ACTIVE RANK-1 UPDATES TO POD BASED REDUCED ORDER MODELS*

ABHINAV GUPTA† AND RENEE SWISCHUK‡

Abstract. Simulating physical systems is often a computationally expensive task. Data driven reduced order models are typically applied in these situations and provide accurate and low cost evaluations of the system. In some cases, changes in resources or system state may cause such models to become unreliable and inaccurate in new domains. Costly reconstructions of the reduced order model at each system change can sacrifice the speed benefits of using a reduced order model. This project investigates a method for efficiently updating a reduced order model online, as new data becomes available corresponding to system changes. New data is incorporated as low rank updates to proper orthogonal decomposition (POD) basis and reduced operators that allow the reduced order model to adapt with the system domain. This dynamic approach is competitive in accuracy compared to rebuilding the method from scratch, but is able to significantly decrease time complexity. The method is demonstrated on the advection-diffusion equation and we show that after sufficient data has been incorporated, our dynamic reduced order model is able to converge to the accuracy of the true model by making efficient updates online.

Key words. Proper Orthogonal Decomposition (POD), Rank-1 Updates, SVD, Reduced order models

1. Introduction. Model reduction of parameterized partial differential equations (PDEs) involves constructing a reduced order model (ROM) using the available full state samples corresponding to a particular region in the parameter space. One such method projects the full order model using the Proper Orthogonal Decomposition (POD) basis. Reduced order models can be efficiently used to predict accurate responses of the system in a local region of the parameter space. But, the ROM becomes inaccurate if the set of parameter values lies far from the original region used to construct the ROM. In many real situations, changes in resources or the evolution of the state may require this parameter region to change over time. Although, rebuilding a new ROM for a changing parameter space can become computationally expensive, as a new POD basis must be constructed with each change. According to Peherstorfer and Willcox [2], an evolving ROM can be constructed online, by representing changes to the model as low rank updates from the new parameter region. Using this low rank representation, a powerful and efficient algorithm for computing an updated singular value decomposition (SVD) [1], and thus a new POD basis, can be applied to dynamically update our ROM at a low cost, while maintaining accuracy over the parameter region. The rest of this paper is formatted as follows. Section 2 includes a discussion of the reduced order modeling framework as well as the problem setting of adapting ROMs online from snapshot updates. We detail the algorithms used for making rank-1 updates to a SVD and adapting our reduced basis and operators. We demonstrate our dynamic ROM in Section 3, with an application of the advection-diffusion equation and analyze the benefits of the method. Section 4 concludes the paper. This project is entirely inspired by Peherstorfer and Willcox [2].

2. Algorithm. In this section, we will see the building blocks of the dynamic ROM.

*Submitted to the instructor of 18.335 on 05/17/2018.

†Department of Mechanical Engineering, MIT (guptaa@mit.edu).

‡Department of Aeronautics and Astronautics, MIT (swischuk@mit.edu).

2.1. Full Order Models. Consider a full order model (FOM) based on a parameterized PDE. Discretization leads to the system of equations

$$\mathbf{A}(\boldsymbol{\mu})\mathbf{y}(\boldsymbol{\mu}) = \mathbf{f}(\boldsymbol{\mu}), \tag{2.1}$$

with $\mathcal{N} \in \mathbb{N}$ spatial dimensions coming from the discretization of PDE. Here $\boldsymbol{\mu} = \left[\mu_1, \ldots, \mu_d\right] \in \mathcal{D}$ with $d \in \mathbb{N}$ is the parameter set, which acts as an input to our system. We have the operator $\mathbf{A}(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$, the state vector $\mathbf{y}(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}}$, and the right-hand side $\mathbf{f}(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}}$.

We assume that the operator $\mathbf{A}(\boldsymbol{\mu})$ can be represented with an affine parameter dependence, thus it can be written as a linear combination,

$$\mathbf{A}(\boldsymbol{\mu}) = \sum_{i=1}^{l_A} \theta_A^i(\boldsymbol{\mu})\mathbf{A}^i, \tag{2.2}$$

of operators independent of $\boldsymbol{\mu}$

$$\mathbf{A}^i, \ldots, \mathbf{A}^{l_A} \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}, \tag{2.3}$$

and $l_A \in \mathbb{N}$ functions $\theta_A^1, \ldots, \theta_A^{l_A} : \mathcal{D} \to \mathbb{R}$. Similarly we can write $\mathbf{f}(\boldsymbol{\mu})$ also as a linear combination of parameter independent vectors $\mathbf{f^1}, \ldots, \mathbf{f^{l_f}} \in \mathbb{R}^{\mathcal{N}}$ and $l_f \in \mathbb{N}$ functions $\theta_f^1, \ldots, \theta_f^{l_f} : \mathcal{D} \to \mathbb{R}$,

$$\mathbf{f}(\boldsymbol{\mu}) = \sum_{i=1}^{l_f} \theta_f^i(\boldsymbol{\mu})\mathbf{f}^i. \tag{2.4}$$

In general, $\mathcal{N}$ is very large, making simulations of the FOM unfeasible. This issue is overcome with the application of reduced order models.

