An Implementation-Based Exploration of HAPOD: Hierarchical Approximate Proper Orthogonal Decomposition

Benjamin J. Beach

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mathematics

Serkan Gugercin, Chair
Jeffrey T Borggaard
Mark Embree

December 20, 2017
Blacksburg, Virginia

Keywords: Reduced Order Modeling, Proper Orthogonal Decomposition, Parallel Computing, Mine Fire Dynamics

An Implementation-Based Exploration of HAPOD: Hierarchical Approximate Proper Orthogonal Decomposition

Benjamin J. Beach

(ABSTRACT)

Proper Orthogonal Decomposition (POD), combined with the Method of Snapshots and Galerkin projection, is a popular method for the model order reduction of nonlinear PDEs. The POD requires the left singular vectors from the singular value decomposition (SVD) of an $n \times m$ "snapshot matrix" S, each column of which represents the computed state of the system at a given time. However, the direct computation of this decomposition can be computationally expensive, particularly for snapshot matrices that are too large to fit in memory. Hierarchical Approximate POD (HAPOD) [1] is a recent method for the approximate truncated SVD that requires only a single pass over S, is easily parallelizable, and can be computationally cheaper than direct SVD, all while guaranteeing the requested accuracy for the resulting basis. This method processes the columns of S in blocks based on a predefined rooted tree of processors, concatenating the outputs from each stage to form the inputs for the next. However, depending on the selected parameter values and the properties of S, the performance of HAPOD may be no better than that of direct SVD. In this work, we numerically explore the parameter values and snapshot matrix properties for which HAPOD is computationally advantageous over the full SVD and compare its performance to that of a parallelized incremental SVD method [2–4]. In particular, in addition to the two major processor tree structures detailed in the initial publication of HAPOD [1], we explore the viability of a new structure designed with an MPI implementation in mind.

This work received support from the National Institute for Occupational Safety and Health under contract 200-2014-59669.

(GENERAL AUDIENCE ABSTRACT)

Singular Value Decomposition (SVD) provides a way to represent numeric data that breaks the data up into its most important components, as well as measuring how significant each part is. This decomposition is widely used to assist in finding patterns in data and making decisions accordingly, or to obtain simple, yet accurate, representations of complex physical processes. Examples of useful data to decompose include the velocity of water flowing past an obstacle in a river, a large collection of images, or user ratings for a large number of movies. However, computing the SVD directly can be computationally expensive, and usually requires repeated access to the entire dataset. As these data sets can be very large, up to hundreds of gigabytes or even several terabytes, storing all of the data in memory at once may be infeasible. Thus, repeated access to the entire dataset requires that the files be read repeatedly from the hard disk, which can make the required computations exceptionally slow. Fortunately, for many applications, only the most important parts of the data are needed, and the rest can be discarded. As a result, several methods have surfaced that can pick out the most important parts of the data while accessing the original data only once, piece by piece, and can be much faster than computing the SVD directly. In addition, the recent bottleneck in individual computer processor speeds has motivated a need for methods that can efficiently run on a large number of processors in parallel. Hierarchical Approximate POD (HAPOD) [1] is a recently-developed method that can efficiently pick out the most important parts of the data while only accessing the original data once, and which is very easy to run in parallel. However, depending on a user-defined algorithm parameter (weight), HAPOD may return more information than is needed to satisfy the requested accuracy, which determines how much data can be discarded. It turns out that the input weights that result in less extra data also result in slower computations and the eventual need for more data to be stored in memory at once. This thesis explores how to choose this input weight to best balance the amount of extra information used with the speed of the method, and also explores how the properties of the data, such as the size of the data or the distribution of levels of significance of each part, impact the effectiveness of HAPOD.

Acknowledgments

I am deeply grateful to my advisors Drs. Serkan Gugercin and Jeff Borggaard for their feedback, patience, and continuous support in the development of this work, particularly in response to mistakes and delays along the way. Their guidance has been integral in both this work and in my development as a computational mathematician.

I would like to thank the third member of my committee, Dr. Mark Embree, for his valuable comments and feedback on this work, and for his ever-friendly, thoughtful, and enthusiastic manner which helped kindle my enthusiasm for computational mathematics.

I would like to thank Dr. Alan Lattimer and the NIOSH mine fire research group at Virginia Tech for providing the simulation data that proved integral to this work.

I would like to thank the National Institute for Occupational Safety and Health for their financial support, and the Department of Mathematics at Virginia Tech for their financial support.

Finally, I am grateful to my colleagues and office-mates at Virginia Tech for their friendship and support along the way. I am grateful to my family, whose continuous love and support helped me persevere through several trying times along the way.

Contents

List of Figures

Chapter 1

Introduction

The problem of finding the Singular Value Decomposition (SVD) of a matrix $A \in \mathbb{R}^{n\times m}$ is applicable to a myriad of applications, including machine learning, computer vision, image processing, reduced-order modeling of dynamical systems, and audio feature extraction, to name a few. This SVD is given by

$$A = U\Sigma V^T, \tag{1.1}$$

where $U \in \mathbb{R}^{n\times m}$ and $V \in \mathbb{R}^{m\times m}$ have orthonormal columns, and $\Sigma \in \mathbb{R}^{m\times m}$ is a diagonal matrix with nonnegative entries on the diagonal, in decreasing order. The diagonal entries of Σ are known as *singular values*. A truncated form of this decomposition is widely used to optimally represent the most significant information in the matrix. This truncation is performed by first discarding all but the maximum r singular values, along with the corresponding columns of U and V, to form Σ_r, U_r and V_r. In this setting, U contains dominant columnwise mode information and V determines the distribution of those modes to the columns A. The optimality of the SVD manifests in several forms, such as those given below [5–8]:

Let M_r be the set of $n \times m$ matrices with rank $\leq r$. Then:

$$\min_{A_r\in M_r} \|A - A_r\|_2 = \|A - U_r\Sigma_r V_r^T\|_2 = \sigma_{r+1} \tag{1.2}$$

$$\min_{A_r\in M_r} \|A - A_r\|_F^2 = \|A - U_r\Sigma_r V_r^T\|_F^2 = \sum_{i=r+1}^{m} \sigma_i^2 \tag{1.3}$$

These optimality conditions of the SVD are especially relevant for a model reduction framework known as Proper Orthogonal Decomposition (POD), combined with Galerkin projection.

1.1 The Need for Model Reduction

Dynamical systems arise in a large number of complex physical applications that are of interest in scientific and industrial settings, including fluid flow in a pipe or past an obstructing object, mine fires and wildfires, and vibration suppression in large structures, to name a few. Ever-increasing demands for accuracy in large-scale systems yield a requirement of very high-fidelity system representations, which can be prohibitively expensive to simulate, particularly if one desires to control the system in real-time, simulate the system over a large time horizon, or explore the behavior of the system over a large parameter space. One of the chief methods to address this challenge is model order reduction, which can provide a low-order representation that accurately represents the full complexity of the dynamical system.

Various methods have been invented to compute such reduced order representations for linear [9–15], bilinear [16–19], realization-independent [14, 20–23], special nonlinear [14, 24, 25], and stochastic [16, 26] systems. However, Proper Orthogonal Decomposition (POD) [27–32] remains the method of choice for general nonlinear systems. This method computes a basis for the reduced order model (ROM) for the system from the left singular vectors of an $n \times m$ "snapshot matrix" S, each column of which contains the measured (or computed) discretized state of the system at a given time. This basis is optimal with respect to the representation of the dominant structures in the snapshot matrix, in the sense of (1.3).

1.2 Hardware Considerations

The computation of POD basis vectors for a dynamical system requires a truncated SVD of the snapshot matrix $S \in \mathbb{R}^{n \times m}$, which can represent millions of degrees of freedom n over (tens of) thousands of snapshots m. The direct computation of the singular value decomposition of such a matrix can be very expensive computationally (roughly order $O(m^2 n)$ for $n \gg m$), and usually requires repeated access to the entire snapshot matrix. The issues are compounded since the snapshot matrices, which are typically dense, can be too large to fit in memory, resulting in a significantly increased computational burden. For example, a snapshot matrix of dimension five million by fifty thousand stored in double precision would fill two terabytes of data, which is simply too large to store in memory on most computational systems. The result is an enormous computational cost when repeated access to the full matrix is required. These challenges, which also apply to many other applications that require the SVD of large datasets, have motivated the development of a number of methods that can compute the *truncated* SVD of a matrix with minimal memory requirements.

In addition, due to the recent barrier on individual processor speeds inherent in silicon-based integrated circuits, the bulk of available computing power increasingly relies on the ability to compute in parallel over a massive number of processors, resulting in an increasing need

for scalable parallel algorithms to perform major operations such as a truncated SVD.

Domain Decomposition [33], discussed in Section 2.2.1, provides an iterative parallelized framework for this SVD computation. In this method, the total memory burden is not reduced, but is distributed over the available processors. Randomized SVD methods [34,35], discussed in section 2.2.2, compute an approximate truncated SVD via the application of S to a sampling of random vectors to obtain an approximate basis for the range of S, from which a remarkably cheap approximate truncated SVD for S can be obtained. However, such methods result in a rank-$(r+p)$ basis with a lower accuracy guarantee (in the sense of (1.2) and (1.3)) than the optimal rank-r basis, where r is the target rank and p is an oversampling parameter. Incremental SVD [2, 3], discussed in Section 2.2.3, provides a framework that reduces both the time and memory complexity of the problem, requiring only a single pass over the original matrix. This method is not inherently parallel in nature (though parallelization can still be achieved via individual matrix operations, as in [4]). Finally, HAPOD [1], the topic of this work, provides a highly general, easily parallelizable framework for the approximation of the dominant r scaled *left* singular vectors $SV_r \approx U_r\Sigma_r$ that can also minimize memory usage, still requiring only a single pass over the original matrix. The accuracy of resulting basis (in the sense of (1.3)) is no worse than the optimal accuracy given the specified tolerance, though the use of extra modes may be required. Better yet, HAPOD can be combined with any existing method for computing the full or truncated SVD up to a specified tolerance, as long as the truncation threshold is represented in the same way. Note that, if the corresponding right singular vectors are needed, they can be computed as $V_r = (\Sigma_r^{-1}U_r^T S)^T$. As a bonus, incremental SVD and HAPOD do not require *a priori* knowledge of the order of the truncated basis.

