Adaptive POD-DEIM basis construction and its application to a nonlinear population balance system

Lihong Feng* Michael Mangold† Peter Benner ‡

April 4, 2017

Abstract

We propose an adaptive algorithm for constructing reduced-order models of nonlinear systems based on proper orthogonal decomposition (POD) combined with the discrete empirical interpolation method (DEIM). Using an efficient output error estimation, the reduced basis and the DEIM interpolation basis are adaptively adjusted to derive a small, yet accurate reduced-order model. The adaptive algorithm is further explored for a population balance system of a crystallization process. Simulation results show that much smaller and reliable reduced-order models can be adaptively obtained using the algorithm with ignorable extra computational load as compared with the standard POD-DEIM method.

Introduction

Distributed systems with spatial and property coordinates, such as the model of a crystallizer, are widely researched in chemical engineering. Simulation of the high fidelity models derived from, e.g., finite element or finite volume discretization, is very time consuming, because of

*Lihong Feng is with Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany. feng@mpi-magdeburg.mpg.de

†Michael Mangold is with Technische Hochschule Bingen, Berlinstrae 109, 55411 Bingen, Germany. m.mangold@th-bingen.de

‡Peter Benner is with Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany. benner@mpi-magdeburg.mpg.de

the high system order (degrees of freedom) and complexity. For the purpose of optimization, predictive control, and other design tasks, surrogate models with much smaller size are often necessary to meet the computational demands, such as limited memory, etc.

Model order reduction (MOR) has become very popular in accelerating simulation of large-scale systems. Using model order reduction, a small-scale, yet accurate model, is derived and is used as surrogate of the large-scale system. Fast simulation can be achieved by only simulating the surrogate reduced-order model. So far, there are many MOR methods proposed, among which proper orthogonal decomposition (POD)[1–5] is one of the most robust and widely used methods for model order reduction of nonlinear dynamical systems. Furthermore, POD has already been applied to population balance systems in crystallization or granulation processes[6–8] and the use of data-based POD for the construction of empirical eigenfunctions for model order reduction has been examined in a variety of distributed parameter processes, both transport-reaction processes modeled by parabolic PDEs and particulate processes described by population balance equations.[9–11] In contrast to the POD method, methods of moments are widely used[12] for a condensed description of the system's dependence on internal coordinates; in this case, the reduced model does not preserve the full information on the particles property distributions, but only on some of their moments.

When POD is applied to nonlinear systems, usually it is combined with interpolation techniques, such as the empirical interpolation method (EIM),[13] or the discrete empirical interpolation method (DEIM),[14] to get a reduced-order model (ROM) with reduced nonlinear complexity. The ROM is derived by computing the POD basis from the singular value decomposition of the snapshot matrix of the solution trajectories. The (D)EIM method is realized by computing an interpolation basis from the snapshots of the nonlinear function of the original full-order model (FOM). Both, the POD basis dimension, and the interpolation basis dimension are usually determined empirically by looking at the decay of the singular values of the snapshot matrices. Usually, the dimension of the reduced basis and of the interpolation basis are taken conservatively large to make sure that the ROM meets the accuracy requirements.

In this work, instead of using empirical observation, we propose to integrate an efficient

error estimation[15] into an adaptive scheme to formulate an automatic way of updating both bases iteratively, and to avoid including redundant basis vectors, once the error estimation is below a tolerance. Similar work on properly matching the reduced basis with the interpolation basis has been proposed in,[16] where the reduced basis method combined with the EIM is employed to obtain parametric ROMs for parameter-dependent systems. The approach from[16] updates both the reduced basis and the interpolation basis simultaneously using the snapshots corresponding to the parameters at which the FOM is worst approximated by the ROM. The adaption was done by trivially adding a single new member to the basis at every iteration. Our proposed method adjusts the dimension of the basis in a different but more efficient way, by which the number of new basis vectors to be added, is adaptively adjusted. It is currently applicable to non-parametric systems, but could be extended to parametric systems. A method of adapting the DEIM basis is proposed in,[17] where the number of the DEIM basis vectors is updated online to better approximate the nonlinear function when it is evaluated at a state which causes big error of the DEIM interpolation. The method considers adapting only the DEIM basis, rather than both the reduced basis and the DEIM basis. A further difference is that the proposed adaptivity scheme is completed in the offline phase, while the method in[17] adapts the DEIM basis in the online phase. The proposed adaptivity method is more suitable when the nonlinear part of the system is included in a low dimensional subspace, and the offline computed interpolation can already approximate it very well, whereas the method in[17] is more suitable when the nonlinear behavior of the system is very complex and offline interpolation becomes inaccurate without online updating.

Furthermore, various adaptive MOR methods have been proposed. The fully adaptive method in[18] adaptively chooses the expansion points and the moments of the transfer function for linear non-parametric systems, see also the references there. There are also online adaptive methods, such as the work in,[19,20] where the reduced-order matrices are constructed online by interpolation, using the offline computed reduced matrices. In this paper, the proposed method addresses a different adaptivity problem, where adaptive adjustment of both the reduced basis and the DEIM basis is the focus of this work.

As mentioned above, we use the error estimation from[15] to automatically construct the basis. It has been tested to be reliable in,[8] yet without being further utilized to automatically construct the ROM. Motivated by the robustness of the error estimation shown in,[8] we successfully exploit the error estimation in bases and ROM construction for the population balance model.

Developing efficient error estimation (bounds) has been an active topic in MOR. Robust error bounds exist for the balanced truncation method,[21–23] moment-matching,[24] as well as the reduced basis method,[25–29] which, nevertheless, are only valid for linear systems, or special (quadratic, cubic) nonlinear systems.[26,27] Error estimation for general nonlinear systems is considered, e.g. in,[15,16,30–32] where the error estimation from[16,30] estimates the error of the state vector. For many applications, the output response rather than the state vector is of importance, and proper error estimation for the output error is preferred. In,[15,32] output error estimators are proposed, and the one from[15] has been shown to be most efficient. The output error estimation in[15] is proposed for general parametric nonlinear systems. In this paper, we have simplified the expression of the error estimation and adapted it to non-parametric nonlinear systems.

Compared with the standard POD-DEIM method, the main contributions of this paper can be summarized as follows. An algorithm is proposed to make the standard POD-DEIM method automatic. An error estimator is successfully applied, so that the ROM derived by POD-DEIM is guaranteed to be reliable.

The paper can be divided into two main parts. The methodological part reviews the POD and DEIM method, and proposes the adaptive POD-DEIM algorithm based on an a posteriori error estimation. The second part addresses application of the adaptive POD-DEIM algorithm to a population balance system of a crystallization process. The experimental results are shown afterwards. Conclusions are given in the end.

POD and DEIM

The proper orthogonal decomposition (POD) method is a widely used approach for model order reduction of linear and nonlinear dynamical systems.[2–5,7,8]

Given a nonlinear system as below

$$\begin{aligned} E\frac{d\mathbf{x}(\mathbf{t})}{dt} &= A\mathbf{x}(t) + \mathbf{f}(\mathbf{x}(\mathbf{t})), \\ \mathbf{y}(t) &= C^T\mathbf{x}(t), \end{aligned} \tag{1}$$

where $E \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}, \mathbf{f} : \mathbb{R}^{\mathcal{N}} \mapsto \mathbb{R}^{\mathcal{N}}, B \in \mathbb{R}^{\mathcal{N}\times m_I}, C \in \mathbb{R}^{\mathcal{N}\times m_O}$, and $\mathbf{x}(t) \in \mathbb{R}^{\mathcal{N}}$ is the state vector, $\mathbf{y} \in \mathbb{R}^{m_O}$ is the output. The key component of POD is the singular value decomposition (SVD) of a snapshot matrix (defined in Algorithm 1). In the following, the SVD is first introduced, and the optimality property of the POD basis is presented, then the POD algorithm is presented in Algorithm 1.

MOR via POD

POD basis For any matrix $X \in \mathbb{R}^{\mathcal{M}\times\mathcal{N}}$, there exist $\tilde{U} = (\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_{\mathcal{M}}) \in \mathbb{R}^{\mathcal{M}\times\mathcal{M}}$ and $\tilde{V} = (\tilde{\mathbf{v}}_1, \ldots, \tilde{\mathbf{v}}_{\mathcal{N}}) \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$, s.t.

$$X = \tilde{U}\Sigma\tilde{V}^T, \quad \text{or} \quad \tilde{U}^T X\tilde{V} = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} := \Sigma \in \mathbb{R}^{\mathcal{M}\times\mathcal{N}}. \tag{2}$$

Here, $D = \text{diag}(\sigma_1, \ldots, \sigma_{d_X})$, with $\sigma_1 \geq \ldots \geq \sigma_{d_X} > 0$. The matrices $\tilde{U}, \tilde{V}$ are orthogonal matrices, i.e. $\tilde{U}^T\tilde{U} = I_{\mathcal{M}}$, and $\tilde{V}^T\tilde{V} = I_{\mathcal{N}}$. $I_{\mathcal{M}} \in \mathbb{R}^{\mathcal{M}\times\mathcal{M}}$ and $I_{\mathcal{N}} \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ are identity matrices of proper dimensions.

For $l_X \in \{1, \ldots, d_X\}$, the vectors $\tilde{\mathbf{u}}_i, i = 1, \ldots, l_X$, define the POD basis of rank l_X. The POD basis $\{\tilde{\mathbf{u}}_i\}_{i=1}^{l_X}$ is optimal, among all rank l_X orthogonal approximations of the columns of X,[33] i.e.

$$\{\tilde{\mathbf{u}}_i\}_{i=1}^{l_X} = \arg\min_{\mathbf{u}_1,\ldots,\mathbf{u}_{l_X}\in\mathbb{R}^{\mathcal{M}}} \sum_{j=1}^{\mathcal{N}} \varepsilon_j, \qquad s.t. \langle \mathbf{u}_i, \mathbf{u}_j\rangle_{\mathbb{R}^{\mathcal{M}}} = \delta_{ij}, 1 \leq i, j \leq l_X. \tag{3}$$

Here, $\varepsilon_j = \|\mathbf{x}_j - \sum_{i=1}^{l_X}\langle \mathbf{x}_j, \mathbf{u}_i\rangle_{\mathbb{R}^{\mathcal{M}}}\mathbf{u}_i\|^2_{\mathbb{R}^{\mathcal{M}}}$, and $\mathbf{x}_j$ is the jth column of X.