2.2. Reduced Order Models. Consider a snapshot matrix containing $m \in \mathbb{N}$ linearly independent state vectors generated by solving 2.1 for parameter sets $\boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_m \in \mathcal{D}^o$,

$$\mathbf{Y}_o = \left[\mathbf{y}(\boldsymbol{\mu_1}), \ldots, \mathbf{y}(\boldsymbol{\mu_m})\right] \in \mathbb{R}^{\mathcal{N}\times m}. \tag{2.5}$$

POD is a method to construct an n-dimensional basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{R}^{\mathcal{N}}$ such that the snapshots 2.5 are optimally represented by their orthogonal projections onto the subspace $\text{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\} \subset \text{span}\{\mathbf{y}(\boldsymbol{\mu_1}), \ldots, \mathbf{y}(\boldsymbol{\mu_m})\} \subset \mathbb{R}^{\mathcal{N}}$. The POD basis vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{R}^{\mathcal{N}}$ are the left singular vectors corresponding to the n largest singular values of the snapshot matrix 2.5. Hence, to compute the POD basis for the snapshots in 2.5, we start with the singular value decomposition (SVD) of the snapshot matrix $\mathbf{Y_o}$, and the singular vectors corresponding to the n largest singular values forms the required POD basis. The truncated SVD is defined as

$$\mathbf{Y}_0 \approx \mathbf{V}_0\boldsymbol{\Sigma}_0\mathbf{W}_0^T, \tag{2.6}$$

where the matrices $\mathbf{V}_o = \left[\mathbf{v}_1, \ldots, \mathbf{v}_n\right] \in \mathbb{R}^{\mathcal{N}\times n}$, $\mathbf{W}_o = \left[\mathbf{w}_1, \ldots, \mathbf{w}_n\right] \in \mathbb{R}^{m\times n}$ are the n left and right singular vectors, respectively and $\boldsymbol{\Sigma}_o \in \mathbb{R}^{n\times n}$ is a diagonal matrix containing the largest n singular values in decreasing value. To derive the ROM of the FOM in 2.1, we start by constructing μ-independent reduced operators,

$$\tilde{\mathbf{A}}_o^i = \mathbf{V}_o^T\mathbf{A}^i\mathbf{V}_o \in \mathbb{R}^{n\times n}, \qquad i = 1, \ldots, l_A \tag{2.7}$$

and the μ-independent reduced right-hand sides

$$\tilde{\mathbf{f}}_o^i = \mathbf{V}_o^T \mathbf{f}^i \in \mathbb{R}^n, \qquad i = 1, \ldots, l_f \tag{2.8}$$

by taking their orthogonal projections on $\mathbf{V}_o$. By defining the reduced state vector as $\tilde{\mathbf{y}}_o = \mathbf{V}_o^T \mathbf{y}^i \in \mathbb{R}^n$, we can write the ROM as,

$$\tilde{\mathbf{A}}_o(\mu)\tilde{\mathbf{y}}_o = \sum_{i=1}^{l_A} \theta_A^i(\boldsymbol{\mu})\tilde{\mathbf{A}}_o^i \tilde{\mathbf{y}}_o = \sum_{i=1}^{l_f} \theta_f^i(\boldsymbol{\mu})\tilde{\mathbf{f}}_o^i = \tilde{\mathbf{f}}_o(\boldsymbol{\mu}) \tag{2.9}$$

Hence instead of solving the full order system of dimension $\mathcal{N}$, we just need to solve a system of reduced dimension n, and in general we choose $n << \mathcal{N}$ based on the singular value spectrum.

2.3. Problem Setting. Since our ROM is based on the POD basis, $\mathbf{V}_o$, corresponding to the region $\mathcal{D}^o \subset \mathcal{D}$, solutions will be highly accurate for any $\boldsymbol{\mu} \in \mathcal{D}^o$. Now, consider that due to an internal system or resource change, our region of interest changes from $\mathcal{D}^0$ to $\mathcal{D}^1$ in the parameter space. Our original ROM becomes obsolete in this new region. In such a case, we would like to adapt our ROM to the new region of interest by updating our POD basis $\mathbf{V}_o$, reduced operators $\tilde{\mathbf{A}}_o^i$ and the reduced right-hand side $\tilde{\mathbf{f}}_o^i$, using full-state samples corresponding to parameters in $\mathcal{D}^1$. Recall that the FOM is very expensive to evaluate, thus samples from this new region will slowly become available over time. Instead of waiting for a sufficient number of new snapshots to become available, we consider successively updating our ROM by initially replacing or appending columns to our original snapshot matrix $\mathbf{Y}_o$. With the arrival of each new snapshot, the SVD is computed and a new ROM is constructed. Eventually, this will produce a ROM that is highly accurate in the new region, $\mathcal{D}^1$, but at an extremely high cost which may render the approach infeasible. Hence, our goal is to perform these updates in a computationally cheap way, using fast low-rank modifications of the thin singular value decomposition [1] as suggested by Peherstorfer [2].

