SIAM J. SCI. COMPUT.
Vol. 45, No. 3, pp. A1148–A1171

LEVEL SET LEARNING WITH PSEUDOREVERSIBLE NEURAL NETWORKS FOR NONLINEAR DIMENSION REDUCTION IN FUNCTION APPROXIMATION*

YUANKAI TENG†, ZHU WANG†, LILI JU‡, ANTHONY GRUBER§, AND GUANNAN ZHANG¶

Abstract. Due to the curse of dimensionality and the limitation on training data, approximating high-dimensional functions is a very challenging task even for powerful deep neural networks. Inspired by the Nonlinear Level-set Learning (NLL) method, which uses the reversible residual network (RevNet), in this paper, we propose a new method of Dimension Reduction via Learning Level Sets (DRiLLS) for function approximation. Our method contains two major components: one is the pseudoreversible neural network (PRNN) module, which effectively transforms high-dimensional input variables to low-dimensional active variables, and the other is the synthesized regression module for approximating function values based on the transformed data in the low-dimensional space. The PRNN not only relaxes the invertibility constraint of the nonlinear transformation present in the NLL method due to the use of RevNet but also adaptively weights the influence of each sample and controls the sensitivity of the function to the learned active variables. The synthesized regression uses Euclidean distance in the input space to select neighboring samples whose projections on the space of active variables are used to perform local least-squares polynomial fitting. This helps to resolve numerical oscillation issues present in traditional local and global regressions. Extensive experimental results demonstrate that our DRiLLS method outperforms both the NLL and active subspace methods, especially when the target function possesses critical points in the interior of its input domain.

Key words. function approximation, dimension reduction, pseudoreversible neural network, level set learning, synthesized regression, sparse data

MSC codes. 65D15, 65D40, 68U07

DOI. 10.1137/21M1459198

1. Introduction.

High-dimensional function approximation plays an important role in building predictive models for a variety of scientific and engineering problems. It is typical for scientists to build an accurate and fast-to-evaluate surrogate model to replace a computationally expensive physical model, in order to reduce the overall cost of a large set of model executions. However, when the dimension of the target

* Submitted to the journal's Methods and Algorithms for Scientific Computing section November 15, 2021; accepted for publication (in revised form) December 15, 2022; published electronically June 5, 2023.

https://doi.org/10.1137/21M1459198

Funding: This work is partially supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Office of Biological and Environmental Research, through the Applied Mathematics, Earth, and Environmental System Modeling and the Scientific Discovery through Advanced Computing programs under university grant DE-SC0022254 (third author) and contract ERKJ387 (fifth author) at Oak Ridge National Laboratory and by the U.S. National Science Foundation under grants DMS-2012469 and DMS-2038080 (second author).

† Department of Mathematics, University of South Carolina, Columbia, SC 29208 USA (yteng@email.sc.edu, wangzhu@math.sc.edu).

‡ Corresponding author. Department of Mathematics, University of South Carolina, Columbia, SC 29208 USA (ju@math.sc.edu).

§ Department of Scientific Computing, Florida State University, Tallahassee, FL 32306 USA (agruber@fsu.edu).

¶ Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831 USA (zhangg@ornl.gov).

function's input space becomes large, the data fitting becomes a computationally challenging task. Due to the curse of dimensionality, an accurate function approximation would require the number of samples in the training dataset to increase exponentially with respect to the dimension of input variables. On the other hand, given the complexity of the underlying physical model, the amount of observational data is often very limited. This causes classical approximation methods such as sparse polynomial approximation (e.g., sparse grids) to fail on high-dimensional problems outside of some special situations. One way to alleviate the challenge is to reduce the input dimension of the target function by finding intrinsically low-dimensional structures.

The existing methods for dimension reduction in function approximation can be divided into two main categories. The first one is to exploit the dependence between input variables to build low-dimensional manifolds in the input space. For example, principal component analysis [1] is widely used, due to its simplicity, to compress the input space to a low-dimension manifold. Isometric feature mapping [29] is an effective method to compute a globally nonlinear low-dimensional embedding of high-dimensional data. Its modification known as locally linear embedding [10, 27] provides solutions to more general cases. However, in practice, there are often no dependences between input variables to exploit, so that the dimension of the input space cannot be effectively reduced by methods reliant on this assumption. This represents a challenging research question for function approximation, namely, *how to effectively reduce the dimension of a function with independent input variables*.

To answer this question, the second category of dimension reduction methods aims at reducing the input dimension by exploiting the relationship between the input and the output, i.e., learning the geometry of a function's level sets. This includes methods such as sufficient dimension reduction [2, 7, 19, 25] and its extensions [8, 9, 18, 20, 21, 22, 24, 31, 32], the active subspace (AS) method [5, 6], and neural network (NN)–based methods [16, 30, 33]. This type of method first identifies a linear/nonlinear transformation that maps the input variables to a handful of active variables (or coordinates), then projects the observational data onto the subspace spanned by the active variables, and finally performs the data fitting in the low-dimensional subspace to determine the function approximation.

The AS method [5, 6] is a popular dimension reduction approach that seeks a set of directions in the input space, named active components, affecting the function value most significantly on average. Given the values of the function $f(\boldsymbol{x})$ and its gradient $\nabla f(\boldsymbol{x})$ at a set of sample points, this method first evaluates the uncentered covariance matrix of the gradient $\boldsymbol{C} = \mathbb{E}[\nabla f(\nabla f)^\top]$, where the symbol $\mathbb{E}[\cdot]$ denotes the expected value. The eigenvectors associated with the leading eigenvalues of $\boldsymbol{C}$, denoted by $\boldsymbol{W}_A$, are selected to define active components $\boldsymbol{z}_A = \boldsymbol{W}_A^\top \boldsymbol{x}$, which is a linear transformation of the input $\boldsymbol{x}$. The subspace spanned by the set of active components $\boldsymbol{z}_A$ describes a low-dimensional linear subspace embedded in the original input space that captures most of the variation in model output. A regression surface $\tilde{f}(\boldsymbol{z}_A)$ is then constructed based on the data projected onto the AS $\{\boldsymbol{z}_A, f(\boldsymbol{x})\}$, i.e., $f(\boldsymbol{x}) \approx \tilde{f}(\boldsymbol{W}_A^\top \boldsymbol{x})$.

Recently, NN-based approaches [16, 30, 33] were developed to extract low-dimensional structures from high-dimensional functions. The Nonlinear Level-set Learning (NLL) method [33] finds a bijective nonlinear transformation that maps an input point $\boldsymbol{x}$ to a new point $\boldsymbol{z}$, which is of the same dimension as $\boldsymbol{x}$, more specifically, $\boldsymbol{z} = \boldsymbol{r}(\boldsymbol{x})$ with $\boldsymbol{r}$ modeled by a reversible residual NN (RevNet) [4, 11]. In

this approach, the transformed variables $\boldsymbol{z}$ are expected to be split into two sets, a set of active variables (coordinates) $\boldsymbol{z}_A = \{z_k\}_{k\in A}$ and a set of inactive variables $\boldsymbol{z}_I = \{z_k\}_{k\in I}$, so that the function value $f(\boldsymbol{r}^{-1}(\boldsymbol{z}))$ is insensitive to perturbations in $\boldsymbol{z}_I$. That is, if $z_k \in \boldsymbol{z}_I$, a small perturbation in z_k within the neighborhood of $\boldsymbol{z}$ would lead to almost no change in function value. Based on this fact, the NLL method employs a loss function that encourages the gradient vector field $\nabla f(\boldsymbol{x})$ to be orthogonal to the derivative of $\boldsymbol{r}^{-1}(\boldsymbol{z})$ with respect to each inactive variable $z_k \in \boldsymbol{z}_I$. Therefore, after a successful training, the NLL method can provide a manifold that captures the low-dimensional structure of the function's level sets. Similar to the AS method, once $\boldsymbol{z}_A$ is determined, a regression surface can be built using the data projected onto the subspace of active variables, $\{\boldsymbol{z}_A, f(\boldsymbol{x})\}$. It has been shown in [33] that NLL outperforms AS when the level sets of the function have nontrivial curvature. An improved algorithm for the NLL method was studied in [13]. However, there still are even simple cases in which the NLL fails to effectively extract low-dimensional manifolds, as shown later in this paper.

In this paper, we introduce a new *Dimension Reduction via Learning Level Sets* (DRiLLS) method for function approximation that improves on existing level set learning methods in the following aspects: (1) To enhance the model's capability, we propose a novel pseudoreversible neural network (PRNN) to model the nonlinear transformation for extracting active variables. (2) The learning process is driven by geometric features of the unknown function, which is reflected in a loss function consisting of three terms: the pseudoreversibility loss, the active direction fitting loss, and the bounded derivative loss. (3) A novel synthesized regression on the manifold spanned by the learned active variables is also proposed, which helps to resolve numerical oscillation issues and provides accuracy benefits over traditional local and global regressions. Extensive numerical experiments demonstrate that the proposed DRiLLS method leads to significant improvements on high-dimensional function approximations with limited or sparse data.

The rest of paper is organized as follows. In section 2, the setting of the function approximation problem is introduced, and the DRiLLS method is proposed and discussed. More specifically, the PRNN module is described in section 2.1 and the synthesized regression module in section 2.2. We then numerically investigate the performance of our DRiLLS method in section 3, including ablation studies in section 3.1, high-dimensional function approximations with limited/sparse data in section 3.2, and a PDE-related application in section 3.3. Finally, some concluding remarks are drawn in section 4.

2. The proposed DRiLLS method. We consider a scalar target function, which is continuously differentiable on a bounded Lipschitz domain Ω in $\mathbb{R}^d$:

$$y = f(\boldsymbol{x}), \quad \boldsymbol{x} = (x_1, x_2, \ldots, x_d) \in \Omega. \tag{2.1}$$

The input variables $x_1, x_2, \ldots, x_d$ are assumed to be independent from each other, which implies that the input space itself does not possess a low-dimensional structure. The goal is to find an approximation $\hat{f}(\boldsymbol{x})$ of the target function given the information of f and ∇f on a set of training samples in Ω. We denote the training dataset by

$$\Xi := \left\{ \left(\boldsymbol{x}^{(n)}, f(\boldsymbol{x}^{(n)}), \nabla f(\boldsymbol{x}^{(n)}) \right) : n = 1, \ldots, N \right\},$$

which contains the input, the output, and the gradient information at the samples. When the number of dimensions d is large, taking a handful of random selections

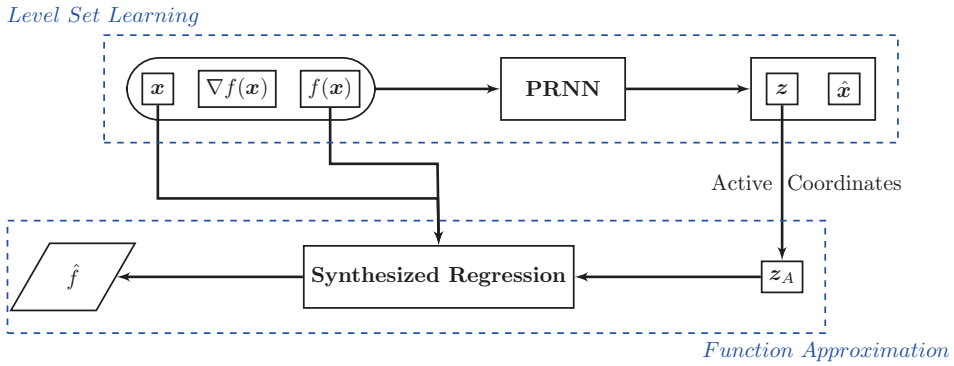


FIG. 1. *The overall structure of the proposed DRiLLS method, which consists of two major components: the PRNN module and the synthesized regression module.*

in each coordinate would result in a huge amount of data, which is infeasible in many application scenarios. Therefore, the sample dataset is usually sparse for high-dimensional problems.

