set P, i.e.,

$$\mathring{\boldsymbol{\mu}} = \underset{\boldsymbol{\mu} \in P}{\operatorname{argmax}}\, \hat{\varepsilon}(\boldsymbol{\mu}). \tag{2.22}$$

If $\hat{\varepsilon}(\mathring{\boldsymbol{\mu}}) > \hat{c}(\bar{\boldsymbol{\mu}}^{(iter)})$, the POD subspace for $\mathring{\boldsymbol{\mu}}$ is enriched by adding one more basis. This is followed by re-evaluating Eqs. (2.20)–(2.22), checking if $\hat{\varepsilon}(\mathring{\boldsymbol{\mu}}) > \hat{c}(\bar{\boldsymbol{\mu}}^{(iter)})$ still holds true, and incrementing the POD subspace by one additional basis if necessary. Note that while undertaking such a procedure, $\bar{\boldsymbol{\mu}}^{(iter)}$ and $\mathring{\boldsymbol{\mu}}$ could vary upon re-evaluations. This step essentially makes sure that the POD subspace for each parameter sample in the set P, at every iteration, is expressive enough, such that the maximal relative error for all parameters in P is always less than the estimated maximal relative error ($\hat{c}$) for any candidate parameter in P^*. Moreover, due to this procedure, we can obtain an update of the energy criterion $\eta(\mathring{\boldsymbol{\mu}})$, which is later used during the online phase to construct an accurate POD subspace for any newly queried parameter $\boldsymbol{\mu}^*$. A more detailed discussion about obtaining and using such an updated energy criterion is provided in Section 4.

Now, if $\hat{c}(\bar{\boldsymbol{\mu}}^{(iter)}) > tol$, we compute the high-fidelity snapshots $U(\bar{\boldsymbol{\mu}}^{(iter)})$. Here, tol is a predefined tolerance level that we intend to achieve. In other words, we terminate the active learning process as soon as the error-estimator-based optimality criterion $\hat{c}(\bar{\boldsymbol{\mu}}^{(iter)})$ reaches a value which is less than the target tolerance. The set P is extended by including the chosen parameter $\bar{\boldsymbol{\mu}}^{(iter)}$. The candidate set is also updated, $P^* = P^* \setminus \bar{\boldsymbol{\mu}}^{(iter)}$. Next, the POD approximation error at $\bar{\boldsymbol{\mu}}^{(iter)}$ is evaluated,

$$E(\bar{\boldsymbol{\mu}}^{(iter)}) = U(\bar{\boldsymbol{\mu}}^{(iter)}) - U_{POD}(\bar{\boldsymbol{\mu}}^{(iter)}), \tag{2.23}$$

where U_{POD} is constructed by adhering to the energy criterion η in Eq. (2.18).

With the necessary POD space enrichment for the pre-existing parameter samples in P, and later, the inclusion of $\bar{\boldsymbol{\mu}}^{(iter)}$, the error snapshots $E^{(iter)}(P)$ are collected to start the next iteration. This is used as the training data for the interpolation procedure to construct the new optimality criteria $\hat{c}^{(iter)}(P^*)$. If required, an adaptive refinement of the POD space is carried out until $\hat{c}(\bar{\boldsymbol{\mu}}^{(iter)}) \geq \hat{\varepsilon}(\mathring{\boldsymbol{\mu}})$ holds true at that iteration. The next parameter $\bar{\boldsymbol{\mu}}^{(iter)}$ is then greedily picked using Eq. (2.21). In this fashion, we iteratively expand the solution snapshots by greedy selection of new parameters along with necessary parameter-specific POD space adaptations, until the tolerance tol is satisfied. The complete active learning procedure is outlined in Fig. 2.2.

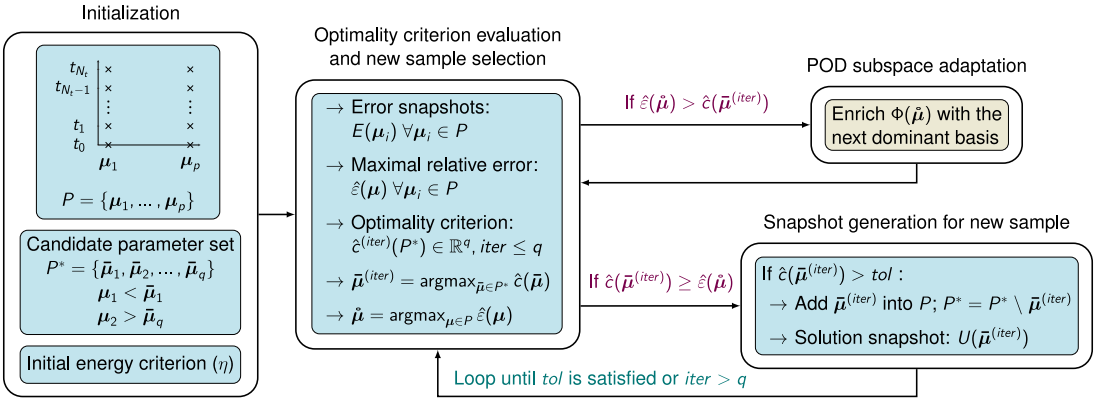


Fig. 2.2. Procedure to actively learn a surrogate model by greedy refinement of the high-fidelity solution snapshots along the parameter domain.

Remark 2.2 (***Choice of candidate parameter set*** P^*)**.** The preparation of P^* is not trivial, in fact, a decent sampling procedure needs to be maintained while choosing the candidate samples for P^*. As the parameter values range over several orders of magnitude for the physical models presented in our numerical experiments, we consider a uniform sampling of the logarithm of the entire parameter space. This ensures a reasonable selection of parameters through all of the parameter space.

Remark 2.3 (***Extension to system of parametric PDEs***)**.** We prepare different snapshot matrices corresponding to all the solution components in the system of equations, i.e., $\{U^{(k)}\}_{k=1}^q$ for q components. This is followed by forming the error matrices $\{E^{(k)}\}_{k=1}^q$, and then proceeding to apply the aforementioned active learning procedure, but now obtaining component-wise error-estimator-based optimality criteria $\{(\hat{c}^{(iter)})^{(k)}\}_{k=1}^q$ by following Eq. (2.20) for each component. The new parameter sample is picked based on the maximal value of the average over all the component-wise optimality criteria. So, Eq. (2.21) is changed to the following form:

$$\bar{\mu}^{(iter)} = \underset{\bar{\mu} \in P^*}{\text{argmax}} \left(\frac{1}{q} \sum_{k=1}^{q} \hat{c}^{(k)}(\bar{\mu}) \right). \tag{2.24}$$

In a similar fashion, we also pick the parameter corresponding to the component-averaged maximal relative error among all parameters in the set P. So, Eq. (2.22) is changed to the following form:

$$\mathring{\mu} = \underset{\mu \in P}{\text{argmax}} \left(\frac{1}{q} \sum_{k=1}^{q} \hat{\varepsilon}^{(k)}(\mu) \right). \tag{2.25}$$

Similar to the scalar equation setting, the POD subspaces for samples in P are now enriched until $\frac{1}{q}\sum_{k=1}^q \hat{c}^{(k)}(\bar{\mu}^{(iter)}) \geq \frac{1}{q}\sum_{k=1}^q \hat{\varepsilon}^{(k)}(\mathring{\mu})$ holds true. Later, if $\frac{1}{q}\sum_{k=1}^q \hat{c}^{(k)}(\bar{\mu}^{(iter)}) > tol$, the set P is extended by adding $\bar{\mu}^{(iter)}$ to it. The candidate set is also updated, $P^* = P^* \setminus \bar{\mu}^{(iter)}$. The greedy active learning iterations are continued until the user-defined tolerance is satisfied.

3. Data-driven surrogate model for parametric systems

In this section, we provide a formulation of a new POD-KSNN reduced-order surrogate model which can be constructed directly from the high-fidelity solution snapshots of Eq. (2.1). The surrogate model is built by employing two neural networks with a shallow network architecture. The shallowness enables a fast offline training procedure as well as a rapid online querying of the surrogate model at new out-of-training parameter locations.

The POD-KSNN surrogate employs a two-step interpolation/regression approach similar to the RBF-ROM in [4]. However, it is unclear as to how the RBF weights are computed in [4]. A straightforward way is to construct several RBF interpolants, each corresponding to a scalar value that needs to be interpolated. But instead of that, we only use a single kernel-based shallow neural network (KSNN) for interpolation/regression of vector quantities, by performing an efficient solve for the weights. Moreover, as opposed to uniform kernel widths and fixed center locations in RBF interpolation, the KSNN is equipped with trainable center-specific kernel widths and center locations. Furthermore, we propose a new alternating dual-staged iterative training (ADSIT) procedure to train the KSNN, allowing us to not only perform interpolation, but also regression.

In the POD-KSNN surrogate, we build the parameter-specific POD subspaces such that they retain the same level of energy or information content, which results in POD subspaces of varying dimensions. Let us start by introducing the kernel-based shallow neural network (KSNN) in Section 3.1, followed by details about the POD-KSNN reduced-order surrogate model in Section 3.2.

Table 3.1
List of radial basis kernels.

Name	Function
Gaussian	$e^{-(d/\epsilon)^2}$
Multi-quadric	$\sqrt{(d/\epsilon)^2+1}$
Inverse multi-quadric	$1/(\sqrt{(d/\epsilon)^2+1})$
Matern52	$\left(1+\sqrt{5}(d/\epsilon)+\frac{5}{3}(d/\epsilon)^2\right)e^{-\sqrt{5}(d/\epsilon)}$
Matern32	$\left(1+\sqrt{3}(d/\epsilon)\right)e^{-\sqrt{3}(d/\epsilon)}$
Matern12	$e^{-(d/\epsilon)}$

3.1. KSNN with an alternating dual-staged iterative training procedure

We formulate the interpolation/regression technique that is used in this work as a kernel-based shallow neural network (KSNN). The network is as shown in Fig. 3.1 with an input, a hidden, and an output layer. The input layer takes in the data points $\boldsymbol{x}_j \in \mathbb{R}^p$, where $j = 1, \dots, \ell$. The activation functions in the hidden layer are the kernels $\phi_i(\boldsymbol{x}) := \phi(\|\boldsymbol{x} - \mathbf{c}_i\|; \epsilon_i)$, operating on multivariate input data $\boldsymbol{x} \in \mathbb{R}^q$, which in turn reduces to a scalar function of the Euclidean norm of $(\boldsymbol{x} - \mathbf{c}_i)$, denoted by $\|\cdot\|$. The vector-valued output $\boldsymbol{y}(\boldsymbol{x}) \in \mathbb{R}^q$ of the network is used to learn (approximate) a vector-valued function $\widehat{\boldsymbol{y}}(\boldsymbol{x}) \in \mathbb{R}^q$. Mathematically, this can be expressed as follows,

$$\widehat{\boldsymbol{y}}(\boldsymbol{x}) \approx \boldsymbol{y}(\boldsymbol{x}) = [\ y_1(\boldsymbol{x}),\ y_2(\boldsymbol{x}),\ \dots,\ y_q(\boldsymbol{x})\ ]^\top \tag{3.1}$$

$$= \begin{bmatrix} \sum_{i=1}^{n_c} w_i^{(1)} \phi(\|\boldsymbol{x} - \mathbf{c}_i\|; \epsilon_i) \\ \sum_{i=1}^{n_c} w_i^{(2)} \phi(\|\boldsymbol{x} - \mathbf{c}_i\|; \epsilon_i) \\ \vdots \\ \sum_{i=1}^{n_c} w_i^{(q)} \phi(\|\boldsymbol{x} - \mathbf{c}_i\|; \epsilon_i) \end{bmatrix}, \tag{3.2}$$

where $\{w_i^{(k)}\}_{i=1}^{n_c}$ are the network weights; $\{\mathbf{c}_i\}_{i=1}^{n_c}$ denote the center locations; and ϕ is a kernel function depending on the radial distance $d := \|\boldsymbol{x} - \mathbf{c}\|$ of input $\boldsymbol{x}$ from a specified center $\mathbf{c}$ and the kernel width ϵ that may vary with $\mathbf{c}$. As an example, Table 3.1 lists various radial basis kernels with the kernel width ϵ, and the radial distance d. Note that later on, in Section 3.2, we use the notation $\mathcal{I}^x(\boldsymbol{x})$ to refer to the interpolant/regressor $\boldsymbol{y}(\boldsymbol{x})$ over the domain $\boldsymbol{x}$.

3.1.1. Computing the output layer weights via efficient solve of a linear system with multiple right-hand sides

Consider that the center location of kernels $\{\mathbf{c}_i\}_{i=1}^{n_c}$, and their widths $\{\epsilon_i\}_{i=1}^{n_c}$ are given. In such a situation, we only need to obtain the output layer weights of the network. All the weights $\{w_i^{(k)}\}_{i=1}^{n_c}$ can be collected in a weight matrix W such that its entries are defined as

$$W_{i,k} := w_i^{(k)}. \tag{3.3}$$

Also, consider that data pairs $\{\boldsymbol{x}_j \in \mathbb{R}^p, \widehat{\boldsymbol{y}}_j \in \mathbb{R}^q\}_{j=1}^{\ell}$ from ℓ samples of a vector-valued function $\widehat{\boldsymbol{y}}(\boldsymbol{x})$ are given.

To obtain the weights in each column of W, we only need to solve a linear system with q right-hand sides, which can be compactly written as

$$DW = Y. \tag{3.4}$$

Here, the distance matrix $D \in \mathbb{R}^{\ell \times n_c}$ holds all the hidden layer activations, with the entries defined as below:

$$D_{j,i} := \phi_i(\boldsymbol{x}_j) = \phi(\|\boldsymbol{x}_j - \mathbf{c}_i\|; \epsilon_i), \tag{3.5}$$

with $j = 1, \dots, \ell$ and $i = 1, \dots, n_c$. The matrix $Y \in \mathbb{R}^{\ell \times q}$ contains all the available samples of the function $\widehat{\boldsymbol{y}}(\boldsymbol{x})$, and is defined as below:

$$Y := [\ \widehat{\boldsymbol{y}}_1(\boldsymbol{x}_1) \mid \widehat{\boldsymbol{y}}_2(\boldsymbol{x}_2) \mid \dots \mid \widehat{\boldsymbol{y}}_\ell(\boldsymbol{x}_\ell)\ ]^\top. \tag{3.6}$$

Interpolation mode: To perform an exact interpolation between the available data pairs $\{\boldsymbol{x}_j, \widehat{\boldsymbol{y}}_j\}_{j=1}^{\ell}$, the total number of kernels, i.e., the size of the hidden layer, is taken to be the same as the total available data samples. In this setting, the kernels are centered around the input data $\{\boldsymbol{x}_j\}_{j=1}^{\ell}$. This translates to $n_c = \ell$ and $\mathbf{c}_i = \boldsymbol{x}_i$ with $i = 1, \dots, \ell$.

We first perform a pivoted LU decomposition of the distance matrix D,

$$D = PLU, \tag{3.7}$$

where P is a permutation matrix, L is a lower triangular matrix with unit diagonal elements, and U is an upper triangular matrix. When such a decomposition is available, each column of W can be obtained by simply performing forward and back substitutions. As a result, we just perform one LU factorization in $\mathcal{O}(\ell^3)$ followed by q operations in $\mathcal{O}(\ell^2)$.

Note that we use the LU decomposition to enable usage of kernels which are not positive definite, thereby allowing our implementation to be more general. However, when a symmetric positive definite kernel is considered, we can use the Cholesky

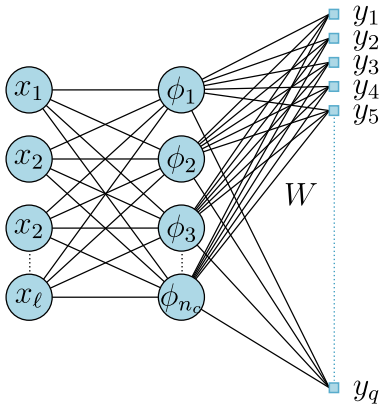


Fig. 3.1. Illustration of a kernel-based shallow neural network.

decomposition instead of LU decomposition and obtain a further reduction in computational complexity from $\frac{2}{3}\ell^3$ to $\frac{1}{3}\ell^3$ during the first step of matrix factorization.

Regression mode: To regress between the available data pairs $\{\boldsymbol{x}_j, \widehat{\boldsymbol{y}}_j\}_{j=1}^{\ell}$, the total number of kernels, i.e., the size of the hidden layer, is user-specified to a value $n_c < \ell$. As a result, the linear system from Eq. (3.4) is overdetermined. In this situation, we solve for q minimum norm solutions corresponding to each column of W, i.e., $W_{:,k} = \text{argmin}_{\tilde{\boldsymbol{w}}} \|Y_{:,k} - D\tilde{\boldsymbol{w}}\|$, where $k = 1, \ldots, q$ and the subscript '$:, k$' means a selection of all row entries in kth column. So, the linear system solve can be viewed as a linear least squares problem.

We resort to solve Eq. (3.4) in a fashion similar to the above interpolation mode. First, a factorization of the distance matrix is performed, incurring computations in $\mathcal{O}(n_c^3)$, followed by subsequent q computations in $\mathcal{O}(n_c^2)$. More precisely, for the q least squares solves, we employ the singular value decomposition of D, and a divide and conquer procedure, by specifying 'gelsd' LAPACK [32] driver in our PyTorch [33] implementation of the network.

3.1.2. Center-specific locally-adaptive kernel widths and center locations via gradient-descent-based optimization

Assume that the network weights W are already known and fixed to those values, we consider the center locations $\{\mathbf{c}_i\}_{i=1}^{n_c}$ and the center-dependent kernel widths $\{\epsilon_i\}_{i=1}^{n_c}$ as the free parameters. Assuming the data pairs $\{\boldsymbol{x}_j, \widehat{\boldsymbol{y}}_j\}_{j=1}^{\ell}$ are available, we formulate an optimization problem and seek to minimize the following cost function:

$$J(\theta) = \frac{1}{\ell q} \sum_{j=1}^{\ell} \|\widehat{\boldsymbol{y}}_j - \boldsymbol{y}(\boldsymbol{x}_j; \theta)\|^2 + \lambda \left(\sum_{i=1}^{n_c} \|\boldsymbol{c}_i\|^2 + \sum_{i=1}^{n_c} \ln(\epsilon_i)^2 \right), \tag{3.8}$$

where $\|\cdot\|$ denotes the l^2 norm and θ refers to the collection of parameters we want to optimize, i.e., $\{\boldsymbol{c}_i, \ln(\epsilon_i)\}_{i=1}^{n_c}$. We intentionally choose to minimize $\ln(\epsilon_i)$, instead of ϵ_i, since the kernel width values could happen to be quite small ($0 < \epsilon_i \ll 1$), resulting in stagnation or very slow updates during optimization. Moreover, this also helps us to maintain positivity of the kernel width values during the optimization procedure.

The first term in the cost function $J(\theta)$ in Eq. (3.8) represents the mean-squared error between the true output data $\widehat{\boldsymbol{y}}$ and the network output $\boldsymbol{y}$, and the second term penalizes the size of the centers and kernel widths. As a result, such a procedure allows us to maintain a good distribution of the kernel widths and centers by avoiding extreme variations between them. The regularization parameter λ can ideally be tuned for optimal recovery of the vector-valued function $\widehat{\boldsymbol{y}}(\boldsymbol{x})$. In all our experiments, we choose it to be 10^{-4}.

For a user-specified hidden layer size, n_c, we specify the initial center locations $\{\mathbf{c}_i\}_{i=1}^{n_c}$ to be n_c internal locations uniformly distributed between $\boldsymbol{x}_{min}$ and $\boldsymbol{x}_{max}$, which are defined as

$$\boldsymbol{x}_{min} := \left[\min_{j \in \{1,2,\ldots,\ell\}} (\boldsymbol{x}_j)_1, \ \min_{j \in \{1,2,\ldots,\ell\}} (\boldsymbol{x}_j)_2, \ \ldots, \ \min_{j \in \{1,2,\ldots,\ell\}} (\boldsymbol{x}_j)_p \right]^\top, \tag{3.9}$$

$$\boldsymbol{x}_{max} := \left[\max_{j \in \{1,2,\ldots,\ell\}} (\boldsymbol{x}_j)_1, \ \max_{j \in \{1,2,\ldots,\ell\}} (\boldsymbol{x}_j)_2, \ \ldots, \ \max_{j \in \{1,2,\ldots,\ell\}} (\boldsymbol{x}_j)_p \right]^\top, \tag{3.10}$$

where $(\boldsymbol{x}_j)_m$ denotes the mth component of $\boldsymbol{x}_j$. The uniform division between $\boldsymbol{x}_{min}$ and $\boldsymbol{x}_{max}$ is performed element-wise.

All the center-specific kernel widths $\{\epsilon_i\}_{i=1}^{n_c}$ are initialized with the same value given by

$$\epsilon_i = \frac{4}{n_c(n_c-1)} \sum_{i',i''=1;\ i' \neq i''}^{n_c} \|\boldsymbol{c}_{i'} - \boldsymbol{c}_{i''}\|, \quad \forall i \in \{1, 2, \ldots, n_c\}. \tag{3.11}$$

This amounts to twice the average distance between all pairs of initialized center locations. The intuition behind such a choice is to have a wide support for each kernel at the onset of optimization, which is derived from the initial distribution of the centers.

