A Training Set Subsampling Strategy for the Reduced Basis Method

Sridhar Chellappa[1] · Lihong Feng[1] · Peter Benner[1]

Abstract

We present a subsampling strategy for the offline stage of the Reduced Basis Method. The approach is aimed at bringing down the considerable offline costs associated with using a finely-sampled training set. The proposed algorithm exploits the potential of the pivoted QR decomposition and the discrete empirical interpolation method to identify important parameter samples. It consists of two stages. In the first stage, we construct a low-fidelity approximation to the solution manifold over a fine training set. Then, for the available low-fidelity snapshots of the output variable, we apply the pivoted QR decomposition or the discrete empirical interpolation method to identify a set of sparse sampling locations in the parameter domain. These points reveal the structure of the parametric dependence of the output variable. The second stage proceeds with a subsampled training set containing a by far smaller number of parameters than the initial training set. Different subsampling strategies inspired from recent variants of the empirical interpolation method are also considered. Tests on benchmark examples justify the new approach and show its potential to substantially speed up the offline stage of the Reduced Basis Method, while generating reliable reduced-order models.

✉ Sridhar Chellappa
chellappa@mpi-magdeburg.mpg.de

Lihong Feng
feng@mpi-magdeburg.mpg.de

Peter Benner
benner@mpi-magdeburg.mpg.de

1   Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstraße 1, 39106 Magdeburg, Germany

🖄 Springer

1 Introduction

Mathematical models in many areas of science and engineering are described by parametric ordinary differential equations (ODEs) or partial differential equations (PDEs). The analytical solution of such ODEs and PDEs is usually intractable, therefore engineers prefer detailed numerical simulations of the models to obtain useful insights into the underlying processes. However, a detailed simulation of the model necessitates a very fine discretization of the governing ODEs or PDEs, both in space and time. The resulting discretized system of equations is referred to as the high-fidelity model or the full-order model (FOM). Numerical solutions of the FOM are computationally expensive to obtain, especially when computing resources are limited or when repeated simulations of the FOM are needed. The latter scenario is common in a multi-query task, such as uncertainty quantification, optimization, etc. With the goal of speeding up simulations, the area of model order reduction (MOR) has gained popularity in the last decades [3,4,7–9]. Different MOR algorithms have been proposed to obtain surrogate models for the FOMs. These surrogates are often referred to as reduced-order models (ROMs) since they have a reduced number of degrees of freedom compared to the FOMs. The simulation results of a suitably developed ROM are indistinguishable from those of the FOM, yet, the ROM incurs only a fraction of the computational cost spent on the FOM. This is a big advantage in real-time simulation or multi-query scenarios. Generally, there exist two classes of ROMs: projection-based ROMs [3] which are obtained by projecting the original model onto an approximation subspace, and data-driven ROMs, which are obtained through a data fit [47]. In the following, we shall limit ourselves to projection-based ROMs.

The Reduced Basis Method (RBM) is a popular, projection-based technique to obtain ROMs of parametric PDEs. The ansatz underlying the RBM is that the solution manifold of the parametric FOM can be well-approximated by a low-dimensional subspace. To achieve efficient online simulation, considerable efforts are made to explore the parameter space and collect solution snapshots in the offline stage. The greedy algorithm explores the parameter space based on suitable *a posteriori* error estimators. At each iteration of the greedy algorithm, the parameter maximizing the estimated error over a training set is selected and the corresponding FOM is solved to obtain new snapshots. The training set is, in essence, a discrete representation of the parameter domain. The snapshots are used to construct a basis for the low-dimensional approximation space, followed by a (Petrov-)Galerkin projection to obtain the ROM. We refer to the recent books [32,49] and the survey works [19,50,53] for a detailed exposition of the theoretical framework underpinning the RBM and also for applications of the RBM in a variety of problems.

The choice of the training set is critical for the success of the RBM. A poorly-sampled training set can result in an inadequate representation of all the solution modes, causing the ROM to fail to meet the desired tolerance criterion for a parameter not present in the training set. Therefore, it is common practice to adopt a finely-sampled training set. However, the computational cost of the offline stage scales with the cardinality of the training set, which becomes high for problems with high-dimensional parameter space. Therefore, a more efficient sampling strategy is desired.

Many works have attempted to address the issue of optimal training set sampling. Notable among them are: the Multi-Stage Greedy algorithm from [55] and the Adaptively Enriching Greedy algorithm in [33]. In the former, the author suggests performing a set of greedy algorithms over randomly sampled training sets; then the resulting ROM is tested over a much larger random training set and the greedy algorithm is re-run on those points failing to meet the tolerance criterion in the larger training set. In the latter work, the authors propose

a saturation criterion which is used to systematically remove parameters from a randomly-sampled training set. New random parameters are then added to the current training set. A larger training set serves as a safety check mechanism at every iteration. However, it may not be efficient, in general, to estimate a robust saturation criterion. The authors of [23,30] propose a localized RBM approach, where a hierarchical tree-based partitioning of the parameter domain is done and separate ROMs for each partition are generated. In [34], the authors consider a two-stage approach that uses the ANOVA expansion together with parameter domain decomposition to address training set complexity. The work [37] considers an anisotropic sampling of the parameter domain using an empirical norm derived from the truncated Hessian of the solution vector with respect to the parameter. No explicit partition of the parameter domain is considered. However, the basis vectors are determined in the online stage. Moreover, the method needs to compute the Hessian at each point in the training set in order to define a distance metric which is subsequently used to add more samples to the training set. The calculation of the Hessian can be very expensive, especially for non-stationary problems. The method proposed in [36] makes only a subset of the finely sampled training set active at a given iteration of the greedy algorithm. A recent extension of this work [35] proposes hybrid strategies combining the ideas from [33,55]. Different strategies are proposed to identify the set of active parameters. The works [17,43] propose a cheap surrogate model for a certain error estimator, based on Kriging and radial basis functions, respectively, and use it to obtain the estimated error for any parameter in a fine training set. A sparse grid-based construction for the training set is suggested in [48].

The authors of [18] perform an eigendecomposition of the Hessian matrix of the output variable with respect to the parameter to identify a small subspace of the high-dimensional parameter space by truncation. The parameters that constitute the training set for the RBM are then sampled from the identified eigenspace. In [56], the so-called *active subspace* [20] of the parameter space is identified by relying on gradient information of the output with respect to the parameter. Both these works are limited to scalar-valued outputs and steady-state problems.

Most of the existing work related to adaptive training set sampling focuses on steady-state or quasi steady-state problems. To the best of our knowledge, only [17,23,30,43] address training set adaptivity for time-dependent problems. The works [23,30] propose a localization strategy that involves constructing multiple ROMs over local parameter domains, while [17, 43] consider adaptively enriching a coarse training set by observing a cheap error surrogate over a fine training set.

In this work, we present a goal-oriented training set sampling strategy that relies on the output quantity of interest (QoI). We aim at identifying the structure of the parameter dependency of the output through the empirical interpolation algorithm [5,15] or a pivoted QR decomposition and utilize this information to find out the parameter importance. Our proposed method is applicable to both steady-state and time-dependent problems with vector-valued outputs. Our central contribution is a two-stage algorithm to control the cardinality of the training set. In the first stage, a coarse RB approximation of the problem is obtained using a fine training set. Then, an approximate output snapshot matrix is derived by time integrating the coarse ROM at all the parameter samples in the fine training set. We apply the pivoted QR decomposition or, alternatively, the discrete empirical interpolation method (DEIM) (or its variants) to the approximate output snapshot matrix. This procedure identifies regions of the parameter space that have a greater contribution to the current RB approximation space. In the second stage, the fine training set is subsampled based on the parameter distribution identified using the pivoted QR decomposition or the DEIM algorithm and leads to a subsampled coarse

training set. The RBM is continued over the coarse training set, until a targeted error tolerance is met.

The paper is organized as follows. In Sect. 2 we describe the preliminaries including the problem setting for the proposed methodology, the RBM and the related hyper-reduction techniques. In Sect. 3 we detail the issue of training set sampling for the RBM and present our main algorithm for efficient training set subsampling. Section 4 is dedicated to numerical examples through which we illustrate various aspects of the proposed subsampling strategy and demonstrate the speedup it offers for two numerical benchmark problems. We conclude by summarizing the proposed method and highlighting possible research directions for the near future. Throughout this work, we have used MATLAB® notation in the presentation of algorithms and numerical experiments.

2 Preliminaries

In this section, we present the continuous problem and the discretized system that the proposed subsampling strategy is valid for. Then, we briefly review the RBM and the associated issue of training set sampling. Afterwards, some hyper-reduction algorithms are reviewed in order to introduce our proposed subsampling algorithms.

2.1 High-fidelity Models

A wide variety of physical and engineering phenomena are modelled via PDEs. Consider the spatial domain $\Omega \subset \mathbb{R}^d$ with ($d = 1, 2, 3$). Let a model of PDEs defined in Ω be denoted by

$$\mathcal{L}\left(\mathbf{v}, \mathbf{w}, t, \boldsymbol{\mu}\right) = \mathbf{0}, \tag{2.1}$$

where $\mathbf{v}$ is the (vector-valued) state variable and describes the particular physical quantity the PDE models, $\mathbf{w} \in \Omega$ is the spatial variable, $0 \leq t \leq T$ denotes the time and $\boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^p$ defines the parameters. The above form is a general description of any time-dependent or steady-state problem with or without parameter variations. The output of the model is usually a function of the solution $\mathbf{v}$ and the parameter $\boldsymbol{\mu}$. After numerical discretization in space and time, we get

$$\mathbf{E}\,\mathbf{x}(t^k, \boldsymbol{\mu}) = \mathbf{A}\,\mathbf{x}(t^{k-1}, \boldsymbol{\mu}) + \mathbf{f}\left(\mathbf{x}(t^{k-1}, \boldsymbol{\mu}), \boldsymbol{\mu}\right) + \mathbf{B}\,\mathbf{u}(t^{k-1}, \boldsymbol{\mu}),$$
$$\mathbf{y}\left(\mathbf{x}(t^k, \boldsymbol{\mu}), \boldsymbol{\mu}\right) = \mathbf{C}\,\mathbf{x}(t^k, \boldsymbol{\mu}). \tag{2.2}$$

Here, $\mathbf{x}(t^k, \boldsymbol{\mu}) \in \mathbb{R}^n$ is the state vector, $\mathbf{u}(t^k, \boldsymbol{\mu}) \in \mathbb{R}^m$ is the input vector and $\mathbf{y}\left(\mathbf{x}(t^k, \boldsymbol{\mu}), \boldsymbol{\mu}\right) \in \mathbb{R}^q$ is the output or quantity of interest. Further, $\mathbf{E}, \mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times m}$ is the input matrix, $\mathbf{C} \in \mathbb{R}^{q \times n}$ is the output matrix and $\mathbf{f}\left(\mathbf{x}(t^k, \boldsymbol{\mu}), \boldsymbol{\mu}\right) \in \mathbb{R}^n$ models the nonlinearity associated with the system.

Remark 2.1 The system matrices ($\mathbf{E}, \mathbf{A}, \mathbf{B}$ and $\mathbf{C}$) can also be time- and/or parameter-dependent. However, we have not made this dependence explicit for the sake of keeping the notations concise. For the case of steady-state problems, the time dependence of the state, input and output vectors and system matrices vanishes and the system simply reads

$$\mathbf{E}\,\mathbf{x}(\boldsymbol{\mu}) = \mathbf{f}\left(\mathbf{x}(\boldsymbol{\mu}), \boldsymbol{\mu}\right) + \mathbf{B}\,\mathbf{u}(\boldsymbol{\mu}),$$
$$\mathbf{y}(\mathbf{x}(\boldsymbol{\mu}), \boldsymbol{\mu}) = \mathbf{C}\,\mathbf{x}(\boldsymbol{\mu}). \tag{2.3}$$

The discretized system in (2.2) or (2.3) is also called the FOM and has a large number of degrees of freedom, i.e., n is very large. The proposed subsampling strategy is applicable to both Eqs. (2.2) and (2.3).

2.2 Reduced Basis Method and the Training Set

The Reduced Basis Method relies on the observation that the solution manifold can be well-approximated by a *low-dimensional* subspace $\mathcal{V}$. Let $[\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_r] =: \mathbf{V} \in \mathbb{R}^{n \times r}$ be a basis of the subspace $\mathcal{V}$. The approximated solution for any parameter is obtained by considering the ansatz $\mathbf{x}(t^k, \boldsymbol{\mu}) \approx \widetilde{\mathbf{x}}(t^k, \boldsymbol{\mu}) = \sum_{i=1}^{r} z_i \mathbf{v}_i$. The parameter-dependent coefficients $\mathbf{z} = [z_1, z_2, \ldots, z_r]^\mathsf{T}$ are obtained by solving the ROM

$$
\begin{aligned}
\mathbf{E}_r \, \mathbf{z}(t^k, \boldsymbol{\mu}) &= \mathbf{A}_r \, \mathbf{z}(t^{k-1}, \boldsymbol{\mu}) + \mathbf{f}_r \left(\mathbf{V}\mathbf{z}(t^{k-1}, \boldsymbol{\mu}), \boldsymbol{\mu} \right) + \mathbf{B}_r \, \mathbf{u}(t^{k-1}, \boldsymbol{\mu}), \\
\widetilde{\mathbf{y}} \left(\mathbf{z}(t^k, \boldsymbol{\mu}), \boldsymbol{\mu} \right) &= \mathbf{C}_r \, \mathbf{z}(t^k, \boldsymbol{\mu}),
\end{aligned}
\tag{2.4}
$$

derived through Galerkin projection of the FOM onto the low-dimensional subspace $\mathcal{V}$. The reduced system matrices $\mathbf{E}_r$, $\mathbf{A}_r \in \mathbb{R}^{r \times r}$, $\mathbf{B}_r \in \mathbb{R}^{r \times m}$ and $\mathbf{C}_r \in \mathbb{R}^{q \times r}$ are obtained as $\mathbf{E}_r := \mathbf{V}^\mathsf{T}\mathbf{E}\mathbf{V}$, $\mathbf{A}_r := \mathbf{V}^\mathsf{T}\mathbf{A}\mathbf{V}$, $\mathbf{B}_r := \mathbf{V}^\mathsf{T}\mathbf{B}$ and $\mathbf{C}_r := \mathbf{C}\mathbf{V}$, respectively. Finally, $\mathbf{f}_r := \mathbf{V}^\mathsf{T}\mathbf{f}\left(\mathbf{V}\mathbf{z}(t^{k-1}, \boldsymbol{\mu}), \boldsymbol{\mu}\right)$.

The greedy algorithm or the POD-greedy (Proper Orthogonal Decomposition-greedy) algorithm are the most popular techniques for constructing the RBM approximation space for the steady-state system (2.3) and the time-dependent system (2.2), respectively. In order to initialize the greedy algorithm, a training set $\varXi_{\text{train}}$ is given *a priori*, from which parameter samples are iteratively selected, so that the corresponding solution vector is iteratively added to the basis matrix $\mathbf{V}$. We summarize the (POD-)greedy algorithm in Algorithm 2.1, which takes both the steady-state case and the time-dependent case into consideration. In Step 8 of Algorithm 2.1, the snapshot matrix $\mathfrak{X}$ for the time-dependent case consists of the solution vector at discretized time instances $\{t^i\}_{i=0}^{K}$ given by:

$$
\mathfrak{X}(\boldsymbol{\mu}) = \left[\mathbf{x}(t^0, \boldsymbol{\mu}) \cdots \mathbf{x}(t^K, \boldsymbol{\mu}) \right] \in \mathbb{R}^{n \times N_t},
$$

where $N_t := (K+1)$. $\mathbf{U}_{\widetilde{\mathfrak{X}}}$ in Step 10 is the left singular vector matrix obtained from the singular value decomposition (SVD) of $\bar{\mathfrak{X}}$, i.e., $\bar{\mathfrak{X}} = \mathbf{U}_{\bar{\mathfrak{X}}} \varSigma_{\bar{\mathfrak{X}}} \mathbf{V}_{\bar{\mathfrak{X}}}^\mathsf{T}$. Here, $\varSigma_{\bar{\mathfrak{X}}}$ contains all the non-zero singular values of $\bar{\mathfrak{X}}$: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{r_X} \geq 0$. If no alternative decision criterion is used, r_{POD} is usually taken to be 1. In Step 13, $\Delta(\boldsymbol{\mu})$ denotes an error estimator for the error in approximating the state variable or the output variable. The error estimator should be much cheaper to compute than the true error. For the sake of clarity, the sketched algorithm is the basic version of the RBM. Several enhancements in the form of primal-dual error estimation, hyper-reduction, adaptive basis construction, etc. exist [6,16,26,28,58].