In what follows, we will successively adapt the ROM in $h = 1, \ldots, m'$ adaptivity steps during the online phase where $m' \in \mathbb{N}$. At each step h we receive a full state-vector $\hat{\mathbf{y}}(\boldsymbol{\mu}_{m+h}) \in \mathbb{R}^{\mathcal{N}}$ with $\boldsymbol{\mu}_{m+h} \in \mathcal{D}^1$. At the first adaptivity step $h = 1$, we receive the sensor sample $\hat{\mathbf{y}}(\boldsymbol{\mu}_{m+1})$. We first consider replacing the snapshot $\mathbf{y}(\boldsymbol{\mu}_1)$ in the snapshot matrix $\mathbf{Y}_o$ with this new sample and denote the new snapshot matrix by $\mathbf{Y}_1$. We continue this process and at step h we obtain the updated snapshot matrix

$$\mathbf{Y}_h = \left[\hat{\mathbf{y}}(\boldsymbol{\mu}_{m+1}), \ldots, \hat{\mathbf{y}}(\boldsymbol{\mu}_{m+h}), \mathbf{y}(\boldsymbol{\mu}_{h+1}), \ldots, \mathbf{y}(\boldsymbol{\mu}_m)\right] \in \mathbb{R}^{\mathcal{N}\times m}. \tag{2.10}$$

We can generalize an update at step h as a rank-1 update to the snapshot matrix at step $h-1$, i.e.

$$\mathbf{Y}_h = \mathbf{Y}_{h-1} + \mathbf{a}\mathbf{e}_h^T, \tag{2.11}$$

where $\mathbf{a} = \hat{\mathbf{y}}(\boldsymbol{\mu}_{m+h}) - \mathbf{y}(\boldsymbol{\mu}_h) \in \mathbb{R}^{\mathcal{N}}$ and $\mathbf{e}_h \in \mathbb{R}^m$ is a unit vector with 1 at the h-th component and 0 everywhere else. This formulation makes the algorithm presented in [1] applicable, which is explained in the following subsection.

2.4. Rank-1 updates to SVD. At adaptivity step, h, we make a rank-1 update to our snapshot matrix. To subsequently update our ROM, we must also incorporate this snapshot information into a new POD basis. This requires a rank-1 update to

the SVD of $\mathbf{Y}_{h-1}$. Assuming, we already know the rank-n SVD of $\mathbf{Y}_{h-1}$, we make use of the algorithm derived by Brand in [1]. Let the rank-n SVD of $\mathbf{Y}_{h-1}$ be given by $\mathbf{V}_{h-1}\mathbf{\Sigma}_{h-1}\mathbf{W}_{h-1}^T$, where $\mathbf{V}_{h-1} \in \mathbb{R}^{\mathcal{N}\times n}$, $\mathbf{W}_{h-1} \in \mathbb{R}^{m\times n}$ and $\mathbf{\Sigma}_{h-1} \in \mathbb{R}^{n\times n}$. We are interested in the SVD of,

$$\mathbf{Y}_{h-1} + \mathbf{a}\mathbf{e}_h^T = \begin{bmatrix}\mathbf{V}_{h-1} & \mathbf{a}\end{bmatrix} \begin{bmatrix}\mathbf{\Sigma}_{h-1} & \mathbf{0} \\ \mathbf{0} & 1\end{bmatrix} \begin{bmatrix}\mathbf{W}_{h-1} & \mathbf{e}_h\end{bmatrix}^T \tag{2.12}$$

Also we are interested in the case where $\text{rank}(\mathbf{Y}_{h-1} + \mathbf{a}\mathbf{e}_h^T) \leq n+1 < \min(\mathcal{N}, m)$ because in general $n << \mathcal{N}$.

Although $\begin{bmatrix}\mathbf{\Sigma}_{h-1} & \mathbf{0} \\ \mathbf{0} & 1\end{bmatrix}$ is diagonal, but $\begin{bmatrix}\mathbf{V}_{h-1} & \mathbf{a}\end{bmatrix}$ and $\begin{bmatrix}\mathbf{W}_{h-1} & \mathbf{e}_h\end{bmatrix}$ are not orthonormal bases. Hence a logical, first step would be to make them orthonormal. Lets first focus on $\begin{bmatrix}\mathbf{V}_{h-1} & \mathbf{a}\end{bmatrix}$. We project out columns of $\mathbf{V}_{h-1}$ from $\mathbf{a}$ and then normalize it. We define,

$$\mathbf{m} = \mathbf{V}_{h-1}^T\mathbf{a}; \quad \mathbf{p} = \mathbf{a} - \mathbf{V}_{h-1}\mathbf{m}; \quad R_a = ||\mathbf{p}||; \quad \mathbf{P} = {R_a}^{-1}\mathbf{p} \tag{2.13}$$

hence,

$$\begin{bmatrix}\mathbf{V}_{h-1} & \mathbf{a}\end{bmatrix} = \begin{bmatrix}\mathbf{V}_{h-1} & \mathbf{P}\end{bmatrix} \begin{bmatrix}\mathbf{I} & \mathbf{m} \\ 0 & R_a\end{bmatrix} \tag{2.14}$$

Similarly for $\begin{bmatrix}\mathbf{W}_{h-1} & \mathbf{e}_h\end{bmatrix}$ we could define,

$$\mathbf{n} = \mathbf{W}_{h-1}^T\mathbf{e}_h; \quad \mathbf{q} = \mathbf{e}_h - \mathbf{W}_{h-1}\mathbf{n}; \quad R_{e_h} = ||\mathbf{q}||; \quad \mathbf{Q} = {R_{e_h}}^{-1}\mathbf{q} \tag{2.15}$$

hence,

$$\begin{bmatrix}\mathbf{W}_{h-1} & \mathbf{e}_h\end{bmatrix} = \begin{bmatrix}\mathbf{W}_{h-1} & \mathbf{Q}\end{bmatrix} \begin{bmatrix}\mathbf{I} & \mathbf{n} \\ 0 & R_{e_h}\end{bmatrix} \tag{2.16}$$

