Computers and Fluids 179 (2019) 704–717

Contents lists available at ScienceDirect

Computers and Fluids

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

Projection-based model reduction: Formulations for physics-based machine learning☆

Renee Swischuk[a], Laura Mainini[a], Benjamin Peherstorfer[b], Karen Willcox[a,*]

[a] Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[b] University of Wisconsin-Madison, Madison, WI 53706, USA

ARTICLE INFO

Article history:
Received 1 February 2018
Revised 6 June 2018
Accepted 31 July 2018
Available online 1 August 2018

Keywords:
Model reduction
Data-driven reduced models
Physics-based machine learning
Proper orthogonal decomposition
Surrogate models

ABSTRACT

This paper considers the creation of parametric surrogate models for applications in science and engineering where the goal is to predict high-dimensional output quantities of interest, such as pressure, temperature and strain fields. The proposed methodology develops a low-dimensional parametrization of these quantities of interest using the proper orthogonal decomposition (POD), and combines this parametrization with machine learning methods to learn the map between the input parameters and the POD expansion coefficients. The use of particular solutions in the POD expansion provides a way to embed physical constraints, such as boundary conditions and other features of the solution that must be preserved. The relative costs and effectiveness of four different machine learning techniques—neural networks, multivariate polynomial regression, k-nearest-neighbors and decision trees—are explored through two engineering examples. The first example considers prediction of the pressure field around an airfoil, while the second considers prediction of the strain field over a damaged composite panel. The case studies demonstrate the importance of embedding physical constraints within learned models, and also highlight the important point that the amount of model training data available in an engineering setting is often much less than it is in other machine learning applications, making it essential to incorporate knowledge from physical models.

1. Introduction

This paper explores connections between model reduction and machine learning, with a focus on using concepts from model reduction to achieve a parametrization of the learning problem that embeds physical constraints and aids interpretability. While machine learning has revolutionized modeling and decision-making in a number of application fields, its black-box approach of learning a model entirely from a stream of large data does not seamlessly carry over to applications in engineering and science. One challenge is to ensure that the models reflect physical constraints, such as conservation laws and other governing equations. A second challenge is that in many engineering and scientific applications, data are expensive to generate (both computationally and experimentally) and thus the amount of data available from which to learn may be relatively sparse. Yet another challenge lies in the need to have confidence in a model's predictive power—especially since engineering models are often used to issue predictions in settings for which little-to-no data are available (e.g., predicting failure boundaries in the design of an engineering system). Data-driven learning has a large role to play in improving model capabilities, but these challenges can only be addressed by an integrated perspective that also accounts for the underlying scientific principles. To quote the recent paper by Coveney, Dougherty and Highfield: "Big Data needs Big Theory too." [1]

The field of *model reduction* encompasses a broad range of methods that seek efficient low-dimensional representations of an underlying high-fidelity model. The majority of model reduction methods have targeted the case where the high-fidelity model is a high-dimensional system of ordinary differential equations or a system of equations stemming from the discretization of partial differential equations that characterize the essential physics of the system under consideration. A large class of model reduction methods are projection-based; that is, they derive the low-dimensional approximation by projection of the original model onto a low-dimensional subspace (or, more generally, a low-dimensional manifold). The projection framework can be combined with various ways of representing parametric dependence, where, for example, the parameters might describe material prop-

☆ Paper submitted as part of the special issue of the 19th International Conference on Finite Elements in Flow Problems.

* Corresponding author.
E-mail address: kwillcox@mit.edu (K. Willcox).

https://doi.org/10.1016/j.compfluid.2018.07.021
0045-7930/

erties, system geometry, system configuration, initial conditions, and boundary conditions. See for example [2–8] for summaries of state-of-the-art in projection-based parametric model reduction methods and applications.

Model reduction has clear connections to machine learning. In fact, many of the methods used to determine the low-dimensional subspace are closely related to machine learning methods (e.g., the proper orthogonal decomposition (POD) [9–11], perhaps the most widely used model reduction method, is very closely related to the principal component analysis). The difference in fields is perhaps largely one of history and perspective: model reduction methods have grown from the scientific computing community, with a focus on *reducing* high-dimensional models that arise from physics-based modeling, whereas machine learning has grown from the computer science community, with a focus on *creating* low-dimensional models from black-box data streams. Yet recent years have seen an increased blending of the two perspectives and a recognition of the associated opportunities.

Model reduction methods have been presented that build off a set of "snapshots"—that is, solutions computed with the high-fidelity model for different inputs. These methods first project snapshot data onto a low-dimensional space and then use the low-dimensional, projected snapshots as training data to build the reduced model as a map from the inputs to the low-dimensional representations of the high-fidelity model data. Both interpolation and machine-learning-based methods have been used to determine this map. The low-dimensional spaces are typically computed with POD from snapshot data of the high-fidelity model, but other basis construction techniques such as the reduced basis method [3] have been used as well. Perhaps the earliest example of this is [12], which computes a POD representation of the temperature field in a Rayleigh–Bénard convection problem, and then uses a cubic spline interpolation to predict the temperature field at Rayleigh numbers not included in the snapshot training data. The work in [13] reconstructs POD representations of aerodynamic quantities from incomplete data using interpolation via the gappy POD [14]. Other approaches use Gaussian process regression to build a model for the POD coefficients as a function of the inputs [15–17]. The work in [18] learns the map from inputs to POD coefficients using an adaptive combination of self-organizing maps and local response surfaces, and the work in [19,20] learns the map from inputs to POD coefficients with neural networks. In [21], learning the POD coefficients is coupled with a greedy approach to actively guide the sampling in the input domain.

Instead of learning a map from the inputs to the coefficients of POD representations, data-driven model reduction seeks to learn the *operators* of reduced models. Thus, these data-driven model reduction techniques provide a way to learn the *dynamics* of the system of interest in the form of reduced-model operators that respect some of the structure that arises through the underlying governing equations. Data-driven model reduction shares similarities with classical system identification [22], where dynamical-system models are extracted from time- and frequency-domain measurements. The eigensystem realization algorithm [23–26] and finite impulse response system identification [22,27–29] are two system identification techniques that seek to learn reduced models from impulse responses of linear time-invariant systems. The Loewner framework [30–34] provides a more flexible approach by learning reduced models from frequency-response data (i.e., transfer function values), instead of requiring impulse response data. The Loewner framework has been extended to learning from time-domain data [35,36]. Vector fitting is a regression-based approach to learn a reduced model from frequency-response data [37,38]. Data-driven model reduction methods based on dynamic mode decomposition learn linear reduced operators that best-fit given data in the L_2 norm [39–41]. The work in [42] uses least-squares regression to find operators in a similar way to dynamic mode decomposition but is applicable to models with low-order polynomial terms. The work in [43] uses a recursive variable selection strategy to learn a reduced model, with a focus on retaining physical interpretability. This approach is combined with model predictive control to optimize control decisions for real-time operation of a reservoir in [44].

A different body of literature leverages sparsity-promoting regression techniques to learn reduced models from data. One line of work employs ℓ_1 regularization to select reduced-model components from a library [41,45]. Another line of work seeks to recover [46,47] or to update [48–50] reduced models from sparse data. The work in [51–54] augments projection-based reduced models with correction terms learned from data. Multifidelity and multi-information source methods provide another way of combining reduced models with data, see, e.g., [55–58].

Despite this wealth of literature, enforcing physical constraints in machine learning-based models remains an open challenge. In this paper we address this challenge by developing a POD representation as a way to create a physics-inspired parametrization that embeds physical constraints. We then present a detailed study of the relative cost and effectiveness of different machine learning techniques in this setting. Section 2 presents the POD methodology and describes the use of particular solutions to enforce problem structure. Section 3 presents the setup of the machine-learning problem in the low-dimensional POD space and briefly describes the four machine learning methods that are employed. More general perspectives on the machine-learning techniques used by our approach are given in, e.g., [59–62]. Sections 4 and 5 present two case studies, respectively prediction of the pressure field around an airfoil and prediction of the strain field over a damaged composite plate. Finally, Section 6 concludes the paper.

2. A physics-inspired parametrization of physical fields

We consider systems in science and engineering that respond to inputs with physical fields, for example representing such physical quantities as pressure, temperature, stress, strain, etc. This section establishes a physics-inspired parametrization of such fields via the proper orthogonal decomposition.

2.1. Numerical approximation of physical fields

Consider a system that maps an input onto a physical field. The physical field is our prediction quantity of interest. We denote a field as a function $q : \mathcal{X} \times \mathcal{T} \times \mathcal{P} \to \mathbb{R}$, with the spatial domain $\mathcal{X}$, time domain $\mathcal{T}$, and input domain $\mathcal{P}$. Thus, the field q varies in space and time, and depends on the input of the system. The focus of this paper is on learning approximate models of q from data $\mathcal{D} \subset \{q(\mathbf{x}, t; \mathbf{p}) \,|\, \mathbf{x} \in \mathcal{X}, t \in \mathcal{T}, \mathbf{p} \in \mathcal{P}\}$ in a way that respects the underlying physical constraints of the system, thus leading to models endowed with scientific interpretability and predictive power.

To begin, we recognize that the behavior of these systems is characterized by physical laws and governing equations, which are often represented in the form of partial differential equations. In a classical computational science setting, numerical models discretize the governing equations, approximating the solution fields in different ways. For example, a finite difference approximation represents the solution at a set of discrete points in the space-time domain, while a finite element method represents the solution using an expansion in a finite number of basis functions. Whichever numerical discretization method is chosen, the result is a numerical model that embeds the physical governing equations. The dimensionality of these models is typically high, which means that large-scale systems of equations have to be solved to evaluate the models. The dimension of numerical models can be in the

range of thousands to millions of unknowns (even more in three-dimensional time-varying simulations) [63,64].

In our data-driven setting, we seek to learn a numerical model from data. While the numerical discretization models described above lend some insight to the form of the model we might learn, it is typically infeasible to attempt to learn directly the large-scale models. Instead, we first introduce the notion of a physics-inspired low-dimensional parametrization. Such a parametrization can be derived using the POD, which computes an expansion basis that enables a low-dimensional representation of a high-dimensional system state. POD is closely related to methods used in other fields such as Karhunen–Loéve expansions in stochastic process modeling [65,66], principal component analysis in statistical analysis [67], and empirical orthogonal eigenfunctions in atmospheric modeling [68]. POD was introduced for the analysis of turbulent flows [9], and POD basis vectors are computed empirically using sampled training data, typically using the method of snapshots, introduced by Sirovich [11].