In case of $1D$ inputs, i.e., p=1, we can avoid computation of the numerous pairwise distances between all the centers, and quite simply initialize the kernel widths in the following fashion:

$$\epsilon_i = \frac{2(\boldsymbol{x}_{max} - \boldsymbol{x}_{min})}{n_c + 1}, \quad \forall i \in \{1, 2, \dots, n_c\}. \tag{3.12}$$

Along with a fixed value for the weights, the network is constructed with the aforementioned initialization for the center locations and kernel widths. Thereby, the network can be trained for optimal values of $\{\boldsymbol{c}_i, \ln(\epsilon_i)\}_{i=1}^{n_c}$ by using any gradient-descent-based optimizer to minimize the loss $J(\theta)$. We employ the Adam optimizer [34] in PyTorch, with a full batch size, resulting in a single iteration for each epoch of optimization. Although we start with the same values for each kernel width ϵ_i, the optimization procedure renders a locally-adaptive distribution for the widths.

3.1.3. Alternating dual-staged iterative training procedure

We train the KSNN by employing an iterative procedure, alternating between two stages: one stage to solve for the output layer weights while the center locations and kernel widths are fixed, whereas, the other stage to optimize the kernel widths and center locations while the output layer weights are fixed. The detailed procedure during each of the two stages is already discussed in Sections 3.1.1 and 3.1.2. As opposed to directly optimizing for all the parameters in the network, by adhering to such an alternating dual-staged iterative training (ADSIT) approach, we exploit the shallowness of the network architecture (by undertaking efficient linear system solves) and considerably accelerate the overall network training.

We assume that the data pairs $\{\boldsymbol{x}_j, \hat{\boldsymbol{y}}_j\}_{j=1}^{\ell}$ are available. The ADSIT approach commences with initializing the centers $\{\boldsymbol{c}_i\}_{i=1}^{n_c}$ and kernel widths $\{\epsilon_i\}_{i=1}^{n_c}$, followed by setting up the network and propagating the inputs $\{\boldsymbol{x}_j\}_{j=1}^{\ell}$. Then, we obtain the output layer weights W, as defined in Eq. (3.3), by efficiently solving the linear system given in Eq. (3.4). The centers and kernel widths are then updated by minimizing $J(\theta)$ (refer Eq. (3.8)) using a gradient-descent-based optimizer. We alternatively iterate between the previous two steps, for a user-defined number of iterations (*alter_iter*) and update the parameters. After the ultimate adaptation of the centers and kernel widths, a final update of the output layer weights is done, which concludes the network training. The complete ADSIT approach is summarized in Algorithm 1.

While training the KSNN in regression mode, we could adapt the centers and kernel widths both; training the KSNN in interpolation mode, we only need to adapt the kernel widths, since the kernels centers are simply the input data locations. After training, we can evaluate the network at any new input $\boldsymbol{x}$ and obtain an approximation $\boldsymbol{y}(\boldsymbol{x})$ for the vector-valued function $\hat{\boldsymbol{y}}(\boldsymbol{x})$. Please refer Appendix A for an illustration of KSNN interpolation and regression via the ADSIT procedure.

Remark 3.1 (***Regularity of the kernel function and its choice***)**.** The kernel function employed during the construction of any KSNN governs the type of function the network can approximate. For instance, to deal with any potential irregularities in the output function being approximated, instead of using infinitely differentiable kernels like the Gaussian kernel, we can employ Matérn class of kernels [35,36], as listed in Table 3.1, possessing limited differentiability. For specific constructions of the network, we particularly use the Matern12 kernel, Matern32 kernel, and Matern52 kernel which are zero time, one time, and two times differentiable, respectively. Please refer Appendix B for further details about the KSNN setups used for each problem investigated in Section 5.

Remark 3.2 (***Preserving positivity when interpolating error values***)**.** In our work, we repeatedly build or retrain a KSNN to construct an interpolant for the norm of the relative error caused in the POD-approximate solution, which is discussed at length in Section 2. While interpolating these small error values, it could happen that the result is a negative value close to zero, which would be

Algorithm 1 `Alternating dual-staged iterative training (ADSIT) for kernel-based shallow neural networks (KSNNs)`

Input: Data pairs $\{\boldsymbol{x}_j \in \mathbb{R}^p, \hat{\boldsymbol{y}}_j \in \mathbb{R}^q\}_{j=1}^{\ell}$ from ℓ samples of a vector-valued function $\hat{\boldsymbol{y}}(\boldsymbol{x})$, kernel ϕ, number of centers n_c, learning rate lr, number of alternating iterations *alter_iter*, number of epochs *epochs*, $iter = 1$.

Output: Trained network providing the approximation $\hat{\boldsymbol{y}}(\boldsymbol{x}) \approx \boldsymbol{y}(\boldsymbol{x}) \in \mathbb{R}^q$.

1: Initialize the center locations $\{\boldsymbol{c}_i\}_{i=1}^{n_c}$ by an element-wise uniform division for n_c locations between $[\boldsymbol{x}_{min}, \boldsymbol{x}_{max}]$, as defined in Eqs. (3.9) and (3.10).
2: Initialize the kernel widths $\{\epsilon_i\}_{i=1}^{n_c}$ by following Eq. (3.11) or Eq. (3.12).
3: Setup the network and propagate inputs $\{\boldsymbol{x}_j\}_{j=1}^{\ell}$.
4: Collect all the hidden layer activations and create matrix D via Eq. (3.5).
5: **while** $iter \leq alter\_iter$ **do**
6: Update the output layer weights W via efficient solve of linear system Eq. (3.4).
7: Adapt the centers $\{\boldsymbol{c}_i\}_{i=1}^{n_c}$ and scaled kernel widths $\{\ln(\epsilon_i)\}_{i=1}^{n_c}$ by minimizing $J(\theta)$ (refer Eq. (3.8)) using a gradient-descent-based optimizer. Make a total of *epochs* updates with lr as the learning rate per update.
8: Recompute matrix D by using the updated centers and kernel widths.
9: **end while**
10: Re-evaluate the output layer weights W by efficiently solving the linear system Eq. (3.4) with the updated distance matrix D.

nonphysical. This phenomenon is dependent on the distribution of the training data, as well as on the kernel width's (ϵ) value. To ensure the positivity of the error values, we modify the training data by taking the logarithm of all the error values used for training. After querying the network, we need to take the exponential of the result, to obtain the correct (positive) interpolated error value.

3.2. POD-KSNN surrogate model

Consider a successful parameter sampling, followed by snapshot data collection of all the chosen parameter points corresponding to the same time horizon by simulating the high-fidelity model Eq. (2.1). For instance, following the setup for Eq. (2.6), we have the snapshot matrices $U(\boldsymbol{\mu}_i)$. We define a vector-valued function

$$\widehat{\mathbf{y}}_u(\boldsymbol{\mu}) = \begin{bmatrix} \mathbf{u}(t_0, \boldsymbol{\mu}) \\ \mathbf{u}(t_1, \boldsymbol{\mu}) \\ \vdots \\ \mathbf{u}(t_{N_t}, \boldsymbol{\mu}) \end{bmatrix}. \tag{3.13}$$

We follow a two-step interpolation approach to construct the non-intrusive reduced-order surrogate model. In the first step, by building a single KSNN (refer to Eq. (3.2)), we interpolate the snapshot data $\{\widehat{\mathbf{y}}_u(\boldsymbol{\mu}_i)\}_{i=1}^m$ in the parameter space corresponding to all time-instances t_j with $j \in \{0, \ldots, N_t\}$. The KSNN's construction and training is done in the offline phase, whereas in the online phase, it is queried at a new parameter instance $\boldsymbol{\mu}^*$. In this case, $\widehat{\mathbf{y}}(\boldsymbol{x})$ in Eq. (3.1) corresponds to $\widehat{\mathbf{y}}_u(\boldsymbol{\mu})$ in Eq. (3.13), $\mathbf{c}_i$ in Eq. (3.2) are $\boldsymbol{\mu}_i$, and the domain $\boldsymbol{x}$ is $\boldsymbol{\mu}$. The resulting interpolant $\mathcal{I}^{\mu}(\boldsymbol{\mu})$ of $\widehat{\mathbf{y}}_u(\boldsymbol{\mu})$ is denoted as

$$\mathcal{I}^{\mu}(\boldsymbol{\mu}) = \begin{bmatrix} \mathbf{u}^I(t_0, \boldsymbol{\mu}) \\ \mathbf{u}^I(t_1, \boldsymbol{\mu}) \\ \vdots \\ \mathbf{u}^I(t_{N_t}, \boldsymbol{\mu}) \end{bmatrix}. \tag{3.14}$$

The entire estimated snapshot matrix $U^I(\boldsymbol{\mu})$ is

$$U^I(\boldsymbol{\mu}) = [\ \mathbf{u}^I(t_0, \boldsymbol{\mu}) \mid \mathbf{u}^I(t_1, \boldsymbol{\mu}) \mid \ \ldots\ \mid \mathbf{u}^I(t_{N_t}, \boldsymbol{\mu})\ ], \tag{3.15}$$

which is obtained via a single KSNN interpolating the solutions $\{\widehat{\mathbf{y}}_u(\boldsymbol{\mu}_i)\}_{i=1}^m$. Due to this reason, we refer to it as μ-KSNN. It is crucial to note that once μ-KSNN is constructed, we do not need to store any snapshot matrices $U(\boldsymbol{\mu}_i)$ that were used during the training phase.

The optimal linear subspace spanned by the approximate snapshot data at $\boldsymbol{\mu}^*$ can be computed via proper orthogonal decomposition (POD) of $U^I(\boldsymbol{\mu}^*)$. In practice, we can compute the POD bases $\tilde{\boldsymbol{\Phi}}(\boldsymbol{\mu}^*) := (\tilde{\phi}_1(\boldsymbol{\mu}^*), \tilde{\phi}_2(\boldsymbol{\mu}^*), \ldots, \tilde{\phi}_s(\boldsymbol{\mu}^*))$ either using the singular value decomposition or (for large-scale problems) using the method of snapshots, when N is very large and $N_t \ll N$ [37]. The interpolated snapshots at $\boldsymbol{\mu}^*$ can then be written as a linear combination of the bases $\tilde{\boldsymbol{\Phi}}(\boldsymbol{\mu}^*)$. By collecting all the coefficients for such a linear combination in a matrix $A(\boldsymbol{\mu}^*)$, the interpolated snapshots $U^I(\boldsymbol{\mu}^*)$ can be represented as,

$$U^I(\boldsymbol{\mu}^*) = \tilde{\boldsymbol{\Phi}}(\boldsymbol{\mu}^*) A(\boldsymbol{\mu}^*), \tag{3.16}$$

$$A(\boldsymbol{\mu}^*) = [\ \alpha^0(\boldsymbol{\mu}^*) \mid \alpha^1(\boldsymbol{\mu}^*) \mid \ \ldots\ \mid \alpha^{N_t}(\boldsymbol{\mu}^*)\ ], \tag{3.17}$$

where $\alpha^j \in \mathbb{R}^s$ is the vector of coefficients at time t_j. For the complete POD space, s equals to the number of nonzero singular values of $U^I(\boldsymbol{\mu}^*)$. However, in practice, usually the POD space is truncated. In that scenario, s corresponds to the number of retained singular values.

To obtain the approximation of the solution corresponding to $\boldsymbol{\mu}^*$ at a new time instance $t^* \in [0, T]$, a second interpolation step is carried out in the time domain by building and training an additional KSNN (refer to Eq. (3.2)). We construct an interpolant $\mathcal{I}^t_{\mu^*}$ for the reduced coordinates $\alpha^j(\boldsymbol{\mu}^*)$ in the time domain,

$$\alpha^I(t) := \mathcal{I}^t_{\mu^*}(t). \tag{3.18}$$

In this case, $\boldsymbol{y}$ in Eq. (3.1) corresponds to α^I in Eq. (3.18) (an s-dimensional vector), $\mathbf{c}_i$ in Eq. (3.2) is t_j, and the domain $\boldsymbol{x}$ is t. The data pairs $\{\boldsymbol{x}_j, \widehat{\boldsymbol{y}}_j\}_{j=1}^{\ell}$ are $\{t_j, \alpha^j(\boldsymbol{\mu}^*)\}_{j=0}^{N_t}$ for training $\mathcal{I}^t_{\mu^*}$. The surrogate approximation of the solution at any test location $\boldsymbol{\mu}^*$ and t^* is obtained as follows,

$$\mathbf{u}_s(t^*, \boldsymbol{\mu}^*) = \tilde{\boldsymbol{\Phi}}(\boldsymbol{\mu}^*)\ \alpha^I(t^*). \tag{3.19}$$

The complete POD-KSNN surrogate model is summarized in Fig. 3.2. We adhere to a two-step interpolation approach to divide the function complexity between the parameter and time domains, thereby allowing us to work with two KSNNs of reduced sizes: μ-KSNN and t-KSNN. The network size (layer width) is directly related to the number of centers or data points under consideration. More data points lead to a wider KSNN, whose training will require solving a larger linear system for the weights. As the number of centers increases, the memory required for the linear system solve goes up considerably. This is due to the quadratic dependence of the required memory on the number of centers. The training could become infeasible in such a scenario. However, the two-step interpolation strategy enables us to isolate the centers between parameter and time domains, thereby relaxing the total permissible center count.

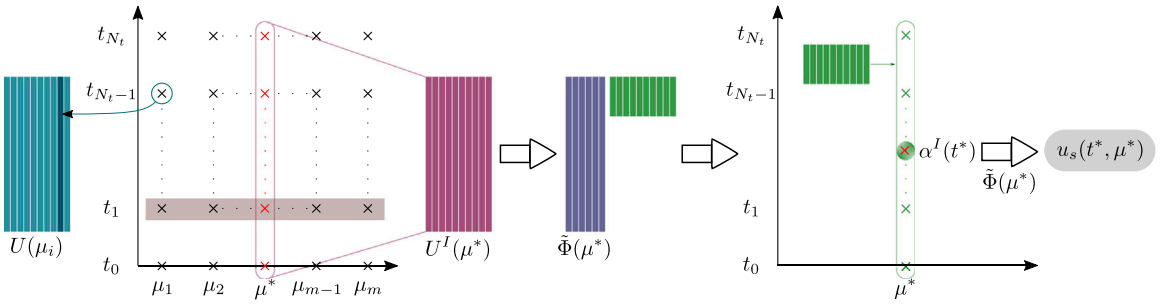


Fig. 3.2. Framework for POD-KSNN data-driven surrogate modeling.

Remark 3.3 (***Extension to system of parametric PDEs***)**.** For problems with more than one PDE, one approach is to prepare different snapshot matrices for all the solution components that are present in the system of equations. Then prepare separate KSNNs for all components to interpolate between the snapshots in the parameter domain. Later, in the second step, we create POD subspaces, individually, for each of the components. An additional KSNN for each component interpolates the reduced coordinates in the time domain. We employ this methodology in our numerical experiments with the shallow water equations.

Another approach is to merge the solutions for all the components and form one big snapshot matrix corresponding to each parameter instance. This allows us to proceed in the same fashion as detailed above in this section. Undertaking this approach of first merging the component snapshots and then applying POD leads to a single ROM, but with larger reduced size.

3.2.1. Additive error decomposition

An important assumption regarding the data-driven surrogate model, made during the construction of the non-intrusive error estimator and the optimality criterion based on it, which is central to the active learning framework, is of the additive error decomposition as given in Eq. (2.2). Since we apply the active learning framework to the POD-KSNN surrogate, it is important to ensure that such an additive error decomposition holds true for it. In the subsequent text, we provide a proof for this.

Theorem 3.1. *For any parametric dynamical system of the form shown in Eq. (2.1), assume that the data-driven surrogate's solution* $\mathbf{u}_s(t, \boldsymbol{\mu}) \in \mathbb{R}^N$ *is given by Eq. (3.19), i.e., POD-KSNN is the surrogate model. In this setting, the total error* $\mathcal{E} \in \mathbb{R}^N$ *of the surrogate in the spatial domain corresponding to some parameter* $\boldsymbol{\mu}$ *and time instance* t *attains an additive decomposition between the POD projection error* $\mathcal{E}_{POD} \in \mathbb{R}^N$ *and the total interpolation error* $\mathcal{E}_\star \in \mathbb{R}^N$ *as shown in Eq. (2.2).*

Proof. Let us start by looking at the error in the surrogate solution $\mathbf{u}_s(t, \boldsymbol{\mu})$,

$$\mathcal{E}(t, \boldsymbol{\mu}) := \mathbf{u}(t, \boldsymbol{\mu}) - \mathbf{u}_s(t, \boldsymbol{\mu}). \tag{3.20}$$

This can be written in the following fashion for the POD-KSNN surrogate model:

$$\mathcal{E} = (\mathbf{u} - \boldsymbol{\Phi}\boldsymbol{\Phi}^\top \mathbf{u}) + (\boldsymbol{\Phi}\boldsymbol{\Phi}^\top \mathbf{u} - \boldsymbol{\Phi}\boldsymbol{\Phi}^\top \mathbf{u}^I) + (\boldsymbol{\Phi}\boldsymbol{\Phi}^\top \mathbf{u}^I - \tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{\Phi}}^\top \mathbf{u}^I) + (\tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{\Phi}}^\top \mathbf{u}^I - \mathbf{u}_s). \tag{3.21}$$

Here, $\boldsymbol{\Phi}(\boldsymbol{\mu}) = (\phi_1(\boldsymbol{\mu}), \phi_2(\boldsymbol{\mu}), \dots, \phi_s(\boldsymbol{\mu}))$ are the bases obtained by performing POD of the solution $\mathbf{u}$; $\mathbf{u}^I(t, \boldsymbol{\mu})$ is an approximation of $\mathbf{u}$ obtained via interpolating the solutions in the parameter domain using μ-KSNN, as shown in Eq. (3.14); $\tilde{\boldsymbol{\Phi}}(\boldsymbol{\mu}) = (\tilde{\phi}_1(\boldsymbol{\mu}), \tilde{\phi}_2(\boldsymbol{\mu}), \dots, \tilde{\phi}_s(\boldsymbol{\mu}))$ are the bases obtained by performing POD of the approximate solution $U^I(\boldsymbol{\mu})$ as given in Eq. (3.15).

By following Eq. (3.19), we can write the surrogate solution as $\mathbf{u}_s(t, \boldsymbol{\mu}) = \tilde{\boldsymbol{\Phi}}(\boldsymbol{\mu})\alpha^I(t)$. Here, an approximation of the reduced solution coordinates, $\alpha^I(t)$, is obtained by interpolating $\{\alpha^j(\boldsymbol{\mu})\}_{j=0}^{N_t}$ from Eq. (3.17) in the time domain using t-KSNN, i.e., $\alpha^I(t) = \mathcal{I}_\mu^t(t)$, similar to Eq. (3.18). This allows us to write Eq. (3.21) in the following way:

$$\mathcal{E} = (\mathbf{u} - \boldsymbol{\Phi}\boldsymbol{\Phi}^\top \mathbf{u}) + (\boldsymbol{\Phi}\boldsymbol{\Phi}^\top \mathbf{u} - \boldsymbol{\Phi}\boldsymbol{\Phi}^\top \mathbf{u}^I) + (\boldsymbol{\Phi}\boldsymbol{\Phi}^\top \mathbf{u}^I - \tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{\Phi}}^\top \mathbf{u}^I) + (\tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{\Phi}}^\top \mathbf{u}^I - \tilde{\boldsymbol{\Phi}}\alpha^I). \tag{3.22}$$

The first term in Eq. (3.22) arises due to the retention of only the leading s POD modes. This results in a parameter-specific linear subspace that captures most of the solution, but not in its entirety. We refer to this omitted contribution as the POD approximation error and define it as,

$$\mathcal{E}_{POD} := (\mathbf{u} - \boldsymbol{\Phi}\boldsymbol{\Phi}^\top \mathbf{u}). \tag{3.23}$$

The second and third terms in Eq. (3.22) arise from the solution approximation $\boldsymbol{u}^I$, due to interpolation in the parameter domain. More precisely, the second term accounts for the solution error resulting from the reduced representation of the approximate solution obtained by projection onto the true solution bases $\boldsymbol{\Phi}$. Whereas, the third term accounts for the solution error caused due to projection of the approximate solution $\mathbf{u}^I$ onto the bases $\tilde{\boldsymbol{\Phi}}$ instead of $\boldsymbol{\Phi}$, which is obtained from a POD of $\mathbf{u}^I$. Finally, the fourth term in Eq. (3.22) provides the solution error caused because of an approximation of the reduced coordinates α^I via interpolation in the time domain. As a result, we can define the total interpolation error in the following way,

$$\mathcal{E}_\star := (\boldsymbol{\Phi}\boldsymbol{\Phi}^\top \mathbf{u} - \boldsymbol{\Phi}\boldsymbol{\Phi}^\top \mathbf{u}^I) + (\boldsymbol{\Phi}\boldsymbol{\Phi}^\top \mathbf{u}^I - \tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{\Phi}}^\top \mathbf{u}^I) + (\tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{\Phi}}^\top \mathbf{u}^I - \mathbf{u}_s). \tag{3.24}$$

Based on Eqs. (3.22)–(3.24), we can clearly deduce that Eq. (2.2) holds. □

Remark 3.4 (***Regarding the balance between the POD projection error and the interpolation error for POD-KSNN***)**.** During the construction of POD-KSNN, if μ-KSNN and t-KSNN are trained in interpolation mode, the POD approximation error $\{E(\boldsymbol{\mu}_i)\}_{i=1}^m$ as defined in Eq. (2.10) is also the POD-KSNN surrogate error at the training parameter samples $\{\boldsymbol{\mu}_i\}_{i=1}^m$. This is because the surrogate approximation of the snapshot matrix for $\{\boldsymbol{\mu}_i\}_{i=1}^m$ becomes $U_{surrogate}(\boldsymbol{\mu}_i) = U_{POD}(\boldsymbol{\mu}_i)$, since the interpolation procedure reproduces the training data. Consequently, for all the training parameter samples, this results in $\epsilon_\star = 0$, where $\epsilon_\star$ is as appearing in Eq. (2.5). This makes ϵ_{POD} the bound for the total error ϵ. As a result, the non-intrusive error estimator $\tilde{\boldsymbol{\varepsilon}}(\boldsymbol{\mu}^*)$ from Eq. (2.14) indeed provides an estimation of the surrogate error. Moreover, due to the KSNN construction using the ADSIT procedure, the locally adaptive kernel widths help us construct accurate μ-KSNN and t-KSNN, potentially rendering low interpolation error $\epsilon_\star$ also for out-of-training parameter and time locations.