The NLL method has achieved successes in high-dimensional function approximation on sparse data for real-world applications, such as composite material design problems [33]; however, it has difficulties in learning level sets of certain functions. In particular, NLL struggles on functions with critical points contained in the interior of the domain Ω, such as the functions $x_1^2+x_2^2$ or $x_1^2-x_2^2$ on $\Omega=[-1,1]^2$ to be discussed later in section 3.1. One reason for such a drawback is that the RevNet employed by NLL enforces invertibility as a hard constraint, which limits the capability of the RevNet in learning the structure of functions whose level sets are not homeomorphic to hyperplanes in the input space. Another reason is that the rate of change in the target function with respect to the inactivate variables is always zero at any interior critical points, as the gradient of the function vanishes there. Hence, the training process tends to ignore samples lying in a small neighborhood of the critical points since they do not contribute much to the training loss.

To overcome these issues and improve the performance of level set learning–based function approximation, the proposed DRiLLS method consists of two major components: (1) the PRNN module, which identifies active variables and reduces the dimension of input space, and (2) the synthesized regression module, which selects neighboring sample points according to their Euclidean distances in the original input space and performs a local least-squares fitting based on the learned active variables to approximate the target function. A schematic diagram of the proposed method is shown in Figure 1.

2.1. The PRNN. To construct the PRNN, we first define a nonlinear mapping from the input $\boldsymbol{x}$ to a new point $\boldsymbol{z}$ of the same dimension. In contrast to the RevNet used by the NLL method, the invertibility of this transformation is relaxed by defining another mapping from $\boldsymbol{z}$ to $\hat{\boldsymbol{x}}$ and encouraging $\hat{\boldsymbol{x}}$ to be close to $\boldsymbol{x}$ in distance. Thus, the reversibility is imposed as a soft constraint on the PRNN model. Specifically, the two nonlinear transformations are denoted by

$$\boldsymbol{z}=\boldsymbol{g}(\boldsymbol{x};\Theta_g) \quad \text{and} \quad \hat{\boldsymbol{x}}=\boldsymbol{h}(\boldsymbol{z};\Theta_h), \tag{2.2}$$

respectively, where $\boldsymbol{g}, \boldsymbol{h}:\mathbb{R}^d\to\mathbb{R}^d$ with Θ_g and Θ_h being their learnable parameters. Since $\boldsymbol{g}$ is not exactly invertible by definition, $\boldsymbol{h}$ can be viewed as a pseudoinverse

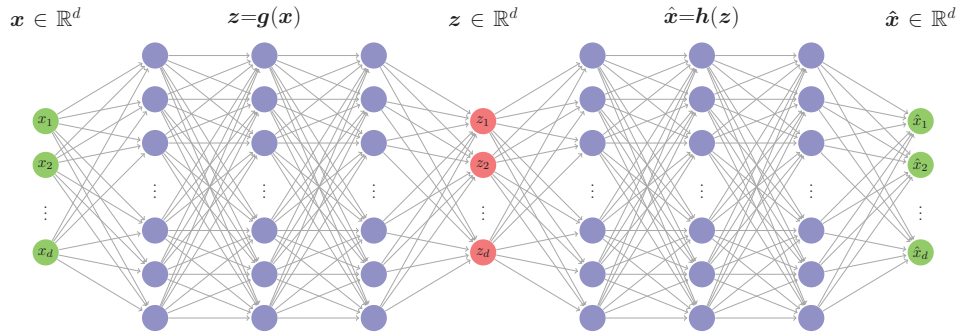


FIG. 2. *The PRNN consists of two FCNNs representing* $\boldsymbol{g}$ *and* $\boldsymbol{h}$*, respectively, that possess the same number of hidden layers (three layers for illustration) and the same number of neurons at each layer.*

function to $\boldsymbol{g}$. Both $\boldsymbol{g}$ and $\boldsymbol{h}$ are represented by a fully connected neural network (FCNN), as displayed in Figure 2. The PRNN network structure is reminiscent of an autoencoder [12], but the dimension of latent space (i.e., the dimension of $\boldsymbol{z}$) remains the same as the dimension of $\boldsymbol{x}$. While there are no theoretical restrictions on the structure of $\boldsymbol{g}$ and $\boldsymbol{h}$, the experiments in section 3 use the same FCNN architecture for both mappings.

2.1.1. The loss function. The learnable parameters Θ_g and Θ_h are updated synchronously during the training process by minimizing the following total loss function:

$$\mathcal{L} = \mathcal{L}_1 + \lambda_1 \mathcal{L}_2 + \lambda_2 \mathcal{L}_3. \tag{2.3}$$

Here, $\mathcal{L}_1$ is the *pseudoreversibility loss*, which measures the difference between $\boldsymbol{x}$ and the PRNN output $\hat{\boldsymbol{x}}$; $\mathcal{L}_2$ is the *active direction fitting loss*, which enforces the tangency between $\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{z}_I}$ and the level sets of f; and $\mathcal{L}_3$ is the *bounded derivative loss*, which regularizes the sensitivity of f with respect to the active variables $\boldsymbol{z}_A$. The weights λ_1 and λ_2 are hyperparameters for balancing the three loss terms. Below, each term of $\mathcal{L}$ is discussed in detail.

The pseudoreversibility loss. In order to train $\hat{\boldsymbol{x}} = \boldsymbol{h}(\boldsymbol{z})$ to be a pseudoinverse of $\boldsymbol{z} = \boldsymbol{g}(\boldsymbol{x})$, the pseudoreversibility condition is simply enforced in the L^2 sense,

$$\mathcal{L}_1 = \frac{1}{N} \sum_{n=1}^{N} ||\boldsymbol{x}^{(n)} - \boldsymbol{h} \circ \boldsymbol{g}(\boldsymbol{x}^{(n)})||_2^2, \tag{2.4}$$

which is the same as the standard loss used to train autoencoders.

The active direction fitting loss. This loss is defined based on the fact that if the kth output z_k of $\boldsymbol{g}(\boldsymbol{x})$ is inactive, a small perturbation of z_k in a neighborhood of $\boldsymbol{z}$ would change the target function f along a direction tangent to its level sets. Specifically, we define the Jacobian matrix of the nonlinear transformation $\boldsymbol{h}$ as

$$\boldsymbol{J}_{\boldsymbol{h}}(\boldsymbol{z}) = [\boldsymbol{J}_1(\boldsymbol{z}), \boldsymbol{J}_2(\boldsymbol{z}), \ldots, \boldsymbol{J}_d(\boldsymbol{z})]$$

with

$$\boldsymbol{J}_i(\boldsymbol{z}) := \left[\frac{\partial \hat{x}_1}{\partial z_i}(\boldsymbol{z}), \ldots, \frac{\partial \hat{x}_d}{\partial z_i}(\boldsymbol{z}) \right]^\top.$$

In the ideal case, if z_k is completely inactive, then the gradient vector field $\nabla f(\boldsymbol{x})$ is orthogonal to $\boldsymbol{J}_k(\boldsymbol{z})$, that is, $\langle \boldsymbol{J}_k(\boldsymbol{z}), \nabla f(\boldsymbol{x})\rangle = 0$ with $\langle\cdot,\cdot\rangle$ denoting the inner product. Thus, the *active direction fitting loss* is defined to encourage the orthogonality, i.e.,

$$\mathcal{L}_2 = \frac{1}{N}\sum_{n=1}^{N}\gamma_n \sum_{i=1}^{d}\left[\omega_i \left\langle \boldsymbol{J_i}(\boldsymbol{z^{(n)}}), \nabla f(\boldsymbol{x}^{(n)})\right\rangle\right]^2, \tag{2.5}$$

where the scaling factors

$$\gamma_n = 1 + \alpha e^{-\|\nabla f(\boldsymbol{x}^{(n)})\|}, \quad n = 1, 2, \ldots, N,$$

contain the hyperparameter $\alpha \geq 0$ and $\omega_1, \omega_2, \ldots, \omega_d \in \{0, 1\}$ are weight hyperparameters determining how strictly the orthogonality condition is enforced for each of the d variables. A typical choice is

$$\boldsymbol{\omega} = (\overbrace{0, \ldots, 0}^{k^*}, \overbrace{1, \ldots, 1}^{d-k^*}), \tag{2.6}$$

where k^* denotes the dimension of the active variables/coordinates. An ideal case would be $k^* = 1$, which implies that there exists only one active variable $\boldsymbol{z}_A = \{z_1\}$ and the intrinsic dimension of $f \circ \boldsymbol{h}(\boldsymbol{z})$ is exactly one when $\mathcal{L}_2 = 0$. The scaling factor γ_i distinguishes $\mathcal{L}_2$ from the one used in [33], and its value changes according to the magnitude of the gradient: It approaches $1+\alpha$ if $\|\nabla f(\boldsymbol{x}^{(n)})\|$ gets close to 0 and stays close to 1 otherwise. Therefore, it serves as a rescaling factor designed to overcome the situation where the contributions of samples near interior critical points are ignored by the optimization due to their small gradients.

The bounded derivative loss. Existing methods such as NLL do not place any restrictions on the active variables because the used RevNet imposes sufficient regularization on those variables. On the other hand, using PRNNs without regularization in $\boldsymbol{z}_A$ may cause the network to learn an AS which changes too fast, producing undesirable oscillations in the target function. To address this issue, we introduce a regularization term into the loss as

$$\mathcal{L}_3 = \frac{1}{N}\sum_{n=1}^{N} \text{sigmoid}\left(\frac{1}{\sigma}\left(\left\|\frac{\partial f \circ \boldsymbol{h}}{\partial \boldsymbol{z}_A}(\boldsymbol{z}^{(n)})\right\| - 1\right)\right), \tag{2.7}$$

where σ is a positive rescaling hyperparameter. The purpose is to regularize the magnitude of $\frac{\partial f\circ \boldsymbol{h}}{\partial \boldsymbol{z}_A}(\boldsymbol{z}^{(n)})$ to be not much greater than one. In the practical implementation, we further approximate $\frac{\partial f\circ \boldsymbol{h}(\boldsymbol{z}^{(n)})}{\partial \boldsymbol{z}_A}$ with $(\nabla f(\boldsymbol{x}^{(n)}))^{\intercal}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}_A}(\boldsymbol{z}^{(n)})$ by considering the pseudoreversibility of the PRNN.

2.2. The synthesized regression. The active variables (coordinates) $\boldsymbol{z}_A$ are naturally identified based on the presetting values of the weights $\boldsymbol{\omega}$. Once the PRNN training is completed, the sample points $\{\boldsymbol{x^{(n)}}, \boldsymbol{n = 1, \ldots, N}\}$ can be nonlinearly projected through PRNN to a much lower dimensional space spanned by $\boldsymbol{z}_A$. Ideally, approximating the high-dimensional function $f(\boldsymbol{x})$ often can be achieved by approximating the low-dimensional function

$$\widetilde{f}(\boldsymbol{z}_A) := f \circ \boldsymbol{g}^{-1}(\boldsymbol{z}) \text{ with } \boldsymbol{z}_A \in \mathbb{R}^{k*}, \tag{2.8}$$

where $k^* \ll d$. Many existing methods could be used, including classic polynomial interpolations, least-squares polynomial fitting [14], and regression by deep NNs. However, because the control on $\boldsymbol{g}$ is quite loose through the PRNN, $\widetilde{f}$ could be very