Algorithm 1 POD algorithm

Input: A sufficiently small SVD tolerance, e.g., $\epsilon_{svd} = 10^{-10}$. Discrete time points $t_1, \ldots, t_m$ in the interesting time interval.

Output: The projection matrix V.

1: Numerically solve the original nonlinear system (1) to get the solutions $\mathbf{x}(t_i)$ at m time instances t_i, $i = 1, \ldots, m$, which are called snapshots and form the snapshot matrix:

$$X = (\mathbf{x}(t_1), \ldots, \mathbf{x}(t_m)).$$

2: Get the projection matrix V of rank r from the SVD of X:

$$X = \tilde{U}\Sigma\tilde{V}^T, V = (\tilde{u}_1, \ldots, \tilde{u}_r),$$

with r satisfying

$$\frac{\sum_{i=r+1}^{d_X} \sigma_i}{\sum_{i=1}^{d_X} \sigma_i} < \epsilon_{svd}. \tag{5}$$

Here Σ is defined as in (2).

ROM construction The POD method uses the POD basis of the snapshot matrix to construct the projection matrix $V \in \mathbb{R}^{\mathcal{N}\times r}$ for model order reduction. The reduced order model (ROM) is obtained via Galerkin projection onto the subspace spanned by the columns of V, and can be written as

$$\begin{aligned} V^T E V \frac{d\mathbf{z}(t)}{dt} &= V^T A V \mathbf{z}(t) + V^T \mathbf{f}(V\mathbf{z}(t)), \\ \hat{\mathbf{y}} &= C^T V \mathbf{z}(t), \end{aligned} \tag{4}$$

where the state vector $\mathbf{x}$ of the original system (1) is approximated by $\mathbf{x}(t) \approx V\mathbf{z}(t)$. Algorithm 1 presents the implementation details. In Algorithm 1, r is the rank of V, it is also the order of the ROM, as well as the dimension of the reduced basis, which is given by the columns of V. A standard way of determining r is by observing the ratio defined in (5). When simulating the ROM (4), one needs to compute a vector $\mathbf{f}(V\mathbf{z}(t))$ of full dimension $\mathcal{N}$. It is the main computational load when simulating the ROM. In the next subsection, a method[14] for interpolating the nonlinear function $\mathbf{f}(\mathbf{x}(t))$ is reviewed, which was proposed to reduce the computational complexity of $\mathbf{f}(V\mathbf{z}(t))$.

DEIM: reducing the complexity of the nonlinear part

The DEIM method proposed in[14] tries to interpolate the nonlinear function using the POD basis of the snapshot matrix F of the nonlinear function. The matrix $F = (\mathbf{f}(\mathbf{x}(t_1)), \ldots, \mathbf{f}(\mathbf{x}(t_m)))$ can be obtained from the snapshots of the solution vector, $\mathbf{x}(t_1), \ldots, \mathbf{x}(t_m)$. Using the POD basis $U_{\mathbf{f}} \in \mathbb{R}^{n \times l}$ of F, one may approximate the nonlinear function $\mathbf{f}(\mathbf{x}(t))$ by

$$\mathbf{f}(\mathbf{x}(t)) \approx U_{\mathbf{f}} c(t).$$

By assuming that the approximation $U_{\mathbf{f}} c(t)$ interpolates $\mathbf{f}(\mathbf{x}(t))$ at l entries, i.e.,

$$P^T \mathbf{f}(\mathbf{x}(t)) = P^T U_{\mathbf{f}} c(t),$$

where $P = [e_{\wp_1}, \ldots, e_{\wp_l}]$ is an index matrix consisting of unit vectors $e_{\wp_i}, i = 1, \ldots, l$. The indices $\wp_i$ select the locations of interpolation.

Suppose that $P^T U_{\mathbf{f}} \in \mathbb{R}^{l \times l}$ is nonsingular, then

$$c(t) = (P^T U_{\mathbf{f}})^{-1} P^T \mathbf{f}(\mathbf{x}(t)),$$

so that $\mathbf{f}(\mathbf{x}(t))$ is approximated as

$$\mathbf{f}(\mathbf{x}(t)) \approx U_{\mathbf{f}} (P^T U_{\mathbf{f}})^{-1} P^T \mathbf{f}(\mathbf{x}(t)). \tag{6}$$

Assume that $\mathbf{f}(\mathbf{x}(t))$ is contained in a low dimensional subspace, then we usually have $l \ll \mathcal{N}$, and the interpolant $U_{\mathbf{f}} (P^T U_{\mathbf{f}})^{-1} P^T \mathbf{f}(\mathbf{x}(t))$ can be computed much cheaper than $\mathbf{f}(\mathbf{x}(t))$. More specifically, $U_{\mathbf{f}} (P^T U_{\mathbf{f}})^{-1}$ can be precomputed independent of the time t, and furthermore, the inverse of the small matrix $P^T U_{\mathbf{f}}$ is easy to compute. As for $P^T \mathbf{f}(\mathbf{x}(t))$, instead of computing all the entries of $\mathbf{f}(\mathbf{x}(t))$, only those entries corresponding to the indices $\wp_i, i = 1, \ldots, l$, need to be computed. The implementation details are presented in Algorithm 2.

Remark 1 *When applying the interpolation (6), the matrix P needs not be explicitly applied,*

Algorithm 2 DEIM algorithm[14]

Input: A sufficiently small SVD tolerance, e.g., $\epsilon_{svd} = 10^{-10}$. The snapshot matrix $F = (\mathbf{f}(\mathbf{x}(t_1)), \ldots, \mathbf{f}(\mathbf{x}(t_m)))$.

Output: The DEIM basis $U_{\mathbf{f}}$ and the indices $\wp_1, \ldots, \wp_l$.

1: Take the first l POD basis vectors of F as the DEIM basis: $[\mathbf{u}_1^F, \ldots, \mathbf{u}_l^F] \in \mathbb{R}^{\mathcal{N}\times l}$, such that l satisfies

$$\frac{\sum_{i=l+1}^{d} \sigma_i}{\sum_{i=1}^{d} \sigma_i} < \epsilon_{svd},$$

where $\sigma_i, i = 1, \ldots, d$ are the nonzero singular values of F.

2: $\wp_1 = \arg\max_{j\in\{1,\ldots,\mathcal{N}\}} |u_{1j}^F|$, where $\mathbf{u}_1^F = (u_{11}^F, \ldots, u_{1\mathcal{N}}^F)^T$.

3: $U_{\mathbf{f}} \leftarrow \mathbf{u}_1^F, P \leftarrow e_{\wp_1}$.

4: for $i = 2$ to l do

5: Solve $(P^T U_{\mathbf{f}})\alpha = P^T \mathbf{u}_i^F$ for $\alpha = (\alpha_1, \ldots, \alpha_{i-1})^T$.

6: $\xi_i = \mathbf{u}_i^F - U_{\mathbf{f}}\alpha$,

7: $\wp_i = \arg\max_{j\in\{1,\ldots,\mathcal{N}\}} |\xi_{ij}|$. Here $\xi_i = (\xi_{i1}, \ldots, \xi_{i\mathcal{N}})^T$.

8: $U_{\mathbf{f}} \leftarrow [U_{\mathbf{f}}, \mathbf{u}_i^F], P \leftarrow [P, e_{\wp_i}]$.

9: end for

instead, only the indices $\wp_i, i = 1, \ldots, l$, are applied to the matrix $U_{\mathbf{f}}$ or the nonlinear vector $\mathbf{f}(\mathbf{x}(t))$. It implicates that $P^T U_{\mathbf{f}}$ simply consists of the rows of $U_{\mathbf{f}}$, corresponding to those indices $\wp_i$, and $P^T\mathbf{f}(\mathbf{x}(t))$ is a shortened vector composed of the few entries of $\mathbf{f}(\mathbf{x}(t))$ corresponding to the indices $\wp_i$.

Here and below, to distinguish the basis for interpolation from the basis for the solution vector $\mathbf{x}(t)$, we call $U_{\mathbf{f}}$ (the column vectors of $U_{\mathbf{f}}$) the DEIM basis, and V (the column vectors of V) the reduced basis. When POD is combined with DEIM, the ROM of the original nonlinear system (1) is in the form of

$$\begin{aligned} V^T E V \tfrac{d\mathbf{z}(t)}{dt} &= V^T A V \mathbf{z}(t) + V^T U_{\mathbf{f}} (P^T U_{\mathbf{f}})^{-1} P^T \mathbf{f}(V\mathbf{z}(t)), \\ \hat{\mathbf{y}}(t) &= C^T V \mathbf{z}(t). \end{aligned} \tag{7}$$

When simulating the ROM, computing the term $V^T U_{\mathbf{f}} (P^T U_{\mathbf{f}})^{-1} P^T \mathbf{f}(V\mathbf{z}(t))$ is much cheaper than computing the term $V^T \mathbf{f}(V\mathbf{z}(t))$ in (4), since $V^T U_{\mathbf{f}} (P^T U_{\mathbf{f}})^{-1}$ can be computed once for all the time instances, and evaluating a short vector $P^T\mathbf{f}(V\mathbf{z}(t)) \in \mathbb{R}^l$ is much faster than evaluating the long vector $\mathbf{f}(V\mathbf{z}(t)) \in \mathbb{R}^{\mathcal{N}}$. Therefore, the ROM (7) obtained by POD-DEIM,

is usually considered as the ROM of the original system.

Adaptive reduced basis and DEIM basis construction

From Algorithm 1 and Algorithm 2, we see that the dimensions of the reduced basis V and the DEIM basis $U_{\mathbf{f}}$ are determined by the tolerance ϵ_{svd} which is set to be very small in general, in order to guarantee the accuracy of the final ROM. On the one hand, a conservatively small ϵ_{svd} may produce a big ROM together with a big number of DEIM basis vectors, making the ROM simulation not as efficient as expected. On the other hand, although a loose (bigger) tolerance constructs a more compact ROM, it loses reliability (accuracy not guaranteed) if used without error estimation.