1.3 Thesis Outline

In this work, we explore the effects of user input parameters and matrix properties on the accuracy and computational cost of HAPOD. The input parameters explored include the tolerance, processor structure, and global weight, and the matrix properties explored include its structure, size, and decay of singular values. The accuracy of the approximation is measured in terms of the number of extra output modes. The purpose of this exploration is to aid in the decision of what parameter values should be used for HAPOD, as well as to improve the understanding of when HAPOD provides benefits over direct SVD and over competing truncated SVD paradigms (specifically incremental SVD). In Chapter 2, some of the primary methods for truncated SVD, particularly in parallel, are described. In Chapter 3, HAPOD is described in more detail, especially with regard to the global HAPOD weight, tolerances, and processor tree structures. Finally, in Chapter 4, a numerical study of HAPOD is presented, including tests on accuracy (Section 4.2) using randomized synthetic data, as well as benchmarking tests (Section 4.3) using randomized (Section 4.3.1) and simulation (Section 4.3.2) data. The simulated data arises from the time-integration of a 2-D full-order

model of a system representing strongly heated flow, a step towards the simulation of mine fires. Section 4.3.2 includes a study of the effectiveness of HAPOD for simulated data, a study of the computational benefits gained from the structure of the matrix, and a comparison of HAPOD with a C++ implementation of Brand's incremental SVD method [2, 4].

Chapter 2

Projection-Based Nonlinear Model Reduction

2.1 Proper Orthogonal Decomposition

Proper Orthogonal Decomposition is the method of choice for the model order reduction of general nonlinear systems of ODEs. Consider a semi-discretized nonlinear system of the form:

$$\begin{cases} \dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) + \mathbf{f}(t, \mathbf{x}(t)) \\ \mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t) + \mathbf{g}(t, \mathbf{x}(t)) \end{cases} \tag{2.1}$$

where, if n is the number of states, p is the number of inputs and q is the number of outputs: $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times p}$, $C \in \mathbb{R}^{q\times n}$, $D \in \mathbb{R}^{q\times p}$, $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$, and $g : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^q$. To form a reduced order model (ROM) of such a system using POD, one begins by first simulating the full-order system in time to form a matrix of snapshots $S = [\mathbf{x}(t_1)\ \mathbf{x}(t_2)\ \cdots\ \mathbf{x}(t_m)]$, where each column represents the state of the system at time $t_i \in \mathbb{R}$. The POD basis vectors are then taken to be the first r left singular vectors of S. That is, suppose the SVD of S is given by $S = U\Sigma V^T$, where $U = [\mathbf{u}_1\ \mathbf{u}_2\ \cdots\ \mathbf{u}_m]$ with $\mathbf{u}_i \in \mathbb{R}^n$, S is a diagonal matrix with positive diagonal entries in decreasing order, and $V \in \mathbb{R}^{m\times m}$. Then the POD basis is given by $U_r = [\mathbf{u}_1\ \mathbf{u}_2\ \cdots\ \mathbf{u}_r]$, yielding the reduced order-model

$$\begin{cases} \dot{\mathbf{x}}_r(t) = A_r\mathbf{x}_r(t) + B_r\mathbf{u}(t) + U_r^T\mathbf{f}(t, U_r\mathbf{x}_r(t)) \\ \mathbf{y_r}(t) = C_r\mathbf{x}_r(t) + D_r\mathbf{u}(t) + \mathbf{g}(t, U_r\mathbf{x}_r(t)) \end{cases} \tag{2.2}$$

where

$$\mathbf{x} \approx U_r \mathbf{x}_r,\ A_r = U_r^T A U_r,\ B_r = U_r^T B,\ C_r = C U_r,\ D = D_r,\ \text{and}\ y \approx y_r. \tag{2.3}$$

Note that the simulation of the system in this form still requires projecting $\mathbf{x}_r$ back to the full space, then performing full-order evaluations of $\mathbf{f}$ and $\mathbf{g}$. Thus, a reduced-order approximation of $\mathbf{f}$ and $\mathbf{g}$ is also needed to produce a truly reduced-order model. This approximation is commonly computed using DEIM [28, 36] to obtain the reduced functions $\mathbf{f_r}$ and $\mathbf{g_r}$ that are of lower order in both input and output.

2.2 Parallel Implementations of Truncated SVD

2.2.1 Domain Decomposition

One option for obtaining an estimate for the left singular vectors of the snapshot matrix is a domain decomposition approach, as described in [33]. In this approach, the problem's spatial domain Ω is first decomposed in N_p subdomains Ω_i, where N_p is the number of available processors; (note that the subdomains may be allowed to have some overlap.) Each corresponding section of the snapshot matrix is then stored on a separate processor, and the first q right singular vectors $V_k^{(1)}$ are computed (where $k \in \{1, 2, ..., N_p\}$ is the processor index). The right singular vectors from all processors are then concatenated horizontally to form $\hat{V}^{(1)} = [V_1^{(1)}\ V_2^{(1)}\ \cdots\ V_{N_p}^{(1)}]$, and the dominant q right singular vectors of $\hat{V}^{(1)}$ are computed to form $\tilde{V}^{(1)}$. Finally, a filtered subspace iteration is performed on W and $\tilde{V}^{(j)}$ to obtain the left singular vectors of W. This algorithm is given below.

Domain Decomposition with Filtered Subspace Iteration

1. $[U_k, S_k, V_k^T] = svd(W_k)$
2. $V_k^{(1)} = V_k(:, 1:q)$
3. $\hat{V}^{(1)} = [V_1^{(1)}\ V_2^{(1)}\ \cdots\ V_{N_p}^{(1)}]$
4. $[J, M, \tilde{V}^{(1)}] = svd(\hat{V}^{(1)})$
5. $\tilde{V}^{(1)} = \tilde{V}^{(1)}(:, 1:q)$
6. for j=1:Jmax
   (a) $W^{(j)} = W\tilde{V}^{(j)}$
   (b) Compute the q dominant *left* singular vectors of $W^{(j)}$: $U_1^{(j)}$
   (c) $C^{(j)} = (U_1^{(j)})^T W^{(j)}$

(d) Compute the q dominant *right* singular vectors of $C^{(j)}$: $\tilde{V}^{(j+1)}$

end

If the iteration succeeds, $U^{(j)}$ converges to the dominant q left singular vectors of W, $C^{(j)}$ converges to the dominant q singular values, and $V^{(j)}$ converges to the dominant q right singular vectors of W. The computation of $W^{(j)}$ can be parallelized using the eigenvalue decomposition of $(W\tilde{V})^T W\tilde{V}$ locally on each processor as described in [33]. However, this algorithm requires *a priori* knowledge of the desired reduced order of the matrix, a choice which necessitates some knowledge of the decay of singular values. As discussed in [33], the convergence rate of this algorithm depends on both the decay of singular values and on the existence of similar dominant temporal structures throughout the domain.

2.2.2 RandSVD

The basic algorithm for random SVD (randSVD) computes an approximate rank-$(r+p)$ truncated SVD of a matrix $S \in \mathbb{R}^{n\times m}$, where r is the target rank and p is an oversampling parameter [35]. This is accomplished by first multiplying the original matrix by a set of $r+p$ randomized vectors $\{\mathbf{x}_i\}|_{i=1}^{r+p}$ to obtain an approximate basis $\{\mathbf{y}_i\}|_{i=1}^{r+p}$ for the range of A. An orthogonal basis U_r for this approximate range is then computed, and used to obtain the truncated SVD. For a relatively small value of p (on the order of $p=5$ or $p=10$), this method can with very high probability attain an error bound of

$$\|S - U_r U_r^T S\|_2 = \|S - U_r \Sigma_r V_r^T\|_2 \leq \left[1 + 11\sqrt{r+p}\sqrt{\min\{m,n\}}\right] \sigma_{r+1}. \tag{2.4}$$

Such a method can achieve a computational complexity as small as $O(mn\log(r)+(m+n)r^2)$, which is attainable via the use of a matrix $\Omega = [x_1\ x_2\ \cdots\ x_{r+p}]$ with special internal structure, allowing for rapid evaluation of the product $S\Omega$ (as described in [35]). However, as discussed in [35], more substantial oversampling may be required to obtain a higher degree of accuracy, depending on the size and decay of singular values of the S and on the structure of the random sampling matrix Ω. In particular, the matrix structures that allow computationally efficient computation of $S\Omega$ also tend to require far more substantial oversampling. There exist a large number of variations on this algorithm which use additional steps to reduce this error bound further, as presented in [34, 35] and many other references found therein. In particular, [35] gives a thorough discussion of several existing methods for randSVD, many of which are easily parallelizable, including error bounds and analysis of computational complexity. In exchange for this computational efficiency, these methods include the need for p extra basis modes to achieve error bounds that are slightly sub-optimal (in the sense of both (1.2) and (1.3)) for even the rank-r case. In addition, as explained in [35], more accurate approximations, particularly for matrices with a slow decay of singular values, can require several matrix multiplication of the form $(AA^T)^q A$, necessitating additional passes over the data. This

multiplication can also incur significant communication overhead if parallelized, since the quantity $(A\Omega)$ would typically be distributed across the processors after its computation, but would be needed by all processors participating in the matrix multiplication. Also, note that this method requires *a priori* knowledge of the target rank r.

2.2.3 Incremental SVD

A number of methods for incremental low-rank SVD have been developed, such as those described in [2,3,32,37–43]. These algorithms have been developed with specific applications to image analysis [37,38,40], model reduction via proper orthogonal decomposition [32,42], computer vision and audio feature extraction [2], and even modeling of consumer movie ratings [3]. In this class of frameworks, the snapshot matrix is processed one column at a time (or in some cases, several, as in [42]), and the basis is updated with each additional column until all columns have been incorporated. This type of algorithm achieves a computational time-complexity as good as $O(mnr)$ and a memory complexity of $O((m+n)r)$, where n is the dimension of the problem, m is the number of snapshots taken, and r is the number of columns in the resulting basis. The information returned varies by algorithm (some may return only the left or only the right singular vectors, for example).