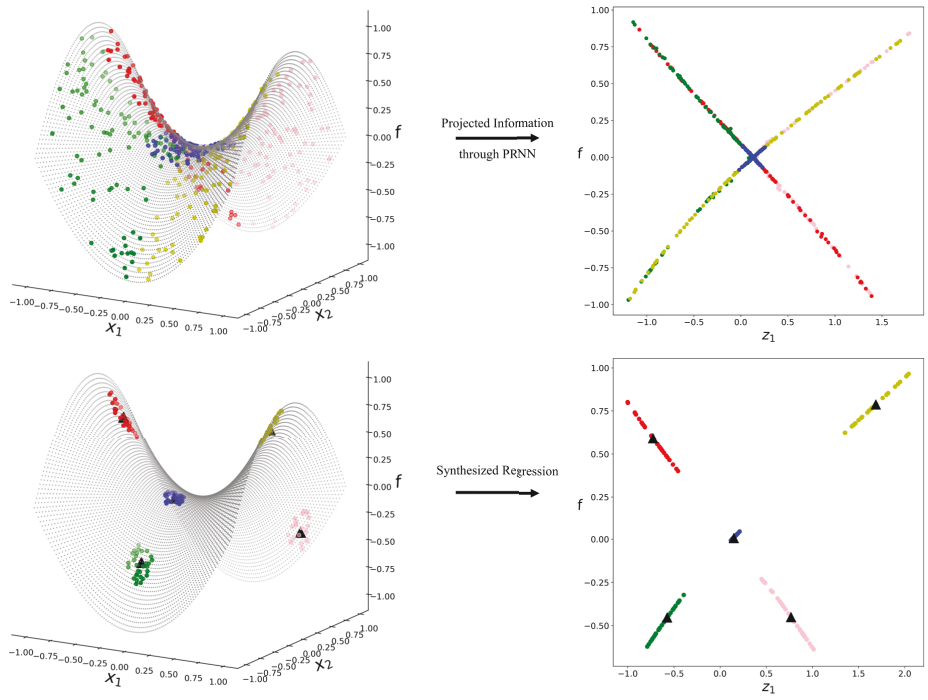


FIG. 3. *Top row: An example illustration of the sample points and their active variables learned by the PRNN, where we take* $f(x_1, x_2) = x_1^2 - x_2^2$ *and set* z_1 *as the active variable and* z_2 *as the inactive variable. As* f *is very oscillatory with respect to* z_1 *and even may not be a function of* z_1*, this case is not suitable with general local or global regression approaches based solely on the projected information in the space of active variables,* $\{(z_1^{(n)}, f(\boldsymbol{x}^{(n)})\}_{n=1}^N$*. Bottom row: The proposed synthesized regression first selects the local neighbor sample points for each of the five new inputs (i.e., the five* ▲*-shaped points whose function values are to be predicted) from the original input space, then performs respective least-squares polynomial data fitting.*

oscillatory with respect to $\boldsymbol{z}_A$ or even make $\widetilde{f}$ fail to form a function. For example, there could exist two sample points $\boldsymbol{x}$ and $\boldsymbol{y}$, which are separated in the input space with different values $f(\boldsymbol{x})$ and $f(\boldsymbol{y})$ but mapped close together in the transformed space, i.e., $(\boldsymbol{g}(\boldsymbol{x}))_A \approx (\boldsymbol{g}(\boldsymbol{y}))_A$. This is often the case for functions with interior critical points. The top row of Figure 3 presents an example illustration of such case, where we take $f(x_1, x_2) = x_1^2 - x_2^2$ and set z_1 as the active variable and z_2 as the inactive variable. Consequently, general global or local regression approaches based solely on the projected information in the space of active variables are not able to effectively handle this case due to large numerical oscillations.

We develop a synthesized regression method to address this type of numerical oscillation problem. The method uses local least-squares polynomial fitting in the space of active variables but selects neighboring sample points based on the Euclidean distance in the original input space to help to keep track of original neighborhood relationships. Our synthesized regression algorithm can be described as follows:

1. Given an unseen input sample $\boldsymbol{x}^*$, we select a set of N_f points closest to $\boldsymbol{x}^*$ from the set of all training samples, denoted by $\{\boldsymbol{x}_*^{(m)}\}_{m=1}^{N_f}$.
2. The N_f samples are fed into the trained PRNN to generate the samples of the active variables $\{\boldsymbol{z}_{*,A}^{(m)} = (\boldsymbol{g}(\boldsymbol{x}_*^{(m)}))_A\}_{m=1}^{N_f}$.

3. We perform least-squares polynomial fitting using the data $\{(\boldsymbol{z}_{*,A}^{(m)}, f(\boldsymbol{x}_*^{(m)})\}_{m=1}^{N_f}$, which is a subset of the training set. The approximation of $f(\boldsymbol{x}^*)$, denoted by $\hat{f}(\boldsymbol{x}^*)$, is defined by the value of the resulting polynomial at $(\boldsymbol{g}(\boldsymbol{x}^*))_A$.

Note that when the graph of $\widetilde{f}$ in $\boldsymbol{z}_A$ has several branches, the first two steps in the proposed synthesized regression encourage localization of the polynomial data fitting to only one of the branches. Indeed, the selected neighbors $\{\boldsymbol{x}_*^{(m)}\}_{m=1}^{N_f}$ to $\boldsymbol{x}^*$ usually stay on the same branch or intersecting region without many oscillations, as shown in the bottom row of Figure 3.

3. Experimental results. The goal of this section is twofold: The first is to test the influence of each ingredient of the proposed DRiLLS method on its overall performance, and the second is to investigate the numerical performance of the method in approximating high-dimensional functions. In particular, an ablation study is implemented in section 3.1, including PRNN versus RevNet and the effect of α in section 3.1.1, the effect of bounded derivative loss in section 3.1.2, and the synthesized regression versus some existing regression methods in section 3.1.3. Then, through extensive comparisons with the AS and the NLL methods, we demonstrate the effectiveness and accuracy of the proposed DRiLLS method under limited/sparse data. Particularly, high-dimensional example functions are considered in section 3.2, and a PDE-related application is given in section 3.3.

The training dataset of size N is randomly generated using the Latin hypercube sampling method [28]. To measure the approximation accuracy, we use the normalized root-mean-square error ($\texttt{NRMSE}$) and the relative l_1 error ($\texttt{RL}_1$) over a test set of M randomly selected input points from the domain

$$\texttt{NRMSE} = \frac{1}{\sqrt{M}} \left(\frac{\|\boldsymbol{f} - \hat{\boldsymbol{f}}\|_2}{\max \boldsymbol{f} - \min \boldsymbol{f}} \right), \quad \texttt{RL}_1 = \frac{\|\boldsymbol{f} - \hat{\boldsymbol{f}}\|_1}{\|\boldsymbol{f}\|_1}, \tag{3.1}$$

where $\boldsymbol{f} = (f(\boldsymbol{x}_{test}^{(1)}), \ldots, f(\boldsymbol{x}_{test}^{(M)}))$ are the exact function values and $\hat{\boldsymbol{f}} = (\hat{f}(\boldsymbol{x}_{test}^{(1)}), \ldots, \hat{f}(\boldsymbol{x}_{test}^{(M)}))$ are the approximated values. In the experiments, we set $M = 1000$ for low-dimensional problems ($d \le 3$) and $M = 10000$ for high-dimensional problems ($d > 3$). This procedure is replicated 10 times, and the average values are reported as the final $\texttt{NRMSE}$ and $\texttt{RL}_1$ errors for function approximation.

Our DRiLLS method is implemented using PyTorch. If not specified otherwise, we choose the following *default model setting*: $\boldsymbol{g}$ and $\boldsymbol{h}$ in the PRNN are constructed by FCNNs that contain four hidden layers with $10d$ (or 200 for $d > 20$) hidden neurons per layer, respectively; Tanh is used as the activation function; the hyperparameters $\lambda_1 = 1$, $\lambda_2 = 1$, $\alpha = 50$, and $\sigma = 0.01$ are selected in the loss function based on the ablation study presented in section 3.1 for low-dimensional cases and the numerical comparison of the proposed method with other methods (AS, NLL, and NN) for high-dimensional cases except stated otherwise; and $N_f = 30$ and cubic polynomial are used for the local least-squares fitting in the synthesized regression. For the *training* of PRNN, we use a combination of the Adam optimizer [17] and the L-BFGS optimizer [23]. The Adam iteration [17] is first applied with the initial learning rate 0.001, and the learning rate decays every 5000 steps by a factor of 0.7 for up to 60000 steps. Then the L-BFGS iteration is applied for a maximum of 1000 steps to accelerate the convergence. The training process is immediately stopped when training error reduces to 5×10^{-5}. Both the AS and the NLL methods used for comparison are implemented

in ATHENA[1] [26], which is a Python package for parameter space dimension reduction in the context of numerical analysis. All the experiments reported in this work are performed on an Ubuntu 20.04.2 LTS desktop with a 3.6-GHz AMD Ryzen 7 3700X CPU, 32 GB of DDR4 memory, and an NVIDIA RTX 2080Ti GPU. Code for this paper is available at https://github.com/sloooWTYK/DRiLLS.

3.1. Ablation studies. We first numerically investigate the effect of major components in the proposed DRiLLS method, including the PRNN, the loss functions, the hyperparameters, and the synthesized regression. Several functions of two dimensions are considered. Since the dimension d is 2, it is natural to take $\boldsymbol{\omega} = (0, 1)$, i.e., $k^* = 1$ in (2.6), with z_1 being the active variable and z_2 the inactive one in the transformed space of $\boldsymbol{z}$. For the same reason, two hidden layers are used for each of the FCNNs representing $\boldsymbol{g}$ and $\boldsymbol{h}$, different from the default settings. From the tests reported in sections 3.1.1 and 3.1.2, we observe that the Adam optimization during PRNN training terminated within 20000 steps in all cases, while the tests in section 3.1.3 required up to 60000 steps to meet the stopping criterion due to more complicated geometric structures in the target function.

To visually evaluate the function approximation, we present two types of plots: The *quiver plot* shows the gradient field of f (blue arrows) and the vector field corresponding to the second Jacobian column $\boldsymbol{J}_2$ (red arrows) on a 15×15 uniform grid, where increased orthogonality between the red and blue arrows indicates increased accuracy in the network mapping; The *regression plot* draws the approximated function values (red circles) over 400 randomly generated points in the domain together with the associated exact function values (blue stars), where good performance is indicated by a thin regression curve and a large degree of overlap between the blue stars and the red circles (exact and approximate function values).

3.1.1. PRNN versus RevNet and the effect of $\boldsymbol{\alpha}$. One of the main differences between the proposed PRNN and the RevNet is their treatments of *reversibility*: The former imposes it as a soft constraint, while the latter imposes a hard constraint (implemented by a special network structure). Furthermore, the special structure of the RevNet requires an equal separation of the inputs into two groups. Thus, if the input space has an odd dimension, it has to be padded with an auxiliary variable (e.g., a column of zero). On the other hand, the PRNN represents a larger class of functions than the RevNet [4], so that a better nonlinear transformation can be found when there is no need for explicit invertibility. To compare these two NN structures, the following two functions are considered for testing:

$$f_1(\boldsymbol{x}) = x_1^2 + x_2^2 \quad \text{and} \quad f_2(\boldsymbol{x}) = \frac{5}{8}x_1^2 + \frac{5}{8}x_2^2 - \frac{3}{4}x_1x_2, \tag{3.2}$$

where the domain of $\boldsymbol{x}$ is either $\Omega_A^2 = [0, 1]^2$ or $\Omega_B^2 = [-1, 1]^2$. Note that both f_1 and f_2 reach their minimum at the origin, which is located in the interior of Ω_B^2 but only on the boundary of Ω_A^2. Since we focus on the influence of reversibility in this section, we temporarily set $\lambda_1 = 1$ and $\lambda_2 = 0$. The corresponding total loss $\mathcal{L}$ for our DRiLLS method defined by (2.3) then becomes

$$\mathcal{L}_{\texttt{PRNN}} := \mathcal{L}_1 + \mathcal{L}_2 \quad \text{and} \quad \mathcal{L}_{\texttt{RevNet}} := \mathcal{L}_2,$$

respectively, because $\mathcal{L}_1$ is automatically zero in the case of RevNet. In the following tests, the RevNet uses 10 RevNet Blocks with two neurons each, as the input space

[1] ATHENA codes available at https://github.com/mathLab/ATHENA.

has the dimension 2 and a step size of 0.25 (see [33] for details about the used RevNet structure). We choose the size of the sample dataset for training to be $N = 500$, and both the PRNN and the RevNet are trained using the same dataset.