4. ActLearn-POD-KSNN surrogate model

We summarize the complete methodology to construct and deploy the ActLearn-POD-KSNN reduced-order surrogate model in this section. Algorithm 2 details the complete offline phase. To iteratively construct the non-intrusive error estimator, a new KSNN Eq. (3.2) is automatically built, trained, and queried at steps 6, 12, 22, and 24 of the algorithm. The parameter set P is actively updated during the offline phase and solution snapshots corresponding to actively selected parameter values are adaptively generated.

Algorithm 2 `Offline phase: ActLearn-POD-KSNN construction`

Input: Initial parameter set P, snapshots $U(\boldsymbol{\mu}_i)$ for $\boldsymbol{\mu}_i \in P$, candidate parameter set P^*, initial uniform energy criterion η, tolerance (tol) for termination of the active learning loop, $iter = 0$.
Output: Updated set P, energy criterion $\hat{\eta}$, trained μ-KSNN ($\mathcal{I}^\mu$).
1: Using η, evaluate truncation level $r_i, \forall \boldsymbol{\mu}_i \in P$, such that Eq. (2.18) holds.
2: Create truncated parameter-specific POD subspaces $\boldsymbol{\Phi}(\boldsymbol{\mu}_i), \forall \boldsymbol{\mu}_i \in P$.
3: Construct POD approximate solutions $U_{POD}(\boldsymbol{\mu}_i)$ in Eq. (2.9).
4: Obtain error snapshots $E(\boldsymbol{\mu}_i)$ by following Eqs. (2.10) and (2.19).
5: Compute maximal relative error $\hat{\varepsilon}(\boldsymbol{\mu}_i), \forall \boldsymbol{\mu}_i \in P$, Eq. (2.13).
6: Construct AL-KSNN in Eq. (2.16) and compute optimality criterion $\hat{c}(\bar{\boldsymbol{\mu}}_j), \forall \bar{\boldsymbol{\mu}}_j \in P^*$, by following Eq. (2.15).
7: Pick a parameter $\bar{\boldsymbol{\mu}}^{(iter)}$ from P^* via Eq. (2.21).
8: Pick a parameter $\mathring{\boldsymbol{\mu}}$ from P via Eq. (2.22).
9: **while** $\hat{\varepsilon}(\mathring{\boldsymbol{\mu}}) > \hat{c}(\bar{\boldsymbol{\mu}}^{(iter)})$ **do**
10: Enrich $\boldsymbol{\Phi}(\mathring{\boldsymbol{\mu}})$ with the next dominant POD basis vector, and update the energy criterion $\eta(\mathring{\boldsymbol{\mu}})$.
11: Recompute $U_{POD}(\mathring{\boldsymbol{\mu}})$, $E(\mathring{\boldsymbol{\mu}})$, and $\hat{\varepsilon}(\mathring{\boldsymbol{\mu}})$.
12: Reconstruct AL-KSNN in Eq. (2.16) and recompute $\hat{c}(\bar{\boldsymbol{\mu}}_j), \forall \bar{\boldsymbol{\mu}}_j \in P^*$.
13: Re-select $\bar{\boldsymbol{\mu}}^{(iter)}$ from P^* Eq. (2.21) and $\mathring{\boldsymbol{\mu}}$ from P Eq. (2.22).
14: **end while**
15: Store $\mathcal{E}^{(iter)} = \hat{c}(\bar{\boldsymbol{\mu}}^{(iter)})$.
16: **while** $\mathcal{E}^{(iter)} > tol$ **do**
17: Extend P by adding $\bar{\boldsymbol{\mu}}^{(iter)}$. Update candidate set, $P^* = P^* \setminus \bar{\boldsymbol{\mu}}^{(iter)}$.
18: Compute the solution snapshots $U(\bar{\boldsymbol{\mu}}^{(iter)})$ using a high-fidelity model.
19: Based on η, calculate the POD truncation level for $\bar{\boldsymbol{\mu}}^{(iter)}$.
20: Construct $\boldsymbol{\Phi}(\bar{\boldsymbol{\mu}}^{(iter)})$, $U_{POD}(\bar{\boldsymbol{\mu}}^{(iter)})$, and $E(\bar{\boldsymbol{\mu}}^{(iter)})$.
21: $iter = iter + 1$.
22: Reconstruct AL-KSNN in Eq. (2.16) and recompute $\hat{c}(\bar{\boldsymbol{\mu}}_j), \forall \bar{\boldsymbol{\mu}}_j \in P^*$.
23: Pick parameters $\bar{\boldsymbol{\mu}}^{(iter)}$ from P^* Eq. (2.21) and $\mathring{\boldsymbol{\mu}}$ from P Eq. (2.22).
24: **POD subspace adaptation**: *Execute the while loop in steps* $9-14$.
25: Store $\mathcal{E}^{(iter)} = \hat{c}(\bar{\boldsymbol{\mu}}^{(iter)})$.
26: **end while**
27: Compute $\hat{\eta}$ following Eq. (4.1).
28: μ-KSNN: Using $U(\boldsymbol{\mu}_i), \forall \boldsymbol{\mu}_i \in P$, as training data, construct $\mathcal{I}^\mu$ in Eq. (3.14) by building and training a KSNN Eq. (3.2).

Since the parameter-specific POD subspaces are refined adaptively during the active learning process, the energy criteria corresponding to the retained POD modes could be different for the selected parameters in the final pool of P. We denote it by $\eta(\boldsymbol{\mu}_i)$ for each $\boldsymbol{\mu}_i \in P$. The minimum energy criterion would be

$$\hat{\eta} = \min_{\boldsymbol{\mu} \in P} \eta(\boldsymbol{\mu}). \tag{4.1}$$

We adhere to this updated energy criterion, $\hat{\eta}$, to construct the POD subspace $\tilde{\boldsymbol{\Phi}}(\boldsymbol{\mu}^*)$, with r^* bases, for any new parameter $\boldsymbol{\mu}^*$ during the online phase. Building upon the successful active learning procedure carried out during the offline phase, we acquire the

surrogate solution at a new time t^* and parameter $\boldsymbol{\mu}^*$ value in the online phase using Algorithm 3. We do not query the full-order model in the online phase.

Algorithm 3 `Online phase: ActLearn-POD-KSNN query`

Input: New parameter $\boldsymbol{\mu}^*$, new time t^*, energy criterion $\hat{\eta}$, μ-KSNN ($\mathcal{I}^{\mu}$).
Output: Surrogate solution at (t^*, $\boldsymbol{\mu}^*$).

1: Evaluate $U^I(\boldsymbol{\mu}^*)$, the KSNN approximate snapshots for $\boldsymbol{\mu}^*$, on the training time grid via Eqs. (3.2) and (3.15).
2: Compute POD bases $\tilde{\boldsymbol{\Phi}}(\boldsymbol{\mu}^*)$ by deciding the truncation level from the energy criterion $\hat{\eta}$ Eq. (4.1).
3: Project $U^I(\boldsymbol{\mu}^*)$ on $\tilde{\boldsymbol{\Phi}}(\boldsymbol{\mu}^*)$ to obtain reduced coordinates $A(\boldsymbol{\mu}^*)$ Eq. (3.17).
4: t-KSNN: Using $A(\boldsymbol{\mu}^*)$ as the training data, construct $\mathcal{I}^t_{\mu^*}$ in Eq. (3.18) by building and training a KSNN Eq. (3.2).
5: Obtain a vector of approximate reduced coordinates, $\alpha^I(t^*)$, at t^* by evaluating $\mathcal{I}^t_{\mu^*}(t^*)$ Eq. (3.18).
6: Compute the surrogate solution $\mathbf{u}_s(t^*, \boldsymbol{\mu}^*)$ using Eq. (3.19).

Remark 4.1 (***Error estimation for the ActLearn-POD-KSNN surrogate in the online phase***)**.** The ultimate construction of AL-KSNN Eq. (2.16), prior to termination of the active learning loop (refer Algorithm 2), is stored. Later, in the online phase, this stored AL-KSNN can be queried at any test parameter samples $\boldsymbol{\mu}^*$ to obtain an estimate of the relative error behavior along the time domain, corresponding to discrete time instances used during the training, i.e., the training time grid. Note that this error estimation is a by-product of the offline active learning procedure. The estimates of the surrogate error along the time domain reported for the examples in Section 5 are evaluated in this fashion.

4.1. Complexity analysis

We report the computational complexity for the offline and online phases of the ActLearn-POD-KSNN surrogate by considering the dominant cost associated to each stage. The offline phase costs can be roughly divided into the following three parts:

- Part A: This includes steps 1 – 26 of Algorithm 2, excluding step 18 concerning evaluation of the high-fidelity model. Here, the dominant cost corresponds to computing the POD subspaces $\boldsymbol{\Phi}(\boldsymbol{\mu}_i), \forall \boldsymbol{\mu}_i \in P$, and the repeated construction of AL-KSNN Eq. (2.16).
- Part B: This concerns the evaluation of the high-fidelity model at a new parameter sample $\bar{\boldsymbol{\mu}}^{(iter)}$ during each iteration, i.e., step 18 of Algorithm 2.
- Part C: This accounts for the construction of μ-KSNN, i.e., step 28 of Algorithm 2.

Similarly, the online phase costs can be divided into the following two parts:

- Part D: This includes steps 1 – 3 of Algorithm 3, with the dominant cost corresponding to computation of POD bases $\tilde{\boldsymbol{\Phi}}(\boldsymbol{\mu}^*)$.
- Part E: This includes steps 4 – 6 of Algorithm 3, with the dominant cost corresponding to construction of t-KSNN.

To ease our further discussions, let us first highlight some computational costs:

- $C_{\text{FOM}} := N_t \cdot \mathcal{O}(N^2)$ gives the cost of evaluating the full-order model Eq. (2.1) for a given parameter sample, by employing some iterative scheme like the conjugate gradient method or the generalized minimal residual method.
- $C_{\text{SVD}} := \mathcal{O}(N N_t^2)$ gives the cost for performing the SVD of a parameter-specific snapshot matrix.
- $C_{\text{KSNN}} := \; alter\_iter \cdot \left(C_{\text{LU}} + n_{out} \cdot C_{\text{f-b}}\right) + alter\_iter \cdot epochs \cdot C_{\text{gd}}^{n_{\text{features}}}$ gives the cost of constructing a KSNN via Algorithm 1. Here, $C_{\text{LU}} := \mathcal{O}(n_c^3)$ is the cost of a LU decomposition, $C_{\text{f-b}} := \mathcal{O}(n_c^2)$ is the cost of a forward–backward substitution to solve a linear system when the LU factors are available, and $C_{\text{gd}}^{n_{\text{features}}}$ represents a single descent step cost of a gradient-descent-based optimizer while optimizing for $n_{\text{features}} = 2n_c$ number of features, i.e., the center locations and their kernel widths. Moreover, n_{out} is the output layer dimension of the KSNN, appearing as q in Algorithm 1; n_c, $alter\_iter$, and $epochs$ are same as in Algorithm 1.

The cost associated with Part A, viz., C_A is

$$C_A := \left(n_{init} + n_{iter}\right) \cdot C_{\text{SVD}} + C_{\text{AL-KSNN}}, \tag{4.2}$$

where n_{init} represents the number of parameters in the initial parameter set P, and n_{iter} gives the total number of active learning iterations until the user-defined tolerance is satisfied. We represent the total cost towards construction of AL-KSNN by $C_{\text{AL-KSNN}}$. Since we repeatedly reconstruct AL-KSNN between the active learning iterations, as well as during each iteration, after every POD subspace enrichment, $C_{\text{AL-KSNN}}$ cannot be directly interpreted as C_{KSNN}, but takes the following form:

$$C_{\text{AL-KSNN}} = \sum_{j=0}^{n_{iter}} n_{adapt}^{(j)} \cdot alter\_iter \cdot \left[\mathcal{O}((n_{init}+j)^3) + N_t \cdot \mathcal{O}((n_{init}+j)^2) + epochs \cdot C_{gd}^{n_{init}+j}\right], \tag{4.3}$$

where $n_{adapt} = \sum_{j=0}^{n_{iter}} n_{adapt}^{(j)}$ represents the total number of POD subspace adaptations over all the iterations of the active learning procedure, with $n_{adapt}^{(j)}$ corresponding to jth iteration. Note that we do not have a-priori knowledge of n_{adapt}. The detailed AL-KSNN

cost is for its construction in interpolation mode, where the centers are taken as the available parameter samples at the specific active learning iteration, and the features optimized by the gradient-descent-based optimizer are the kernel widths. Moreover, the term $(n_{init} + j)$ in Eq. (4.3) accounts for the increment in AL-KSNN's hidden layer size n_c as the training set grows during active learning.

The cost associated with Part B, viz., C_B is

$$C_B := \left(n_{init} + n_{iter}\right) \cdot C_{\text{FOM}}. \tag{4.4}$$

And the cost associated with Part C, viz., C_C is

$$C_C := C_{\mu\text{-KSNN}}, \tag{4.5}$$

where the detailed cost for constructing μ-KSNN can be given by C_{KSNN} with the specific choice $n_{out} = N \cdot N_t$. Moreover, $n_c = \left(n_{init} + n_{iter}\right)$ should be used in the definitions of C_{LU} and $C_{\text{f-b}}$, and $n_{features} = \left(n_{init} + n_{iter}\right)$ for $C_{\text{gd}}^{n_{\text{features}}}$, as we train for the kernel widths.

In the online phase, the cost associated with Part D, viz., C_D is

$$C_D := C_{\text{SVD}}, \tag{4.6}$$

And the cost C_E associated with Part E is

$$C_E := C_{t\text{-KSNN}}, \tag{4.7}$$

where the detailed cost for constructing t-KSNN can be given by C_{KSNN} with the specific choice $n_{out} = r^*$, with r^* being the size of bases $\tilde{\boldsymbol{\Phi}}(\boldsymbol{\mu}^*)$. Moreover, $n_c = N_t$ should be used in the definitions of C_{LU} and $C_{\text{f-b}}$, and $n_{features} = N_t$ for $C_{\text{gd}}^{n_{\text{features}}}$, when only the kernel widths are trained.

Note that the choice of *alter_iter* and *epochs* varies for AL-KSNN, μ-KSNN, and t-KSNN. During our discussions above, for better readability, we avoid having an explicit distinction between these values for each of the KSNN constructions. Their exact values, specific to the various KSNN constructions, for all of the examples reported in Section 5, is provided in Appendix B.

Although the online computational cost depends on the dimension of the full-order model, N, the online phase is fast, as evident from the numerical results presented in Section 5. This is because only a single operation depending on N is required, in contrast to N_t operations depending on N, as required by a FOM evaluation.

Remark 4.2 (***t*-KSNN in regression mode**). When N_t is very large, it becomes beneficial to build the t-KSNN in regression mode, as it reduces the online phase cost associated to Part E, i.e., C_E. In such a situation, $n_c = n_c^{\text{time}}$, where n_c^{time} is a user-defined number of centers in the temporal domain, such that $n_c^{\text{time}} \ll N_t$. In this case, both the center locations and kernel widths could be learned, such that $n_{features} = 2n_c^{\text{time}}$.

Remark 4.3 (***Timings reported in Section* 5**). The execution times reported in Section 5 for parts A–E include all the relevant steps from Algorithms 2 and 3, as described at the beginning of this subsection on complexity analysis, and do not simply correspond to the dominant costs highlighted for each part during our detailed analysis.

4.2. An extension for noisy data

We investigate a setting where the solution data available from the parametric dynamical system is noisy. Particularly, we are interested in situations where the solution fields contain outliers and corrupt values that differ from the underlying distribution of true data, usually referred to as salt-and-pepper noise. We leverage existing work on robust principal component analysis [38] to extract an approximation of the true solution as a low-rank matrix and the noise as a sparse matrix. This allows us to obtain approximate noise-free POD modes which are essential to evaluate the error-estimator-based optimality criterion. Moreover, since as part of the surrogate construction, we interpolate between the true solution in the parameter domain, availability of approximate noise-free solution at training parameter locations also become crucial.

Without a-priori knowledge of the rank or sparsity pattern, decomposing a matrix into low-rank and sparse factors is challenging. In fact, the problem is not tractable in its general form. So, authors in [39] suggested to extract the low-rank and sparse factors by reformulating the problem as a convex optimization problem, referred to as *principal component pursuit*. It can be written in the following fashion:

$$\min_{L,S} \|L\|_* + \gamma \|S\|_1 \text{ such that } L + S = \tilde{U}, \tag{4.8}$$

where $\tilde{U} \in \mathbb{R}^{N\times(N_t+1)}$ is the noisy snapshot matrix, $L \in \mathbb{R}^{N\times(N_t+1)}$ is the low-rank part of $\tilde{U}$, $S \in \mathbb{R}^{N\times(N_t+1)}$ is the sparse part of $\tilde{U}$, $\|\cdot\|$ is the l^1 norm, $\|\cdot\|_*$ is the nuclear norm, given by the sum of singular values, and $\gamma > 0$ gives a relative weight between the low-rank and sparse parts.

The augmented Lagrangian function of Eq. (4.8) can be written as,

$$\mathcal{L}(L, S, Z) := \|L\|_* + \gamma\|S\|_1 - \langle Z, L + S - \tilde{U}\rangle + \frac{\beta}{2}\|L + S - \tilde{U}\|_F, \tag{4.9}$$

Algorithm 4 Robust POD via the Alternating Direction Method

Input: Noisy snapshot matrix $\tilde{U} \in \mathbb{R}^{N\times(N_t+1)}$, number of iterations *num_iter*, $iter = 0$.
Output: Low rank factor $L \in \mathbb{R}^{N\times(N_t+1)}$, sparse factor $S \in \mathbb{R}^{N\times(N_t+1)}$.

1: $\beta = \frac{N(N_t+1)}{4\|\tilde{U}\|_1}, \gamma = \frac{1}{\sqrt{\max(N,(N_t+1))}}$.
2: Initialize $L^{(iter)} = S^{(iter)} = Z^{(iter)} = 0 \in \mathbb{R}^{N\times(N_t+1)}$.
3: **while** $iter < num\_iter$ **do**
4: $X = \tilde{U} - S^{(iter)} + \frac{1}{\beta} Z^{(iter)}$.
5: Singular value decomposition: $X = U_s^{(iter)} \Sigma^{(iter)} (V_s^{(iter)})^\top$.
6: $L^{(iter+1)} = U_s^{(iter)} S_{\frac{1}{\beta}}[\Sigma^{(iter)}](V_s^{(iter)})^\top$.
7: $Y = \tilde{U} - L^{(iter+1)} + \frac{1}{\beta} Z^{(iter)}$.
8: $S^{(iter+1)} = S_{\frac{\gamma}{\beta}}[Y]$.
9: $Z^{(iter+1)} = Z^{(iter)} - \beta(L^{(iter+1)} + S^{(iter+1)} - \tilde{U})$.
10: $iter = iter + 1$.
11: **end while**

where $Z \in \mathbb{R}^{N\times(N_t+1)}$ is the Lagrange multiplier of the linear constraint, $\langle\cdot\rangle$ is the standard trace inner product, $\|\cdot\|_F$ is the Frobenius norm, and $\beta > 0$ is the penalty parameter for the violation of the linear constraint. Using the augmented Lagrangian method (ALM) [40,41], it is possible to iteratively solve for the low rank and sparse parts, until convergence, by first minimizing $\mathcal{L}$ to obtain $L^{(iter)}$, $S^{(iter)}$, and then updating the multipliers,

$$Z^{(iter+1)} = Z^{(iter)} - \beta(L^{(iter)} + S^{(iter)} - \tilde{U}). \tag{4.10}$$

Authors in [42] follow such an approach to remove salt-and-pepper noise from fluid flow particle image velocimetry measurements, and perform modal analysis of the flows, including POD and dynamic mode decomposition [43,44].

Instead of ALM, the alternating direction method (ADM) [40,45,46] exploits the separable structure present in the objective function, as well as in the constraint. We resort to ADM, and minimize for L and S sequentially in the following fashion,

$$L^{(iter+1)} \in \underset{L\in\mathbb{R}^{N\times(N_t+1)}}{\operatorname{argmin}} \mathcal{L}(L, S^{(iter)}, Z^{(iter)}), \tag{4.11}$$

$$S^{(iter+1)} \in \underset{S\in\mathbb{R}^{N\times(N_t+1)}}{\operatorname{argmin}} \mathcal{L}(L^{(iter+1)}, S, Z^{(iter)}), \tag{4.12}$$

$$Z^{(iter+1)} = Z^{(iter)} - \beta(L^{(iter+1)} + S^{(iter+1)} - \tilde{U}). \tag{4.13}$$

Consider a soft-thresholding or shrinkage operator, given by

$$S_\tau[C] := \begin{cases} C_{ij} - \tau & \text{if } C_{ij} > \tau, \\ C_{ij} + \tau & \text{if } C_{ij} < -\tau, \\ 0 & \text{otherwise,} \end{cases} \tag{4.14}$$

where $C \in \mathbb{R}^{N\times(N_t+1)}$, C_{ij} is its ij^{th} entry, and $\tau > 0$. The solution of optimization problems outlined in Eqs. (4.11) and (4.12) can be obtained by iterative soft-thresholding [46] of matrices $\Sigma^{(iter)}$ and Y, as outlined in steps 6 and 8 of Algorithm 4, followed by iterative updates of the Lagrange multiplier using Eq. (4.13). The specific choice of parameters β and γ, appearing in Algorithm 4, follows from [46] and [38], respectively.