Of particular interest is the method proposed by Matthew Brand [2,3], for which both the left and right singular vectors are obtained. A notable property of this framework is that it can account for noise or for missing values in the incoming data. In this method, the matrix U of left singular vectors is decomposed as $U_r^{(j)} = Q^{(j)}Q_r^{(j)}$ at each stage of the computation, where $Q^{(j)} \in \mathbb{R}^{n\times r}$ and $Q_r^{(j)} \in \mathbb{R}^{r\times r}$. This decomposition allows the desired computational efficiency, while also yielding improved numerical orthogonality of the basis $U_r^{(j)}$. With this decomposition, the algorithm can be given as follows. Given a snapshot matrix $S = [\mathbf{s}_1\ \mathbf{s}_2\ \cdots\ \mathbf{s}_m]$ and a target linearity tolerance ε:

Incremental SVD

1. $\sigma^{(1)} = \|\mathbf{s_1}\|_2$
2. $W^{(1)} = \dfrac{\mathbf{s_1}}{\|\mathbf{s_1}\|_2}$
3. $W_r^{(1)} = [1]$
4. $r = 1$
5. for j=2:m
   (a) $\mathbf{l} = (Q_r^{(j)})^T(Q^{(j)})^T\mathbf{s}_j$
   (b) $\mathbf{p} = \|\mathbf{s}_j - Q^{(j)}Q_r^{(j)}\mathbf{l}\|_2$

(c) $k = \|\mathbf{p}\|_2$

(d) $\mathbf{q} = \mathbf{p}/k$

(e) $Q = \begin{bmatrix} diag(\sigma^{(j)}) & \mathbf{l} \\ 0 & k \end{bmatrix}$

(f) $[U',\ \sigma',\ V'] = SVD(Q)$

(g) If $k < \epsilon$, then $\mathbf{s}_i$ is well-represented by the columns of $U_r^{(j)}$, and the system is updated as

i. $W_r^{(j+1)} = Q_r^{(j)} U'_{1:r,1:r}$

ii. $\sigma^{(i+1)} = \sigma'_{1:r}$

iii. Reorthogonalize $Q_r^{(j)}$ if necessary

(h) Otherwise, the system is updated as

i. $W^{(j+1)} = \begin{bmatrix} Q^{(j)} & \mathbf{q} \end{bmatrix}$

ii. $W_r^{(j+1)} = \begin{bmatrix} W^{(j_r)} & 0 \\ 0 & 1 \end{bmatrix} U'$

iii. $\sigma^{(i+1)} = \sigma'$

iv. $r = r + 1$

v. Reorthogonalize $W^{(j+1)}$ and $W_r^{(j+1)}$ if necessary

end

6. $U_r = W^{(m)} W_r^{(m)}$

7. $\Sigma_r = diag(\sigma^{(m)})$

This algorithm closely follows the one summarized in [4], except that the computation of U_r at each step of the simulation has been circumvented to reach the desired computational complexity, and the notation has been modified somewhat. Note that additional steps are added in [2] to account for noise and for missing values in the data; these steps are performed at each step in the computation, and can be performed in such a way that the computational complexity is unaffected. Note that the decomposition $U_r^{(j)} = Q^{(j)} Q_r^{(j)}$ allows an extremely cheap update (merely concatenation) for $Q^{(j)}$, yielding the desired computational complexity of $O(mnr + mr^3)$. Without this decomposition, the direct computation $U_r^{(j+1)} = U_r^{(j)} U'$ is required at each stage of the simulation, incurring a computational cost of $O(mnr^2)$.

In addition, this incremental algorithm has been parallelized and implemented in C++ by William Arrighi, Geoffrey Oxberry, Tanya Vassilevska, and Kyle Chand at Lawrence Livermore National Lab [4]. This code parallelizes the individual linear algebra operations on the matrix U during its construction, so that the rows of U are distributed among the available processors throughout the process. Brand's algorithm is implemented both with and without the decomposition $U_r^{(j)} = Q^{(j)} Q_r^{(j)}$. The idea behind this decomposition was

that the decomposed version may result in a less pronounced numerical loss of orthogonality, reducing the need for re-orthogonalization of the matrix U. Additionally, the decomposition also allows the computation of the full-order basis U to be deferred, with a very cheap update of Q_1, potentially resulting in significant computational savings (as demonstrated in Section 4.3.2) and the desired complexity of $O(mnr + mr^3)$ [2].

2.2.4 HAPOD

HAPOD is a hierarchical paradigm for computing the basis vectors for proper orthogonal decomposition. This method requires a rooted tree of processors and an existing algorithm for computing the POD bases via truncated SVD. The inputs to both HAPOD and the truncated SVD algorithm should be the same: a target tolerance ε and an $m \times n$ snapshot matrix S. If $S = U\Sigma V^T$ is the singular value decomposition of S, the outputs to both methods are approximations to either U_r and Σ_r or simply $U_r\Sigma_r$, where U_r contains the first r columns of U and Σ_r is the upper-left $r \times r$ block of Σ.

HAPOD operates by first splitting the snapshot matrix S into k column-wise slices $S = [S_1\ S_2\ \cdots\ S_k]$. Each slice is then transferred to a leaf of the rooted tree, where POD bases are computed for the slice using an input tolerance based on: the total tree depth, the target tolerance, the number of columns in each slice, and a globally chosen weight ω. The resulting bases $U_i\sigma_i$ of sibling nodes are then concatenated, and another truncated SVD step is then performed on the parent node. This process is repeated until a final SVD step is performed on the root node. With carefully chosen input tolerances, Himpe et al. [1] have shown that the bases returned by HAPOD will result in no less accuracy (with respect to the mean projection error (defined in Section 3.1) than the optimal accuracy achievable via direct truncated SVD, though extra modes may be required to reach this level of accuracy. The controls the balance between number of extra required modes and the computational efficiency of HAPOD by adjusting the SVD tolerances used for the intermediate and final truncated SVD steps.

The chief advantages of HAPOD include reduced memory requirements, simple parallelization, generality, and a potential reduction in computational complexity. The computational advantages of HAPOD rely primarily on the elimination of modes in the intermediate stages of the computation. The reduced memory requirements stem from the fact that once a slice of S has been read into memory for an SVD step, the data will no longer be needed for any future SVD steps, and can thus be discarded to make room for the next slice. The easy parallelizeability is a consequence of the underlying tree structure with which HAPOD is built. The reduction in computational complexity is dependent on the tree structure and on assumptions on the behavior of the Kolmogorov N-width of the snapshot matrix S as its width increases, as mentioned in [1].

Chapter 3

HAPOD Strategies

3.1 The Global Weight

The HAPOD weight is used as a factor in choosing the SVD cutoff tolerances used in the intermediate and final stages of the SVD. These weights were formulated to simultaneously control the accuracy resulting from HAPOD and the number of output modes. This formulation, which is detailed below, closely mirrors the formulation given in [1], but with modified notation.

First, we define the truncated SVD function as follows.

Definition 3.1. *Let $S \in \mathbb{R}^{n\times m}$ be an input matrix $S = [\mathbf{s}_1\ \mathbf{s}_2\ \cdots\ \mathbf{s}_m]$ and let $\varepsilon > 0 \in \mathbb{R}$ be a target tolerance. Let $U\Sigma V^T$ be the SVD of S, where $U = [\mathbf{u}_1\ \mathbf{u}_2\ \ldots\ \mathbf{u}_m]$ and the ith diagonal element of Σ is given by σ_i. Define the truncation order r as:*

$$r = \min\left\{ r \in \{0, 1, 2, \ldots, m\} \middle| \sum_{i=r+1}^{m} \sigma_i^2 \leq \varepsilon^2 \right\}. \tag{3.1}$$

Then the function TSVD is defined, for a matrix S and a cutoff tolerance ε, to be the scaled left singular vectors of the truncated SVD of S:

$$TSVD(S, \varepsilon) = (U_r \Sigma_r), \tag{3.2}$$

where $U_r = [\mathbf{u}_1\ \mathbf{u}_2\ \cdots\ \mathbf{u}_r]$ and Σ_r is the $r \times r$ diagonal matrix with diagonal elements $[\sigma_1\ \sigma_2\ \cdots\ \sigma_r]$. Finally, for the sake of convenience, let $|TSVD(S, \varepsilon)|$ represent the number of modes output by $TSVD(S, \varepsilon)$.

To aid with the notation for the definition of HAPOD, we will define a concatenation function for a set of column slices as follows.

Definition 3.2. *Let $\tilde{S}$ be an ordered set of matrices $\tilde{S} = \bigcup_{i=1}^{M} \tilde{S}_i$, where $\tilde{S}_i \in \mathbb{R}^{n \times m_i}$. Then let the concatenation function $F\left(\tilde{S}\right)$ be defined as*

$$F\left(\tilde{S}\right) := \left[\tilde{S}_1 \ \tilde{S}_2 \ \cdots \ \tilde{S}_M\right]. \tag{3.3}$$

To further aid with the notation for the definition of HAPOD, we introduce the following notation for the underlying rooted tree.

Definition 3.3. *Let τ be a rooted tree of processors with root node ρ and a set of leaf nodes $L := \{\ell_1, \ell_2, \cdots, \ell_{n_L}\}$. Then define the following sets for each node α of the tree τ:*

1. *N_α is the set of descendants of α, including α itself*
2. *$C_\alpha = \{C_1, C_2, \ldots, C_{N_c}\}$ is the set of immediate children of α*
3. *$L_\alpha := L \cup N_\alpha$ is the set of leaf nodes with ancestor α (so that $L_\ell = \{\ell\}$ for a leaf node $\ell \in L$).*

With this notation, we formally define HAPOD as follows.

Definition 3.4. *Let S, τ, L, N_α, and C_α be defined as in Definitions 3.1 and 3.3. Furthermore, let the columns of S be partitioned as $\bigcup_{\ell \in L} S_\ell$, where the elements S_ℓ are pairwise disjoint, so that each column of S belongs to exactly one slice S_ℓ and leaf processor ℓ. Finally, let ε_τ be a function defined by $\varepsilon_\tau(\alpha) = \varepsilon_\alpha$, where ε_α is the truncated SVD tolerance to use on node α. Then the function $HAPOD$ is defined recursively as follows:*

$$HAPOD(S, \varepsilon_\tau, \alpha) := \begin{cases} TSVD(S_\alpha, \varepsilon_\tau(\alpha)), & \alpha \in L; \\ TSVD\left(F\left(\bigcup_{\beta \in C_\alpha} HAPOD(S, \varepsilon_\tau, \beta)\right), \varepsilon_\tau(\alpha)\right), & otherwise. \end{cases} \tag{3.4}$$

That is, for each non-leaf node, HAPOD returns the truncated SVD of the concatenation of the outputs of its children using tolerance $\varepsilon_\tau(\alpha)$. Finally, the final output of HAPOD is defined as the output of HAPOD for the root node ρ of the underlying tree τ:

$$HAPOD(S, \varepsilon_\tau) := HAPOD(S, \varepsilon_\tau, \rho). \tag{3.5}$$

Finally, for the sake of convenience, let $|HAPOD(S, \varepsilon_\tau, \alpha)|$ represent the number of modes output by $HAPOD(S, \varepsilon_\tau, \alpha)$, and similarly for $HAPOD(S, \varepsilon_\tau)$.

The above definition is preferred here over a more algorithmic definition since it simplifies the expression of the following major theorems proven in [1], which are pivotal in the choice of the global HAPOD weight ω. These theorems provide upper bounds for both the *accuracy* of HAPOD (Theorem 3.5) and the number of modes produced by HAPOD (Theorem 3.6), while establishing a choice of tolerances and the role of the global weight in controlling the balance between the accuracy and efficiency of HAPOD (Theorem 3.7).

