SIAM J. SCI. COMPUT. © 2014 Society for Industrial and Applied Mathematics
Vol. 36, No. 1, pp. A168–A192

LOCALIZED DISCRETE EMPIRICAL INTERPOLATION METHOD[*]

BENJAMIN PEHERSTORFER[†], DANIEL BUTNARU[†], KAREN WILLCOX[‡], AND HANS-JOACHIM BUNGARTZ[†]

Abstract. This paper presents a new approach to construct more efficient reduced-order models for nonlinear partial differential equations with proper orthogonal decomposition and the discrete empirical interpolation method (DEIM). Whereas DEIM projects the nonlinear term onto one global subspace, our localized discrete empirical interpolation method (LDEIM) computes several local subspaces, each tailored to a particular region of characteristic system behavior. Then, depending on the current state of the system, LDEIM selects an appropriate local subspace for the approximation of the nonlinear term. In this way, the dimensions of the local DEIM subspaces, and thus the computational costs, remain low even though the system might exhibit a wide range of behaviors as it passes through different regimes. LDEIM uses machine learning methods in the offline computational phase to discover these regions via clustering. Local DEIM approximations are then computed for each cluster. In the online computational phase, machine-learning-based classification procedures select one of these local subspaces adaptively as the computation proceeds. The classification can be achieved using either the system parameters or a low-dimensional representation of the current state of the system obtained via feature extraction. The LDEIM approach is demonstrated for a reacting flow example of an H_2-Air flame. In this example, where the system state has a strong nonlinear dependence on the parameters, the LDEIM provides speedups of two orders of magnitude over standard DEIM.

Key words. discrete empirical interpolation method, clustering, proper orthogonal decomposition, model reduction, nonlinear partial differential equations

AMS subject classifications. 65M02, 62H30

DOI. 10.1137/130924408

1. Introduction.

A dramatic increase in the complexity of today's models and simulations in computational science and engineering has outpaced advances in computing power, particularly for applications where a large number of model evaluations is required, such as uncertainty quantification, design and optimization, and inverse problems. In many cases, reduced-order models can provide a similar accuracy to high-fidelity models but with orders of magnitude reduction in computational complexity. They achieve this by approximating the solution in a lower-dimensional, problem-dependent subspace. With localization approaches, the dimension of the reduced model, and thus the computational cost of solving it, can be further reduced by constructing not just one but multiple local subspaces and then, depending on the current state of the system, selecting an appropriate local subspace for the approximation. We present a localization approach based on machine learning techniques to approximate nonlinear terms in reduced-order models of nonlinear partial differential equations (PDEs) with the discrete empirical interpolation method (DEIM). This paper proposes a localization approach tailored to the DEIM setting and

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section June 10, 2013; accepted for publication (in revised form) November 4, 2013; published electronically February 11, 2014. This work was supported in part by a grant from the MIT-Germany Seed Fund.
http://www.siam.org/journals/sisc/36-1/92440.html

[†]Department of Informatics, Technische Universität München, 85748 Garching, Germany (pehersto@in.tum.de, butnaru@in.tum.de).

[‡]Department of Aeronautics and Astronautics, MIT, Cambridge, MA 02139 (kwillcox@mit.edu, bungartz@in.tum.de). The third author's work was supported by the Air Force Office of Scientific Research MURI program on Uncertainty Quantification under grant FA9550-09-0613.

addresses the particular questions of how to construct the local subspaces and how to select an appropriate approximation subspace with respect to the current state of the system.

Projection-based model reduction methods reduce the complexity of solving PDEs by employing Galerkin projection of the equations onto the subspace spanned by a set of basis vectors. In addition to truncated balanced realization [28] and Krylov subspace methods [19], a popular method to construct the basis vectors is proper orthogonal decomposition (POD) [34, 7, 31]. For many applications, the dynamics of the system underlying the governing equations can often be represented by a small number of POD modes, leading to significantly reduced computational complexity but maintaining a high level of accuracy when compared to the original high-fidelity model.

With POD, efficient reduced-order models can be constructed for PDEs with affine parameter dependence or low-order polynomial nonlinearities [12]. However, POD-Galerkin poses a challenge if applied to systems with a general nonlinear term, because the cost to evaluate the projected nonlinear function still requires computations that scale with the dimension of the original system. This can lead to reduced-order models with similar computational costs as the original high-fidelity model. A number of solutions to this problem have been proposed. In [32], the nonlinear function is linearized at specific points of a trajectory in the state space and then approximated by threading linear models at those points along the trajectory. Another approach is based on subsampling the nonlinear term in certain components and reconstructing all other components via gappy POD [4]. The Gauss–Newton with approximated tensors (GNAT) method [11] and the empirical interpolation method (EIM) [6, 21] are two other approaches to approximately represent the nonlinear term with sparse sampling. Here, we use the discrete version of EIM, that is, DEIM [12]. EIM and DEIM construct a separate subspace for the approximation of the nonlinear term of the PDE, select interpolation points via a greedy strategy, and then combine interpolation and projection to approximate the nonlinear function in the subspace.

If the system exhibits a wide range of behaviors, many DEIM basis vectors and interpolation points are required to accurately approximate the nonlinear term. Therefore, our localized discrete empirical interpolation method (LDEIM) constructs not just one global DEIM interpolant but rather multiple local DEIM interpolants, each tailored to a particular system behavior. In the context of model reduction, similar concepts have been proposed in different contexts. In [23, 33], reduced-order models based on adaptive reduced bases are discussed. In [22, 14], the parameter and time domains are split recursively into subdomains and for each subdomain a separate reduced-order model, including a separate EIM interpolant of the nonlinear term, is built. In that approach, reduced-order models might be constructed multiple times if the system exhibits similar state solutions at different points in time (which necessarily correspond to different time subdomains). Furthermore, the switch from one reduced model to another requires a projection of the state solution onto the basis of the new reduced model, which might introduce numerical instabilities [25]. A similar approach is followed in [16, 15, 17]. These methods have in common that they recursively split the parameter domain, which might in practice lead to a large number of subdomains because a poor division in the beginning cannot be corrected later on. To avoid these drawbacks, in [1, 36] similar snapshots are grouped into clusters with unsupervised learning methods and for each cluster a local reduced-order model is built. A local reduced-order model is then selected with respect to the current state of the system. This localization approach in [1] is applied to the projection basis for the state (i.e., the POD basis) and also for the approximation of the nonlinear term with the GNAT method [11]. One drawback of this approach is that unsupervised learning methods can encounter difficulties such as unstable clustering behavior for large numbers of clusters if the clustering method and its parameters are not carefully chosen [35, 18, 3]. Furthermore, the given procedure in [1] requires precomputing auxiliary quantities to ensure an online selection procedure that is independent of the dimension of the original system and that scales linearly with the number of clusters; however, the number of the auxiliary quantities, which are computed in the offline phase, scales cubically with the number of clusters.

This work develops a localization approach for the DEIM approximation of a nonlinear reduced model. For many applications, we have observed that it is the growth of the DEIM basis dimension (and the associated growth in the number of interpolation points) that counters the computational efficiency of the POD-DEIM reduced-order model (see, e.g., [38]). By applying localization to the DEIM representation of the nonlinear term, we achieve substantial improvements in computational efficiency for applications that otherwise require a large number of DEIM basis vectors. In [1], it has already been observed that finding a local approximation subspace is an unsupervised learning task. We carry this observation over to DEIM and additionally consider the selection of the local subspaces as a supervised learning task. This allows us to develop our localized DEIM with all the advantages of [1, 36] but with two additional benefits. First, our method can handle large numbers of clusters because rather than directly clustering the high-dimensional snapshots with respect to the Euclidean distance, we instead use a DEIM-based distance measure or feature extraction to cluster in a lower-dimensional representation. Thus, we expect a more stable clustering because we cluster points in low-dimensional subspaces and not in high-dimensional spaces, where clustering with respect to general distance measures becomes difficult [30, 26]. Second, the runtime of the online phase in LDEIM is independent of the number of clusters because we employ nearest neighbor classifiers for the adaptive selection of the local interpolants. In addition, because the DEIM approximation can be fully decoupled from the POD basis, no basis transformation of the state vector is required when we switch from one DEIM interpolant to another.

The remainder of this paper is organized as follows. In section 2 we briefly review POD-DEIM-Galerkin, define our notation, and discuss the limitations of DEIM. Then, in section 3, our localized DEIM is developed in a general setting where we highlight the close relationship to machine learning. We continue with two variants—parameter-based LDEIM and state-based LDEIM—in sections 4 and 5, respectively. Finally, our method is applied to benchmark examples and a reacting flow simulation of an H_2-Air flame in section 6 before conclusions are drawn in section 7.

2. Problem formulation.

Our starting point is the system of nonlinear equations

$$\boldsymbol{A}\boldsymbol{y}(\boldsymbol{\mu}) + \boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu})) = 0 \tag{2.1}$$

stemming from a discretization of a parametrized PDE, where the operators $\boldsymbol{A} \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ and $\boldsymbol{F} : \mathbb{R}^{\mathcal{N}} \to \mathbb{R}^{\mathcal{N}}$ correspond to the linear and the nonlinear term of the PDE, respectively. The solution or state vector $\boldsymbol{y}(\boldsymbol{\mu}) = [y_1(\boldsymbol{\mu}), \dots, y_{\mathcal{N}}(\boldsymbol{\mu})]^T \in \mathbb{R}^{\mathcal{N}}$ for a particular parameter $\boldsymbol{\mu} \in \mathcal{D} \subset \mathbb{R}^d$ is an $\mathcal{N}$-dimensional vector. The components of the nonlinear function $\boldsymbol{F}$ are constructed by the scalar function $F : \mathbb{R} \to \mathbb{R}$ which is evaluated at each component of $\boldsymbol{y}(\boldsymbol{\mu})$, i.e., $\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu})) = [F(y_1(\boldsymbol{\mu})), \dots, F(y_{\mathcal{N}}(\boldsymbol{\mu}))]^T$.

The Jacobian of the system (2.1) is given by

$$\boldsymbol{J}(\boldsymbol{y}(\boldsymbol{\mu})) = \boldsymbol{A} + \boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{y}(\boldsymbol{\mu}))\,, \tag{2.2}$$

where $\boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{y}(\boldsymbol{\mu})) = \operatorname{diag}\{F'(y_1(\boldsymbol{\mu})), \dots, F'(y_{\mathcal{N}}(\boldsymbol{\mu}))\} \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ because $\boldsymbol{F}$ is evaluated at $\boldsymbol{y}(\boldsymbol{\mu})$ componentwise.

2.1. POD. We use POD to compute a reduced basis of dimension $N \ll \mathcal{N}$. We select the sampling points $\mathcal{P} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_m\} \subset \mathcal{D}$ and build the state snapshot matrix $\boldsymbol{Y} = [\boldsymbol{y}(\boldsymbol{\mu}_1), \dots, \boldsymbol{y}(\boldsymbol{\mu}_m)] \in \mathbb{R}^{\mathcal{N}\times m}$ whose ith column contains the solution $\boldsymbol{y}(\boldsymbol{\mu}_i)$ of (2.1) with parameter $\boldsymbol{\mu}_i$. We compute the singular value decomposition of $\boldsymbol{Y}$ and put the first N left singular vectors corresponding to the N largest singular values as the POD basis vectors in the columns of the matrix $\boldsymbol{V}_N = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_N] \in \mathbb{R}^{\mathcal{N}\times N}$. With Galerkin projection, we obtain the reduced-order system of (2.1)

$$\boldsymbol{V}_N^T \boldsymbol{A} \boldsymbol{V}_N \tilde{\boldsymbol{y}}(\boldsymbol{\mu}) + \boldsymbol{V}_N^T \boldsymbol{F}(\boldsymbol{V}_N \tilde{\boldsymbol{y}}(\boldsymbol{\mu})) = 0, \tag{2.3}$$

where $\boldsymbol{V}_N \tilde{\boldsymbol{y}}(\boldsymbol{\mu})$ replaces the state vector $\boldsymbol{y}(\boldsymbol{\mu})$. We call $\tilde{\boldsymbol{y}}(\boldsymbol{\mu}) \in \mathbb{R}^N$ the reduced state vector. The reduced Jacobian is

$$\boldsymbol{J}_N = \boldsymbol{V}_N^T \boldsymbol{A} \boldsymbol{V}_N + \boldsymbol{V}_N^T \boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{V}_N \tilde{\boldsymbol{y}}(\boldsymbol{\mu})) \boldsymbol{V}_N. \tag{2.4}$$

This POD-Galerkin method is usually split into a computationally expensive offline and a rapid online phase. This splitting into offline and online phases works well for the linear operator. In the offline phase, the snapshot matrix $\boldsymbol{Y}$ and the POD basis $\boldsymbol{V}_N$ are computed. The POD basis $\boldsymbol{V}_N$ is then used to construct the reduced operator $\tilde{\boldsymbol{A}} = \boldsymbol{V}_N^T \boldsymbol{A} \boldsymbol{V}_N \in \mathbb{R}^{N\times N}$. In the online phase, only $\tilde{\boldsymbol{A}}$ is required to solve the reduced-order system (2.3). However, this efficient offline/online splitting does not hold for the nonlinear operator $\boldsymbol{F}$, since we cannot precompute a reduced nonlinear operator as we have done with $\boldsymbol{A}$ but instead must evaluate the nonlinear function $\boldsymbol{F}$ at all $\mathcal{N}$ components of $\boldsymbol{V}_N \tilde{\boldsymbol{y}}(\boldsymbol{\mu})$ in the online phase. If $\mathcal{N}$ is large and the evaluation of $\boldsymbol{F}$ expensive, the time to evaluate the nonlinear term in the reduced system (2.3) will dominate the overall solution time and supersede the savings obtained for the linear term through the POD-Galerkin method.