2.2. Computing the POD basis

Consider the field $q(\cdot,\ t;\ \mathbf{p})$ at time $t \in \mathcal{T}$ and input $\mathbf{p} \in \mathcal{P}$. To compute the POD basis, we consider finite-dimensional approximations $\mathbf{q}(t;\mathbf{p}) \in \mathbb{R}^{n_x}$ of $q(\cdot,\ t;\ \mathbf{p})$, where n_x is the (typically large) dimension of the finite-dimensional discretization of the spatial domain. In the POD literature, $\mathbf{q}(t;\ \mathbf{p})$ is called a "snapshot" [11] and we will collect many such snapshots in order to compute the POD basis. These snapshots may be computational solutions generated by a numerical model, or they may be sensed data (or a combination thereof). Consider the set of $n_s = n_t n_p$ snapshots, $\left\{\mathbf{q}(t_i;\mathbf{p}_j)\,|\,i=1,\ldots,n_t,\ j=1,\ldots,n_p\right\}$, which are snapshots at n_t different time instances $t_1,\ldots,t_{n_t} \in \mathcal{T}$ and n_p different inputs $\mathbf{p}_1,\ldots,\mathbf{p}_{n_p} \in \mathcal{P}$. Define the snapshot matrix $\boldsymbol{Q} \in \mathbb{R}^{n_x \times n_s}$, which contains the snapshots $\mathbf{q}(t_i;\ \mathbf{p}_j)$ as its columns. Thus, each row in the snapshot matrix corresponds to a spatial location (e.g., a discretization point for a finite difference model snapshot or a sensor location for sensed data snapshots) and each column corresponds to a snapshot.

The (thin) singular value decomposition of $\boldsymbol{Q}$ is written

$$\boldsymbol{Q} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{W}^{\top}, \tag{1}$$

where the columns of the matrices $\boldsymbol{V} \in \mathbb{R}^{n_x \times n_s}$ and $\boldsymbol{W} \in \mathbb{R}^{n_s \times n_s}$ are the left and right singular vectors of $\boldsymbol{Q}$, respectively. The singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{n_s} \geq 0$ of $\boldsymbol{Q}$ give the diagonal matrix $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_{n_s}) \in \mathbb{R}^{n_s \times n_s}$. The POD basis of dimension r, $\boldsymbol{V}_r = [\boldsymbol{v}_1,\ldots,\boldsymbol{v}_r]$, is then defined as the r left singular vectors of $\boldsymbol{Q}$ that correspond to the r largest singular values. This yields an orthonormal basis that provides an efficient low-dimensional representation of the snapshot data. Among all orthonormal bases of size r, the POD basis minimizes the least squares error of snapshot reconstruction,

$$\min_{\boldsymbol{V}_r \in \mathbb{R}^{n_x \times r}} ||\boldsymbol{Q} - \boldsymbol{V}_r\boldsymbol{V}_r^{\top}\boldsymbol{Q}||_F^2 = \sum_{k=r+1}^{n_s} \sigma_k^2. \tag{2}$$

The sum of the squares of the singular values corresponding to those left singular vectors not included in the POD basis gives the square of the error in the snapshot representation. Thus, the singular values provide quantitative guidance for choosing the size of the POD basis, based on the number of basis vectors needed to accurately represent the given snapshot data. A typical approach is to choose r so that

$$\frac{\sum_{k=1}^{r}\sigma_k^2}{\sum_{k=1}^{n_s}\sigma_k^2} > \kappa, \tag{3}$$

where κ is a user-specified tolerance, typically taken to be 90% or greater. The lefthand side of (3) is often referred to as the relative "cumulative energy" captured by the first r POD modes.

2.3. Parametrizing physical fields in the POD basis

The POD basis is learned from snapshot data of the system of interest and so provides a physics-based parametrization of the field q. For example, the field q can be approximated by a linear expansion in the POD basis:

$$\tilde{\mathbf{q}}(t;\mathbf{p}) = \sum_{k=1}^{r} \boldsymbol{v}_k \alpha_k(t;\mathbf{p}), \tag{4}$$

where $\alpha_k(t;\ \mathbf{p})$ denote the POD expansion coefficients and $\tilde{\mathbf{q}}(t;\mathbf{p})$ denotes the POD approximation of the field $q(\cdot,\ t;\ \mathbf{p})$ at time t and input $\mathbf{p}$. Given a snapshot $\mathbf{q}(t;\ \mathbf{p})$, we can compute its representation in the POD basis via the coefficients $\alpha_k(t;\mathbf{p}) = \boldsymbol{v}_k^{\top}\mathbf{q}(t;\mathbf{p}),\ k = 1,\ldots,r$, where we have used that the POD basis vectors are orthonormal. Our learning task is now transformed into learning a model for the POD coefficients $\alpha_k(t;\ \mathbf{p})$. This transformation has two advantages. First, the dimension of the unknowns has been reduced from n_x in the original discrete representation $\mathbf{q}(t;\ \mathbf{p})$ to r in the POD representation. As we will see in the example problems, typically $r \ll n_x$ for the target problems of interest. Second, the representation (4) provides a mechanism for embedding physical constraints.

2.4. Enforcing physical constraints in POD parametrizations

Mathematically, physical constraints may be enforced in a variety of ways. One approach is to impose constraints on the inference of the α_k coefficients; that is, to pose the learning problem as a constrained optimization problem. This is the approach used in [69] to conserve linear and quadratic constraints arising from the physical design problem. Another approach is to embed the constraints into the form of the POD representation. For example, we can consider an alternative representation to (4) as

$$\tilde{\mathbf{q}}(t;\mathbf{p}) = \bar{\mathbf{q}} + \sum_{k=1}^{r} \bar{\boldsymbol{v}}_k \alpha_k(t;\mathbf{p}), \tag{5}$$

where $\bar{\mathbf{q}}$ is a "particular solution," also referred to in some literature as a "static correction" [70]. The particular solution is chosen to embody particular attributes of the solution that we wish to enforce. In (5) we use the notation $\bar{\boldsymbol{v}}_k$ to emphasize that the POD basis vectors may be different to those used in (4).

As one example, consider the case where the particular solution $\bar{\mathbf{q}}$ is chosen to satisfy a particular set of prescribed inhomogeneous boundary conditions and the POD modes $\bar{\boldsymbol{v}}$ are defined so that they satisfy homogeneous boundary conditions. Then by construction, $\tilde{\mathbf{q}}$ in (5) will satisfy the inhomogeneous boundary conditions regardless of the values of α_k. To see this, we partition a quantity of interest vector as $\mathbf{q} = [\mathbf{q}^b\mathbf{q}^f]$, into entries associated with the prescribed boundary conditions, $\mathbf{q}^b$, and the remaining free entries, $\mathbf{q}^f$. Now define the particular solution $\bar{\mathbf{q}} = [\bar{\mathbf{q}}^b\ \bar{\mathbf{q}}^f]$, where $\bar{\mathbf{q}}^b$ are the desired prescribed inhomogeneous boundary conditions and $\bar{\mathbf{q}}^f$ are the remaining entries of the particular solution. The POD modes $\bar{\boldsymbol{v}}$ are computed using the same methodology described in Section 2.2, but operating on the modified snapshot set $\left\{\mathbf{q}(t_i;\mathbf{p}_j) - \bar{\mathbf{q}}\,|\,i=1,\ldots,n_t,\ j=1,\ldots,n_p\right\}$. Note that by subtracting the particular solution $\bar{\mathbf{q}}$, the modified snapshots $\mathbf{q}(t_i;\mathbf{p}_j) - \bar{\mathbf{q}}$ satisfy homogeneous boundary conditions, that is, they have the form $\mathbf{q}(t_i;\mathbf{p}_j) - \bar{\mathbf{q}} = [\mathbf{0}\ \left(\mathbf{q}^f(t_i;\mathbf{p}_j) - \bar{\mathbf{q}}^f\right)]$. Then by the properties of the singular value decomposition (i.e., that the singular vectors $\boldsymbol{v}_k$ must be linear combinations of the modified snapshots), the POD basis vectors also satisfy homogeneous boundary conditions

and the representation (5) will recover the desired inhomogeneous boundary conditions.

The idea expressed in (5) can be extended to include multiple particular solutions, as well as particular solutions scaled by functions of time and inputs or particular solutions that themselves depend upon time and/or inputs, chosen to satisfy more complicated physical conditions. In this case, we write

$$\tilde{\mathbf{q}}(t;\mathbf{p}) = \bar{\mathbf{q}}(t;\mathbf{p}) + \sum_{k=1}^{r} \bar{\boldsymbol{\nu}}_k \alpha_k(t;\mathbf{p}), \tag{6}$$

where again one must make corresponding modifications to subtract out particular solutions from the snapshot set, so that the modified snapshots—and thus the POD modes—satisfy homogenous conditions in the appropriate entries.

2.5. Particular solution illustrative example

As an illustrative example, consider the use of particular solutions to enforce boundary conditions in a model predicting the evolution of temperature in a one-dimensional heated rod of length L. The output quantity of interest, $q(x, t)$, is the temperature field over the rod, which varies as a function of distance along the rod, $0 \le x \le L$, and time, $t \ge 0$. The evolution of the temperature is governed by the heat equation, along with specified boundary conditions and initial conditions. To demonstrate how to determine an appropriate particular solution, consider the specific case that the boundary at the left end of the rod ($x = 0$) is prescribed to follow a time-dependent forcing function, $q(0,t) = \gamma_0 f(t)$, where γ_0 is a scalar amplitude and $f(t)$ specifies the time-varying component of the boundary condition, and the boundary at the right end of the rod is constrained to a fixed temperature value, $q(L,t) = \gamma_L$.

We wish to create a POD reconstruction of the form (6) that respects these two boundary conditions. To do this, we first solve an auxiliary problem with the same heated rod problem setup but with boundary conditions $q(0,t) = 0$ and $q(L,t) = 1$. Denote the resulting steady-state solution as $\bar{q}^L(x)$. This first auxiliary problem solution is used to enforce the boundary condition at $x = L$. Second, solve an auxiliary problem with boundary conditions $q(0,t) = 1$ and $q(L,t) = 0$, and denote the resulting steady-state solution as $\bar{q}^0(x)$. This second auxiliary problem solution is used to enforce the boundary condition at $x = 0$. Then to reconstruct solutions for our original problem, we define the particular solution $\bar{q}(x,t) = \gamma_0 f(t)\bar{q}^0(x) + \gamma_L \bar{q}^L(x)$. It can be seen that subtracting this particular solution off each snapshot (noting that when considering each snapshot, $f(t)$ must be evaluated at the time corresponding to that snapshot) yields a modified snapshot set with homogenous boundary conditions, which in turn leads to a POD basis that satisfies homogenous boundary conditions. Reconstruction of solutions via (6) is then guaranteed to recover the boundary conditions.

Fig. 1 plots the auxiliary solutions $\bar{q}^0$ and $\bar{q}^L$, along with the mean of a snapshot set generated by sampling different values of time and thermal diffusivity for a case with boundary conditions $q(0,t) = 3\sin 2t$ and $q(L,t) = 3$. It can be seen that the auxiliary solution $\bar{q}^L$ is qualitatively similar to the mean (which is to be expected in this specific problem setup since the average boundary condition at the left end is zero), whereas $\bar{q}^0$ introduces the behavior needed to model the time-dependent boundary condition applied to the left end of the rod. Fig. 2 shows the time-averaged errors in reconstructions (where the POD coefficients are determined using a decision tree, as described in Section 3). The enforcement of the boundary conditions is clearly indicated by the zero errors at $x = 0$ and $x = L$.