Rewriting 2.12 as,

$$\mathbf{Y}_{h-1} + \mathbf{a}\mathbf{e}_h^T = \begin{bmatrix}\mathbf{V}_{h-1} & \mathbf{P}\end{bmatrix} \mathbf{K} \begin{bmatrix}\mathbf{W}_{h-1} & \mathbf{Q}\end{bmatrix}^T \tag{2.17}$$

where,

$$\mathbf{K} = \begin{bmatrix}\mathbf{I} & \mathbf{m} \\ 0 & R_a\end{bmatrix} \begin{bmatrix}\mathbf{\Sigma}_{h-1} & \mathbf{0} \\ \mathbf{0} & 1\end{bmatrix} \begin{bmatrix}\mathbf{I} & \mathbf{n} \\ 0 & R_{e_h}\end{bmatrix}^T = \begin{bmatrix}\mathbf{\Sigma}_{h-1} & \mathbf{0} \\ \mathbf{0} & 0\end{bmatrix} + \begin{bmatrix}\mathbf{m} \\ R_a\end{bmatrix} \begin{bmatrix}\mathbf{n} \\ R_{e_h}\end{bmatrix}^T \tag{2.18}$$

Since $\mathbf{K} \in \mathbb{R}^{(n+1)\times(n+1)}$, taking its SVD (let $\mathbf{K} = \mathbf{V'}\mathbf{\Sigma'}\mathbf{W'}^T$) is comparatively cheaper than the SVD of our snapshot matrix. Also it turns out that $\mathbf{V'}$ and $\mathbf{W'}$ are the required rotations to convert $\begin{bmatrix}\mathbf{V}_{h-1} & \mathbf{P}\end{bmatrix}$ and $\begin{bmatrix}\mathbf{W}_{h-1} & \mathbf{Q}\end{bmatrix}$ to left and right singular vectors respectively. Finally we can write,

$$\begin{aligned}\mathbf{Y}_{h-1} + \mathbf{a}\mathbf{e}_h^T &= \left(\begin{bmatrix}\mathbf{V}_{h-1} & \mathbf{P}\end{bmatrix} \mathbf{V'}\right) \mathbf{\Sigma'} \left(\begin{bmatrix}\mathbf{W}_{h-1} & \mathbf{Q}\end{bmatrix} \mathbf{W'}\right)^T \\ &= \mathbf{V}_h\mathbf{\Sigma}_h\mathbf{W}_h^T \end{aligned} \tag{2.19}$$

which is the required SVD.

The above SVD update algorithm could also be extended to the case when we want to append new columns to the snapshot matrix. After h-adaptivity steps, we obtain the following snapshot matrix,

$$\mathbf{Y}_h = \begin{bmatrix}\mathbf{y}(\mu_\mathbf{1}), \ldots, \mathbf{y}(\mu_\mathbf{m}), \hat{\mathbf{y}}(\mu_{m+1}), \ldots, \hat{\mathbf{y}}(\mu_{m+h})\end{bmatrix} \in \mathbb{R}^{\mathcal{N}\times(m+h)} \tag{2.20}$$

Appending new snapshots can also be written in the form of a rank-1 update as in 2.11, where now $\mathbf{a} = \hat{\mathbf{y}}(\boldsymbol{\mu}_{m+h}) \in \mathbb{R}^{\mathcal{N}}$ and $\mathbf{e}_h \in \mathbb{R}^{m+h}$ with all 0's except a 1 in the $(m+h)^{th}$ element.

The projection steps in 2.13 and 2.15 are currently written in a Classical Gram Schmidt (CGS) fashion, we can replace them with Modified Gram Schmidt (MGS) fashion without incurring any additional cost, as shown in Algorithms 2.3 and 2.4. The effect of CGS vs MGS on accuracy of the scheme is be discussed later in the Section 3.

2.5. Adapting the reduced operators. After computing the updated POD basis $\mathbf{V}_h$, we can now adapt our reduced operators and right hand side vectors as,

$$\tilde{\mathbf{A}}_h^i = \mathbf{V}_h^T \mathbf{A}^i \mathbf{V}_h, \qquad i = 1, \ldots, l_A \tag{2.21}$$

$$\tilde{\mathbf{f}}_h^i = \mathbf{V}_h^T \mathbf{f}^i, \qquad i = 1, \ldots, l_f \tag{2.22}$$

This may involve unnecessary matrix multiplications, and one would like to make use of the previous reduced operators, $\tilde{\mathbf{A}}_{h-1}^i$ and $\tilde{\mathbf{f}}_{h-1}^i$.

Consider rewriting $\mathbf{V}_h$ from 2.19 as

$$\mathbf{V}_h = \mathbf{V}_{h-1}\mathbf{V}_h' + \mathbf{c}\mathbf{d}^T, \tag{2.23}$$

where $\mathbf{V}_h' = \mathbf{V}'(1:\text{end}-1, 1:\text{end}-1)$, $\mathbf{c} = \mathbf{P}$ and $\mathbf{d} = \mathbf{V}'(\text{end}, 1:\text{end}-1)^T$ in MATLAB's notation. Taking advantage of the structure 2.23, we represent 2.21 as