2.2. DEIM. One way to overcome this weakness of the POD-Galerkin method is with DEIM [12]. It approximates the nonlinear function $\boldsymbol{F}$ on a linear subspace spanned by basis vectors $\boldsymbol{U} = [\boldsymbol{u}_1, \dots, \boldsymbol{u}_M] \in \mathbb{R}^{\mathcal{N}\times M}$ that are obtained by applying POD to snapshots $\mathcal{S} = \{\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}_1)), \dots, \boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}_m))\}$ of the nonlinear term. The system $\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu})) \approx \boldsymbol{U}\boldsymbol{\alpha}(\boldsymbol{\mu})$ to obtain the coefficients $\boldsymbol{\alpha}(\boldsymbol{\mu}) \in \mathbb{R}^M$ is overdetermined. Thus, DEIM selects only M rows of $\boldsymbol{U}$ to compute the coefficients $\boldsymbol{\alpha}(\boldsymbol{\mu})$. Formally, a matrix $\boldsymbol{P} = [\boldsymbol{e}_{p_1}, \dots, \boldsymbol{e}_{p_M}] \in \mathbb{R}^{\mathcal{N}\times M}$ is introduced, where $\boldsymbol{e}_i$ is the ith column of the identity matrix. The DEIM interpolation points $p_1, \dots, p_M$ are selected with a greedy algorithm. Assuming $\boldsymbol{P}^T\boldsymbol{U}$ is nonsingular, the coefficients $\boldsymbol{\alpha}(\boldsymbol{\mu})$ can be determined from $\boldsymbol{P}^T\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu})) = (\boldsymbol{P}^T\boldsymbol{U})\boldsymbol{\alpha}(\boldsymbol{\mu})$ and we obtain

$$\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu})) \approx \boldsymbol{U}(\boldsymbol{P}^T\boldsymbol{U})^{-1}\boldsymbol{P}^T\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu})). \tag{2.5}$$

In the following, we denote a DEIM approximation or DEIM interpolant with the tuple $(\boldsymbol{U}, \boldsymbol{P})$. We obtain the POD-DEIM-Galerkin reduced-order system

$$\underbrace{\boldsymbol{V}_N^T \boldsymbol{A} \boldsymbol{V}_N}_{N\times N} \tilde{\boldsymbol{y}}(\boldsymbol{\mu}) + \underbrace{\boldsymbol{V}_N^T \boldsymbol{U}(\boldsymbol{P}^T\boldsymbol{U})^{-1}}_{N\times M} \boldsymbol{P}^T \boldsymbol{F}(\boldsymbol{V}_N \tilde{\boldsymbol{y}}(\boldsymbol{\mu})) = 0 \tag{2.6}$$

and the reduced Jacobian

$$\tilde{\boldsymbol{J}}_{\boldsymbol{F}}(\tilde{\boldsymbol{y}}(\boldsymbol{\mu})) = \underbrace{\boldsymbol{V}_N^T \boldsymbol{A} \boldsymbol{V}_N}_{N\times N} + \underbrace{\boldsymbol{V}_N^T \boldsymbol{U}(\boldsymbol{P}^T\boldsymbol{U})^{-1}}_{N\times M} \underbrace{\boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{P}^T \boldsymbol{V}_N \tilde{\boldsymbol{y}}(\boldsymbol{\mu}))}_{M\times M} \underbrace{\boldsymbol{P}^T \boldsymbol{V}_N}_{M\times N}. \tag{2.7}$$

We see in (2.6) and (2.7) that with the POD-DEIM-Galerkin method we evaluate the nonlinear term $\boldsymbol{F}$ only at M instead of at $\mathcal{N}$ points. Similarly to $N \ll \mathcal{N}$ for the POD basis, we assume $M \ll \mathcal{N}$ for DEIM and thus expect savings in computational costs. We refer to [12] for more details on DEIM in general and the greedy algorithm to select the interpolation points in particular.

2.3. Limitations of DEIM. Whether DEIM does indeed lead to savings in the runtime depends on the number M of basis vectors and interpolation points. The DEIM approximates the nonlinear term $\boldsymbol{F}$ in the subspace of dimension M spanned by $\boldsymbol{U}$. For some nonlinear systems, a large number M of DEIM basis vectors are required to accurately represent $\boldsymbol{F}$ over the range of situations of interest. We demonstrate this on a simple interpolation example as follows.

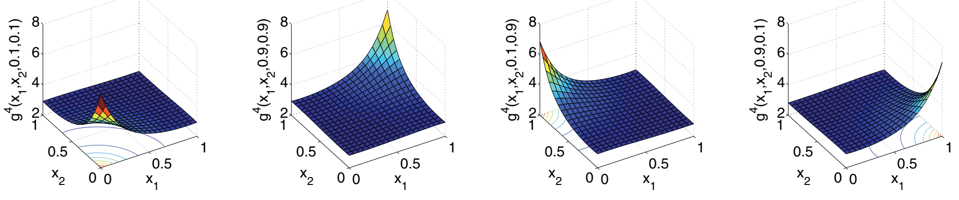
Consider the spatial domain $\Omega = [0,1]^2$ and the parameter domain $\mathcal{D} = [0,1]^2$. We define the function $g^1 : \Omega \times \mathcal{D} \to \mathbb{R}$ with

$$g^1(\boldsymbol{x};\boldsymbol{\mu}) = \frac{1}{\sqrt{((1-x_1)-(0.99\cdot\mu_1-1))^2 + ((1-x_2)-(0.99\cdot\mu_2-1))^2 + 0.1^2}}. \tag{2.8}$$

The parameter $\boldsymbol{\mu} = (\mu_1, \mu_2)$ controls the gradient of the peak near the corner $(1,1)$ of the spatial domain $[0,1]^2$. Based on g^1, let us consider the function

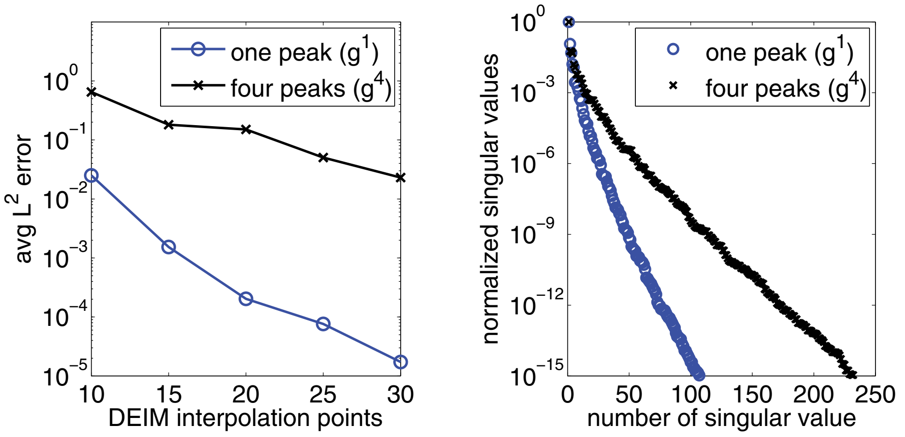
$$\begin{aligned} g^4(\boldsymbol{x};\boldsymbol{\mu}) = {} & g^1(\boldsymbol{x};\boldsymbol{\mu}) + g^1(1-x_1, 1-x_2; 1-\mu_1, 1-\mu_2) \\ & + g^1(1-x_1, x_2; 1-\mu_1, \mu_2) + g^1(x_1, 1-x_2; \mu_1, 1-\mu_2). \end{aligned} \tag{2.9}$$

Depending on the parameter $\boldsymbol{\mu}$, the function g^4 has a sharp peak in one of the four corners of Ω; see Figure 2.1. We discretize the functions g^1 and g^4 on a 20×20 equidistant grid in Ω and randomly sample on a 25×25 equidistant grid in $\mathcal{D}$. From



(a) $\boldsymbol{\mu} = (0.1, 0.1)$ (b) $\boldsymbol{\mu} = (0.9, 0.9)$ (c) $\boldsymbol{\mu} = (0.1, 0.9)$ (d) $\boldsymbol{\mu} = (0.9, 0.1)$

FIG. 2.1. *Depending on the parameter $\boldsymbol{\mu}$, the function g^4 has a sharp peak in one of the four corners of the spatial domain.*



(a) error (b) singular values

Fig. 2.2. *In* (a) *the averaged L^2 error versus the number M of DEIM interpolation points corresponding to the function with one (g^1) and four (g^4) peaks, respectively. The more peaks, the worse the result. This is reflected in the decay of the singular values shown in* (b).

the 625 snapshots, we build the DEIM approximations. Figure 2.2 shows the averaged L^2 error of the approximations over a set of test samples $\{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_{121}\}$ that correspond to an 11×11 equidistant grid in $\mathcal{D}$. The results for g^4 are worse than for g^1. This is reflected in the slower decay of the singular values of the snapshot matrix corresponding to g^4. Even though g^4 is just a sum of g^1 functions, and, depending on the parameter $\boldsymbol{\mu}$, only one of the summands determines the behavior of g^4, we still obtain worse results than for g^1. The reason is that the DEIM basis must represent all four peaks. It cannot focus on only one (local) peak as is possible when we consider one g^1 function only. This also means that if we choose the parameter $\boldsymbol{\mu} = (0.1, 0.9)$ which leads to only one sharp peak of g^4 near the corner $(0.1, 0.9)$ of Ω, just a few DEIM basis vectors are relevant for the approximation and all others are ignored. This is a clear waste of resources and motivates our development of LDEIM.

3. Localized discrete empirical interpolation method.

DEIM approximates the nonlinear term $\boldsymbol{F}$ on a single linear subspace which must represent $\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}))$ well for all $\boldsymbol{\mu} \in \mathcal{D}$. Instead, we propose to compute several local DEIM approximations, which are each adapted to a small subset of parameters, or which are each fitted to a particular region in state space. We refer to this approach as the localized discrete empirical interpolation method (LDEIM). In this section, we first introduce the general idea of LDEIM and then propose two specific LDEIM variants. We start by discussing the computational procedure of LDEIM in more detail. LDEIM constructs several local DEIM approximations $(\boldsymbol{U}_1, \boldsymbol{P}_1), \dots, (\boldsymbol{U}_k, \boldsymbol{P}_k)$ offline and then selects one of them online. We will see that the two building blocks of LDEIM are the corresponding construction and selection procedures, for which machine learning methods play a crucial role. In this section, we also discuss the error and the computational costs of LDEIM approximations. Finally, two specific LDEIM variants—parameter-based LDEIM and state-based LDEIM—are introduced and the differences between them are discussed.

3.1. LDEIM.

Let $\mathcal{S} = \{\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}_1)), \dots, \boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}_m))\}$ be the set of nonlinear snapshots. In the offline phase of LDEIM, we group similar snapshots together and obtain a partition $\mathcal{S}_1 \uplus \cdots \uplus \mathcal{S}_k$ of $\mathcal{S}$ with k subsets. Here, snapshots are considered to be similar and should be grouped together if they can be approximated well with the same set of DEIM basis vectors and interpolation points. For each of these subsets, we construct a local DEIM approximation with the classical DEIM procedure. Thus, for each set $\mathcal{S}_i$, we obtain a $(\boldsymbol{U}_i, \boldsymbol{P}_i)$ where the basis and the interpolation points are adapted to the snapshots in $\mathcal{S}_i$ only. Furthermore, also in the offline phase, we compute the so-called classifier $c : \mathcal{Z} \to \{1, \dots, k\}$. Its purpose is to select a good local DEIM approximation $(\boldsymbol{U}_i, \boldsymbol{P}_i)$ with respect to an indicator $\boldsymbol{z} \in \mathcal{Z}$. The indicator $\boldsymbol{z}$ must describe $\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}))$ well enough to decide which local DEIM approximation should be used. The classifier is trained on the indicators of the nonlinear snapshots. Many different indicators are possible. For example, a simple indicator is the parameter $\boldsymbol{\mu}$. However, this is not the only possibility and is not always a good choice. We defer the discussion of specific indicators to section 3.4, where we then also specify the domain $\mathcal{Z}$. The output of the offline phase contains the local DEIM approximations $(\boldsymbol{U}_1, \boldsymbol{P}_1), \dots, (\boldsymbol{U}_k, \boldsymbol{P}_k)$ and the classifier $c : \mathcal{Z} \to \{1, \dots, k\}$. In the online phase, we compute $\boldsymbol{z}$, evaluate the classifier, and, depending on its value, switch between the local DEIM approximations. This workflow is sketched in Figure 3.1.