It is noticed that the standard greedy algorithm (Algorithm 2.1) does not address the issue of properly choosing $\varXi_{\text{train}}$. When the parameter space dimension is high ($p \gg 2$) or when the number of time steps N_t is large, the overall computational cost for the greedy algorithm to construct $\mathbf{V}$ can be substantial. The technique considered in this work aims at reducing the offline cost by subsampling a fine training set. We detail this in Sect. 3. Before that, since it will be used later, we briefly review two hyper-reduction procedures—the DEIM algorithm (and its variants) and the Gappy-POD method.

Algorithm 2.1 Reduced Basis Method (RBM)

Input: Training set (Ξ), tolerance for the ROM (`tol`), maximum iterations (n_{max}).
Output: V.
1: Initialize $\mathbf{V} = [\ ], \epsilon = 1 + \texttt{tol}, \text{iter} = 1$.
2: Select $\boldsymbol{\mu}^*_{(1)}$ randomly from Ξ_{train}.
3: **while** $\epsilon > \texttt{tol}$ && $\text{iter} \leq n_{max}$ **do**
4:   **if** steady-state **then**
5:     Solve FOM (2.3) at $\boldsymbol{\mu}^*_{(\text{iter})}$ and obtain solution $\mathbf{x}(\boldsymbol{\mu}^*_{(\text{iter})})$.
6:     Set $\mathbf{V} \leftarrow \texttt{orth}\left(\left[\mathbf{V} \ \mathbf{x}(\boldsymbol{\mu}^*_{(\text{iter})}) \right] \right)$.   % `orth` *means orthogonalizing* $\mathbf{x}(\boldsymbol{\mu}^*_{(\text{iter})})$ *against the*
       column vectors in the current matrix $\mathbf{V}$.
7:   **else**
8:     Solve FOM (2.2) at $\boldsymbol{\mu}^*_{(\text{iter})}$ and obtain snapshot matrix $\boldsymbol{\mathfrak{X}}$.
9:     Set $\bar{\boldsymbol{\mathfrak{X}}} = \boldsymbol{\mathfrak{X}} - \mathbf{V}\mathbf{V}^\mathsf{T}\boldsymbol{\mathfrak{X}}$.
10:     Enrich $\mathbf{V}$ with r_{POD} left singular vectors of $\bar{\boldsymbol{\mathfrak{X}}}$ as
       $\mathbf{V} \leftarrow \texttt{orth}\left(\left[\mathbf{V} \ \mathbf{U}_{\bar{\boldsymbol{\mathfrak{X}}}}(:, 1 : r_{POD}) \right] \right)$.
11:   **end if**
12:   $\text{iter} = \text{iter} + 1$.
13:   Set $\boldsymbol{\mu}^*_{(\text{iter})} = \arg\max_{\boldsymbol{\mu} \in \Xi_{train}} \Delta(\boldsymbol{\mu})$.
14:   $\epsilon = \Delta(\boldsymbol{\mu}^*_{(\text{iter})})$.
15: **end while**

2.3 Discrete Empirical Interpolation Method

The DEIM algorithm [5,15,26] was introduced in the context of MOR, for efficient calculation of nonlinear or nonaffine terms of the ROM. The algorithm proceeds by collecting snapshots of the nonlinear vector in the FOM in (2.2) or (2.3) for different values of $\boldsymbol{\mu} \in \Xi_{train}$ given by

$$\boldsymbol{\mathfrak{F}} = \left[\mathbf{f}(\mathbf{x}, \boldsymbol{\mu}_1) \cdots \mathbf{f}(\mathbf{x}, \boldsymbol{\mu}_{N_{train}}) \right] \in \mathbb{R}^{n \times N_{train}}.$$

In order to avoid any n-dependent operations to evaluate the nonlinear vector involved in simulating the ROM, DEIM considers an approximation of the nonlinear vector given by

$$\mathbf{f}(\mathbf{x}, \boldsymbol{\mu}) \approx \widetilde{\mathbf{f}}(\mathbf{x}, \boldsymbol{\mu}) = \mathbf{U}\boldsymbol{\alpha}, \tag{2.5}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^\ell$. The columns of $\mathbf{U} \in \mathbb{R}^{n \times \ell}$ are the interpolation basis vectors obtained via SVD of $\boldsymbol{\mathfrak{F}}$. The number of basis vectors ℓ can be determined through an information-theoretic criterion that depends on the relative energy content of the singular values $\{\sigma_i\}_{i=1}^{r_X}$ and reads $\frac{\sum_{i=\ell+1}^{r_X} \sigma_i}{\sum_{i=1}^{r_X} \sigma_i} < \epsilon_{SVD}$ where ϵ_{SVD} is a user-defined tolerance. Since (2.5) is an overdetermined system with $n \gg \ell$, DEIM solves for $\boldsymbol{\alpha}$ by selecting ℓ rows from $\mathbf{U}$ and enforces interpolation as below:

$$\mathbf{P}^\mathsf{T}\mathbf{f}(\mathbf{x}, \boldsymbol{\mu}) = \mathbf{P}^\mathsf{T}\mathbf{U}\boldsymbol{\alpha}. \tag{2.6}$$

The indices of the rows where interpolation is enforced are given by $\{p_1, \dots, p_\ell\}$. The ith column denoted as $e_{p_i} \in \mathbb{R}^n$ of the matrix $\mathbf{P} \in \mathbb{R}^{n \times \ell}$ is essentially the ith canonical unit vector with zeros at all but the p_i^{th} entry. A greedy procedure, shown in Algorithm 2.2, is used to identify $\mathbf{P}$. The algorithm ensures that $(\mathbf{P}^\mathsf{T}\mathbf{U})$ is nonsingular, so that $\widetilde{\mathbf{f}}(\mathbf{x}, \boldsymbol{\mu})$ at any

---

Algorithm 2.2 Discrete Empirical Interpolation Method (DEIM) as in [15]

Input: Snapshots of the nonlinear vector ($\mathfrak{F}$), tolerance for the SVD (ϵ_{SVD}).
Output: **U**, **P** and **I**.
1: Perform SVD of $\mathfrak{F}$; collect the first ℓ left singular vectors $\{\mathbf{u}_j\}_{j=1}^{\ell} \subset \mathbb{R}^n$.
2: Set $[\sim, p_1] = \max(|\mathbf{u}_1|)$.   % $|\cdot|$ *denotes the absolute value.*
3: Set $\mathbf{U} = \mathbf{u}_1$, $\mathbf{I} = p_1$ and $\mathbf{P} = [e_{p_1}]$.
4: **for** $i = 2$ to ℓ **do**
5:     Solve $\mathbf{P}^\mathsf{T}\mathbf{U}\boldsymbol{\alpha} = \mathbf{P}^\mathsf{T}\mathbf{u}_i$.
6:     Define residual $\mathbf{r} = \mathbf{u}_i - \mathbf{U}\boldsymbol{\alpha}$.
7:     Set $[\sim, p_i] = \max(|\mathbf{r}|)$.
8:     Update $\mathbf{U} := [\mathbf{U}\ \mathbf{u}_i]$, $\mathbf{I} := [\mathbf{I}\ p_i]$ and $\mathbf{P} := [\mathbf{P}\ e_{p_i}]$.
9: **end for**

---

parameter $\boldsymbol{\mu}$ is given by

$$\widetilde{\mathbf{f}}(\mathbf{x}, \boldsymbol{\mu}) = \mathbf{U}\left(\mathbf{P}^\mathsf{T}\mathbf{U}\right)^{-1}\mathbf{P}^\mathsf{T}\mathbf{f}(\mathbf{x}, \boldsymbol{\mu}),$$

where $\mathbf{U}\left(\mathbf{P}^\mathsf{T}\mathbf{U}\right)^{-1} \in \mathbb{R}^{n \times \ell}$ can be precomputed and stored. Moreover, the original nonlinear vector needs to be evaluated only at ℓ points, limiting the cost of evaluating the nonlinear vector to order ℓ and independent of n. The error in approximating the nonlinear vector is quantified as

$$\left\|\mathbf{f}(\mathbf{x}, \boldsymbol{\mu}) - \widetilde{\mathbf{f}}(\mathbf{x}, \boldsymbol{\mu})\right\|_2 \leq \left\|\left(\mathbf{P}^\mathsf{T}\mathbf{U}\right)^{-1}\right\|_2 \|\mathbf{f}(\mathbf{x}, \boldsymbol{\mu}) - \mathbf{U}\mathbf{U}^\mathsf{T}\mathbf{f}(\mathbf{x}, \boldsymbol{\mu})\|_2. \tag{2.7}$$

The greedy choice of the interpolation indices in the DEIM algorithm is geared towards minimizing the term $\left\|\left(\mathbf{P}^\mathsf{T}\mathbf{U}\right)^{-1}\right\|_2$ appearing in the error bound. At each iteration, the new sampling point is chosen as the one resulting in the maximum reduction in $\left\|\left(\mathbf{P}^\mathsf{T}\mathbf{U}\right)^{-1}\right\|_2$.

Several variants of the DEIM algorithm have been proposed [21,41,44–46]. Among those, the QDEIM algorithm from [21] and the DEIM-based oversampling strategies proposed in [45] shall be of particular interest to our discussion in Sect. 3.

2.3.1 QDEIM

In contrast to the original DEIM algorithm, the QDEIM approach from [21] relies on a column-pivoted QR decomposition to identify the interpolation points. This is different from the sequential, greedy choice of interpolation points in DEIM (see Steps 4–9 in Algorithm 2.2). QDEIM is proven to have a sharper error bound and is also computationally more efficient and straightforward to implement.

2.3.2 KDEIM

The K in the KDEIM algorithm refers to the k-means clustering algorithm. The k-means algorithm is applied to the matrix $\mathbf{U}$ of (truncated) left singular vectors obtained from SVD of the snapshot matrix $\mathfrak{F}$, then rows with similar response are assigned to the same cluster. The standard k-means objective function is recast as a relaxed trace maximization problem which is then solved using the QR decomposition. We refer the interested reader to [45] for a deeper discussion. Another early work to consider QR decomposition based clustering in the context of MOR was [39], which used it for reducing networked multi-agent systems.

2.3.3 Gappy-POD

The number of interpolation points of the DEIM algorithm and its variants discussed so far equals to the number ℓ of interpolation basis vectors. However, in many cases it is beneficial to consider $m > \ell$ interpolation points. The Gappy-POD method and other related approaches fall into this category [12,24]. The coefficient matrix $\mathbf{P}^\mathsf{T}\mathbf{U}$ of the linear system in (2.6) is no longer square and, therefore, does not possess a unique inverse; instead it is solved using the pseudoinverse. The Gappy-POD approximation of the nonlinear vector is given as

$$\widetilde{\mathbf{f}}\left(\mathbf{x}, \boldsymbol{\mu}\right) = \mathbf{U}\left(\mathbf{P}^\mathsf{T}\mathbf{U}\right)^\dagger \mathbf{P}^\mathsf{T}\mathbf{f}\left(\mathbf{x}, \boldsymbol{\mu}\right), \tag{2.8}$$

where the matrix $\mathbf{P}$ now has $m > \ell$ columns and we have $\mathbf{P}^\mathsf{T}\mathbf{U} \in \mathbb{R}^{m \times \ell}$. In [45], the authors discuss two different oversampling strategies. The first approach, called Gappy-POD Eigenvector, considers the optimization point of view – newly added interpolation points are those that lead to the largest decrease of $\| \left(\mathbf{P}^\mathsf{T}\mathbf{U}\right)^\dagger \|$. The second oversampling strategy, Gappy-POD Clustering, proposed in [45] can be viewed as Gappy-POD based on interpreting the QR decomposition as a clustering algorithm. The additional samples from $\ell + 1$ till m are identified based on the mutual entropy of the columns. For a more elaborate discussion we refer to [45,57]. In the next section, we aim to make use of DEIM and its variants, as well as Gappy-POD to select *important* parameter samples from the parameter domain.

For the pseudocode and the details, we refer the reader to the work [21] for the QDEIM algorithm and the work [45] in case of the KDEIM and Gappy-POD algorithms.

3 Proposed Subsampling Strategy for the Training Set

A representative training set $\varXi_{\text{train}}$ is crucial for RBM to obtain a ROM that satisfies the required tolerance. While a densely-sampled $\varXi_{\text{train}}$ is needed to accurately represent the parameter space, it incurs high computational cost. In contrast, a randomly sampled coarse training set may fail to capture all the variations of the solution over the parameter space and result in a ROM that fails to meet the tolerance. Therefore, a wisely sampled coarse training set is desired to make the greedy algorithm efficient while retaining the required accuracy of the ROM. We now discuss two observations that motivate our proposed approach for training set sampling.

3.1 Motivating Observations

We detail two observations that pertain to the greedy algorithm in the RBM, the DEIM algorithm as well as the QR pivoting. We shall see that these two observations have motivated us to develop a subsampling strategy for the RBM training set.

3.1.1 Greedy Parameters, QR Pivots, and DEIM Interpolation Points

Our first observation concerns the parameters $\boldsymbol{\mu}^*$ selected by the greedy algorithm. The second observation is their resemblance to the QR pivots and the DEIM interpolation points.

From our experience, the greedy algorithm tends to repeatedly pick parameter samples from a small subset of the training set, especially for time-dependent problems. This same phenomenon has been reported in other existing works [27,29,31,37,42,50,58]. The solution

(or output) vectors at these parameter values usually exhibit large variability. While the greedy algorithm scans through all the parameter samples in the training set, a majority of those samples are never picked. The fact that a few parameters get repeatedly picked reveals that there are still unresolved modes and hence more POD modes, corresponding to the selected parameter, are needed to get a good approximation. These few parameters picked by the greedy algorithm, usually represent solutions that are less smooth and hence are more difficult to approximate. An example of this phenomenon occurs in fluid dynamics problems where the low viscosity solutions develop shock and need a large number of POD modes to approximate. We illustrate this observation through the standard greedy algorithm applied to the discretized 1-D viscous Burgers' equation with $n = 1000$. The detailed description of the model is presented in Sect. 4. The parameter considered is the viscosity. We use 100 equispaced parameter samples from the domain $\mathcal{P} = [0.005\,,\, 1]$ to form the training set Ξ_{train}. A ROM with error below the tolerance $\texttt{tol} = 10^{-6}$ is requested from the greedy algorithm. In Table 1a, we provide the parameters picked by the greedy algorithm at each iteration from the training set. Noticeably, among the 100 parameter samples in the training set, only 6 contribute to generating the basis $\mathbf{V}$ that approximates the solution manifold. Of these 6 samples, the sample $\mu = 0.005$ is picked fourteen times. This is not surprising since this parameter corresponds to the solution vector with the smallest viscosity and is the most difficult to approximate.