The testing results of $f_1(\boldsymbol{x})$ are presented in Figure 4 for the case $\boldsymbol{x} \in \Omega_A^2$ and in Figure 5 for the case $\boldsymbol{x} \in \Omega_B^2$, where several choices of α are considered, i.e., the first column for $\alpha = 0$, the second column for $\alpha = 25$, and the third column for $\alpha = 50$. It is observed that both network structures, the PRNN and the RevNet, work well for f_1 with the domain Ω_A^2, as shown in Figure 4. It is worth noting that $\frac{\partial f_1}{\partial x_1}(\boldsymbol{x}) \geq 0$ and $\frac{\partial f_1}{\partial x_2}(\boldsymbol{x}) \geq 0$ for any $\boldsymbol{x} \in \Omega_A^2$; i.e, the behavior of f_1 in Ω_A^2 is somehow monotonic. However, when the domain is changed to Ω_B^2, the behavior of f_1 in Ω_A^2 is not monotonic anymore, and the RevNet encounters difficulties in finding the appropriate active variable. Indeed, as shown in the third row of Figure 5, the gradient is not orthogonal to $\boldsymbol{J}_2$ at many points no matter the value of α, which indicates the function value is still sensitive to the first inactivate variable z_2. This further leads to larger errors in the regression process and function approximation, as seen in the fourth row of Figure 5.

The testing results of $f_2(\boldsymbol{x})$ are displayed in Figures 6 and 7 for the function, respectively, defined in Ω_A^2 and Ω_B^2. We remark that the behavior of f_2 in either Ω_A^2 or in Ω_B^2 is not monotonic at all. It is observed from Figures 6 and 7 that the PRNN achieves superior performances on both domains: The quiver plots indicate that the RevNet has difficulty in ensuring the function value to be insensitive to z_2 in both Ω_A^2 and Ω_B^2 cases, and the associated regression plots show that the RevNet produces a more erroneous function approximation. The PRNN, on the contrary, still works well on both domains, which further leads to more accurate function approximations.

Meanwhile, we also observe that the value of α does not have much impact on the performance of RevNet. For the PRNN, the effect of α on the performance also seems negligible for the case $\boldsymbol{x} \in \Omega_A^2$ but becomes significantly different for the case $\boldsymbol{x} \in \Omega_B^2$. As α increases from 0 to 25 and 50, the learned level sets and the function approximations get more and more accurate, especially for f_2. As shown in the first rows of Figures 5 and 7, red arrows are well perpendicular to the blue arrows in the quiver plots for two larger values of α, manifesting more effective dimension reductions. Moreover, fewer blue dots are visible in the regression plots in the third column than in the first two columns, which indicates less discrepancy between the predicted values and the exact function values.

3.1.2. The effectiveness of the bounded derivative loss. We use f_2 with the domain Ω_B^2 to investigate the effect of the bounded derivative loss $\mathcal{L}_2$, which is a new loss term, compared to those used in the NLL method. The purpose of $\mathcal{L}_2$ is to reduce the oscillation in the function values after they are projected onto the active variable space; thus, it mainly can be regarded as a regularization term.

To check whether the proposed bounded derivative loss helps the training process of the proposed PRNN in our DRiLLS method, we vary the value of λ_2 from 0 to 1 and 100 while fixing the other experimental settings. The training dataset again has 500 samples. The evolutions of the total loss $\mathcal{L}$, the pseudoreversibility loss $\mathcal{L}_1$, and the active direction fitting loss $\mathcal{L}_2$ during the training process are presented in Figure 8. It is observed that the pseudoreversibility loss $\mathcal{L}_1$ is not affected by the choice of λ_2, but the total training loss and the active direction fitting loss both decay faster when $\lambda_2 = 1$ than when $\lambda_2 = 0$. Conversely, the even larger value $\lambda_2 = 100$ does not further accelerate the training process.

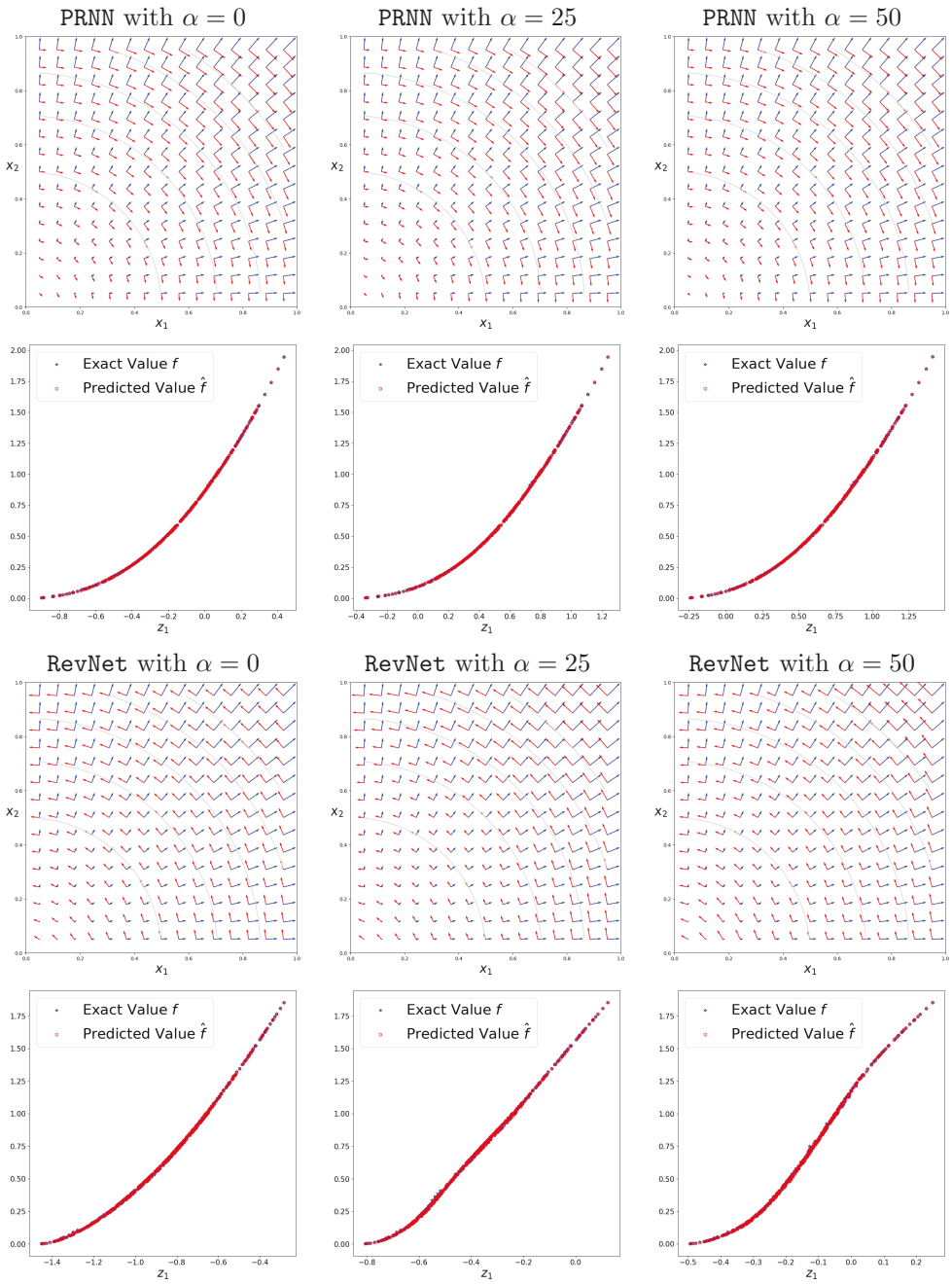


FIG. 4. *Level set learning and function approximation results produced by our DRiLLS method with PRNN (the quiver plot in row* 1 *and the regression plot in row* 2*) or the RevNet (the quiver plot in row* 3 *and the regression plot in row* 4*) for* $f_1(\boldsymbol{x}) = x_1^2 + x_2^2$ *in* $\Omega_A^2 = [0,1]^2$ *at three different values of* $\alpha = 0, 25, 50$*, respectively. There is no critical point in the interior of the domain* Ω_A*, and both PRNN and RevNet successfully learn the level sets of the target function.*

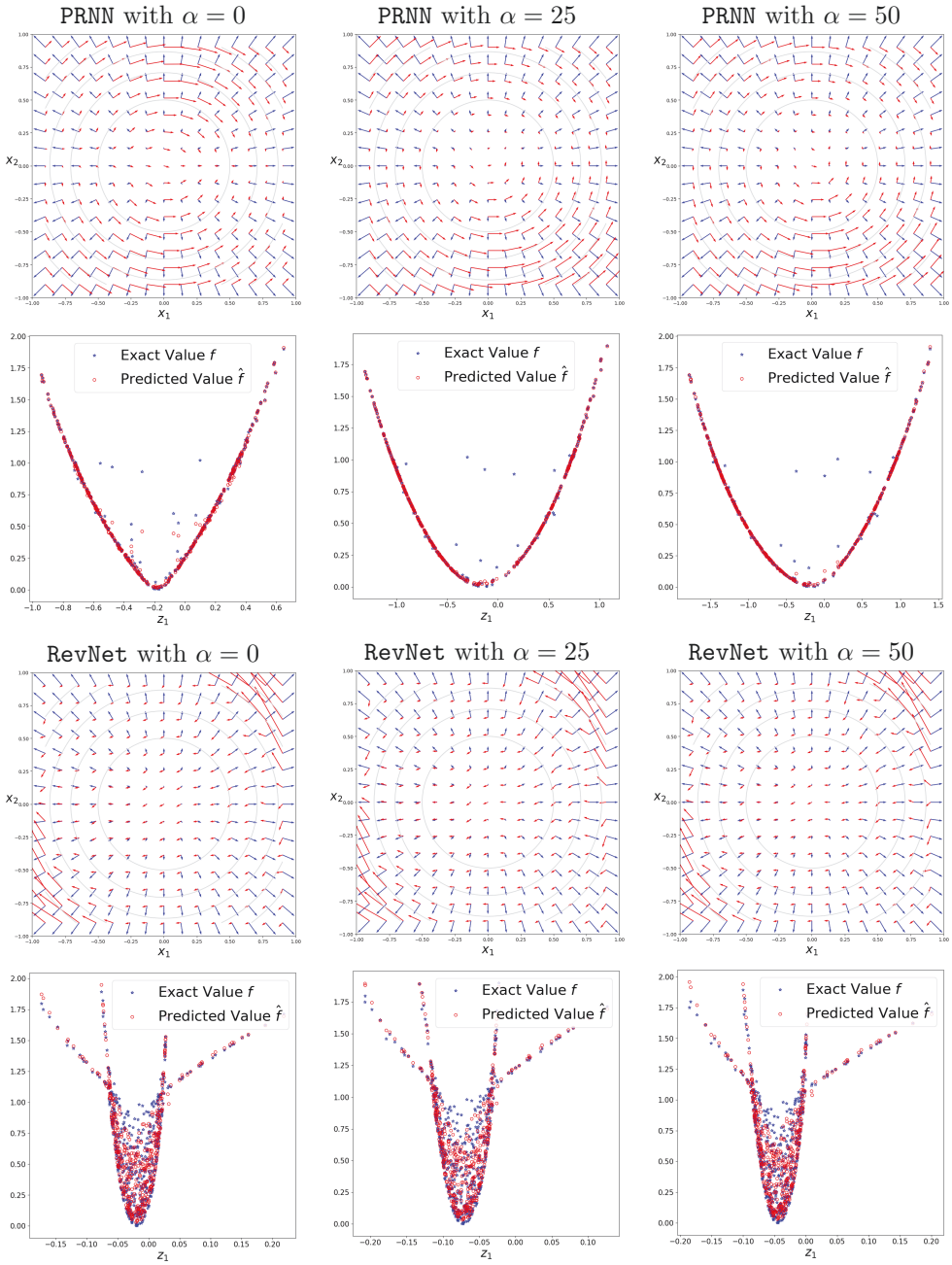


FIG. 5. *Level set learning and function approximation results produced by our DRiLLS method with the PRNN (the quiver plot in row* 1 *and the regression plot in row* 2*) or the RevNet (the quiver plot in row* 3 *and the regression plot in row* 4*) for* $f_1(\boldsymbol{x}) = x_1^2 + x_2^2$ *in* $\Omega_B^2 = [-1, 1]^2$ *at three different values of* $\alpha = 0, 25, 50$*, respectively. RevNet fails to learn the level sets of the target function because it cannot handle the interior critical point at the origin. In comparison, the PRNN successfully learns these level sets partly because it does not enforce the hard reversibility around the critical point.*