With LDEIM, the Galerkin reduced-order system takes the shape

$$\boldsymbol{V}_N^T \boldsymbol{A} \boldsymbol{V}_N \tilde{\boldsymbol{y}}(\boldsymbol{\mu}) + \boldsymbol{V}_N^T \boldsymbol{U}_{c(\cdot)} (\boldsymbol{P}_{c(\cdot)}^T \boldsymbol{U}_{c(\cdot)})^{-1} \boldsymbol{P}_{c(\cdot)}^T \boldsymbol{F}(\boldsymbol{V}_N \tilde{\boldsymbol{y}}(\boldsymbol{\mu})) = 0\,, \tag{3.1}$$

instead of (2.6), and the reduced Jacobian is

$$\tilde{\boldsymbol{J}}_{\boldsymbol{F}}(\tilde{\boldsymbol{y}}(\boldsymbol{\mu})) = \boldsymbol{V}_N^T \boldsymbol{A} \boldsymbol{V}_N + \boldsymbol{V}_N^T \boldsymbol{U}_{c(\cdot)} (\boldsymbol{P}_{c(\cdot)}^T \boldsymbol{U}_{c(\cdot)})^{-1} \boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{P}_{c(\cdot)}^T \boldsymbol{V}_N \tilde{\boldsymbol{y}}(\boldsymbol{\mu})) \boldsymbol{P}_{c(\cdot)}^T \boldsymbol{V}_N\,, \tag{3.2}$$

instead of (2.7). The DEIM basis $\boldsymbol{U}$ and the matrix $\boldsymbol{P}$ depend through the classifier c on the indicator $\boldsymbol{z}$ and thus on the nonlinear term $\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}))$ evaluated at $\boldsymbol{y}(\boldsymbol{\mu})$. Instead of one $\boldsymbol{V}_N^T \boldsymbol{U}(\boldsymbol{P}^T \boldsymbol{U})^{-1}$ for the nonlinear term $\boldsymbol{F}$, we now precompute k matrices

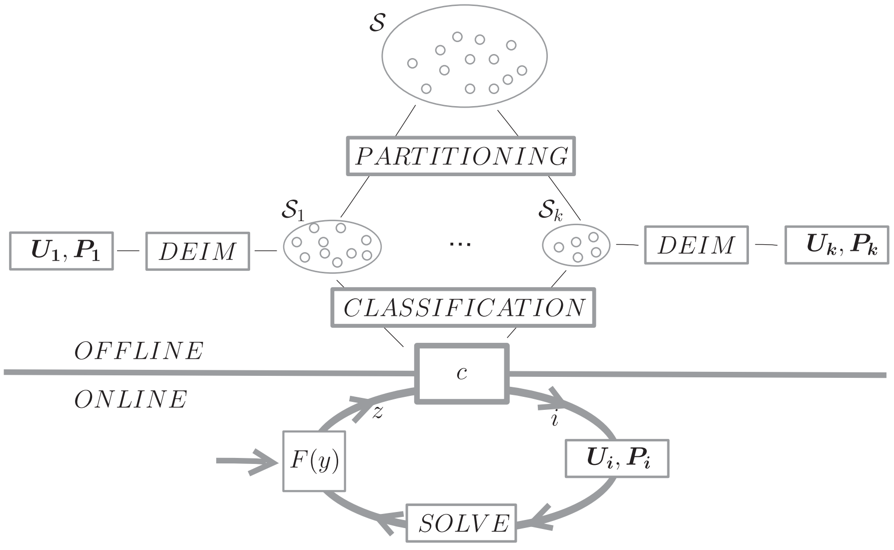


FIG. 3.1. *LDEIM workflow. Offline, a partitioning method splits the snapshots $\mathcal{S}$ into several groups $\mathcal{S}_i$. For each group a separate DEIM approximation is computed and stored. A classifier c learns the mapping between the indicators $\boldsymbol{z}$ and the particular group i to which it has been assigned. Online, the classifier c selects a local DEIM approximation $(\boldsymbol{U}_i, \boldsymbol{P}_i)$ with respect to the indicator $\boldsymbol{z}$.*

$$\boldsymbol{V}_N^T \boldsymbol{U}_i (\boldsymbol{P}_i^T \boldsymbol{U}_i)^{-1}\,, \qquad i = 1, \dots, k, \tag{3.3}$$

from which one is picked according to c in the online phase. It is important to note that DEIM can be fully decoupled from the POD projection of the state. Thus, when we change the DEIM basis, we do not influence the POD basis $\boldsymbol{V}_N$.

3.2. Learning local bases. The offline phase of LDEIM consists of two steps. First, it groups similar snapshots together to obtain a partition of the set $\mathcal{S}$ for which the local DEIM interpolants are constructed, and second, it computes the classifier $c : \mathcal{Z} \to \{1, \dots, k\}$ to later select an appropriate local interpolant. These are both machine learning tasks.

We first consider the partitioning of $\mathcal{S}$. In terms of machine learning this means we want to cluster the data points in $\mathcal{S}$ with respect to a certain clustering criterion [8]. The input to the clustering method is the set $\mathcal{S}$, and the output is a partition $\mathcal{S}_1 \uplus \cdots \uplus \mathcal{S}_k$ in k subsets or clusters. In the following, we use k-means clustering (Lloyd's algorithm), along with other partitioning approaches. The k-means algorithm is a standard clustering algorithm that has been applied successfully to many different applications. Usually, we determine the number of clusters k in advance. Many rules of thumb, as well as sophisticated statistical methods, are available to estimate the number of clusters from the data points. We refer to [20] for an overview and to [10] for an approach well suited for k-means.

Having obtained a partition of $\mathcal{S}$, we compute the function $c : \mathcal{Z} \to \{1, \dots, k\}$. In machine learning, this task is called classification [8]. Inputs are the partition of $\mathcal{S}$ and the indicators $\{\boldsymbol{z}_1, \dots, \boldsymbol{z}_m\}$ corresponding to the nonlinear snapshots $\{\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}_1)), \dots, \boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}_m))\}$. The result of the classification is the classifier $c : \mathcal{Z} \to \{1, \dots, k\}$. Classification is a ubiquitous task in data mining with many classification methods available. Here we employ nearest neighbor classifiers. They can track curved classification boundaries and are cheap to evaluate if the number of neighbors is kept low. Low computational cost is crucial here, since we evaluate the classifier during the online phase.

In principle, any clustering and classification methods can be employed for LDEIM. Besides the advantages discussed above, we use k-means and nearest neighbor classification because they are widely used and readily available in many software libraries. Note, however, that even though k-means and nearest neighbor classification are the core of the following clustering and classification methods, a thorough preprocessing of the data and some modifications are necessary to achieve a stable clustering and a reliable selection procedure. More details will follow in sections 4 and 5.

3.3. Error and computational costs of LDEIM approximation. The error estimates associated with DEIM (e.g., [12, 37, 13]) can be carried over to LDEIM. Even though we choose between multiple local DEIM approximations, the eventual approximation itself is just a classical DEIM approximation where these error estimates hold.

In the offline phase of LDEIM, we incur additional computational costs to perform the clustering of $\mathcal{S}$ and the classification to obtain c. In addition, we precompute several matrices (3.3) instead of only one as in the case of DEIM. This leads to an overall increase in the offline costs, although for large-scale problems this increase is small compared to the dominating cost of simulation to obtain the snapshots (which remains the same between DEIM and LDEIM). Note that, even though not considered further here, the local DEIM interpolants are fully decoupled from each other and thus can easily be computed in parallel in the offline phase. The online phase of LDEIM is very similar to that of DEIM. The only additional cost incurred is that to evaluate the classifier c. As discussed previously, we employ a nearest neighbor classifier, the cost of which is small. (In particular, the cost does not depend on the number m of nonlinear snapshots in $\mathcal{S}$ or on the number of clusters k.) Evaluation of the classifier also requires computing the indicator z for the current $\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}))$. We show in sections 4 and 5 that for two specific indicators, the costs to obtain z are low. From these observations, we can summarize that using LDEIM is advantageous for applications for which one can tolerate an increase in offline costs that is outweighed by an online cost benefit due to the reduction in dimension of the DEIM basis. We also note that we refrain from performing online basis updates to increase the accuracy as proposed in, e.g., [36], because this would greatly increase the costs of the online phase.

3.4. Parameter-based LDEIM and state-based LDEIM. So far, we have discussed LDEIM in a very general setting. In particular, we have not specified the indicator $\boldsymbol{z}$ of $\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}))$, i.e., we have not specified the domain $\mathcal{Z}$ of the classifier $c : \mathcal{Z} \to \{1, \dots, k\}$. The indicator $\boldsymbol{z}$ must contain enough information to select a good local DEIM approximation. In the following, we introduce two specific indicators leading to two LDEIM variants—parameter-based and state-based LDEIM.

In the parameter-based LDEIM, the indicator $\boldsymbol{z}$ of $\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}))$ is the parameter $\boldsymbol{\mu} \in \mathcal{D}$. The domain $\mathcal{Z}$ becomes the parameter domain $\mathcal{D}$ and we obtain the classifier $c : \mathcal{D} \to \{1, \dots, k\}$. Thus, we decide with respect to the parameter $\boldsymbol{\mu}$ which local DEIM approximation is used. The parameter-based LDEIM is closely related to other localization approaches in model order reduction [16, 15, 17, 14]. It will be discussed in detail in section 4. In contrast, the state-based LDEIM computes a low-dimensional representation of $\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}))$ with feature selection and feature extraction. Thus, the indicator $\boldsymbol{z}$ is directly derived from $\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}))$, i.e., from the nonlinear term $\boldsymbol{F}$ evaluated at $\boldsymbol{y}(\boldsymbol{\mu})$, and not from the parameter $\boldsymbol{\mu}$. The domain $\mathcal{Z}$ of the classifier $c : \mathcal{Z} \to \{1, \dots, k\}$ becomes a low-dimensional subspace of $\mathbb{R}^{\mathcal{N}}$. The details follow in section 5. Note that we can distinguish between parameter-based and state-based LDEIM by considering the domain $\mathcal{Z}$ of the classifier c.

We summarize that parameter-based LDEIM requires the parameter to decide which local DEIM interpolant to select, whereas state-based LDEIM solely relies on a low-dimensional representation of the state.

We emphasize the difference between the parameter-based and state-based LDEIM by considering the Newton method to solve our nonlinear system. In case of the parameter-based LDEIM, we pick a local DEIM interpolant $(\boldsymbol{U}_i, \boldsymbol{P}_i)$ depending on the parameter $\boldsymbol{\mu}$ before we start with the first Newton iteration. Since the indicator $\boldsymbol{z} = \boldsymbol{\mu}$ does not change during the Newton iterations, we always keep $(\boldsymbol{U}_i, \boldsymbol{P}_i)$ and never switch between the local DEIM approximations. However, for the case of the state-based LDEIM, the indicator $\boldsymbol{z}$ is directly derived from the nonlinear term independent from the parameter $\boldsymbol{\mu}$. Therefore, if we evaluate the classifier c after each Newton iteration, we might get different local DEIM approximations in each iteration because the (reduced) state vector $\tilde{\boldsymbol{y}}(\boldsymbol{\mu})$ and thus the nonlinear term $\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}(\boldsymbol{\mu}))$ change. This means that the state-based LDEIM can switch between different DEIM approximations even within the Newton method, whereas the parameter-based LDEIM keeps one local DEIM interpolant fixed until convergence. In the following two sections, we present the parameter-based and state-based LDEIM in detail.

4. Parameter-based LDEIM.

In this section, we consider the parameter-based LDEIM where the classifier is $c : \mathcal{D} \to \{1, \dots, k\}$ with domain $\mathcal{D}$. The parameter-based LDEIM is motivated by the relationship $\boldsymbol{\mu} \to \boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}))$ showing that the value of the nonlinear function $\boldsymbol{F}$ is influenced by the parameter $\boldsymbol{\mu}$ through the state vector $\boldsymbol{y}(\boldsymbol{\mu})$. Therefore, the parameter $\boldsymbol{\mu}$ may be a good indicator for the behavior of the function $\boldsymbol{F}$. In the previous section, we introduced the partitioning of the set $\mathcal{S}$ and the selection of a local DEIM approximation as two building blocks of LDEIM. As partitioning approaches, we now present a splitting and a clustering method, which are especially well suited for the parameter-based LDEIM.

4.1. Splitting of the parameter domain.

Each snapshot in $\mathcal{S}$ is associated to one parameter $\boldsymbol{\mu}$ in the parameter domain $\mathcal{D}$. These parameters are collected in the set $\mathcal{P} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_m\} \subset \mathcal{D}$. If we split the parameter domain $\mathcal{D}$ into k subdomains, we can derive the corresponding partition of $\mathcal{P}$ and thus of the set of snapshots $\mathcal{S}$. Hence, we have divided our snapshots into k groups (or clusters). Note that similar procedures have previously been used in the context of model reduction; see, e.g., [22, 14].

Consider the parameter domain $\mathcal{D} = [a, b]^d$. Let $\epsilon > 0$ be a tolerance value and M the number of basis vectors and interpolation points of a local DEIM approximation. The parameter domain $\mathcal{D}$ is split into subdomains $\mathcal{D}_1, \dots, \mathcal{D}_k$ in a recursive fashion. We start with a DEIM interpolant $(\boldsymbol{U}, \boldsymbol{P})$, constructed from $\mathcal{S}$, and split the parameter domain $\mathcal{D}$ into 2^d subdomains $\mathcal{D}_1, \dots, \mathcal{D}_{2^d}$ of equal size if the DEIM residual

$$\max_{\boldsymbol{w} \in S} \left\{ \| \boldsymbol{U} (\boldsymbol{P}^T \boldsymbol{U})^{-1} \boldsymbol{P}^T \boldsymbol{w} - \boldsymbol{w} \|_2 \right\} \tag{4.1}$$

is greater than ϵ. Then the corresponding subsets $\mathcal{P} = \mathcal{P}_1 \uplus \cdots \uplus \mathcal{P}_{2^d}$ and $\mathcal{S} = \mathcal{S}_1 \uplus \cdots \uplus \mathcal{S}_{2^d}$ are constructed and a $(\boldsymbol{U}_i, \boldsymbol{P}_i)$ is built for each S_i. We continue this splitting process in each subdomain $\mathcal{D}_i$ as long as the DEIM residual (4.1) of $\mathcal{S}_i$ with $(\boldsymbol{U}_i, \boldsymbol{P}_i)$ is above the tolerance and there are more than M snapshots left in the current set $\mathcal{S}_i$. The result is the partition of $\mathcal{S}$ and $\mathcal{D}$ with the corresponding k local DEIM approximations $(\boldsymbol{U}_1, \boldsymbol{P}_1), \dots, (\boldsymbol{U}_k, \boldsymbol{P}_k)$. The algorithm is shown in Algorithm 1.