$$\begin{aligned}
\mathbf{V}_h^T \mathbf{A}^i \mathbf{V}_h &= \left(\mathbf{V}_{h-1}\mathbf{V}_h' + \mathbf{c}\mathbf{d}^T\right)^T \mathbf{A}^i \left(\mathbf{V}_{h-1}\mathbf{V}_h' + \mathbf{c}\mathbf{d}^T\right) \\
&= \mathbf{V}_h'^T \mathbf{V}_{h-1}^T \mathbf{A}^i \mathbf{V}_{h-1} \mathbf{V}_h' + \mathbf{d}\mathbf{c}^T \underbrace{\mathbf{A}^i \mathbf{V}_{h-1}}_{\mathbf{B}_{h-1}^i} \mathbf{V}_h' + \underbrace{\mathbf{V}_h'^T \mathbf{V}_{h-1}^T \mathbf{A}^i \mathbf{c}\mathbf{d}^T + \mathbf{d}\mathbf{c}^T \mathbf{A}^i \mathbf{c}\mathbf{d}^T}_{\mathbf{C}_h^i \mathbf{c}\mathbf{d}^T} \\
&= \underbrace{\mathbf{V}_h'^T}_{n\times n} \underbrace{\tilde{\mathbf{A}}_{h-1}^i}_{n\times n} \underbrace{\mathbf{V}_h'}_{n\times n} + \underbrace{\mathbf{d}\mathbf{c}^T}_{n\times\mathcal{N}} \underbrace{\mathbf{B}_{h-1}^i}_{\mathcal{N}\times n} \underbrace{\mathbf{V}_h'}_{n\times n} + \underbrace{\mathbf{C}_h^i}_{n\times\mathcal{N}} \underbrace{\mathbf{c}\mathbf{d}^T}_{\mathcal{N}\times n}
\end{aligned} \tag{2.24}$$

where $\mathbf{B}_{h-1}^i = \mathbf{A}^i \mathbf{V}_{h-1} \in \mathbb{R}^{\mathcal{N}\times n}$ and $\mathbf{C}_h^i = \mathbf{V}_h^T \mathbf{A}^i \in \mathbb{R}^{n\times\mathcal{N}}$ are some auxiliary quantities for $i = 1, \ldots, l_A$. These auxiliary quantities can also be constructed recursively,

$$\mathbf{B}_h^i = \mathbf{B}_{h-1}^i \mathbf{V}_h' + \mathbf{A}^i \mathbf{c}\mathbf{d}^T, \qquad i = 1, \ldots, l_A \tag{2.25}$$

$$\mathbf{C}_h^i = \mathbf{V}_h'^T \mathbf{C}_{h-1}^i + \mathbf{d}\mathbf{c}^T \mathbf{A}^i, \qquad i = 1, \ldots, l_A \tag{2.26}$$

and $\mathbf{B}_o^i = \mathbf{A}^i \mathbf{V}_o$ and $\mathbf{C}_0^i = \mathbf{V}_0^T \mathbf{A}^i$ are computed initially. Similarly, we can update our reduced right hand side as

$$\tilde{\mathbf{f}}_h^i = \mathbf{V}_h' \tilde{\mathbf{f}}_{h-1}^i + (cd')'\mathbf{f}^i \tag{2.27}$$

2.6. Implementation. By combining low rank updates to SVD and the adapted reduced operators explained in Sections 2.4 and 2.5, we can compute a dynamic ROM. Algorithm 2.1 explains the rank-1 updates to our singular vectors and values, and Algorithm 2.2 shows how to update the auxillary quantities. The entire process is shown in Algorithm 2.5. The basic steps are to begin with an initial ROM and at each snapshot update, compute our new SVD, update our snapshot matrix, update our auxillary quantities and compute the reduced operators needed to make predictions with our new ROM.

Algorithm 2.1 Rank-1 Update to SVD

Function Rank1UpdateSVD($V_{h-1}, S_{h-1}, W_{h-1}, a, b$)
$m = V_{h-1}^T a; \quad p = a - V_{h-1} m; \quad R_a = ||p|| l \quad P = R_a^{-1} p;$
$n = W_{h-1}^T b; \quad q = b - W_{h-1} n; \quad R_b = ||q||; \quad Q = R_b q;$

$$K = \begin{bmatrix} S & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} + \begin{bmatrix} m \\ ||p|| \end{bmatrix} \begin{bmatrix} n \\ ||q|| \end{bmatrix}^T$$

$[V', S', W'] = SVD(K)$
$V_h = \begin{bmatrix} V_{h-1} & P \end{bmatrix} V'$
$W_h = \begin{bmatrix} W_{h-1} & Q \end{bmatrix} W'$
return V_h, V', S', W_h, W'

Algorithm 2.2 Auxillary Quantities

Function auxQU($B_{h-1}, C_{h-1}, A, c, d, V_h'$)
$B_h^i = B_{h-1}^i V_h' + A^i c d^T$ for $i = 1, ..., l_A$
$C_h^i = V_h'^T C_{h-1}^i + d c^T A^i$ for $i = 1, ..., l_A$
return C_h, B_h

Algorithm 2.3 MGS for $\mathbf{p}$

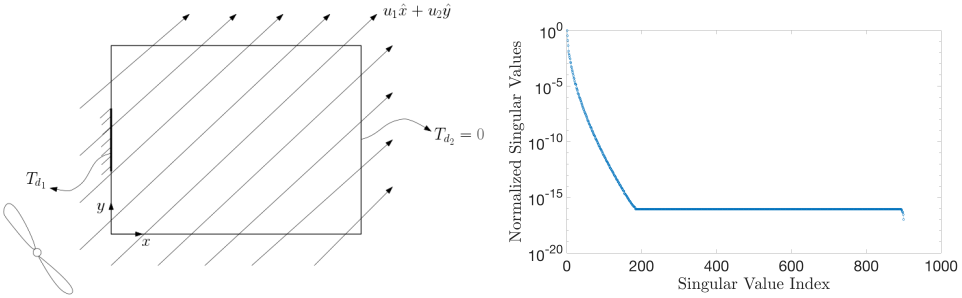
$\mathbf{p} = \mathbf{a}$
for $i = 1 : n$
  $\mathbf{m}(i, 1) = \mathbf{V}_{h-1}(:, i)^T \mathbf{p}$
  $\mathbf{p} = \mathbf{p} - \mathbf{V}_{h-1}(:, i)(\mathbf{m}(i, 1))$