Next, we make an important connection between the parameters selected by the greedy algorithm and the pivots obtained through a pivoted QR decomposition of the transpose of the output snapshot matrix defined in (3.1).

For the same viscous Burgers' equation, we collect the snapshots of the scalar-valued outputs $\mathbf{y}$ at all the parameters in Ξ_{train} into a snapshot matrix given by

$$
\mathfrak{Y} := \begin{bmatrix} [\mathbf{y}(\boldsymbol{x}(t^0, \boldsymbol{\mu}_1), \boldsymbol{\mu}_1)]^{\mathsf{T}} & \cdots & [\mathbf{y}(\boldsymbol{x}(t^K, \boldsymbol{\mu}_1), \boldsymbol{\mu}_1)]^{\mathsf{T}} \\ \vdots & \ddots & \vdots \\ [\mathbf{y}(\boldsymbol{x}(t^0, \boldsymbol{\mu}_{N_{\text{train}}}), \boldsymbol{\mu}_{N_{\text{train}}})]^{\mathsf{T}} & \cdots & [\mathbf{y}(\boldsymbol{x}(t^K, \boldsymbol{\mu}_{N_{\text{train}}}), \boldsymbol{\mu}_{N_{\text{train}}})]^{\mathsf{T}} \end{bmatrix}, \tag{3.1}
$$

Each row of the matrix consists of the snapshots of the output at $K + 1$ time instances corresponding to a given parameter. Consider first the well-known pivoted QR decomposition of a matrix $\mathbf{D}$ given by

$$
\mathbf{D}\Pi = \mathbf{Q}\mathbf{R} = \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ 0 & \mathbf{R}_{22} \end{bmatrix}, \tag{3.2}
$$

where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is upper triangular. The pivots are given by the column permutation matrix Π. We apply the QR decomposition to $\mathfrak{Y}^{\mathsf{T}}$ and identify the pivots. A comparison of the parameters corresponding to the first ten pivots and the parameters selected in the greedy algorithm is shown in Table 1 and Fig. 1. Of the ten pivots, six are identical with the greedy parameters. This close connection between the pivots of the QR decomposition and the greedy parameters chosen in the RBM has been, to the best of our knowledge, discussed only in [2,40]. It is based on the interpretation of the QR decomposition as a greedy column selection procedure. Note that the application of a QR decomposition assumes the existence of the FOM solution for all the parameters in the training set. In practice, we do not have this information. Instead, we propose to apply the pivoted QR decomposition to the transpose of an approximate output snapshot matrix, in order to identify *important* parameters which can then be used to subsample the fine training set in the RBM.

In Sect. 2.3, the usage of DEIM and QDEIM algorithms were discussed in the context of MOR. The DEIM algorithm uses a greedy sparse sampling of the left singular vector matrix ($\mathbf{U}$) of the snapshots to identify interpolation points, whereas QDEIM performs a

Table 1 Greedy parameters and QR Pivots for the Burgers' equation

(a) Greedy parameters picked by RBM.

Parameter	0.005	0.0151	0.0251	0.0352	0.0553	1
Repetitions	14	1	1	1	1	1

(b) First 10 pivots for the QR decomposition of the transposed true output snapshot matrix $\mathfrak{Y}$.

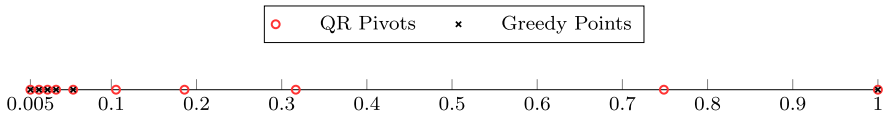Pivots 0.005 0.0151 0.0251 0.0352 0.0553 0.1055 0.1859 0.3166 0.7487 1

Fig. 1 Greedy parameters for the Burgers' equation and QR pivots of the true output snapshots matrix $\mathfrak{Y}$

QR decomposition with column pivoting on $\mathbf{U}^{\mathsf{T}}$. This implicates a similar phenomenon as observed above: QR with pivoting could select points of importance on different demands. It is also noticed that QR with pivoting connects the greedy algorithm with DEIM (QDEIM), which indicates that DEIM and QDEIM could also be used to select representative parameter samples if either is applied to the output snapshot matrix.

Figure 1 shows that the pivots of the QR decomposition on $\mathfrak{Y}^{\mathsf{T}}$ gives similar points as those selected by the greedy algorithm. It then indicates that sample points selected by the greedy algorithm in a way are highly related to the interpolation points of QDEIM, if the same snapshot matrix is considered by both the greedy algorithm and QDEIM, which is, in our case, the output snapshot matrix $\mathfrak{Y}$. By exploiting this interpretation, we propose to use DEIM or other variants of DEIM in order to adapt the training set during the greedy algorithm.

To further support and motivate our proposed scheme, in the next subsection, we show that DEIM also has the similar capability of identifying the most representative parameter samples for dynamics, as that exhibited by the greedy algorithm in the RBM.

3.1.2 DEIM and Parametric Anisotropy

For a function of two variables $\mathbf{f}(\mathbf{x}, \boldsymbol{\mu}) : \mathbb{R}^n \times \mathbb{R}^p \rightarrow \mathbb{R}^n$, the DEIM algorithm first identifies a linear subspace $\mathbf{U}$ and a small subset of points in the $\mathbf{x}$ variable, based on the snapshots matrix $\mathfrak{F}$. One can analogously consider the mapping $\mathbf{f}(\boldsymbol{\mu}, \mathbf{x}) : \mathbb{R}^p \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ through a transpose of $\mathfrak{F}$. However, now the DEIM algorithm identifies a small subset of points in the $\boldsymbol{\mu}$ variable. We illustrate this on a toy example from [1]. Consider the following nonlinear, two parameter function:

$$\mathbf{f}(\mathbf{x}_1, \mathbf{x}_2, \boldsymbol{\mu}) = \frac{1 + \frac{\pi^2}{4}(\mu_2 - \mu_1 - (\mu_1 + \mu_2)\mathbf{x}_2)^2 \sin^2(\frac{\pi}{2}(\mathbf{x}_1 + 1))}{1 + (\mu_1 + \mu_2)\cos(\frac{\pi}{2}(\mathbf{x}_1 + 1))}, \quad (3.3)$$

where $\mathbf{x} := [\mathbf{x}_1, \mathbf{x}_2] \in \mathbb{R}^{n \times 2}$ is the spatial variable obtained from the discretization of the two dimensional domain $\Omega := [-1, 1] \times [-1, 1]$ with 50 points in each spatial direction, resulting in $n = 2500$. The parameter $\boldsymbol{\mu} := (\mu_1, \mu_2) \in \mathbb{R}^2$ belongs to the domain $\mathcal{P} := [-0.4, 0.4] \times [-0.4, 0.4]$. We collect 1600 snapshots of the function in the snapshots matrix $\mathfrak{F} \in \mathbb{R}^{2500 \times 1600}$, based on uniform, equally spaced samples of the parameter. In Fig. 2a, the

singular values of the snapshot matrix $\mathfrak{F}$ are plotted. The rapid decay clearly demonstrates the reducibility of this function. We apply a cut-off of $\epsilon_{\text{SVD}} = 10^{-10}$ for Algorithm 2.2 applied to both $\mathfrak{F}$ and $\mathfrak{F}^{\mathsf{T}}$. The greedy interpolation points in Fig. 2b are those corresponding to the indices in the row vector $\mathbf{I}$, obtained from applying Algorithm 2.2 to $\mathfrak{F}$. The greedy points in Fig. 2b are those $\boldsymbol{\mu}$ corresponding to the indices stored in $\mathbf{I}$, obtained from applying Algorithm 2.2 to $\mathfrak{F}^{\mathsf{T}}$. The distribution of the points determined by DEIM for both the spatial and parameter variable have a characteristic structure. The number of interpolation points with SVD truncation tolerance $\epsilon_{\text{SVD}} = 10^{-10}$ was 48, a mere 3% of the total points. The spatial greedy points illustrate that while the variable x_1 is equally important over the entire range of $[-1, 1]$, the x_2 variable has almost all its variation concentrated at $x_2 \in \{-1, 0, 1\}$. For the parameter variable, the greedy algorithm picks most of the samples from the boundary of the domain and from the diagonal going from the lower left to the upper right. There is a dense concentration of points around the corners $(-0.4, -0.4)$ and $(0.4, 0.4)$. The choice of the greedy points is closely related to the structure of the function $\mathbf{f}$ being approximated. In most of the existing MOR literature, the DEIM algorithm has been used mainly as a tool to speed up evaluations of nonlinear or nonaffine (parametric) functions in a ROM. However, through the toy example, we have demonstrated its capability to expose the nature of parametric dependence of a function. As seen in Fig. 2c, it is able to identify the regions in the parameter space where the function has large variations.

3.2 Subsampling the Training Set

Based on the observations in Sects. 3.1.1 and 3.1.2, it is evident that a substantial computational effort can be saved in the offline stage of the RBM if we appropriately (optimally) sample the training set. The rationale for the proposed approach is the following: the standard greedy algorithm scans through the entire training set at each iteration and evaluates the error estimator at each parameter. This approach can incur significant computational cost for training sets with a large number of parameters. The proposed algorithm aims at picking out a small subset of the training set containing the most informative parameters. As will be demonstrated numerically, the parameters match closely to those chosen by the standard greedy algorithm.

Based on our observation of Fig. 1 in Sect. 3.1.1 and Fig. 2 in Sect. 3.1.2, we propose to apply the pivoted QR decomposition and the DEIM algorithm (and its variants) to the snapshot matrix of the approximate output vector $\widetilde{\mathbf{y}}\big(\mathbf{z}(t^k, \boldsymbol{\mu}), \boldsymbol{\mu}\big)$. More specifically, we consider the output snapshot matrix given by

$$
\widetilde{\mathfrak{Y}} := \begin{bmatrix} [\widetilde{\mathbf{y}}(\mathbf{z}(t^0, \boldsymbol{\mu}_1), \boldsymbol{\mu}_1)]^{\mathsf{T}} & \cdots & [\widetilde{\mathbf{y}}(\mathbf{z}(t^K, \boldsymbol{\mu}_1), \boldsymbol{\mu}_1)]^{\mathsf{T}} \\ \vdots & \ddots & \vdots \\ [\widetilde{\mathbf{y}}(\mathbf{z}(t^0, \boldsymbol{\mu}_{N_{\text{train}}}), \boldsymbol{\mu}_{N_{\text{train}}})]^{\mathsf{T}} & \cdots & [\widetilde{\mathbf{y}}(\mathbf{z}(t^K, \boldsymbol{\mu}_{N_{\text{train}}}), \boldsymbol{\mu}_{N_{\text{train}}})]^{\mathsf{T}} \end{bmatrix} \in \mathbb{R}^{N_{\text{train}} \times q N_t} \quad (3.4)
$$

with each row containing the snapshots of the approximated output quantity at $K + 1$ time instances corresponding to a given parameter sample.

Remark 3.1 In case of stead-state systems with a single output we apply the proposed subsampling approach on the approximate state snapshots. For this, we define $\widetilde{\mathfrak{Y}} := \widetilde{\mathfrak{X}}^{\mathsf{T}}$, $\widetilde{\mathfrak{X}}$ being the snapshot matrix of the approximate state vector $(\widetilde{\mathbf{x}}(\boldsymbol{\mu}) = \mathbf{V}\mathbf{z}(\boldsymbol{\mu}))$ such that

$$
\widetilde{\mathfrak{X}} := \big[\widetilde{\mathbf{x}}(\boldsymbol{\mu}_1), \cdots, \widetilde{\mathbf{x}}(\boldsymbol{\mu}_{N_{\text{train}}}) \big] \in \mathbb{R}^{n \times N_{\text{train}}}.
$$

(a) Singular value decay.

(b) Spatial greedy points.
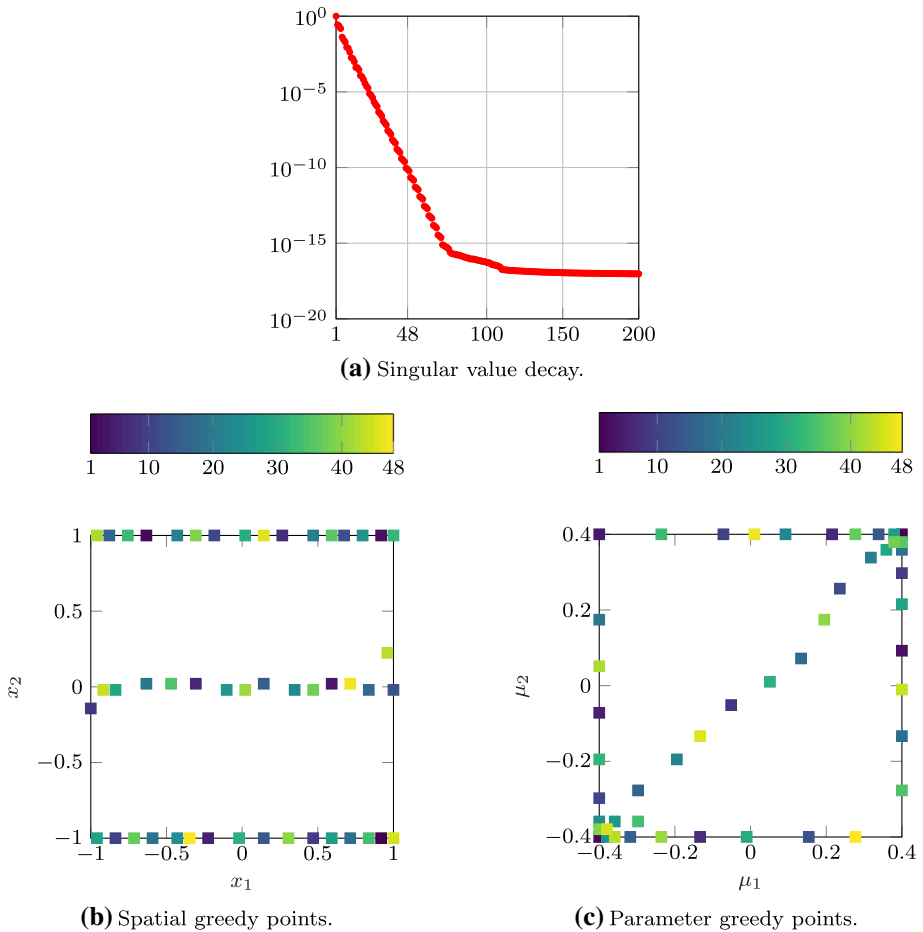
(c) Parameter greedy points.

Fig. 2 Toy problem demonstrating anisotropic choice of interpolation points. The colourbars indicate the order of selection of the parameters. Points in the blue end of the spectrum were selected earlier while those in the green/yellow regions of the spectrum were picked later during the course of the algorithm (Color figure online)

For steady-state systems with multiple outputs, we define $\widetilde{\mathfrak{Y}} := \left[\widetilde{\mathbf{y}}(\boldsymbol{\mu}_1), \cdots, \widetilde{\mathbf{y}}(\boldsymbol{\mu}_{N_{\text{train}}}) \right]^{\mathsf{T}} \in \mathbb{R}^{N_{\text{train}} \times q}$.

Note that $\widetilde{\mathfrak{Y}}$ can be obtained from a coarse or low-fidelity ROM of the original system without doing FOM simulation at all the parameter samples. We propose two sampling strategies: (i) apply pivoted QR decomposition to $\widetilde{\mathfrak{Y}}^{\mathsf{T}}$, (ii) apply DEIM or its variants to $\widetilde{\mathfrak{Y}}$, in order to identify the structure of the parametric dependence of the output variable. Once the distribution of the interpolation points is identified, we can then adapt the training set for subsequent iterations of the greedy algorithm. We now outline the proposed approach and discuss different computational strategies.