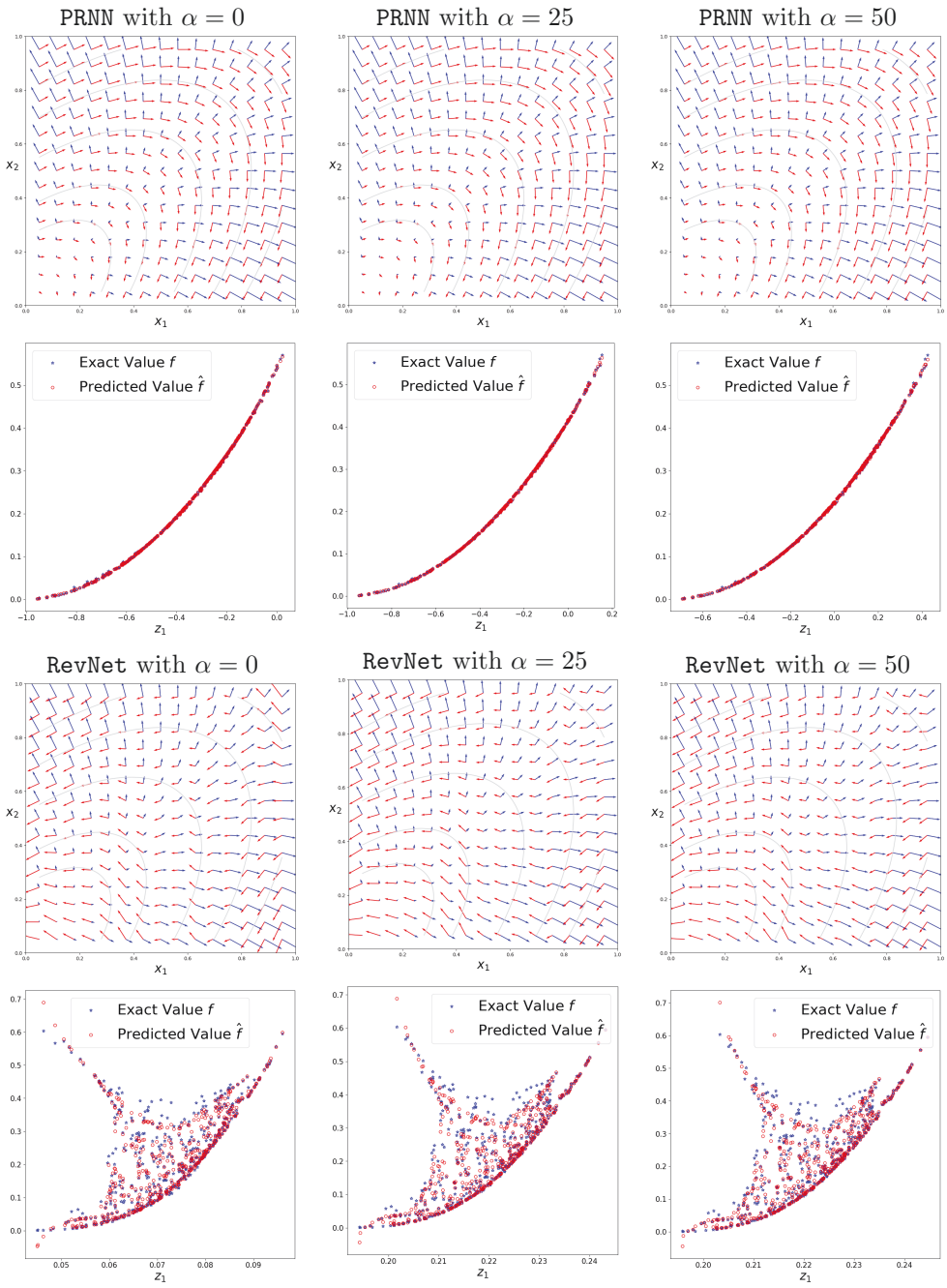


FIG. 6. *Level set learning and function approximation results produced by our DRiLLS method with the PRNN (the quiver plot in row* 1 *and the regression plot in row* 2*) or the RevNet (the quiver plot in row* 3 *and the regression plot in row* 4*) for* $f_2(\boldsymbol{x}) = \frac{5}{8}x_1^2 + \frac{5}{8}x_2^2 - \frac{3}{4}x_1x_2$ *in* $\Omega_A^2 = [0,1]^2$ *at three different values of* $\alpha = 0, 25, 50$*, respectively. PRNN successfully learns the level sets of the target function, but RevNet is somehow unable.*

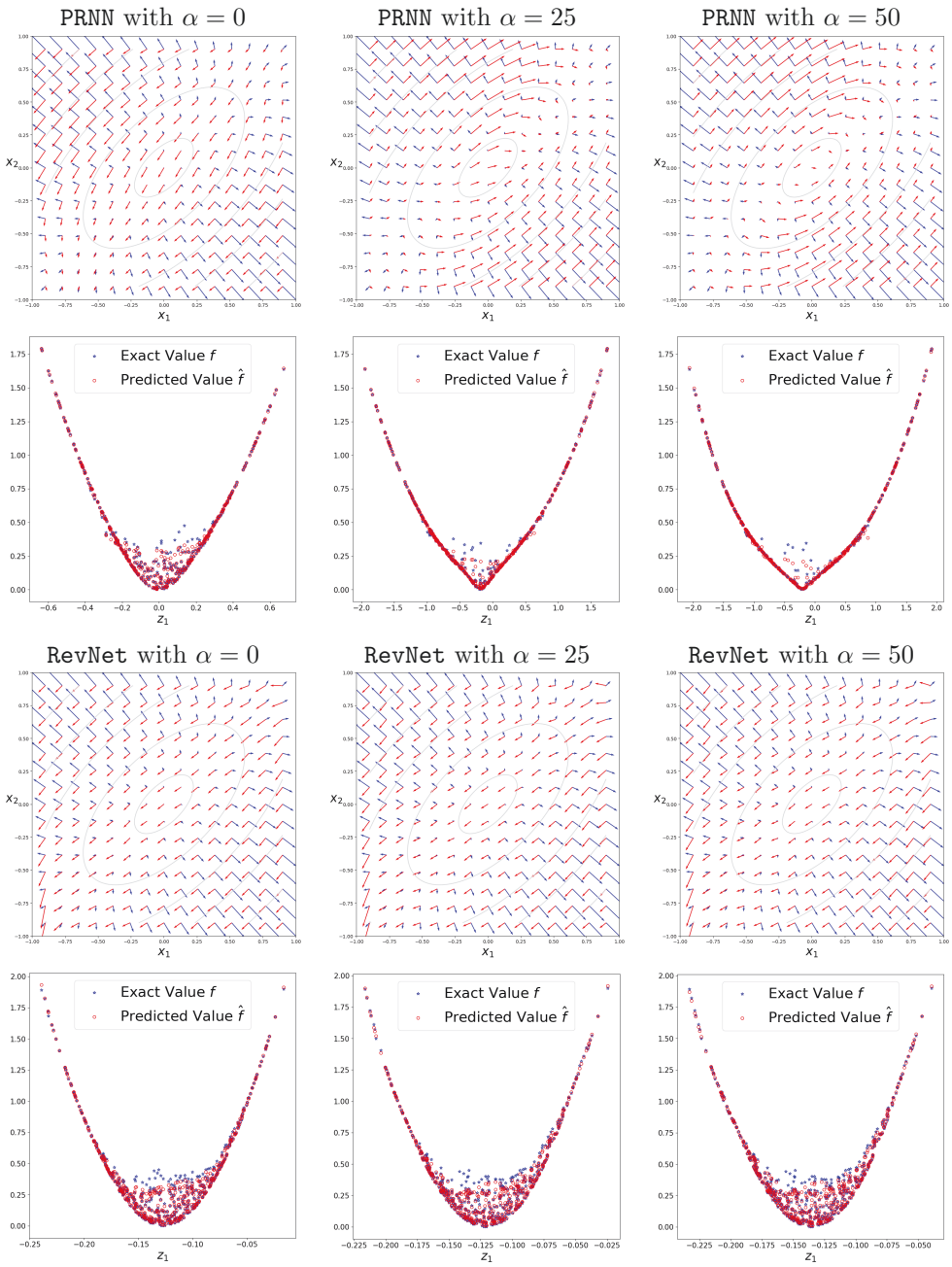
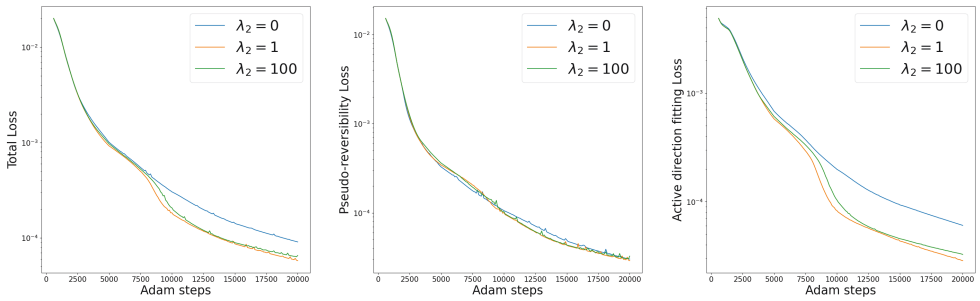


FIG. 7. *Level set learning and function approximation results produced by our DRiLLS method with the PRNN (the quiver plot in row* 1 *and the regression plot in row* 2*) or the RevNet (the quiver plot in row* 3 *and the regression plot in row* 4*) for* $f_2(\boldsymbol{x}) = \frac{5}{8}x_1^2 + \frac{5}{8}x_2^2 - \frac{3}{4}x_1x_2$ *in* $\Omega_B^2 = [-1,1]^2$ *at three different values of* $\alpha = 0, 25, 50$*, respectively. PRNN successfully learns the level sets of the target function, but RevNet is somehow unable.*



(a) Total Loss (b) Pseudo-reversibility Loss (c) Active direction fitting Loss

FIG. 8. *Evolutions of the total loss $\mathcal{L}$ (left), the pseudoreversibility loss $\mathcal{L}_1$ (middle), and the active direction fitting loss $\mathcal{L}_2$ (right) with three different values of $\lambda_2 = 0, 1, 100$ during the training process of the PRNN for f_2 in $\Omega_B^2 = [-1, 1]^2$.*

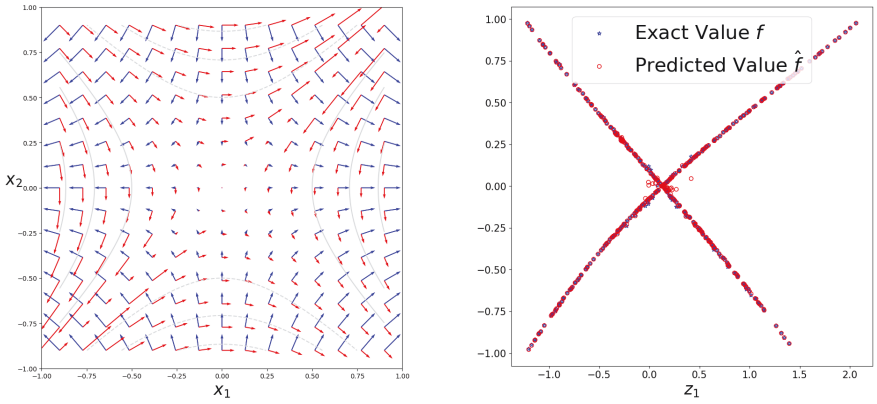


FIG. 9. *Level set learning and function approximation results produced by our DRiLLS method for f_3 in $\Omega_B^2 = [-1, 1]^2$: the quiver plot (left) and the regression plot (right). The synthesized regression approach successfully overcomes the numerical oscillation issue illustrated in the top row of Figure 3.*

3.1.3. The synthesized regression versus other regression methods. Once the transformation to the active variable $\boldsymbol{z}_A$ is obtained through the PRNN, we apply the proposed synthesized regression for approximating the target function. To better demonstrate the advantage of our synthesized regression, we consider the following example featured in Figure 3:

$$f_3(\boldsymbol{x}) = x_1^2 - x_2^2 \quad \text{for} \quad \boldsymbol{x} \in \Omega_B^2. \tag{3.3}$$

Due to the complicated behavior of the function f_3 in $[-1, 1]^2$, we set the size of training dataset $N = 2500$ in the PRNN, and the associated quiver and regression plots produced by our method are presented in Figure 9. The former demonstrates the efficacy of PRNN dimension reduction, as the derivative in the function with respect to z_2 is tangent to the level sets, and the latter indicates that accurate regressions have been obtained, as almost all the blue stars and red circles coincide with each other, though the graph of f has several branches.