Algorithm 4 summarizes the complete robust POD algorithm used to clean noisy snapshot matrices in the offline phase of ActLearn-POD-KSNN. The denoising step is invoked each time a new noisy snapshot matrix $\tilde{U}(\bar{\boldsymbol{\mu}}^{(iter)})$ is obtained for a new parameter sample $\bar{\boldsymbol{\mu}}^{(iter)}$ during the active learning loop. As a result, we then obtain a low-rank factor $L(\bar{\boldsymbol{\mu}}^{(iter)})$ comprising the parameter-specific solution information along the complete time trajectory. We take this as an approximation of the true solution $U(\bar{\boldsymbol{\mu}}^{(iter)})$, i.e., $U(\bar{\boldsymbol{\mu}}^{(iter)}) \approx L(\bar{\boldsymbol{\mu}}^{(iter)})$. As a result, the POD modes $\Phi(\bar{\boldsymbol{\mu}}^{(iter)})$ required to obtain the error snapshots $E(\bar{\boldsymbol{\mu}}^{(iter)})$ for computing the error-estimator-based optimality criterion are essentially the clean POD modes of $L(\bar{\boldsymbol{\mu}}^{(iter)})$. The online phase of ActLearn-POD-KSNN stays unchanged for the setting of noisy data.

5. Numerical results

We validate the proposed active learning framework with POD-KSNN reduced-order surrogate models by performing numerical experiments on four test cases. The first test case is the Burgers' equation, which is parametrized by the viscosity. Given a low enough viscosity value, it is known to develop an advecting shock in finite time, even when starting with smooth solutions. Additionally, we also parametrize the initial condition with viscosity. The second test case is the shallow water equation, which is parametrized

by the viscosity and the mean-free path. It is used to model the flow under a pressure surface in a fluid. Its solution comprises two waves moving with opposing characteristic speeds. Due to a periodic boundary condition, the traveling waves repeatedly interact with each other over time.

The third test case is the $2D$ parabolic thermal block equation, comprising a multi-dimensional parameter space, reflecting the variation in heat conductivity values across the various spatial subdomains in the problem. Due to the spatially varying heat conductivity, the solution profile exhibits parameter-varying jumps around the subdomains. The fourth test case is the incompressible Navier–Stokes equation, parametrized by the Reynolds number. More specifically, we consider the fluid behavior in two laminar flow regimes, where the switch follows a Hopf bifurcation. Moreover, with the extension proposed in Section 4.2, we also study the performance of ActLearn-POD-KSNN surrogate when the Navier–Stokes solution data is corrupted by noise.

In the remainder of this section, we provide details about all the problem setups and our results for them. Please refer Appendix B for further details about the KSNN setups used for each problem. Appendix C also provides a comparison between ActLearn-POD-KSNN and POD-KSNN.

5.1. Burgers' equation

We consider the following viscous Burgers' equation in a 1D spatial domain with Dirichlet boundary conditions:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \tag{5.1}$$

where the solution is denoted by u, changing with time $t \geq 0$, space $x \in [0, 1]$, and viscosity ν. The viscosity can be viewed as a parameter that controls the competing effects of diffusion and advection, resulting in a varying solution behavior for a wide range of ν. We choose the following initial condition, such that the solution space at $t = 0$ is also parameter-dependent:

$$u(x,0) = \frac{x}{1 + \sqrt{\frac{1}{\kappa'}}\exp\left(\mathrm{Re}\,\frac{x^2}{4}\right)}; \qquad \kappa' = \exp(Re/8), \quad Re = 1/\nu. \tag{5.2}$$

Instead of simulating the equation using a numerical discretization technique, we opt to utilize the exact solution by converting Eq. (5.1) into a parabolic nonlinear PDE through the application of the Cole–Hopf transformation [47]. The exact solution takes the following form:

$$u(x,t) = \frac{\frac{x}{t+1}}{1 + \sqrt{\frac{t+1}{\kappa'}}\exp\left(\mathrm{Re}\,\frac{x^2}{4t+4}\right)}; \qquad \kappa' = \exp(Re/8), \quad Re = 1/\nu, \tag{5.3}$$

where we refer to Re as the Reynolds number, denoting the inverse of viscosity.

For our experiments, the spatial domain has 150 grid nodes, and the time-domain is $t \in [0, 2]$ with 100 time-steps. The parametric range for the Reynolds number is taken from 10 to 5500. The total number of discrete parameters we consider when accounting for both the candidate set P^* and the parameter set P are 100. These Re values correspond to 100 evenly distributed logarithmic ν values in $\left[log_{10}(\frac{1}{5500}), log_{10}(\frac{1}{10})\right]$, ensuring a decent pool of parameters.

To begin the active learning procedure, the parameter set P is initiated by 21 viscosity values corresponding to the following indices,

$$\{0, 99, 10, 20, 30, 40, 50, 60, 70, 80, 90, 5, 15, 25, 35, 45, 55, 65, 75, 85, 95\}.$$

Here, the ν values are ordered from lowest to highest and the index starts from 0 when counting the 100 values. We report the indices instead of exact values for ease of readability. High-fidelity snapshots are generated for all the viscosities in P at 100 instances of $t \in [0, 2]$. Using these snapshots, POD-approximate solutions are computed such that the POD subspace retains 99.99% of the energy, i.e., $\eta(\nu) = 10^{-4}$, $\forall \nu \in P$.

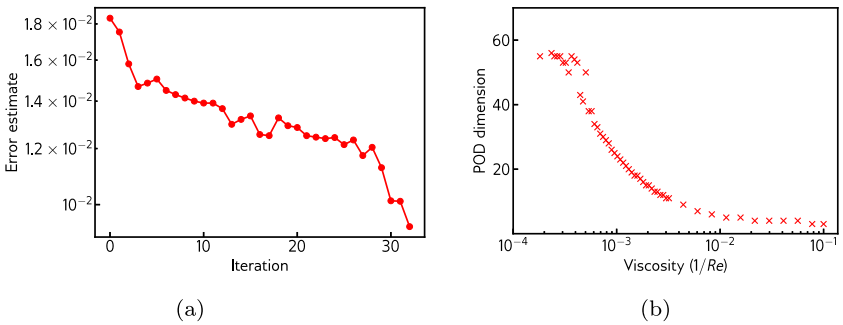


Fig. 5.1. Active learning for Burgers' equation: (a) shows the optimality criterion $\mathcal{E}^{(iter)}$ (refer Algorithm 2) varying with greedy iterations of the active learning process; (b) shows the final POD subspace dimension for each of the chosen parameter samples.

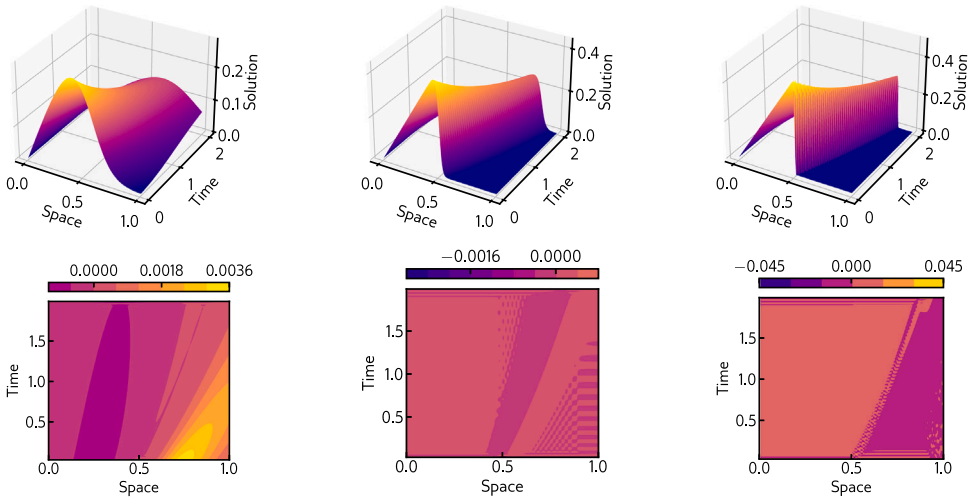


Fig. 5.2. Burgers' equation: The ActLearn-POD-KSNN solution (shown in first row), and the solution error (shown in second row). The error values correspond to the point-wise difference in the space–time domain between the ActLearn-POD-KSNN solution and the true solution. The Re values going from left to right in the rows are in the following order: $\{40, 350, 3000\}$; and their respective POD subspace dimensions are $\{8, 31, 58\}$. All these Re values and discrete time instances required to generate the plots are outside of the training set.

During the active learning loop, we sample from the candidate set P^* at each iteration. The initial state of P^* for the reported results comprises 79 candidate values—upon exclusion of 21 values present in P (initially) from the total 100 values. Fig. 5.1 sheds light on the outcome of the active learning procedure. More specifically, Fig. 5.1(a) shows the optimality criterion $\mathcal{E}^{(iter)}$ (see Algorithm 2) varying with greedy iterations of the active learning process. $\mathcal{E}^{(iter)}$ decreases over iterations, and we stop the loop after a tolerance of 10^{-2} is met. Upon termination, a total of 33 new parameter points are selected in a greedy fashion.

The ultimate choice of viscosity values and their corresponding POD subspace dimensions are shown in Fig. 5.1(b). The reported dimensions also account for the POD space adaptation (refinement) when required through the iterations, as discussed in Section 2.2. For low viscosities, the subspace dimension is comparatively higher. And new selections are mostly concentrated in regions where the nature of the POD subspace changes significantly. Among all the individual parametric POD subspaces, the lowest energy criterion $\hat{\eta}$ in Eq. (4.1) is 2.676×10^{-11}. This is used as a condition for deciding the POD subspace dimension for any newly queried parameter.

Fig. 5.2 shows the solution of the ActLearn-POD-KSNN surrogate model over the entire space–time domain, allowing us to see how the solution evolves over time. The Reynolds numbers $\{40, 100, 350, 1250, 3000\}$ are taken outside the training set, and the POD subspace dimensions corresponding to these Reynolds numbers are $\{8, 13, 31, 58, 58\}$. The solution is computed on a new test time grid with 99 instances starting from 0.01 with a step size of 0.02. The error contours in the second row of Fig. 5.2 show the point-wise error in the space–time domain between the surrogate solution and the ground truth. The solution error stays reasonable for the entire range of Re values.

Fig. 5.3 shows the behavior of true and estimated relative error values along the time domain. More specifically, to obtain an estimate of the relative solution error in the surrogate's approximation at a newly queried parameter μ^*, following Remark 4.1,

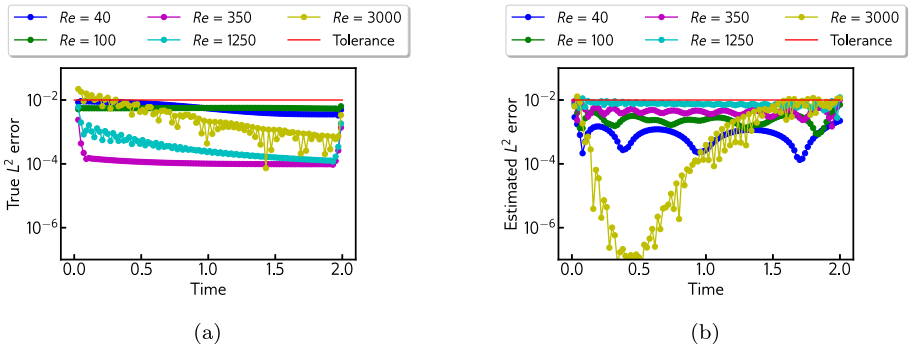


Fig. 5.3. Plot (a) shows the true error of the ActLearn-POD-KSNN solution for the Burgers' equation on a new time grid corresponding to several out-of-training Reynolds numbers. The tolerance used for termination of the active learning procedure is also shown for comparison. Plot (b) shows the estimated error on the original time grid.

we query the AL-KSNN Eq. (2.16). Doing this, we obtain an error estimate $\tilde{\epsilon}(\mu^*)$ for all the time instances in Eq. (2.14). Such an estimation of the error is shown in Fig. 5.3(b) for the training (original) time grid corresponding to several out-of-training Reynolds number. Whereas, Fig. 5.3(a) shows the true relative error values of the ActLearn-POD-KSNN solution. Here, the time instances are different from the training time grid. For the most part, the true relative errors are bounded by the tolerance criterion 10^{-2} which is used for the active learning loop.

5.2. Shallow water equations

The free-surface flows in water bodies like channels or rivers can be modeled using the shallow water equations [48]. They are obtained from the incompressible Navier–Stokes equations under the condition that the fluid flow's vertical extent is significantly smaller than its horizontal extent. The conservation of mass and momentum takes the following form:

$$\partial_t h + \partial_x(hu) = 0, \tag{5.4}$$

$$\partial_t(hu) + \partial_x\left(hu^2 + \frac{1}{2}gh^2\right) = -\frac{\nu}{\lambda}u, \tag{5.5}$$

where $h(t,x)$ is the depth of the channel, $u(t,x)$ is the depth-averaged velocity along the length of the channel, ν is the dynamic viscosity, λ is the mean-free path, g is the gravitational acceleration. This form of the shallow water equations is also known as the Saint-Venant system, which naturally allows us to capture a constant vertical velocity profile in a channel. To carry out the experiments, the initial condition of the height is taken as a smooth bump described by the following nonlinear function:

$$h(0,x) = 1 + \exp(3\cos(\pi(x+0.5)) - 4). \tag{5.6}$$

The initial velocity is taken to be constant along x,

$$u(0,x) = 0.25. \tag{5.7}$$

The periodic spatial domain is $\Omega \in [-1,1]$ with 601 grid nodes, and the time-domain is $t \in [0,2]$ in which the solution is stored at 200 uniform time steps. The high-fidelity solution of the system of equations is computed using a discontinuous Galerkin solver with local polynomial reconstruction of degree 1 [49]. The Riemann solver utilized to compute the numerical flux is local Lax–Friedrichs. For integration in time, we use a second-order strong stability preserving Runge–Kutta scheme. The equations are solved in non-dimensional form, and all the reported parameter values are non-dimensional. For details about the conversion to dimensionless form, please refer to [50]. Upon performing a singular value decomposition of the parameter-specific snapshot matrices for the fluid height and velocity, we can plot their respective singular value decays. We observe that the decay is already not very fast for higher viscosity values, which gets exacerbated further as the viscosity values are decreased. The hyperbolic nature of these equations renders an even greater challenge for constructing an efficient reduced-order surrogate.

For our experiments, we fix $\lambda = 0.1$, and vary the viscosity ν from 1 to 10^{-5}. The total number of discrete parameters we consider when accounting for both the candidate set P^* and the parameter set P are 100. These ν values correspond to 100 evenly distributed logarithmic ν values in $\left[log_{10}(10^{-5}), log_{10}(1)\right]$, ensuring a decent pool of parameters.

To begin the active learning procedure, the parameter set P is initiated by 11 viscosity values corresponding to the following indices,

$$\{0, 99, 10, 20, 30, 40, 50, 60, 70, 80, 90\}.$$

Here, the ν values are ordered from lowest to highest and the index starts from 0 when counting the 100 values. High-fidelity snapshots are generated for all the viscosities in P for 200 time instances. Using these snapshots, POD-approximate solutions are computed such that the POD subspace retains 99.99997% of the energy.

During the active learning loop, we sample from the candidate set P^* at each iteration. The initial state of P^* for the reported results comprises 89 candidate values—upon exclusion of the 11 values initially present in P from the total 100 values. Fig. 5.4 sheds

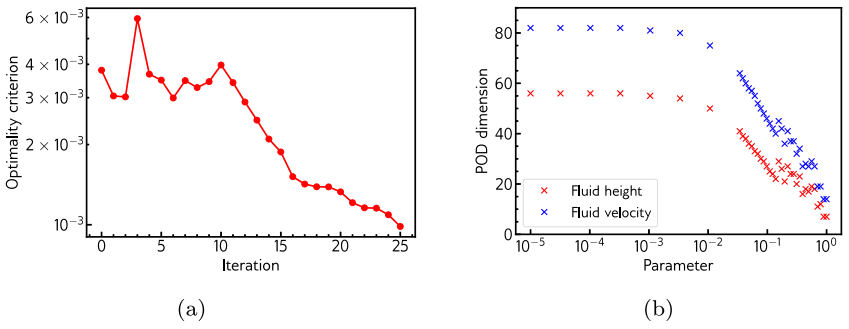


Fig. 5.4. Active learning for shallow water equations: (a) shows the optimality criterion $\mathcal{E}^{(iter)}$ (refer Algorithm 2) varying with greedy iterations of the active learning process; (b) shows the final POD subspace dimension for each of the chosen parameter samples.

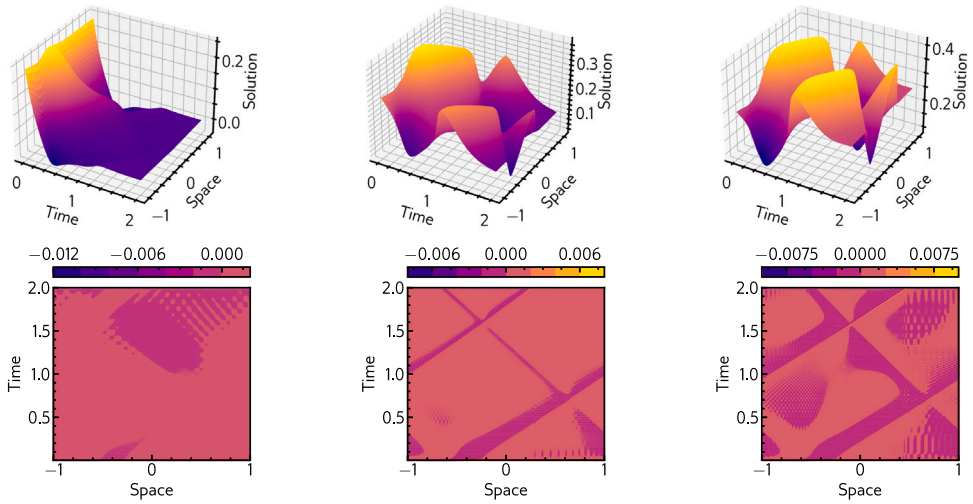


Fig. 5.5. Shallow water equations' *fluid velocity*: The ActLearn-POD-KSNN solution (shown in first row) and the solution error (shown in second row). The error values correspond to the point-wise difference in the space–time domain between the ActLearn-POD-KSNN solution and the true solution. The viscosity ν going from left to right in the rows are in the following order: $\{5\times10^{-1}, 5\times10^{-2}, 5\times10^{-5}\}$; and their respective POD subspace dimensions are $\{32, 97, 113\}$. All these ν values and time instances are outside of the training set.

light on the outcome of the active learning procedure. More specifically, Fig. 5.4(a) shows variation of the optimality criterion $\mathcal{E}^{(iter)}$ (see Algorithm 2) with greedy iterations of the active learning process. $\mathcal{E}^{(iter)}$ decreases over iterations, and we stop the loop after a tolerance of 10^{-3} is met. Upon termination, a total of 26 new viscosity samples are greedily selected.

The ultimate choice of viscosity values and their corresponding POD subspace dimensions are shown in Fig. 5.4(b). Like for the Burgers' equation, here as well the reported dimensions account for the POD subspace refinement through the iterations. Also like Burgers' equation, for low values of viscosities, the subspace dimension is comparatively higher. New selections are mostly concentrated in regions where the nature of the POD subspace changes significantly. Here, this is towards the moderate to high viscosity regions. Among all the individual parametric POD subspaces, the lowest energy criterion $\hat{\eta}$ in Eq. (4.1) for the fluid height is 2.989×10^{-10}, and for the fluid velocity is 6.362×10^{-10}. These are used as conditions for deciding the POD subspace dimension for the fluid height and velocity at any newly queried value of ν in the online phase (refer Algorithm 3).

In Fig. 5.5, the fluid velocity obtained from the ActLearn-POD-KSNN surrogate model is shown over the entire space–time domain. We can see that the ActLearn-POD-KSNN solution is successfully able to capture the multiple shock interactions over time. The viscosity values are taken outside the training set: $\{5\times10^{-1}, 5\times10^{-2}, 5\times10^{-3}, 5\times10^{-4}, 5\times10^{-5}\}$, and the POD subspace dimensions for the fluid velocity corresponding to these viscosity values are $\{32, 97, 112, 113, 113\}$. The solution is computed on a new test time grid with 499 instances starting from 0.004 with a step size of 0.004. The error contours in the second row of Fig. 5.5 show the point-wise error in the space–time domain, between the surrogate solution and the ground truth, highlighting the excellent accuracy of the predicted fluid velocity. The fluid height has similar behavior, but for compactness, its solution plots are omitted. Instead,

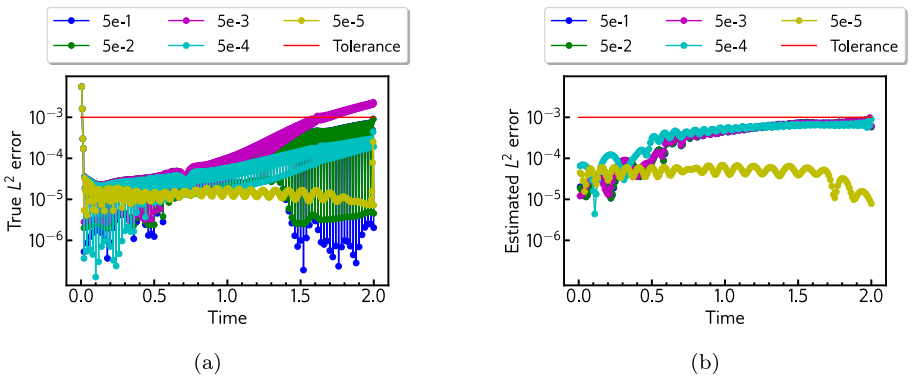


Fig. 5.6. Plot (a) shows the true error of the ActLearn-POD-KSNN solution for the *fluid height* of the shallow water equations on a new time grid corresponding to several out-of-training samples of ν. The tolerance used for termination of the active learning procedure is also shown for comparison. Plot (b) shows the estimated error for the fluid height on the original time grid.