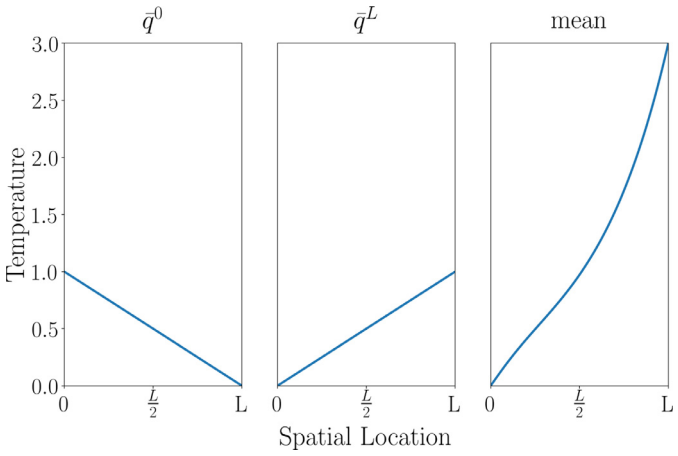


Fig. 1. Heated rod example: auxiliary solutions $\bar{q}^0$ (left) and $\bar{q}^L$ (center) used to create the particular solution, and snapshot mean (right).

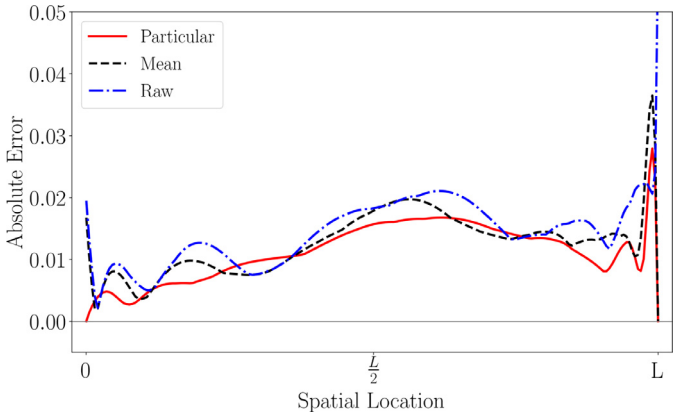


Fig. 2. Heated rod example: Time averaged errors in prediction of the temperature field over the spatial domain. Compared are enforcing the boundary conditions using a particular solution (solid), subtracting the mean solution from the snapshots (dash), and raw snapshots (dash dot).

3. Machine learning methods

The numerical examples in this paper use four different machine learning methods to infer the physics-based low-dimensional models. This section describes the learning problem setup and then provides a brief overview of each machine learning method, noting that these are standard implementations of existing methods and details are given only for completeness. Specific implementation choices for each modeling approach are given in Sections 4 and 5 for the aerodynamic and structural example problem, respectively.

3.1. Learning problem setup

In each case, we learn a surrogate model for the map $\boldsymbol{\alpha} : \mathcal{P} \to \mathcal{A}$ from inputs $\mathbf{p} \in \mathcal{P}$ to outputs $\boldsymbol{\alpha}(\mathbf{p}) \in \mathcal{A}$. The outputs $\boldsymbol{\alpha}(\mathbf{p})$ are the POD coefficients defined in Section 2; we consider the specific case of r POD coefficients, $\boldsymbol{\alpha}(\mathbf{p}) = [\alpha_1(\mathbf{p}), \ldots, \alpha_r(\mathbf{p})]$. The inputs are system parameters; we consider the specific case of m inputs: $\mathbf{p} = [p_1, \ldots, p_m]$. Note that we dropped the dependence of $\boldsymbol{\alpha}$ on time for ease of exposition.

Consider the case that we have n_s snapshots, where each snapshot corresponds to a different input. We collect the inputs corresponding to each snapshot in the matrix $\mathbf{P} \in \mathbb{R}^{n_s \times m}$. We collect the corresponding POD coefficients for each snapshot in the matrix $\mathbf{A} \in \mathbb{R}^{n_s \times r}$. The remainder of this section provides an overview of

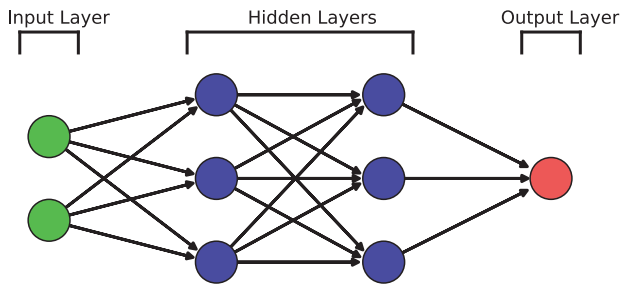


Fig. 3. Structure of a neural net with a two dimensional input, two hidden layers and a one dimensional output.

the four methods used to make predictions. In each of these methods, our input and output data, $\mathbf{P}$ and $\mathbf{A}$, are divided into training and test sets denoted by ($\mathbf{P}_{\text{train}}$, $\mathbf{A}_{\text{train}}$) and ($\mathbf{P}_{\text{test}}$, $\mathbf{A}_{\text{test}}$), respectively. The goal is to learn the map $\boldsymbol{\alpha} : \mathcal{P} \to \mathcal{A}$ from the training data ($\mathbf{P}_{\text{train}}$, $\mathbf{A}_{\text{train}}$).

3.2. Neural network

The first model considered is a fully connected, feed forward neural network. Neural network models estimate the output $\boldsymbol{\alpha}(\mathbf{p})$ at an input $\mathbf{p}$ using weights and biases that are adapted to the data during training. With the use of adaptivity, these models are structured but can achieve great flexibility. A disadvantage of using a neural net is that interpretation of the resulting model may be difficult.

A neural net consists of sequential layers of nodes that define a mapping between an input and an output. The basic structure of a neural net is shown in Fig. 3. The first layer of nodes is called the input layer, which directly receives the input to the model. The number of nodes in the input layer is equal to the number of inputs. In each hidden (i.e., not an input or output) layer, l, there are ℓ_l nodes. The ith node in layer l is denoted as η_l^i for $i \in \{1, \ldots, \ell_l\}$. Each node takes an input, evaluates an activation function, g, and produces an intermediate output, o_l^i, which is used as an input to nodes in the next layer, $l+1$. Every node in a given layer is connected to every node in the following layer (fully connected) and information is passed forward through the network (feed forward). Each of these connections are given a weight, w_l^i, and a bias, b_l^i, which defines the importance and the effect of a particular input to the output [71]. The activation function at each node is a function of $h_l^i = w_l^i o_{l-1}^i + b_l^i$ and thus depends on the input to the node, the weight assigned to the connection to that node and the bias. The output for node η_l^i is determined by evaluating the activation function $o_l^i = g(h_l^i)$. The last layer is the output layer, which produces the final output. The number of nodes in the last layer is equal to the number of output values.

To train a neural net, training inputs, $\mathbf{P}_{\text{train}}$, and corresponding training outputs, $\mathbf{A}_{\text{train}}$, are provided to the model. The training process iterates forward and backwards over the network, adjusting weights and biases so as to minimize the mean squared error between predicted and actual training outputs [72]. Each pair of forward and backward passes is referred to as an epoch. The computational complexity of training a neural net is approximately linear with the size of the network (nodes and layers), which in turn depends upon the dimension of the input and output, and with the number of epochs, assuming a smooth activation function [73].

3.3. Multivariate polynomial regression

The second model considered is multivariate polynomial regression (MPR), which approximates the outputs $\boldsymbol{\alpha}(\mathbf{p})$ using least squares regression. For the applications in this paper, we use quadratic polynomial functions. Thus, our model for a POD coefficient $\boldsymbol{\alpha}$ is written as

$$\boldsymbol{\alpha}(\mathbf{p}) = b_0 + \sum_{i=1}^{m} b_i p_i + \sum_{i=1}^{m} \sum_{j=i}^{m} c_{ij} p_i p_j, \tag{7}$$

where the b_i, $i = 0, 1, \ldots, m$ and c_{ij}, $i = 1, \ldots, m$, $j = i, \ldots, m$ are the coefficients of the quadratic model. These coefficients are determined via least squares regression that minimizes the mean squared error between the predicted and actual training outputs.

The largest computational complexity in fitting this model arises from solving the least squares system, the dimension of which scales quadratically with the number of input parameters m. Still, in our numerical experiments below, the quadratic regression model is much cheaper to train and to evaluate than the neural network model.

3.4. k-nearest-neighbors regression model

The k-nearest-neighbors (kNN) model can be interpreted as a localized form of multivariate regression: the approximation is computed over a local set of k training samples that are the closest neighbors of the evaluation point $\mathbf{p}$ in the input space. Due to the local nature of kNN approximations, the resulting models do not provide a global interpretation of the underlying relationship between inputs and outputs. Therefore kNN is an unstructured but flexible scheme, balancing the simplicity of polynomial regression with the flexibility of localized models.

In this paper, we consider a standard implementation of kNN that approximates $\boldsymbol{\alpha}(\mathbf{p})$ by averaging the outputs associated with the k nearest neighbors. The predicted output of the kNN model is the weighted average of the training outputs at each of the neighbors. This is in essence a localized linear regression model. During training, a k-d tree partitions the data along each input dimension and can be constructed in $\mathcal{O}(mn_{\text{train}})$ time, where m is the dimension of the input, without explicitly calculating any m-dimensional distances. This allows for fast nearest neighbor searches averaging on the order of $\mathcal{O}(\log(n_{\text{train}}))$ [74,75].

3.5. Decision tree regression model

The fourth model considered expands on the use of nearby neighbors to make predictions via localized regression models. A decision tree partitions the input domain into many regions and makes local average estimates of the output. This is performed in a top down fashion where the data are recursively partitioned at each node using a greedy algorithm to group similar data together. Each node is assigned a certain split criteria, which aims to make a split that creates similar valued subsets of data. Nodes in the tree continue to split the data into left and right child nodes using a series of logical statements. This process continues until certain stopping criteria are met and nodes cease to split, becoming *leaf* nodes. After the data are partitioned into multiple regions, a piecewise constant regression model is built. Within each region (i.e., within each leaf node), a constant function is fit as the average of each value in the region.