An efficient output error estimation for general projection based model order reduction methods was proposed in,[15] and has been applied to various linear and nonlinear systems.[8,15] We propose to use the error estimation to check the error of the ROM produced by POD-DEIM. Based on the error estimation, we develop an algorithm for adaptively modifying the number r and l of the reduced basis and the DEIM basis, respectively. The idea is that starting from a set of r POD and l DEIM modes with small values r, l, we keep updating the values of both, using an adaptivity criterion. The ROM is updated accordingly, and its error is checked by the error estimation. The adaptive process continues until the ROM meets the error tolerance ϵ, indicated by the error estimation. Since the adaptivity criterion depends on the error estimation, we first introduce the error estimation, and show its application to the ROM (7). The adaptive algorithm is presented afterwards.

Output error estimation for POD-DEIM

The output error estimation in[15] was developed for general nonlinear parametric systems, with the system matrices E, A, as well as the nonlinear vector $\mathbf{f}(\mathbf{x}(t))$ being parametrized, and it is applicable to general projection based model order reduction methods. Here, we characterize the error estimation in Theorem 1 for the specified nonlinear system (1) with no parameters.

As presented in,[15] the approximate solution to the dual system is obtained by reducing the dual system. Here, we generalize the approximate solution to any possible solution which approximately solves the dual system, and is not necessarily obtained from the reduced dual system.

In the following, we assume that the original system (1) has only a single output, i.e. $C \in \mathbb{R}^{\mathcal{N}}$ is a vector, and the output y is simply a scalar. There exists a direct extension to multiple-output systems.[15]

Assume that the original system and the ROM are numerically solved by the same time discretization scheme with fixed time step for simplicity: $\Delta\tilde{t} = \tilde{t}_{k+1} - \tilde{t}_k, \forall k = 0, \ldots, K$, though the error estimation was described for varying time steps in.[15] The original system (1) is then temporally discretized as

$$\begin{aligned} M\mathbf{x}_{k+1} &= D\mathbf{x}_k + \mathbf{f}(\mathbf{x}_k), \\ y_{k+1} &= C^T\mathbf{x}_{k+1}, \end{aligned} \tag{8}$$

and the ROM (7) is discretized as

$$\begin{aligned} V^TMV\mathbf{z}_{k+1} &= V^TDV\mathbf{z}_k + V^TU_{\mathbf{f}}(P^TU_{\mathbf{f}})^{-1}P^T\mathbf{f}(V\mathbf{z}_k), \\ y_{k+1} &= C^TV\mathbf{z}_{k+1}. \end{aligned} \tag{9}$$

Assume further that $\hat{\mathbf{x}}^{du}$ is an approximate solution to the dual system,

$$M^T\mathbf{x}^{du} = -C. \tag{10}$$

The residual caused by $\hat{\mathbf{x}}_{k+1} := V\mathbf{z}_{k+1}$ is defined as

$$\mathbf{r}_{k+1} := D\hat{\mathbf{x}}_k + \mathbf{f}(\hat{\mathbf{x}}_k) - M\hat{\mathbf{x}}_{k+1},$$

and the residual caused by $\hat{\mathbf{x}}^{du}$ is defined as

$$\mathbf{r}^{du} = -C - M^T\hat{\mathbf{x}}^{du}.$$

The two residuals are involved in constructing the final error estimation, which measures the error of the ROM at each time step $\tilde{t}_k, k = 1, \ldots, K$. Theorem 1 gives the error bound of the ROM obtained by POD-DEIM.

Theorem 1 *Given the nonlinear system (8) and the ROM (9) constructed by POD-DEIM, assume that M is nonsingular, then the output error at time step $\tilde{t}_{k+1}$ is bounded by*

$$|y_{k+1} - \hat{y}_{k+1}| \leq \Phi_{k+1} \|\mathbf{r}_{k+1}\|_2, k = 0, \ldots, K-1,$$

where $\Phi_{k+1} := S_{k+1}(\|M^{-1}\|_2 \|\mathbf{r}^{du}\|_2 + \|\hat{\mathbf{x}}^{du}\|_2)$, and S_{k+1} is a scaling variable defined as

$$S_{k+1} = \frac{\|\tilde{\mathbf{r}}_{k+1}\|_2}{\|\mathbf{r}_{k+1}\|_2},$$

where $\tilde{\mathbf{r}}_{k+1} = D\mathbf{x}_k + \mathbf{f}(\mathbf{x}_k) - M\hat{\mathbf{x}}_{k+1} = M(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1})$.

Theorem 1 is a direct result from Theorem 4.1 in.[15]

Efficiently computing the error bound

To enable efficient computation of the error bound, we need to address the following issues.

- Notice that S_{k+1} involves computing $\tilde{\mathbf{r}}_{k+1}$, for which the snapshots of $\mathbf{f}(\mathbf{x}_k)$, and the true solution vector $\mathbf{x}_k$ at time instances $\tilde{t}_k$ must be computed. To avoid computing the true solution vector, we can simply use the snapshots from the reduced basis construction in Algorithm 1. Usually, the time instances $\tilde{t}_k$ are different from $t_i, i = 1, \ldots, m$, for the snapshot construction (see Algorithm 1), and the number K is larger than m. However, we can use the average value of S_i over t_i to approximate S_k at $\tilde{t}_k$, i.e.

$$S_k \approx S := \frac{\sum_{i=1}^{m} \tilde{\mathbf{r}}(t_i)}{\sum_{i=1}^{m} \mathbf{r}(t_i)}.$$

  Since $\tilde{\mathbf{r}}(t_i)$ can be obtained for free from the snapshots $\mathbf{x}(t_i)$ during the reduced basis construction, no extra computation involving solving the original large system is incurred.

As a result, a simplified output error estimation is given as

$$|y_{k+1} - \hat{y}_{k+1}| \lesssim \bar{\Phi}_{k+1} \|\mathbf{r}_{k+1}\|_2, k = 0, \ldots, K-1,$$

where $\bar{\Phi}_{k+1} := S(\|M^{-1}\|_2 \|\mathbf{r}^{du}\|_2 + \|\hat{\mathbf{x}}^{du}\|_2)$.

- To compute the residual $\mathbf{r}_{k+1}$ in the error estimation, we need to evaluate $\mathbf{f}(\hat{\mathbf{x}}_{k+1})$, which involves evaluation in the original large dimensional space $\mathbb{R}^{\mathcal{N}}$. To speed up the evaluation, we can replace $\mathbf{f}(\mathbf{x})$ with the DEIM interpolation $\mathcal{I}(\mathbf{f}(\mathbf{x})) := U_{\mathbf{f}}(P^T U_{\mathbf{f}})^{-1} P^T \mathbf{f}(\mathbf{x})$, and the error estimation needs to be modified accordingly. To this end, we define a residual involving the DEIM interpolation,

$$\mathbf{r}_{k+1}^{\text{DEIM}} := D\hat{\mathbf{x}}_k + \mathcal{I}(\mathbf{f}(\hat{\mathbf{x}})) - M\hat{\mathbf{x}}_{k+1}.$$

Then we have

$$\begin{aligned} \mathbf{r}_{k+1} &= D\hat{\mathbf{x}}_k + \mathbf{f}(\hat{\mathbf{x}}_k) - M\hat{\mathbf{x}}_{k+1} \\ &= D\hat{\mathbf{x}}_k + \mathcal{I}(\mathbf{f}(\hat{\mathbf{x}}_k)) - M\hat{\mathbf{x}}_{k+1} + \mathbf{f}(\hat{\mathbf{x}}_k) - \mathcal{I}(\mathbf{f}(\hat{\mathbf{x}}_k)) \\ &= \mathbf{r}_{k+1}^{\text{DEIM}} + \underbrace{\mathbf{f}(\hat{\mathbf{x}}_k) - \mathcal{I}(\mathbf{f}(\hat{\mathbf{x}}_k))}_{:=\mathbf{e}_k^{\text{DEIM}}}, \end{aligned}$$

so that

$$\|\mathbf{r}_{k+1}\|_2 \leq \|\mathbf{r}_{k+1}^{\text{DEIM}}\|_2 + \|\mathbf{e}_k^{\text{DEIM}}\|_2. \tag{11}$$

The final error estimation by considering the above DEIM interpolation for $\mathbf{f}(\mathbf{x})$ in $\mathbf{r}_{k+1}$ is given as

$$|y_{k+1} - \hat{y}_{k+1}| \lesssim \Delta_{k+1} := \bar{\Phi}_{k+1} \|\mathbf{r}_{k+1}^{\text{DEIM}}\|_2 + \bar{\Phi}_{k+1} \|\mathbf{e}_k^{\text{DEIM}}\|_2, k = 0, \ldots, K-1. \tag{12}$$

Remark 2 *According to the analysis in,[31] $\mathbf{e}_k^{DEIM}$ can be computed by*

$$\mathbf{e}_k^{DEIM} = \Pi_2 (I - \Pi) \mathbf{f}(\hat{\mathbf{x}}_k),$$

where

$$\Pi := U_{\mathbf{f}}(P^T U_{\mathbf{f}})^{-1} P^T,$$

with $U_{\mathbf{f}} = (\mathbf{u}_1^F, \ldots, \mathbf{u}_l^F) \in \mathbb{R}^{\mathcal{N}\times l}$ *and* $P \in \mathbb{R}^{\mathcal{N}\times l}$ *being the current DEIM basis and index matrix computed from Algorithm 2. For this estimation to be effectively computed, it was assumed that* $l^*(\geq l)$ *DEIM basis vectors of* $\mathbf{f}(\mathbf{x})$ *can exactly represent* $\mathbf{f}(\mathbf{x})$*, i.e.*

$$\mathbf{f}(\mathbf{x}_k) = U_{\mathbf{f}}^*((P^*)^T U_{\mathbf{f}})^{-1}(P^*)^T \mathbf{f}(\mathbf{x}_k),$$

where $U_{\mathbf{f}}^* = (\mathbf{u}_1^F, \ldots, \mathbf{u}_{l^*}^F)$*, and* P^* *is the corresponding index matrix. Then* Π_2 *is defined based on* $U_{\mathbf{f}}^*$ *and* P^**:*

$$\Pi_2 := (I - \Pi)U'((P')^T(I - \Pi)U')^{-1}(P')^T,$$

where $U' = U(:, l+1 : l^*)$*, and* $P' = P^*(:, l+1 : l^*)$ *such that* $U_{\mathbf{f}}^* = [U_{\mathbf{f}}, U']$*, and* $P^* = [P, P']$*. Here and below,* $Q(:, l_1 : l_2)$ *is the MATLAB notation for a part of any matrix* Q*, meaning the matrix consisting of the* l_1*th column till the* l_2*th column of* Q*.*

It can be seen that the DEIM error is updated upon the update of the current DEIM basis $U_{\mathbf{f}}$*. In the next subsection, the DEIM error estimation* $\|\mathbf{e}_k^{DEIM}\|_2$ *will be used to adaptively update the DEIM basis* $U_{\mathbf{f}}$*.*

Adaptive POD-DEIM algorithm

As discussed above, the number r of the reduced basis vectors computed in Algorithm 1 and the number l of the DEIM basis vectors computed in Algorithm 2 are usually selected according to an empirically given SVD tolerance ϵ_{svd}, which has no direct relation to the error of the ROM. The fact is that the reduced basis V and the DEIM basis $U_{\mathbf{f}}$ are implicitly related and different combinations of the two bases result in different ROMs with different size r and different accuracy.