Table 5.1
Shallow water equations: Comparison between runtime (in seconds) of the full-order model (FOM) and the ActLearn-POD-KSNN surrogate model. The simulations are performed for the shallow water equations at two spatial grid sizes. All the reported timings are the average of three independent executions. The division of offline and online phases in several parts follow from Section 4.1 and Remark 4.3. Note that the timing reported for Part B also includes the time taken to evaluate the FOM at the initial set of parameter samples, prior to initiating the active learning procedure.

FOM solver	ActLearn-POD-KSNN surrogate model				
	Offline phase			Online phase	
	Part A	Part B	Part C	Part D	Part E
249.56	0.8842	37×249.56	1.2456	0.0137	0.0027
	Total: 9235.85			Total: 0.0164	

in Fig. 5.6, we show the behavior of true and estimated relative error values along the time domain for the fluid height. More specifically, Fig. 5.6(a) shows the true relative error values of the fluid height obtained from the ActLearn-POD-KSNN surrogate. Here, the time instances are different from the training time grid. Moreover, an estimation of the error in the fluid height at several out-of-training parameters is shown in Fig. 5.6(b) for the training time grid. Similar to the Burgers' equation, the reported estimates are evaluated by following Remark 4.1 and querying AL-KSNN Eq. (2.16). For the most part, the true relative errors are bounded by the tolerance criterion 10^{-3} which is used for the active learning loop.

We report runtime of the full-order shallow water equation solver and the ActLearn-POD-KSNN surrogate model in Table 5.1. The numerical tests are carried out on a laptop with Intel® Core™ i5-1035G1 CPU @ 1.00 GHz and 16 GB of RAM. All the reported timings are the average of three independent executions. The timings reported under offline phase and online phase follow the division in several parts as per Section 4.1 and Remark 4.3. The time for active sampling and other offline computations required for building the surrogate, excluding the full-order model query time, is only 1.0649 s, i.e., $C_A + C_C$. The total full-order model query time during the offline phase depends on the number of parameter samples picked by the active learning procedure until its termination, and on the time it takes for generating the full-order solution for one parameter sample. Once the surrogate is built, a fast approximation of the solution is possible, in just 0.0164 s. Compared to evaluating the full-order solver at a new parameter sample, we can query the surrogate and obtain the approximate solution with a speedup of greater than $\mathcal{O}(10^4)$, as evident from Table 5.1. Note that the surrogate model becomes efficient as soon as it is called more often for unseen parameter values than used in the offline time.

5.3. Thermal block

We consider the following parabolic equation in a 2D spatial domain:

$$\frac{\partial \theta(t, \mathbf{z}; \boldsymbol{\sigma})}{\partial t} + \nabla \cdot (-\mu(\mathbf{z}; \boldsymbol{\sigma}) \nabla \theta(t, \mathbf{z}; \boldsymbol{\sigma})) = 0, \tag{5.8}$$

where the solution is the scalar temperature field θ, changing with time $t \geq 0$, space $\mathbf{z} := (x \times y) \in \Omega \in \mathbb{R}^2$, and heat conductivity μ. The initial temperature in the entire domain Ω is taken to be zero, i.e., $\theta(0, \mathbf{z}; \boldsymbol{\sigma}) = 0$. Fig. 5.7 illustrates the setup for this problem. The following conditions are specified at the four boundaries denoted in Fig. 5.7(a):

$$\mu(\mathbf{z}; \boldsymbol{\sigma}) \nabla \theta(t, \mathbf{z}; \boldsymbol{\sigma}) \cdot n(\mathbf{z}) = 1, \qquad \mathbf{z} \in \Gamma_{in}, \tag{5.9}$$

$$\mu(\mathbf{z}; \boldsymbol{\sigma}) \nabla \theta(t, \mathbf{z}; \boldsymbol{\sigma}) \cdot n(\mathbf{z}) = 0, \qquad \mathbf{z} \in \Gamma_{N}, \tag{5.10}$$

$$\theta(t, \mathbf{z}; \boldsymbol{\sigma}) = 0, \qquad \mathbf{z} \in \Gamma_{D}. \tag{5.11}$$

As shown in Fig. 5.7(a), the spatial domain $\Omega := [0, 1]^2$ is partitioned into five subdomains: $\Omega = \Omega_0 \cup \Omega_1 \cup \Omega_2 \cup \Omega_3 \cup \Omega_4$. The heat conductivity is different for each subdomain, and is given by

$$\mu(\mathbf{z}; \boldsymbol{\sigma}) := \begin{cases} 1 & \text{for } \mathbf{z} \in \Omega_0, \\ \sigma_i & \text{for } \mathbf{z} \in \Omega_i, \ i \in \{1, 2, 3, 4\}. \end{cases} \tag{5.12}$$

Due to the difference in heat conductivity value between the domains, we can observe discontinuities in the temperature distribution. For instance, when $\boldsymbol{\sigma} = [10^{-4}, 10^{-4}, 10^{-4}, 10^{-2}]$, at $t = 1$, the temperature along the spatial domain appears to be like shown in Fig. 5.7(b). Furthermore, as $\boldsymbol{\sigma}$ attains different values, the level of jumps in the temperature around the four subdomains will vary, showcasing non-smooth transitions in the solution across the parameter domain. Another challenging aspect with this example is the high-dimensional parameter domain. To illustrate the ideas presented in this work, we consider thermal conductivity values from the following domain: $\boldsymbol{\sigma} \in [10^{-4}, 10^{-1}] \times [10^{-4}, 10^{-1}] \times [10^{-4}, 10^{-1}] \times 10^{-2}$.

The spatial domain comprises of an unstructured mesh with 7488 grid nodes, and the time-domain is taken as $t \in [0, 1]$ with 100 time-steps. The high-fidelity solution is computed by performing a finite element discretization in space using the FEniCS package [51] to obtain the system matrices. Later, the semi-discrete system is solved via implicit Euler scheme in time with the PyMOR package [52]. Further details about the discretization can be found in [53]. The total number of discrete parameters ($\boldsymbol{\sigma}$) we consider when accounting for both the candidate set P^* and the parameter set P are 4096. We divide the logarithmic values of

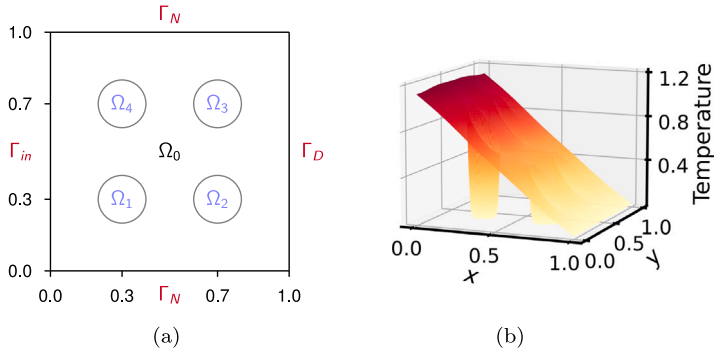


Fig. 5.7. Thermal block: Schematic of the spatial domain in (a); Temperature profile for $\boldsymbol{\sigma} = [10^{-4}, 10^{-4}, 10^{-4}, 10^{-2}]$ at $t = 1$ in (b).

the complete range for $\{\sigma_i\}_{i=1}^3$, i.e., $[log_{10}(10^{-4}), log_{10}(10^{-1})]$, into 15 uniform intervals. This renders us 16 distinct values for each, σ_1, σ_2, and σ_3. The $4096 = 16^3$ parameter samples correspond to the tensor product of these 16 values for each of $\{\sigma_i\}_{i=1}^3$, and σ_4 is fixed to 10^{-2}. These 4096 samples are shown in Fig. 5.8(b) by small gray dots.

At the start of the active learning procedure, the parameter set P is initiated by 16 parameter samples, where 8 samples correspond to the extreme points of the $3D$ parameter domain: $\{10^{-4}, 10^{-1}\} \otimes \{10^{-4}, 10^{-1}\} \otimes \{10^{-4}, 10^{-1}\} \otimes \{10^{-2}\}$, and the remaining 8 samples correspond to the extremes of an internal, smaller, cube: $\{10^{-3}, 10^{-2}\} \otimes \{10^{-3}, 10^{-2}\} \otimes \{10^{-3}, 10^{-2}\} \otimes \{10^{-2}\}$. High-fidelity snapshots are generated for the parameters in P at 100 instances of $t \in [0, 1]$. Using these snapshots, POD-approximate solutions are computed such that the POD subspace retains energy corresponding to $\eta(\boldsymbol{\sigma}) = 5 \times 10^{-7}$ for all the samples.

During the active learning loop, we sample from the candidate set P^* at each iteration. The initial state of P^* for the reported results comprise of 4080 candidate values—upon exclusion of 16 values initially included in P, from the total 4096 values. Fig. 5.8 sheds light on the outcome of the active learning procedure. More specifically, Fig. 5.8(a) shows the optimality criterion $\mathcal{E}^{(iter)}$ (see Algorithm 2) varying with greedy iterations of the active learning process. A total of 67 new parameter samples are selected, until the termination of the active learning procedure, when a tolerance of 10^{-2} is met.

The chosen set of parameter samples are denoted in Fig. 5.8(b) via red crosses. We observe a sparse selection of samples through the central regions of the parameter domain, with a few samples picked along most of the edges of the domain, and a high concentration near the lower left corner. This region corresponds to low values of $\{\sigma_i\}_{i=1}^3$, and the sample selection becomes progressively sparse as we move away from it. The solution discontinuities around the subdomains become more predominant as $\{\sigma_i\}_{i=1}^3$ attain values close to 10^{-4}. So, by observing the solution in the spatial domain for different parameter samples, we can intuitively understand this type of selection to be favoring regions in the parameter domain where the solution changes the most.

Similar to the other examples, POD space refinement is also carried out whenever necessary over the active learning iterations. Among all the individual parametric POD subspaces, the lowest energy criterion $\hat{\eta}$ in Eq. (4.1) is 2.313×10^{-9}. This is used as a condition for deciding the POD subspace dimension for any newly queried parameter in the online phase.

To evaluate the performance of the surrogate over different regions of the $3D$ parameter domain, we consider 27 parameter samples that are outside the training set. They can be concisely shown by the following Cartesian product: $\{3 \times 10^{-4}, 3 \times 10^{-3}, 3 \times 10^{-2}\} \otimes \{3 \times 10^{-4}, 3 \times 10^{-3}, 3 \times 10^{-2}\} \otimes \{3 \times 10^{-4}, 3 \times 10^{-3}, 3 \times 10^{-2}\} \otimes \{10^{-2}\}$. For four representative test samples out of these, we show the ActLearn-POD-KSNN surrogate solution and its relative error in the $2D$ spatial domain at a test time instance of $t = 0.595$ in Fig. 5.9. The errors are all below 10^{-3} and are usually high around the subdomains due to the temperature jumps.

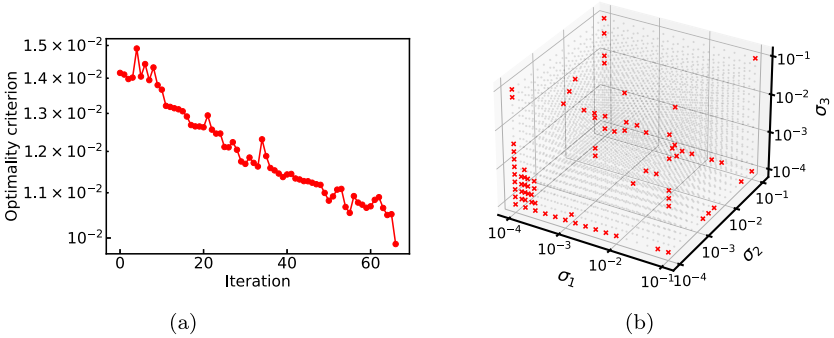


Fig. 5.8. Active learning for thermal block: (a) shows the optimality criterion $\mathcal{E}^{(iter)}$ (refer Algorithm 2) varying with greedy iterations of the active learning process; (b) shows the final set of chosen parameter samples (red crosses), and the complete pool of 4096 discrete samples (small gray dots) in the parameter domain. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

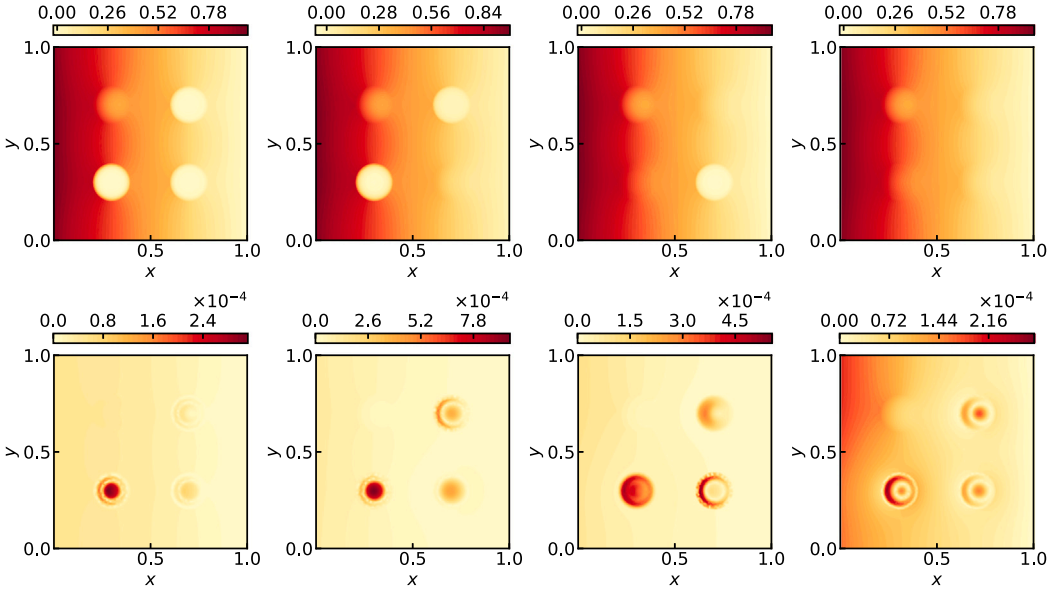


Fig. 5.9. Thermal block: The ActLearn-POD-KSNN solution (shown in first row) and the relative solution error (shown in second row) at a representative test time instance $t = 0.595$. The test σ values going from left to right in the rows are in the following order: $[3\times10^{-4}, 3\times10^{-4}, 3\times10^{-4}, 10^{-2}]$, $[3\times10^{-4}, 3\times10^{-2}, 3\times10^{-4}, 10^{-2}]$, $[3\times10^{-2}, 3\times10^{-4}, 3\times10^{-2}, 10^{-2}]$, $[3\times10^{-2}, 3\times10^{-2}, 3\times10^{-2}, 10^{-2}]$; and their respective POD subspace dimensions are $\{7, 7, 7, 6\}$.

For all 27 test parameters, we evaluate the surrogate solution at a test time grid with 99 time instances, starting from 0.015, with a step size of 0.01. The relative error $E_j(\boldsymbol{\mu}_i^*)$ of the surrogate solution is obtained for each time instance t_j in the test time grid, for all the test parameters $\boldsymbol{\mu}_i^*$. Due to limited space, we show the average error values over the test parameters, along time, i.e. $\frac{1}{27}\sum_{i=1}^{27} E_j(\boldsymbol{\mu}_i^*),\ j = 1, \ldots, 99$, in Fig. 5.10, by the curve labeled 'True'. Whereas, the curve labeled 'Estimate' in Fig. 5.10 shows the estimated relative errors $\tilde{\varepsilon}(\boldsymbol{\mu}_i^*)$ corresponding to time instances in the training time grid. For each $\boldsymbol{\mu}_i^*$, following Remark 4.1, AL-KSNN Eq. (2.16) is queried. The estimated errors values in Fig. 5.10 are also the average of the $\tilde{\varepsilon}$ over all the test parameter samples $\boldsymbol{\mu}_i^*$. Unlike other examples, we observe that the estimated error values are less that the true error values. This can be explained due to the high interpolation errors in the temperature field, which arises due to the jumps around the subdomains. However, for the most part, except for a few time instances near $t = 0$, and one time instance near $t = 1$, the true relative errors along the time domain are still bounded by the tolerance criterion (10^{-2}) used for the active learning loop.

We report runtime of the full-order thermal block solver and the ActLearn-POD-KSNN surrogate model in Table 5.2. The numerical tests are carried out on a system with Intel® Core™ i7-11800H @ 2.30 GHz and 32 GB of RAM. Compared to evaluating the full-order solver at a new parameter sample, we can query the surrogate and obtain the approximate solution with a speedup of around $\mathcal{O}(10^1)$.

The observed speed up for this example is not as high as the other three examples, mainly because of the linear nature of the underlying equation. More specifically, with the implicit time stepping, we are able to directly make temporal evaluations of the solution at the time grid of interest for which we want to collect the snapshots. This is unlike the other nonlinear examples, where

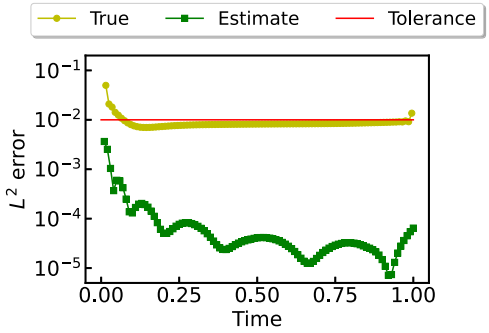


Fig. 5.10. Thermal block: Comparison between the true error of the ActLearn-POD-KSNN solution on the test time grid and the estimated error on the original time grid. The reported error values correspond to the average over all 27 test parameter samples. The tolerance used for termination of the active learning procedure is also shown for comparison.

Table 5.2
Thermal block: Comparison between runtime (in seconds) of the full-order model (FOM) and the ActLearn-POD-KSNN surrogate model. All the reported timings are the average of three independent executions. The division of offline and online phases in several parts follow from Section 4.1 and Remark 4.3. Note that the timing reported for Part B also includes the time taken to evaluate the FOM at the initial set of parameter samples, prior to initiating the active learning procedure.

FOM solver	ActLearn-POD-KSNN surrogate model				
	Offline phase			Online phase	
	Part A	Part B	Part C	Part D	Part E
1.12	24.56	83×1.12	20.70	0.01	0.11
	Total: 138.22			Total: 0.12	

one needs to specify a very fine time step size during the FOM simulation, as compared to the time step size corresponding to which the snapshot data needs to be stored, leading to significantly longer simulation time of the full-order system.

The thermal block problem is illustrative of how the active learning framework can also be used for problems with multi-dimensional parameter domains. Moreover, due to the parameter-varying jumps in the temperature profile, solution interpolation in the parameter domain is challenging. But the KSNN with locally-varying kernel widths allow us to perform a reasonable interpolation, such that the final surrogate error still stays under or close to the specified error tolerance.

5.4. Navier–Stokes equations

We consider the following incompressible Navier–Stokes equations in a 2D spatial domain:

$$\rho\left(\partial_t \mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = \nabla\cdot\boldsymbol{\sigma}(\mathbf{u}, p), \tag{5.13}$$

$$\nabla\cdot\mathbf{u} = 0, \tag{5.14}$$

where $\mathbf{u}$ denotes the velocity vector field, and p denotes the scalar pressure field. They vary with time $t \geq 0$, space $\mathbf{z} := (x \times y) \in \Omega \in \mathbb{R}^2$, fluid density ρ, and the dynamic viscosity μ of the fluid. In Eq. (5.13), $\boldsymbol{\sigma}$ is the stress tensor, which has the following form,

$$\boldsymbol{\sigma}(\mathbf{u}, p) = 2\mu\boldsymbol{\epsilon}(\mathbf{u}) - pI. \tag{5.15}$$

Here, $\boldsymbol{\epsilon}$ is the strain rate tensor, given by,

$$\boldsymbol{\epsilon}(\mathbf{u}) = \frac{1}{2}\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right). \tag{5.16}$$

As shown in Fig. 5.11, the spatial domain comprises of a $2D$ channel with a circular obstacle. Except for the area occupied by the obstacle, the fluid can flow through the entire domain, which is denoted by $\Omega := \Omega_{fluid}$. The following conditions are specified at the five boundaries highlighted in red in Fig. 5.11:

$$u_x = \frac{4U\,y(0.41 - y)}{0.41^2}, \qquad \mathbf{z} \in \Gamma_{in}, \tag{5.17}$$

$$u_y = 0, \qquad \mathbf{z} \in \Gamma_{in}, \tag{5.18}$$

$$\boldsymbol{\sigma}(\mathbf{u}, p)\mathbf{n} = 0, \qquad \mathbf{z} \in \Gamma_{out}, \tag{5.19}$$

$$\mathbf{u} = 0, \qquad \mathbf{z} \in \Gamma_{cyl} \cup \Gamma_{top} \cup \Gamma_{bottom}, \tag{5.20}$$

where u_x and u_y are the components of $\mathbf{u}$ in the $x-$ and $y-$direction, $U = 1.5$ is the maximum velocity magnitude attainable with the prescribed parabolic velocity profile, and $\mathbf{n}$ denotes the outward normal vector to Ω_{fluid}. The initial velocity in the entire domain Ω_{fluid} is taken to be zero, i.e., $\mathbf{u} = 0$.