Efficient methods for constructing decision trees are available [76]. Once a tree has been learned, predictions can be made rapidly. Given an input, only simple logic statements are processed to reach a leaf node where an average value is computed to obtain the output. Partitioning the training data along each input dimension results in a computational complexity of $\mathcal{O}(mn_{\text{train}} \log(n_{\text{train}}))$

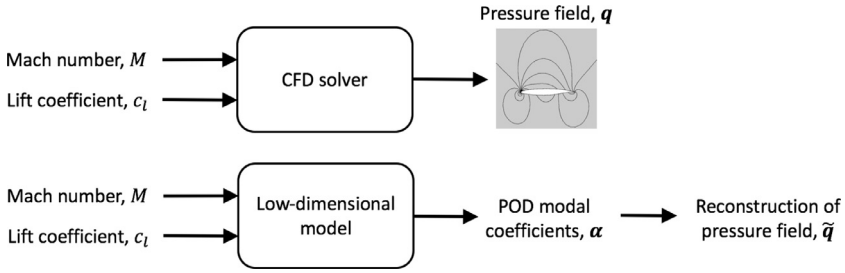
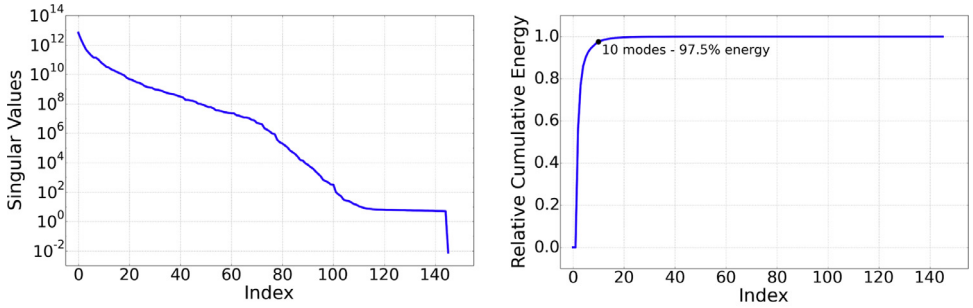


Fig. 4. Inputs and output quantity of interest for the aerodynamic example. High-fidelity CFD solver (top) and low-dimensional models (bottom).



(a) Singular values. (b) Cumulative energy.

Fig. 5. POD singular values and relative cumulative energy for the airfoil pressure field snapshot set.

to build the tree. If a decision tree is approximately balanced, this allows for fast predictions on the order of $\mathcal{O}(\log(n_{\text{train}}))$.

The following sections will illustrate how the proposed parametrization of the output space using POD is an effective approach for two engineering applications, and will explore tradeoffs in computational cost and prediction accuracy among the different machine learning methods applied in the POD space.

4. Aerodynamic example

The first case study considers the prediction of the flow around an airfoil, using data generated from a large-scale computational fluid dynamics (CFD) simulation.

4.1. Problem setup: predicting the flow over an airfoil

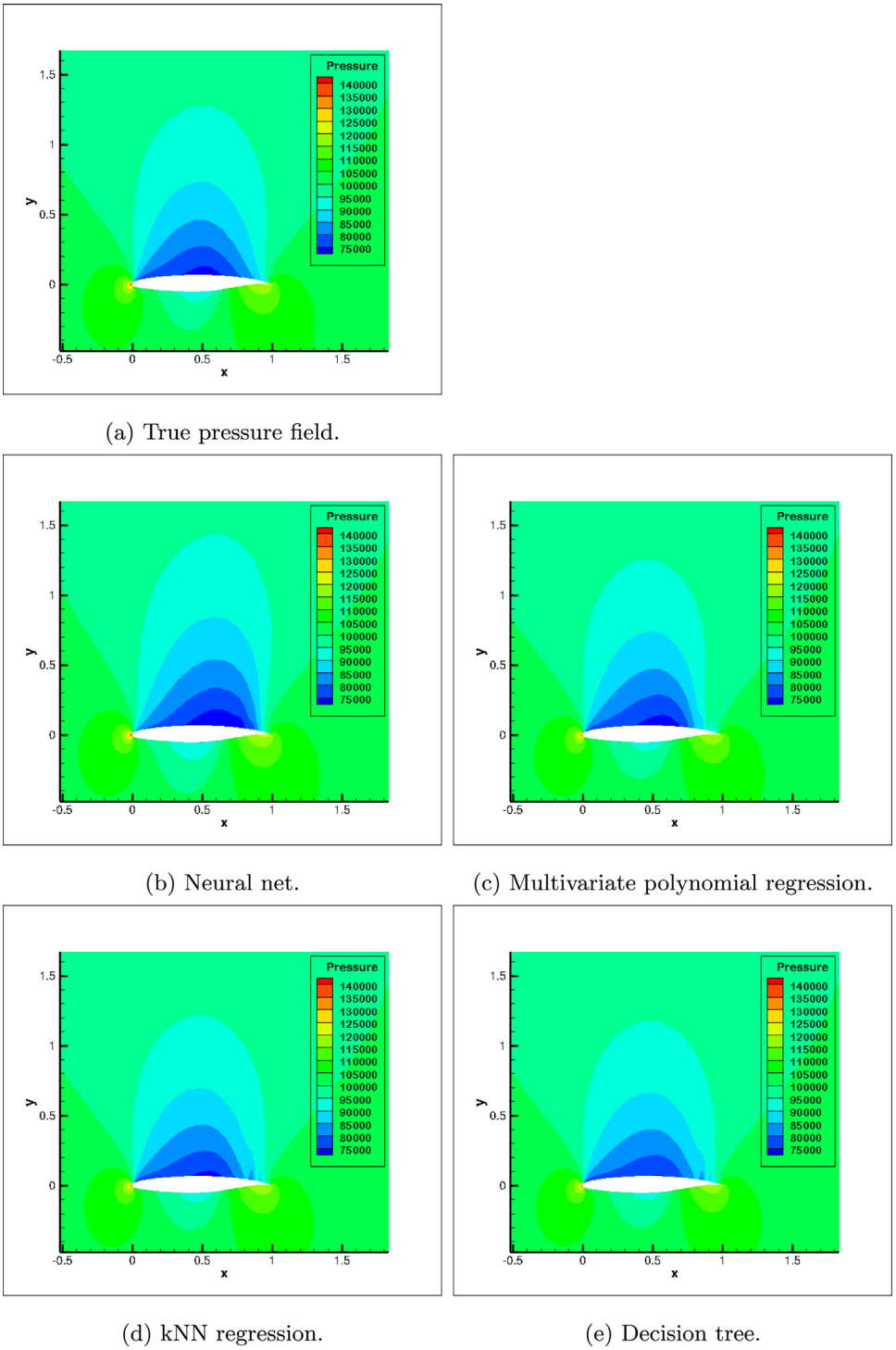
The input parameters considered are the freestream Mach number, M, and the airfoil lift coefficient, c_l. Thus we have $m = 2$ parameters with the input parameter vector $\mathbf{p} = [p_1\ p_2] = [M\ c_l]$. The output quantity of interest is the pressure field around the airfoil, which varies as a function of the input parameters. In reality, the pressure field is a continuous (infinite-dimensional field), varying over the spatial domain. An expensive physics-based model computes a high-fidelity finite-dimensional approximation of the pressure field using a CFD model. In this example, we use the SU2 CFD tool suite [77], a multi-purpose open-source solver, specifically developed for aerospace applications. SU2 uses a finite volume method to discretize the underlying partial differential equations. Here we use the Euler equations to model the inviscid steady flow over the airfoil. We consider a range of Mach numbers, spanning subsonic and transonic flow regimes. Flow tangency boundary conditions are imposed on the airfoil surface and the farfield boundary is approximately 20 chord lengths away from the airfoil.

SU2's discretization of the pressure field has $n_x = 9027$ degrees of freedom; that is, each SU2 pressure field solution is a vector of dimension $n_x = 9027$, where each entry corresponds to the predicted pressure at a different spatial location in the computational domain. We use training data generated by SU2 to learn a cheap surrogate model, but clearly this dimensionality of $n_x = 9027$ is too high to use directly as the model output in our machine learning setting. We use the methodology described in Section 2 to reduce the dimensionality of the output representation using the POD. To achieve this, snapshots are generated for a domain of Mach numbers from $M = 0.6$ to $M = 0.8$ in increments of 0.01. At each Mach number, the following seven lift coefficients are used: $c_l = 0.4, 0.5, 0.6, 0.7, 0.8, 0.85, 0.9$. This provides a total of $n_s = 147$ snapshots, where each snapshot is a high-fidelity pressure field solution, represented as a high-dimensional vector. From these snapshots, we compute the POD basis vectors. For each snapshot, we compute the corresponding POD expansion coefficients, which describe the representation of the snapshot in the POD basis according to (5), where $\bar{\mathbf{q}}$ is set to be the mean over all training snapshots. We then use the four machine learning methods described in Section 3 to fit models between the input parameters $\mathbf{p}$ and the POD coefficients $\boldsymbol{\alpha}(\mathbf{p})$. Fig. 4 summarizes the input and output configuration associated with the high-fidelity SU2 solver and the low-dimensional model.

4.2. Aerodynamic results

First, we determine the appropriate number of POD modes to use in our reduced model. Fig. 5 shows the POD singular values for the full dimensional pressure fields and the corresponding cumulative energy as defined in (3).

Based on Fig. 5, we use the first $r = 10$ POD basis vectors to represent the pressure fields. The coefficients $\boldsymbol{\alpha}(\mathbf{p}) = [\alpha_1, \ldots, \alpha_{10}]$ are thus the predictions of our surrogate models and then the predicted pressure fields are reconstructed using (5). To assess the accuracy of each model, we withhold the set of samples containing a particular Mach number from the training set and make predictions of the pressure solutions at the withheld Mach number. At each Mach number there are seven lift coefficients, so withholding a Mach number involves withholding seven samples as a test set. To illustrate the performance of the various models, Fig. 6 shows the actual and predicted pressure fields for the case of Mach number $M = 0.7$ and lift coefficient $c_l = 0.7$. All pressure fields pro-



(a) True pressure field.

(b) Neural net.

(c) Multivariate polynomial regression.

(d) kNN regression.

(e) Decision tree.

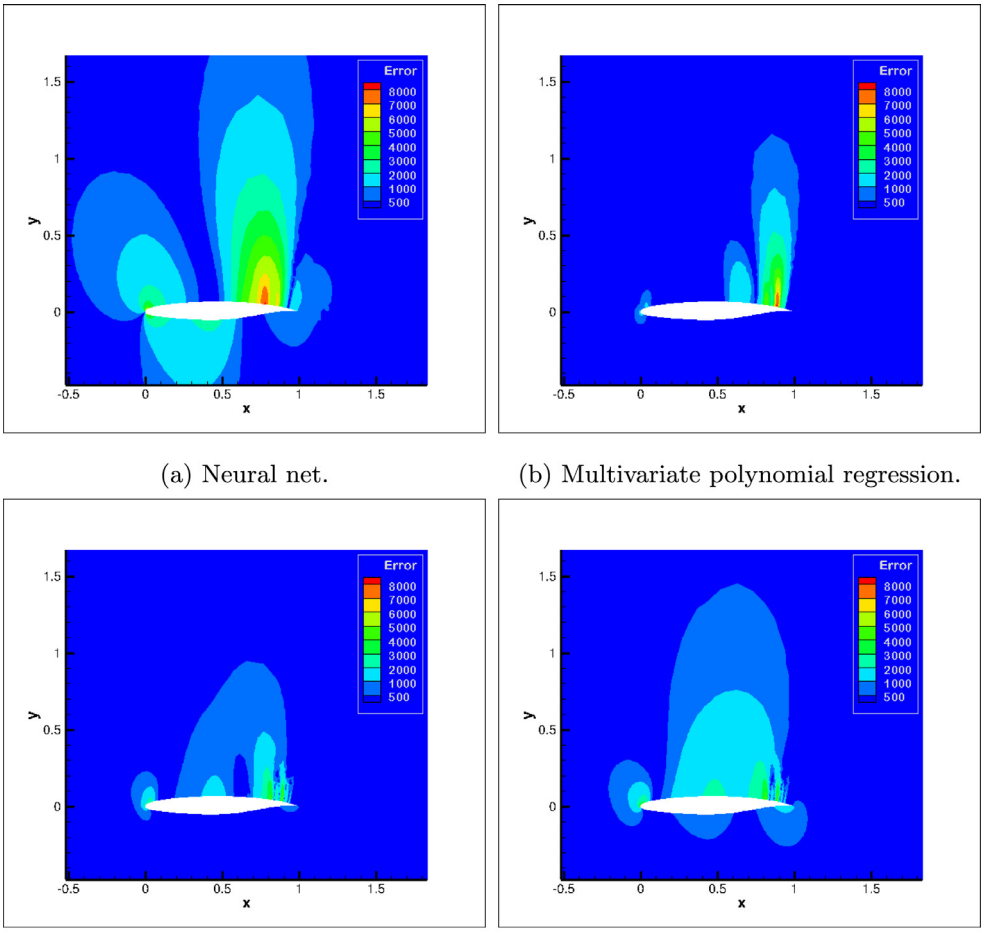
Fig. 6. True pressure field (a) and predictions using POD in combination with four machine learning techniques (b)–(e).

duced with Mach 0.7 have been held out of the training set used for making these predictions.