We will show that the error estimation Δ_{k+1} (12) can aid us to develop a heuristic adaptive POD-DEIM algorithm which iteratively updates the dimension of the reduced basis r, and the dimension of the DEIM basis l, and finally constructs more compact ROMs meeting different

accuracy requirements automatically.

From the structure of the error estimation in (12), we see that it is composed of two parts, i.e.

$$\Delta_{k+1} = \Delta_{k+1}^{\text{POD}} + \Delta_{k}^{\text{DEIM}},$$

where $\Delta_{k+1}^{\text{POD}} = \bar{\Phi}_{k+1}\|\mathbf{r}_{k+1}^{\text{DEIM}}\|_2$ and $\Delta_{k}^{\text{DEIM}} = \bar{\Phi}_{k+1}\|\mathbf{e}_{k}^{\text{DEIM}}\|_2$. The average output error over all the time instances can be evaluated via the average error estimation, i.e.

$$\underbrace{\frac{1}{K}\sum_{k=0}^{K-1}|y_{k+1} - \hat{y}_{k+1}|}_{\bar{\varepsilon}_y} \lesssim \underbrace{\frac{1}{K}\sum_{k=0}^{K-1}\Delta_{k+1}^{\text{POD}}}_{\bar{\Delta}^{\text{POD}}} + \underbrace{\frac{1}{K}\sum_{k=0}^{K-1}\Delta_{k}^{\text{DEIM}}}_{\bar{\Delta}^{\text{DEIM}}}. \tag{13}$$

Here, $\bar{\varepsilon}_y$ represents the average output error. The variable $\bar{\Delta}^{\text{POD}}$ estimates the average error of the reduced basis approximation and determines the dimension r, whereas the variable $\bar{\Delta}^{\text{DEIM}}$ estimates the average error of the DEIM interpolation, and determines the dimension l. In general, $\bar{\Delta}^{\text{POD}}$ or $\bar{\Delta}^{\text{DEIM}}$ decays while the dimension of the respective basis increases.

Given a user defined error tolerance of the ROM, ϵ, we define the two error ratios,

$$p_r := \frac{\bar{\Delta}^{\text{POD}}}{\epsilon},$$

and

$$p_l := \frac{\bar{\Delta}^{\text{DEIM}}}{\epsilon}.$$

The magnitude of each basis increment then depends on the logarithm of each ratio. The bigger the ratio, the more basis vectors should be added accordingly, in order to increase the accuracy of the corresponding basis approximation. Algorithm 3 describes the process of adaptively adjusting the basis dimensions r and l, where the output error of the ROM is estimated by the error estimator

$$\bar{\Delta} := \bar{\Delta}^{\text{POD}} + \bar{\Delta}^{\text{DEIM}}. \tag{14}$$

Remark 3 *The input of Algorithm 3 actually needs to be obtained from implementing Algo-*

Algorithm 3 Adaptive POD-DEIM algorithm

Input: offline computation: Given a sufficiently small SVD tolerance, e.g., $\epsilon_{svd} = 10^{-10}$, implement Algorithm 1 to get the reduced basis $V^* \in \mathbb{R}^{n\times r^*}$, and implement Algorithm 2 to get the DEIM basis $(\mathbf{u}_1^F, \ldots, \mathbf{u}_{l^*}^F) \in \mathbb{R}^{n\times l^*}$ and the index matrix P^*. Assume that the corresponding DEIM interpolation is *exact*, so that the DEIM error introduced in Remark 2 can be used to compute the error estimation $\bar{\Delta}$.
Set initial values of r and l: $r = r_0 < r^*$ and $l = l_0 < l^*$, and a user defined tolerance ϵ for the error of the final ROM.

Output: The final ROM constructed based on the final updated r, l.

1: Form the reduced basis by taking r columns from V^*: $V = V^*(:, 1 : r)$, and form the DEIM basis by taking the first l basis vectors from $(\mathbf{u}_1^F, \ldots, \mathbf{u}_{l^*}^F)$, i.e. $U_{\mathbf{f}} = (\mathbf{u}_1^F, \ldots, \mathbf{u}_l^F)$. Form the corresponding index matrix P: $P = P^*(:, 1 : l)$. Formulate the ROM in (7) using V, $U_{\mathbf{f}}$ and P.
2: Solve the ROM following (9) and compute the error estimation $\bar{\Delta} = \bar{\Delta}^{\text{POD}} + \bar{\Delta}^{\text{DEIM}}$.
3: **while** $\bar{\Delta} \geq \epsilon$ **do**
4: Calculate $p_r = \frac{\bar{\Delta}^{\text{POD}}}{\epsilon}$,
5: calculate $p_l = \frac{\bar{\Delta}^{\text{DEIM}}}{\epsilon}$.
6: $\delta_r = 1 + \lfloor \log_{10}(p_r) \rfloor$,
7: $\delta_l = 1 + \lfloor \log_{10}(p_l) \rfloor$.
8: $r = r + \delta_r$, if $\delta_r > 0$,
9: $l = l + \delta_l$, if $\delta_l > 0$.
10: Update $V = V^*(:, 1 : r)$, $U_{\mathbf{f}} = (\mathbf{u}_1^F, \ldots, \mathbf{u}_l^F)$, and $P = P^*(:, 1 : l)$. Formulate the ROM in (7).
11: Solve the ROM following (9) and compute the error estimation $\bar{\Delta} = \bar{\Delta}^{\text{POD}} + \bar{\Delta}^{\text{DEIM}}$.
12: **end while**

rithm 1 and Algorithm 2 once, respectively. The algorithm can be divided into offline and online stages w.r.t. the adaptivity process, where the adaptive process (Step 3-14) is the online stage, and the process of input data collection is the offline stage which is exactly the process of the standard POD-DEIM method. Therefore, the online adaptive stage does not involve any additional snapshot computations, and only introduces small amount of extra computation, as compared to the standard POD-DEIM method. Algorithm 3 could be extended to deal with parametric systems. In these cases, the POD and DEIM basis should be constructed based on the snapshots from the sampled parameters in a training set.[15,16] The error estimator used to validate the ROM, should be an estimator for parametric systems, e.g. the one proposed in.[15]

Remark 4 *In Algorithm 3, we use $\bar{\Delta}$ to estimate the average error of the ROM over all the time instances. If the average error is below the error tolerance, the algorithm stops updating l and r, as well as the ROM. As shown in Steps 6-7, after computing the logarithm of the error*

ratios p_r and p_l, we take the integer part of them to get natural numbers which are then used as the values of basis increase δ_r, δ_l for each basis increment. δ_r and δ_l are automatically set to 1, in case we get zeros after using the floor function, in order to avoid non-increment. Note that the algorithm can only get a final ROM with basis dimensions equal or larger than the initial basis dimensions r_0 and l_0. When the initial dimensions r_0, l_0 are too big for a given tolerance, the proposed algorithm cannot adjust them to derive a smaller ROM. Since the goal of the algorithm is to derive a ROM with small enough dimension, the initial dimensions r_0, l_0 should be set to rather small values.

Remark 5 *Algorithm 3 is an abstract version of the adaptive scheme including the key idea of adaptivity. The robustness of the algorithm may depend on the problem under consideration. Furthermore, to make the algorithm practically efficient, specific issues might be dealt with for the particular model considered. For example, we have refined Algorithm 3 to obtain Algorithm 4 in the next section, which has taken the stability issue into consideration.*

Adaptive POD-DEIM applied to a population balance model

As an application example for the proposed adaptive algorithm, we use a model of a continuous crystallization process for the separation of enantiomers. We first introduce the model and then present Algorithm 4 which is a more specialized version of Algorithm 3 for the population balance model, by adding more details concerning specific issues, such as stability of the ROM.

Population balance model of a crystallization process for enantiomer separation

Enantiomers are a class of molecules that appear in two variants, which have the same sum formula and consist of the same molecular groups, but with an opposite orientation in space, like mirror images of each other or like right and left hand. Due to identical physical and chemical properties of the two enantiomers, standard separation techniques like distillation are not applicable, but more advanced approaches like adsorption, membrane separation, or crystallization

are needed.[34,35] A continuous crystallization process for enantiomer separation was recently investigated experimentally in.[36] In this contribution, a simplified scheme of the process in[36] is considered, a model for which was presented in.[8]

The key element of the crystallization process shown in Figure 1 is a fluidized bed crystallizer with a conical shape and a varying cross-sectional area $A(x)$. The liquid feed contains a racemic mixture of both enantiomers. It enters the crystallizer at the bottom. The crystallizer is operated in the metastable region, i.e. seeding crystals may grow, but nucleation of new crystals is suppressed. When seeding the crystallizer initially with crystals of the desired enantiomer, only these crystals grow selectively and a separation of the enantiomers is achieved. Small crystals tend to move upwards with the fluid flow. Fines escape through the top of the crystallizer. This removes nuclei of the undesired enantiomer from the crystallizer and increases the stability of the process. Large crystals sink to the bottom of the crystallizer. They are sent to a fragmentation device like a mill or an ultrasonic attenuator and are recycled as small seeding crystals into the process.