The performance of the proposed synthesized regression is also compared with some other popular regression methods based on the same PRNN transformed data,

TABLE 1
Approximation errors by different regression methods on the same PRNN transformed data for f_3 in $\Omega_B^2 = [-1,1]^2$.

	Synthesized regression	Direct local fitting	Global fitting	Neural network
NRMSE	0.86%	20.49%	20.16%	19.93%
RL_1	1.32%	91.11%	93.89%	91.27%

including the polynomial regression in local and global fashions and the nonlinear regression by NNs. In particular, cubic polynomial fitting is applied, and the NN regression uses an FCNN of three hidden layers with 20 neurons in each layer. The function approximation errors are summarized in Table 1, which shows that the direct local fitting, the global fitting, and the NN regression all fail to provide accurate predictions, while the synthesized regression performs significantly well.

3.2. High-dimensional function approximation with limited data. Here we compare our DRiLLS method with two popular dimension reduction methods, the AS and the NLL, for function approximation with limited/sparse data. To ensure a fair comparison, the proposed synthesized regression will be applied to all compared methods for regression after AS variables are identified. In particular, the dimension of the active variables is set to $k^* = 1$ and 2 for DRiLLS and NLL and similarly for AS, which are often the typical choices in practice. We consider the following four functions:

$$\begin{aligned} f_4(\boldsymbol{x}) &= \sum_{i=1}^{d} x_i^2, & f_5(\boldsymbol{x}) &= \sin\left(\sum_{i=1}^{d} x_i^2\right), \\ f_6(\boldsymbol{x}) &= \prod_{i=1}^{d} \frac{1}{1+x_i^2}, & f_7(\boldsymbol{x}) &= -x_d^2 + \sum_{i=1}^{d-1} x_i^2. \end{aligned} \tag{3.4}$$

For f_4, f_5 and f_6, $\Omega_A^d = [0,1]^d$ with $d = 10, 20$ and $\Omega_B^d = [-1,1]^d$ with $d = 8, 12$ are, respectively, used as the domain of the functions. For f_7, we only take $\Omega_B^d = [-1,1]^d$ with $d = 8, 12$ as the domain. Obviously, the behaviors of these functions are much more complicated in Ω_B^d than in Ω_A^d. We observed that for all the tests reported in this section, their training processes again terminated within 60000 Adam optimization steps.

Numerical approximation errors produced by our DRiLLS method as well as the NLL and AS methods are reported for the above examples in Tables 2–5, where the sizes of the training dataset are selected as $N = 500$, 2500, and 10000 for Ω_A^d and $N = 2500$, 10000, and 40000 for Ω_B^d. Note that the training samples in all these cases are very sparse due to the high dimension of input space. Some observations from these tables are summarized below.

For the functions f_4, f_5, f_6 with the domain Ω_A^{10}, both DRiLLS and NLL perform better than AS, as their approximation errors are several times smaller than that of AS with all tested sizes of training samples 500, 2500, and 10000. When 500 samples are used for training, DRiLLS performs similarly to NLL but gradually outperforms NLL when the size of the training dataset increases to 2500 and 10000. For the functions f_4, f_5, f_6 with the domain Ω_A^{20}, NLL performs better than AS. When 500 training samples are used, DRiLLS achieves worse results than NLL. For f_4 and f_5, it even yields errors bigger than AS. However, once the size of the training dataset size increases to 2500 and 10000, the performance of DRiLLS improves significantly: Its errors are close to that of NLL for f_4 and f_5 and better than that of NLL for f_6.

TABLE 2
Numerical approximation errors produced by DRiLLS, NLL, and AS for f_4 on various domains.

f_4 in $\Omega_A^{10} = [0,1]^{10}$	500		2500		10000	
	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	0.60%	0.80%	0.22%	0.31%	0.20%	0.26%
DRiLLS ($k^* = 2$)	0.61%	0.82%	0.22%	0.31%	0.20%	0.26%
NLL ($k^* = 1$)	0.32%	0.43%	0.30%	0.43%	0.32%	0.45%
NLL ($k^* = 2$)	0.37%	0.50%	0.40%	0.56%	0.37%	0.51%
AS ($k^* = 1$)	3.52%	5.51%	3.20%	4.88%	2.68%	4.19%
AS ($k^* = 2$)	3.81%	5.84%	3.49%	5.22%	2.92%	4.39%
f_4 in $\Omega_A^{20} = [0,1]^{20}$	500		2500		10000	
	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	9.74%	11.18%	0.57%	0.62%	0.39%	0.40%
DRiLLS ($k^* = 2$)	11.25%	12.68%	0.68%	0.75%	0.73%	0.71%
NLL ($k^* = 1$)	0.31%	0.33%	0.28%	0.31%	0.37%	0.38%
NLL ($k^* = 2$)	0.39%	0.40%	0.35%	0.36%	0.40%	0.38%
AS ($k^* = 1$)	3.83%	4.52%	3.74%	4.35%	3.68%	4.24%
AS ($k^* = 2$)	4.27%	4.84%	4.07%	4.60%	3.97%	4.46%
f_4 in $\Omega_B^{8} = [-1,1]^{8}$	2500		10000		40000	
	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	4.26%	3.19%	2.25%	1.51%	1.53%	1.17%
DRiLLS ($k^* = 2$)	2.63%	2.52%	1.55%	1.39%	1.05%	0.87%
AS ($k^* = 1$)	11.42%	19.14%	9.73%	15.49%	7.53%	12.14%
AS ($k^* = 2$)	12.66%	20.13%	9.96%	16.25%	8.04%	13.62%
f_4 in $\Omega_B^{12} = [-1,1]^{12}$	2500		10000		40000	
	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	13.71%	19.66%	3.59%	2.40%	2.22%	1.59%
DRiLLS ($k^* = 2$)	13.88%	19.69%	2.93%	2.47%	1.86%	1.80%
AS ($k^* = 1$)	12.96%	19.38%	12.17%	17.45%	10.98%	15.22%
AS ($k^* = 2$)	14.35%	20.12%	12.70%	18.14%	11.55%	15.96%

For the functions f_4, f_5, f_6, f_7 with the domain Ω_B^8, DRiLLS achieves the best performance among all three methods with all tested sizes of training samples 2500, 10000, and 40000. Particularly, NLL does not work at all, partially due to the existence of interior critical points in the domain. For the functions f_4, f_5, f_6, f_7 with the domain Ω_B^{12}, NLL still does not work at all, as in the case of Ω_B^8. For the training dataset of the size 2500, both DRiLLS and AS cannot achieve good approximations. However, once the sample size increases to 10000 and 40000, DRiLLS yields a much more accurate function approximation whose errors are several times smaller than that of AS with one or two active coordinates.

For the functions f_4, f_5, f_6 with the domains Ω_A^{10} and Ω_A^{20}, DRiLLS with $k^* = 2$ achieves approximation errors very close to that with $k^* = 1$. For f_4, f_5, f_6, f_7 with the domains Ω_B^8 and Ω_B^{12}, DRiLLS with $k^* = 2$ usually achieves slightly smaller errors than that with $k^* = 1$, but these values are generally on the same order. This makes sense since $k^* = 1$ is the ideal and natural choice when the level sets of the target function are well captured. On the other hand, AS with two active variables performs almost the same as that with only one active variable, partly because the input dimension cannot be effectively reduced by linear transformations when the level sets are nonlinear [33].

Overall, the approximation error by our DRiLLS method quickly decreases as the size of training dataset increases, and our method significantly outperforms the NLL and AS methods when the dataset size becomes relatively large.

TABLE 3
Numerical approximation errors produced by DRiLLS, NLL, and AS for f_5 on various domains.

f_5 in	500		2500		10000	
$\Omega_A^{10} = [0,1]^{10}$	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	1.54%	3.56%	0.71%	1.63%	0.51%	1.18%
DRiLLS ($k^* = 2$)	1.58%	3.68%	0.73%	1.60%	0.54%	1.21%
NLL ($k^* = 1$)	1.19%	2.27%	1.08%	2.52%	1.13%	2.76%
NLL ($k^* = 2$)	1.42%	2.56%	1.03%	2.33%	0.84%	1.95%
AS ($k^* = 1$)	8.95%	22.95%	8.07%	20.66%	6.91%	17.55%
AS ($k^* = 2$)	10.02%	24.97%	8.58%	21.80%	7.23%	18.38%
f_5 in	500		2500		10000	
$\Omega_A^{20} = [0,1]^{20}$	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	28.73%	79.29%	3.75%	7.72%	2.81%	5.73%
DRiLLS ($k^* = 2$)	33.28%	87.72%	4.06%	7.72%	3.08%	5.82%
NLL ($k^* = 1$)	5.09%	7.46%	3.48%	5.78%	2.58%	4.92%
NLL ($k^* = 2$)	5.92%	8.13%	4.17%	6.22%	3.07%	5.02%
AS ($k^* = 1$)	13.89%	32.84%	12.96%	30.99%	12.60%	29.90%
AS ($k^* = 2$)	15.54%	35.73%	14.04%	32.87%	13.42%	31.44%
f_5 in	2500		10000		40000	
$\Omega_B^{8} = [-1,1]^{8}$	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	9.18%	15.90%	6.34%	9.96%	4.56%	6.66%
DRiLLS ($k^* = 2$)	8.06%	13.56%	5.10%	8.19%	3.11%	4.90%
AS ($k^* = 1$)	25.21%	62.51%	21.29%	52.42%	17.29%	42.63%
AS ($k^* = 2$)	26.61%	64.48%	22.42%	54.13%	18.31%	44.15%
f_5 in	2500		10000		40000	
$\Omega_B^{12} = [-1,1]^{12}$	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	26.86%	74.17%	17.17%	30.54%	11.95%	20.81%
DRiLLS ($k^* = 2$)	22.54%	49.60%	15.55%	27.14%	9.62%	15.90%
AS ($k^* = 1$)	26.82%	73.84%	24.58%	67.50%	22.15%	60.43%
AS ($k^* = 2$)	28.66%	76.43%	26.76%	70.45%	23.78%	62.81%

3.3. A PDE-related application. The PDE-constrained optimization and optimal control problems in engineering applications often lead to multiquery computing scenarios where multiple numerical PDE solves are required as parameters of the problems change, which results in huge or even prohibitive computational costs. On the other hand, the goal of multiquery numerical simulations is often to determine the response of certain quantities of interest (QoI) to the varying parameters and/or the sensitivity of the QoI with respect to its parameters. Therefore, a function approximation method can be used to model the QoI offline as a function of system parameters, which can then be applied online to provide real-time responses. In the following, we demonstrate the performance of our DRiLLS method through a thermal block engineering problem, which is a popular test case for model order reduction algorithms [15].