The proposed sampling procedure consists of two stages. The first stage is identical to the standard RBM procedure outlined in Algorithm 2.1. A finely sampled training set Ξ_{train}^{f} is used. We consider two different stopping criteria for the first stage—(a) the first stage runs

---

Algorithm 3.1 Reduced Basis Method with Training Set Subsampling (Scheme 1)

Input: Training set (Ξ_{train}^f), tolerance for the ROM (tol), coarse tolerance (tol^c), maximum iteration (n_g).
Output: **V**.
1: Initialize **V** = [ ], $\epsilon = 1 + $tol, iter = 1.
2: Select $\boldsymbol{\mu}_{(1)}^*$ randomly from Ξ_{train}^f.

   Stage 1

3: **while** $\epsilon > $ tol^c **do**
4:    Solve FOM (2.2) at $\boldsymbol{\mu}_{(\text{iter})}^*$ and obtain snapshot matrix $\mathfrak{X}$.
5:    Set $\bar{\mathfrak{X}} = \mathfrak{X} - \mathbf{V}\mathbf{V}^\mathsf{T}\mathfrak{X}$.
6:    Enrich **V** with r_{POD} left singular vectors of $\bar{\mathfrak{X}}$ as
     $\mathbf{V} \leftarrow \text{orth}\left(\left[\mathbf{V}\ \mathbf{U}_{\bar{\mathfrak{X}}}(:,\ 1\ :\ r_{\text{POD}})\right]\right)$.
7:    iter = iter + 1.
8:    Set $\boldsymbol{\mu}_{(\text{iter})}^* = \arg\max_{\boldsymbol{\mu} \in \Xi_{\text{train}}^f} \Delta(\boldsymbol{\mu})$.
9:    $\epsilon = \Delta(\boldsymbol{\mu}_{(\text{iter})}^*)$.
10: **end while**

   Stage 2

11: Perform pivoted QR decomposition of $\widetilde{\mathfrak{Y}}^\mathsf{T}$, or apply DEIM or a DEIM variant to $\widetilde{\mathfrak{Y}}$ and identify the indices
    I of the QR pivots or DEIM interpolation points.
12: Identify new training set Ξ_{train} using distribution of **I**.
13: **while** $\epsilon > $ tol && iter $\leq n_g$ **do**
14:    Solve FOM (2.2) at $\boldsymbol{\mu}_{(\text{iter})}^*$ and obtain snapshot matrix $\mathfrak{X}$.
15:    Set $\bar{\mathfrak{X}} = \mathfrak{X} - \mathbf{V}\mathbf{V}^\mathsf{T}\mathfrak{X}$.
16:    Enrich **V** with r_{POD} left singular vectors of $\bar{\mathfrak{X}}$ as
     $\mathbf{V} \leftarrow \text{orth}\left(\left[\mathbf{V}\ \mathbf{U}_{\bar{\mathfrak{X}}}(:,\ 1\ :\ r_{\text{POD}})\right]\right)$.
17:    iter = iter + 1.
18:    Set $\boldsymbol{\mu}_{(\text{iter})}^* = \arg\max_{\boldsymbol{\mu} \in \Xi_{\text{train}}} \Delta(\boldsymbol{\mu})$.
19:    $\epsilon = \Delta(\boldsymbol{\mu}_{(\text{iter})}^*)$.
20: **end while**

---

until the maximum estimated error is below a coarse tolerance denoted by tol$^c >$ tol,
where tol is the desired error tolerance for the final ROM, or (b) at two successive iterations,
the number of DEIM interpolation points or QR pivots does not change. The value of tol^c is
user-defined and is of order $\mathcal{O}(1)$ in this work. Based on the two stopping criteria, two different
schemes of training set subsampling are presented in Algorithms 3.1 and 3.2, respectively.
For both algorithms, we do not reset the value of iter at the end of Stage 1, so the final
value of iter upon convergence for Algorithms 3.1 and 3.2 is the total number of iterations
required by either algorithms to converge to the desired tolerance. It is worth pointing out
that an *a posteriori* error estimator [16] is used in both stages of the proposed algorithms so
that the parameter sample picked at each iteration is the one at which an estimated *output*
error is the largest. Furthermore, the adaptive basis enrichment technique proposed in [16] is
implemented and serves as a possible solution to the issue of repeated parameter selection. The
number of POD modes corresponding to a selected parameter sample is adaptively decided:
when the estimated error is large, a higher number of POD modes (r_{POD}) for the selected
parameter are added; otherwise fewer POD modes are added. This reduces the chance of the
same parameter sample being repeatedly chosen at subsequent iterations. This adaptive basis
enrichment is implemented for both stages of our proposed method. We now discuss several
practical computational strategies in connection with Steps 11 and 12 in Algorithm 3.1 and
Steps 12, 13 and 16 in Algorithm 3.2.

Algorithm 3.2 Reduced Basis Method with Training Set Subsampling (Scheme 2)

Input: Training set ($\varXi_{\text{train}}^{f}$), tolerance for the ROM (tol), maximum iteration (n_g).
Output: V.
1: Initialize $\mathbf{V} = [\ ]$, $\epsilon = 1 + \text{tol}$, iter $= 1$.
2: Select $\boldsymbol{\mu}_{(1)}^{*}$ randomly from $\varXi_{\text{train}}^{f}$.

   Stage 1

3: **while** not terminated **do**
4:    Solve FOM (2.2) at $\boldsymbol{\mu}_{(\text{iter})}^{*}$ and obtain snapshot matrix $\mathfrak{X}$.
5:    Set $\bar{\mathfrak{X}} = \mathfrak{X} - \mathbf{V}\mathbf{V}^{\mathsf{T}}\mathfrak{X}$.
6:    Enrich $\mathbf{V}$ with r_{POD} left singular vectors of $\bar{\mathfrak{X}}$ as
     $\mathbf{V} \leftarrow \text{orth}\left(\left[\mathbf{V}\ \mathbf{U}_{\bar{\mathfrak{X}}}(:,\ 1\ :\ r_{\text{POD}})\right]\right)$.
7:    iter $=$ iter $+ 1$.
8:    Set $\boldsymbol{\mu}_{(\text{iter})}^{*} = \arg\max_{\boldsymbol{\mu} \in \varXi_{\text{train}}^{f}} \Delta(\boldsymbol{\mu})$.
9:    $\epsilon = \Delta(\boldsymbol{\mu}_{(\text{iter})}^{*})$.
10:   Perform pivoted QR decomposition of $\widetilde{\mathfrak{Y}}^{\mathsf{T}}$, or apply DEIM or a DEIM variant to $\widetilde{\mathfrak{Y}}$ and identify the
     indices $\mathbf{I}$ of the QR pivots or DEIM interpolation points.
11:   **if** iter ≥ 2 **then**
12:      Check if length($\mathbf{I}_{\text{iter - 1}}$) $==$ length($\mathbf{I}_{\text{iter}}$).
13:      If true **break** and proceed to **Stage 2**.
14:   **end if**
15: **end while**

   Stage 2

16: Identify new training set $\varXi_{\text{train}}$ using distribution of $\mathbf{I}$.
17: **while** $\epsilon > \text{tol}$ && iter $\leq n_g$ **do**
18:    Solve FOM (2.2) at $\boldsymbol{\mu}_{(\text{iter})}^{*}$ and obtain snapshot matrix $\mathfrak{X}$.
19:    Set $\bar{\mathfrak{X}} = \mathfrak{X} - \mathbf{V}\mathbf{V}^{\mathsf{T}}\mathfrak{X}$.
20:    Enrich $\mathbf{V}$ with r_{POD} left singular vectors of $\bar{\mathfrak{X}}$ as
     $\mathbf{V} \leftarrow \text{orth}\left(\left[\mathbf{V}\ \mathbf{U}_{\bar{\mathfrak{X}}}(:,\ 1\ :\ r_{\text{POD}})\right]\right)$.
21:    iter $=$ iter $+ 1$.
22:    Set $\boldsymbol{\mu}_{(\text{iter})}^{*} = \arg\max_{\boldsymbol{\mu} \in \varXi_{\text{train}}} \Delta(\boldsymbol{\mu})$.
23:    $\epsilon = \Delta(\boldsymbol{\mu}_{(\text{iter})}^{*})$.
24: **end while**

Remark 3.2 The active subspaces method (ASM) is an approach for parameter space reduction that has been recently applied in the context of MOR [20,52,56] mainly for the case of scalar valued outputs. The ASM identifies a set of important directions in the parameter space onto which the parameter vectors are projected. It does this by means of Monte Carlo sampling of the gradients (with respect to the parameter) of the scalar-valued output quantity at a selection of parameter samples. The *active subspaces* are the eigenspaces of the (truncated) covariance matrix of the gradients. Compared to ASM, our approach differs in two significant ways. Firstly, the proposed subsampling strategy is applicable to vector-valued output quantities. Secondly, ASM requires calculation of the gradient of the output. Moreover, the user still has to define the training set over which the gradient samples are acquired. Our proposed approach does not require the calculation of any additional quantity.

Remark 3.3 Our proposed subsampling strategy occuring in Step 11 of Algorithm 3.1 and Step 10 of Algorithm 3.2 shares similarity with the column subset selection problem (CSSP) in the fields of numerical linear algebra and data mining [38]. For some general data matrix $\mathbf{D} \in \mathbb{R}^{N \times M}$, the CSSP aims to identify $h < M$ independent columns of the matrix $\mathbf{D}$ such that

the residual $\|\mathbf{D} - \mathbf{P}_h \mathbf{D}\|$ is minimized. Here, $\mathbf{P}_h = \mathbf{S}\mathbf{S}^\dagger$ is a projection matrix and $\mathbf{S} \in R^{N \times h}$ consists of the h extracted columns from $\mathbf{D}$. A number of algorithms, both deterministic and randomized, have been proposed to solve the CSSP [10,11,13]. One popular approach is to apply some variant of the QR decomposition (column-pivoted, rank-revealing, hybrid, etc.) either to the data matrix $\mathbf{D}$ or to the transpose of the (truncated) left (or right) singular matrix $\mathbf{U}$ (or $\mathbf{W}$) of $\mathbf{D}$. If we consider $\mathbf{D}$ as the approximate output snapshot matrix, then our proposed algorithm using pivoted QR (or QDEIM) can be seen as a special case of the CSSP.

3.2.1 Training Set Subsampling: Scheme 1

We describe the proposed approach for the first scheme detailed in Algorithm 3.1. In the first stage, a low-fidelity ROM is built with a coarse tolerance $\texttt{tol}^c$, over a finely sampled training set Ξ_{train}^f. The intuition is that a low-fidelity approximation is sufficient to discover the parametric dependence of the output variable. At the end of the first stage, DEIM (or one of its variants) is applied to $\widetilde{\mathfrak{Y}}$ to identify the interpolation points or a pivoted QR decomposition of $\widetilde{\mathfrak{Y}}^\mathsf{T}$ is used to identify the pivots. Once the set of interpolation points (or pivots) p_i is identified, we proceed to suitably subsample the finely sampled training set based on the distribution of the identified interpolation points or pivots. Different possibilities exist to achieve this. A simple approach is to consider the training set for the second stage Ξ_{train} populated only by the identified interpolation points or pivots. Consider the fine training set $\Xi_{\text{train}}^f := \{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_{N_{\text{train}}^f}\}$ with the subscript denoting the index corresponding to the position of a parameter in the set. Let $\mathbf{I}$ be the vector whose elements are the indices of the chosen interpolation points or pivots. We define the subsampled training set Ξ_{train} as the one consisting of all those parameters $\boldsymbol{\mu}_z$ from Ξ_{train}^f such that their indices are present in $\mathbf{I}$, i.e., $\Xi_{\text{train}} := \{\boldsymbol{\mu}_{z;z\in\mathbf{I}}\}$. If there are only a few interpolation points, this approach would lead to a rapid second stage of the algorithm. However, there may exist the pitfall that it may result in an *overfit* by which we mean that the resulting ROM after Stage 2 satisfies the desired tolerance $\texttt{tol}$ only over the subsampled training set and does not generalize to other parameters in the parameter domain. We illustrate this phenomenon in the numerical tests. Another possible approach is to define a training budget m for the second stage and use an oversampling strategy like the Gappy-POD to ensure that the training set for the second stage consists of a total of m parameter samples.

3.2.2 Training Set Subsampling: Scheme 2

The first scheme of our proposed training set adaptation method requires a user-defined coarse tolerance. Choosing such a tolerance is rather heuristic. For some problems, a rough approximation may be enough to suitably capture all the parametric dependences, whereas a finer approximation may be needed for others. Therefore, for the second scheme we define a heuristic criterion that leads automatically to the second stage. To achieve this, we apply DEIM approximation (or a variant of DEIM) to the matrix $\widetilde{\mathfrak{Y}}$ or the pivoted QR factorization to $\widetilde{\mathfrak{Y}}^\mathsf{T}$, at each iteration of the greedy algorithm. Whenever the DEIM interpolations or pivoted QR decompositions at two successive iterations turn out to be equal, we terminate the first stage. This can be easily calculated by comparing the number of interpolation indices or the pivot indices at two successive iterations. Then, the subsampling of the training set for the second stage is carried out with similar approaches as discussed above. Following this, the second stage is run, until the required tolerance $\texttt{tol}$ is met. In Algorithm 3.2, $\mathbf{I}_{\texttt{iter}}$ and $\mathbf{I}_{\texttt{iter - 1}}$ in Stage 1 refer to the vector containing the interpolation indices identified by

DEIM or its variants, or the indices of the QR pivots at the current iteration and the previous iteration, respectively.

Each of the two schemes has its own benefits or shortcomings. For Scheme 1, the burden of choosing an appropriate coarse tolerance lies with the user. This is highly problem-dependent. In the limit that $\texttt{tol}^c = \texttt{tol}$, Scheme 1 is just the standard Greedy algorithm (Algorithm 2.1) with fixed training set. If $\texttt{tol}^c \gg \texttt{tol}$, a very fast second stage can be ensured, leading to considerable speedup of the offline stage of the RBM. However, it is not known *a priori* if the chosen coarse tolerance is good. Scheme 2, on the other hand, automatizes the switching from Stage 1 to Stage 2 of the subsampling strategy by considering a heuristic criterion for the subspace approximation of the snapshot matrix $\widetilde{\mathfrak{Y}}$. But, this comes with the additional cost of performing the DEIM algorithm or the pivoted QR decomposition, repeatedly. The success of both schemes is also highly dependent on the strategy adopted to construct the subsampled training set. In the numerical tests, we shall consider two approaches. In the first approach, we consider as many parameters in the subsampled training set as the number of DEIM interpolation points or QR pivots. For the second approach, we fix the cardinality of the subsampled training set to be m and then use oversampling strategies based on the Gappy-POD method to choose those m parameters in a principled way.

3.2.3 Complexity Analysis

The fine training set is used in Stage 1 of both Algorithms 3.1 and 3.2. However, the computational complexity is not high for Algorithm 3.1, since we use a coarse tolerance $\texttt{tol}^c$ in Stage 1, so that the greedy algorithm converges within much fewer steps than when using the user desired tolerance in Stage 2. The computational complexity will grow with the decrease of the coarse tolerance used in Stage 1. However, as we have observed, a moderate value of $\texttt{tol}^c$ is enough to figure out the parameter dependency of the output. The number of FOM simulations is indeed independent of the size of the fine training set, since the FOM simulation is implemented only at those "selected" parameter samples. The situation for Algorithm 3.2 is different since it involves the DEIM or QR algorithms at each iteration (see Step 10) to compute the interpolation points. A fine training set will indeed increase the cost of this step. Nevertheless, one can readily use recent techniques based on randomized linear algebra (such as randomized SVD, randomized QR, etc.) to keep the costs under control, see [22,54].