Algorithm 2.4 MGS for $\mathbf{q}$

$\mathbf{q} = \mathbf{e}_h$
for $i = 1 : n$
  $\mathbf{n}(i, 1) = \mathbf{W}_{h-1}(:, i)^T \mathbf{q}$
  $\mathbf{q} = \mathbf{q} - \mathbf{W}_{h-1}(:, i)(\mathbf{n}(i, 1))$

Algorithm 2.5 Dynamic POD Updates

Function AdaptPOD($V_{h-1}, S_{h-1}, W_{h-1}, \tilde{A}_{h-1}, \tilde{f}_{h-1}, A, f, B_{h-1}, C_{h-1}$)
Receive new snapshot, $\hat{y}_h(\mu)$
Update snapshot matrix, Y
  $a = \hat{y}_h(\mu) - Y(:, h)$
  $e_h = h^{th}$ canonical unit vector.
  $Y = Y + a e_h^T$
$[V_h, V', S', W_h, W'] = \text{Rank1UpdateSVD}(V_{h-1}, S_{h-1}, W_{h-1}, a, e_h)$
$c = P; d = V'[end, 1 : end - 1]^T; V_h' = V'[1 : end - 1, 1 : end - 1]$
$[B_h, C_h] = \text{auxQU}(B_{h-1}, C_{h-1}, A, c, d, V_h')$
Adapt reduced operators
  $\tilde{A}_h^i = V_h'^T \tilde{A}_{h-1}^i V_h' + d c^T B_{h-1}^i V_h' + C_h^i c d^T$
  $\tilde{f}^i{}_h = V_h^T f_{h-1}^i + (c d^T)^T f^i$
return



(a) Schematic of the physical domain of interest. $T_{d_1} = 50$

(b) Singular values for 900 snapshots collected in a single region of the input domain.

Fig. 1: (a) Physical domain considered for Eq. 3.1 and (b) decay in singular values.

3. Numerical Results.

3.1. Application. To evaluate the performance of the algorithm, we consider an application to the 2-D advection-diffusion equation. The equation is as follows

$$u_1 \frac{\partial T}{\partial x} + u_2 \frac{\partial T}{\partial y} - \kappa \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) = 0, \tag{3.1}$$

where $\boldsymbol{\mu} = [u_1, u_2] \in \mathcal{D}$ is our input representing the velocity in the x and y directions, T is our solution quantity (state vector) representing the temperature distribution in the 2-D space and κ is the coefficient of thermal diffusivity. A schematic of our domain is presented in Figure 1a. We have a non-zero Dirichlet boundary condition on a partial portion of the left boundary, $T = 50$ from $y = [0.3, 0.7]$, while all other boundary have zero Dirichlet boundary condition. The solution to this equation is generally well behaved allowing the dynamics of the system to be captured well by a Galerkin projection based technique such as POD. We consider a reduced order model that has been built using 900 full order solutions simulated in the region $\boldsymbol{\mu} \in [1, 2] \times [0, 1] = \mathcal{D}^0 \subset \mathcal{D}$. Figure 1b shows a sharp decrease in the normalized magnitude of the singular values, indicating a well posed problem for POD based model reduction.

In general, ROMs perform well in the local region of the parameter space that was used to construct the model. Figure 2c shows a high accuracy solution when the ROM is applied locally. Although, if we attempt to produce solutions that lie in a new parameter domain $\boldsymbol{\mu} \in [0, 1] \times [1, 2] \in \mathcal{D}^1 \subset \mathcal{D}$, the performance diminishes as shown in Figure 2d. This type of behavior provides a setting that will allow us to emphasize the advantages of having a dynamically updated ROM.

Our adaptivity steps are performed by collecting new snapshots from a different region of the input domain. The original region is $\mathcal{D}^0$ and at each update we will incorporate snapshots from region $\mathcal{D}^1$. We will demonstrate the method of replacing snapshots as well as appending snapshots.

3.2. Comparisons and Analysis of Results. As a benchmark comparison, we consider the approach of taking the full SVD at each update step. While this is expensive, it is highly accurate and provides a robust measure. As a performance comparison we consider the approach of appending columns, instead of replacing them, as well as using the static ROM that we started with.