The crystal growth and transportation process are modeled using a population balance approach, i.e. the properties of a particle population consisting of an infinite number of individuals are simulated. For details of the model, the reader is referred to.[8] Only the main model equations are summarized in this contribution. The particle phase is described by a number size density $n(x, L, t)$ denoting the number of particles with size L per volume at time t at a point x in space. A mass balance of the particle phase leads to the following population balance

equation:

$$
\begin{aligned}
A(x)\left.\frac{\partial n}{\partial t}\right|_{x,L,t} &= -\frac{\partial}{\partial x}\left(A(x)\; v_P(x,L,t)\; n(x,L,t)\right) \\
&\quad +D\frac{\partial}{\partial x}\left(A(x)\left.\frac{\partial n}{\partial x}\right|_{x,L,t}\right) \\
&\quad -A(x)\; G\;\left.\frac{\partial n}{\partial L}\right|_{x,L,t} \\
&\quad +\dot{V}_{fr}\left(n_{fr}(L)\frac{\int_0^\infty n(x,l,t)\; l^3\; dl}{\int_0^\infty n_{fr}(l)\; l^3\; dl} - n(x,L,t)\right)\hat{\delta}(x-x_{fr})
\end{aligned}
\tag{15}
$$

$$(0 < x < H, \quad 0 < L, \quad t > 0)$$

The first term on the right-hand side of (15) describes the advective transport of particles with velocity $v_P(x, L, t)$, which depends nonlinearly on the number size density $n(x, L, t)$(see[8] for details)

The second term on the right-hand side of the population balance equation (15) stands for particle transport by dispersion with dispersion coefficient D. The third term is due to crystal growth. The fourth term describes the effect of the fragmenter on the crystal population. $\dot{V}_{fr}$ is the volume flow to and from the fragmenter.

The system output y is considered as the total volume fraction of particles as a physically meaningful quantity, i.e.

$$
y(t) = \frac{\int_0^H \int_0^\infty \frac{\pi}{6} L^3\, n(x,L,t)\, dL\; A(x)\, dx}{\int_0^H A(x)\, dx} = \frac{\int_0^H q_3(x,L,t) A(x)\, dx}{\int_0^H A(x)\, dx}. \tag{16}
$$

Model order reduction for the population balance model

Spatial discretization of the population balance model (15) together with the boundary conditions, and the output equation (16) lead to a large-scale system

$$\begin{aligned} \frac{d\mathbf{n}(\mathbf{t})}{dt} &= A\mathbf{n}(t) + \mathbf{f}(\mathbf{n}(\mathbf{t})), \\ y(t) &= C^T\mathbf{n}(t), \end{aligned} \tag{17}$$

where $\mathbf{n}(t) \in \mathbb{R}^{\mathcal{N}}$ is the number size density $n(x, L, t)$ discretized in the one-dimensional space x, and in the particle size L (see[8] for details). In the following, the system (17) is also called the crystallizer model for simplicity, and is referred to as the full order model (FOM) as compared to the ROM. $A \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ is the system matrix corresponding to the discretized linear part in (15), and $\mathbf{f}(\mathbf{n}(\mathbf{t})) \in \mathbb{R}^{\mathcal{N}}$ is the nonlinear term. For discretization of the space coordinate x, an equidistant grid with 230 grid points is used; the discretization of the property coordinate L is done on a logarithmically distributed grid with 80 grid points. This results in a total number of $\mathcal{N} = 230 \times 80 = 18400$ degrees of freedom.

To obtain a steady behavior of the system (17), numerical simulation needs to be done in the time interval $[0, 10000]$s (seconds), which is usually time consuming. In,[8] the standard POD method was used to get the ROM of the original large system (17),

$$\begin{aligned} \frac{d\mathbf{n}(\mathbf{t})}{dt} &= V^TAV\mathbf{n}(t) + V^T\mathbf{f}(V\mathbf{z}(\mathbf{t})), \\ \hat{y}(t) &= C^TV\mathbf{z}(t). \end{aligned} \tag{18}$$

In addition, an empirical interpolation method (EIM)[13] was employed to reduce the complexity of the nonlinear term $V^T\mathbf{f}(V\mathbf{z}(\mathbf{t}))$, and as a result the final ROM is

$$\begin{aligned} \frac{d\mathbf{n}(\mathbf{t})}{dt} &= V^TAV\mathbf{n}(t) + V^T\Theta\beta(t), \\ \hat{y}(t) &= C^TV\mathbf{z}(t), \end{aligned} \tag{19}$$

where $\beta(t)$ is the vector of interpolation coefficients, $\Theta = (\xi_1, \ldots, \xi_l)$ represents the interpolation basis, and is slightly different from the DEIM basis. In particular, the EIM basis is iter-

atively constructed simultaneously with the interpolation coefficients from the nonlinear snapshots; whereas the DEIM basis vectors are just the POD basis vectors of the nonlinear snapshot matrix F. The corresponding interpolation coefficients are then constructed iteratively from the DEIM basis. The error estimation proposed in[15] was applied in[8] to verify the reliability of the reduced order model.

In this work, we apply the error estimation to the proposed adaptive POD-DEIM algorithm, and aim at automatically constructing a reliable ROM of (17) by adaptively adjusting the reduced basis dimension and the DEIM basis dimension.

Implementing adaptive POD-DEIM

When applying Algorithm 3 to the crystallizer model (17), we need to add more details to the algorithm by taking, e.g., stability of the ROM into consideration. For the crystallizer model, we have observed that for certain combinations of (r, l), the ROM becomes unstable. In such situations, (r, l) needs to be adjusted accordingly to avoid instability. Furthermore, we find that the dimension of the DEIM basis should be sufficiently bigger than the reduced basis dimension to guarantee stability [1] and to avoid producing ROMs which are stiff, or hard to solve. Based on the above observations, we further refine Algorithm 3, and derive Algorithm 4, which is especially suitable for the crystallizer model. In Algorithm 4, the values of $\tilde{\delta}_r$ (Step 11) and $\tilde{\delta}_l$ (Step 20) are heuristically determined. For this model, we find that $\tilde{\delta}_r = 4$ is big enough to guarantee stability of the ROM, and $\tilde{\delta}_l = 2$ is sufficient to avoid constructing stiff ROMs.

Applying Algorithm 4 requires first implementing Algorithm 1 and Algorithm 2. For implementation of Algorithm 1, we need to take a proper number of snapshots at proper time instances $t_i, i = 1, \ldots, m$. Figure 2 plots the output $y(t)$ of the crystallizer model (17) in the time interval $[0, 10000]s$. The times T_1, T_2, ... denote the time points where the output $y(t)$ has a local minimum. The output approaches a stable periodic orbit, i.e. the waveforms in the periods $[T_i, T_{i+1}], i = 1, \ldots, 4$ are quite similar. This motivates us to:

[1] For example, when elements of the solution vector become infinite (==Inf in MATLAB), we find that the ROM is unstable. For this example, we use the function ode45 in MATLAB to solve the ROMs at each iteration. The temporal time steps are automatically determined by the MATLAB function.

Algorithm 4 Adaptive POD-DEIM algorithm for the Crystallizer model

Input: Same as for Algorithm 3.

Output: The final ROM constructed based on the final updated r, l.

Form the reduced basis by taking r columns from V^*: $V = V^*(:, 1:r)$, and form the DEIM basis by taking the first l vectors from $(\mathbf{u}_1^F, \ldots, \mathbf{u}_{l^*}^F)$, i.e. $U_{\mathbf{f}} = (\mathbf{u}_1^F, \ldots, \mathbf{u}_l^F)$. Form the index matrix P: $P = P^*(:, 1:l)$. Formulate the ROM in (7) using V, $U_{\mathbf{f}}$ and P.
Solve the ROM following (9) and compute the error estimation $\bar{\Delta} := \bar{\Delta}^{\text{POD}} + \bar{\Delta}^{\text{DEIM}}$.
while $\bar{\Delta} \geq \epsilon$ **do**
  Calculate $p_r = \frac{\bar{\Delta}^{\text{POD}}}{\epsilon}$,
  calculate $p_l = \frac{\bar{\Delta}^{\text{DEIM}}}{\epsilon}$.
  $\delta_r = 1 + \lfloor \log_{10}(p_r) \rfloor$,
  $\delta_l = 1 + \lfloor \log_{10}(p_l) \rfloor$.
  $r = r + \delta_r$, if $\delta_r > 0$,
  $l = l + \delta_l$, if $\delta_l > 0$.
  if ROM is unstable **then**
    if $l - r > \tilde{\delta}_r$ **then**
      $r = r + 1$,
    else
      $r = r + 1$,
      $l = r + \tilde{\delta}_r + 1$.
    end if
  end if
  if $l \leq r$ **then**
    $l = r + \delta_l$, if $\delta_l > 1$,
    $l = r + \tilde{\delta}_l$, if $\delta_l \leq 1$.
  end if
  Update $V = V^*(:, 1:r)$, $U_{\mathbf{f}} = (\mathbf{u}_1^F, \ldots, \mathbf{u}_l^F)$, and $P = P^*(:, 1:l)$. Formulate the ROM in (7).
  Solve the ROM following (9) and compute the error estimation $\bar{\Delta} := \bar{\Delta}^{\text{POD}} + \bar{\Delta}^{\text{DEIM}}$.
end while

- use only one of them plus the transient stage in the time interval $[0, T_1]$ to generate the snapshots for transient-periodic simulation,
- use only one of the periods, namely $[T_1, T_2]$, to generate snapshots for only stable periodic simulation.

In this way, the offline computational time is significantly reduced. We will show in the simulation results that the ROM constructed by using only the snapshots in the time interval $[0, T_2]$ reproduces the waveform of the original output in the whole time interval $[0, T_5]$, and the ROM constructed by using only the snapshot in $[T_1, T_2]$ is able to catch the periodic behavior in the time interval $[T_1, T_5]$. The snapshots are taken every $\Delta t = 10$ seconds from $[0, T_2]$ or $[T_1, T_2]$,

e.g., the snapshot matrix $X = (\mathbf{n}(t_1), \ldots, \mathbf{n}(t_m))$, $t_1 = 0$, $\Delta t = t_{i+1} - t_i$, $t_m = T_2$.