In this section, we roughly compare the computational complexity of the standard RBM without training set subsampling in Algorithm 2.1 to that of RBM using the two subsampling strategies introduced in Algorithms 3.1 and 3.2. Our complexity analysis considers the worst-case costs involved with each algorithm and is meant as a simplified illustration of the benefits of subsampling the training set. For simplicity, we only count the dominant costs in each algorithm.

We begin by introducing some notation:

- Let $\mathcal{C}_{\text{FOM}} := O(n^2) \cdot N_t$ denote the cost of solving the FOM (for e.g., (2.2)) using some iterative scheme (GMRES, etc.), at a fixed parameter to obtain the snapshots.
- Let $\mathcal{C}_{\text{SVD}} := O(n^3)$ be the worst-case cost for performing the SVD.
- Let $\mathcal{C}_{\text{Err}} := O(s)$ be the cost associated with evaluating the error estimator $\Delta(\boldsymbol{\mu})$ for a fixed parameter.
- Note that the dimension of the snapshot matrix in (3.4) is $N_{\text{train}}^f \times q N_t$. For large-scale systems, usually we have $N_{\text{train}}^f \leq n$, if using, e.g., a sparse grid sampling, and often $q N_t \leq n$. Therefore, the cost for the DEIM algorithm (or its variants) or the pivoted QR decomposition should be less than $O(n^3)$. In general, we denote it as $\mathcal{C}_{\text{ss}} := O(n^3)$.

The cost associated with Algorithm 2.1, viz., $\mathcal{C}_I$ is:

$$\mathcal{C}_I := \left(\mathcal{C}_{\text{FOM}} + \mathcal{C}_{\text{SVD}} + \mathcal{C}_{\text{Err}} \cdot N_{\text{train}}^f \right) \cdot N_I$$

with N_{train}^f being the cardinality of the fine training set and N_I the number of iterations taken by the greedy algorithm to converge to the desired tolerance. The cost incurred by Algorithm 3.1, $\mathcal{C}_{II}$, is divided between Stage 1 and Stage 2. This is given by:

$$\mathcal{C}_{II} := \overbrace{\left(\mathcal{C}_{\text{FOM}} + \mathcal{C}_{\text{SVD}} + \mathcal{C}_{\text{Err}} \cdot N_{\text{train}}^f \right) \cdot N_{II,1}}^{\text{Stage 1 cost}}$$
$$+ \underbrace{\mathcal{C}_{\text{ss}} + \left(\mathcal{C}_{\text{FOM}} + \mathcal{C}_{\text{SVD}} + \mathcal{C}_{\text{Err}} \cdot N_{\text{train}} \right) \cdot N_{II,2}}_{\text{Stage 2 cost}} .$$

In the above expression, $N_{II,1}$, $N_{II,2}$ are, respectively, the number of iterations of the greedy algorithm in Stage 1 and Stage 2 of Algorithm 3.1. Typically, $N_{II,2} > N_{II,1}$ since we use a coarse tolerance for Stage 1. Moreover, N_{train} is the cardinality of the subsampled training set Ξ_{train}. Finally, the computational cost of Algorithm 3.2, $\mathcal{C}_{III}$, is as follows:

$$\mathcal{C}_{III} := \overbrace{\left(\mathcal{C}_{\text{FOM}} + \mathcal{C}_{\text{SVD}} + \mathcal{C}_{\text{Err}} \cdot N_{\text{train}}^f + \mathcal{C}_{\text{ss}} \right) \cdot N_{III,1}}^{\text{Stage 1 cost}}$$
$$+ \underbrace{\left(\mathcal{C}_{\text{FOM}} + \mathcal{C}_{\text{SVD}} + \mathcal{C}_{\text{Err}} \cdot N_{\text{train}} \right) \cdot N_{III,2}}_{\text{Stage 2 cost}}$$

with $N_{III,1}$, $N_{III,2}$ being, respectively, the number of iterations of the greedy algorithm in Stage 1 and Stage 2 of Algorithm 3.2. For Algorithm 3.1 or Algorithm 3.2 to be computationally better alternatives to Algorithm 2.1, we should have $\mathcal{C}_I > \mathcal{C}_{II}$ and $\mathcal{C}_I > \mathcal{C}_{III}$.
Cost benefit of Algorithm 3.1: To compare the costs of Algorithms 2.1 and 3.1, we look at the expression $\mathcal{C}_I - \mathcal{C}_{II}$.

$$\mathcal{C}_I - \mathcal{C}_{II} := \mathcal{C}_{\text{FOM}}(N_I - N_{II,1} - N_{II,2})$$
$$+ \mathcal{C}_{\text{SVD}}(N_I - N_{II,1} - N_{II,2})$$
$$+ \mathcal{C}_{\text{Err}}(N_{\text{train}}^f \cdot N_I - N_{\text{train}}^f \cdot N_{II,1} - N_{\text{train}} \cdot N_{II,2})$$
$$- \mathcal{C}_{\text{ss}}.$$

By assuming $N_I \approx N_{II,1} + N_{II,2}$ we have for the above expression

$$\mathcal{C}_I - \mathcal{C}_{II} \approx \mathcal{C}_{\text{Err}}(N_{\text{train}}^f - N_{\text{train}})N_{II,2} - \mathcal{C}_{\text{ss}}.$$

For $N_{\text{train}}^{f} \gg N_{\text{train}}$, the first term would dominate leading to $\mathcal{C}_I > \mathcal{C}_{II}$. Thus, we see that computational cost incurred by Algorithm 3.1 is less than that of Algorithm 2.1. In our analysis, we have assumed that $N_I \approx N_{II,1} + N_{II,2}$. While this helps to simplify the analysis, it does not always hold. As we shall see in Sect. 4, for the example of Burgers' equation this assumption is true, while it does not hold for the thermal block example. A more relaxed assumption is $N_I \gtrapprox N_{II,1} + N_{II,2}$ using which, it can still be seen that $C_I > C_{II}$.

Cost benefit of Algorithm 3.2: We look at the difference in costs $\mathcal{C}_I - \mathcal{C}_{III}$.

$$
\begin{aligned}
\mathcal{C}_I - \mathcal{C}_{III} := & \mathcal{C}_{\text{FOM}}(N_I - N_{III,1} - N_{III,2}) \\
& + \mathcal{C}_{\text{SVD}}(N_I - N_{III,1} - N_{III,2}) \\
& + \mathcal{C}_{\text{Err}}(N_{\text{train}}^{f} \cdot N_I - N_{\text{train}}^{f} \cdot N_{III,1} - N_{\text{train}} \cdot N_{III,2}) \\
& - \mathcal{C}_{\text{ss}} N_{III,1}.
\end{aligned}
$$

By making the assumption $N_I \approx N_{III,1} + N_{III,2}$, we have for the above expression

$$
\mathcal{C}_I - \mathcal{C}_{III} \approx \mathcal{C}_{\text{Err}}(N_{\text{train}}^{f} - N_{\text{train}}) N_{III,2} - \mathcal{C}_{\text{ss}} N_{III,1}.
$$

Usually $N_{\text{train}}^{f} \gg N_{\text{train}}$ and moreover, $N_{III,1} < N_{III,2}$. Therefore, the first term would dominate, leading to $\mathcal{C}_I > \mathcal{C}_{III}$. Once again, we see that the RBM using the subsampling strategy (Algorithm 3.2) incurs a smaller cost when compared to Algorithm 2.1. In the case of Algorithm 3.2, just like earlier, the assumption $N_I \approx N_{III,1} + N_{III,2}$ need not always hold. From the numerical tests in the next section, we actually have $N_I \gtrapprox N_{III,1} + N_{III,2}$. The numerical results in Tables 2 and 3 show that Algorithms 3.1 and 3.2 indeed achieve speedups that can only be secured when they possess less computational complexity than Algorithm 2.1.

Remark 3.4 (High-dimensional parameter spaces) Both Algorithms 3.1 and 3.2, can be used when the parameter space is high-dimensional. The cost in Stage 2 will not be affected much, since a small training set identified from Stage 1 is used. The main increase in cost is due to the need to solve additional ROMs and estimate the error in Stage 1 of the proposed algorithms (Step 8 in Algorithms 3.1 and 3.2). If sparse grid sampling and a cheap error estimator are used, the cost will not increase fast. Actually, we can go a step further and make use of cheaply computable surrogate models of the error estimator as done in [17]. In that work, we considered a radial basis surrogate for the error estimator that is adaptively updated during the greedy algorithm. We only evaluate the actual error estimator over a few parameter samples. We then use this data to form a surrogate model, which can be used to evaluate the error for different parameter samples in the fine training set in Stage 1.

Remark 3.5 (Role of output quantity of interest) In many cases, there is often the requirement, based on the application, to have a good approximation for the entire state vector. However, in this work we have specifically focussed on a *goal-oriented* approach. For several applications, like in control systems or fluid dynamics, only a small number of state variables may be of interest. By focussing on those states alone, the resulting ROM dimension can be considerably lowered when compared to the case where the entire state needs to be well approximated. Also, it is indeed true that different output QoIs may have different influences on the parameter samples chosen. However, if some QoIs give rise to quantities with similar emphasis, then they may result in similar parameter samples being chosen. For example, a QoI defined by the mean of the solution over the spatial domain and a QoI defined by the sum of the solution over the spatial domain should result in similar samples. But a QoI defined by the mean of the

solution over space probably gives different samples from the QoI defined by the maximum of the state over the spatial domain.

4 Numerical Results

We test the proposed adaptive training set subsampling algorithm on two examples. They are:

1. Viscous Burgers' equation with one parameter,
2. thermal block with four parameters.

The first example is a nonlinear system, while the other is linear. All numerical tests are carried out in MATLAB® 8.5.0.197613 (R2015a) on a laptop with Intel® Core™ i5-7200U @ 2.5 GHz and 8 GB of RAM. Next, we describe the metrics used in all the numerical tests:

- The results of the proposed algorithms (Algorithms 3.1 and 3.2) are compared against a standard implementation of the POD-Greedy algorithm (Algorithm 2.1) with a fixed training set. The implementation adopts Galerkin projection. The number of POD modes r_{POD}, r_{EI} to enrich the RB and DEIM bases is determined at each iteration based on the adaptive approach proposed in [16]. We have also used the primal-dual error estimator proposed by the authors of [16] for our implementation of Algorithms 2.1, 3.1 and 3.2. The dual RB basis required for the error estimator is generated separately.
- The fixed training set used for Algorithm 2.1 and the initial fine training set used for Algorithms 3.1 and 3.2 are the same.
- For Algorithms 3.1 and 3.2, we apply (i) the pivoted QR decomposition to the transposed approximate output snapshot matrix $\widetilde{\mathfrak{Y}}^{\mathsf{T}}$ and, (ii) the DEIM variants on the approximate output snapshot matrix $\widetilde{\mathfrak{Y}}$ as two approaches to subsample the fine training set.
- The cut-off criterion to determine the number of pivots (h) for the pivoted QR decomposition in (3.2) of the approximate output snapshot matrix in (3.4) is based on the magnitude of the diagonal elements in the upper triangular matrix $\mathbf{R}$, i.e., we set $h = q$ based on the the smallest q such that $|\mathbf{R}(q + 1, q + 1)|/|\mathbf{R}(1, 1)| < \epsilon_{\text{QR}}$, with $q \in \{1, 2, \ldots, \min(N_{\text{train}}, N_t)\}$. The pivoted QR decomposition can effectively identify the rank of a matrix with a small diagonal $\mathbf{R}(q + 1, q + 1)$. Although there are cases when the column pivoted QR decomposition fails, they are rare in practice [14]. A more robust rank-revealing QR factorization [14] can also be straightforwardly applied to our proposed subsampling algorithm. However, in this work, we simply use column pivoted QR. The intrinsic MATLAB® command `qr` is used with the options `vector` enabled, i.e., we call $[\mathbf{Q}, \mathbf{R}, p_{\text{qr}}] = \text{qr}(\widetilde{\mathfrak{Y}}^{\mathsf{T}}, \text{'vector'})$. It returns the pivot indices p_{qr} as a vector, from which we select the first h as our subsampling indices, i.e., $\mathbf{I} = p_{\text{qr}}(1 : h)$. Here, $\mathbf{I}$ is the vector whose elements are the indices of the QR pivots and it lets us choose the subsampled training set $\varXi_{\text{train}}$ for Stage 2 of our proposed method, based on the fine training set $\varXi_{\text{train}}^f$ from Stage 1.
- Our implementation of the `k-means` algorithm for KDEIM is based on the intrinsic MATLAB® function `kmeans`. We use five different initializations and pick the best configuration among the five.
- The maximum true error over the test set is defined as

$$\epsilon_{\text{t}}^{\max} := \max_{\boldsymbol{\mu} \in \varXi_{\text{test}}} \left(\frac{1}{K + 1} \sum_{k=0}^{K} \| \mathbf{y}\left(\mathbf{x}(t^k, \boldsymbol{\mu}) \right) - \widetilde{\mathbf{y}}\left(\mathbf{z}(t^k, \boldsymbol{\mu}) \right) \| \right).$$

– Reported runtimes for all the algorithms are obtained by considering the median value
  of five independent runs.
– The quantity Iterations reported in Tables 2 to 6 refers to the total number of iterations
  (`iter`) of the corresponding greedy algorithm (Algorithms 2.1, 3.1 and 3.2) to converge
  to the desired tolerance.

4.1 Viscous Burgers' Equation

We consider the following viscous Burgers' equation defined on a 1-D domain $\Omega := [0, 1]$:

$$\frac{\partial x}{\partial t} + x \frac{\partial x}{\partial w} = \mu \frac{\partial^2 x}{\partial w^2} + s(x, t), \quad x(0, t) = 0 \ \& \ \frac{\partial x(1, t)}{\partial w} = 0, \tag{4.1}$$
$$y = x(1, t),$$

where $s(x, t)$ denotes a forcing term defined later. The domain is discretized using the finite
difference method with step size $\Delta w = 0.001$. We make use of a second order central
difference discretization for the diffusion term and a first-order upwind scheme for the con-
vection term. The resulting FOM is of dimension $n = 1000$. For the time variable $t \in [0, 2]$,
we make use of a first order implicit-explicit (IMEX) scheme with the diffusion term dis-
cretized implicitly and the nonlinear convection term discretized explicitly. The step size is
$\Delta t = 0.001$. The constant input to the system is set to be $s(x, t) \equiv 1$ and the initial condition
is $\mathbf{x}_0 := \mathbf{0} \in \mathbb{R}^n$. The parameter domain of the viscosity μ is $\mathcal{P} := [0.005, 1]$. The training
set Ξ_{train} consists of 100 equally spaced samples in $\mathcal{P}$. For the RBM, we fix the tolerance to
be `tol` $= 1 \cdot 10^{-6}$. To validate the ROM, we use a test set Ξ_{test} containing 300 randomly
sampled parameters, different from those in the training set.

4.1.1 Greedy Algorithm with Fixed Training Set

We begin by applying the standard greedy algorithm (Algorithm 2.1) to the discretized
model of the Burgers' equation. The greedy algorithm requires $t_{\text{greedy}} = 505.47$ seconds and
19 iterations to converge to the defined tolerance of $1 \cdot 10^{-6}$. The resulting ROM has RB
dimension $r_{\text{POD}} = 32$ along with $r_{\text{EI}} = 33$ basis vectors for the DEIM projection matrix.
The maximum true error over the test set is $\epsilon_t^{\max} = 2.07 \cdot 10^{-8}$. Although the POD-Greedy
algorithm with a fixed training set results in a ROM that meets the specified tolerance, its
offline time is high and there is scope for improvement by considering a subsampled training
set.