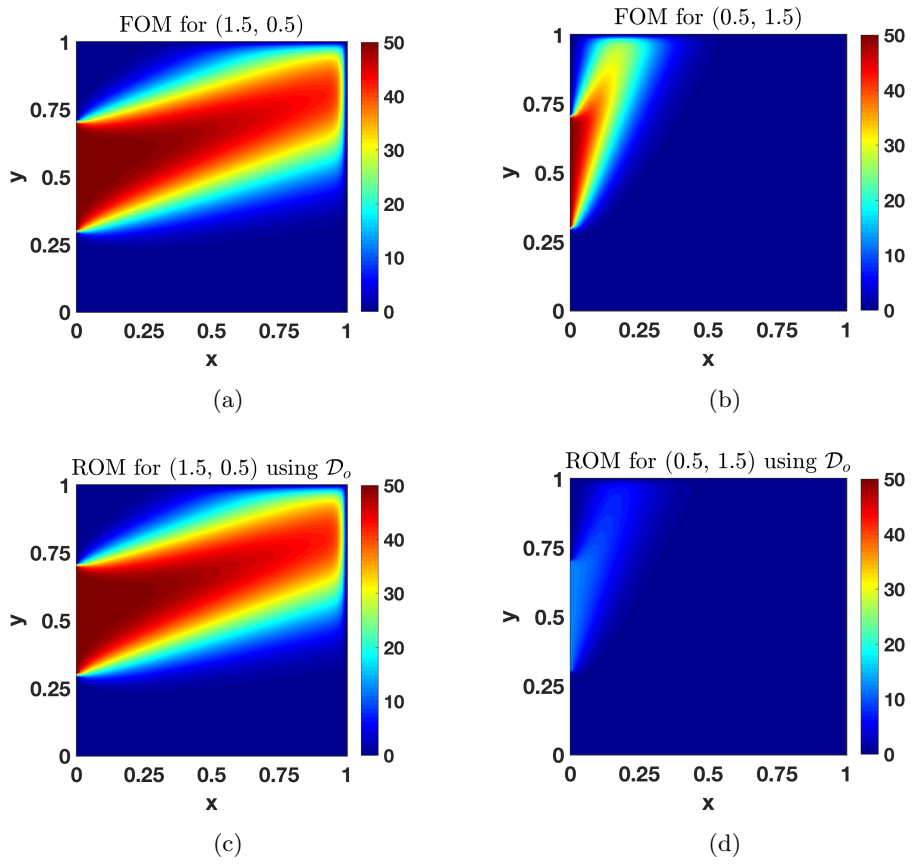
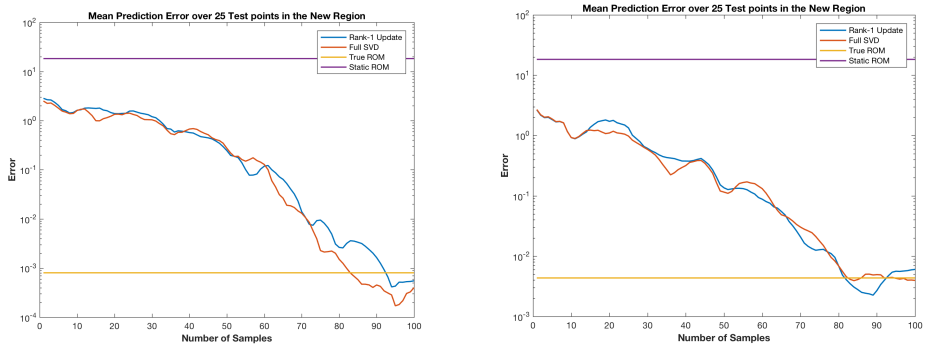


Fig. 2: Local performance of a ROM. Top: Full order model solutions. Bottom: Reduced order model solutions. Left: a solution produced by an input that lives in the domain of the ROM. Right: a solution produced by an input that lives outside the range of the domain.

The method of appending columns should offer similar reduction in computational cost when compared to full SVD, since the algorithm for computing rank-1 updates of an SVD can be easily adapted to account for appending columns. Although, the accuracy of this method is difficult to predict, particularly since the number of POD basis vectors required typically increases with the size of the snapshot matrix. That said, fixing the number of POD basis vectors for this method, should ideally make it perform worse. Although, there is the possibility that the two regions share similar characteristics that may be captured better when all the data is considered. In our application, we have chosen a PDE with a linear behavior that produces predictable results. If instead we chose a system whose dynamics change drastically throughout the input domain or is highly non-linear, we would expect very poor performance when appending columns if we do not increase the number of POD basis. Take for example, structural health monitoring. If we have a ROM for the behavior of some material when it is undamaged, and the material then becomes damaged, the dynamics of the



(a) Snapshots replaced (b) Snapshots appended

Fig. 3: Relative average error for the prediction of 25 test samples in the new region against the FOM when dynamic ROM is updated using cheap Rank-1 updates vs taking full SVD at each adaptivity step.

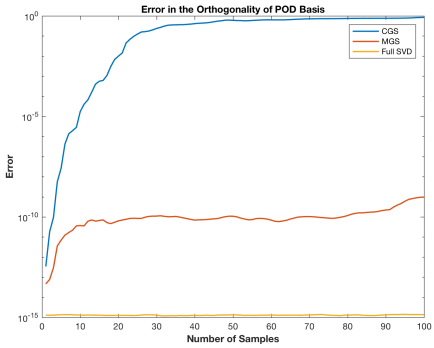


Fig. 4: Error in maintaining orthogonality of POD basis when projection step in Rank-1 update to SVD algorithm is done in CGS vs MGS fashion. Benchmark for comparison is taking full SVD at each adaptivity step.

model that described the damaged material will be entirely different. This situation is one where the proposed method is a clear choice.

The method of using the static ROM, is a computationally free method online. The original ROM is produced offline and no extra costs occur when data is being collected online. While this method is extremely cheap, the performance in new regions is very likely to be bad. The main goal of this comparison is to point out the advantages that can be gained while considering new data online.