Simulation results

In this section we present the results of the adaptive scheme presented in Algorithm 4. We show the results of the transient-periodic simulation in the first subsection. If one is interested in only the periodic behavior of the system, it is sufficient that the ROM is able to reproduce the output in the periodic stage, regardless of the transient behavior. The results of the ROM for the stable periodic orbit are shown in the second subsection. All the simulation results are done in MATLAB® version R2013b, on a Laptop with Intel(R) core(TM) i7-5500U CPU @2.40GHz, 8.00GB RAM.

Simulation of the transient behavior

Figure 3(a) displays singular values of the snapshot matrix X taken from the time interval $[0, T_2]$. Given a prescribed tolerance for the SVD, e.g., $\epsilon_{svd} = 10^{-10}$, it can be easily checked that $r^* = 61$ makes

$$\frac{\sum_{i=r^*+1}^{d_X} \sigma_i}{\sum_{i=1}^{d_X} \sigma_i} < \epsilon_{svd} = 10^{-10}.$$

Therefore, we take $r^* = 61$ reduced basis vectors for the input data of Algorithm 4. The singular values of the snapshot matrix $F = (\mathbf{f}(\mathbf{n}(t_1)), \ldots, \mathbf{f}(\mathbf{n}(t_m)))$ are in Figure 3(b). Similarly, $l^* = 66$ results in

$$\frac{\sum_{i=l^*+1}^{d} \sigma_i^F}{\sum_{i=1}^{d} \sigma_i^F} < \epsilon_{svd} = 10^{-10},$$

where σ_i^F are the singular values of F. Therefore, we take $l^* = 66$ DEIM basis vectors, and the corresponding index matrix P^* computed by Algorithm 2, to constitute the input data of Algorithm 4.

Consequently, without adaptively choosing the bases dimensions, the ROM $\Sigma_{r^*l^*}$ derived

by the standard POD-DEIM method would be of order $r^* = 61$ with DEIM basis dimension $l^* = 66$. The output error of the ROM is $\bar{\Delta} = 3.97 \cdot 10^{-4}$ estimated by the error estimator $\bar{\Delta}$ defined in (14). In Table 1, we list different ROMs constructed by Algorithm 4, with the initial value $r_0 = 3, l_0 = 6$, according to different tolerances. In Table 2, we change the initial bases dimensions to $r_0 = 7, l_0 = 7$, and derive another group of ROMs. In Tables 1-3, the "Runtime" is the computational time of adaptively constructing the ROM, including the ROM simulation time. In the last column, the ROM simulation time appears as the percentage out of the corresponding runtime. Tables 1-2 show that the adaptive process is robust, and independent of the initial choices r_0, l_0. The output error estimation $\bar{\Delta}$ always tightly bounds the true error $\bar{\varepsilon}_y$ as defined in (13), showing its efficiency and reliability.

We see that the reduced basis and the DEIM basis of the ROMs constructed by Algorithm 3 are all taken from the bases of the ROM with bases dimensions r^* and l^*, defined here as $\Sigma_{r^*l^*}$. Therefore the accuracy of the ROMs may not be higher than the ROM $\Sigma_{r^*l^*}$. Since the output error of $\Sigma_{r^*l^*}$ is $\mathcal{O}(10^{-4})$, the smallest error tolerance we take in the two tables is $1 \cdot 10^{-3}$. Furthermore, the adaptive algorithm computes ROMs which are more compact than $\Sigma_{r^*l^*}$, but with similar accuracy.

Simulation of the periodic orbit

If only the periodic behavior of the output is of interest, then the starting transient stage in the time interval $[0, T_1]$ (Figure 2) needs not be recovered by the ROM, and the corresponding snapshot matrices $X_{T_{12}}$ and $F_{T_{12}}$ are composed of only the snapshots in $[T_1, T_2]$. Excluding the snapshots in $[0, T_1]$, we have observed from Figure 4 that the singular value decay of each matrix becomes much faster. Using the same tolerance for the SVD decomposition, $\epsilon_{svd} = 10^{-10}$, we obtain $r^* = 38$ reduced basis vectors from $X_{T_{12}}$, and $l^* = 41$ DEIM basis vectors from $F_{T_{12}}$. The ROM constructed by the standard POD-DEIM method with r^*, l^* has output error of $\bar{\Delta} = 5.34 \cdot 10^{-8}$, estimated by the error estimator $\bar{\Delta}$. This indicates that it is possible for Algorithm 4 to adaptively construct a ROM meeting an error tolerance as small as $\epsilon = 1 \cdot 10^{-7}$. The ROMs constructed by Algorithm 4 according to different accuracy requirements ϵ are listed in Table 3.

This again demonstrates the efficient performance of the adaptive algorithm under the guidance of a tight error estimator. In Figure 5, we plot the dimension increments of the reduced basis and the DEIM basis at each iteration of the adaptive algorithm for different tolerances ϵ. We see that the increments of both bases are adaptive, and especially, the increment is bigger in the beginning when the error is larger. It is indicated in Figure 5 that for $\epsilon = 10^{-6}$, 8 iterations are run to get the final ROM, whereas for $\epsilon = 10^{-7}$, there are only 7 iterations. This explains the corresponding runtime in Table 3, where $\epsilon = 10^{-6}$ costs more time than $\epsilon = 10^{-7}$.

Computational time comparison

Table 4 lists the ROMs constructed by the standard POD-DEIM method for the transient case and the periodic case, respectively. The "Sim. time" in the table is the time spent on solving the ROM. Comparing the simulation time in Table 4 with the runtime in Tables 1-3, we see that the adaptive algorithm constructs the ROMs without introducing much extra computational time. The ROM simulation time listed in Tables 1-3 is much less than the ROM simulation time in Table 4. In many cases, the total runtime of the adaptive algorithm is even less than the ROM simulation time of the standard method in Table 4. In the last column of Table 4, we show the ratio between the runtime ($\text{Sim}_{r,l}$) of the adaptive algorithm ($\epsilon = 1 \cdot 10^{-3}$) and the runtime ($\text{Sim}_{r^*,l^*}$) of the standard POD-DEIM method. The ratio 0.87 corresponds to the adaptive result ($r = 16, l = 19$) in Table 2 and the ratio 0.55 corresponds to the adaptive result ($r = 7, l = 10$) in Table 3. Here, $\epsilon = 1 \cdot 10^{-3}$ is the acceptable accuracy of the ROM for this model.

ROM validation over the whole time interval

The ROMs constructed by the adaptive algorithm are based on the snapshots in the time intervals $[0, T_2]$ and $[T_1, T_2]$, respectively. In Figures 6-7, we plot the outputs (in percentage) of the ROMs over the extended intervals $[0, T_5]$ and $[T_1, T_5]$, and compare them with the corresponding outputs of the FOM (17). The actual output error of the ROM is also presented. The ROM in Figure 6 is of dimension $r = 16$ presented in Table 1. From Figure 6(b), we see the error of the ROM over the whole time interval is still below the error tolerance $\epsilon = 1 \cdot 10^{-3}$ (Table 1),

showing that the ROM has not introduced extra error due to the extended simulation interval. The ROM in Figure 7 is of dimension $r = 14$ as listed in Table 3. Although the error of the ROM in the time interval $[T_1, T_2]$ is below the error tolerance $\epsilon = 1 \cdot 10^{-5}$, the overall error is up to $1.6 \cdot 10^{-3}$, see Figure 7(b). This is nevertheless not surprising, since the ROM is constructed by using only the snapshots in the second subinterval $[T_1, T_2]$ (see Figure 1). The overall error is still acceptable for practical applications. In Table 5, we list the simulation time of solving the FOM and the corresponding two ROMs taken from Table 1 and Table 3, respectively, over the extended time intervals $[0, T_5]$ and $[T_1, T_5]$, where $T_5 = 10^4$seconds. When solving the FOM, around 1100 time integration steps spent in every 10 seconds to get the final accurate solution. It is obvious that the ROMs not only largely reduce the dimension of the FOMs, but also significantly speed up the simulation.

Conclusions

We have proposed an algorithm to adaptively adjust the reduced basis dimension and the DEIM basis dimension, so that a more compact reduced order model can be obtained as compared to the standard POD-DEIM method. The idea of adaptivity is heuristic but efficient and simple to implement. The efficiency and reliability of the adaptive algorithm is secured by an error estimation specified for the POD-DEIM reduced-order model. Application of the algorithm to a population balance system arising from modelling of a crystallization process is explored. It is observed that the DEIM basis needs to be larger than the reduced basis to guarantee stability of the ROM. The adaptive scheme shows robustness in reducing both the complexity and the simulation time of the model. Extension of the proposed algorithm to parametric systems will be considered in the future.

Acknowledgments

The second author acknowledges the support by DFG in the framework of SPP 1679.

Literature Cited

1. Sirovich L. Turbulence and the dynamics of coherent structures. I - coherent structures. II - symmetries and transformations. III - dynamics and scaling. *Quarterly of Applied Mathematics*. 1987;45(3):561–571, 573–582, 583–590.

2. Kunisch K, Galerkin SV. Galerkin proper orthogonal decomposition methods for a general equation in fluid dynamics. *SIAM Journal on Numerical Analysis*. 2002;40:492–515.

3. Astrid P, Weiland S, Willcox K, Backx T. Missing point estimation in models described by proper orthogonal decomposition. *IEEE Transactions on Automatic Control*. 2008; 53(10):2237–2251.

4. Willcox K. Unsteady flow sensing and estimation via the gappy proper orthogonal decomposition. *Computers & Fluids*. 2006;35(2):208–226.

5. Benner P, Gugercin S, Willcox K. A survey of model reduction methods for parametric systems. *SIAM Review*. 2015;57(4):483–531.

6. Krasnyk M, Mangold M, Ganesan S, Tobiska L. Numerical reduction of a crystallizer model with internal and external coordinates by proper orthogonal decomposition. *Chemical Engineering Science*. 2012;70:77–86.