The true pressure field is shown in Fig. 6a. Fig. 6b shows the predicted pressure field using a neural network model. In this case, the neural net is implemented with one hidden layer containing 50 nodes and a sigmoid activation function defined as

$$g(h) = \frac{1}{1 + e^{-h}}. \tag{8}$$

The neural net cost function is defined as the mean squared error and is minimized using stochastic gradient descent with a learning rate of 0.1 and 10,000 epochs. Fig. 6c shows the predicted pressure field using multivariate polynomial regression. Our two predictor variables are Mach number, M, and lift coefficient, c_l, and the regression model is computed using quadratic (degree 2) polynomials following (7). Fig. 6d shows the predicted pressure field using kNN regression. The number of neighbors, k, is set to 5 and the weights are inversely proportional to the distance to each neighbor. Fig. 6e shows the predicted pressure field using decision tree regression. No restrictions were set on the depth of the tree, resulting in 279 nodes in total, 140 of those as leaf nodes. Fig. 7 shows



(a) Neural net. (b) Multivariate polynomial regression.

(c) kNN Regression. (d) Decision tree.

Fig. 7. The error field produced by predictions using POD in combination with four machine learning techniques.

the pointwise absolute error of the fields corresponding to each of the predictions in Fig. 6. It can be seen that the neural network model has the largest regions of significant error of all the surrogate models. The kNN and decision tree models both do an excellent job of capturing the pressure field, and for the quadratic regression model the error is mainly localized towards the trailing edge of the airfoil upper surface.

To further quantify the performance of the methodology, we calculate the mean absolute error (MAE) over the entire field for each of the seven lift coefficient values. For each Mach number in turn, we repeat the process of holding out samples for that Mach number, training the surrogate models, predicting the pressure fields, and computing the MAE over the held-out test set. Fig. 8 plots the results. For each Mach number, the plot shows the minimum, mean and maximum of the seven MAEs for that Mach number. Again it can be seen that the neural network models have the worst performance (note the different scale on the neural net plot), despite having the largest training time and largest prediction time (see Fig. 9). For these times, the models are implemented in Python 2.7 and tested on a 2.3 GHz Intel Core i5.

Decision trees, quadratic polynomial regression and kNN regression have clearly outperformed the neural network in this example. The relationship between the inputs (Mach number and lift coefficient) and the output quantity of interest (pressure) is nonlinear and in theory should be better captured by a neural network model than by the simpler regression models. However, in this example the limited amount of training data is limiting the performance of the neural net. This suggests that while neural nets have become the machine learning models of choice in many applications with massive amounts of data (e.g., retail, finance), in an engineering setting the training data is often expensive to obtain and is sparse—as in this example, the training data are generated by running an expensive simulation—and other modeling strategies may be more appropriate. In this example, the strong performance of the kNN and decision tree models is notable and relates to the power of localization. As noted above, the relationship between inputs and outputs in this aerodynamic problem is nonlinear, but it is well known that locally linearized models can provide accurate representations—in fact, locally linearized models are already widely used in many CFD physics-based engineering models. The kNN and decision tree modeling approaches further exploit the combined power of localization and linearization by also learning the neighborhood of locality during the model training phase.

5. Structural example

The second case study considers the prediction of the structural response of a composite panel, using data generated from a large-scale finite element model.

5.1. Problem setup: damaged composite panel

We model the structural state of a composite panel undergoing local degradation of stiffness properties associated with the presence of a damage condition. The structural component under test is a composite panel (18×18 square inches) made of four layers of

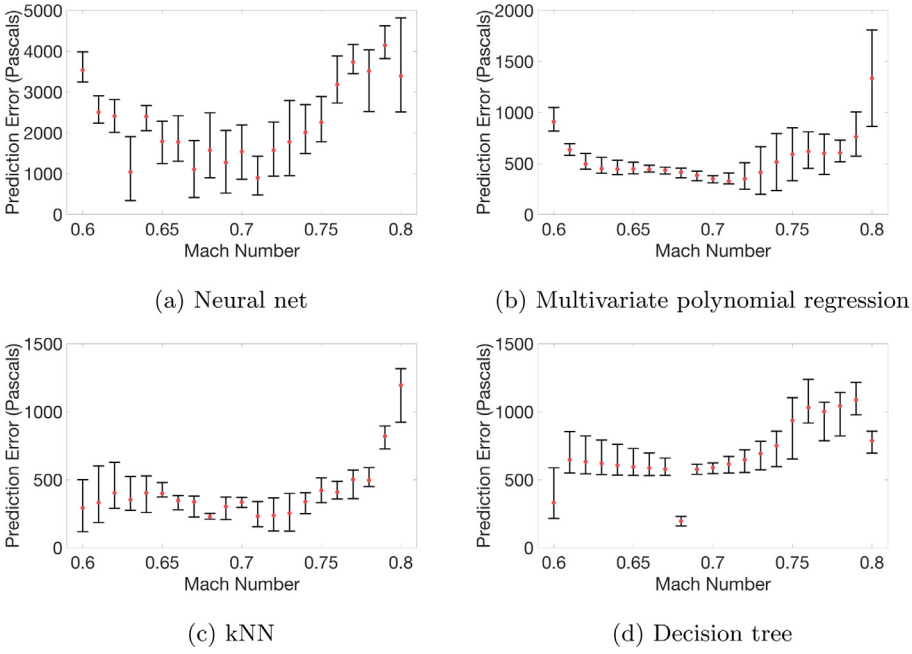


Fig. 8. The mean absolute error (MAE) over all lift coefficients when making predictions of a pressure field for a Mach number that has been held out during training. Each whisker shows the minimum, mean and maximum MAE.

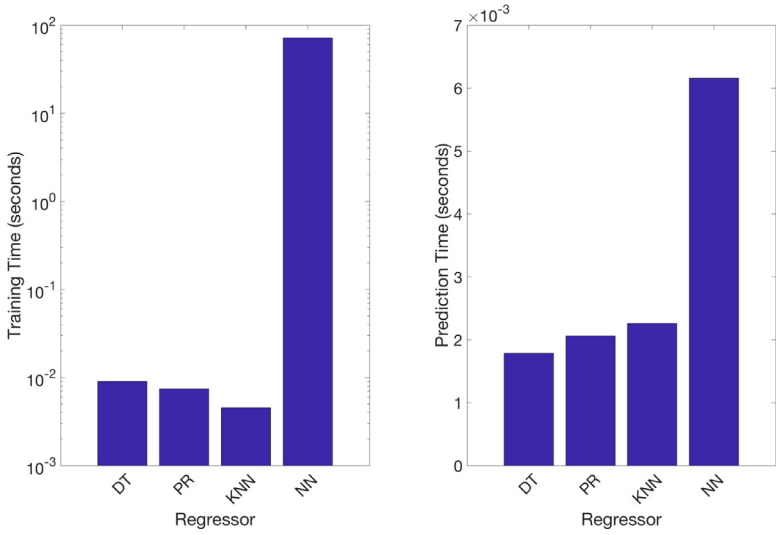


Fig. 9. Time comparison for training and prediction using four different regression models.

plain-weave carbon-fiber plies with symmetric stacking sequence $[\pm 45°, 0/90°, 0/90°, \pm 45°]$. The panel is clamped along the four edges with bolts. Two additional plain-weave plies ($[\pm 45°, 0/90°]$) are 2 in wide and reinforce the borders where bolt holes are located. We consider compression static loading applied as imposed displacement (-0.01 in), uniformly distributed along the top edge (see Fig. 10).

The structural state is computed numerically for different damage conditions. We adopt a finite element model of the panel and discretize the domain with $n_x = 3921$ two-dimensional laminate plate elements characterized with the carbon-fiber layup properties. The presence of the damage is modeled by weakening the local values of the stiffness for the affected elements and plies. In particular, we consider rectangular damage regions spanning the central portion of the panel and involving the first 1, 2 or 3 plies. The $m = 5$ input parameters describe the damage location (y and z coordinates of damage centroid) and the damage extent (in plane extension Δy and Δz, and number of affected plies ϱ), as indicated in Fig. 10. Thus, the input parameter vector is

$$\mathbf{p} = [p_1, p_2, p_3, p_4, p_5] = [y, z, \Delta y, \Delta z, \varrho], \tag{9}$$

with ranges shown in Table 1. The output quantity of interest is the strain field over the panel. We focus here on predicting a single component of strain—the normal component of strain measured along the first orthotropic axis of ply 4—giving the output quantity

Table 1
Composite panel input parameter space: boundaries and units.

Parameter component	Range	Units
y – Damage centroid coordinate along y	[2,8]	in.
z – Damage centroid coordinate along z	[2,8]	in.
Δy – Damage extension along y	[4,14]	in.
Δz – Damage extension along z	[4,14]	in.
ϱ – number of affected plies	[1,3]	plies

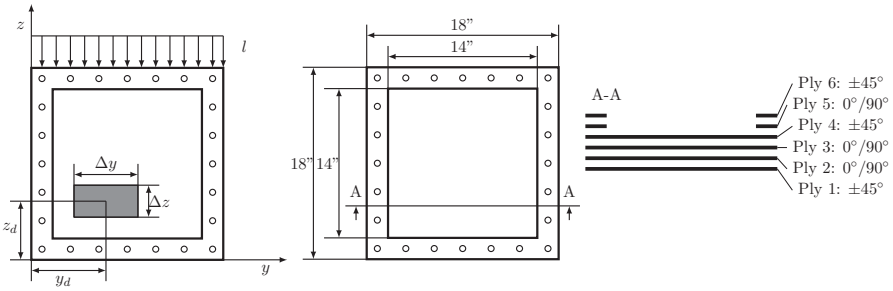


Fig. 10. Left: Composite panel parameters (damage location and damage size) and loading definition. Right: Panel layout and layers sequence.