Figure 3a shows the comparison results when new snapshots are replaced. In this figure, the true model corresponds to the ROM constructed using only snapshots in the new region. Figure 3b shows the same comparisons except we are now appending new snapshots. In this figure, the true model corresponds to the ROM constructed using snapshots from both regions. In both plots we are able to see the error in our

adaptive reduced order model converging to the error of the true model. We also see that the our dynamic ROM is able to follow the benchmark comparison of taking the full SVD at each step. In Figure 4, we confirm the fact that in computing the projection steps 2.13 and 2.15 in a CGS fashion, we indeed loose orthogonality of POD basis after a few adaptivity step as compared to the MGS fashion (Algorithms 2.3 and 2.4).

3.3. Time Complexity. Calculating the SVD of a matrix is typically done in two steps [3]. The first step is to bidiagonalize the matrix using a method such as householder reflectors. The second step is to then diagonalized this matrix using some variant of QR or increasingly now by using a divide and conquer algorithm [3]. Let us consider our snapshot matrix at each adaptivity step $\mathbf{Y}_h \in \mathbb{R}^{\mathcal{N}\times m}$ with $\mathcal{N} >> m$ (typically), the first step of bidiagonalizing $\mathbf{Y}_h$ can be done in $\mathcal{O}(\mathcal{N}m^2)$ flops [3]. In standard packages, the second step can be completed iteratively on the order of $\mathcal{O}(m^2)$ flops. The first step is the most expensive and dominates the cost, leaving the computational complexity of calculating the SVD on the order of $\mathcal{O}(\mathcal{N}m^2)$. Hence if full SVD of the snapshot matrix is computed at each adaptivity step, then this would lead to a cost of $\mathcal{O}(\mathcal{N}m^2)$ for each update.

The method proposed in [1] provides a much more efficient alternative to the expensive task of computing the SVD at each update. Updating the singular vectors and values using low rank updates as outlined in Algorithm 2.1 reduces the need to ever recompute a full SVD of matrix of size $\mathcal{N} \times m$. The only step in this algorithm which requires the computation of a full SVD is for the matrix $\mathbf{K} \in \mathbb{R}^{(n+1)\times(n+1)}$ which costs $\mathcal{O}(n^3)$. The other most expensive steps are doing the rotations $\begin{bmatrix}\mathbf{V}_{h-1} & \mathbf{P}\end{bmatrix}\mathbf{V}'$ and $\begin{bmatrix}\mathbf{W}_{h-1} & \mathbf{Q}\end{bmatrix}\mathbf{W}'$ which entails a cost of $\mathcal{O}(\mathcal{N}n^2)$ and $O(mn^2)$ respectively. This may not seem very advantageous compared the the approach of taking the full SVD which had a complexity of $\mathcal{O}(\mathcal{N}m^2)$, but recall that $\mathcal{N} >> n$ and $m >> n$, where $\mathcal{N}$ is typically on the order of millions (consider a 3-D discretization for example), whereas m is almost always on the order of hundreds.

Directly updating the reduced operators in step 2.21 has a cost of $\mathcal{O}(\mathcal{N}^2 n l_A)$, which is unavoidable if computing the full SVD at each step. While when using the Rank-1 SVD update algorithm 2.1, we have the advantage of writing our update POD basis as $\mathbf{V}_h = \mathbf{V}_{h-1}\mathbf{V}' + \mathbf{c}\mathbf{d}^T$ (2.23) and update our reduced operators following 2.24. The steps involving update to auxiliary matrices 2.26 is linear in $\mathcal{N}$ only in the case when the operator matrices $\mathbf{A}^i, \quad i = 1, \ldots, l_A$ are sparse, which is in general the case with physical models. Hence overall we are able to keep the time complexity of each adaptivity step as $\mathcal{O}(\mathcal{N}n^2 l_A)$, which is linear in $\mathcal{N}$, while computing full SVD at every step would be quadratic in $\mathcal{N}$.

4. Conclusions. In this project we investigated and implemented an algorithm for efficiently updating reduced order models when new data gets available corresponding to some new region of the parameter space. This was performed by generalizing new data as rank-1 updates to a snapshot matrix and computing the fast rank-1 update to its SVD. Hence at every adaptivity step, we were able to efficiently update our POD basis, as well as reduced operators, and maintain an accurate reduced order model (ROM) as it is changed with the evolving parameter region. By doing a time complexity analysis, we proved that this method would significantly decrease runtime when adapting a ROM to the new region in the online phase, while ultimately converging in accuracy to the true ROM. We considered the effects of implementation styles using CGS vs MGS and confirmed that the POD basis lacks orthogonality in the CGS case. This method allows dynamical systems to be captured in real time,

whereas in general the computational burden of building a new reduced order model from scratch renders this process infeasible. An important application of this work is in the field of systems with changing unobservable parameters. This application requires an additional additive update to reduced operators that remains as a future extension of this project.

Acknowledgments. We would like to acknowledge the thought provoking discussions we had with Jing Lin and Arkopal Dutt of the MSEAS lab as well as Boris Kramer of the acdl.

REFERENCES

[1] M. Brand, *Fast low-rank modifications of the thin singular value decomposition*, Linear algebra and its applications, 415 (2006), pp. 20–30.
[2] B. Peherstorfer and K. Willcox, *Dynamic data-driven reduced-order models*, Computer Methods in Applied Mechanics and Engineering, 291 (2015), pp. 21–41.
[3] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, vol. 50, Siam, 1997.