Consider the heat diffusion on the domain $\Omega = [0,1]^2$ as follows:

$$\begin{aligned} -\nabla \cdot (\kappa(x,y;\boldsymbol{\mu})\nabla u(x,y;\boldsymbol{\mu})) &= 0 \quad \text{in}\,\Omega, \\ u(x,y;\boldsymbol{\mu}) &= 0 \quad \text{on}\,\Gamma_D, \\ (\kappa(x,y;\boldsymbol{\mu})\nabla u(x,y;\boldsymbol{\mu})) \cdot \mathbf{n}(x,y) &= i \quad \text{on}\,\Gamma_{N,i} \text{ for } i = 0,1, \end{aligned} \tag{3.5}$$

where the zero Dirichlet boundary condition is imposed on the upper boundary, denoted by Γ_D; the left and right edges of the domain are insulated, denoted by $\Gamma_{N,0}$; and unit flux is assumed on the lower boundary, denoted by $\Gamma_{N,1}$. Suppose that the domain is uniformly divided into p subblocks $\{\Omega_i\}_{i=1}^p$ and that a piecewise constant

TABLE 4
Numerical approximation errors produced by DRiLLS, NLL, and AS for f_6 on various domains.

f_6 in	500		2500		10000	
$\Omega_A^{10} = [0,1]^{10}$	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	0.32%	1.44%	0.12%	0.39%	0.09%	0.27%
DRiLLS ($k^* = 2$)	0.29%	1.31%	0.11%	0.37%	0.09%	0.27%
NLL ($k^* = 1$)	0.73%	3.29%	0.59%	2.42%	0.32%	1.41%
NLL ($k^* = 2$)	0.95%	2.76%	0.51%	2.12%	0.57%	2.31%
AS ($k^* = 1$)	2.19%	9.91%	2.03%	8.73%	1.76%	7.58%
AS ($k^* = 2$)	2.58%	10.73%	2.09%	9.30%	1.97%	7.99%
f_6 in	500		2500		10000	
$\Omega_A^{20} = [0,1]^{20}$	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	1.28%	13.35%	0.27%	2.48%	0.14%	1.27%
DRiLLS ($k^* = 2$)	1.34%	13.63%	0.23%	1.97%	0.14%	1.23%
NLL ($k^* = 1$)	0.67%	5.11%	1.76%	16.86%	0.96%	8.23%
NLL ($k^* = 2$)	0.85%	6.88%	0.46%	4.20%	0.54%	4.83%
AS ($k^* = 1$)	1.98%	17.25%	1.55%	15.93%	1.96%	15.73%
AS ($k^* = 2$)	2.37%	19.84%	1.90%	17.48%	1.99%	16.93%
f_6 in	2500		10000		40000	
$\Omega_B^8 = [-1,1]^8$	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	4.31%	11.76%	2.56%	6.36%	1.74%	3.95%
DRiLLS ($k^* = 2$)	3.38%	8.15%	2.09%	5.05%	1.06%	2.66%
AS ($k^* = 1$)	8.43%	36.86%	6.43%	27.92%	4.94%	20.31%
AS ($k^* = 2$)	8.96%	38.55%	6.90%	29.48%	5.17%	21.98%
f_6 in	2500		10000		40000	
$\Omega_B^{12} = [-1,1]^{12}$	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	8.70%	61.31%	3.27%	17.78%	2.38%	12.27%
DRiLLS ($k^* = 2$)	9.82%	64.64%	2.80%	14.60%	2.02%	10.16%
AS ($k^* = 1$)	8.04%	62.19%	7.01%	53.50%	6.11%	44.14%
AS ($k^* = 2$)	10.13%	67.74%	8.34%	56.40%	6.61%	46.68%

TABLE 5
Numerical approximation errors produced by DRiLLS, NLL, and AS for f_7 on various domains.

f_7 in	2500		10000		40000	
$\Omega_B^8 = [-1,1]^8$	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	3.30%	5.26%	2.14%	3.28%	1.73%	2.36%
DRiLLS ($k^* = 2$)	3.25%	3.87%	1.63%	1.94%	1.13%	1.34%
AS ($k^* = 1$)	10.83%	24.73%	8.73%	19.55%	6.94%	15.51%
AS ($k^* = 2$)	11.84%	26.12%	9.61%	21.26%	7.52%	16.64%
f_7 in	2500		10000		40000	
$\Omega_B^{12} = [-1,1]^{12}$	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^* = 1$)	12.81%	22.42%	11.42%	20.09%	2.18%	2.52%
DRiLLS ($k^* = 2$)	9.49%	16.02%	2.91%	2.98%	1.74%	1.67%
AS ($k^* = 1$)	12.38%	22.36%	11.54%	19.95%	10.02%	17.34%
AS ($k^* = 2$)	13.65%	23.71%	12.22%	21.14%	10.77%	18.41%

diffusion coefficient is assumed in each subblock whose magnitude could vary in a prescribed interval. That is,

$$\kappa(x, y; \boldsymbol{\mu}) := \sum_{i=1}^{p} \mu_i \chi_{\Omega_i}(x, y)$$

with $\boldsymbol{\mu} = (\mu_1, \mu_2, \cdots, u_p) \in \mathcal{P} := [0.1, 10]^p$. The QoI is the average temperature at the lower boundary

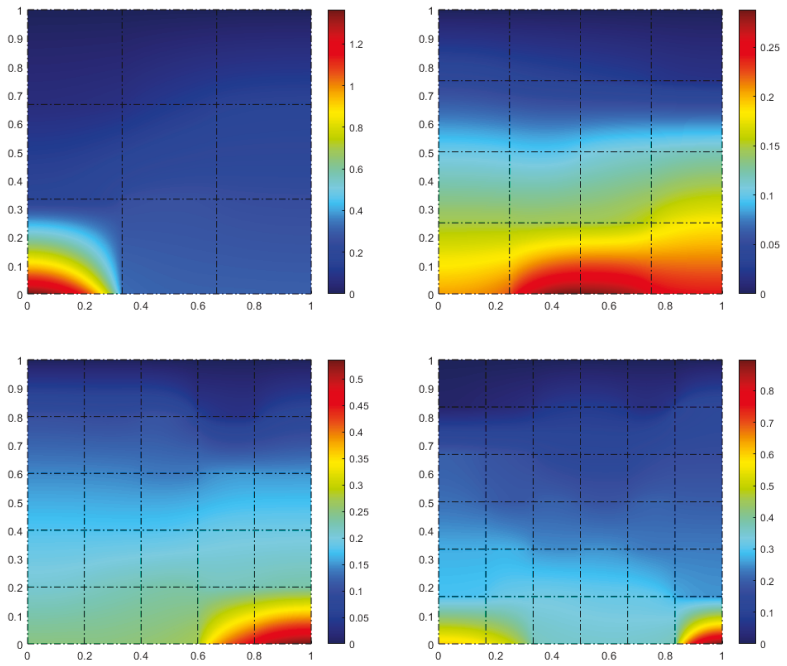


FIG. 10. *Some sample solutions to the thermal block problem for the cases of* $p = 9, 16, 25, 36$, *respectively.*

$$f(\boldsymbol{\mu}) := \int_{\Gamma_{N,1}} u(x, y; \boldsymbol{\mu})\, ds, \tag{3.6}$$

which can be regarded as a function of p input variables. To learn this function, N samples are randomly selected in the parameter space $\mathcal{P}$, and the finite element method is taken to solve (3.5) and evaluate $f(\boldsymbol{\mu})$ [3]. The derivative information, $\nabla_{\boldsymbol{\mu}} f$, is then calculated using an adjoint approach. As an example, several sample solutions of the thermal block problem are plotted in Figure 10 for the cases we consider here.

Assume the dimension of the parametric space to be $p = 9$, 16, 25, and 36, respectively. The DRiLLS method is used to learn and predict the QoI (3.6). For all cases, $\boldsymbol{g}$ and $\boldsymbol{h}$ are modeled by two FCNN with four hidden layers, and the size of the training dataset is $N = 10000$. These experiments use the ideal choice $\boldsymbol{\omega} = (0, 1, \cdots, 1)$ because the results in section 3.2 show that our DRiLLS method can perform well even with $k^* = 1$. Moreover, $\lambda_1 = 1$, $\lambda_2 = 100$, $\alpha = 50$ are set in the loss function. We observe that this example benefits from some L-BFGS optimization steps in addition to the 60000 Adam optimization steps used by default. To evaluate the sensitivity of the QoI with respect to transformed variables $\boldsymbol{z}$, we calculate the following relative sensitivity with respect to z_i: For $i = 1, \ldots, p$,

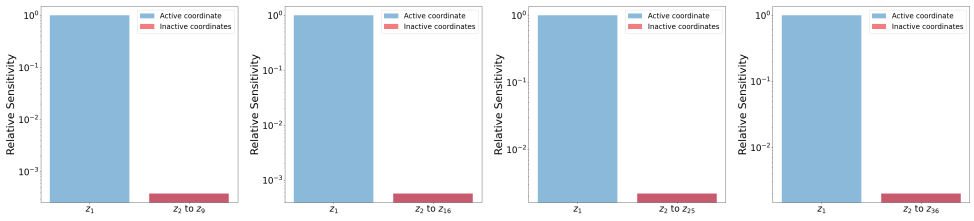
$$\mathtt{RS_i} := \frac{\left| \frac{1}{N} \sum_{n=1}^{N} \nabla f(\boldsymbol{\mu}_{test}^{(n)}) \cdot \frac{\partial \hat{\boldsymbol{\mu}}_{test}^{(n)}}{\partial z_i} \right|}{\sum_{m=1}^{p} \left| \frac{1}{N} \sum_{n=1}^{N} \nabla f(\boldsymbol{\mu}_{test}^{(n)}) \cdot \frac{\partial \hat{\boldsymbol{\mu}}_{test}^{(n)}}{\partial z_m} \right|}, \tag{3.7}$$

where $\hat{\boldsymbol{\mu}}_{test}^{(n)} = \boldsymbol{h} \circ \boldsymbol{g}(\boldsymbol{\mu}_{test}^{(n)})$. The numerical results on relative sensitivities of the QoI produced by our DRiLLS method are reported in Table 6 and also visually illustrated

TABLE 6

Results on the relative sensitivities of the QoI (3.6) *to the transformed variables* $\boldsymbol{z}$ *produced by our DRiLLS method with* $k^* = 1$.

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9
9D	9.99e−01	2.94e−05	5.12e−05	3.31e−05	2.95e−05	1.71e−05	5.23e−05	5.84e−05	1.12e−04
16D	9.99e−01	2.34e−05	3.01e−05	4.95e−05	6.28e−05	2.33e−05	2.00e−05	4.24e−05	4.31e−05
25D	9.97e−01	1.46e−04	6.99e−05	1.14e−04	2.73e−05	7.01e−05	1.05e−04	4.27e−05	1.09e−04
36D	9.98e−01	3.51e−05	1.19e−04	2.55e−05	6.42e−05	2.54e−05	5.61e−05	6.74e−05	6.87e−05
	z_{10}	z_{11}	z_{12}	z_{13}	z_{14}	z_{15}	z_{16}	z_{17}	z_{18}
16D	1.31e−05	7.36e−05	5.08e−05	3.58e−05	4.58e−05	2.24e−05	3.07e−05		
25D	3.67e−05	1.49e−04	1.55e−04	8.76e−05	1.50e−04	1.32e−04	1.21e−04	5.11e−05	6.97e−05
36D	8.46e−05	4.10e−05	3.55e−05	2.44e−05	6.33e−05	3.02e−05	7.03e−05	8.38e−05	5.61e−05
	z_{19}	z_{20}	z_{21}	z_{22}	z_{23}	z_{24}	z_{25}	z_{26}	z_{27}
25D	6.96e−05	4.92e−05	1.18e−04	7.68e−05	1.07e−04	8.32e−05	5.26e−05		
36D	2.77e−5	4.01e−05	1.13e−04	2.32e−05	4.07e−05	5.33e−05	8.13e−05	1.12e−04	3.21e−05
	z_{28}	z_{29}	z_{30}	z_{31}	z_{32}	z_{33}	z_{34}	z_{35}	z_{36}
36D	4.23e−05	3.78e−05	8.15e−05	5.28e−05	4.94e−05	3.22e−05	4.33e−05	6.73e−05	1.28e−04