Theorem 3.5. *Let S, S_ℓ, τ, L, N_α, C_α, and L_α be defined as in Definitions 3.1, 3.3, and 3.4. Let $\tilde{S}_\alpha$ be defined as the matrix formed by concatenating all snapshots input to descendants of the node α:*

$$\tilde{S}_\alpha := F\left(\bigcup_{\ell \in L} S_\ell\right), \tag{3.6}$$

and let $\tilde{S}_\alpha$ be written as $[\tilde{\mathbf{s}}_1 \ \tilde{\mathbf{s}}_2 \ \cdots \ \tilde{\mathbf{s}}_{M_\alpha}]$. Then the projection error of the HAPOD bases output at node alpha (in the sense of (1.3)) is then bounded by:

$$\sum_{i=1}^{M_\alpha} \|\tilde{\mathbf{s}}_i - U_r U_r^T \tilde{\mathbf{s}}_i\|_2^2 \leq \sum_{\beta \in N_\alpha} \varepsilon_\tau(\beta)^2. \tag{3.7}$$

Theorem 3.6. *With the same notation as in Theorem 3.5, let α be a node of the rooted tree τ. Then we have*

$$|HAPOD(S, \varepsilon_\tau, \alpha)| \leq |TSVD(\tilde{S}_\alpha, \varepsilon_\tau(\alpha))|. \tag{3.8}$$

That is, the number of modes output by HAPOD at each node α is bounded above by the number of modes output by truncated SVD using the same tolerance ε_α.

The above results motivate the following choices for the HAPOD tolerances.

Theorem 3.7. *With the same notation as in Theorem 3.6, let $\omega \in (0, 1)$ be a global weight, and let ε be a user-defined tolerance. Then we define the tolerance at a node α in the tree τ with root node ρ as*

$$\varepsilon_\tau(\alpha) = \begin{cases} \omega \cdot \varepsilon \cdot \sqrt{m}, & \alpha = \rho; \\ \sqrt{\dfrac{M_\alpha}{L-1}} \cdot \varepsilon \cdot \sqrt{1-\omega^2}, & otherwise, \end{cases} \tag{3.9}$$

where M_α is the number of modes of S that have been incorporated so far, as defined in Theorem 3.5. This choice of tolerance yields the following bounds for the projection error and the number of output modes, which follow easily from Theorems 3.5 and 3.6. Let $U_r \Sigma_r = HAPOD(S, \varepsilon_\tau)$. Then

$$\frac{1}{m} \sum_{i=1}^{m} \left\| U_r U_r^T \mathbf{s}_i - \mathbf{s}_i \right\|_2^2 \leq \varepsilon^2 \ \text{ and } |HAPOD(S, \varepsilon_\tau)| \leq |TSVD(S, \omega \cdot \varepsilon \cdot \sqrt{m})|. \tag{3.10}$$

Moreover, at each intermediate node α, we have:

$$|HAPOD(S, \varepsilon_\tau)| \leq \left| TSVD\left(\tilde{S}_\alpha, (L-1)^{-1/2} \cdot \sqrt{1-\omega^2} \cdot \varepsilon\right)\right|. \tag{3.11}$$

Note that these tolerances as given yield an approximation to TSVD$(S, \sqrt{m} \cdot \varepsilon)$. This choice was made in [1] in order to use a tolerance measure independent of the total number of input modes m, in order to both eliminate the dependence of the meaning of ε on the number of input modes and to yield a method for which the total number of modes to incorporate need not be known a priori (before the final step of truncated SVD). Note also that by the above result, HAPOD yields a projection basis that is at least as accurate as the basis computed by TSVD$(S, \sqrt{m} \cdot \varepsilon)$. Thus, by the optimality condition (1.3) of truncated SVD, the basis output by HAPOD cannot contain any *fewer* modes than does truncated SVD with tolerance $\sqrt{m} \cdot \varepsilon$. Thus, we obtain the following bounds on the total number of modes output by HAPOD:

$$|\text{TSVD}(S, \varepsilon \cdot \sqrt{m})| \leq |HAPOD(S, \varepsilon_\tau)| \leq |\text{TSVD}(S, \omega \cdot \varepsilon \cdot \sqrt{m})|. \tag{3.12}$$

That is, the number of modes output by HAPOD is bounded by the number of modes output by truncated SVD with a tolerance of $\sqrt{m} \cdot \varepsilon$ and $\sqrt{m} \cdot \omega \cdot \varepsilon$.

From (3.11) and (3.12), the role of the global weight ω becomes clear: a value of $\omega \approx 0$ results in a looser tolerance at the intermediate stages of HAPOD, and thus a cheaper and less memory-intensive computation, at the cost of a larger number of extra modes at the final stage. On the other hand, a value of $\omega \approx 1$ results in very few extra modes at the final stage in exchange for a more costly and memory-intensive computation. The number of extra modes at a given weight is determined by difference between the exact truncation orders of the SVD at the tolerances $\sqrt{m}\omega\varepsilon$ and $\sqrt{m}\varepsilon$. The difference depends strongly on the "slope" of the decay of singular values of the snapshot matrix S, a notion which is explored in more detail in Section 4.2.1.

3.2 Major Tree Structures

Two major rooted tree structures are posited in [1]: the "live" and "distributed" structures (Figure 3.1). The chief advantage of the "live" structure lies in optimized memory savings, since each new slice is integrated into the basis before the next is read in. In addition, this live structure results in a minimized snapshot matrix before the final SVD step, which can greatly reduce the computational cost if the decay of singular values is sufficiently fast. However, computations using this structure can only be performed using a maximum of two MPI

processors, affording little potential for parallelization. Also, this method can become very slow if the matrix allows for very little reduction in the number of modes for the requested tolerance, since the intermediate SVD steps can become quite large. On the other hand, the "distributed" structure allows for maximum parallelization (and thus a cheaper intermediate stage), and the computational cost tends to be comparable to that of the full-order POD step in the worst case (when no modal elimination is achieved). However, the method also tends to result in a much larger final SVD step due to a lack of intermediate steps, which reduces the computational savings and increases the memory cost.

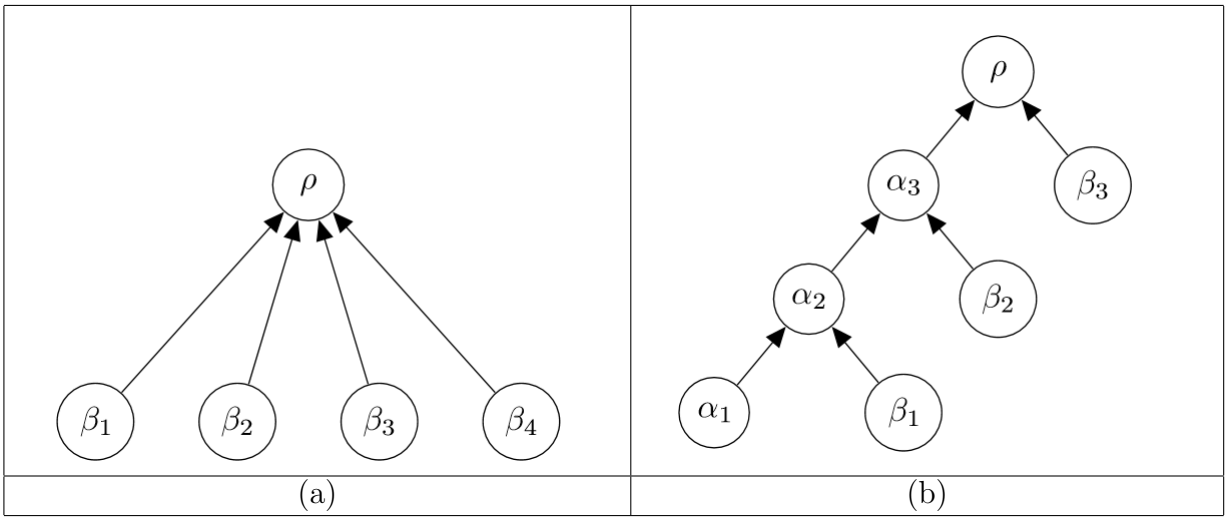


Figure 3.1: Examples of processor hierarchies that can be used as inputs to HAPOD. In each case, ρ is used to denote the root node. The leaves are the β_i nodes, along with α_1. (a) The distributed approximate POD hierarchy. (b) The live approximate POD heirarchy.

The details of these tree structures may depend, for example, on the size of the snapshot matrix or on the available processor structure.

3.3 Hybrid Structure

A new hybrid rooted tree structure is presented in this work in addition to the structures discussed in [1]. This structure, as shown in Figure 3.2, is a combination of the basic structures presented by Himpe et al. [1] conceived with MPI-level parallelization in mind. Each processor is given a subset of the slices of the snapshot matrix, and performs live HAPOD on its set of slices (using a non-root tolerance at the head node). The resulting bases are then combined in one final truncated SVD step to obtain the final POD basis. The aim of this structure is to combine the advantages of the live and distributed structures: taking advantage of the easy parallelization offered by the distributed method while also

leveraging the memory savings afforded by the live method.

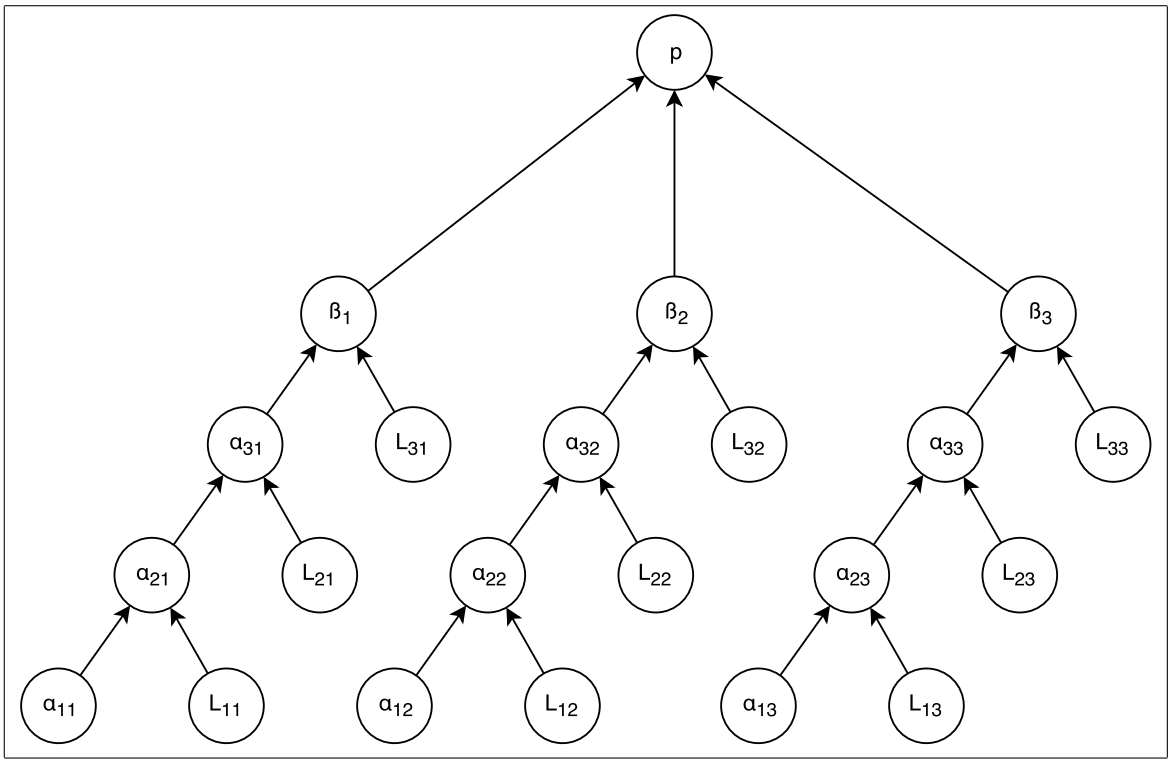


Figure 3.2: The hybrid (or combined) processor heirarchy.