4.1.2 Greedy Algorithm Schemes 1 and 2

We apply Algorithms 3.1 and 3.2 to the Burgers' equation, making use of both pivoted QR
and the two DEIM variants (QDEIM and KDEIM) to identify the interpolation points $\mathbf{I}$. Further,
we consider three different SVD and QR cut-off tolerances ($\epsilon_{\text{SVD}}, \epsilon_{\text{QR}}$) for the pivoted QR
and DEIM variants $\{1 \cdot 10^{-4}, 1 \cdot 10^{-6}, 1 \cdot 10^{-8}\}$ to highlight the progressive refinement of the
adapted training set in 'difficult regions' of the parameter space. The results are summarized
in Tables 2 and 3 for Algorithms 3.1 and 3.2, respectively. The matrix $\widetilde{\mathfrak{Y}} \in \mathbb{R}^{100 \times 81}$, for
either algorithms, was assembled by collecting the snapshots of the output vector at every
25th time step. The training set in Stage 2 of both algorithms consists of interpolation points
identified by QR, QDEIM or KDEIM. As revealed in the results, this choice is sufficient to
produce ROMs that meet the required tolerance over the test set. For this example, there is

Table 2 Results of Algorithm 3.1 with varying ϵ_{SVD}, ϵ_{QR} for Burgers' equation

Method	Fixed				Adapted					
		$\epsilon_{SVD}, \epsilon_{QR} = 1 \cdot 10^{-4}$			$\epsilon_{SVD}, \epsilon_{QR} = 1 \cdot 10^{-6}$			$\epsilon_{SVD}, \epsilon_{QR} = 1 \cdot 10^{-8}$		
		QDEIM	KDEIM	QR	QDEIM	KDEIM	QR	QDEIM	KDEIM	QR
N_{train}	100	7	7	8	11	11	12	15	15	15
$\epsilon_i^{\max}$	$2.07 \cdot 10^{-8}$	$4.22 \cdot 10^{-7}$	$4.15 \cdot 10^{-7}$	$3.43 \cdot 10^{-8}$	$1.87 \cdot 10^{-8}$	$1.87 \cdot 10^{-8}$	$3.43 \cdot 10^{-8}$	$1.87 \cdot 10^{-8}$	$1.84 \cdot 10^{-8}$	$3.28 \cdot 10^{-8}$
$(r_{\text{POD}}, r_{\text{EI}})$	(32,33)	(31,32)	(30,31)	(31,32)	(31,32)	(31,32)	(31,32)	(31,32)	(31,32)	(31,32)
Iterations (iter)	19	17	17	17	18	18	17	18	18	17
Offline time (s)	505.47	91.00	91.16	94.90	110.10	110.35	109.54	125.07	125.59	120.53
Speedup	–	5.6	5.5	5.3	4.6	4.6	4.6	4.0	4.0	4.2

Table 3 Results of Algorithm 3.2 with varying ϵ_{SVD}, ϵ_{QR} for Burgers' equation

Method	Fixed	Adapted								
		$\epsilon_{SVD}, \epsilon_{QR} = 1 \cdot 10^{-4}$			$\epsilon_{SVD}, \epsilon_{QR} = 1 \cdot 10^{-6}$			$\epsilon_{SVD}, \epsilon_{QR} = 1 \cdot 10^{-8}$		
		QDEIM	KDEIM	QR	QDEIM	KDEIM	QR	QDEIM	KDEIM	QR
N_{train}	100	7	7	8	11	11	12	15	15	15
ϵ_i^{max}	$2.07 \cdot 10^{-8}$	$4.22 \cdot 10^{-7}$	$3.43 \cdot 10^{-7}$	$3.43 \cdot 10^{-8}$	$1.87 \cdot 10^{-8}$	$5.96 \cdot 10^{-8}$	$3.43 \cdot 10^{-8}$	$1.87 \cdot 10^{-8}$	$2.59 \cdot 10^{-8}$	$3.28 \cdot 10^{-8}$
(r_{POD}, r_{EI})	(32,33)	(31,32)	(30,31)	(31,32)	(31,32)	(31,32)	(31,32)	(31,32)	(30,31)	(31,32)
Iterations (iter)	19	17	16	17	18	18	17	18	16	17
Offline time (s)	505.47	90.78	88.52	94.58	110.12	110.10	109.55	145.22	135.82	120.27
Speedup	–	5.6	5.7	5.3	4.6	4.6	4.6	3.5	3.7	4.2
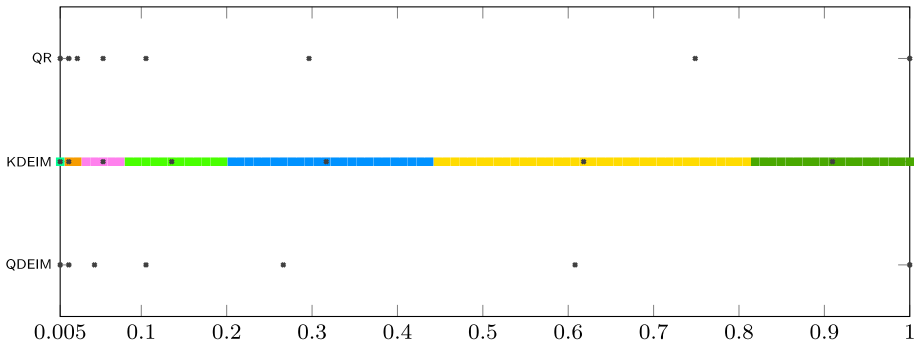
Fig. 3 Algorithm 3.2 for the Burgers' Equation with SVD, QR tolerance ϵ_{SVD}, $\epsilon_{\text{QR}} = 1 \cdot 10^{-4}$. The crossmarks denote the parameters in the subsampled training set. For KDEIM each colour represents one cluster; the centroids of each of the clusters make up the subsampled training set (Color figure online)
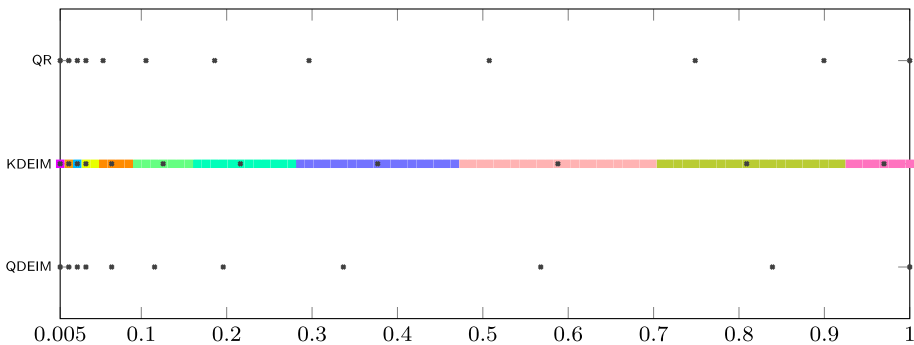
Fig. 4 Algorithm 3.2 for the Burgers' Equation with SVD, QR tolerance ϵ_{SVD}, $\epsilon_{\text{QR}} = 1 \cdot 10^{-6}$. The crossmarks denote the parameters in the subsampled training set. For KDEIM each colour represents one cluster; the centroids of each of the clusters make up the subsampled training set (Color figure online)

not a big difference between the results of the two algorithms. Both schemes produce ROMs of almost identical RB, DEIM basis sizes (r_{POD}, r_{EI}) and result in nearly the same maximum error over the test set.

We show the subsampled training sets resulting from Algorithm 3.2 using the pivoted QR, QDEIM and KDEIM variants, with different SVD, QR tolerances in Figs. 3, 4, 5. The black crosses denote those samples from the fine training set which were retained in Stage 2 of the algorithm. For the QDEIM variant, it is clear that the subsampled parameters are concentrated more around the lower viscosity regions of the parameter space. Thus, the method is able to successfully identify the *physically more relevant* points. Moreover, the parameter samples identified by QDEIM are very close to the ones identified by the method using a pivoted QR decomposition. This is not surprising since the former determines the interpolation points through a pivoted QR decomposition of $\mathbf{U}^{\mathsf{T}}$ ($\mathbf{U}$ is the left singular matrix of $\widetilde{\mathfrak{Y}}$) whereas the latter applies the pivoted QR decomposition directly to $\widetilde{\mathfrak{Y}}^{\mathsf{T}}$. We also show the results of KDEIM, where the subsampled (selected) parameter samples and their corresponding clusters are presented. The subsampled points in this case are the centroids of the clusters. The clusters are smaller in size for the low viscosity regions, while they are comparatively larger in the high viscosity zone. The resulting subsampled training sets from Algorithm 3.1 display a similar trend.
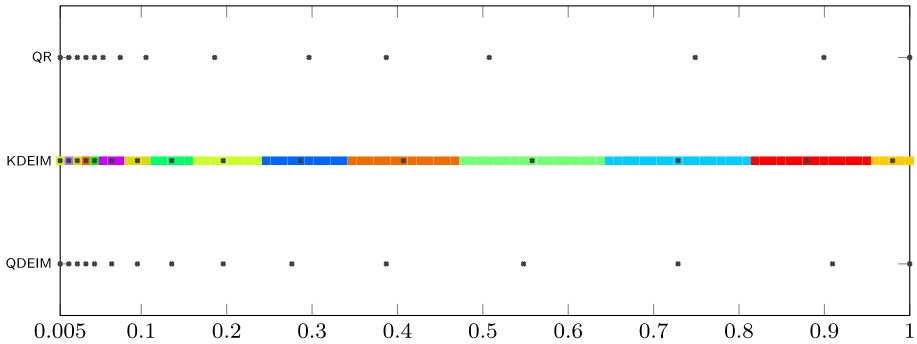
Fig. 5 Algorithm 3.2 for the Burgers' Equation with SVD, QR tolerance ϵ_{SVD}, $\epsilon_{QR} = 1 \cdot 10^{-8}$. The crossmarks denote the parameters in the subsampled training set. For KDEIM each colour represents one cluster; the centroids of each of the clusters make up the subsampled training set (Color figure online)

For a given ϵ_{SVD} or ϵ_{QR}, the runtimes for the QDEIM and KDEIM based training set adaptation are very close. Using the pivoted QR leads to a subsampled training set with one sample more than that generated by using QDEIM or KDEIM. This results in a marginally higher offline time for this method. One observation worth remarking is that, for some instances, using the KDEIM approach to identify the adapted training set leads to the greedy algorithm converging in fewer iterations. This is most likely due to the fact that the identified parameters in this case represent cluster centroids and are more representative of the average behaviour. This yields a more uniform approximation throughout the parameter domain, with each cluster average being well-represented. The QDEIM version, on the other hand, tends to identify points based on the SVD of the output snapshot matrix and tends to favour points away from the mean behaviour. We illustrate this in Fig. 6 for the case of $\epsilon_{SVD} = 10^{-8}$ for Algorithm 3.1. It is evident that while the subsampling strategy using QDEIM results in a smaller magnitude of the maximum error over the test set ($4.55 \cdot 10^{-8}$ vs. $6.88 \cdot 10^{-8}$), the approach using the KDEIM-based sampling leads to a more uniform distribution of the error over the test set.

On average, for a given SVD tolerance, Algorithm 3.1 is faster than Algorithm 3.2. This can be attributed to the fact that for the latter, the DEIM variant or the pivoted QR decomposition needs to be performed repeatedly to check the criterion in Step 12 of Algorithm 3.2. Since this involves performing an SVD, the associated costs are higher. The proposed subsampling algorithms result in a noticeable speedup of the POD-Greedy algorithm. For Algorithm 3.1, the maximum achieved speedup was 5.6 for $\epsilon_{SVD} = 1 \cdot 10^{-4}$ using QDEIM. However, the least speedup noticed was 4.0 for $\epsilon_{SVD} = 1 \cdot 10^{-8}$ using QDEIM. Also, for Algorithm 3.2 the maximum achieved speedup was 5.7 for $\epsilon_{SVD} = 1 \cdot 10^{-4}$ using KDEIM while the minimum speedup was 3.5 for $\epsilon_{SVD} = 1 \cdot 10^{-8}$ using QDEIM. Finally, we see from Tables 2 and 3 that the number of iterations to converge for Algorithm 2.1, Algorithms 3.1 and 3.2 are nearly the same. Thus, our assumption in the analysis from Sect. 3.2.3 that $N_I \approx N_{II,1} + N_{II,2}$ and $N_I \approx N_{III,1} + N_{III,2}$ holds true.

4.2 Thermal Block

The second example is a benchmark model of the time-dependent heat transfer in a thermal block. The governing PDE is given by
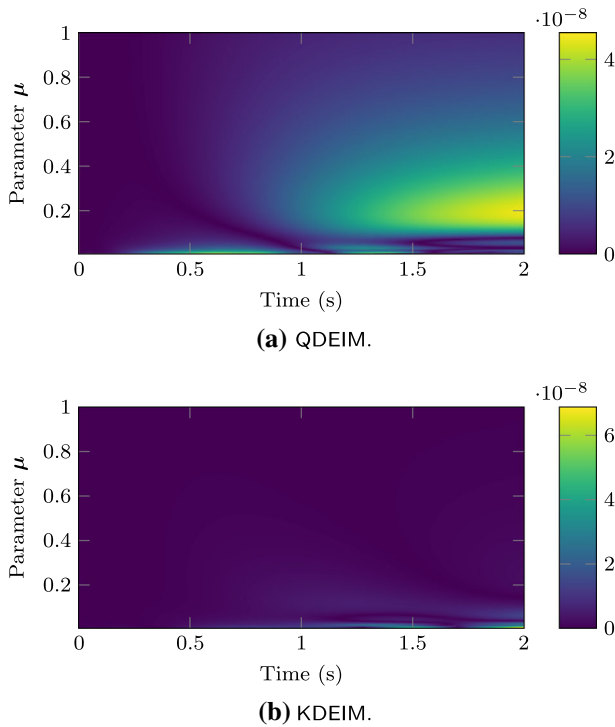
(a) QDEIM.

(b) KDEIM.