It is not necessary to choose the number of subdomains k in advance because the number k is influenced by the tolerance ϵ. During the splitting process, we compute $\mathcal{O}(k \log(k))$ local DEIM approximations in the offline phase. The classifier $c : \mathcal{D} \to \{1, \dots, k\}$ can be easily evaluated at $\boldsymbol{\mu}$ by storing the centers of the subdomains $\mathcal{D}_1, \dots, \mathcal{D}_k$ and comparing them to the parameter $\boldsymbol{\mu}$. One evaluation of c is in $\mathcal{O}(k)$. Since $k < 100$ in all the following examples, the additional costs incurred by the classification of $\boldsymbol{\mu}$ are usually low.

4.2. Clustering of snapshots.

The splitting-based partitioner constructs the groups $\mathcal{S}_i$ based on the DEIM residual (4.1) as a property of the entire group of snapshots. If the group residual is below a certain threshold, then the identified group (subdomain) is accepted and no further splitting takes place.

Another way to construct groups is to freely group the snapshots $\mathcal{S}$ into clusters using k-means. The assignment of a snapshot to a cluster i is based on the individual property of a snapshot in $\mathcal{S}$ that its DEIM approximation within the ith group is the best among the rest of the clusters (groups). Whereas splitting can put two snapshots into one subset (cluster) only if their parameters lie near each other in $\mathcal{D}$ with respect to the Euclidean norm, clustering with k-means can assign two snapshots to one cluster even though their parameters might be very different. This is a more flexible way to derive a partition of $\mathcal{S}$.

ALGORITHM 1. Splitting of the parameter domain $\mathcal{D}$.

```
1: procedure $\mathcal{D}$-SPLITTING($\epsilon$, $M$, $\mathcal{D}$, $\mathcal{S}$)
2:     $(\boldsymbol{U}, \boldsymbol{P}) \leftarrow \mathrm{DEIM}(\mathcal{S}, M)$
3:     $r \leftarrow \max_{\boldsymbol{w} \in \mathcal{S}} \left\{ \|\boldsymbol{U}(\boldsymbol{P}^T\boldsymbol{U})^{-1}\boldsymbol{P}^T\boldsymbol{w} - \boldsymbol{w}\|_2 \right\}$
4:     $\ell \leftarrow$ empty list
5:     if $r > \epsilon$ and $|\mathcal{S}| > M$ then
6:         partition $\mathcal{D}$ into $2^d$ subdomains (squares) $\mathcal{D}_1, \dots, \mathcal{D}_{2^d}$
7:         partition $\mathcal{S}$ into $2^d$ subsets $\mathcal{S}_1, \dots, \mathcal{S}_{2^d}$
8:         for $(\tilde{\mathcal{D}}, \tilde{\mathcal{S}})$ in $\{(\mathcal{D}_i, \mathcal{S}_i) | i = 1, \dots, 2^d\}$ do
9:             $\ell_i \leftarrow \mathcal{D}$-SPLITTING($\epsilon$, $M$, $\tilde{\mathcal{D}}$, $\tilde{\mathcal{S}}$)
10:            $\ell \leftarrow$ concatenate lists $\ell$ and $\ell_i$
11:        end for
12:    else
13:        $\ell \leftarrow$ append $(\mathcal{D}, \mathcal{S}, \boldsymbol{U}, \boldsymbol{P})$ to list $\ell$
14:    end if
15: return $\ell$
16: end procedure
```

In addition to the number of clusters k, we must define three things before we cluster with k-means. First, we define the data points that we want to cluster. In our case, these are the snapshots in $\mathcal{S}$. Second, we define the centroid of a cluster. Here, the centroid of a cluster is its (local) DEIM approximation. Third, we need a clustering criterion. In k-means, the clustering criterion is evaluated for each data point with each cluster centroid, to decide to which cluster the data point should be assigned. By choosing the local DEIM approximations as the centroids, we assign a data point to the cluster where the corresponding DEIM residual is smallest. The motivation for this clustering criterion is the greedy procedure where the DEIM residual is used to select the DEIM interpolation points [12].

Initially, all snapshots are randomly assigned to a start cluster. This leads to a partition of $\mathcal{S}$ into $\mathcal{S}_1 \uplus \dots \uplus \mathcal{S}_k$ and the associated partition of $\mathcal{P}$ into $\mathcal{P}_1 \uplus \dots \uplus \mathcal{P}_k$. With the local DEIM approximations computed from a given clustering, a k-means iteration reassigns the snapshots to new clusters according to the DEIM residual. After several iterations, the k-means algorithm is stopped if no swapping takes place or a maximum number of iterations has been reached.

Though k-means is widely used in many applications, it has the drawback that it only finds a local optimum to the minimization problem underlying its clustering idea [24]. Thus, the solution (clustering) depends on the initial cluster assignment or seed with which k-means is started. There exist many strategies for the seed of k-means but either they scale poorly with the number of data points or they are mere rules of thumb [3, 5, 2, 27]. For this reason we use a random initial cluster assignment. However, this random initial guess might fit poorly with the data and thus the clustering result might not group the snapshots in $\mathcal{S}$ as desired. To ensure a good clustering, we perform several k-means replicates and select the cluster assignment with the minimal within-cluster sum. This is a standard way to cope with randomized initial guesses in k-means.

The result of the clustering method is a partition $\mathcal{S} = \mathcal{S}_1 \uplus \dots \uplus \mathcal{S}_k$ and $\mathcal{P} = \mathcal{P}_1 \uplus \dots \uplus \mathcal{P}_k$. This gives rise to the training data set

Algorithm 2. Construction procedure of parameter-based LDEIM.

```
1: procedure $\mathcal{D}$-Clustering($\mathcal{S}$, $k$)
2:     $\{\mathcal{P}_i, \mathcal{S}_i\}_{i=1}^k \leftarrow$ initial random partitioning of $\mathcal{S}$ and corresponding $\mathcal{P}$
3:     $clustering \leftarrow$ zeros-filled array of size $|\mathcal{S}|$
4:     for $maxIter$ do
5:         $\{\mathbf{U}_i, \mathbf{P}_i\}_{i=1}^k \leftarrow \{\text{DEIM}(\mathcal{S}_i)\}_{i=1}^k$
6:         for $j = 1$ to $|\mathcal{S}|$ do
7:             $clustering[j] \leftarrow \underset{i=1,\dots,k}{\arg\min} \|\mathbf{U}_i(\mathbf{P}_i^T\mathbf{U}_i)^{-1}\mathbf{P}_i^T\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}_j)) - \boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}_j))\|_2$
8:         end for
9:         $\{\mathcal{P}_i, \mathcal{S}_i\}_{i=1}^k \leftarrow$ new partitioning based on updated $clustering$
10:    end for
11:    $c \leftarrow$ train classifier on $\mathcal{P}_1 \times \{1\} \cup \cdots \cup \mathcal{P}_k \times \{k\}$
12: return $(c, \{\mathbf{U}_i, \mathbf{P}_i\}_{i=1}^k)$
13: end procedure
```

$$\mathcal{P}_1 \times \{1\} \cup \cdots \cup \mathcal{P}_k \times \{k\} \subset \mathcal{D} \times \{1, \dots, k\} \tag{4.2}$$

for the classifier $c : \mathcal{D} \to \{1, \dots, k\}$. It is unreasonable to simply compare a parameter $\boldsymbol{\mu}$ with the centers of the clusters $\mathcal{P}_1, \dots, \mathcal{P}_k$ as we have done in the previous section because the clusters most probably do not have a rectangular shape. Therefore, we employ a nearest neighbor classifier.

5. State-based LDEIM.

In the parameter-based LDEIM the indicator $\boldsymbol{z}$ is the parameter $\boldsymbol{\mu}$ and thus a local DEIM interpolant is selected with respect to the parameter of the system. Here, we introduce state-based LDEIM where the indicator $\boldsymbol{z}$ is directly derived from the nonlinear term $\boldsymbol{F}$ evaluated at $\boldsymbol{y}(\boldsymbol{\mu})$ or, more precisely, evaluated at the state $\boldsymbol{y}(\boldsymbol{\mu})$ of the previous iteration or time step. With this approach it is possible to switch the local DEIM interpolant whenever a change in the system occurs even when the parameter $\boldsymbol{\mu}$ does not change (e.g., in every time step or in every Newton iteration as discussed in section 3). The state-based LDEIM is appropriate for time-dependent problems and for nonlinear problems where several iterations are necessary to obtain the output of interest (e.g., using a Newton method).

In the following, we first present an efficient way to compute the indicator $\boldsymbol{z}$ directly from $\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}))$, and then discuss a clustering and classification method that can deal with such indicators. We develop the approach in the context of the Newton method, although there are many other situations where state-based LDEIM is applicable. We denote with $\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu})$ the reduced state vector after the jth Newton iteration.

5.1. Low-dimensional representations via feature extraction.

In principle, we could train a classifier $c : \mathbb{R}^{\mathcal{N}} \to \{1, \dots, k\}$ with domain $\mathbb{R}^{\mathcal{N}}$ directly on the partition $\mathcal{S} = \mathcal{S}_1 \uplus \cdots \uplus \mathcal{S}_k$ obtained in the offline phase and then evaluate c at $\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ after the jth Newton iteration to get the index of the local DEIM interpolant for the $j+1$th iteration. However, this would require us to evaluate $\boldsymbol{F}$ at all $\mathcal{N}$ components of $\boldsymbol{V}_N\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu})$. We cannot afford this during the online phase. Therefore, to obtain a more cost-efficient indicator, we construct a map $\tilde{\boldsymbol{F}} : \mathbb{R}^N \to \mathbb{R}^{\tilde{M}}$, with $\tilde{M} \ll \mathcal{N}$, that will replace $\boldsymbol{F}$ in the evaluation of the indicator. The vector $\boldsymbol{z} = \tilde{\boldsymbol{F}}(\tilde{\boldsymbol{y}}(\boldsymbol{\mu}))$ becomes the indicator of $\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}(\boldsymbol{\mu}))$. To construct the classifier $c : \mathbb{R}^{\tilde{M}} \to \{1, \dots, k\}$, we compute the indicators $\tilde{\mathcal{S}} = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_m\} = \{\tilde{\boldsymbol{F}}(\boldsymbol{V}_N^T\boldsymbol{y}(\boldsymbol{\mu}_1)), \dots,$

$\tilde{\boldsymbol{F}}(\boldsymbol{V}_N^T\boldsymbol{y}(\boldsymbol{\mu}_m))\}$ of the snapshots in $\mathcal{S}$ and then train c on the indicators $\tilde{\mathcal{S}}$ with respect to the partition $\mathcal{S} = \mathcal{S}_1 \uplus \cdots \uplus \mathcal{S}_k$. We compute $\boldsymbol{z}^j = \tilde{\boldsymbol{F}}(\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ in the jth Newton iteration and evaluate the classifier at $\boldsymbol{z}^j$ to get the index of the local DEIM interpolant for the $j+1$th iteration.

In machine learning, this way of constructing the indicator $\boldsymbol{z}$ is called feature extraction or feature selection [8]. With $\tilde{\boldsymbol{F}}(\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ we represent the high-dimensional vector $\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ in a low-dimensional space $\mathbb{R}^{\tilde{M}}$, where $\tilde{\boldsymbol{F}}(\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ still contains the relevant information to correctly classify the point $\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$. It is important to note that the main purpose of the indicator $\boldsymbol{z}^j = \tilde{\boldsymbol{F}}(\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ is not to be a good approximation of $\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ but only to decide with $c(\boldsymbol{z}^j)$ which local DEIM approximation to use for the approximation in the $j+1$th iteration. Since we need the indicator $\boldsymbol{z}^j = \tilde{\boldsymbol{F}}(\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ of $\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ whenever we want to switch the local DEIM approximation, the evaluation of $\tilde{\boldsymbol{F}}$ must be cheap to ensure a rapid online phase.

We propose two different maps $\tilde{\boldsymbol{F}}$ to compute the indicators. Let $(\boldsymbol{U}_g, \boldsymbol{P}_g)$ be the (global) DEIM approximation with $\tilde{M}$ basis vectors and $\tilde{M}$ interpolation points constructed from the set of nonlinear snapshots $\mathcal{S}$. We define the DEIM-based feature extraction as

$$\tilde{\boldsymbol{F}}_D(\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu})) = (\boldsymbol{P}_g^T\boldsymbol{U}_g)^{-1}\boldsymbol{P}_g^T\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu})) \tag{5.1}$$

and the point-based feature extraction as

$$\tilde{\boldsymbol{F}}_P(\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu})) = \boldsymbol{P}_g^T\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu})). \tag{5.2}$$

Both $\tilde{\boldsymbol{F}}_D$ and $\tilde{\boldsymbol{F}}_P$ require us to evaluate only $\tilde{M}$ components of $\boldsymbol{F}$. The DEIM-based feature extraction $\tilde{\boldsymbol{F}}_D$ maps $\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ onto the coefficients $\boldsymbol{\alpha}(\boldsymbol{\mu}) \in \mathbb{R}^{\tilde{M}}$ of the DEIM linear combination $\boldsymbol{U}\boldsymbol{\alpha}(\boldsymbol{\mu})$. This is a good representation of the important information contained in $\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ because of the properties of the POD basis $\boldsymbol{U}$ underlying DEIM. The motivation for (5.2) is the greedy algorithm of the DEIM [12], which can be considered as feature extraction. It selects those components of $\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ which are used to compute the coefficients of the linear combination with the DEIM basis $\boldsymbol{U}$. Thus, the selected components play an essential role in capturing the behavior of the nonlinear term [12]. The point-based feature extraction does not require the matrix-vector product with $(\boldsymbol{P}_g^T\boldsymbol{U}_g)^{-1} \in \mathbb{R}^{\tilde{M}\times\tilde{M}}$ and thus it is computationally cheaper than the DEIM-based map $\tilde{\boldsymbol{F}}_D$.