The fluid density is taken as $\rho = 1\ \mathrm{kg/m^3}$. We consider the dynamic viscosity μ as a free parameter, which can attain different values such that the Reynolds number Re varies in $[5, 200]$. This problem setup is similar to the DFG flow around a cylinder

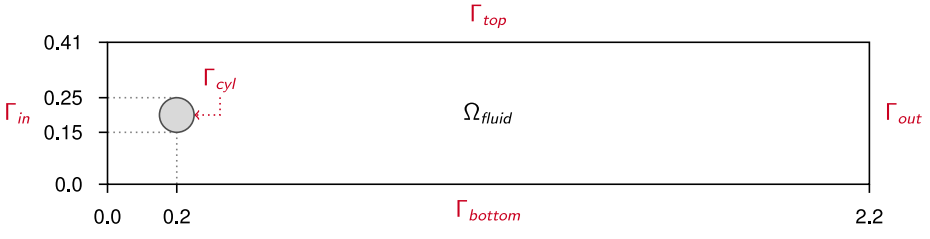


Fig. 5.11. Navier–Stokes equations: Schematic of the spatial domain.

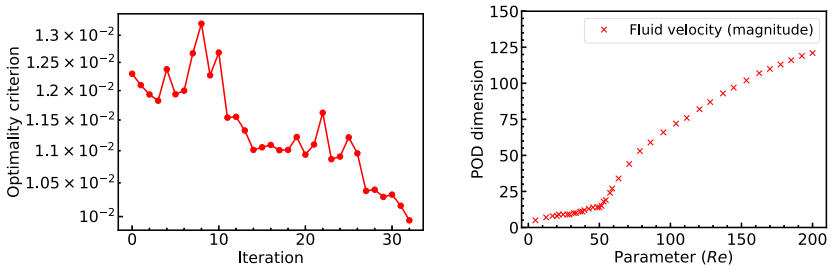


Fig. 5.12. Active learning for Navier–Stokes equations: (a) shows the optimality criterion $\mathcal{E}^{(iter)}$ (refer Algorithm 2) varying with greedy iterations of the active learning process; (b) shows the final POD subspace dimension for each of the chosen parameter samples.

benchmark $2D$-2 [54], where they fix the Reynolds number to 100. To illustrate the applicability of our ideas, we consider a parametric version of the problem encompassing two distinct laminar flow regimes—one where a fixed pair of symmetric vortices are observed in the cylinder wake; another where a von Kàrmàn vortex street is observed in the cylinder wake. We are interested in the magnitude of fluid velocity over the domain. It is straight-forward to generate surrogates for each of the velocity components and the pressure field by following Remarks 2.3 and 3.3, as we have done for both solution components of the shallow water equations in Section 5.2.

The spatial domain comprises of an unstructured mesh with 37,514 grid nodes, and the time-domain is taken as $t \in [0, 5]$ with 1000 time-steps. The high-fidelity solution is computed by the incremental pressure correction scheme (IPCS) [55], using the FEniCS package [51]. We perform finite element discretization with Taylor–Hood elements in space, and a combination of biconjugate gradient stabilized method with the conjugate gradient method in time. As a whole, in the candidate set P^* and the parameter set P, we consider a total of 131 discrete Re values evenly distributed in $[5, 200]$.

To begin the active learning procedure, the parameter set P is initiated by the following 7 Re values: $\{5, 38, 71, 104, 137, 170, 200\}$. As a result, the initial state of P^* comprise 124 candidate values, i.e., excluding the 7 values considered in P from the total pool of 131 values. High-fidelity snapshots are generated for all $Re \in P$ at 1000 instances of $t \in [0, 5]$. Using the collected snapshots, POD-approximate solutions are computed such that the POD subspace retains energy corresponding to $\eta(\sigma) = 5 \times 10^{-6}$ for all the Re samples.

Fig. 5.12 sheds light on the outcome of the active learning procedure. More specifically, Fig. 5.12(a) shows the optimality criterion $\mathcal{E}^{(iter)}$ (see Algorithm 2) varying with greedy iterations of the active learning process. When the tolerance of 10^{-2} is met, a total of 33 new parameter locations are selected. This selection of Re values and their corresponding POD subspace dimensions

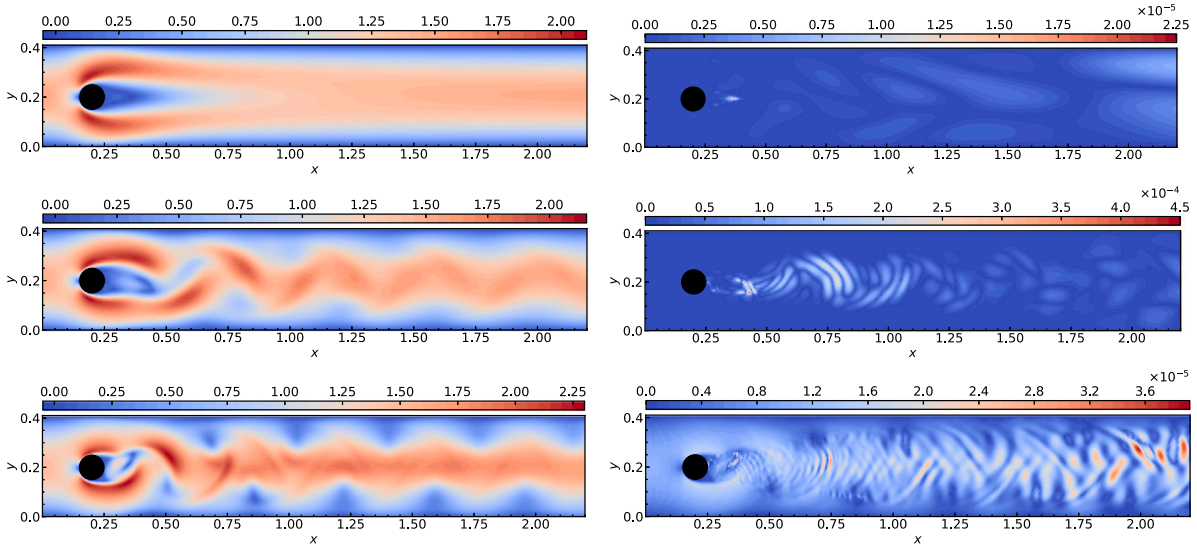


Fig. 5.13. Navier–Stokes equations: The ActLearn-POD-KSNN solution (shown in first column) and the relative solution error (shown in second column) at a representative test time instance $t = 3.512$. The test Re values going from top to bottom in both columns are in the following order: $\{25, 75, 180\}$. Their respective POD subspace dimensions are $\{13, 105, 208\}$.

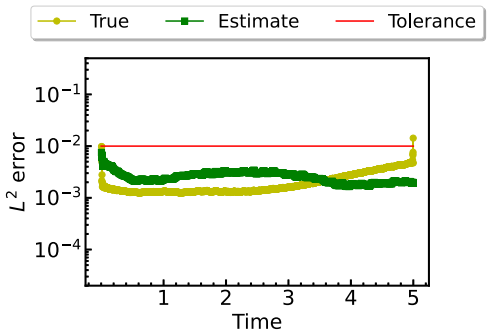


Fig. 5.14. Navier–Stokes equations: Comparison between the true error of the ActLearn-POD-KSNN solution on the test time grid and the estimated error on the original time grid. The reported error values correspond to the average over all 5 test parameter samples. The tolerance used for termination of the active learning procedure is also shown for comparison.

Table 5.3
Navier–Stokes equations: Comparison between runtime (in seconds) of the full-order model (FOM) and the ActLearn-POD-KSNN surrogate model. All the reported timings are the average of three independent executions. The division of offline and online phases in several parts follow from Section 4.1 and Remark 4.3. Note that the timing reported for Part B also includes the time taken to evaluate the FOM at the initial set of parameter samples, prior to initiating the active learning procedure.

FOM solver	ActLearn-POD-KSNN surrogate model				
	Offline phase			Online phase	
	Part A	Part B	Part C	Part D	Part E
38,742	249.90	40×38742.18	85.39	1.30	1.88
		Total: 1,550,022		Total: 3.18	

are shown in Fig. 5.12(b). The reported dimensions also account for the refinement in the POD space, whenever required through the iterations, as discussed in Section 2.2.

Due to the vortex shedding phenomenon, and subsequent increase of convective effects as the Reynolds number increases, the subspace dimensions are comparatively higher for high Re, as evident from Fig. 5.12(b). Moreover, we observe a high concentration of sample selection around the Hopf bifurcation point of $Re = 47$, where the fluid flow behavior switches from one regime to another. Among all the individual parametric POD subspaces, the lowest energy criterion $\hat{\eta}$ in Eq. (4.1) is 3.033×10^{-7}. This is used as a condition for deciding the POD subspace dimension for any newly queried parameter.

To evaluate the performance of the surrogate, we consider 5 Re samples outside the training set, spread over different regions of the parameter range. The test Re values are $\{25, 75, 115, 145, 180\}$; with the POD subspace dimensions corresponding to these Reynolds numbers being $\{13, 105, 169, 192, 208\}$. For three representative test samples out of these, we show the ActLearn-POD-KSNN surrogate solution and its relative error in the $2D$ spatial domain, at a test time instance of $t = 3.512$, in Fig. 5.13. We can observe that the error values are quite low.

For all 5 test parameters, we evaluate the surrogate solution at a test time grid with 2498 time instances, starting from 0.006, with a step size of 0.002. The relative error of the surrogate solution is obtained for each time instance in the test time grid, for all the test parameters. The average errors $\frac{1}{5}\sum_{i=1}^{5} E_j(\mu_i^*)$, $j = 1, \ldots, 2498$, over the test parameters, changing with time, are reported in Fig. 5.14 by the curve labeled 'True'. Whereas, the curve labeled 'Estimate' in Fig. 5.14 shows the estimated relative error values in space corresponding to time instances in the training time grid. These estimated errors values are also the average of the error estimator $\tilde{\varepsilon}(\mu_i^*)$ (AL-KSNN, Remark 4.1) over all test parameter values μ_i^*. They are close to the true error values. Moreover, except for the last test time instance, the true relative errors along the time domain are under the tolerance criterion (10^{-2}) used for the active learning loop. It is particularly important to note that the surrogate is able to provide the solution accurately, also in the initial transient regime, when the flow is not fully developed yet.

We report runtime of the full-order Navier–Stokes equations and the ActLearn-POD-KSNN surrogate model in Table 5.3. The numerical tests are carried out on a single node of the cluster Mechthild available at the Max Planck Institute Magdeburg. The node is equipped with two Intel® Xeon Silver 4110 central processing units and 192 GB of RAM. Compared to evaluating the full-order solver at a new parameter sample, we can query the surrogate and obtain the approximate solution with a speedup of greater than $\mathcal{O}(10^4)$. The surrogate construction pays off as soon as it is queried more often for unseen parameter values than used in the offline phase. It is important to note that even for this considerably high-dimensional problem, with numerous temporal snapshots, the computational cost of the active learning part, as well as the interpolant construction part, both in the offline and the online stages, stays manageable. In fact, it stays significantly cheaper than a single FOM evaluation at any given parameter sample.

5.4.1. Noisy data

We investigate the performance of the proposed active learning framework when the available data is corrupted by salt-and-pepper noise. In particular, we artificially corrupt the velocity field data coming from the full-order model solver. After the snapshot

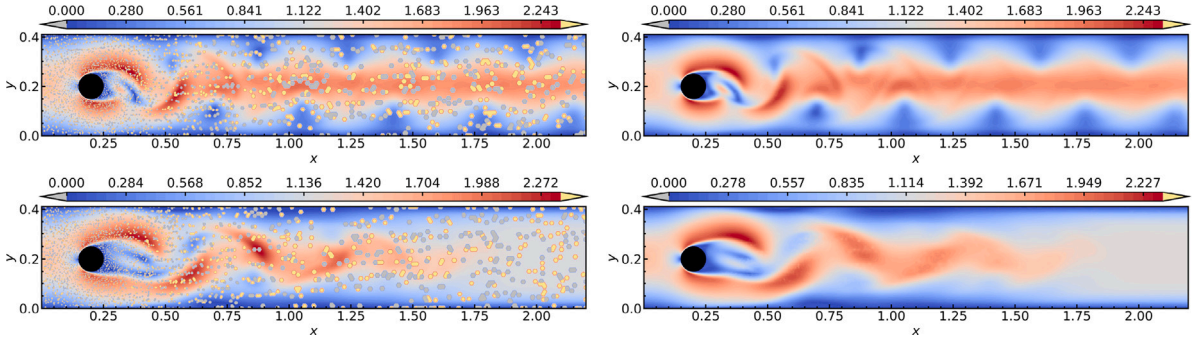


Fig. 5.15. The corrupted solution field (shown in first column) and the RPOD denoised solution (shown in second column) for $Re = 200$. Plots in the first row correspond to time instance $t = 4$, whereas, plots in the second row correspond to time instance $t = 1.5$. Gray and yellow dots represent the corrupt data locations with a respective magnitude of $-\sigma$ and $+\sigma$.

matrix corresponding to any given parameter sample is generated, velocities at 20% of the grid nodes and temporal locations are randomly assigned a value of $\pm 10\sigma$, where σ is the standard deviation of the velocity values present in the snapshot matrix. This form of artificial corruption is similar to how authors in [42] generated noisy data from the true solution data. In this work, we sequentially corrupt all the parameter-specific snapshot matrices, after each full-order model query, as part of the active learning loop. The 20% spatio-temporal corruption points are randomly selected for each parameter sample, resulting in different locations where the noise is introduced for each parameter. Moreover, the magnitudes of corrupted values are also different for each parameter sample, as the standard deviation of the velocity values depend on the flow profile, which is varying with each parameter.

The noisy snapshot data corresponding to each parameter location used during the offline phase, undergoes a cleaning step via robust POD (refer Algorithm 4), and provides a low-rank denoised snapshot matrix. For illustration, Fig. 5.15 shows the noisy and denoised velocity fields corresponding to a training parameter location of $Re = 200$, and time instances $t = 4$ and $t = 1.5$. With a closer look at the denoised velocity fields, we can notice that for the flow at $t = 4$, from the fully-developed regime, we recover the true velocity profile with an excellent accuracy. However, for the flow at $t = 1.5$, from the transient regime, we can still notice slight noise-induced corruption of the vortices shed in the cylinder wake. The level of denoising can be further improved by undertaking more RPOD iterations. However, to keep the computational costs reasonable, we restrict to 1000 iterations in the reported results. An average execution time of 3578.23 s is observed for denoising each snapshot matrix corresponding to any particular Reynolds number. Fig. 5.16 shows the singular value decays of the true, noisy, and denoised velocity fields at $Re = \{5, 65, 200\}$. We observe a partial recovery of the singular values upon denoising.

The setup for the active learning procedure is similar to the noise-free Navier–Stokes equations' setting. The initial energy criterion is taken to be $\eta(\sigma) = 7 \times 10^{-6}$, with which the POD-approximate solutions are computed. Fig. 5.17 sheds light on the outcome of the active learning procedure. More specifically, Fig. 5.17(a) shows the optimality criterion $\mathcal{E}^{(iter)}$ (see Algorithm 2) varying with greedy iterations of the active learning process. As opposed to the noise-free case, a total of 57 new parameter locations are selected, until the tolerance of 10^{-2} is sufficed.

The final selection of Re values and their corresponding POD subspace dimensions are shown in Fig. 5.17(b). We notice more crowding around the bifurcation point of $Re = 47$, as observed for the noise-free case. Moreover, the selection pattern until 33 iterations is similar to the noise-free case. The main reason behind the additional parameter selection is the error present in the denoising step. More precisely, Fig. 5.16 highlights that we only recover a portion of the singular values upon denoising. This is to

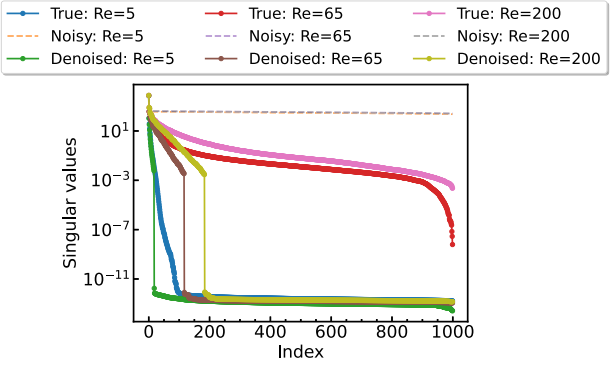


Fig. 5.16. Comparison between the singular value decays of the true, noisy, and denoised snapshot data at three representative parameter locations.

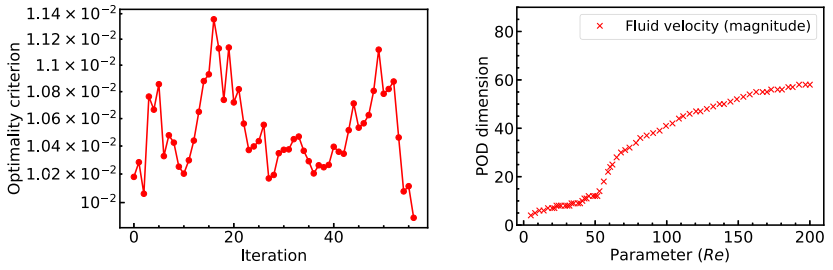


Fig. 5.17. Active learning with noisy data from Navier–Stokes equations: (a) shows the optimality criterion $\mathcal{E}^{(iter)}$ (refer Algorithm 2) varying with greedy iterations of the active learning process; (b) shows the final POD subspace dimension for each of the chosen parameter samples.

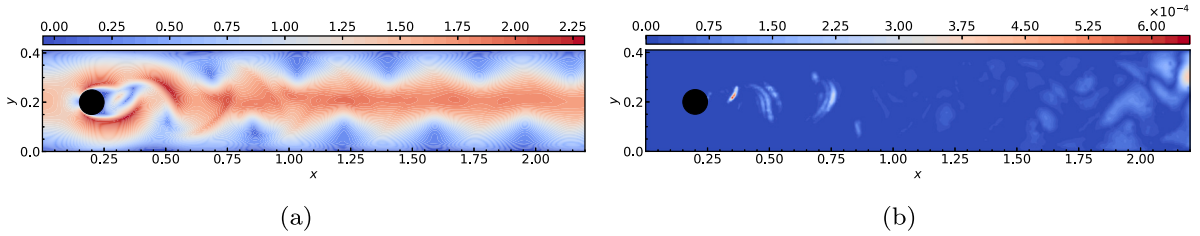


Fig. 5.18. Noisy data from Navier–Stokes equations: The ActLearn-POD-KSNN solution in (a) and the relative solution error in (b) at a representative test time instance $t = 3.512$. The representative test parameter value is $Re = 180$.

be expected since we attempt at learning a denoised low rank matrix. However, we have observed that the situation improves as the RPOD iterations are increased, allowing us to recover more dominant modes. So empirically, we can expect fewer parameter selections as the number of cleaning iterations are increased.

Among all the enriched parametric POD subspaces, the lowest energy criterion $\hat{\eta}$ in Eq. (4.1) is 5.678×10^{-7}, which is used as a condition for deciding the POD subspace dimension for any newly queried parameter. To evaluate the performance of the surrogate, the same 5 test samples are considered, as for the noise-free case. The POD subspace dimensions corresponding to $Re = \{25, 75, 115, 145, 180\}$ are $\{11, 49, 63, 74, 79\}$, respectively. Note that for higher Reynolds numbers, the dimension of the POD subspaces are relatively low, in comparison to the noise-free case. This is because of the partial recovery of the singular value decay spectrum (and associated modes), as shown in Fig. 5.16. In Fig. 5.18, we show the ActLearn-POD-KSNN surrogate solution and its relative error at a representative test parameter of $Re = 180$, and test time instance of $t = 3.512$. We notice that the solution error is quite small and below 10^{-3}. However, it is important to note that we observe the errors to be of $\mathcal{O}(10^{-4})$, which is one order higher than the errors reported for $Re = 200$ in Fig. 5.13, for the noise-free case.

Similar to the noise-free case, we further evaluate the surrogate solution at the complete test time grid, for all the test parameters. The true and estimated relative errors of the surrogate solution, along time, are reported in Fig. 5.19. Same as before, these are the average error values over all the test parameters. The true relative errors of the surrogate (with respect to the noise-free solution)

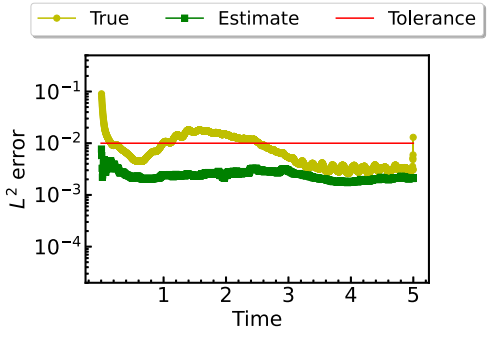


Fig. 5.19. Noisy data from Navier–Stokes equations: Comparison between the true error of the ActLearn-POD-KSNN solution on the test time grid and the estimated error on the original time grid. The reported error values are the average over all 5 test parameter samples. The tolerance used for termination of the active learning procedure is also shown for comparison.

are above the tolerance at the onset of fluid flow in the domain, and for some part during the initial transient regime. The latter behavior is expected, as we know from Fig. 5.15 that with the current level of denoising, there is still presence of slight noise-induced corruption of the solution field during the transient regime. Moreover, since we initialize the problem with a zero fluid velocity in the spatial domain, the velocity magnitude at the onset of the flow is quite small. As a result, even the presence of a mild corruption in the denoised solution values (in this situation) can result in significant relative errors, like observed in Fig. 5.19. Note that except for these two regions, we observe the relative errors to be under the tolerance of 10^{-2}.