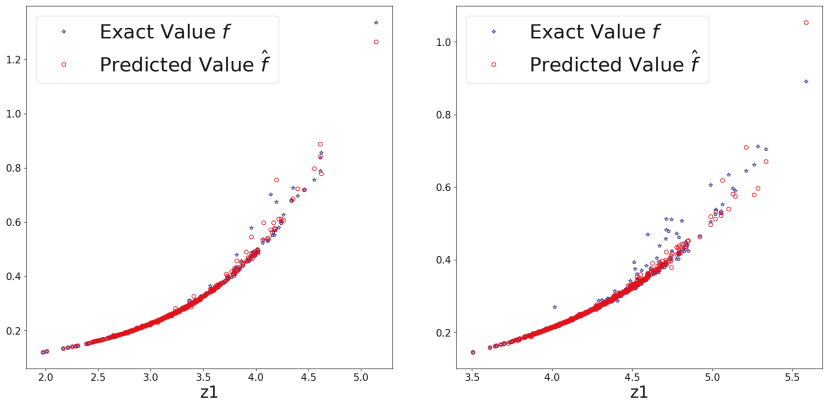


(a) 9D (b) 16D (c) 25D (d) 36D

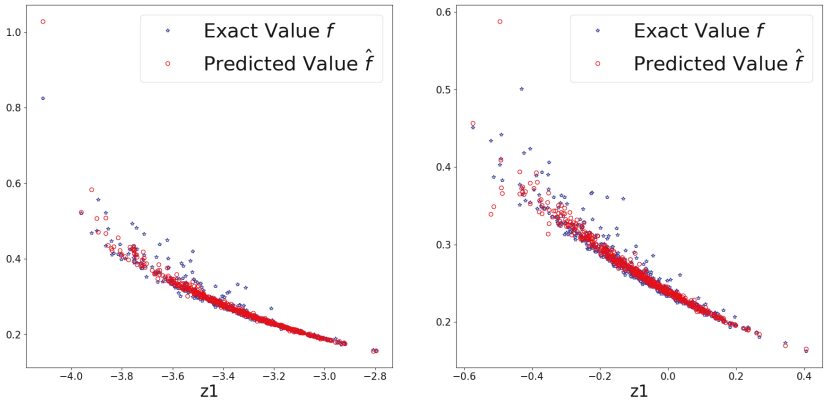
FIG. 11. *Visual illustration of the relative sensitivities of the QoI* (3.6) *to the transformed variables* $\boldsymbol{z}$ *produced by our DRiLLS method with* $k^* = 1$.

in Figure 11. In all cases, it is seen that the target function is sensitive only to the active variable z_1, as desired. The regression plots are presented in Figure 12 and again show good agreement between the exact values and the predicted values. For the purpose of comparison, the approximation errors produced by our DRiLLS method (with $k^* = 1$) are presented in Table 7 together with those by the NLL and AS methods (with $k^* = 1$ and 2). Besides, we also adopt an NN to train the mapping from $\boldsymbol{x}$ to $f(\boldsymbol{x})$ directly. The associated approximation results are listed in Table 7. For building the NN, we use one FCNN with nine hidden layers and $10d$ (or 200 for $d > 20$) hidden neurons. The NN is trained using the Adam optimizer with the initial learning rate 0.001, decaying every 10000 steps by a factor of 0.7, for 60000 steps in total with no early stop criteria. It is obvious that the DRiLLS method significantly outperforms the other three methods, especially when the input parameter space of the problem has a high dimension.

4. Conclusions. In this paper, an NN-based method, DRiLLS, has been proposed for dimension reduction via learning level sets in function approximation, which performs very well on applications involving high-dimensional limited/sparse data. This model consists of a PRNN module and a synthesized regression module: The former aims to find a handful of active variables that reduce the dimension of original input space and to capture the level set information of the target function, while the latter is designed for effectively approximating function values based on these active variables and neighborhood relationships in the input space. Particularly, the PRNN



(a) 9D (b) 16D



(c) 25D (d) 36D

FIG. 12. *The QoI* (3.6) *approximated by our DRiLLS method with* $k^* = 1$ *for the thermal block problem.*

employs two feed-forward FCNNs to model the nonlinear transformations $\boldsymbol{z} = \boldsymbol{g}(\boldsymbol{x}; \Theta_g)$ and $\hat{\boldsymbol{x}} = \boldsymbol{h}(\boldsymbol{z}; \Theta_h)$, which transform the original data $\boldsymbol{x}$ but do not change its dimension. A new total loss function has been introduced which involves a pseudoreversibility term enforcing $\boldsymbol{h} \circ \boldsymbol{g}$ to be close to the identity mapping, an active direction fitting loss compelling changes in target function value caused by small perturbations of the inactive variables to be tangent to the function's level sets, and a bounded derivative loss regularizing the graph of the target function as the input variables change from $\boldsymbol{x}$ to $\boldsymbol{z}_A$. Once the PRNN is trained, function values are predicted using a synthesized regression that selects neighboring training samples based on the distance information from the original input space and projects them to the space of the active variables to perform local least-squares polynomial fitting.

Some ablation studies are carried out to show the effect of the major components and hyperparameters of our DRiLLS method. Several high-dimensional function approximation examples and a PDE-related application problem are also used to investigate and compare the performance of the proposed method with the popular nonlinear and linear dimension reduction methods NN, NLL, and AS, and experi-

TABLE 7
Numerical approximation errors produced by DRiLLS, NN, NLL, and AS for the QoI (3.6) *in the thermal block problem when the number of input parameters is* 9, 16, 25, *or* 36.

	9D		16D		25D		36D	
	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1	NRMSE	RL_1
DRiLLS ($k^*=1$)	1.14%	1.57%	1.68%	1.81%	2.84%	2.88%	3.20%	2.93%
NN	1.22%	1.83%	2.95%	4.93%	10.01%	17.05%	10.27%	13.85%
NLL ($k^*=1$)	4.87%	13.40%	6.51%	12.57%	7.48%	11.28%	8.20%	10.15%
NLL ($k^*=2$)	4.61%	12.92%	6.77%	13.22%	8.22%	12.25%	8.77%	10.90%
AS ($k^*=1$)	5.00%	13.80%	6.51%	12.65%	7.39%	11.13%	8.07%	10.00%
AS ($k^*=2$)	4.44%	13.00%	6.19%	12.61%	7.42%	11.55%	8.18%	10.35%

mental results demonstrate that the DRiLLS method is superior in many examples of high-dimensional function approximation with limited/sparse data. Note that our method requires gradient information of the target function just as the NLL and AS methods. As a next step, we will explore gradient-free approaches to level set learning and function approximation. We will further combine our method with model order reduction methods to develop efficient intrusive computational surrogates for complex systems with high-dimensional parameters.

REFERENCES

[1] H. ABDI AND L. J. WILLIAMS, *Principal component analysis*, Wiley Interdiscip. Rev. Comput. Stat., 2 (2010), pp. 433–459.

[2] K. P. ADRAGNI AND R. D. COOK, *Sufficient dimension reduction and prediction in regression*, Philos. Trans. Roy. Soc. A, 367 (2009), pp. 4385–4405.

[3] S. C. BRENNER AND L. R. SCOTT, *The Mathematical Theory of Finite Element Methods*, Vol. 3, Springer, New York, 2008.

[4] B. CHANG, L. MENG, E. HABER, L. RUTHOTTO, D. BEGERT, AND E. HOLTHAM, *Reversible architectures for arbitrarily deep residual neural networks*, in Proceedings of the 32nd AAAI Conference on Artificial Intelligence, S. A. McIlraith and K. Q. Weinberger, eds., AAAI Press, Washington, DC, 2018, pp. 2811–2818.

[5] P. G. CONSTANTINE, *Active Subspaces: Emerging Ideas for Dimension Reduction in Parameter Studies*, SIAM, Philadelphia, 2015.

[6] P. G. CONSTANTINE, E. DOW, AND Q. WANG, *Active subspace methods in theory and practice: Applications to kriging surfaces*, SIAM J. Sci. Comput., 36 (2014), pp. A1500–A1524.

[7] R. D. COOK AND L. NI, *Sufficient dimension reduction via inverse regression: A minimum discrepancy approach*, J. Amer. Statist. Assoc., 100 (2005), pp. 410–428.

[8] R. D. COOK AND S. WEISBERG, *Sliced inverse regression for dimension reduction: Comment*, J. Amer. Statist. Assoc., 86 (1991), pp. 328–332.

[9] R. DENNIS COOK, *SAVE: A method for dimension reduction and graphics in regression*, Comm. Statist. Theory Methods, 29 (2000), pp. 2109–2121.

[10] D. L. DONOHO AND C. GRIMES, *Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data*, Proc. Natl. Acad. Sci., 100 (2003), pp. 5591–5596.

[11] A. N. GOMEZ, M. REN, R. URTASUN, AND R. B. GROSSE, *The reversible residual network: Backpropagation without storing activations*, in Advances in Neural Information Processing Systems, Vol. 30, Curran Associates, Inc., 2017, pp. 2211–2221.

[12] I. GOODFELLOW, Y. BENGIO, AND A. COURVILLE, *Deep Learning*, MIT Press, Cambridge, MA, 2016.

[13] A. GRUBER, M. GUNZBURGER, L. JU, Y. TENG, AND Z. WANG, *Nonlinear level set learning for function approximation on sparse data with applications to parametric differential equations*, Numer. Math. Theory Methods Appl., 14 (2021), pp. 839–861.

[14] L. GUO, A. NARAYAN, AND T. ZHOU, *Constructing least-squares polynomial approximations*, SIAM Rev., 62 (2020), pp. 483–508.

[15] B. Haasdonk, *Chapter 2: Reduced basis methods for parametrized PDEs—A tutorial introduction for stationary and instationary problems*, in Model Reduction and Approximation: Theory and Algorithms, SIAM, Philadelphia, 2017.
[16] S. Karumuri, R. Tripathy, I. Bilionis, and J. Panchal, *Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks*, J. Comput. Phys., 404 (2020), 109120.
[17] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in Proceedings of the 3rd International Conference on Learning Representations, 2015.
[18] K.-Y. Lee, B. Li, and F. Chiaromonte, *A general theory for nonlinear sufficient dimension reduction: Formulation and estimation*, Ann. Statist., 41 (2013), pp. 221–249.
[19] B. Li, *Sufficient Dimension Reduction: Methods and Applications with R*, CRC Press, Boca Raton, FL, 2018.
[20] B. Li and J. Song, *Nonlinear sufficient dimension reduction for functional data*, Ann. Statist., 45 (2017), pp. 1059–1095.
[21] K.-C. Li, *Sliced inverse regression for dimension reduction*, J. Amer. Statist. Assoc., 86 (1991), pp. 316–327.
[22] K.-C. Li, *On principal Hessian directions for data visualization and dimension reduction: Another application of Stein's lemma*, J. Amer. Statist. Assoc., 87 (1992), pp. 1025–1039.
[23] J. Nocedal, *Updating quasi-Newton matrices with limited storage*, Math. Comput., 35 (1980), pp. 773–782.
[24] Q. Pan and D. Dias, *Sliced inverse regression-based sparse polynomial chaos expansions for reliability analysis in high dimensions*, Reliab. Eng. Syst. Safety, 167 (2017), pp. 484–493.
[25] A. Pinkus, *Ridge Functions*, Cambridge Tracts in Mathematics, Cambridge University Press, Cambridge, 2015.
[26] F. Romor, M. Tezzele, and G. Rozza, *ATHENA: Advanced techniques for high dimensional parameter spaces to enhance numerical analysis*, Software Impacts, 10 (2021), 100133.
[27] S. T. Roweis and L. K. Saul, *Nonlinear dimensionality reduction by locally linear embedding*, Science, 290 (2000), pp. 2323–2326.
[28] B. Tang, *Orthogonal array-based Latin hypercubes*, J. Amer. Statist. Assoc., 88 (1993), pp. 1392–1397.
[29] J. B. Tenenbaum, V. De Silva, and J. C. Langford, *A global geometric framework for nonlinear dimensionality reduction*, Science, 290 (2000), pp. 2319–2323.
[30] R. K. Tripathy and I. Bilionis, *Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification*, J. Comput. Phys., 375 (2018), pp. 565–588.
[31] H.-M. Wu, *Kernel sliced inverse regression with applications to classification*, J. Comput. Graph. Statist., 17 (2008), pp. 590–610.
[32] Y.-R. Yeh, S.-Y. Huang, and Y.-J. Lee, *Nonlinear dimension reduction with kernel sliced inverse regression*, IEEE Trans. Knowledge Data Eng., 21 (2009), pp. 1590–1603.
[33] G. Zhang, J. Zhang, and J. Hinkle, *Learning nonlinear level sets for dimensionality reduction in function approximation*, in *Advances in Neural Information Processing Systems*, Vol. 32, Curran Associates, Inc., 2019, pp. 13220–13229.