5.2. Efficient computation of the indicator. In contrast to the parameter-based LDEIM where $\boldsymbol{z}$ was simply the parameter $\boldsymbol{\mu}$, the computation of the indicator $\boldsymbol{z}^j = \tilde{\boldsymbol{F}}(\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ introduces additional costs. If we consider the two proposed representations (5.1) and (5.2), we see that the nonlinear term is evaluated at $\tilde{M}$ components, and a matrix-vector product with $(\boldsymbol{P}_g^T\boldsymbol{U}_g)^{-1} \in \mathbb{R}^{\tilde{M}\times\tilde{M}}$ is required in the case of the DEIM-based feature extraction. Whereas the computational costs of the matrix-vector product with a matrix of size $\tilde{M}\times\tilde{M}$ are negligible, the costs of evaluating the nonlinear term $\boldsymbol{F}$ at $\tilde{M}$ components for the feature extraction might be quite high even though $\tilde{M}$ is usually much smaller than M. Note that the $\tilde{M}$ components required for the DEIM approximation underlying the feature extraction are most probably different from the M components used in the local DEIM approximations. Therefore, we do not evaluate $\boldsymbol{F}$ but instead interpolate $\boldsymbol{F}$ with the local DEIM approximation at the $\tilde{M}$ components required to get the indicator. Thus, for each local DEIM approximation $(\boldsymbol{U}_i, \boldsymbol{P}_i)$, we store the matrix

$$\boldsymbol{W}_i^D = (\boldsymbol{P}_g^T\boldsymbol{U}_g)^{-1}\boldsymbol{P}_g^T\boldsymbol{U}_i(\boldsymbol{P}_i^T\boldsymbol{U}_i)^{-1} \in \mathbb{R}^{\tilde{M}\times M} \tag{5.3}$$

for the DEIM-based feature extraction (5.1) and the matrix

$$\boldsymbol{W}_i^P = \boldsymbol{P}_g^T \boldsymbol{U}_i (\boldsymbol{P}_i^T \boldsymbol{U}_i)^{-1} \in \mathbb{R}^{\tilde{M} \times M} \tag{5.4}$$

for the point-based feature extraction (5.2). Now, suppose the local DEIM interpolant $(\boldsymbol{U}_i, \boldsymbol{P}_i)$ has been selected for the approximation in the jth Newton iteration. Then we store the vector

$$\tilde{\boldsymbol{f}}^j = \boldsymbol{P}_i \boldsymbol{F}(\boldsymbol{V}_N \tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))\,.$$

This vector is used to compute the local DEIM approximation in the system (3.1) in the jth iteration, but it is also used to compute the indicator $\boldsymbol{z}^j$ of $\boldsymbol{F}(\boldsymbol{V}_N \tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$ with the matrix (5.3) and (5.4), respectively.

We emphasize that this allows us to perform the feature extraction without any additional evaluations of the nonlinear term $\boldsymbol{F}$. Thus, just as in the parameter-based LDEIM, we evaluate the nonlinear term only at the M components for the local DEIM approximation. This interpolation introduces a small additional error that can lead to a different selection of the local DEIM approximation than if we evaluated the nonlinear term at the $\tilde{M}$ components. The numerical results in section 6 show that this error has a small effect on the overall accuracy of the state-based LDEIM.

5.3. Clustering and classification method for state-based LDEIM. In contrast to the clustering method used in the parameter-based LDEIM, the state-based LDEIM does not directly cluster the high-dimensional snapshots in $\mathcal{S}$ but clusters their indicators in $\tilde{\mathcal{S}} = \{\tilde{\boldsymbol{F}}(\boldsymbol{V}_N^T \boldsymbol{y}(\boldsymbol{\mu}_1)), \dots, \tilde{\boldsymbol{F}}(\boldsymbol{V}_N^T \boldsymbol{y}(\boldsymbol{\mu}_m))\}$. The data in $\tilde{\mathcal{S}}$ are clustered with k-means with respect to the Euclidean norm and with a random initial clustering. The cluster centroids are now points in $\mathbb{R}^{\tilde{M}}$ and in each iteration of k-means, a point $\boldsymbol{z} \in \tilde{\mathcal{S}}$ is assigned to the cluster

$$\underset{i \in \{1,\dots,k\}}{\arg\min}\ \|\boldsymbol{s}_i - \boldsymbol{z}\|_2\,,$$

where $\boldsymbol{s}_i$ is the cluster centroid of cluster i. The Euclidean norm is a sufficient choice here because the indicators in $\tilde{\mathcal{S}}$ already contain only the most important information about the snapshots. Note that it is cheaper to cluster $\tilde{\mathcal{S}} \subset \mathbb{R}^{\tilde{M}}$ instead of $S \subset \mathbb{R}^{\mathcal{N}}$ as in the parameter-based LDEIM.

The result of the clustering method is a partition $\tilde{\mathcal{S}}_1 \uplus \cdots \uplus \tilde{\mathcal{S}}_k$ of $\tilde{\mathcal{S}}$ and therefore also of $\mathcal{S}$ into k subsets. For each of the subsets $\mathcal{S}_1, \dots, \mathcal{S}_k$, we compute a DEIM approximation $(\boldsymbol{U}_1, \boldsymbol{P}_1), \dots, (\boldsymbol{U}_k, \boldsymbol{P}_k)$. The classifier $c : \mathcal{Z} \to \{1, \dots, k\}$ with $\mathcal{Z} = \mathbb{R}^{\tilde{M}}$ is then trained on the data

$$\tilde{\mathcal{S}}_1 \times \{1\} \cup \cdots \cup \tilde{\mathcal{S}}_k \times \{k\} \subset \mathbb{R}^{\tilde{M}} \times \{1, \dots, k\}. \tag{5.5}$$

As in the parameter-based LDEIM, we employ a nearest neighbor classifier.

5.4. Offline and online procedure. The computational procedure of the state-based LDEIM follows the usual decomposition into an offline and an online phase. In the offline phase, we cluster the set of snapshots $\mathcal{S}$ and construct the classifier $c : \mathcal{Z} \to \{1, \dots, k\}$. In the online phase, we solve the nonlinear reduced model using the Newton method, where we employ the local DEIM approximation chosen by the classifier.

Algorithm 3. Construction procedure of state-based LDEIM.

```
1: procedure conSLDEIM($M$, $\tilde{M}$, $k$, $\mathcal{S}$, $\tilde{\boldsymbol{F}}$, $\boldsymbol{V}_N$)
2:     $\tilde{\mathcal{S}} \leftarrow \{\tilde{\boldsymbol{F}}(\boldsymbol{V}_N^T\boldsymbol{y}(\boldsymbol{\mu}_1)), \dots, \tilde{\boldsymbol{F}}(\boldsymbol{V}_N^T\boldsymbol{y}(\boldsymbol{\mu}_m))\}$
3:     $(\tilde{\mathcal{S}}_1, \dots, \tilde{\mathcal{S}}_k) \leftarrow k$-means$(\tilde{\mathcal{S}}, k, \|\cdot\|_2)$
4:     $c \leftarrow$ train classifier on $\tilde{\mathcal{S}}_1 \times \{1\} \cup \ldots \cup \tilde{\mathcal{S}}_k \times \{k\}$
5:     $(\boldsymbol{U}_g, \boldsymbol{P}_g) \leftarrow$ DEIM$(\mathcal{S}, \tilde{M})$
6:     $\ell \leftarrow$ empty list
7:     for $i = 1 : k$ do
8:         $\mathcal{S}_i = \{\boldsymbol{F}(\boldsymbol{y}(\boldsymbol{\mu}_n)) \,|\, \tilde{\boldsymbol{F}}(\boldsymbol{V}_N\boldsymbol{y}(\boldsymbol{\mu}_n)) \in \tilde{\mathcal{S}}_i\}$
9:         $(\boldsymbol{U}_i, \boldsymbol{P}_i) \leftarrow$ DEIM$(\mathcal{S}_i, M)$
10:        $\boldsymbol{W}_i \leftarrow$ depending on the feature extraction store either matrix (5.3) or (5.4)
11:        $\ell \leftarrow$ append $((\boldsymbol{U}_i, \boldsymbol{P}_i), \boldsymbol{W}_i)$ to list $\ell$
12:    end for
13: return $(\ell, (\boldsymbol{U}_g, \boldsymbol{P}_g))$
14: end procedure
```

5.4.1. Offline phase. The core of the offline phase is the construction procedure summarized in Algorithm 3. Inputs are the number M of local DEIM basis vectors and interpolation points, the dimensions $\tilde{M}$ of the indicator $\boldsymbol{z}$, the number of clusters k, the set of snapshots $\mathcal{S}$, the map $\tilde{\boldsymbol{F}}$, and the POD basis $\boldsymbol{V}_N$. First, the indicators of the snapshots in $\mathcal{S}$ are computed and stored in $\tilde{\mathcal{S}}$. Then, they are clustered with the k-means clustering method as discussed in section 5.3. The result is the partition of $\tilde{\mathcal{S}}$ and of $\mathcal{S}$ into k subsets. For each subset $\mathcal{S}_i$, the local DEIM approximation $(\boldsymbol{U}_i, \boldsymbol{P}_i)$ is built and the matrix $\boldsymbol{W}_i$ is stored. The matrix $\boldsymbol{W}_i$ is either (5.3) or (5.4). The global DEIM approximation $(\boldsymbol{U}_g, \boldsymbol{P}_g)$ as well as the matrix $\boldsymbol{W}_i$ are required for the efficient construction of the indicator $\boldsymbol{z}$ in the online phase.

The k-means clustering method is initialized with a random cluster assignment. As discussed in section 5.3, the k-means clustering is repeated several times to ensure a good clustering. Still, in certain situations, this might not be enough. Therefore, we additionally split off a small test data set and repeat the whole construction procedure in Algorithm 3 several times for $\mathcal{S}$. We then select the result of the run where the DEIM residual (4.1) for the test data set is smallest.

5.4.2. Online phase. In the online phase, our task is to select a local DEIM approximation $(\boldsymbol{U}_i, \boldsymbol{P}_i)$ for the next Newton iteration. The selection procedure is shown in Algorithm 4. The inputs are the list ℓ containing the local DEIM approximations and the matrices $\boldsymbol{W}_1, \dots, \boldsymbol{W}_k$ as computed in Algorithm 3, the index $i \in \{1, \dots, k\}$ of the local DEIM approximation employed in the j-th Newton iteration, the state vector $\tilde{\boldsymbol{y}}^{j+1}(\boldsymbol{\mu})$ computed in the jth iteration, and the vector $\tilde{\boldsymbol{f}}^j = \boldsymbol{P}_i\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^j(\boldsymbol{\mu}))$. With the matrix $\boldsymbol{W}_i$ the indicator $\boldsymbol{z}^j$ is computed. Then the index i of the local DEIM approximation for the $j+1$th Newton iteration is updated by evaluating the classifier c at z^j. We can then evaluate the nonlinear term $\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^{j+1}(\boldsymbol{\mu}))$ at the indices given by the matrix $\boldsymbol{P}_i$ and store the values in $\tilde{\boldsymbol{f}}^{j+1}$. The outputs are the vector $\tilde{\boldsymbol{f}}^{j+1}$ and the index i of the local DEIM approximation for the $j+1$th Newton iteration.

Note that if the Newton method does not converge, we switch to a classical DEIM approximation with M basis vectors and M interpolation points. This is not

Algorithm 4. Selection procedure of state-based LDEIM.

```
1: procedure selSLDEIM($\boldsymbol{V}_N$, $\boldsymbol{F}$, $\ell$, $c$, $i$, $\tilde{\boldsymbol{y}}^{j+1}(\boldsymbol{\mu})$, $\tilde{\boldsymbol{f}}^j$)
2:     $\boldsymbol{z} \leftarrow \boldsymbol{W}_i\tilde{\boldsymbol{f}}^j$
3:     $i \leftarrow c(\boldsymbol{z})$
4:     $\tilde{\boldsymbol{f}}^{j+1} \leftarrow \boldsymbol{P}_i\boldsymbol{F}(\boldsymbol{V}_N\tilde{\boldsymbol{y}}^{j+1}(\boldsymbol{\mu}))$
5: return $(i, \tilde{\boldsymbol{f}}^{j+1})$
6: end procedure
```

needed in the parameter-based LDEIM because there we do not switch the local DEIM interpolant between Newton iterations. The fall back to the classical DEIM is necessary only in exceptional cases, e.g., if the solution lies just between two clusters and we jump back and forth between them.

6. Numerical results.

In this section, we show that using LDEIM we achieve the same level of accuracy as with DEIM but with fewer interpolation points. In section 6.1, parameter-based LDEIM is demonstrated on three benchmark problems. In section 6.2, we consider a reacting flow example of a two-dimensional premixed H_2-Air flame where we compare DEIM to parameter-based LDEIM and state-based LDEIM. We employ the k-means implementation available in MATLAB and the nearest neighbor search engine of the FLANN library [29].