This heirarchy is intended for a hardware architecture in which a number of same-level processors are available for simultaneous use, as may be the case for a single computing node. Here, each major branch with sub-root β_i is owned by a single MPI process, which performs HAPOD using the live structure until all of that processor's snapshots have been incorporated. The results from all processors are then concatenated, and a single distributed step is performed. This choice of structure minimizes communication costs between the MPI processes, since the basis on each processor is reduced as far as possible before it is passed on to the root processor. In this work, four MPI processors are used, and the root node acts as both the β_1 branch and as the root node ρ.

This structure can be generalized for arbitrary physical processor trees, such as the common case where a number of computing nodes are available, each with its own set of processors. In such a case, the basic rooted tree can be given by the available physical processor structure, and live HAPOD can then be performed at each *leaf* of this tree. Such a structure could make maximal use of the hardware architecture while also taking advantage of the memory savings afforded by the live HAPOD structure.

Note that a reversed hybridization is used for one of the numerical examples given in [1], in which each node of a *live* structure is replaced with a copy of the *distributed* structure,

instead of vice-versa. This reversed hybridization allows snapshots to be incorporated as they are generated, but may behave quite differently in terms of both computational and memory cost. For example, if the number of available processors is relatively small, the reversed hybridization is potentially more expensive computationally if the decay of singular values is very slow, since multiple live steps will be required on large matrices (potentially of order $> \frac{m}{2}$). In addition, this method may require more communication between processors, as the local processors must transfer their results to the root node at each stage before incorporating a new set of snapshots, whereas the hybrid method studied in this work requires only one communication step that takes place *after* the number of snapshots owned by each process has been minimized. Conversely, however, this increased risk may correlate with increased potential computational savings for a matrix with a very fast decay of singular values. A comparison of these hybridization methods is deferred to a later work.

Chapter 4

Numerical Exploration/Study

4.1 Description of the Data

The purpose of this work is to investigate the user inputs and matrix properties for which HAPOD begins to gain computational advantages over direct truncated SVD, as well as to shed light on how to choose the input parameters for HAPOD, such as the weight and the size of each sub-matrix. To this end, tests were performed to benchmark the method with various tree structures against direct POD, and to investigate the accuracy and number of extra output modes relative to full POD.

Several parameters were varied in this investigation: The dimensions of the $\mathbf{n} \times \mathbf{m}$ snapshot matrix S, the HAPOD weight $\boldsymbol{\omega}$, the input POD tolerance $\boldsymbol{\varepsilon}$, and the shape of the singular value curve of S. This shape was defined via a single parameter $\mathbf{p}$, along with statically-defined bounds for the singular values. For each test, the columns of S are divided into $\lfloor \sqrt{m} \rfloor$ slices for the application of HAPOD, using each of the three proposed tree structures. The error measurement used for this work was the mean projection error of the columns of S:

$$err = \frac{1}{m} \sum_{i=1}^{m} \|U_r U_r^T \mathbf{s}_i - \mathbf{s}_i\|_{l_2}^2 = \frac{1}{m} \|U_r U_r^T S - S\|_F^2 \tag{4.1}$$

where $U \Sigma V^T$ is the SVD of S, U_r contains the first r columns of U, and $\mathbf{s}_i$ is the ith column of S. For the application of truncated SVD, this error, which is equivalent to the measure in (1.3), is equal to the sum of the squares of the truncated singular values:

$$\|U_r U_r^T S - S\|_F = \sum_{i=r+1}^{m} \sigma_i^2. \tag{4.2}$$

Accordingly, the truncation order r is chosen as the smallest r such that:

$$\frac{1}{m} \sum_{i=r+1}^{m} \sigma_i^2 \leq \varepsilon^2 \tag{4.3}$$

where ε is the desired tolerance. This is the tolerance used in Section 4.3.2. The factor of $\frac{1}{m}$ on this tolerance is meant to make the choice of tolerance independent of the matrix width for a given distribution of singular values, as in Section 3.1. For any application of HAPOD, it is known that the resulting projection error as defined in (4.1) is bounded above simply by ε^2, as discussed in Section 3.1 and proven in [1]. However, note that this tolerance is *absolute*, requiring approximate *a priori* knowledge of the total energy of the system to be meaningful.

It would perhaps be more useful to know the total energy (the sum of squares of the singular values) of the snapshot matrix beforehand, so that the user can opt to keep, for example, 99.9% of the energy of the system. This is easily accomplished in section 4.3.2 with the observation that the square of the Frobenius norm of S is equivalent to the total energy of the snapshot matrix. Thus, such a specification is easily accomplished at only the cost of a single pass over the columns of the snapshot matrix, an operation which is easily parallelized, to compute the total energy

$$E = \sum_{i=1}^{m} \sigma_i^2.$$

This way, if the tolerance $\tilde{\varepsilon}$ represents the percentage of the total energy to truncate, the new tolerance is then computed from this energy as

$$\varepsilon = \frac{E \cdot \tilde{\varepsilon}}{\sqrt{m}}, \tag{4.4}$$

so that the truncation order $\tilde{r}$ is given as the largest $\tilde{r}$ such that

$$\frac{1}{E} \sum_{i=r+1}^{m} \sigma_i^2 \leq \tilde{\varepsilon}^2. \tag{4.5}$$

This relative definition of the tolerance is used to standardize the computation of HAPOD in Section 4.3.2 as the total number of incorporated nodes was varied.

The sample snapshot matrices were created by generating a random matrix of the desired size, computing the SVD of the matrix, replacing the singular values with the desired distribution, then re-assembling the matrix with the new singular values. Each singular value distribution was constructed based on a parameter p using a maximum singular value of 1 and a minimum singular value of $1e-20$. Two types of curvature were defined: "slow-decay" (convex curvature) and "fast-decay" (concave curvature), where convexity is defined based on a "semilogy" chart type (in MATLAB notation). In each case, the logarithm (base 10) of the singular value curvature is defined via a pth-order component-wise polynomial map y from $[\sigma_{min}, \sigma_{max}]^m$ to $[\sigma_{min}, \sigma_{max}]^m$, where in this case, $[\sigma_{min}, \sigma_{max}] = [-20, 0]$. These maps are applied to the uniformly spaced set of m points on $[\sigma_{min}, \sigma_{max}]$, in decreasing order:

$$\mathbf{x} = \{\sigma_{max}, \sigma_{max} - h, \sigma_{max} - 2h, \cdots, \sigma_{min}\} \tag{4.6}$$

where $h = \left(\frac{\delta\sigma}{m-1}\right)$ and $\delta\sigma = \sigma_{max} - \sigma_{min}$). For each component, the "slow-decay" map is defined via a downward-facing pth order polynomial with a vertex of "multiplicity" $p-1$ at the point $(\sigma_{min}, \sigma_{max})$, and passing through the point $(\sigma_{max}, \sigma_{min})$. Conversely, each element of the the "fast-decay" map is defined via an upward-facing pth order polynomial, this time with a vertex of "multiplicity" m at the point $(\sigma_{max}, \sigma_{min})$, and passing through the point $(\sigma_{min}, \sigma_{max})$

Define $\delta\sigma = \sigma_{max} - \sigma_{min}$. Then, for the fast-decay case, the map is defined componentwise as

$$y_i(x_i) = \frac{1}{(\delta\sigma)^{p-1}}(x_i - \sigma_{min})^p + \sigma_{min}, \tag{4.7}$$

while for the slow-decay case, the map is defined componentwise as

$$y_i(x_i) = \frac{1}{(\delta\sigma)^{p-1}}(\sigma_{max} - x_i)^p + \sigma_{max}. \tag{4.8}$$

The singular value distributions for several such maps are shown in Figures 4.1 and 4.2 below. Note that these maps are identical for $p = 1$, and that they generalize nicely for non-integer positive values of p. Note also that the general shape of these distributions will not change with m; only the resolution will change. This fact limits the potential gains of HAPOD in terms of computational complexity, with respect to the number of snapshots: since $r = \alpha m$ for some constant $0 < \alpha < 1$ for each parameter set, the computational complexity of the final SVD step is bounded below at $O(nr^2) = O(\alpha^2 nm^2) = O(nm^2)$. On the other hand, in a physical simulation, the model order reduction ratio $\frac{r}{m}$ may be expected to decrease as m grows large ($r/m = 1/\sqrt{m}$ if $r = \sqrt{m}$, for example), resulting in more substantial gains in

computational complexity. Another significant factor to note is that the singular value directions are randomized; if the snapshot matrices used for testing had actually originated from simulation data, the more slowly-varying temporal structure should allow for more compression in the intermediate stages of HAPOD, and thus more pronounced computational gains given the same singular value profiles.

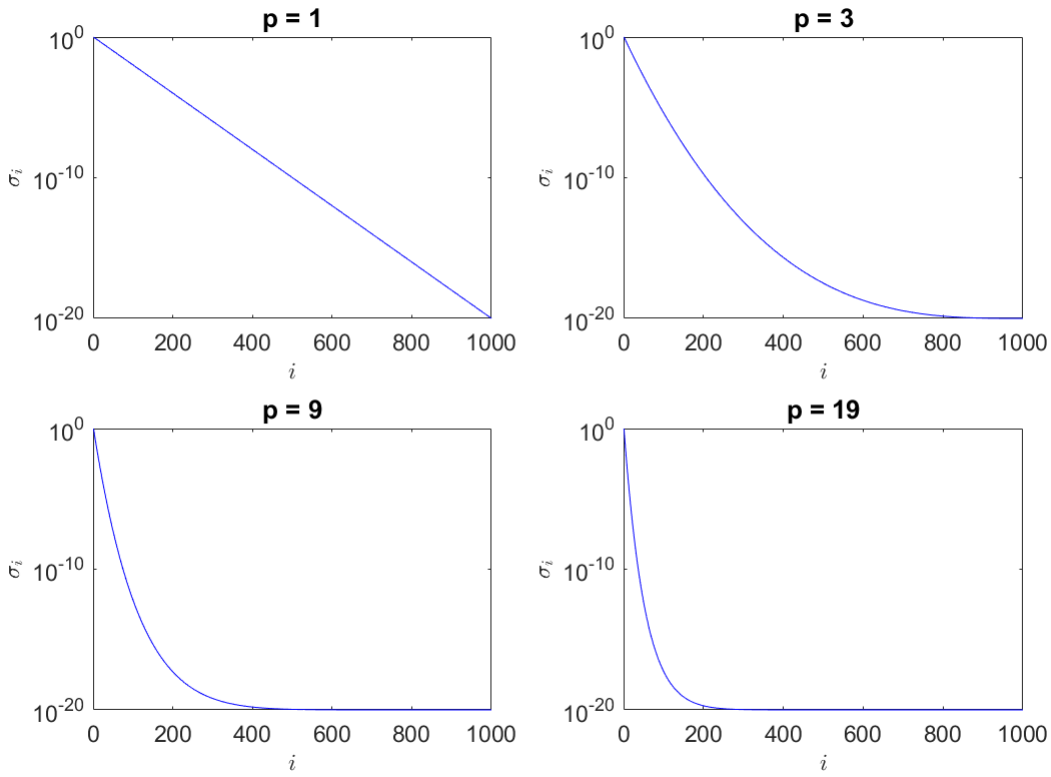


Figure 4.1: Fast-decaying singular value distributions, using $n = 1000$.