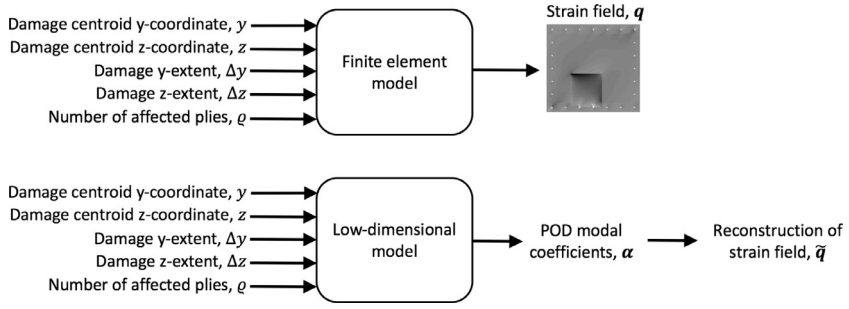
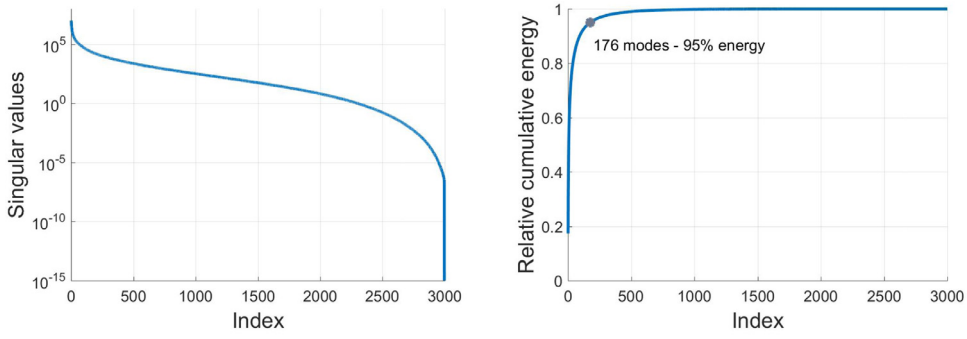


Fig. 11. Inputs and output quantity of interest for the structural example. High-fidelity finite element solver (top) and low-dimensional models (bottom).



(a) Singular values. (b) Cumulative energy.

Fig. 12. POD singular values and relative cumulative energy for the $n_{\text{train}} = 3000$ snapshots of normal component of strain along first main orthotropic axis of ply 4.

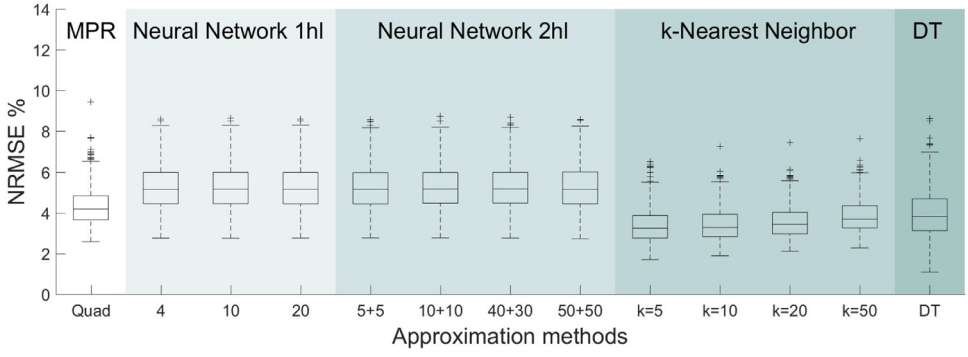
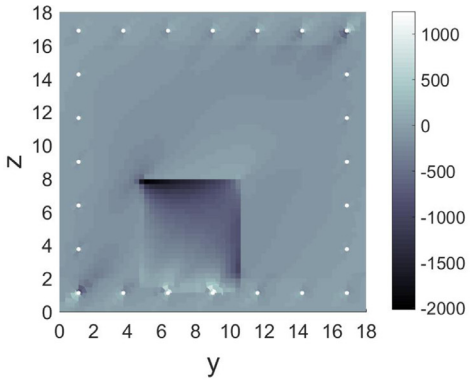


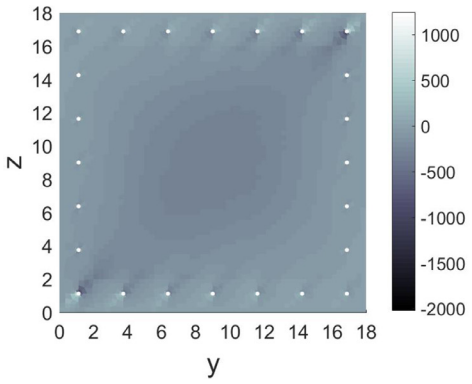
Fig. 13. Distributions of Normalized Root Mean Square Error (NRMSE) computed for the $n_{\text{test}} = 500$ cases of the test set for different surrogate models: multivariate polynomial regression (MPR) with a quadratic expansion; 1hl neural net with 4, 10 and 20 hidden neurons; 2hl neural net with 10 (5 + 5), 20 (10 + 10), 70 (40 + 30), and 100 (50 + 50) hidden neurons; kNN schemes with different neighborhood cardinality k; a decision tree.

of interest vector $\mathbf{q}(\mathbf{p}) \in \mathbb{R}^{n_x}$. Fig. 11 summarizes the input and output configuration associated with the high-fidelity finite element solver and the low-dimensional model.

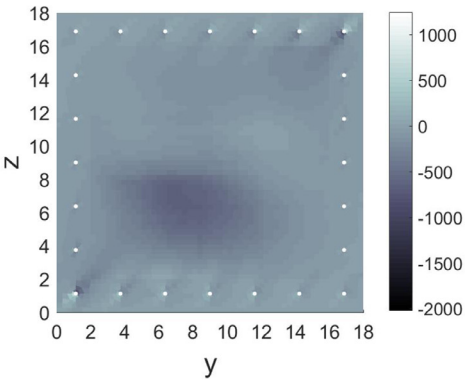
The finite element model is used to compute sets of snapshot data for training and testing. The training set consists of $n_{\text{train}} = 3000$ damage cases determined with a Latin hypercube exploration of the parameter space; this dataset provides the snapshot information to compute the POD modes and learn approximate models for the POD coefficients. The test set collects $n_{\text{test}} = 500$ damage cases determined with a second Latin hypercube exploration of the parameter space; this dataset is used to evaluate the accuracy



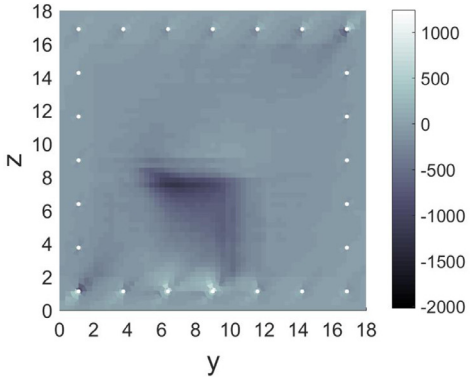
(a) True strain field.



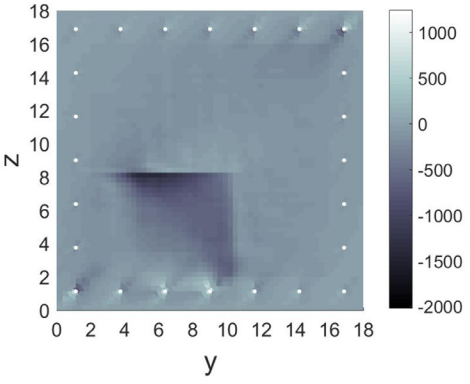
(b) Neural net: 1hl with 4 nodes .



(c) Multivariate polynomial regression.



(d) kNN regression with $k = 5$.



(e) Decision tree.

Fig. 14. True strain field (a) and predictions using POD in combination with four machine learning techniques (b)–(e), for a 6.074×6.938 square inch damage centered at (7.69,4.51) and affecting 3 plies ($\mathbf{p} = [7.69, 4.51, 6.074, 6.938, 3]$).

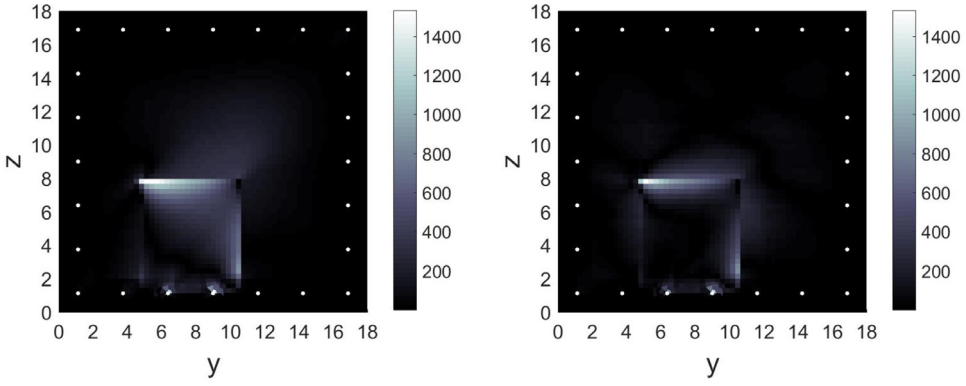
and efficiency of the approximation techniques discussed in this paper.

5.2. Structural results

Fig. 12 plots the POD singular values and relative cumulative energy associated with the POD modes computed from the training set. For this case, $r = 176$ POD modes recover 95% of the cumulative energy. We compute the POD coefficients $\boldsymbol{\alpha}(\mathbf{p})$ for each snapshot in the training set. The POD coefficients of the snapshots in the training set are then used to learn a map from inputs to POD coefficients for each of the $r = 176$ retained POD modes. These models are then used to predict $\boldsymbol{\alpha}(\mathbf{p})$ for each of the $n_{\text{test}} = 500$ damage cases of the test set. Finally, the predicted POD coefficients and the POD basis vectors are combined to estimate the strain field in the form of POD expansions (5), where $\bar{\mathbf{q}}$ is set to be the mean over all training snapshots.

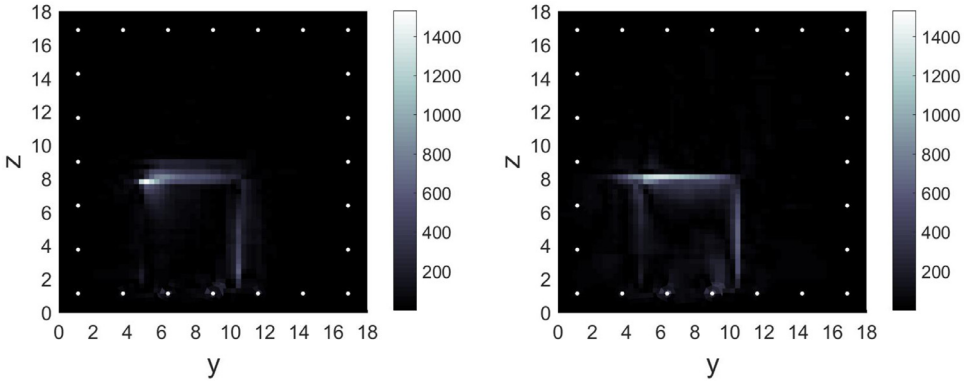
To assess the performance of the machine learning methods on this example, we compute the normalized root mean square error (NRMSE) between the actual and reconstructed strain fields over the test set. The NRMSE measures the generalization performance of the methods and is computed as:

$$\text{NRMSE} = \frac{\|\tilde{\mathbf{q}} - \mathbf{q}\|_2}{(q_{\max} - q_{\min})\sqrt{n_x}} \times 100\% \tag{10}$$



(a) Neural net: 1hl with 4 nodes.