6.1. Parameter-based LDEIM.

In section 2.3 in (2.8), we introduced the function $g^1 : \Omega \times \mathcal{D} \rightarrow \mathbb{R}$ with the parameter $\boldsymbol{\mu} \in \mathcal{D}$ controlling the gradient of the peak in the corner $(1, 1)$ of the domain Ω. Based on g^1, we defined in (2.9) the function g^4. Depending on the parameter $\boldsymbol{\mu}$, the function g^4 has a peak in one of the four corners of Ω. Let us define $g^2 : \Omega \times \mathcal{D} \rightarrow \mathbb{R}$ as

$$g^2(\boldsymbol{x}; \boldsymbol{\mu}) = g^1(\boldsymbol{x}; \boldsymbol{\mu}) + g^1(1 - x_1, 1 - x_2; 1 - \mu_1, 1 - \mu_2)\,,$$

where the parameters control a peak in the left bottom or the right top corner of Ω. We discretize g^1, g^2, and g^4 on a 20×20 equidistant grid in Ω, sample on a 25×25 grid in $\mathcal{D}$ to compute 625 snapshots, and compute the DEIM interpolant and the parameter-based LDEIM interpolant with splitting and clustering. For the splitting approach, we set the tolerance ϵ in Algorithm 1 to 1e-07, 1e-06, and 1e-05 for g^1, g^2, and g^4, respectively. For the clustering approach, the number of clusters is $k = 4$. Note that we cannot employ state-based LDEIM here because we have a pure interpolation task and no time steps or Newton iterations; see section 5. The interpolants are tested with the snapshots corresponding to the 11×11 equidistant grid in $\mathcal{D}$. As in section 2.3, we compare the averaged L^2 error of the approximations with respect to the analytically given functions g^1, g^2, and g^4. The results are shown in Figure 6.1.

The plots in the first row of Figure 6.1 indicate the subdomains of $\mathcal{D}$ obtained with the splitting approach. For example, consider g^2. The domain is split most near the left bottom and right top corners, i.e., in locations where, depending on the parameter, a peak can occur. We find similar behavior for g^1 and g^4. In the second row, we plot the parameters in $\mathcal{D}$ corresponding to the 25×25 snapshots and color them according to the cluster assignment obtained with the parameter-based LDEIM with clustering. Again, the clusters divide $\mathcal{D}$ according to where the sharp peaks occur. We see that the clusters allow a more flexible partition of $\mathcal{D}$. In particular,

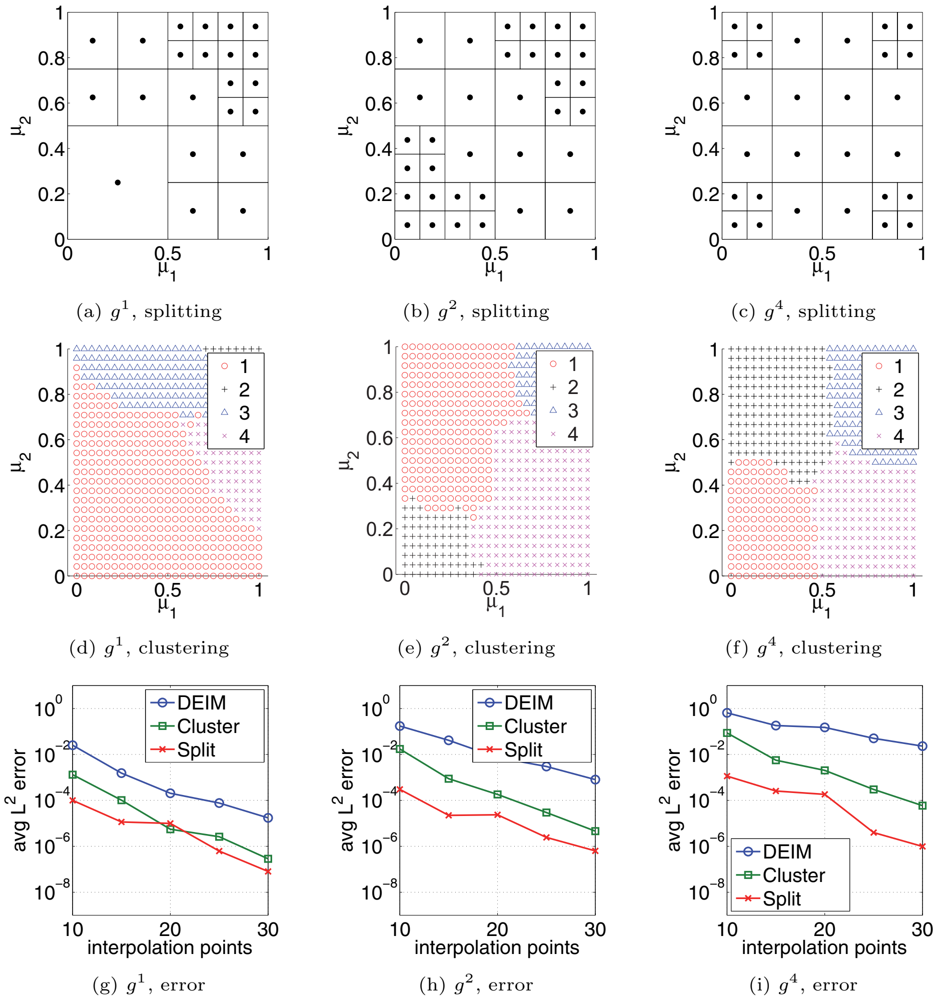
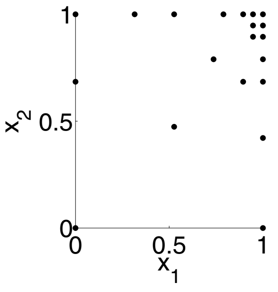
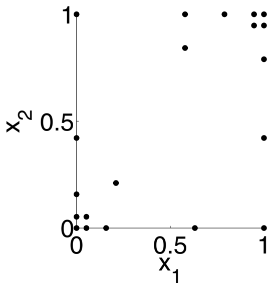


Fig. 6.1. *Parameter-based LDEIM applied to the three benchmark examples g^1 (left), g^2 (middle), and g^4 (right). Both splitting and clustering methods group the snapshots in a reasonable way. The splitting and the clustering are shown for* 20 *DEIM basis vectors and interpolation points. Accuracy compared to DEIM improves around two orders of magnitude and up to four orders in the case of g^4.*

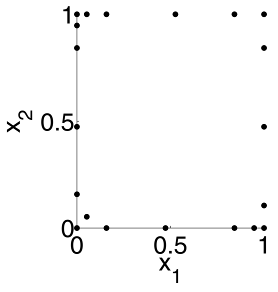
this can be seen for g^1, where we obtain clusters with curvilinear boundaries. In the third row of Figure 6.1, we plot the averaged L^2 error against the number of DEIM interpolation points. LDEIM achieves an accuracy up to four orders of magnitude higher than DEIM. In section 2.3, we argued that DEIM approximates the function g^4 worse than g^1 because the DEIM interpolant has to capture all four peaks of g^4 at once. If we now compare the result obtained with LDEIM for g^4 with the result of DEIM for g^1, we see that we have a similarly good performance with LDEIM for g^4 as with DEIM for g^1. This is expected because each cluster corresponds to exactly one peak, i.e., one summand in (2.9), and thus the four local DEIM approximations together should be able to approximate g^4 as well as one global DEIM interpolant approximates g^1.



function g^1



function g^2



function g^4

FIG. 6.2. *The DEIM interpolation points for the functions g^1, g^2, and g^4. We set the number of DEIM basis vectors and interpolation points to* 20.

Figure 6.2 shows the interpolation points picked by the DEIM. It can be seen that DEIM concentrates on the right top corner in case of g^1. For the functions g^2 and g^4, however, DEIM distributes the points roughly equally among the four corners; thus, many more interpolation points would be required to cover all corners sufficiently. In Figure 6.3, we plot the LDEIM interpolation points of the four interpolants corresponding to the four clusters of LDEIM shown in Figure 6.1. In case of the function g^4, each cluster corresponds to one corner of the spatial domain and thus each local DEIM interpolant can place its interpolation points near its corner. We find a similar situation for function g^2 where cluster 2 and cluster 3 correspond to the peaks. For function g^1 too, the localization achieves an improvement by placing the interpolation points near either the top or the right edge of the domain.

6.2. Reacting flow simulation. We consider a model of a steady premixed H_2-Air flame. We briefly introduce the problem, its governing equations, and the POD-DEIM reduced-order model, but we refer to [9] for a detailed discussion.

We simulate the two-dimensional premixed H_2-Air flame underlying the one-step reaction mechanism

$$2\mathrm{H}_2 + \mathrm{O}_2 \rightarrow 2\mathrm{H}_2\mathrm{O}\,,$$

where H_2 is the fuel, O_2 is the oxidizer, and H_2O is the product. The evolution of the flame in the domain Ω is given by the nonlinear advection-diffusion-reaction equation

$$\kappa \Delta \boldsymbol{y} - w \nabla \boldsymbol{y} + \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{\mu}) = 0\,, \tag{6.1}$$

where $\boldsymbol{y} = [y_{\mathrm{H}_2}, y_{\mathrm{O}_2}, y_{\mathrm{H}_2\mathrm{O}}, T]^T$ contains the mass fractions of species $\mathrm{H}_2, \mathrm{O}_2$, and $\mathrm{H}_2\mathrm{O}$ and the temperature. The constants $\kappa = 2.0\mathrm{cm}^2/\mathrm{sec}$ and $w = 50\mathrm{cm/sec}$ are the molecular diffusivity and the velocity in x_1 direction, respectively. The geometry of the domain Ω is shown in Figure 6.4. With the notation of Figure 6.4, we have homogeneous Dirichlet boundary conditions on the mass fractions on Γ_1 and Γ_3 and homogeneous Neumann conditions on temperature and mass fractions on Γ_4, Γ_5, and Γ_6. We have Dirichlet conditions on Γ_2 with $y_{\mathrm{H}_2} = 0.0282, y_{\mathrm{O}_2} = 0.2259, y_{\mathrm{H}_2\mathrm{O}} = 0, y_T = 950\mathrm{K}$ and on Γ_1, Γ_3 with $y_T = 300\mathrm{K}$.

The nonlinear reaction source term $\boldsymbol{f}(\boldsymbol{y}, \boldsymbol{\mu}) = [f_{\mathrm{H}_2}(\boldsymbol{y}, \boldsymbol{\mu}), f_{\mathrm{O}_2}(\boldsymbol{y}, \boldsymbol{\mu}), f_{\mathrm{H}_2\mathrm{O}}(\boldsymbol{y}, \boldsymbol{\mu}), f_T(\boldsymbol{y}, \boldsymbol{\mu})]^T$ in (6.1) has the components

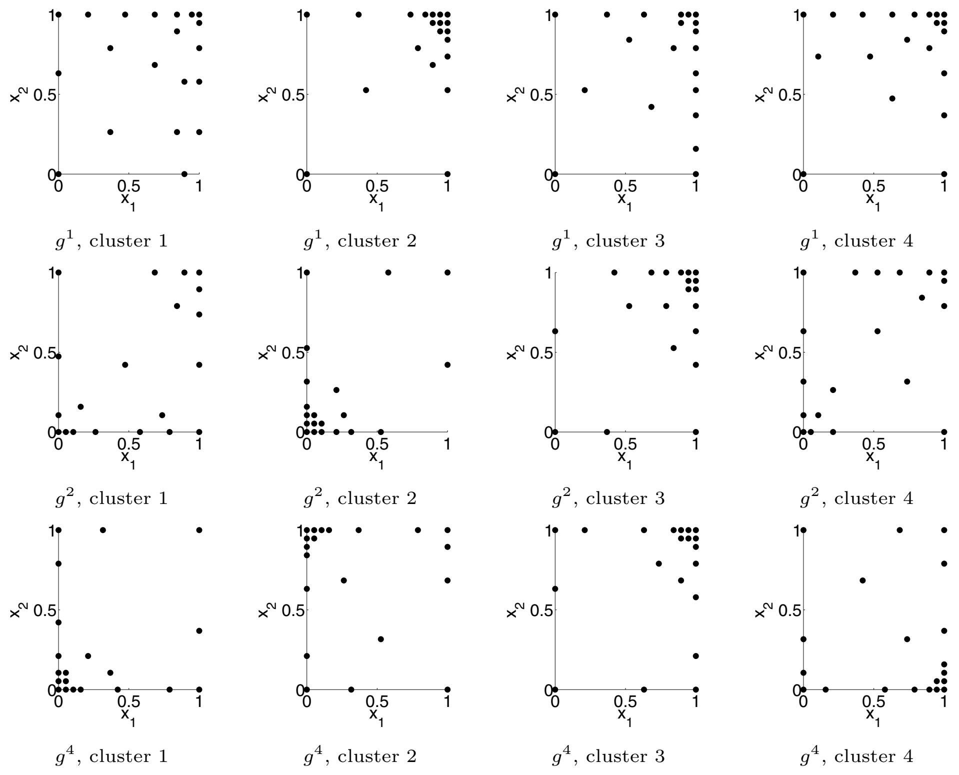


FIG. 6.3. *The interpolation points corresponding to the local DEIM approximations obtained with parameter-based LDEIM with clustering for the functions g^1, g^2, and g^4. We set the number of local DEIM modes to* 20. *The clustering allows LDEIM to concentrate the interpolation points in only certain parts of the spatial domain, i.e., near the corners.*