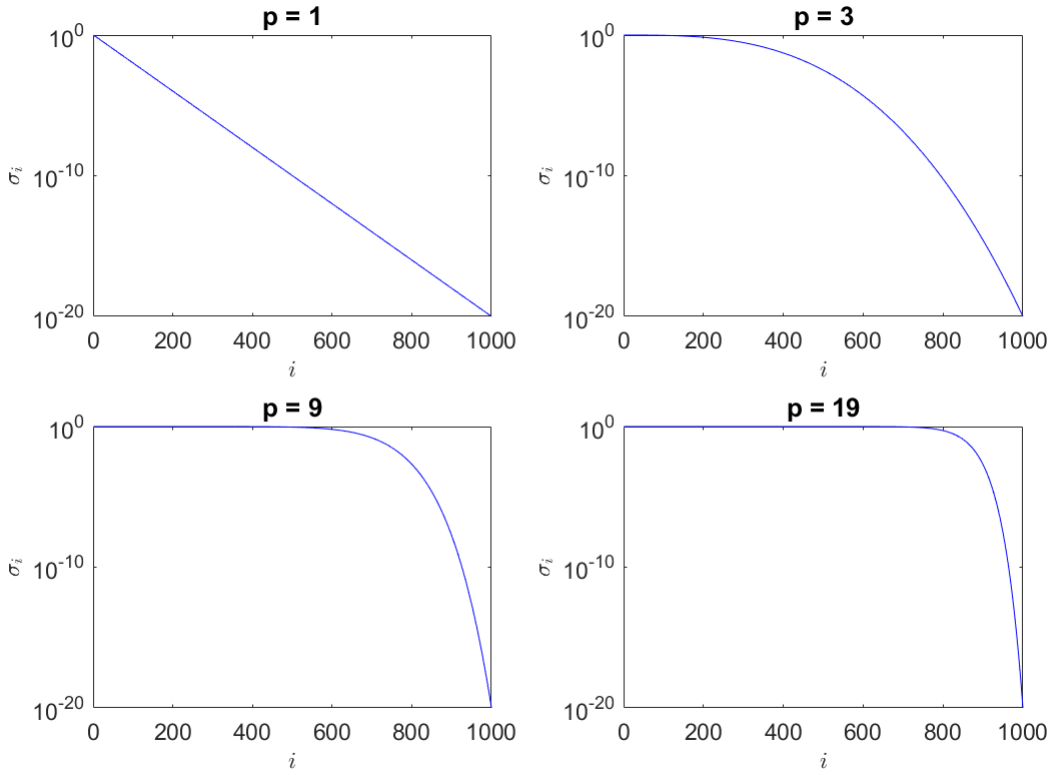


Figure 4.2: Slow-decaying singular value distributions, using n=1000.

The HAPOD weight ω was chosen by controlling the balance between the weight tolerance multiplier at the root node, ω, and the same multiplier at the child nodes, $\sqrt{1-\omega^2}$. This was accomplished by enforcing, for a parameter $k > 0$, that $\omega = k\sqrt{1-\omega^2}$, yielding

$$\omega = \frac{k}{\sqrt{1+k^2}}. \tag{4.9}$$

Finally, benchmarking and other computational investigations were performed using data from a 2-D simulation of strongly heated flow (a step towards simulation of a mine fire), simulated using a Boussinesq formulation, as is detailed in [25]. The dominant three temperature modes for the simulation are shown in Figure 4.4, and some representative temperature snapshots are shown in Figure 4.5. This simulation was run for 1001 time steps on a 201-by-401 2-D grid on a rectangular domain, resulting in a snapshot matrix $S \in \mathbb{R}^{80601\times 1001}$ for each variable, with twice the number of rows if the velocity vectors are to be reduced in tandem.

Three investigations were performed using this velocity data. Firstly, benchmarking using

the velocity data was performed using each of the three processor structures considered in this work in order to investigate the effectiveness of HAPOD when using simulated data with respect to the number of included bases. Secondly, benchmarking is performed using both the temperature data and a matrix with the same singular value distribution, but random singular vectors U and V. This is done in order to investigate the effects of matrix structure on performance. Finally, the performance of HAPOD using the temperature data is compared with that of a parallelized implementation of incremental SVD. The singular value distributions for this data are shown in Figure 4.3.

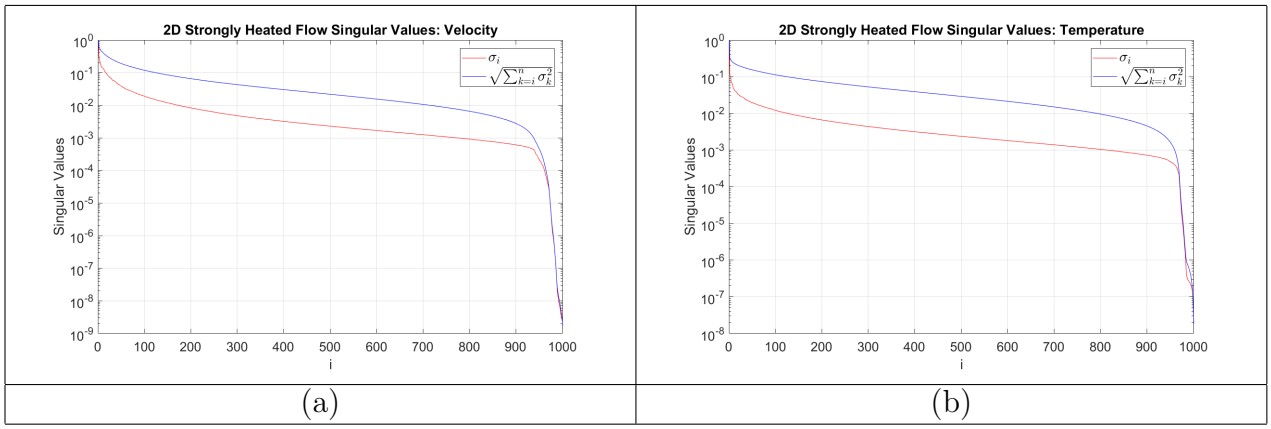


Figure 4.3: Singular value distributions of mine fire simulation data, including singular values and the theoretical energy at the cutoff points for each order n. These singular values and cutoff energies are scaled by the maximum observed singular value and cutoff energy, respectively.

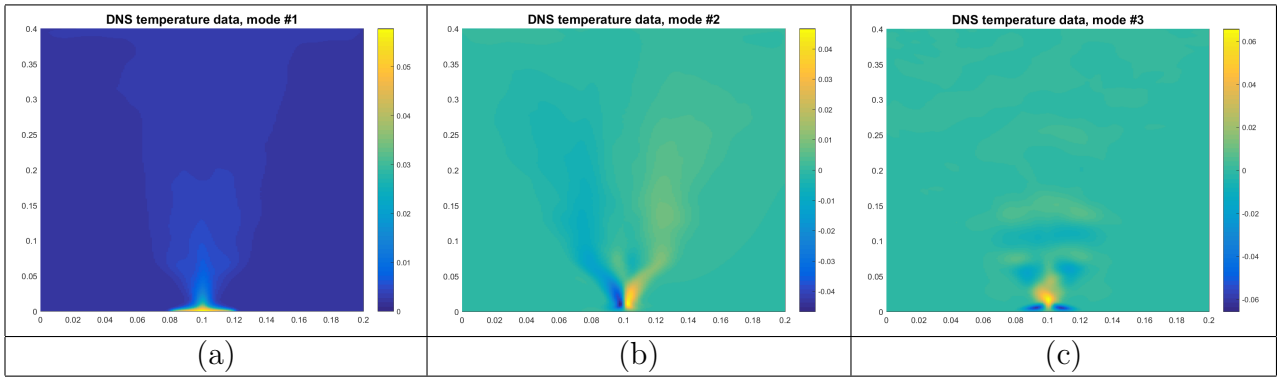


Figure 4.4: The dominant 3 modes of the temperature data from the strongly heated flow simulation.

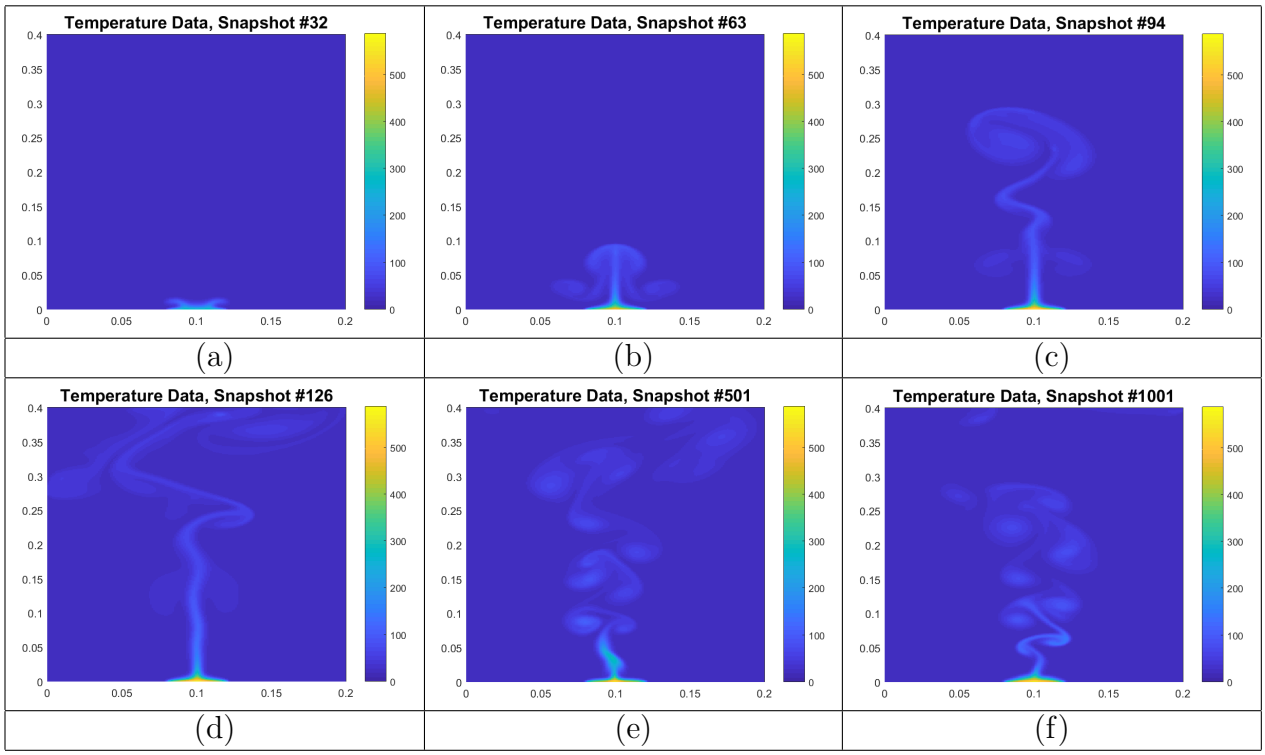


Figure 4.5: Some select snapshots of the strongly heated flow simulation.