7. Mangold M. Model reduction of a batch drum granulator by proper orthogonal decomposition. In: *Proceedings of 8th IFAC Symposium on Advanced Control of Chemical Processes*. 2012; pp. 856–861.

8. Mangold M, Feng L, Khlopov D, Palis S, Benner P, Binev D, Morgenstern AS. Nonlinear model reduction of a continuous fluidized bed crystallizer. *Journal of Computational and Applied Mathematics*. 2015;289:253–266.

9. Baker J, Christofides PD. Finite-Dimensional Approximation and Control of Nonlinear Parabolic PDE Systems. *Int J Contr*. 2000;73:439–456.

10. Armaou A, Christofides PD. Dynamic Optimization of Dissipative PDE Systems Using Nonlinear Order Reduction. *Chem Eng Sci*. 2002;57:5083–5114.

11. Varshney A, Pitchaiah S, Armaou A. Feedback control of dissipative PDE systems using adaptive model reduction. *AIChE Journal*. 2009;55:906–918.

12. Marchisio DL, Fox RO. Solution of population balance equations using the direct quadrature method of moments. *Journal of Aerosol Science*. 2005;36(1):43–73.

13. Barrault M, Maday Y, Nguyen NC, Patera AT. An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique Academie des Sciences Paris*. 2004;339(9):667–672.

14. Chaturantabut S, Sorensen DC. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*. 2010;32(5):2737–2764.

15. Zhang Y, Feng L, Li S, Benner P. An efficient output error bound for model order reduction of parametrized evolution equations. *SIAM Journal on Scientific Computing*. 2015; 37(6):B910–B936.

16. Drohmann M, Haasdonk B, Ohlberger M. Reduced basis approximation for nonlinear parametrized evolution equations based on empirical operator interpolation. *SIAM Journal on Scientific Computing*. 2012;34:937–969.

17. Peherstorfer B, Willcox K. Online adaptive model reduction for nonlinear systems via low-rank updates. *SIAM Journal on Scientific Computing*. 2015;37(4):A2123–A2150.

18. Feng L, Korvink JG, Benner P. A fully adaptive scheme for model order reduction based on moment-matching. *IEEE Transactions on Components, Packaging and Manufacturing Technology*. 2015;5(12):1872–1884.

19. Degroote J, Vierendeels J, Willcox K. Interpolation among reduced-order matrices to obtain parameterized models for design, optimization and probabilistic analysis. *International Journal for Numerical Methods in Fluids*. 2010;63:207–230.

20. Panzer H, Mohring J, Eid R, Lohmann B. Parametric model order reduction by matrix interpolation. *Automatisierungstechnik*. 2010;58:475–484.

21. Moore BC. Principal component analysis in linear systems: controllability, observability, and model reduction. *IEEE Transactions on Automatic Control*. 1981;26:17–32.

22. Baur U, Benner P, Feng L. Model Order Reduction for Linear and Nonlinear Systems: A System-Theoretic Perspective. *Archives of Computational Methods in Engineering*. 2014; 21(4):331–358.

23. Benner P. System-theoretic methods for model reduction of large-scale systems: simulation, control, and inverse problems. In: *MATHMOD 2009, 6th Vienna International Conference on Mathematical Modelling, ARGESIM Report*, vol. 35. 2009; pp. 126–145.

24. Feng L, Antoulas AC, Benner P. Some a posteriori error bounds for model order reduction of parametrized linear systems. *Max Planck Institute Preprint, MPIMD 15-17*. 2015; Available from `http://www.mpi-magdeburg.mpg.de/preprints/`.

25. Grepl MA, Patera AT. A posteriori error bounds for reduced-basis approximations of parametrized parabolic partial differential equations. *ESAIM: M2AN Mathematial Modelling and Numerical Analysis*. 2005;39(1):157–181.

26. Veroy K, Prud'homme C, Rovas D, Patera AT. A posteriori error bounds for reduced-basis approximation of parametrized noncoercive and nonlinear elliptic partial differential equations. In: *AIAA conference papers: 16th AIAA Computational Fluid Dynamics Conference*. 2003; pp. 2003–3847.

27. Veroy K, Patera AT. Certified real-time solution of the parametrized steady incompressible Navier-Stokes equations: rigorous reduced-basis a posteriori error bounds. *International Journal for Numerical Methods in Fluids*. 2005;47(8):773–788.

28. Rozza G, Huynh DBP, Patera AT. Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations. *Archives of Computational Methods in Engineering*. 2008;15:229–275.

29. Rozza G, Huynh DBP, Manzoni A. Reduced basis approximation and a posteriori error estimation for Stokes flows in parametrized geometries: roles of the inf-sup stability constants. *Numerische Mathematik*. 2013;125(1):115–152.

30. Grepl MA, Maday Y, Nguyen NC, Patera AT. Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations. *ESAIM: M2AN Mathematial Modelling and Numerical Analysis*. 2007;41(3):575–605.

31. Wirtz D, Sorensen DC, Haasdonk B. A posteriori error estimation for DEIM reduced nonlinear dynamical systems. *SIAM Journal on Scientific Computing*. 2014;36(2):A311–A338.

32. Zhang Y, Feng L, Li S, Benner P. Accelerating PDE constrained optimization by the reduced basis method: application to batch chromatography. *International Journal for Numerical Methods in Engineering*. 2015;104:983–1007.

33. Volkwein S. Model reduction using proper orthogonal decomposition. *Tech. rep.* 2010.

34. Alvarez A, Myerson A. Continuous plug flow crystallization of pharmaceutical compounds. *Crystal Growth & Design*. 2010;10:2219–2228.

35. Lorenz H, Seidel-Morgenstern A. Processes to separate enantiomers. *Angewandte Chemie International Edition*. 2014;53(5):1218–1250.

36. Binev D, Seidel-Morgenstern A, Lorenz H. Continuous separation of isomers in fluidized bed crystallizers. *Crystal Growth & Design*. 2016;16(3):1409–1419.

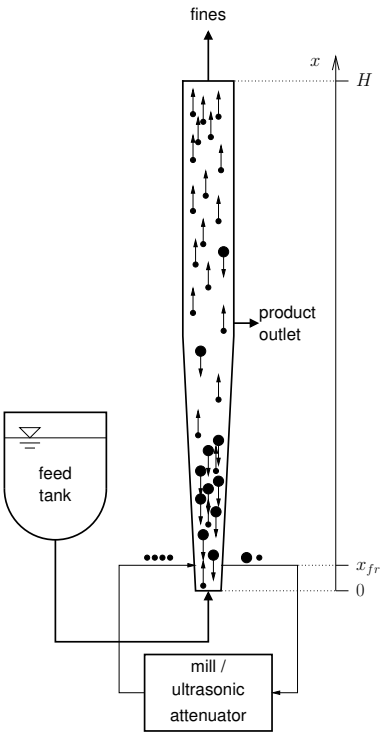


Figure 1: Process scheme for the continuous separation of enantiomers by preferential crystallization.

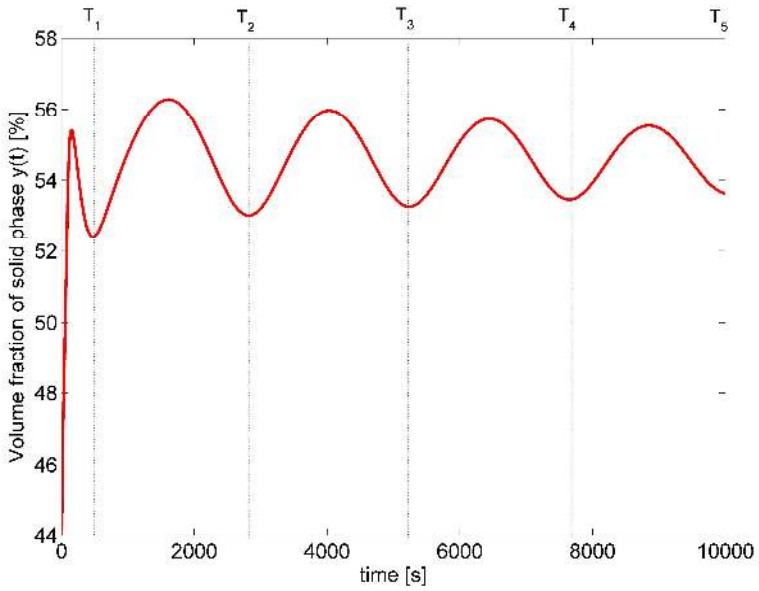
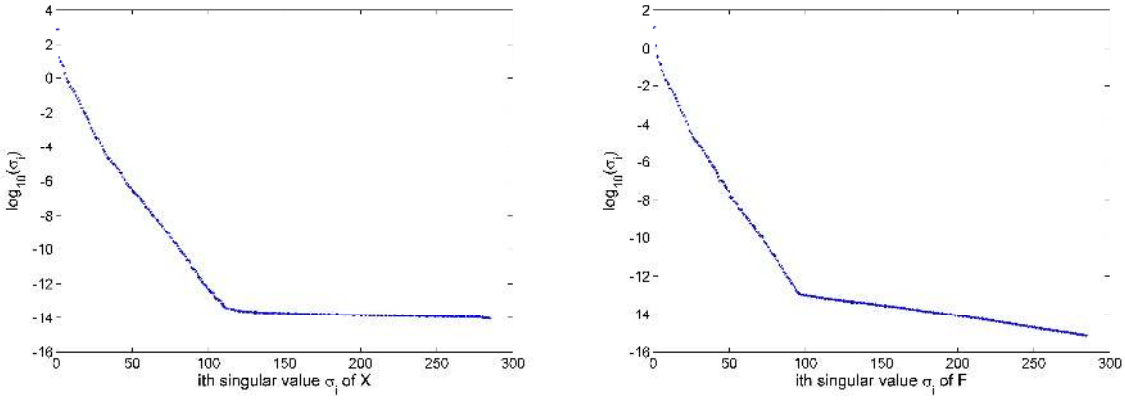


Figure 2: Output in percentage: total volume fraction of the solid phase $y(t)$ in (16).



(a) (b)

Figure 3: Singular value decay of X and F, respectively.