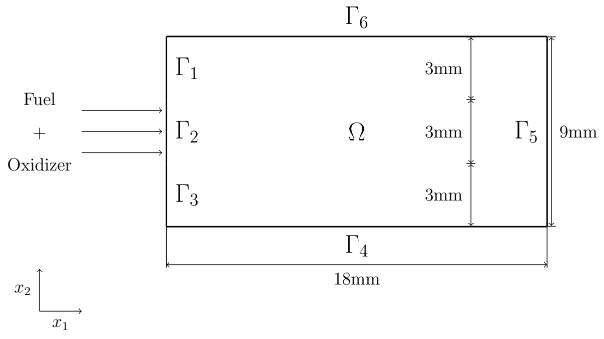


FIG. 6.4. *The spatial domain of the reacting flow simulation* [9].

$$f_i(\boldsymbol{y},\boldsymbol{\mu}) = -\nu_i \left(\eta_{\mathrm{H}_2} y_{\mathrm{H}_2}\right)^2 \left(\eta_{\mathrm{O}_2} y_{\mathrm{O}_2}\right) \mu_1 \exp\left(-\frac{\mu_2}{RT}\right), \quad i = \mathrm{H}_2, \mathrm{O}_2, \mathrm{H}_2\mathrm{O},$$
$$f_T(\boldsymbol{y},\boldsymbol{\mu}) = Q f_{\mathrm{H}_2\mathrm{O}}(\boldsymbol{y},\boldsymbol{\mu}),$$

where ν_i and η_i are constant parameters, $R = 8.314472\mathrm{J}/(\mathrm{mol\,K})$ is the universal gas constant, and $Q = 9800\mathrm{K}$ is the heat of reaction. The parameters $\boldsymbol{\mu} = (\mu_1, \mu_2) \in \mathcal{D}$ with $\mathcal{D} = [5.5\mathrm{e}{+}11, 1.5\mathrm{e}{+}13] \times [1.5\mathrm{e}{+}03, 9.5\mathrm{e}{+}03]$ are the preexponential factor and the activation energy, respectively. Equation (6.1) is discretized using the finite difference method on a 73×37 grid leading to $\mathcal{N} = 10,804$ degrees of freedom. The result is a nonlinear system of discrete algebraic equations

$$\boldsymbol{A}\boldsymbol{y} + \boldsymbol{F}(\boldsymbol{y}, \boldsymbol{\mu}) = 0\,, \tag{6.2}$$

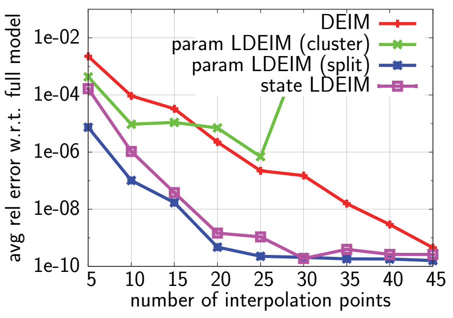
where now the vector $\boldsymbol{y} \in \mathbb{R}^{\mathcal{N}}$ contains the mass fractions and temperature at the grid points. The matrix $\boldsymbol{A} \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ corresponds to the linear differential operators and the function $\boldsymbol{F} : \mathbb{R}^{\mathcal{N}} \to \mathbb{R}^{\mathcal{N}}$ to the nonlinear source term. The nonlinear equations (6.2) are solved with the Newton method. In [9], the POD-DEIM reduced-order system is derived as

$$\boldsymbol{V}_N^T\boldsymbol{A}\overline{\boldsymbol{y}} + \boldsymbol{V}_N^T\boldsymbol{A}\boldsymbol{V}_N\tilde{\boldsymbol{y}} + \boldsymbol{V}_N^T\boldsymbol{U}(\boldsymbol{P}^T\boldsymbol{U})^{-1}\boldsymbol{F}(\boldsymbol{P}^T\overline{\boldsymbol{y}} + \boldsymbol{P}^T\boldsymbol{V}_N\tilde{\boldsymbol{y}}, \boldsymbol{\mu}) = 0 \tag{6.3}$$

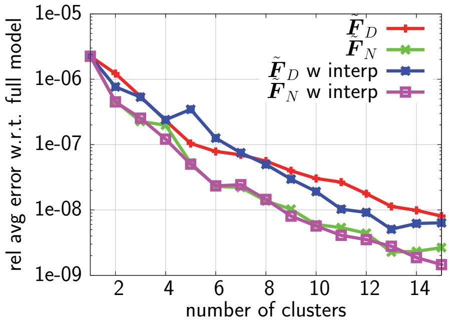
with the arithmetic mean $\overline{\boldsymbol{y}}$ of the set of snapshots $\{\boldsymbol{y}_1, \dots, \boldsymbol{y}_m\}$, the POD basis $\boldsymbol{V}_N \in \mathbb{R}^{\mathcal{N}\times N}$ computed from $\{\boldsymbol{y}_j - \overline{\boldsymbol{y}}\}_{j=1}^m$, and the DEIM interpolant $(\boldsymbol{U}, \boldsymbol{P})$ with M modes. The snapshots are computed for the parameters on a 50×50 equidistant grid in $\mathcal{D}$.

Instead of DEIM, we employ parameter-based and state-based LDEIM, solve the POD-LDEIM system for parameters on a 24×24 grid in $\mathcal{D}$, and compute the average relative error of the temperature with respect to the full-order finite difference model. The results are shown in Figure 6.5. Figure 6.5(a) compares DEIM, parameter-based LDEIM with splitting and clustering, and state-based LDEIM. For splitting, we set the tolerance ϵ to 1e-08, which is about two orders below what DEIM achieves. For the parameter-based LDEIM with clustering, the number of clusters is set to 5. More clusters lead to unstable behavior. For the state-based LDEIM, however, we set the number of clusters to 15. The state-based LDEIM uses the point-based feature extraction (5.2) with $\tilde{M} = 5$ dimensions. In all cases, we have 40 POD modes. In Figure 6.5(a), we see that the results of the parameter-based LDEIM with clustering do not improve after about 10 DEIM modes. Again, the clustering becomes unstable. However, this is not the case for state-based LDEIM, which achieves about two orders of magnitude better accuracy than DEIM. The same holds for the splitting approach.

In Figure 6.5(b), we compare the feature extractions $\tilde{\boldsymbol{F}}_D$ and $\tilde{\boldsymbol{F}}_P$ introduced in (5.2) and (5.1), respectively. We show the difference with respect to accuracy between evaluating the nonlinear term $\boldsymbol{F}$ and interpolating the required values with



(a) comparison of DEIM and LDEIM



(b) effect of feature extraction on state-based LDEIM

FIG. 6.5. *For the reacting flow example:* (a) *the comparison between DEIM, parameter-based LDEIM, and state-based LDEIM;* (b) *the effect of* $\tilde{\boldsymbol{F}}_D$ *and* $\tilde{\boldsymbol{F}}_P$ *feature extraction with and without interpolation. The results are shown for* 40 *POD and* 20 *DEIM modes.*

the matrices defined in (5.3) and (5.4); cf. section 5.2. The figure shows that for this problem there is no large difference between the two feature extraction methods and thus there is no significant loss of accuracy if we interpolate the values of $\boldsymbol{F}$ at the points required by the feature extraction instead of directly evaluating $\boldsymbol{F}$.

Figure 6.6 plots the temperature of the flame for parameters $\boldsymbol{\mu} = (7.775\text{e}{+}12, 5.5\text{e}{+}03)$ and the interpolation points selected by standard DEIM. We see that the points are concentrated near the inflow boundary Γ_2; cf. Figure 6.4. In Figure 6.7 the interpolation points of the local DEIM interpolants for state-based LDEIM with four clusters are shown. The points either are focused near the top (clusters 1 and 4) corner of the boundary Γ_2 or the bottom (cluster 2) of Γ_2 or are roughly equally distributed near Γ_2 and in the region with the highest temperature (cluster 3).

In Figure 6.8, we fix the number of POD and DEIM modes and increase the number of clusters in state-based LDEIM. In Figure 6.8(a), we see that if the number of POD and DEIM modes is high, e.g., 40/40, then increasing the number of clusters does not lead to improved accuracy. Even though the DEIM approximation gets more and more accurate as we increase the number of clusters, the POD basis is

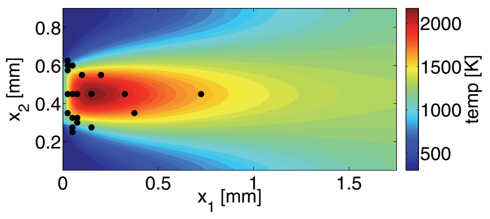
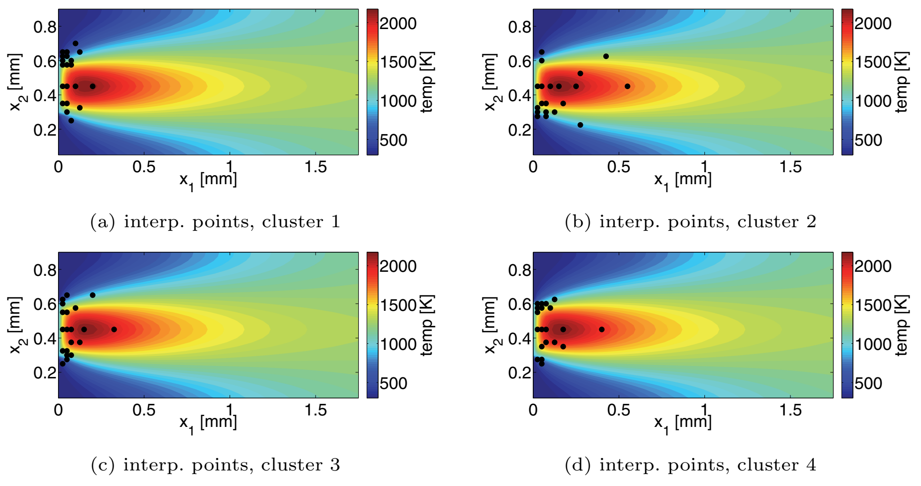


FIG. 6.6. *Interpolation points of standard DEIM for the reacting flow example. The points are concentrated near the top, middle, and bottom of boundary Γ_2 where the fuel and oxidizer are injected; cf. Figure* 6.4.



(a) interp. points, cluster 1

(b) interp. points, cluster 2

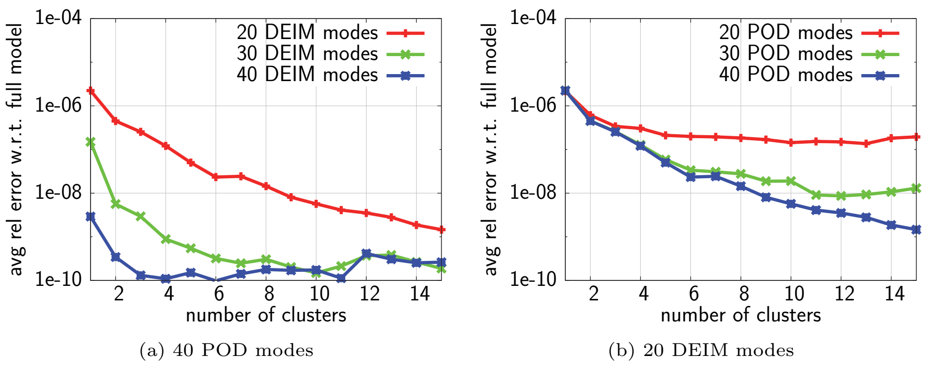
(c) interp. points, cluster 3

(d) interp. points, cluster 4

FIG. 6.7. *For the reacting flow example, the local DEIM interpolation points corresponding to state-based LDEIM with four clusters are shown. The interpolation points for clusters* 1 *and* 4 *are concentrated at the top corner of the inflow boundary Γ_2, the points for cluster* 2 *are concentrated at the bottom of Γ_2, and the interpolation points for cluster* 3 *are roughly equally distributed near Γ_2 and in the region with the highest temperature of the flame.*

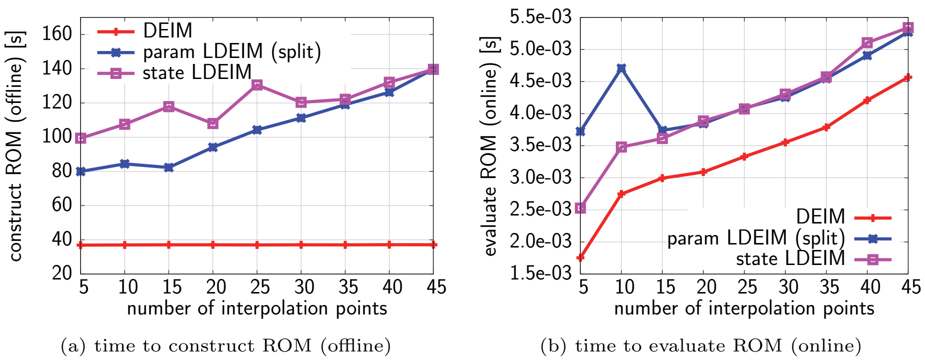
fixed and thus limits the accuracy of the overall result (although we note that the magnitude of the errors is already very small). A localization approach for the POD basis could further improve the accuracy; see, e.g., [14, 1]. Note that although it might not help to increase the number of clusters of the state-based LDEIM here, it also does not deteriorate the results. Thus, in contrast to parameter-based LDEIM with clustering, the clustering in state-based LDEIM does not become unstable for the problems studied. For these examples, state-based LDEIM is not sensitive to the number of clusters. These observations are confirmed in Figure 6.8(b).