(b) Multivariate polynomial regression.



(c) kNN Regression with $k = 5$.

(d) Decision tree.

Fig. 15. The error field produced by predictions using POD in combination with four machine learning techniques for $\mathbf{p} = [7.69, 4.51, 6.074, 6.938, 3]$.

where $q_{\max}$ and $q_{\min}$ denote the maximum and minimum values in the true strain field snapshot $\mathbf{q}$. Fig. 13 illustrates the NRMSE for the different methods with various modeling choices. Each box in the plot shows the quartiles of NRMSE; outliers are defined using 1.5 times the interquartile range and are marked with a + symbol. Moving from the left of the plot, MPR denotes the multivariate polynomial regression model, using a quadratic expansion of the form (7). The next batch of results are for the neural network models. The neural net implementation for this structural example uses a feedforward neural net with sigmoid activation functions. In particular, we consider two architectures: (i) a single hidden layer of neurons (denoted 1hl) with 4, 10 and 20 hidden neurons, and (ii) two hidden layers of neurons (denoted 2hl) with 10 (5 + 5), 20 (10 + 10), 70 (40 + 30), and 100 (50 + 50) hidden neurons. In all cases, the mean squared error is minimized during training using stochastic gradient descent with a learning rate of 0.1 and 1000 epochs. Moving to the right, the next batch of results in Fig. 13 are for the kNN model with $k = 5, 10, 20$ and 50 neighbors. The weights in these kNN models are inversely proportional to the distance to each neighbor. The rightmost result is a decision tree. No restrictions were set on the depth of the tree, resulting in 5993 nodes in total, 2997 of those as leaf nodes.

Among the methods considered for this study, kNN, decision tree and multivariate polynomial regression models show better median performance in NRMSE than the neural networks, with kNN producing the lowest error. In particular, the kNN model with $k = 5$ neighbors gives the lowest NRMSE mean (3.37%) and interquartile range (1.10%) over the entire test set. For the structural problem at hand, the strain quantity of interest field exhibits discontinuities along the edges of the damage region (as can be seen in Fig. 14a); hence, the relationship between inputs and POD coefficients is nonlinear and the neural network architectures might be expected to work better. However, the great flexibility of these networks comes with the risk of overfitting the training dataset and thus losing generalization capabilities. As we saw in the aerodynamic example, a lack of sufficient training data may be preventing the neural network models from showing better generalization performance.

Fig. 14 compares approximations of the strain field computed with the four different machine learning methods for one particular damage case of the test set $\mathbf{p} = [7.69, 4.51, 6.074, 6.938, 3]$. Fig. 15 illustrates the corresponding pointwise errors in the strain field predictions. The neural network prediction in Fig. 14b is not able to capture visual characteristics of the strain field. This is surprising behavior as the NRMSE for this method is less than 10%. This poor performance is likely due to our implementation and the relative lack of training data. A single neural net has been fit to predict a $r = 176$ dimensional output with a relatively small training set ($n_{train} = 3000$). An implementation involving 176 neural networks each fit to a single output may perform better in this example. As shown in Fig. 14c, the quadratic regression model smears out the strain field and does not capture the discontinuities that characterize the strain field along the borders of the damage region. This is due to the strong assumption of smoothness in the map $\boldsymbol{\alpha} : \mathcal{P} \to \mathcal{A}$ from inputs $\mathbf{p} \in \mathcal{P}$ to outputs $\boldsymbol{\alpha}(\mathbf{p}) \in \mathcal{A}$, an assumption that is poor for the nature of the physics in this particular problem. The kNN prediction in Fig. 14d has a similar result to the polynomial regression, but this method is able to highlight more distinctive visual discontinuities in the strain field. The accuracy of the kNN model is also apparent in the error plots in

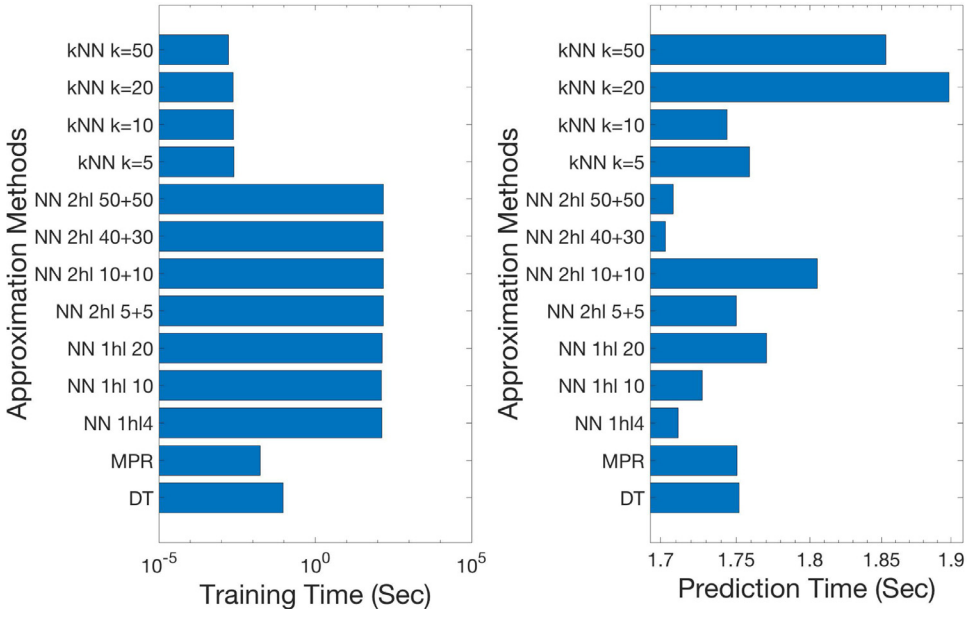


Fig. 16. Training time and online execution time for different surrogate models: decision tree; multivariate polynomial regression (MPR); 1hl neural networks with 4, 10 and 20 hidden neurons; 2hl neural networks with 10 (5 + 5), 20 (10 + 10), 70 (40 + 30), and 100 (50 + 50) hidden neurons; and kNN schemes with different neighborhood cardinality k.

Fig. 15c. The decision tree prediction in Fig. 14e is able to capture the sharp discontinuities well, but the location of the discontinuities is incorrect. While the location of the discontinuities is accurate in the z-direction, they span well past the range in the y-direction. This is possibly explained by the lack of restrictions set on the decision tree and the risk of overfitting.

Fig. 16 illustrates the training time and online execution time for the different approximation methods employed for this example. Training time is the time required to train the model using $n_{train} = 3000$ training samples. Prediction time is the time required to predict all $n_{test} = 500$ test cases. For these times, the models are implemented in Python 2.7 and tested on a 2.3 GHz Intel Core i5. Among the methods compared in Fig. 16, the kNN models are the fastest to learn. They are also efficient to evaluate when the number of neighbors is low. Training neural networks is computationally more expensive. Note that the prediction times in Fig. 16 are larger than those for the aerodynamic example in Fig. 9. This is because both the dimension of the inputs and of the POD expansion are much larger ($r = 176$ and $m = 5$ for the structural problem compared to $r = 10$ and $m = 2$ for the aerodynamic problem).

6. Conclusions

The case studies in this paper have demonstrated that the POD is an effective way to parametrize a high-dimensional output quantity of interest in order to define a low-dimensional map suitable for data-driven learning. Not only does the POD representation provide computational tractability by making the map low-dimensional, it also provides a way to embed physical constraints and it aids in interpretability of the resulting learned models. The case studies in this paper have also highlighted the important point that the availability of data in an engineering setting is typically much less than it is in other machine learning applications. This is because engineering data are often generated from expensive physics-based simulation codes or from sensors embedded on a physical system. In either case, it is reasonable to expect hundreds or thousands of training data points, but not millions as is the case in many non-engineering machine learning applications. The results in this paper highlight the need to account for this fact in choosing an appropriate machine learning strategy. While neural networks have considerable modeling flexibility, there are pitfalls when training data coverage of the input space is sparse. In such settings, a simpler kNN or polynomial regression model may be safer choices—even when the underlying physical phenomena exhibit nonlinear behavior, the power of the POD representation can transform the problem into one that is amenable to a simpler representation. Lastly we note that for the two examples studied in this paper, which are representative of many problems in engineering, the kNN models with a small number of neighbors outperformed other machine learning methods. This shows the power of using simple, explainable models but combining them with localization over the parameter space—an approach long used in engineering modeling, but formalized through the kNN training process.

Acknowledgments

This work was supported in part by AFOSR grant FA9550-16-1-0108 under the Dynamic Data Driven Application System Program, by the Air Force Center of Excellence on Multi-Fidelity Modeling of Rocket Combustor Dynamics, Award Number FA9550-17-1-0195, and by the MIT-SUTD International Design Center.

References

[1] Coveney PV, Dougherty ER, Highfield RR. Big data need big theory too. Philos Trans R Soc Lond A 2016;374(2080):1–11.
[2] Antoulas AC, Sorensen DC, Gugercin S. A survey of model reduction methods for large-scale systems. Contemp Math 2001;280:193–219.
[3] Rozza G, Huynh DBP, Patera AT. Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations. Arch Comput Methods Eng 2007;15(3):1–47.
[4] Chinesta F, Ladeveze P, Cueto E. A short review on model order reduction based on proper generalized decomposition. Arch Comput Methods Eng 2011;18(4):395.
[5] Quarteroni A, Rozza G, editors. Reduced order methods for modeling and computational reduction. Springer; 2014.
[6] Benner P, Gugercin S, Willcox K. A survey of projection-based model reduction methods for parametric dynamical systems. SIAM Rev 2015;57(4):483–531.
[7] Hesthaven J, Rozza G, Stamm B. Certified reduced basis methods for parametrized partial differential equations. Springer; 2016.
[8] Chinesta F, Huerta A, Rozza G, Willcox K. Model reduction methods. In: Encyclopedia of computational mechanics, second edition; 2017. p. 1–36.