The details pertaining to the exact reproduction of these results, including random number generator details and the details of the matrix dissemination amongst the MPI processors, are given an Appendix A.

4.2 Accuracy of HAPOD

Several tests were performed via Matlab to explore both the accuracy of live HAPOD and the resulting number of POD modes relative to those obtained using full truncated POD. This investigation was performed while varying the HAPOD weight, input tolerance, matrix dimensions, and the decay of singular values. The goal of these investigations was to explore the boundary where HAPOD becomes advantageous over full truncated SVD, and to explore how the choices of parameters such as weight impacts the balance between computational cost and the inflation of the number of modes in the resulting ROM.

4.2.1 Effects of the Weight

In this section, the effects of the HAPOD weight on the resulting accuracy and on the number of extra modes were investigated using $m = 1000$, $n = 2000$, and $\varepsilon = 1e-6$, where ε is defined as in (4.3). The singular value decay parameters used were $p = 1$, $p = 3$, and $p = 19$ (as defined in (4.7) and (4.8)), using both rapidly-decaying and slowly-decaying curvatures, while the HAPOD weight parameters were $k \in \{1/5, 1/4.5, ...1, 1.5, 2, ..., 5\}$ (as defined in (4.9)). The results of this test, shown in Figures 4.6 and 4.7, indicated that any gains in accuracy over direct truncated SVD were attained through the incorporation of extra input modes, resulting from a tighter tolerance at the root node. This phenomenon was more pronounced for smaller values of k (and thus smaller weights): naturally, the smaller the weight, the tighter the tolerance at the final step, and thus the more extra modes were retained. Finally, as expected, the resulting accuracy was never any *less* than the SVD accuracy, either.

These results also revealed that the number of extra modes at a given weight seemed to be primarily dependent on the logarithmic "slope" of the singular value curve near the cutoff point. For example, if the weight were $w = 0.1$, then the tolerance at the final step would be multiplied by 0.1. Thus, the shallower the slope of the singular value curve near the chosen cutoff point, the more singular values that lie within an order of magnitude of the cutoff point, and thus the more extra modes are incorporated for this weight. (Note that this cutoff point is summation-based, and thus will occur below the effective tolerance $\epsilon\sqrt{m}$.)

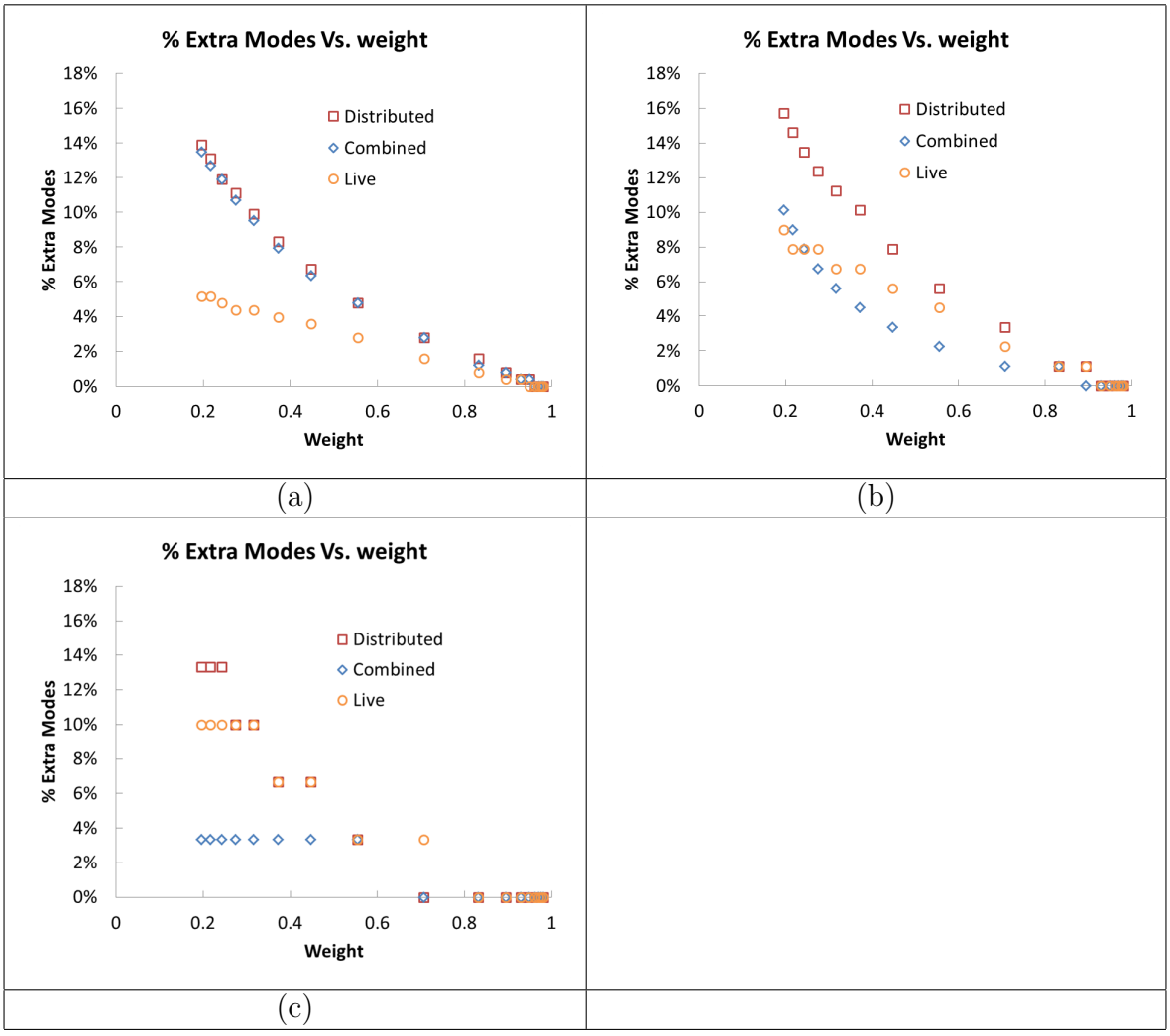


Figure 4.6: The percentage of extra modes observed (with respect to m) at various weights with $n = 2000$, $m = 1000$, a tolerance of $\varepsilon = 1e-6$, and fast-decaying singular value curves with (a) $p = 1$, (b) $p = 3$, and (c) $p = 9$.