The runtimes of the offline and online phases of DEIM and LDEIM are plotted in Figure 6.9. The computations were repeated five times, and reported are the averaged runtimes on an Intel SandyBridge-EP Xeon E5-2670. We have the same parameters as for the error plot in Figure 6.5(a). In Figure 6.9(a) we plot the time spent in the offline phase to construct a ROM with the classical DEIM, parameter-based LDEIM with splitting (tolerance is 1e-08), and state-based LDEIM (15 clusters). Note that we did not include the time for the computation of the snapshots. Because LDEIM



(a) 40 POD modes (b) 20 DEIM modes

FIG. 6.8. *For the reacting flow example, the number of POD/DEIM modes is fixed and the number of clusters is increased. Increasing the number of clusters improves the accuracy of the DEIM approximation but the POD basis that approximates the state limits improvement in the overall result. Results are shown for state-based LDEIM with point-based feature extraction.*



(a) time to construct ROM (offline) (b) time to evaluate ROM (online)

FIG. 6.9. *For the reacting flow example:* (a) *the runtimes for constructing the POD-DEIM and POD-LDEIM reduced-order models (offline phase);* (b) *the runtime of evaluating the reduced-order model. The results are shown for* 40 *POD modes and correspond to the error plot in Figure* 6.5(a). *Note that we do not show the results for parameter-based LDEIM with clustering because it becomes unstable; cf. Figure* 6.5(a).

Table 6.1

State-based LDEIM with 40 *POD and* 40 *DEIM modes for our two feature extraction methods with up to* 90 *clusters for the reacting flow example. The runtime of the selection procedure of state-based LDEIM does not increase when we increase the number of clusters. We also report the averaged error of the temperature with respect to the full-order model. The errors confirm that the clustering remains stable as we increase the number of clusters.*

	$\tilde{\boldsymbol{F}}_P$ w/out interp		$\tilde{\boldsymbol{F}}_D$ w/out interp		$\tilde{\boldsymbol{F}}_P$ with interp		$\tilde{\boldsymbol{F}}_D$ with interp	
k	Error	Time	Error	Time	Error	Time	Error	Time
10	2.15e-10	1.000	2.65e-10	1.000	2.02e-10	0.230	3.11e-10	0.792
30	4.37e-10	0.969	2.94e-10	0.967	1.01e-09	0.223	4.77e-09	0.797
50	6.73e-09	0.978	2.06e-09	0.983	4.26e-09	0.223	7.04e-09	0.794
70	4.84e-09	0.964	2.20e-09	0.981	4.85e-09	0.222	1.49e-07	0.806
90	4.66e-09	0.942	5.67e-09	0.982	3.40e-09	0.218	6.40e-07	0.816

partitions the snapshots and then builds multiple local DEIM interpolants, the offline phase corresponding to LDEIM requires about two to three times more time than with the classical DEIM. Furthermore, the offline runtime of state-based LDEIM is not deterministic because the number of iterations until convergence of the clustering method (k-means) depends on the random initial clustering; however, we do not observe any large outliers. In Figure 6.9(b) we report the runtime of evaluating the reduced-order model (online phase). It shows that LDEIM has a constant overhead compared to DEIM due to the selection procedure; however, this overhead is quickly compensated by the reduction in the dimension of the local DEIM interpolants. For example, to achieve an error below 10^{-9}, the classical DEIM requires 45 interpolation points with a runtime of $4.5 \cdot 10^{-3}$ seconds, whereas about 20 points are sufficient for both LDEIM variants leading to a runtime below $4.0 \cdot 10^{-3}$ seconds; cf. Figures 6.5(a) and 6.9(b). We emphasize that we employ a standard MATLAB implementation that is not fine-tuned for rapid offline and online phases (no parallelization, no vectorization). Furthermore, we note that the overhead of the selection procedure becomes less and less important as the costs of evaluating the nonlinear term increase.

Finally, let us consider the error and runtime results shown in Table 6.1. We show the averaged relative error and the online runtimes of the selection procedure of state-based LDEIM with 40 POD and 40 DEIM modes with up to 90 clusters. Again, the computations were repeated five times and the averaged results are reported. We show the results for the two feature extraction methods $\tilde{\boldsymbol{F}}_D$ and $\tilde{\boldsymbol{F}}_P$ with and without interpolating the nonlinear term as discussed in section 5.2. The errors confirm once again that the clustering remains stable as we increase the number of clusters, even though there is a slight increase in the error corresponding to $\tilde{\boldsymbol{F}}_D$ with interpolation. The runtimes in Table 6.1 are normalized with respect to the runtimes for 10 clusters for the respective feature extraction method without interpolation. We see that an increase in the number of clusters does not entail a runtime increase. Furthermore, interpolating the nonlinear term pays off well in the case of nodal-based feature extraction $\tilde{\boldsymbol{F}}_P$ (four times faster) but has only a small effect when $\tilde{\boldsymbol{F}}_D$ is employed. That is because the evaluation of $\tilde{\boldsymbol{F}}_D$ is more expensive than $\tilde{\boldsymbol{F}}_P$—it requires an additional matrix-vector product—and thus the evaluation of the nonlinear term does not have such a large share in the overall computational costs.

7. Conclusion.

The localized discrete empirical interpolation method was presented. Instead of only one DEIM interpolant, LDEIM computes several interpolants by partitioning the set of nonlinear snapshots into k subsets in the offline phase. In the online phase, one of the local interpolants is selected for the actual approximation. In parameter-based LDEIM, the local interpolant is selected with respect to the parameter of the system. In the state-based LDEIM, a low-dimensional representation of the nonlinear term evaluated at the state vector of the system is constructed to indicate which local DEIM approximation to use.

Machine learning methods play a crucial role in all steps of LDEIM. In the offline phase, the snapshots are clustered with k-means to construct the local interpolants, and in the online phase, the selection procedure relies on classification where nearest neighbor classifiers were employed here. Furthermore, the low-dimensional representation of the nonlinear term in the state-based LDEIM is computed with feature extraction. This low-dimensional representation also leads to a lower-dimensional clustering task and thus to a more stable clustering approach.

As for model order reduction in general, our LDEIM is suited for applications that can cope with an increased offline phase in favor of rapid online evaluations. The only additional costs incurred by LDEIM over DEIM in the online phase are the evaluation costs of the selection procedure. Due to the properties of nearest neighbor classifiers, however, these costs are low if compared to the computational costs of the rest of the procedure. For three benchmark problems, LDEIM achieved accuracy improvements up to four orders of magnitude with respect to DEIM for the same number of interpolation points. In the reacting flow example, the accuracy of the reduced-order model with LDEIM was about two orders of magnitude better than with DEIM.

REFERENCES

[1] D. Amsallem, M. Zahr, and C. Farhat, *Nonlinear model order reduction based on local reduced-order bases*, Internat. J. Numer. Methods Engrg., 92 (2012), pp. 891–916.

[2] M. R. Anderberg, *Cluster Analysis for Applications*, Academic Press, New York, 1973.

[3] D. Arthur and S. Vassilvitskii, *k-means++: The advantages of careful seeding*, in Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07, SIAM, Philadelphia 2007, pp. 1027–1035.

[4] P. Astrid, S. Weiland, K. Willcox, and T. Backx, *Missing point estimation in models described by proper orthogonal decomposition*, IEEE Trans. Automat. Control, 53 (2008), pp. 2237–2251.

[5] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, *Scalable k-means++*, Proc. VLDB Endow., 5 (2012), pp. 622–633.

[6] M. Barrault, Y. Maday, N. C. Nguyen, and A. T. Patera, *An empirical interpolation method: Application to efficient reduced-basis discretization of partial differential equations*, C. R. Math., 339 (2004), pp. 667–672.

[7] G. Berkooz, P. Holmes, and J. L. Lumley, *The proper orthogonal decomposition in the analysis of turbulent flows*, Annu. Rev. Fluid Mech., 25 (1993), pp. 539–575.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2007.

[9] M. Buffoni and K. Willcox, *Projection-based model reduction for reacting flows*, in Proceedings of the 40th Fluid Dynamics Conference and Exhibit, Fluid Dynamics and Co-located Conferences, AIAA, 2010.

[10] J. Burkardt, M. Gunzburger, and H.-C. Lee, *POD and CVT-based reduced-order modeling of Navier-Stokes flows*, Comput. Methods Appl. Mech. Engrg., 196 (2006), pp. 337–355.

[11] K. Carlberg, C. Bou-Mosleh, and C. Farhat, *Efficient non-linear model reduction via a least-squares Petrov-Galerkin projection and compressive tensor approximations*, Internat. J. Numer. Methods Engrg., 86 (2011), pp. 155–181.

[12] S. Chaturantabut and D. Sorensen, *Nonlinear model reduction via discrete empirical interpolation*, SIAM J. Sci. Comput., 32 (2010), pp. 2737–2764.

[13] S. Chaturantabut and D. Sorensen, *A state space error estimate for POD-DEIM nonlinear model reduction*, SIAM J. Numer. Anal., 50 (2012), pp. 46–63.

[14] M. Drohmann, B. Haasdonk, and M. Ohlberger, *Adaptive reduced basis methods for non-linear convection-diffusion equations*, in Finite Volumes for Complex Applications VI:

Problems & Perspectives, J. Fot, J. Fürst, J. Halama, R. Herbin, and F. Hubert eds., vol. 4 of Springer Proc. Math., Springer, 2011, pp. 369–377.
[15] J. Eftang, D. Knezevic, and A. T. Patera, *An hp certified reduced basis method for parametrized parabolic partial differential equations*, Math. Comput. Model. Dyn. Syst., 17 (2011), pp. 395–422.
[16] J. Eftang, A. T. Patera, and E. Rønquist, *An hp certified reduced basis method for parametrized elliptic partial differential equations*, SIAM J. Sci. Comput., 32 (2010), pp. 3170–3200.
[17] J. Eftang and B. Stamm, *Parameter multi-domain hp empirical interpolation*, Internat. J. Numer. Methods Engrg., 90 (2012), pp. 412–428.
[18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, *A density-based algorithm for discovering clusters in large spatial databases with noise*, in Proceedings of KDD'96, 1996, pp. 226–231.
[19] P. Feldmann and R. W. Freund, *Efficient linear circuit analysis by Pade approximation via the Lanczos process*, IEEE Trans., Computer-Aided Design of Integrated Circuits and Systems, 14 (1995), pp. 639–649.
[20] A. D. Gordon, *Classification*, Chapman and Hall, London, 1999.
[21] M. A. Grepl, Y. Maday, N. C. Nguyen, and A. T. Patera, *Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations*, ESAIM Math. Model. Numer. Anal., 41 (2007), pp. 575–605.
[22] B. Haasdonk, M. Dihlmann, and M. Ohlberger, *A training set and multiple bases generation approach for parameterized model reduction based on adaptive grids in parameter space*, Math. Comput. Model. Dyn. Syst., 17 (2011), pp. 423–442.
[23] B. Haasdonk and M. Ohlberger, *Space-adaptive reduced basis simulation for time-dependent problems*, in Proceedings of the Vienna International Conference on Mathematical Modelling, 2009.
[24] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer, New York, 2009.
[25] S. R. Idelsohn and A. Cardona, *A reduction method for nonlinear structural dynamic analysis*, Comput. Methods Appl. Mech. Engrg., 49 (1985), pp. 253–279.
[26] H.-P. Kriegel, P. Kröger, and A. Zimek, *Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering*, ACM Trans. Knowl. Discov. Data, 3 (2009), pp. 1:1–1:58.
[27] J. B. MacQueen, *Some methods for classification and analysis of multivariate observations*, in Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, L. M. Le Cam and J. Neyman, eds., vol. 1, University of California Press, 1967, pp. 281–297.
[28] B. Moore, *Principal component analysis in linear systems: Controllability, observability, and model reduction*, IEEE Trans., Automat. Control, 26 (1981), pp. 17–32.
[29] M. Muja and D. Lowe, *Fast matching of binary features*, in Proceedings of the Conference on Computer and Robot Vision (CRV), 2012, pp. 404–410.
[30] M. Radovanovic, A. Nanopoulos, and M. Ivanovic, *Hubs in space: Popular nearest neighbors in high-dimensional data*, J. Mach. Learn. Res., 11 (2010), pp. 2487–2531.
[31] M. Rathinam and L. Petzold, *A new look at proper orthogonal decomposition*, SIAM J. Numer. Anal., 41 (2003), pp. 1893–1925.
[32] M. Rewieski and J. White, *Model order reduction for nonlinear dynamical systems based on trajectory piecewise-linear approximations*, Linear Algebra Appl., 415 (2006), pp. 426–454.
[33] M. A. Singer and W. H. Green, *Using adaptive proper orthogonal decomposition to solve the reaction–diffusion equation*, Appl. Numer. Math., 59 (2009), pp. 272–279.
[34] L. Sirovich, *Turbulence and the dynamics of coherent structures*, Quart. Appl. Math., 45 (1987), pp. 561–571.
[35] U. von Luxburg, *A tutorial on spectral clustering*, Stat. Comput., 17 (2007), pp. 395–416.
[36] K. Washabaugh, D. Amsallem, M. Zahr, and C. Farhat, *Nonlinear model reduction for CFD problems using local reduced-order bases*, in Proceedings of the 42nd AIAA Fluid Dynamics Conference and Exhibit, Fluid Dynamics and Co-located Conferences, 2012.
[37] D. Wirtz, D. Sorensen, and B. Haasdonk, *A-posteriori error estimation for DEIM reduced nonlinear dynamical systems*, SIAM J. Sci. Comput., submitted.
[38] Y. B. Zhou, *Model Reduction for Nonlinear Dynamical Systems with Parametric Uncertainties*, M. S. thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 2012.