Fig. 6 Error plot for Algorithm 3.1 with tolerance $\epsilon_{\text{SVD}} = 10^{-8}$ applied to the Burgers' equation. The error between the true and reduced outputs $\|\mathbf{y}\left(\mathbf{x}(t^k, \boldsymbol{\mu})\right) - \tilde{\mathbf{y}}\left(\mathbf{z}(t^k, \boldsymbol{\mu})\right)\|$ is plotted over the duration of the simulation for all parameters in the test set

$$\frac{\partial \theta(\mathbf{w}, t, \boldsymbol{\mu})}{\partial t} + \nabla \cdot (-\gamma(\mathbf{w}, \boldsymbol{\mu})\nabla\theta(\mathbf{w}, t, \boldsymbol{\mu})) = 0, \qquad t \in [0, T]. \tag{4.2}$$

The domain $\Omega := (0, 1) \times (0, 1) \in \mathbb{R}^2$ is partitioned into five regions — $\Omega = \Omega_0 \cup \Omega_1 \cup \Omega_2 \cup \Omega_3 \cup \Omega_4$ as shown in Fig. 7. The left boundary of the domain (Γ_{in}) is associated with an input heat flux of unit magnitude, the top and bottom boundaries (Γ_{N}) are associated with a Neumann boundary condition with zero flux and finally the right boundary (Γ_{D}) is fixed at zero. The state variable is the temperature $\theta(\mathbf{w}, t)$ at a given spatial location $\mathbf{w} \in \Omega$, for a given time t. The initial condition is $\theta(\mathbf{w}, 0) = 0$. The output is the average temperature measured at Ω_2. The problem is parametrized by the heat conductivity γ in the subdomains ($\Omega_0, \Omega_1, \Omega_2, \Omega_3$ and Ω_4); $\gamma(\mathbf{w}, \boldsymbol{\mu}) = 1$ when $\mathbf{w} \in \Omega_0$ and $\gamma(\mathbf{w}, \boldsymbol{\mu}) = \kappa_i$ whenever $\mathbf{w} \in \Omega_i$, $i = 1, 2, 3, 4$. We define the parameter vector $\boldsymbol{\mu} = [\kappa_1, \kappa_2, \kappa_3, \kappa_4]$. The governing PDE is discretized in space using linear finite elements with respect to a simplicial triangulation of the domain Ω obtained via the software gmsh [25]. It is further discretized in time using the implicit Euler scheme for a time ranging from $t \in [0, 1]$, with step size $\Delta t = 0.01$. The spatially discretized system has dimension $n = 7488$. For more details on the model and the spatial discretization, the reader is referred to [51]. The discretized heat equation can be written in the form of (2.2). Since the problem is linear, we have $\mathbf{f} \equiv 0$. For the numerical results, the parameter $\boldsymbol{\mu}$ is sampled from the domain $\mathcal{P} := [1 \cdot 10^{-5}, 1 \cdot 10^{-2}] \times [1 \cdot 10^{-5}, 1 \cdot 10^{-2}] \times [1 \cdot 10^{-4}, 1] \times [1 \cdot 10^{-1}, 1]$. For purposes
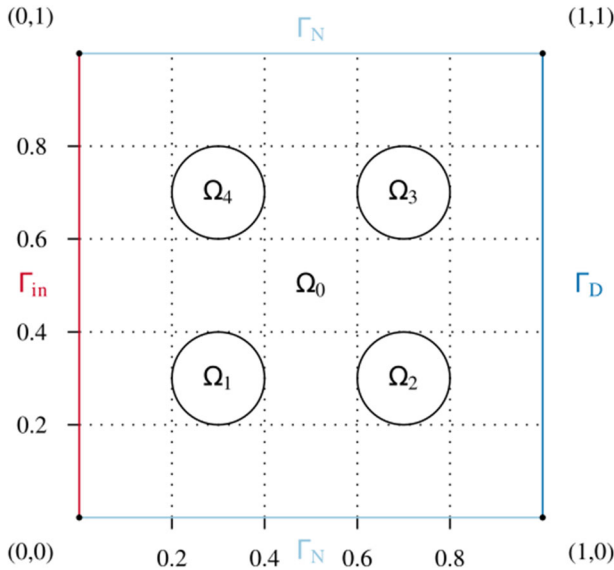
Fig. 7 Thermal Block Example: Spatial domain and boundaries

of illustration, we consider the three parameter version of the thermal block problem by fixing κ_4 to its mean value, i.e., $\kappa_4 = 0.5$. The training set $\varXi_{\text{train}}$ consists of a tensor grid of $N_{\text{train}} = 6^3 = 216$ parameters, with 6 parameters sampled for each κ_i, $i = 1, 2, 3$. The test parameter set consists of 100 parameters, randomly sampled from $\mathcal{P}$. The tolerance for the greedy algorithm is set to be $\texttt{tol} = 1 \cdot 10^{-3}$.

4.2.1 Greedy Algorithm with Fixed Training Set

Applying Algorithm 2.1 with a fixed training set to the thermal block example results in a ROM of dimension $r_{\text{POD}} = 74$, taking 53 iterations to converge in $t_{\text{greedy}} = 694.73$ seconds. The maximum error over the test set is $\epsilon_t^{\max} = 9.78 \cdot 10^{-4}$. In Fig. 8, the training set $\varXi_{\text{train}}$ and the greedy parameters identified by Algorithm 2.1 are shown. Of the 216 parameters in the training set, only 44 are chosen. The greedy parameters have a larger concentration at and around $(0.01, 0.01, 0.0001)$, the upper right corner of the figure. In fact, the regions around the vicinity of the upper and right wall of the grid posses many greedy samples near them.

4.2.2 Greedy Algorithm Schemes 1 and 2

Similar to the Burgers' equation, we now apply the proposed training set subsampling schemes to the thermal block example. We shall also illustrate the advantages of using oversampling. For both Algorithms 3.1 and 3.2, we consider $\epsilon_{\text{QR}}, \epsilon_{\text{SVD}} = 1 \cdot 10^{-10}$ and a coarse tolerance $\texttt{tol}^c = 1$. The approximation to $\mathfrak{Y}$ is obtained by taking snapshots at every time step of the implicit Euler scheme. The results are summarized in Tables 4 and 5 for Algorithms 3.1 and 3.2, respectively. The first scheme does not lead to a successful ROM for both QDEIM and KDEIM whereas using the pivoted QR decomposition on $\widetilde{\mathfrak{Y}}^{\mathsf{T}}$ to identify the subsampled training set produces a successful ROM.
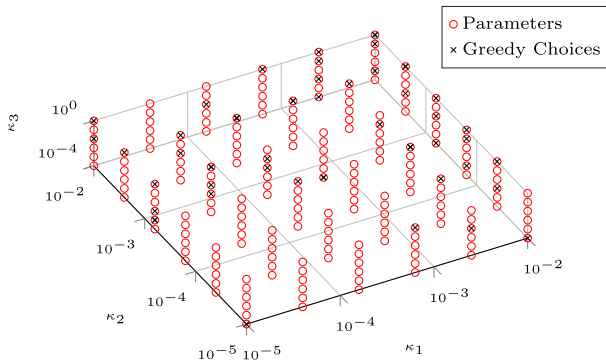
Fig. 8 Thermal Block: Fine training set with 216 parameters and the 44 greedy parameters picked by Algorithm 2.1

Table 4 Thermal Block Results for Algorithm 3.1 for QDEIM, KDEIM and QR

Method	Fixed	Adapted		
		$\epsilon_{\text{SVD}}, \epsilon_{\text{QR}} = 1 \cdot 10^{-10}$		
		QDEIM	KDEIM	QR
N_c	216	19	19	20
$\epsilon_t^{\max}$	$9.78 \cdot 10^{-4}$	$1.10 \cdot 10^{-3}$	$2.10 \cdot 10^{-3}$	$6.45 \cdot 10^{-4}$
r_{POD}	74	64	64	66
Iterations (iter)	53	42	43	44
Offline time (s)	694.73	121.36	123.27	126.35
Speedup	–	5.7	5.6	5.5

For the second scheme of the proposed algorithm, both QDEIM and KDEIM result in a subsampled training set of cardinality $N_c = 19$ while the pivoted QR approach gives $N_c = 20$. However, both QR and QDEIM are unsuccessful in meeting the required ROM tolerance for the test set. On the other hand, KDEIM results in a ROM satisfying the tolerance, taking a significantly smaller number of iterations (40) to converge. The results seem to indicate that the subsampling approach is not entirely able to capture the full range of features over the training set. This is mainly due to the smaller number of parameters $N_c = 19$, that the algorithm results in. Recall that for the standard greedy approach, 44 unique greedy parameters were determined. However, it is also to be noted that the performance of the ROMs resulting from either scheme on the test set is not bad. The maximum error is only slightly higher than the desired tolerance of tol $= 1 \cdot 10^{-3}$.

Next, we perform oversampling to identify more parameters. For the standard DEIM approach, the number of interpolation points is equal to the rank ℓ of the truncated left singular vectors of the snapshots matrix. For oversampling, we set the number of interpolation points to be $m = 2\ell$ and test the approaches based on maximizing the smallest singular value (Gappy-POD Eigenvector) and the approach based on clustering (Gappy-POD Clustering), both originally proposed in [45]. The results are given in Table 6. We see that both oversampling approaches result in ROMs that are validated to be accurate over the test set. The Gappy-POD Clustering method results in the smallest test error among the two and

Table 5 Thermal Block Results for Algorithm 3.2 for QDEIM, KDEIM and QR

Method	Fixed	Adapted		
		$\epsilon_{\text{SVD}}, \epsilon_{\text{QR}} = 1 \cdot 10^{-10}$		
		QDEIM	KDEIM	QR
N_{train}	216	19	19	20
$\epsilon_t^{\max}$	$9.78 \cdot 10^{-4}$	$1.10 \cdot 10^{-3}$	$9.36 \cdot 10^{-4}$	$1.60 \cdot 10^{-3}$
r_{POD}	74	65	62	57
Iterations (`iter`)	53	45	40	35
Offline time (s)	694.73	121.58	110.13	73.38
Speedup	–	5.7	6.3	9.5

Table 6 Thermal Block Results for Algorithm 3.1 with oversampling

Method	Oversampling	
	$m = 2\ell$	
	Gappy-POD Eigenvector	Gappy-POD Clustering
N_c	38	38
$\epsilon_t^{\max}$	$9.91 \cdot 10^{-4}$	$7.96 \cdot 10^{-4}$
r_{POD}	70	62
Iterations (`iter`)	47	40
Offline time (s)	177.51	151.39
Speedup	3.9	4.6

takes 40 iterations to converge. Notice that, compared to the previous two approaches based on QDEIM and KDEIM, the oversampling approach requires more time. This is not surprising, since a larger number of parameters is included in the coarse training set at Stage 2 of Algorithms 3.1 and 3.2. The speedup of the Gappy-POD Eigenvector variant is 3.9, while a speedup of 4.6 is achieved by the Gappy-POD Clustering variant. We show the subsampled training sets of both the approaches in Fig. 9. In particular, parameter samples anticipated by the Gappy-POD Clustering variant bear a close resemblance to the greedy parameter distribution in Fig. 8. In Fig. 10, we plot the mean error over time for each parameter in the test set. It is evident that both the proposed oversampling strategies, Gappy-POD Eigenvector (Fig. 10a) and Gappy-POD Clustering (Fig. 10b) have been successful in keeping the error below the desired tolerance, uniformly for all the parameters in the test set. Lastly, for the thermal block example, the total number of iterations to converge for Algorithms 2.1, 3.1 and 3.2 is not equal and varies based on the particular subsampling strategy used (see Tables 4, 5 and 6). Our initial assumptions in Sect. 3.2.3 that $N_I \approx N_{II,1} + N_{II,2}$, $N_I \approx N_{III,1} + N_{III,2}$ do not hold while we see that $N_I \gtrapprox N_{II,1} + N_{II,2}$, $N_I \gtrapprox N_{III,1} + N_{III,2}$ is always the case.
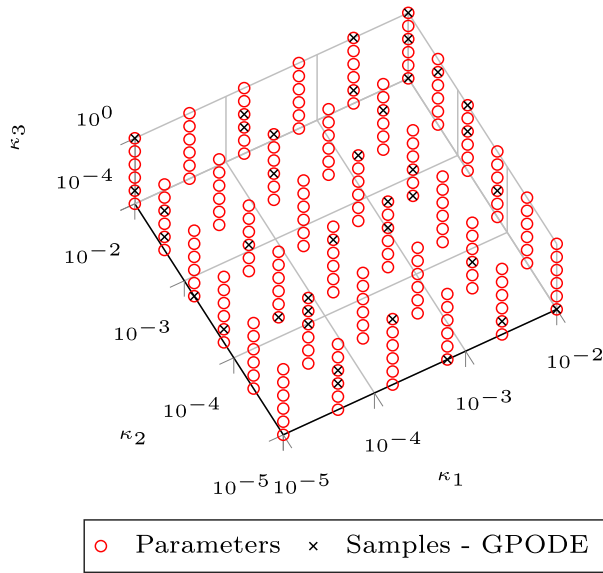
(a) Subsampled parameters using Gappy-POD Eigenvector.
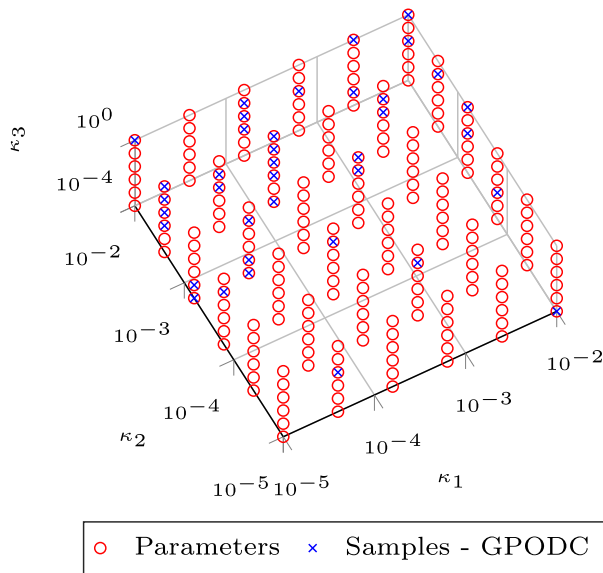
(b) Subsampled parameters using Gappy-POD Clustering.

Fig. 9 Subsampling strategy using Gappy-POD with oversampling for the Thermal Block

(a) Gappy-POD Eigenvector.
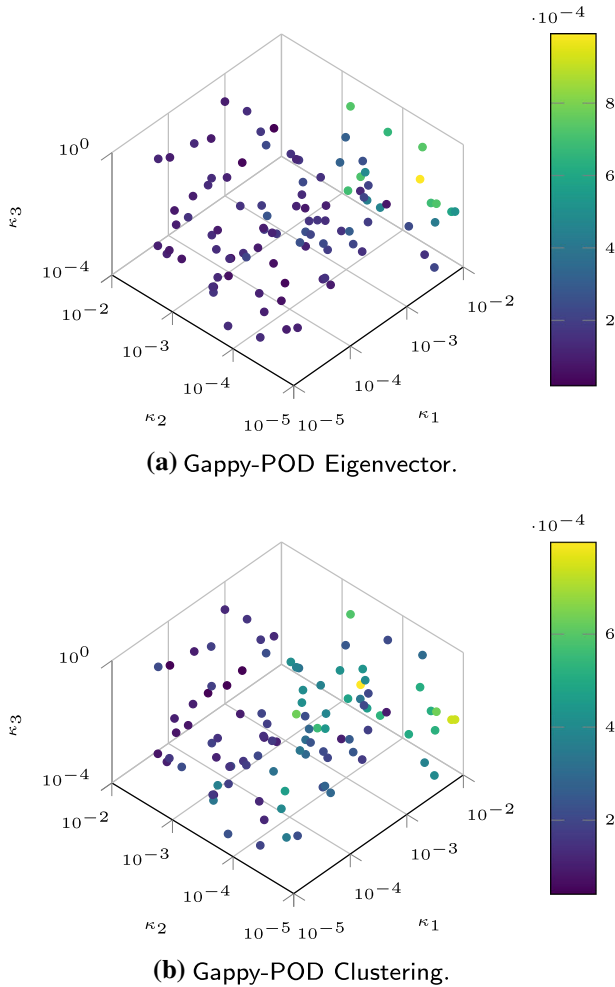

(b) Gappy-POD Clustering.

Fig. 10 Error plot for Algorithm 3.1 with coarse tolerance $\texttt{tol}^c = 1$ and subsampling based on Gappy-POD Eigenvector and Gappy-POD Clustering applied to the thermal block example. The mean error over time between the true and reduced outputs - $(1/K + 1)\sum_{i=0}^{K} \| \mathbf{y}\left(\mathbf{x}(t^k, \boldsymbol{\mu})\right) - \widetilde{\mathbf{y}}\left(\mathbf{z}(t^k, \boldsymbol{\mu})\right) \|$ - is plotted for all parameters in the test set

5 Conclusions

We presented an efficient method to subsample the training set in the offline stage of the RBM. The proposed two-stage strategy is goal-oriented. It uses the pivoted QR decomposition, the DEIM algorithm or its variants to approximate the parameter-to-output map for time-dependent problems, taking advantage of the information from the pivots or the interpolation points to generate the subsampled training set. Different strategies to identify the interpolation points based on variants of the DEIM algorithm were discussed. The strategy of retaining as many parameters as the number of DEIM interpolation points is simple and leads to considerable speedup. However, there may be occasions where it is not robust, as

demonstrated in the thermal block example. For such scenarios, we propose a principled oversampling strategy based on the Gappy-POD approach, to add additional parameters to the subsampled training set. This approach, while slightly more expensive, is robust and leads to reliable ROMs. The different subsampling strategies were tested on two numerical examples and were shown to yield, on average, a speedup of up to 5 for the offline stage of the RBM, without compromising the quality of the generated ROMs.