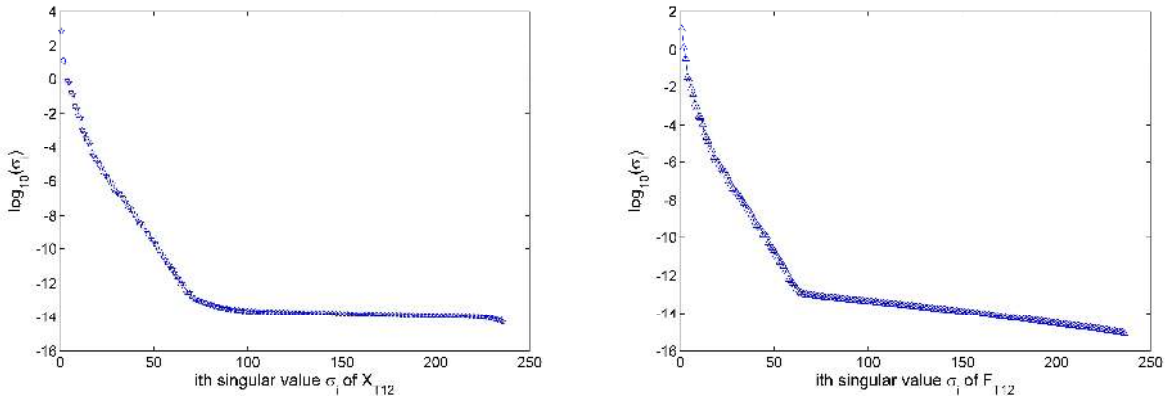


Figure 4: Singular value decay of $X_{T_{12}}$ and $F_{T_{12}}$, respectively.

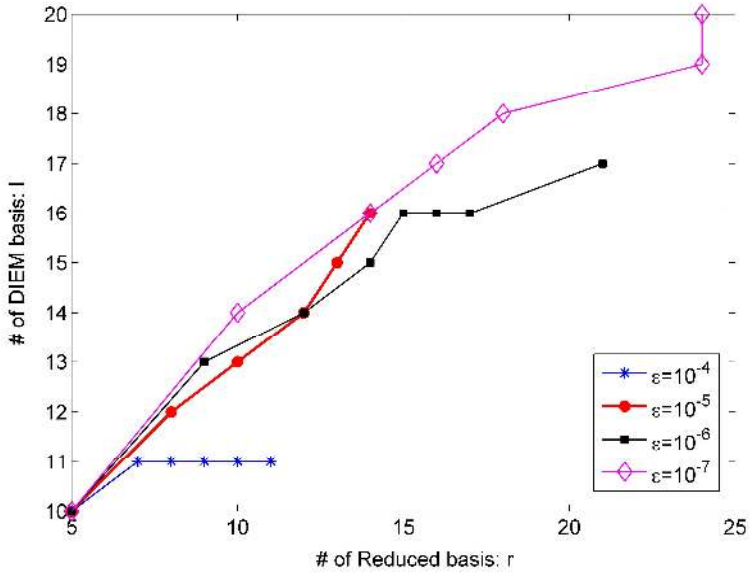
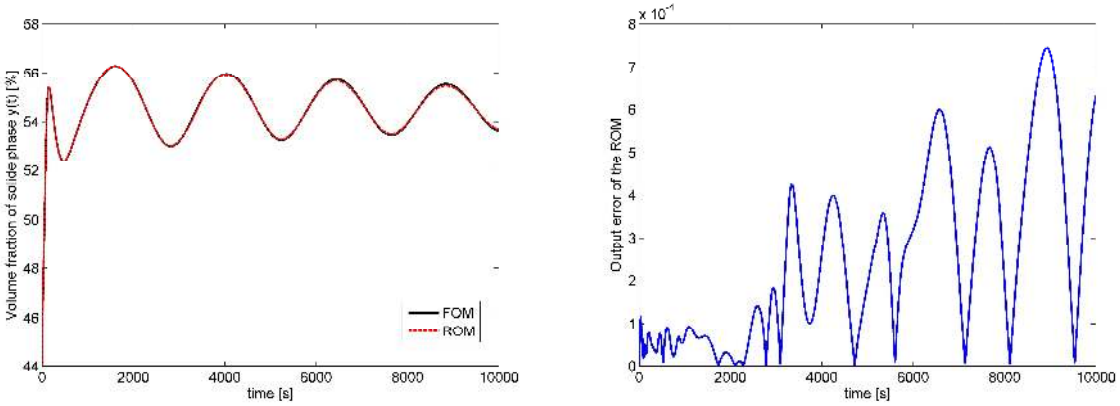
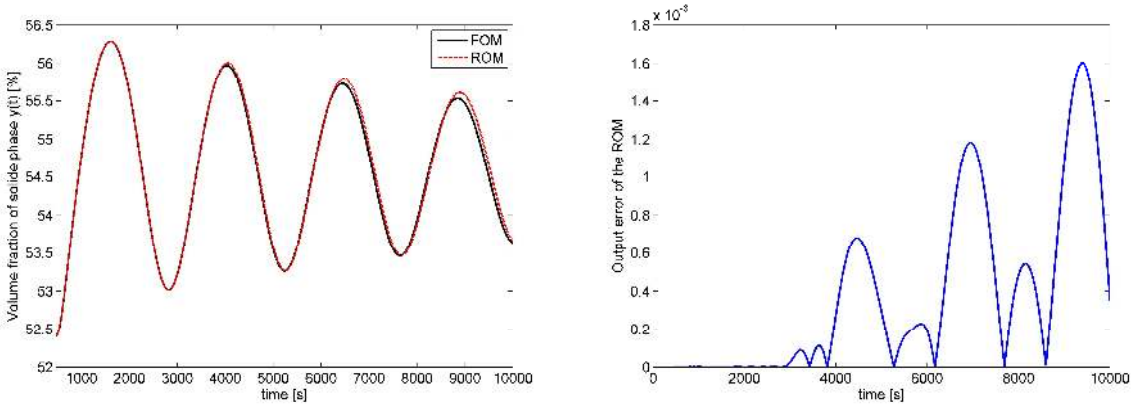


Figure 5: Adaptive adjustment of r and l for different error tolerances ϵ.



(a) (b)

Figure 6: Output comparison over the whole interval $[0, T_5]$.



(a) (b)

Figure 7: Output comparison over the periodic interval $[T_1, T_5]$.

Table 1: ROMs adaptively obtained for different tolerances ϵ, $r_0 = 3, l_0 = 6$, $[T_0, T_2]$.

ϵ	(r, l)	$\bar{\Delta}$	$\bar{\varepsilon}_y$	Runtime(s)	ROM sim. time(%)
$1 \cdot 10^{-1}$	$(6, 11)$	0.0278	0.0023	149.05	29
$1 \cdot 10^{-2}$	$(12, 14)$	$7.4 \cdot 10^{-3}$	$5.55 \cdot 10^{-4}$	450.78	10
$1 \cdot 10^{-3}$	$(16, 19)$	$7.54 \cdot 10^{-4}$	$5.39 \cdot 10^{-5}$	478.72	10

Table 2: ROMs adaptively obtained for different tolerances ϵ, $r_0 = 7, l_0 = 7$, $[T_0, T_2]$.

ϵ	(r, l)	$\bar{\Delta}$	$\bar{\varepsilon}_y$	Runtime(s)	ROM sim. time(%)
$1 \cdot 10^{-1}$	$(8, 13)$	0.0218	0.0018	71.82	60
$1 \cdot 10^{-2}$	$(12, 14)$	0.0074	$5.55 \cdot 10^{-4}$	315.40	14
$1 \cdot 10^{-3}$	$(16, 19)$	$7.54 \cdot 10^{-4}$	$5.39 \cdot 10^{-5}$	416.88	11

Table 3: ROMs adaptively obtained for different tolerances ϵ, $r_0 = 5, l_0 = 10$, $[T_1, T_2]$.

ϵ	(r, l)	$\bar{\Delta}$	$\bar{\varepsilon}_y$	Runtime(s)	ROM sim. time(%)
$1 \cdot 10^{-1}$	$(5, 10)$	0.0057	$5.01 \cdot 10^{-4}$	49.47	72
$1 \cdot 10^{-2}$	$(5, 10)$	0.0057	$5.01 \cdot 10^{-4}$	49.47	72
$1 \cdot 10^{-3}$	$(7, 10)$	$9.65 \cdot 10^{-4}$	$8.39 \cdot 10^{-5}$	149.71	24
$1 \cdot 10^{-4}$	$(11, 11)$	$5.59 \cdot 10^{-5}$	$4.31 \cdot 10^{-6}$	287.23	13
$1 \cdot 10^{-5}$	$(14, 16)$	$9.29 \cdot 10^{-6}$	$5.33 \cdot 10^{-7}$	314.54	12
$1 \cdot 10^{-6}$	$(21, 17)$	$3.72 \cdot 10^{-7}$	$2.50 \cdot 10^{-8}$	563	16
$1 \cdot 10^{-7}$	$(24, 20)$	$9.41 \cdot 10^{-8}$	$3.56 \cdot 10^{-9}$	428.29	20

Table 4: ROMs constructed by standard POD-DEIM with $\epsilon_{svd} = 10^{-10}$.

ROM : $\Sigma_{r^*l^*}$	time interval	$\bar{\Delta}$	$\bar{\varepsilon}_y$	Sim. time (s)	$\frac{\text{Sim}_{r,l}}{\text{Sim}_{r^*,l^*}}$
$r^* = 61, l^* = 66$	$[0, T_2]$	$3.97 \cdot 10^{-4}$	$3.37 \cdot 10^{-5}$	477.40	0.87
$r^* = 38, l^* = 41$	$[T_1, T_2]$	$5.34 \cdot 10^{-8}$	$4.01 \cdot 10^{-10}$	272.72	0.55

Table 5: Comparison over the whole time interval.

model	time interval	dimension	Sim. time (s)	speed-up factor
FOM	$[0, T_5]$	$n = 18400$	7030.9	–
ROM	$[0, T_5]$	$r = 16, l = 19$	187.06	38
FOM	$[T_1, T_5]$	$n = 18400$	6722.8	–
ROM	$[T_1, T_5]$	$r = 14, l = 16$	156.20	43