6. Conclusions

We have proposed an active learning framework for parametric non-linear dynamical systems, which generates solution snapshots at new parameter locations by evaluating the high-fidelity model only when necessary. This, in turn, improves the accuracy of the data-driven surrogate model. The central driving force of the active learning process is a non-intrusive error-estimator-based optimality criterion, enabling a synergetic parameter-specific POD subspace adaptation along with the training set adaptation. The proposed optimality criterion is designed from the parameter-specific relative POD approximation errors. Through active learning, we iteratively arrive at a good selection of solution snapshots which are then used to train the data-driven surrogate. In doing so, we relax the vast data requirement for training data-driven surrogate models to some extent, and also provide an estimation of the surrogate error.

The numerical results show that the developed active learning framework iteratively detects locations in the parameter domain where the variation in solution features is high, and prefers new snapshot generation in those regions. For the Burgers' equation, the ActLearn-POD-KSNN surrogate model is able to successfully gauge the variation in its initial conditions and capture the transport of shock profiles accurately in time, over the entire range of viscosity values. Moreover, for the shallow water equations, the surrogate model is able to efficiently predict, at new parameter locations, the interacting shock waves that morph into each other over time.

The interpolation steps in the active learning loop as well as within the surrogate model's construction are carried out by automatically building, training, and evaluating several kernel-based shallow neural networks (KSNNs). Such a shallow architecture results in a fast offline training stage, as well as a fast online evaluation stage, further reducing the overall computational burden. Moreover, the KSNN is equipped with center-dependent kernel widths, which can be trained along with the center location of each kernel, by using the alternating dual-staged iterative training procedure (ADSIT). This allows us to have an improved interpolation/regression performance, crucial for automatic deployment of the network on any dataset, particularly beneficial for the error estimator construction (AL-KSNN).

By equipping the AL-KSNN, μ-KSNN, and t-KSNN in the ActLearn-POD-KSNN surrogate with locally adaptive kernel widths, we are able to actively build the surrogate even for the $2D$ thermal block example, which showcases a multi-dimensional parameter space and parameter-varying jumps in the solution profile. Moreover, we observe that for the $2D$ incompressible Navier–Stokes equations, the active sampling favors locations in the parameter space close to the Hopf bifurcation point, thereby rendering an accurate surrogate solution across the entire parameter domain. The actively learned surrogate is also able to provide accurate solutions in the initial transient regime, when the flow is not fully developed yet. Furthermore, by extending the ActLearn-POD-KSNN surrogate with robust POD, we obtain good results even when the velocity field is corrupted by noise.

The parameter-specific adaptive POD subspaces make our approach efficient, even for problems with mixed—convective and diffusive—phenomena, where each of them dominate in certain regions. Additionally, we observe that the true surrogate errors stay under or are very close to the tolerance level used to terminate the active learning procedure. This indicates reliability of the proposed non-intrusive error estimation, providing us a good measure to gauge the accuracy of the constructed ActLearn-POD-KSNN surrogate model. The training strategy for our ActLearn-POD-KSNN surrogate model is problem independent, and automatically selects the parameter locations whose additional solution snapshots would most improve the solution accuracy. This minimizes the user interaction for data-driven surrogates built using machine learning, and the fast online deployment phase brings us a step closer to real-time simulations for high-fidelity parametric physical systems.

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.

Data availability

The Python source code relevant to this research work is available on GitLab [56], accessible through the following link: https://gitlab.mpi-magdeburg.mpg.de/kapadia/active-learning-surrogate-modeling. Furthermore, to ensure long-term accessibility and preservation, we have archived it in a Zenodo repository [57], accessible through the following link: https://doi.org/10.5281/zenodo.10237734.

Acknowledgment

Harshit Kapadia is supported by the International Max Planck Research School for Advanced Methods in Process and Systems Engineering (IMPRS-ProEng).

Appendix A. KSNN regression via the ADSIT procedure

In this appendix, we provide an illustration of KSNNs trained in regression mode to approximate $1D$ functions by employing the alternating dual-staged iterative training (ADSIT) procedure as outlined in Section 3.1.3. We take the input domain as $x \in [-1, 1]$ and consider the following two functions:

$$\hat{y}^{(a)}(x) = \sin(4\pi|x|^{\frac{3}{2}}) + x^2, \qquad \hat{y}^{(b)}(x) = \tanh(100x). \tag{A.1}$$

Including the endpoints of the input domain, we take 256 uniform samples of both functions $\hat{y}^{(a)}(x)$ and $\hat{y}^{(b)}(x)$ creating data pairs $\{x_j, \hat{y}_j^{(a)}\}_{j=1}^{256}$ and $\{x_j, \hat{y}_j^{(b)}\}_{j=1}^{256}$, respectively. Also, note that the KSNN's are always queried at 512 uniform samples of $x \in [-1, 1]$, including the endpoints.

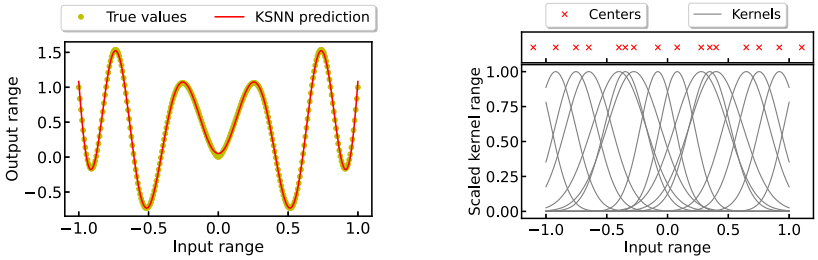


Fig. A.1. KSNN regression for $\hat{y}^{(a)}(x)$ from Eq. (A.1) when both center-specific kernel widths and center locations are trained. (a) compares the KSNN prediction with the true values, and (b) shows the trained kernels and their center locations.

Using the data pairs $\{x_j, \hat{y}_j^{(a)}\}_{j=1}^{256}$ and hidden layer size $n_c = 16$, a KSNN is trained by following the setup for Fig. A.1 in Table A.1. Here, both the kernel widths and center locations are trained. The KSNN approximation for $\hat{y}^{(a)}(x)$ is shown in Fig. A.1 along with the learned kernels and their center locations.

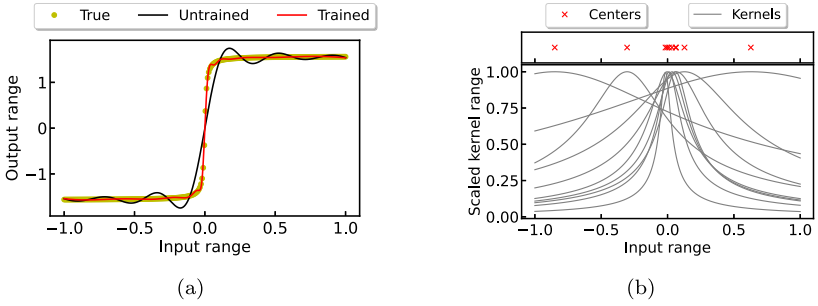


Fig. A.2. KSNN regression for $\hat{y}^{(b)}(x)$ from Eq. (A.1) when both center-specific kernel widths and center locations are trained. (a) compares the trained KSNN prediction and untrained KSNN prediction with the true values, and (b) shows the trained kernels and their center locations.

Now, using the data pairs $\{x_j, \hat{y}_j^{(b)}\}_{j=1}^{256}$ and hidden layer size $n_c = 10$, a KSNN is trained by following the setup for Fig. A.2 in Table A.1. Here as well, both the kernel widths and center locations are trained. The KSNN approximation for $\hat{y}^{(b)}(x)$ is shown in Fig. A.2 along with the learned kernels and their center locations. Additionally, we also show the approximation of the network with the initialized center locations and kernel widths. This is referred to as untrained in Fig. A.2. Along with a local adaptation of the kernel widths, we notice a crowding of the kernels close to the sharp discontinuity around $x = 0$. As a result, the ADSIT procedure allows us to control the accuracy of KSNN and renders an accurate prediction.

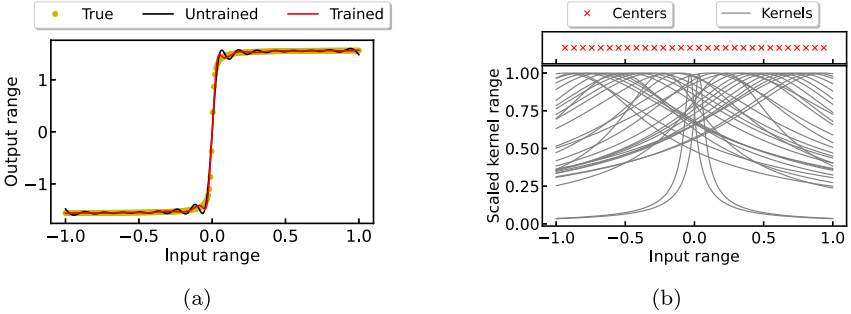


Fig. A.3. KSNN regression for $\hat{y}^{(b)}(x)$ from Eq. (A.1) when only center-specific kernel widths are trained. (a) compares the trained KSNN prediction and untrained KSNN prediction with the true values, and (b) shows the trained kernels and their center locations.

Next, we illustrate the KSNN approximation when only the kernel widths are trained. We use the data pairs $\{x_j, \widehat{y}_j^{(b)}\}_{j=1}^{256}$ and hidden layer size $n_c = 30$ for creating a KSNN by following the setup for Fig. A.3 in Table A.1. This KSNN's approximation for $\widehat{y}^{(b)}(x)$ is shown in Fig. A.3 along with the learned kernels and their (untrained) center locations. Like before, we also show the untrained network's approximation in Fig. A.3. We notice that the kernel widths for two kernels located in the vicinity of the discontinuity are significantly less, allowing the KSNN to approximate the jump in the function value accurately.

Table A.1
KSNN implementation details for regression of the $1D$ function from Eq. (A.1) as shown in Figs. A.1 to A.3.

Setup	Kernel	*alter_iter*	*epochs*	*lr*	n_c
Fig. A.1	Gaussian	10	5	0.1	16
Fig. A.2	Inverse multi-quadric	200	1	0.1	10
Fig. A.3	Inverse multi-quadric	50	1	0.1	30

Appendix B. ActLearn-POD-KSNN implementation details

In Table B.1, we provide details about the KSNN setups for all the numerical experiments considered in our work. Note that for the Burgers' equation and shallow water equations, we do not train for the kernel widths, but user-specify a fixed width for all the centers. This is denoted by ϵ in the table. However, for the thermal block and Navier–Stokes equations, center-specific locally-adaptive kernel widths are learned using the ADSIT procedure.

Table B.1
KSNN implementation details for the optimality criterion evaluation in the active learning loop (AL-KSNN), for the parametric interpolant construction (μ-KSNN), and for the temporal interpolant construction (t-KSNN). The four models are encoded as follows: Model 1: Burgers' equation; Model 2: Shallow water equations; Model 3: Thermal block; Model 4: Navier–Stokes equations.

Model & KSNN detail	Kernel	*alter_iter*	*epochs*	*lr*	ϵ
Model 1: AL-KSNN	Multi-quadric	0	–	–	10^{-2}
Model 1: μ-KSNN	Multi-quadric	0	–	–	10^{-2}
Model 1: t-KSNN	Multi-quadric	0	–	–	10^{-2}
Model 2: AL-KSNN	Multi-quadric	0	–	–	10^{-5}
Model 2: μ-KSNN	Multi-quadric	0	–	–	10^{-3}
Model 2: t-KSNN	Multi-quadric	0	–	–	10^{-3}
Model 3: AL-KSNN	Matern12	20	5	10^{-3}	–
Model 3: μ-KSNN	Matern32	30	3	10^{-2}	–
Model 3: t-KSNN	Matern52	30	1	10^{-2}	–
Model 4: AL-KSNN	Matern12	20	5	10^{-3}	–
Model 4: μ-KSNN	Matern52	20	1	10^{-2}	–
Model 4: t-KSNN	Matern52	10	1	10^{-2}	–

Appendix C. Comparison: ActLearn-POD-KSNN vs POD-KSNN

In this appendix, we try to provide a comprehensive comparison between the solution accuracy for ActLearn-POD-KSNN and POD-KSNN, highlighting the contributions and benefits of the active learning framework. The comparison is provided for two of the four reported models in Section 5, i.e, Burgers' equation and shallow water equations.

C.1. Burgers' equation

In Fig. C.1, we provide a comparative study between ActLearn-POD-KSNN solution error and POD-KSNN solution errors that are obtained by randomly picking 30 and 54 parameter samples for preparing the training snapshots of the surrogate. We label the

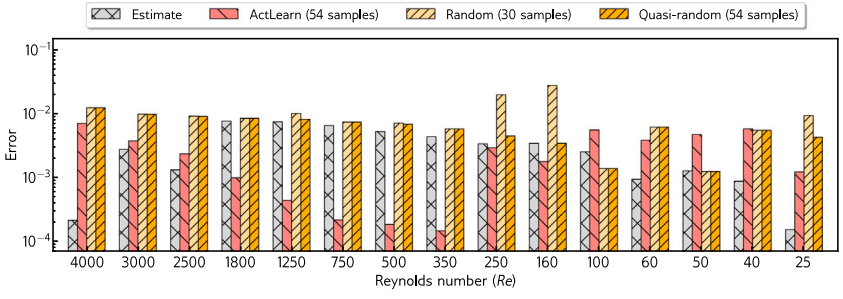
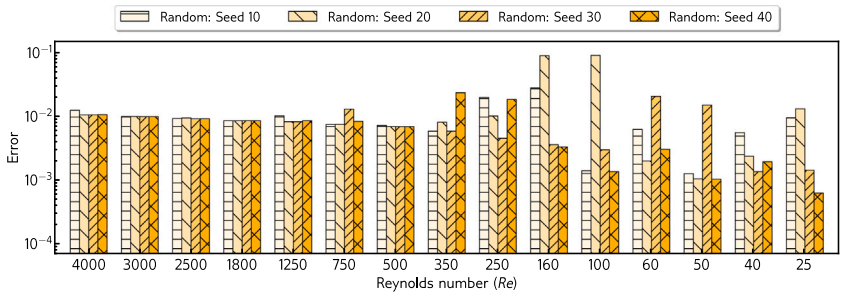
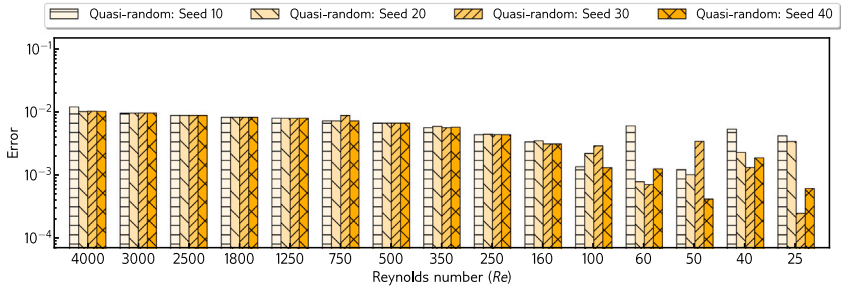


Fig. C.1. Burgers' equation: Error comparison between the ActLearn-POD-KSNN solution and the POD-KSNN solutions upon a random (30 samples, seed 10) and quasi-random (54 samples, seed 10) selection of parametric training data. The values labeled 'ActLearn', 'Random', and 'Quasi-random' are the time-averaged (over the test time grid) relative l^2 errors in the spatial domain. The values labeled 'Estimate' are the time-average (over the training time grid) of the error estimate values given by Eq. (2.16). All the reported Reynolds numbers are outside of the training set.



(a) Error in the solution upon random selection of parametric training data.



(b) Error in the solution upon quasi-random selection of parametric training data.

Fig. C.2. Burgers' equation: Error comparison between the POD-KSNN solutions upon a random (30 samples) and quasi-random (54 samples) selection of parametric training data with four different starting seeds: $\{10, 20, 30, 40\}$. The error values are the time-averaged (over the test time grid) relative l^2 errors in the spatial domain. All the reported Reynolds numbers are outside of the training set.

surrogate solution error obtained by training with 54 random samples as quasi-random because this choice is actually informed by the total number of parameter samples ($21 + 33 = 54$) upon termination of the active learning loop. The random selection from a pool of 100 values of Re (which are the same as described in Section 5.1 during the preparation of sets P and P^*) is done with a fixed representative seed value of 10 using the NumPy library written for Python programming language. The surrogate solutions $(\mathbf{u}_s)_k(Re) := \mathbf{u}_s(t_k, Re)$ are first computed over the test time grid (t_k with $k = 1, \ldots, 99$, $t_1 = 0.01$, a uniform step size of 0.02, and the total discrete time instances are $\tilde{N}_t = 99$) after which the relative l^2 error values in space are obtained,

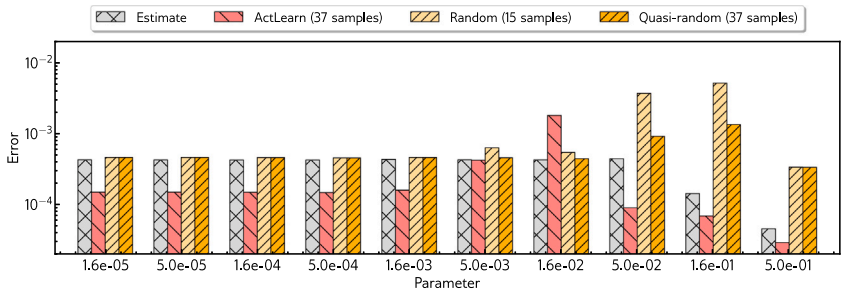
$$\varepsilon_k(Re) := \frac{\|\mathbf{u}_k(Re) - (\mathbf{u}_s)_k(Re)\|_2}{\|\mathbf{u}_k(Re)\|_2}. \tag{C.1}$$

We then take the average of these l^2 errors over all the discrete time instances, i.e., Error $:= (1/\tilde{N}_t)\sum_k \varepsilon_k(Re)$, and report the final result for active, random, and quasi-random sampling in Fig. C.1. In a similar fashion, we report the time-average of the error estimate values $\tilde{\varepsilon}_j(Re)$ defined in Eq. (2.16) on the training time grid (t_j with $j = 1, \ldots, 100$, $t_1 = 0.0$, a uniform step size of 0.02, and the total discrete time instances are $N_t = 100$), i.e., Estimate $:= (1/N_t)\sum_j \tilde{\varepsilon}_j(Re)$. We observe that the error in the ActLearn-POD-KSNN solution is bounded by the tolerance of 10^{-2} specified for termination of the active learning procedure.

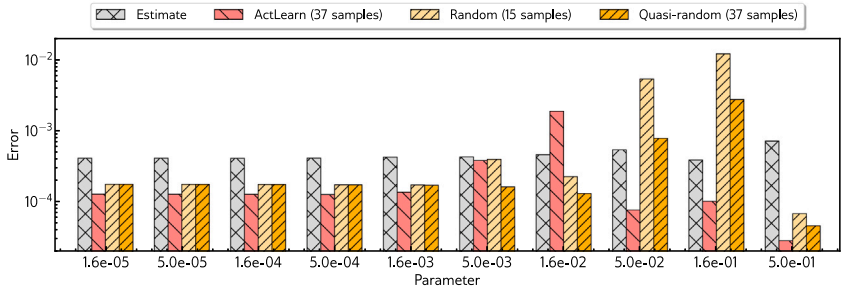
In Fig. C.2, we compare the POD-KSNN solution error for scenarios when 30 and 54 parameters are randomly picked with four different starting seed values: $\{10, 20, 30, 40\}$. Similar to Fig. C.1, the reported error values are the time-average (over the test time grid) of the relative l^2 error in space. Fig. C.2(a) shows that with 30 random samples the error goes up to 10^{-1}, and there is also significant difference among the solution error values corresponding to certain testing Reynolds numbers for different starting seeds. We observe from Fig. C.2(b) that the error values reduce by adding more samples, but there is still a noticeable variation between the values for different starting seeds. This situation is addressed by the active learning framework. Moreover, the procedure provides an idea about the number of parameter samples required for preparing the snapshot training data such that the constructed surrogate model approximates the solution (at any new parameter and time instance) up to some predefined accuracy level, as specified by the tolerance value.

C.2. Shallow water equations

Fig. C.3 provides a comparative study between the ActLearn-POD-KSNN solution error and the POD-KSNN solution errors that are obtained by randomly picking 15 and 37 parameter samples. The errors in fluid height and velocity are respectively reported in Figs. C.3(a) and C.3(b). Similar to the Burgers' equation example, we label the surrogate solution error obtained by training with 37 random samples as quasi-random, because this choice is inspired from the active learning procedure. For the random selection from 100 values of ν (which are the same as described before during the preparation of sets P and P^*), we again fix the random seed in

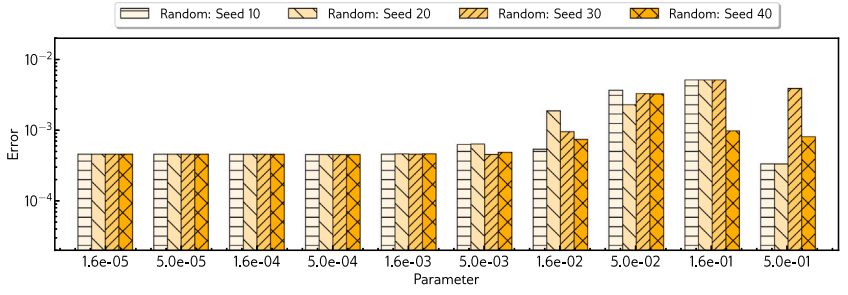


(a) Error in the fluid height for several out-of-training parameter samples ν.