There exist several promising possibilities for further applications and improvements. One possible idea is in relation to the KDEIM-based subsampling strategy. It is worthwhile to consider the development of localized RBMs for each cluster identified by the algorithm. Such an approach could help overcome existing challenges related to training set adaptation methods that partition the parameter domain based on binary trees [23,30]. Another promising idea is related to data assimilation within the model-based RBM framework. Assuming output data is available by means of sensors or other measurements, it should be possible to consider a QR decomposition of this parametric data matrix to identify a good initial training set. Such an approach is capable of incorporating data in a natural fashion, to aid the development of problem-tailored ROMs.

Code Availability The companion code and data used to compute the results are available at https://doi.org/ 10.5281/zenodo.4593144 under the MIT License.

References

1. Aanonsen, T.: Empirical interpolation with application to reduced basis approximations. Master's thesis, Master Thesis, Norwegian University of Science and Technology (NTNU), Trondheim, Norway (2009). URL http://hdl.handle.net/11250/258487
2. Antil, H., Chen, D., Field, S.: A note on qr-based model reduction: Algorithm, software, and gravitational wave applications. Comput. Sci. Eng. **20**(4), 10–25 (2018). https://doi.org/10.1109/MCSE.2018.042781323
3. Antoulas, A.C.: Approximation of Large-Scale Dynamical Systems, Adv. Des. Control, vol. 6. SIAM Publications, Philadelphia, PA (2005). https://doi.org/10.1137/1.9780898718713
4. Antoulas, A.C., Beattie, C.A., Gugercin, S.: Interpolatory Methods for Model Reduction. Society for Industrial and Applied Mathematics, Philadelphia, PA (2020). https://doi.org/10.1137/1.9781611976083
5. Barrault, M., Maday, Y., Nguyen, N.C., Patera, A.T.: An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. C.R. Acad. Sci. Paris **339**(9), 667–672 (2004). https://doi.org/10.1016/j.crma.2004.08.006
6. Benaceur, A., Ehrlacher, V., Ern, A., Meunier, S.: A progressive reduced basis/empirical interpolation method for nonlinear parabolic problems. SIAM J. Sci. Comput. **40**(5), A2930–A2955 (2018). https://doi.org/10.1137/17M1149638
7. Benner, P., Cohen, A., Ohlberger, M., Willcox, K. (eds.): Model Reduction and Approximation: Theory and Algorithms. Computational Science & Engineering. SIAM Publications, Philadelphia, PA (2017). https://doi.org/10.1137/1.9781611974829

8. Benner, P., Grivet-Talocia, S., Quarteroni, A., Rozza, G., Schilder W. Silveira, L.M. (eds.): Model Order Reduction. Volume 3: Applications. De Gruyter (2021). https://doi.org/10.1515/9783110499001

9. Benner, P., Gugercin, S., Willcox, K.: A survey of model reduction methods for parametric systems. SIAM Review **57**(4), 483–531 (2015). https://doi.org/10.1137/130932715

10. Boutsidis, C., Mahoney, M.W., Drineas, P.: An improved approximation algorithm for the column subset selection problem. In: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 968–977. SIAM, Philadelphia, PA (2009)

11. Broadbent, M.E., Brown, M., Penner, K.: Subset selection algorithms: Randomized vs. deterministic. SIAM Undergrad. Res. Online **3**, 50–71 (2010). https://doi.org/10.1137/09S010435

12. Carlberg, K., Amsallem, D., Avery, P., Zahr, M., Farhat, C.: The GNAT nonlinear model reduction method and its application to fluid dynamics problems. 6th AIAA Theoretical Fluid Mechanics Conference, Honolulu pp. 1–24 (2011). https://doi.org/10.2514/6.2011-3112

13. Çivril, A., Magdon-Ismail, M.: Column subset selection via sparse approximation of SVD. Theoret. Comput. Sci. **421**, 1–14 (2012). https://doi.org/10.1016/j.tcs.2011.11.019

14. Chan, T.F.: Rank revealing QR factorizations. Linear Algebra Appl. **88**(89), 67–82 (1987). https://doi.org/10.1016/0024-3795(87)90103-0

15. Chaturantabut, S., Sorensen, D.C.: Nonlinear model reduction via discrete empirical interpolation. SIAM J. Sci. Comput. **32**(5), 2737–2764 (2010). https://doi.org/10.1137/090766498

16. Chellappa, S., Feng, L., Benner, P.: Adaptive basis construction and improved error estimation for parametric nonlinear dynamical systems. Internat. J. Numer. Methods Engrg. **121**(23), 5320–5349 (2020). https://doi.org/10.1002/nme.6462

17. Chellappa, S., Feng, L., Benner, P.: An adaptive sampling approach for the reduced basis method. In: Realization and Model Reduction of Dynamical Systems - A Festschrift in Honor of the 70th Birthday of Thanos Antoulas. Springer (2021). URL https://arxiv.org/abs/1910.00298. Accepted March 2020

18. Chen, P., Ghattas, O.: Hessian-based sampling for high-dimensional model reduction. Int. J. Uncertain. Quantif. **9**(2), 103–121 (2019). https://doi.org/10.1615/Int.J.UncertaintyQuantification.2019028753

19. Chen, P., Quarteroni, A., Rozza, G.: Reduced basis methods for uncertainty quantification. SIAM/ASA J. Uncertain. Quantif. **5**(1), 813–869 (2017). https://doi.org/10.1137/151004550

20. Constantine, P.G., Dow, E., Wang, Q.: Active subspace methods in theory and practice: applications to kriging surfaces. SIAM J. Sci. Comput. **36**(4), A1500–A1524 (2014). https://doi.org/10.1137/130916138

21. Drmač, Z., Gugercin, S.: A new selection operator for the discrete empirical interpolation method–improved a priori error bound and extensions. SIAM J. Sci. Comput. **38**(2), A631–A648 (2016). https://doi.org/10.1137/15M1019271

22. Duersch, J.A., Gu, M.: Randomized QR with column pivoting. SIAM J. Sci. Comput. **39**(4), C263–C291 (2017). https://doi.org/10.1137/15M1044680

23. Eftang, J.L., Knezevic, D.J., Patera, A.T.: An hp certified reduced basis method for parametrized parabolic partial differential equations. Math. Comput. Model. Dyn. Syst. **17**(4), 395–422 (2011). https://doi.org/10.1080/13873954.2011.547670

24. Everson, R., Sirovich, L.: Karhunen-Loève procedure for gappy data. J. Opt. Soc. Am. A: **12**(8), 1657–1664 (1995)

25. Geuzaine, C., Remacle, J.F.: Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. Internat. J. Numer. Methods Eng. **79**(11), 1309–1331 (2009). https://doi.org/10.1002/nme.2579

26. Grepl, M., Maday, Y., Nguyen, N.C., Patera, A.T.: Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations. ESAIM: Math. Model. Numer. Anal. **41**(3), 575–605 (2007). https://doi.org/10.1051/m2an:2007031

27. Grepl, M.A.: Certified reduced basis methods for nonaffine linear time-varying and nonlinear parabolic partial differential equations. Math. Models Methods Appl. Sci. **22**(3), 1150015, 40 (2012). https://doi.org/10.1142/S0218202511500151

28. Grepl, M.A., Patera, A.T.: A posteriori error bounds for reduced-basis approximations of parametrized parabolic partial differential equations. ESAIM: Math. Model. Numer. Anal. **39**(1), 157–181 (2005). https://doi.org/10.1051/m2an:2005006

29. Haasdonk, B.: Reduced basis methods for parametrized PDEs—a tutorial introduction for stationary and instationary problems. In: Model Reduction and Approximation, Comput. Sci. Eng., vol. 15, pp. 65–136. SIAM, Philadelphia, PA (2017). https://doi.org/10.1137/1.9781611974829.ch2

30. Haasdonk, B., Dihlmann, M., Ohlberger, M.: 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**(4), 423–442 (2011). https://doi.org/10.1080/13873954.2011.547674

31. Haasdonk, B., Ohlberger, M.: Reduced basis method for finite volume approximations of parametrized linear evolution equations. M2AN Math. Model. Numer. Anal. **42**(2), 277–302 (2008). https://doi.org/10.1051/m2an:2008001

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

33. Hesthaven, J.S., Stamm, B., Zhang, S.: Efficient greedy algorithms for high-dimensional parameter spaces with applications to empirical interpolation and reduced basis methods. ESAIM: Math. Model. Numer. Anal. **48**(1), 259–283 (2014). https://doi.org/10.1051/m2an/2013100

34. Hesthaven, J.S., Zhang, S.: On the use of ANOVA expansions in reduced basis methods for parametric partial differential equations. J. Sci. Comput. **69**(1), 292–313 (2016). https://doi.org/10.1007/s10915-016-0194-9

35. Jiang, J., Chen, Y.: Adaptive greedy algorithms based on parameter-domain decomposition and reconstruction for the reduced basis method. Internat. J. Numer. Methods Engrg. **121**(23), 5426–5445 (2020). https://doi.org/10.1002/nme.6544

36. Jiang, J., Chen, Y., Narayan, A.: Offline-enhanced reduced basis method through adaptive construction of the surrogate training set. J. Sci. Comput. **73**(2–3), 853–875 (2017). https://doi.org/10.1007/s10915-017-0551-3

37. Maday, Y., Stamm, B.: Locally adaptive greedy approximations for anisotropic parameter reduced basis spaces. SIAM J. Sci. Comput. **35**(6), A2417–A2441 (2013). https://doi.org/10.1137/120873868

38. Mahoney, M.W.: Algorithmic and statistical perspectives on large-scale data analysis. In: Combinatorial Scientific Computing, Chapman & Hall/CRC Comput. Sci. Ser., pp. 427–469. CRC Press, Boca Raton, FL (2012). https://doi.org/10.1201/b11644-17

39. Mlinarić, P., Grundel, S., Benner, P.: Efficient model order reduction for multi-agent systems using QR decomposition-based clustering. In: 54th IEEE Conference on Decision and Control (CDC), Osaka, Japan, pp. 4794–4799 (2015). https://doi.org/10.1109/CDC.2015.7402967

40. Narayan, A.: Reduced order modeling and numerical linear algebra (2020). URL https://icerm.brown.edu/materials/Slides/sp-s20/Tutorial-Introductory_Talk_-_Snapshot-based_model_reduction_and_numerical_linear_algebra_part_1_]_Akil_Narayan,_University_of_Utah.pdf. ICERM Special Semester on Model and dimension reduction in uncertain and dynamic systems, Brown University, Providence, USA

41. Negri, F., Manzoni, A., Amsallem, D.: Efficient model reduction of parametrized systems by matrix discrete empirical interpolation. J. Comput. Phys. **303**, 431–454 (2015). https://doi.org/10.1016/j.jcp.2015.09.046

42. Nguyen, N.C., Rozza, G., Patera, A.T.: Reduced basis approximation and a posteriori error estimation for the time-dependent viscous Burgers' equation. Calcolo **46**(3), 157–185 (2009). https://doi.org/10.1007/s10092-009-0005-x

43. Paul-Dubois-Taine, A., Amsallem, D.: An adaptive and efficient greedy procedure for the optimal training of parametric reduced-order models. Internat. J. Numer. Methods Engrg. **102**(5), 1262–1292 (2015). https://doi.org/10.1002/nme.4759

44. Peherstorfer, B., Butnaru, D., Willcox, K., Bungartz, H.J.: Localized discrete empirical interpolation method. SIAM J. Sci. Comput. **36**(1), A168–A192 (2014). https://doi.org/10.1137/130924408

45. Peherstorfer, B., Drmač, Z., Gugercin, S.: Stability of discrete empirical interpolation and gappy proper orthogonal decomposition with randomized and deterministic sampling points. e-prints arXiv:1808.10473v3, (2018)

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

47. Peherstorfer, B., Willcox, K.: Data-driven operator inference for nonintrusive projection-based model reduction. Comp. Meth. Appl. Mech. Eng. **306**, 196–215 (2016). https://doi.org/10.1016/j.cma.2016.03.025

48. Peherstorfer, B., Zimmer, S., Bungartz, H.J.: Model reduction with the reduced basis method and sparse grids. In: Sparse grids and applications, Lect. Notes Comput. Sci. Eng., vol. 88, pp. 223–242. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-31703-3

49. Quarteroni, A., Manzoni, A., Negri, F.: Reduced Basis Methods for Partial Differential Equations, La Matematica per il 3+2, vol. 92. Springer International Publishing (2016). https://doi.org/10.1007/978-3-319-15431-2

50. Quarteroni, A., Rozza, G., Manzoni, A.: Certified reduced basis approximation for parametrized partial differential equations and applications. J. Math. Ind. **1**, Art. 3, 44 (2011). https://doi.org/10.1186/2190-5983-1-3

51. Rave, S., Saak, J.: A non-stationary thermal-block benchmark model for parametric model order reduction. e-print 2003.00846, arXiv (2020). URL https://arxiv.org/abs/2003.00846. Math.NA
52. Rozza, G., Hess, M., Stabile, G., Tezzele, M., Ballarin, F.: Basic ideas and tools for projection-based model reduction of parametric partial differential equations. In: P. Benner, S. Grivet-Talocia, A. Quarteroni, G. Rozza, W. Schilders, L.M. Silveira (eds.) Snapshot-Based Methods and Algorithms, pp. 1–47. De Gruyter (2021). https://doi.org/10.1515/9783110671490-001
53. Rozza, G., Huynh, D.B.P., Patera, A.T.: Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations: application to transport and continuum mechanics. Arch. Comput. Methods Eng. **15**(3), 229–275 (2008). https://doi.org/10.1007/s11831-008-9019-9
54. Saibaba, A.K.: Randomized discrete empirical interpolation method for nonlinear model reduction. SIAM J. Sci. Comput. **42**(3), A1582–A1608 (2020). https://doi.org/10.1137/19M1243270
55. Sen, S.: Reduced-basis approximation and a posteriori error estimation for many-parameter heat conduction problems. Numerical Heat Transfer, Part B: Fundamentals **54**(5), 369–389 (2008). https://doi.org/10.1080/10407790802424204
56. Tezzele, M., Ballarin, F., Rozza, G.: Combined parameter and model reduction of cardiovascular problems by means of active subspaces and POD-Galerkin methods. In: Mathematical and numerical modeling of the cardiovascular system and applications, SEMA SIMAI Springer Ser., vol. 16, pp. 185–207. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96649-6_8
57. Zha, H., He, X., Ding, C., Gu, M., Simon, H.D.: Spectral relaxation for k-means clustering. In: T.G. Dietterich, S. Becker, Z. Ghahramani (eds.) Advances in Neural Information Processing Systems 14, pp. 1057–1064. MIT Press (2002). http://papers.nips.cc/paper/1992-spectral-relaxation-for-k-means-clustering.pdf
58. Zhang, Y., Feng, L., Li, S., Benner, P.: An efficient output error estimation for model order reduction of parametrized evolution equations. SIAM J. Sci. Comput. **37**(6), B910–B936 (2015). https://doi.org/10.1137/140998603

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.