[9] Lumley J. The structures of inhomogeneous turbulent flow. In: Atmospheric turbulence and radio wave propagation; 1967. p. 166–78.
[10] Holmes P, Lumley J, Berkooz G. Turbulence, coherent structures, dynamical systems and symmetry. Cambridge, UK: Cambridge University Press; 1996.
[11] Sirovich L. Turbulence and the dynamics of coherent structures. part 1: coherent structures. Q Appl Math 1987;45(3):561–71.
[12] Ly H, Tran H. Modeling and control of physical processes using proper orthogonal decomposition. J Math Comput Model 2001;33:223–36.
[13] Bui-Thanh T, Damodaran M, Willcox K. Aerodynamic data reconstruction and inverse design using proper orthogonal decomposition. AIAA J 2004;42(8):1505–16.
[14] Everson R, Sirovich L. The Karhunen-Loeve procedure for gappy data. J Opt Soc Am 1995;12:1657–64.
[15] Audouze C, De Vuyst F, Nair PB. Reduced-order modeling of parameterized PDEs using time-space-parameter principal component analysis. Int J Numer Methods Eng 2009;80(8):1025–57.
[16] Wirtz D, Karajan N, Haasdonk B. Surrogate modeling of multiscale models using kernel methods. Int J Numer Methods Eng 2014;101(1):1–28.
[17] Audouze C, De Vuyst F, Nair PB. Nonintrusive reduced-order modeling of parametrized time-dependent partial differential equations. Numer Methods Partial Differ Equ 2013;29(5):1587–628.
[18] Mainini L, Willcox K. Surrogate modeling approach to support real-time structural assessment and decision making. AIAA J 2015;53(6):1612–26.
[19] Ulu E, Zhang R, Kara LB. A data-driven investigation and estimation of optimal topologies under variable loading configurations. Comput Methods Biomech BiomedEng 2016;4(2):61–72.
[20] Hesthaven JS, Ubbiali S. Non-intrusive reduced order modeling of nonlinear problems using neural networks. J Comput Phys 2018;363:55–78.
[21] Chen W, Hesthaven JS, Junqiang B, Yang Z, Tihao Y. A greedy non-intrusive reduced order model for fluid dynamics. J Northwest Polytech Univ 2017.
[22] Ljung L. System identification. Prentice Hall; 1987.
[23] Viberg M. Subspace-based methods for the identification of linear time-invariant systems. Automatica 1995;31(12):1835–51.
[24] Kramer B, Gugercin S. Tangential interpolation-based Eigensystem realization algorithm for MIMO systems. Math Comput Model Dyn Syst 2016;22(4):282–306.
[25] Qin SJ. An overview of subspace identification. Comput Chem Eng 2006;30(10–12):1502–13.
[26] Reynders E. System identification methods for (operational) modal analysis: review and comparison. Arch Comput Methods Eng 2012;19(1):51–124.
[27] Abderrahim K, Mathlouthi H, Msahli F. New approaches to finite impulse response systems identification using higher-order statistics. IET Signal Proc 2010;4(5):488–501.
[28] Rabiner L, Crochiere R, Allen J. FIR system modeling and identification in the presence of noise and with band-limited inputs. IEEE Trans Acoust 1978;26(4):319–33.
[29] Mendel J. Tutorial on higher-order statistics (spectra) in signal processing and system theory: theoretical results and some applications. Proc IEEE 1991;79:278–305.
[30] Antoulas AC, Anderson BDQ. On the scalar rational interpolation problem. IMA J Math Control Inf 1986;3(2–3):61–88.
[31] Lefteriu S, Antoulas AC. A new approach to modeling multiport systems from frequency-domain data. Comput Aided Des Integr CircuitsSyst IEEE Trans 2010;29(1):14–27.
[32] Mayo A, Antoulas AC. A framework for the solution of the generalized realization problem. Linear Algebra Appl 2007;425(2–3):634–62.
[33] Beattie C, Gugercin S. Realization-independent $\mathcal{H}_2$-approximation. In: Proc. IEEE conf. decis. control, Maui, HI, USA; 2012. p. 4953–8.
[34] Schulze P, Unger B, Beattie C, Gugercin S. Data-driven structured realization. Linear Algebra Appl 2018;537:250–86.
[35] Ionita AC, Antoulas AC. Matrix pencils in time and frequency domain system identification. In: Developments in control theory towards glocal control. In: Control, Robotics & Sensors. Institution of Engineering and Technology; 2012. p. 79–88.
[36] Peherstorfer B, Gugercin S, Willcox K. Data-driven reduced model construction with time-domain Loewner models. SIAM J Scient Comput 2017;39(5):A2152–78.
[37] Drmač Z, Gugercin S, Beattie C. Quadrature-based vector fitting for discretized $\mathcal{H}_2$ approximation. SIAM J Scient Comput 2015;37(2):A625–52.
[38] Drmač Z, Gugercin S, Beattie C. Vector fitting for matrix-valued rational approximation. SIAM J Scient Comput 2015;37(5):A2346–79.
[39] Tu JH, Rowley CW, Luchtenburg DM, Brunton SL, Kutz JN. On dynamic mode decomposition: theory and applications. J Comput Dyn 2014;1(2):391–421.
[40] Proctor JL, Brunton SL, Kutz JN. Dynamic mode decomposition with control. SIAM J Appl Dyn Syst 2016;15(1):142–61.
[41] Proctor JL, Brunton SL, Brunton BW, Kutz JN. Exploiting sparsity and equation-free architectures in complex systems. Eur Phys J Spec Top 2014;223(13):2665–84.
[42] Peherstorfer B, Willcox K. Data-driven operator inference for nonintrusive projection-based model reduction. Comput Methods Appl Mech Eng 2016;306:196–215.
[43] Castelletti A, Galelli S, Restelli M, Soncini-Sessa R. Data-driven dynamic emulation modelling for the optimal management of environmental systems. Environ Model Softw 2012;34:30–43.
[44] Galelli S, Castelletti A, Goedbloed A. High-performance integrated control of water quality and quantity in urban water reservoirs. Water Resour Res 2015;51:9053-9072.
[45] Brunton SL, Proctor JL, Kutz JN. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proc Natl Acad Sci 2016;113(15):3932–7.
[46] Dean S, Mania H, Matni N, Recht B, Tu S. On the sample complexity of the linear quadratic regulator. ArXiv e-prints 2017. arXiv:1710.01688.
[47] Tu S, Recht B. Least-Squares temporal difference learning for the linear quadratic regulator. ArXiv e-prints 2017. arXiv:1712.08642.
[48] Balzano L, Nowak R, Recht B. Online identification and tracking of subspaces from highly incomplete information. In: 2010 48th annual Allerton conference on communication, control, and computing (Allerton); 2010. p. 704–11.
[49] Peherstorfer B, Willcox K. Online adaptive model reduction for nonlinear systems via low-rank updates. SIAM J Scient Comput 2015;37(4):A2123–50.
[50] Zimmermann R, Peherstorfer B, Willcox K. Geometric subspace updates with applications to online adaptive nonlinear model reduction. SIAM J Matrix Anal Appl 2018;39(1):234–61.
[51] Yano M, Penn JD, Patera AT. A model-data weak formulation for simultaneous estimation of state and model bias. CR Math 2013;351(23–24):937–41.
[52] Maday Y, Patera A, Penn JD, Yano M. PBDW State estimation: noisy observations; configuration-adaptive background spaces; physical interpretations. In: ESAIM: Proc, 50; 2015. p. 144–68.
[53] Parish EJ, Duraisamy K. A paradigm for data-driven predictive modeling using field inversion and machine learning. J Comput Phys 2016;305:758–74.
[54] Singh AP, Duraisamy K, Zhang ZJ. Augmentation of turbulence models using field inversion and machine learning. In: 55th AIAA aerospace sciences meeting. In: AIAA SciTech Forum. American Institute of Aeronautics and Astronautics; 2017. p. 1–18.
[55] Lam R, Allaire D, Willcox K. Multifidelity optimization using statistical surrogate modeling for non-hierarchical information sources. In: 56th AIAA/ASCE/AHS/ASC structures, structural dynamics, and materials conference; 2015. p. 0143.
[56] Poloczek M, Wang J, Frazier P. Multi-information source optimization. In: Advances in neural information processing systems; 2017. p. 4291–301.
[57] Ghoreishi SF, Allaire DL. A fusion-based multi-information source optimization approach using knowledge gradient policies. American Institute of Aeronautics and Astronautics; 2018. doi: 10.2514/6.2018-1159.
[58] Peherstorfer B, Willcox K, Gunzburger M. Survey of multifidelity methods in uncertainty propagation, inference, and optimization. SIAM Rev 2018. (to appear).
[59] Bishop C. Pattern recognition and machine learning. Springer; 2006.
[60] Hastie T, Tibshirani R, Friedman J. The elements of statistical learning. Springer; 2009.
[61] Murphy K. Machine learning. MIT Press; 2012.
[62] Haykin SS. Neural networks and learning machines. Pearson; 2008.
[63] Akçelik V, Bielak J, Biros G, Epanomeritakis I, Fernandez A, Ghattas O, et al. High resolution forward and inverse earthquake modeling on terascale computers. In: SC03: proceedings of the international conference for high performance computing, networking, storage, and analysis. ACM/IEEE; 2003. Gordon Bell Prize for Special Achievement.
[64] Rudi J, Malossi ACI, Isaac T, Stadler G, Gurnis M, Ineichen Y, et al. An extreme-scale implicit solver for complex PDEs: highly heterogeneous flow in Earth's mantle. In: SC15: proceedings of the international conference for high performance computing, networking, storage and analysis. ACM; 2015. Winner of Gordon Bell Prize. 5:1–5:12.
[65] Loéve M. Probability theory. New York: D. Van Nostrand Company Inc.; 1955.
[66] Kosambi D. Statistics in function space. J Indian Math Soc 1943;7:76–88.
[67] Hotelling H. Analysis of a complex of statistical variables with principal components. J Educ Psychol 1933;24:417–441,498–520.
[68] North G, Bell T, Cahalan R, Moeng F. Sampling errors in the estimation of empirical orthogonal functions. Mon Weather Rev 1982;110(7):699–706.
[69] Raghavan B, Hamdaoui M, Xiao M, Breitkopf P, Villon P. A bi-level meta-modeling approach for structural optimization using modified POD bases and diffuse approximation. Comput Struct 2013;127:19–28.
[70] Hall K, Thomas JP, Dowell EH. Proper orthogonal decomposition technique for transonic unsteady aerodynamic flows. AIAA J 2000;38(10):1853–62.
[71] Nielsen MA. Neural networks and deep learning. Determination Press USA; 2015.
[72] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature 1986;323(6088):533.
[73] Livni R, Shalev-Shwartz S, Shamir O. On the computational efficiency of training neural networks. In: Advances in neural information processing systems; 2014. p. 855–63.
[74] Bentley JL. Multidimensional binary search trees used for associative searching. Commun ACM 1975;18(9):509–17.
[75] Friedman JH, Bentley JL, Finkel RA. An algorithm for finding best matches in logarithmic expected time. ACM Trans Math Softw 1977;3(3):209–26.
[76] Breiman L, Friedman J, Stone CJ, Olshen RA. Classification and regression trees. CRC Press; 2017.
[77] Palacios F, Colonno MR, Aranake AC, Campos A, Copeland SR, Economon TD, et al. Stanford University Unstructured (SU2): an open-source integrated computational environment for multi-physics simulation and design. In: 51st AIAA aerospace sciences meeting, Grapevine, Texas; 2013. p. 1–60.