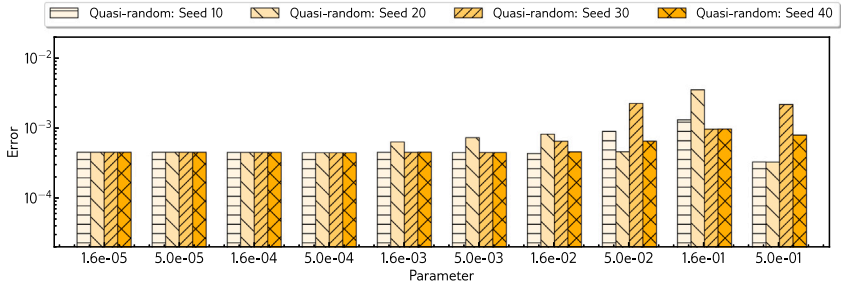


(b) Error in the fluid velocity for several out-of-training parameter samples ν.

Fig. C.3. Shallow water equations: Error comparison between the ActLearn-POD-KSNN solution and the POD-KSNN solutions upon a random (15 samples, seed 10) and quasi-random (37 samples, seed 10) selection of parametric training data. The values labeled 'ActLearn', 'Random', and 'Quasi-random' are the time-averaged (over the test time grid) relative l^2 errors in the spatial domain. The values labeled 'Estimate' are the time-average (over the training time grid) of the error estimate values given by Eq. (2.16).

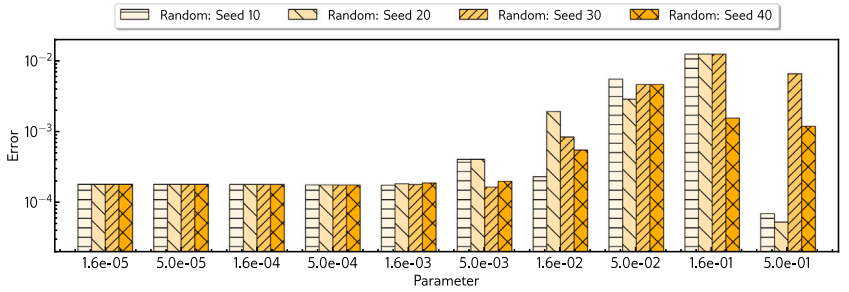


(a) Error in the fluid height upon random selection of parametric training data.

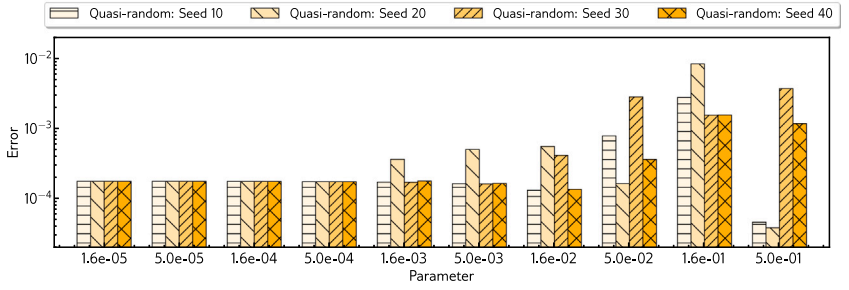


(b) Error in the fluid height upon quasi-random selection of parametric training data.

Fig. C.4. Shallow water equations: Error comparison between the POD-KSNN fluid height solution upon a random (15 samples) and quasi-random (37 samples) selection of parametric training data with four different starting seeds: $\{10, 20, 30, 40\}$. The error values are the time-averaged (over the test time grid) relative l^2 errors in the spatial domain. All the reported parameter samples ν are outside of the training set.



(a) Error in the fluid velocity upon random selection of parametric training data.



(b) Error in the fluid velocity upon quasi-random selection of parametric training data.

Fig. C.5. Shallow water equations: Error comparison between the POD-KSNN fluid velocity solution upon a random (15 samples) and quasi-random (37 samples) selection of parametric training data with four different starting seeds: $\{10, 20, 30, 40\}$. The error values are the time-averaged (over the test time grid) relative l^2 errors in the spatial domain. All the reported parameter samples ν are outside of the training set.

NumPy to 10. All the surrogate error values and the estimates in Fig. C.3 are computed in the same way as those in Fig. C.1, i.e., the relative l^2 errors in the spatial domain are time-averaged over the test time grid (t_k with $k = 1, \ldots, 499$, $t_1 = 0.004$, a uniform step size of 0.004, and the total discrete time instances are $\tilde{N}_t = 499$), whereas, the reported error estimate values are time-averaged over the training time grid (t_j with $j = 1, \ldots, 200$, $t_1 = 0.0$, a uniform step size of 0.01, and the total discrete time instances are $N_t = 200$).

In Figs. C.4 and C.5, we compare the error in the fluid height and velocity approximated by the POD-KSNN surrogate for scenarios when 15 and 37 parameters are randomly picked with four different starting seed values: $\{10, 20, 30, 40\}$. Similar to Fig. C.3, the reported error values are the time-average (over the test time grid) of the relative l^2 error in space. Figs. C.4(a) and C.5(a) show that with 15 random samples, the error goes up to 10^{-2}, and similar to our observation for Burgers' equation, there is also noticeable difference among the solution error values corresponding to certain testing viscosities for different starting seeds. The error values reduce by adding more samples as seen in Figs. C.4(b) and C.5(b), but there is still a noticeable variation between the values for different starting seeds. So, the accuracy of the surrogate solution is dependent on how the random sampling is done, i.e., the choice of the starting seed. The active learning procedure resolves this situation by picking the parameter samples based on an optimality criterion.

From Fig. C.3 we notice that the error of the ActLearn-POD-KSNN solution is generally the lowest and bounded by the tolerance of 10^{-3} specified for termination of the active learning procedure. For $\nu = 1.6 \times 10^{-2}$, the error in the fluid height is 1.79×10^{-3} and the fluid velocity is 1.91×10^{-3}, which is slightly higher than the tolerance. However, these error values are still lower than the maximum error values we observe with random and quasi-random sampling in Figs. C.4 and C.5, i.e., for $\nu = 1.6 \times 10^{-1}$ with a starting seed of 20. The active learning procedure gives an idea about the most informative parameter samples useful for preparing the snapshot training data. This way the ActLearn-POD-KSNN surrogate solution provides a reasonable accuracy without oversampling the parameter space for preparation of the training snapshot data, thereby staying computationally efficient.

Remark C.1 (***Execution time for POD-KSNN surrogate model***)**.** For the POD-KSNN surrogate with random sampling, it is not a priori clear how many parameter samples one shall select for any desired accuracy level of the surrogate solution. As a result, to compare the computational runtime of the ActLearn-POD-KSNN surrogate with the POD-KSNN surrogate, we build the POD-KSNN surrogate by randomly picking 50 viscosity samples (with the random seed as 20), expecting a decent error behavior. The total offline time for the POD-KSNN surrogate is 12479.42 seconds, as opposed to 9235.85 seconds for the ActLearn-POD-KSNN surrogate. Whereas, the online time for the POD-KSNN surrogate is 0.021 seconds, as opposed to 0.0164 seconds for the ActLearn-POD-KSNN surrogate. Please refer Table 5.1 for further details about the ActLearn-POD-KSNN surrogate timings for the shallow water equations.

References

[1] M. Guo, J.S. Hesthaven, Reduced order modeling for nonlinear structural analysis using Gaussian process regression, Comput. Methods Appl. Mech. Engrg. 341 (2018) 807–826, http://dx.doi.org/10.1016/j.cma.2018.07.017.

[2] M. Guo, J.S. Hesthaven, Data-driven reduced order modeling for time-dependent problems, Comput. Methods Appl. Mech. Engrg. 345 (2019) 75–99, http://dx.doi.org/10.1016/j.cma.2018.10.029.

[3] M. Kast, M. Guo, J.S. Hesthaven, A non-intrusive multifidelity method for the reduced order modeling of nonlinear problems, Comput. Methods Appl. Mech. Engrg. 364 (2020) 112947, http://dx.doi.org/10.1016/j.cma.2020.112947.

[4] D. Xiao, F. Fang, C. Pain, I. Navon, A parameterized non-intrusive reduced order model and error analysis for general time-dependent nonlinear partial differential equations and its applications, Comput. Methods Appl. Mech. Engrg. 317 (2017) 868–889, http://dx.doi.org/10.1016/j.cma.2016.12.033.

[5] W. Chen, J.S. Hesthaven, B. Junqiang, Y. Qiu, Z. Yang, Y. Tihao, Greedy nonintrusive reduced order model for fluid dynamics, AIAA J. 56 (12) (2018) 4927–4943, http://dx.doi.org/10.2514/1.J056161.

[6] W.J. Kostorz, A.H. Muggeridge, M.D. Jackson, An efficient and robust method for parameterized non-intrusive reduced-order modeling, Internat. J. Numer. Methods Engrg. 121 (2020) 4674–4688, http://dx.doi.org/10.1002/nme.6461.

[7] S. Dutta, M.W. Farthing, E. Perracchione, G. Savant, M. Putti, A greedy non-intrusive reduced order model for shallow water equations, J. Comput. Phys. 439 (2021) 110378, http://dx.doi.org/10.1016/j.jcp.2021.110378.

[8] J.N. Kani, A.H. Elsheikh, Reduced-order modeling of subsurface multi-phase flow models using deep residual recurrent neural networks, Transp. Porous Media 126 (2018) 713–741, http://dx.doi.org/10.1007/s11242-018-1170-7.

[9] J.S. Hesthaven, S. Ubbiali, Non-intrusive reduced order modeling of nonlinear problems using neural networks, J. Comput. Phys. 363 (2018) 55–78, http://dx.doi.org/10.1016/j.jcp.2018.02.037.

[10] A. Mohan, D.V. Gaitonde, A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks, arXiv preprint arXiv:1804.0926v1, 2018, http://dx.doi.org/10.48550/arXiv.1804.09269, physics.comp-ph.

[11] Z. Wan, P. Vlachas, P. Koumoutsakos, T. Sapsis, Data-assisted reduced-order modeling of extreme events in complex dynamical systems, PLoS One 13 (5) (2018) 1–22, http://dx.doi.org/10.1371/journal.pone.0197704.

[12] Q. Wang, J.S. Hesthaven, D. Ray, Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem, J. Comput. Phys. 384 (2019) 289–307, http://dx.doi.org/10.1016/j.jcp.2019.01.031.

[13] K. Bhattacharya, B. Hosseini, N.B. Kovachki, A.M. Stuart, Model reduction and neural networks for parametric PDEs, SMAI J. Comput. Math. 7 (2021) 121–157, http://dx.doi.org/10.5802/smai-jcm.74.

[14] S.A. Renganathan, R. Maulik, V. Rao, Machine learning for nonintrusive model order reduction of the parametric inviscid transonic flow past an airfoil, Phys. Fluids 32 (4) (2020) 047110, http://dx.doi.org/10.1063/1.5144661.

[15] W. Chen, Q. Wang, J.S. Hesthaven, C. Zhang, Physics-informed machine learning for reduced-order modeling of nonlinear problems, J. Comput. Phys. 446 (2021) 110666, http://dx.doi.org/10.1016/j.jcp.2021.110666.

[16] S. Fresca, L. Dedé, A. Manzoni, A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs, J. Sci. Comput. 87 (61) (2021) 1–36, http://dx.doi.org/10.1007/s10915-021-01462-7.

[17] F.J. Gonzalez, M. Balajewicz, Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems, arXiv preprint arXiv:1808.01346v2, 2018, http://dx.doi.org/10.48550/arXiv.1808.01346, math.DS.

[18] S.E. Otto, C.W. Rowley, Linearly recurrent autoencoder networks for learning dynamics, SIAM J. Appl. Dyn. Syst. 18 (1) (2019) 558–593, http://dx.doi.org/10.1137/18M1177846.

[19] A. Quarteroni, A. Manzoni, F. Negri, Reduced Basis Methods for Partial Differential Equations, in: La Matematica per il 3+2, vol. 92, Springer International Publishing, 2016, http://dx.doi.org/10.1007/978-3-319-15431-2.

[20] J.S. Hesthaven, G. Rozza, B. Stamm, Certified Reduced Basis Methods for Parametrized Partial Differential Equations, in: SpringerBriefs in Mathematics, Springer International Publishing, 2016, http://dx.doi.org/10.1007/978-3-319-22470-1.

[21] S. Chellappa, L. Feng, P. Benner, An adaptive sampling approach for the reduced basis method, in: Realization and Model Reduction of Dynamical Systems - A Festschrift in Honor of the 70th Birthday of Thanos Antoulas, Springer, Cham, 2022, pp. 137–155, http://dx.doi.org/10.1007/978-3-030-95157-3_8.

[22] D. Hartmann, L.K. Mestha, A deep learning framework for model reduction of dynamical systems, in: Proceedings of 2017 IEEE Conference on Control Technology and Applications, CCTA, 2017, pp. 1917–1922, http://dx.doi.org/10.1109/CCTA.2017.8062736.

[23] K. Lee, K.T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, J. Comput. Phys. 404 (2020) 108973, http://dx.doi.org/10.1016/j.jcp.2019.108973.

[24] B.A. Freno, K.T. Carlberg, Machine-learning error models for approximate solutions to parameterized systems of nonlinear equations, Comput. Methods Appl. Mech. Engrg. 348 (2019) 250–296, http://dx.doi.org/10.1016/j.cma.2019.01.024.

[25] J.N. Kani, H. Elsheikh, DR-RNN: A deep recurrent neural network for model reduction, arXiv preprint arXiv:1709.00939v1, 2017, http://dx.doi.org/10.48550/arXiv.1709.00939, cs.CE.

[26] D. Xiao, Error estimation of the parametric non-intrusive reduced order model using machine learning, Comput. Methods Appl. Mech. Engrg. 355 (2019) 513–534, http://dx.doi.org/10.1016/j.cma.2019.06.018.

[27] S. Trehan, K.T. Carlberg, L.J. Durlofsky, Error modeling for surrogates of dynamical systems using machine learning, Internat. J. Numer. Methods Engrg. 112 (12) (2017) 1801–1827, http://dx.doi.org/10.1002/nme.5583.

[28] Q. Zhuang, D. Hartmann, H.J. Bungartz, J.M. Lorenzi, Active-learning-based nonintrusive model order reduction, Data-Cent. Eng. 4 (2023) e2, http://dx.doi.org/10.1017/dce.2022.39.

[29] B. Peherstorfer, K. Willcox, Online adaptive model reduction for nonlinear systems via low-rank updates, SIAM J. Sci. Comput. 37 (4) (2015) A2123–A2150, http://dx.doi.org/10.1137/140989169.

[30] B. Peherstorfer, Model reduction for transport-dominated problems via online adaptive bases and adaptive sampling, SIAM J. Sci. Comput. 42 (5) (2020) A2803–A2836, http://dx.doi.org/10.1137/19M1257275.

[31] J. Reiss, P. Schulze, J. Sesterhenn, V. Mehrmann, The shifted proper orthogonal decomposition: A mode decomposition for multiple transport phenomena, SIAM J. Sci. Comput. 40 (3) (2018) A1322–A1344, http://dx.doi.org/10.1137/17M1140571.

[32] E. Anderson, Z. Bai, C. Bischof, L.S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, third ed., Society for Industrial and Applied Mathematics, 1999, http://dx.doi.org/10.1137/1.9780898719604.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems. Vol. 32, Curran Associates Inc, Red Hook, NY, USA, 2019, URL https://papers.nips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[34] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: International Conference on Learning Representations, ICLR, San Diego, CA, USA, 2015.

[35] C.E. Rasmussen, C.K.I. Williams, Gaussian Processes for Machine Learning, The MIT Press, 2005, http://dx.doi.org/10.7551/mitpress/3206.001.0001.

[36] M.L. Stein, Interpolation of spatial data, in: Springer Series in Statistics, Springer, New York, NY, 2012, http://dx.doi.org/10.1007/978-1-4612-1494-6.

[37] L. Sirovich, Turbulence and the dynamics of coherent structures part I: Coherent structures, Quart. Appl. Math. 45 (3) (1987) 561–571, http://dx.doi.org/10.1090/qam/910462.
[38] E.J. Candès, X. Li, Y. Ma, J. Wright, Robust principal component analysis? J. Assoc. Comput. Mach. 58 (3) (2011) 1–37, http://dx.doi.org/10.1145/1970392.1970395.
[39] V. Chandrasekaran, S. Sanghavi, P.A. Parrilo, A.S. Willsky, Rank-sparsity incoherence for matrix decomposition, SIAM J. Optim. 21 (2) (2011) 572–596, http://dx.doi.org/10.1137/090761793.
[40] D.P. Bertsekas, Constrained Optimization and Lagrange Multiplier Methods, Athena Scientific, 1996, ISBN:1886529043.
[41] Z. Lin, M. Chen, Y. Ma, The augmented Lagrange multiplier method for exact recovery of corrupted low-rank matrices, arXiv preprint arXiv:1009.5055v3, 2013, http://dx.doi.org/10.48550/arXiv.1009.5055, math.OC.
[42] I. Scherl, B. Strom, J.K. Shang, O. Williams, B.L. Polagye, S.L. Brunton, Robust principal component analysis for modal decomposition of corrupt fluid flows, Phys. Rev. Fluids 5 (2020) 054401, http://dx.doi.org/10.1103/PhysRevFluids.5.054401.
[43] P.J. Schmid, Dynamic mode decomposition of numerical and experimental data, J. Fluid Mech. 656 (2010) 5–28, http://dx.doi.org/10.1017/S0022112010001217.
[44] C.W. Rowley, I. Mezić, S. Bagheri, P. Schlatter, D.S. Henningson, Spectral analysis of nonlinear flows, J. Fluid Mech. 641 (2009) 115–127, http://dx.doi.org/10.1017/S0022112009992059.
[45] J. Nocedal, S.J. Wright, Numerical Optimization, in: Springer Series in Operations Research and Financial Engineering, Springer, New York, NY, 2006, http://dx.doi.org/10.1007/978-0-387-40065-5.
[46] X. Yuan, J. Yang, Sparse and low rank matrix decomposition via alternating direction method, Pac. J. Optim. 9 (2009) URL https://www.researchgate.net/publication/228928183_Sparse_and_low_rank_matrix_decomposition_via_alternating_direction_method.
[47] L.C. Evans, Partial Differential Equations, second ed., in: Graduate Studies in Mathematics, vol. 19, American Mathematical Society, 2010, ISBN: 9780821849743.
[48] H. Chanson, Environmental Hydraulics of Open Channel Flows, Butterworth-Heinemann, Oxford, 2004, http://dx.doi.org/10.1016/B978-0-7506-6165-2.X5028-0.
[49] H. Kapadia, Discontinuous Galerkin Schemes for Extended Shallow Water Models, Master's thesis, RWTH Aachen University, Aachen, Germany, 2019.
[50] J. Kowalski, M. Torrilhon, Moment approximations and model cascades for shallow flow, Commun. Comput. Phys. 25 (3) (2018) 669–702, http://dx.doi.org/10.4208/cicp.OA-2017-0263.
[51] A. Logg, K. Mardal, G.N. Wells (Eds.), Automated Solution of Differential Equations by the Finite Element Method, in: Lecture Notes in Computational Science and Engineering, Springer, Berlin, Heidelberg, 2012, http://dx.doi.org/10.1007/978-3-642-23099-8.
[52] R. Milk, S. Rave, F. Schindler, pyMOR – generic algorithms and interfaces for model order reduction, 38 (5) (2016) S194–S216, http://dx.doi.org/10.1137/15M1026614.
[53] P. Mlinarić, S. Rave, J. Saak, Parametric model order reduction using pyMOR, in: P. Benner, T. Breiten, H. Faßbender, M. Hinze, T. Stykel, R. Zimmermann (Eds.), Model Reduction of Complex Dynamical Systems, in: International Series of Numerical Mathematics, vol. 171, Birkhäuser, Cham, 2021, pp. 357–367, http://dx.doi.org/10.1007/978-3-030-72983-7_17.
[54] E.H. Hirschel (Ed.), Flow Simulation with High-Performance Computers II, in: Notes on Numerical Fluid Mechanics, vol. 48, Vieweg+Teubner Verlag Wiesbaden, 1996, http://dx.doi.org/10.1007/978-3-322-89849-4.
[55] K. Goda, A multistep technique with implicit difference schemes for calculating two- or three-dimensional cavity flows, J. Comput. Phys. 30 (1) (1979) 76–95, http://dx.doi.org/10.1016/0021-9991(79)90088-3.
[56] H. Kapadia, Active-Learning-Driven Surrogate Modeling, GitLab, 2023, https://gitlab.mpi-magdeburg.mpg.de/kapadia/active-learning-surrogate-modeling.
[57] H. Kapadia, Code and Data for "Active-Learning-Driven Surrogate Modeling for Efficient Simulation of Parametric Nonlinear Systems", Zenodo, 2023, http://dx.doi.org/10.5281/zenodo.10237734.