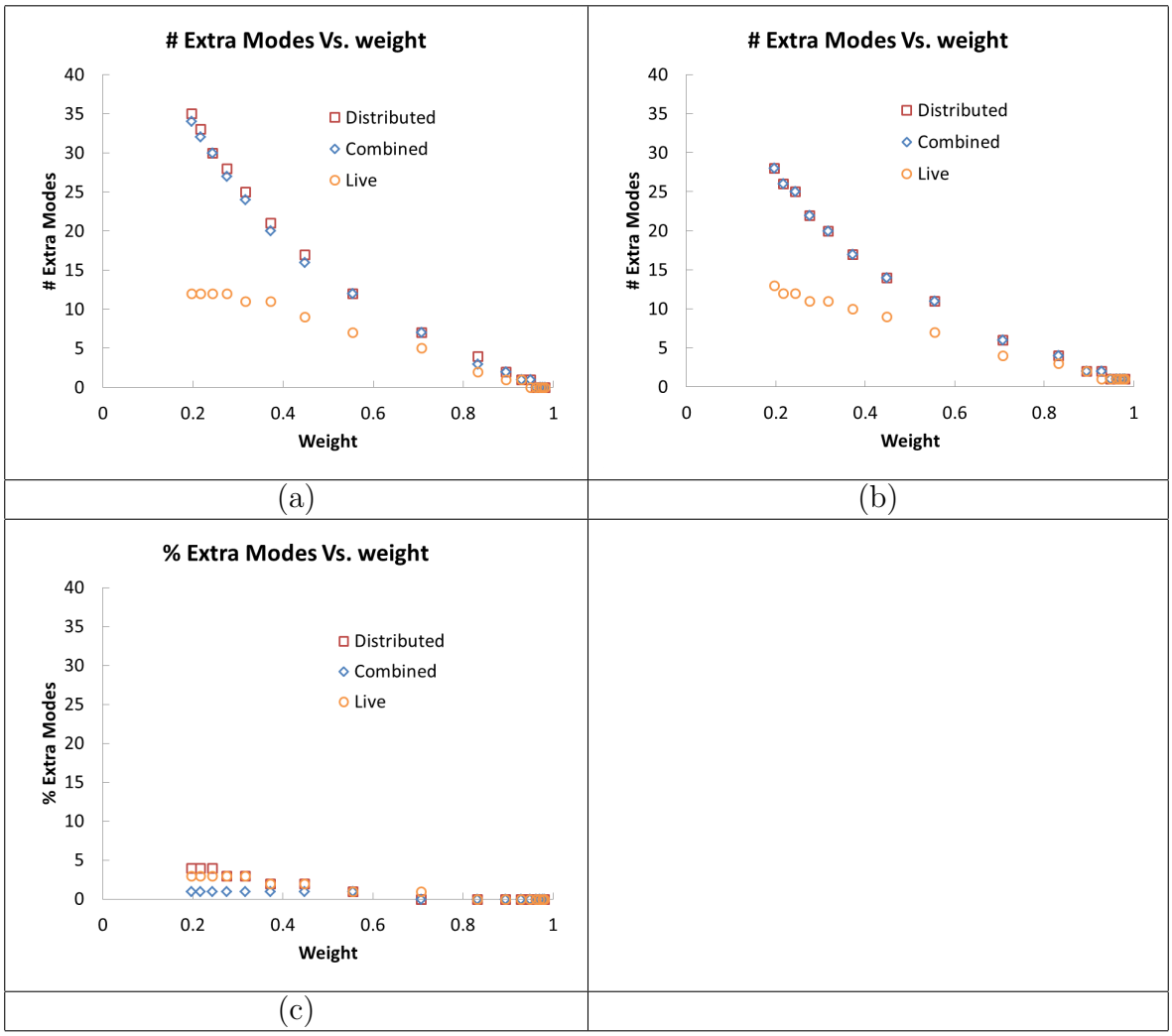


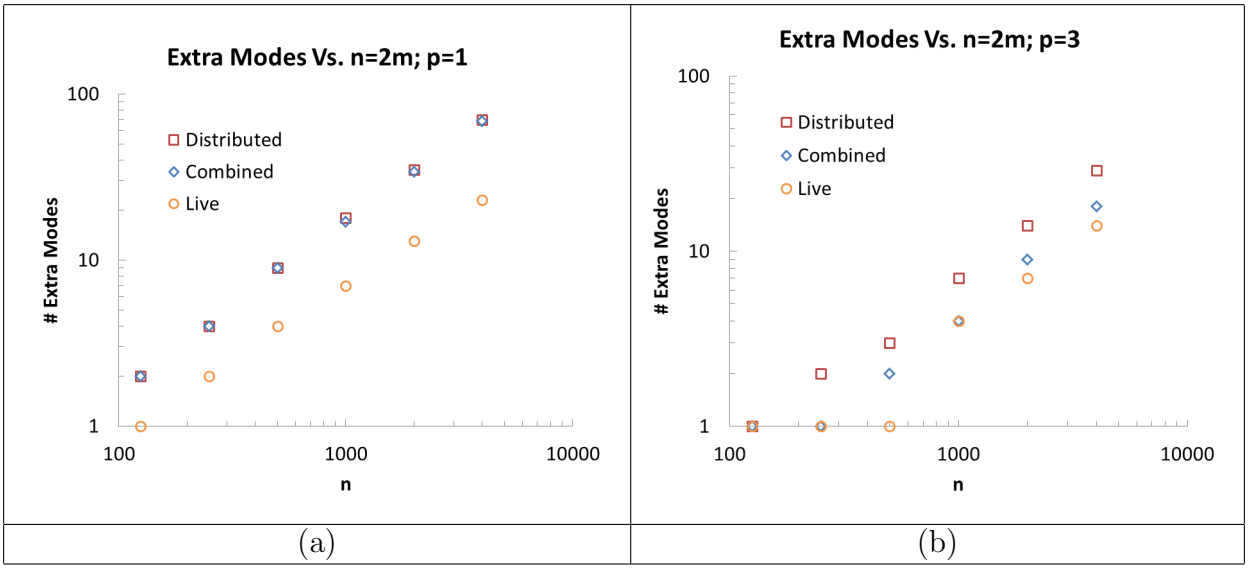
Figure 4.7: The percentage of extra modes observed (with respect to m) at various weights with $n = 2000$, $m = 1000$, a tolerance of $\varepsilon = 1e-6$, and slow-decaying singular value curves with (a) $p = 1$, (b) $p = 3$, and (c) $p = 9$.

4.2.2 Effects of Matrix Size

In the second test, the effects of the size of the snapshot matrix were investigated using $\varepsilon = 1e-6$ and a weight parameter of $k = 2$. The matrix dimensions were fixed at $n = 2m$, with $n \in \{125, 250, 500, 1000, 2000, 4000\}$. The results corroborate the notion that any gains in accuracy are directly tied to larger numbers of modes (Figures 4.8 and 4.10). Finally, as expected, a larger matrix with a fixed structure and singular value curvature directly leads to more extra modes, and the number of extra modes seems to scale linearly with the number of columns in the snapshot matrix, as shown in Figure 4.8. Indeed, once the number of extra

modes is normalized by m, the percentage of extra modes becomes relatively constant with respect to m, as shown in Figure 4.9. This is the case for both $p = 1$ and $p = 3$, though the $p = 3$ case shows more deviation from this trend due to the higher variance resulting from the smaller number of total modes.

It is worth noting that n makes no difference in either the HAPOD accuracy or the number of extra modes at the end of the simulation, assuming the shape of the singular value curve remains unchanged. This is because the fraction of extra modes is dependent largely on the singular value distribution, the matrix structure, and the input parameters (HAPOD weight and target tolerance), while varying n on its own has no impact on any of these parameters (assuming a randomized matrix where the shape of the singular value curve is held constant, with $n \geq m$). Indeed, in a low-weight test in which only n was varied at various values of p, this result was corroborated: n had no impact on either the final HAPOD accuracy or on the number of resulting HAPOD modes (no more than a single mode of difference at a low weight), as shown in Figure 4.11. Furthermore, the test revealed that the number of *intermediate* modes before the final step was also independent of n (with up to one or two modes of variance stemming from the randomization of the original matrix), as shown in Figure 4.12.



(a) (b)

Figure 4.8: Extra modes vs. $n = 2m$ with a weight of $\omega \approx 0.2$, a tolerance of $\varepsilon = 1e - 6$, and (a) $p = 1$, (b) $p = 3$.

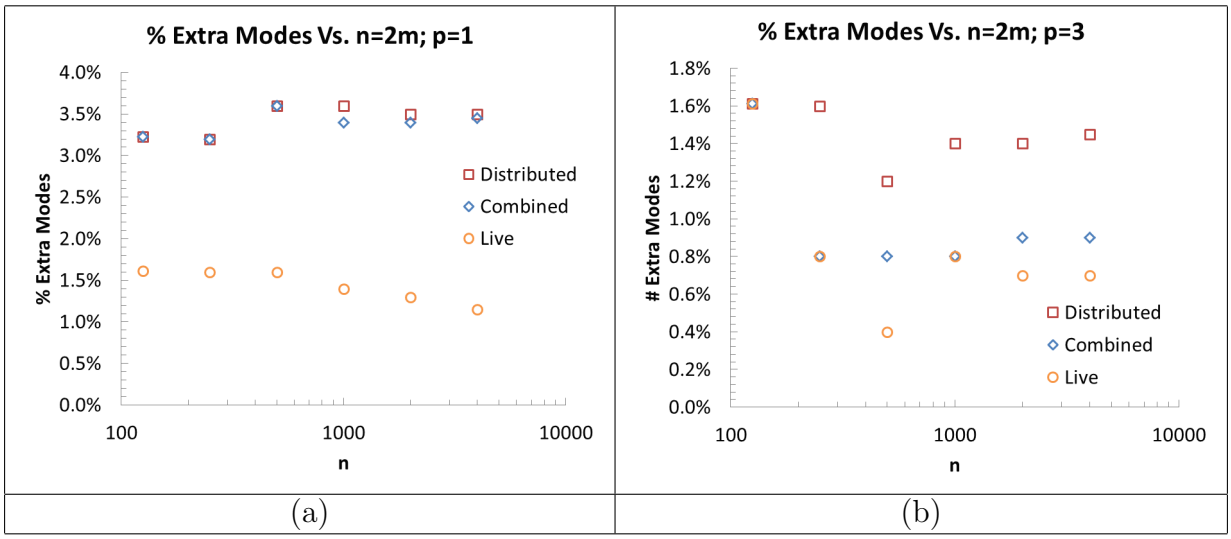


Figure 4.9: Percentage of extra modes (with respect to m) vs. $n = 2m$ with a weight of $\omega \approx 0.2$, a tolerance of $\varepsilon = 1e-6$, and (a) $p = 1$, (b) $p = 3$.

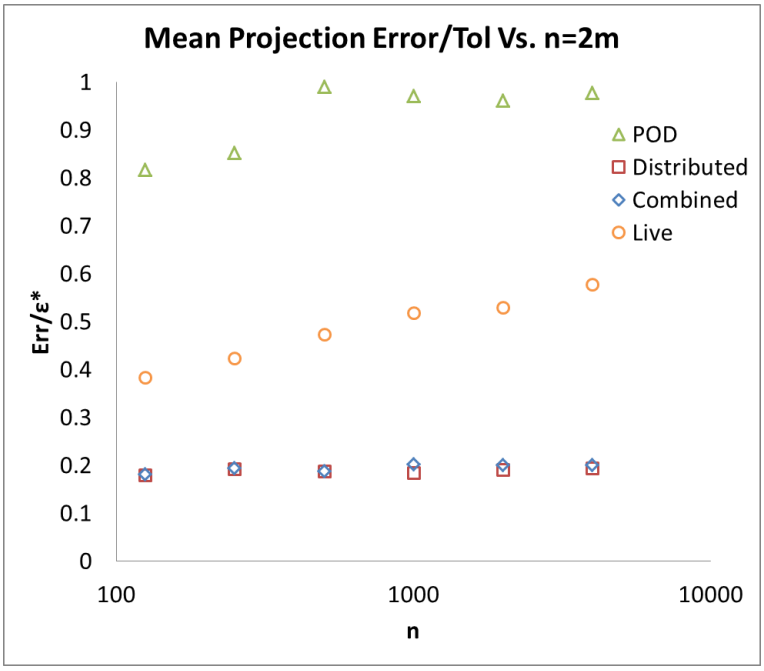


Figure 4.10: Mean POD projection error vs. $n = 2m$ with a weight of $\omega \approx 0.2$, a tolerance of $\varepsilon = 1e-6$, and $p = 1$.

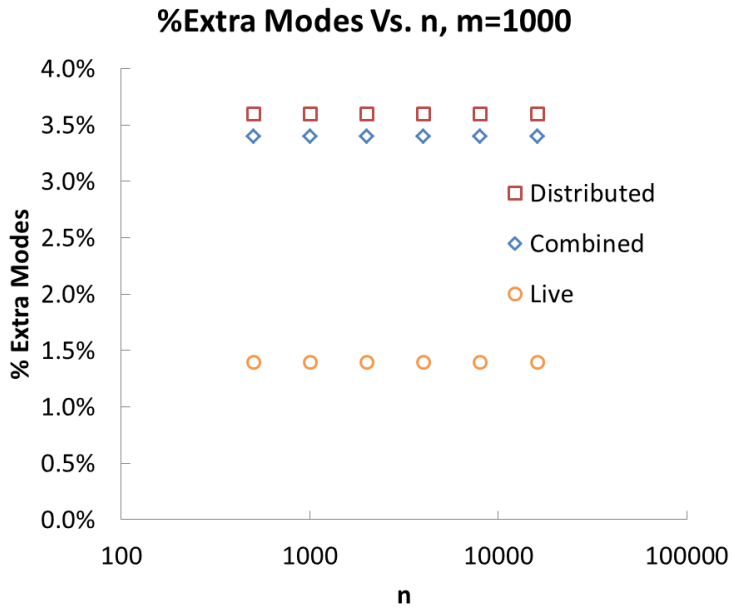


Figure 4.11: Percentage of extra modes (with respect to m) vs. n with $m = 1000$, a weight of $\omega \approx 0.2$, a tolerance of $\varepsilon = 1e-6$, and $p = 1$.

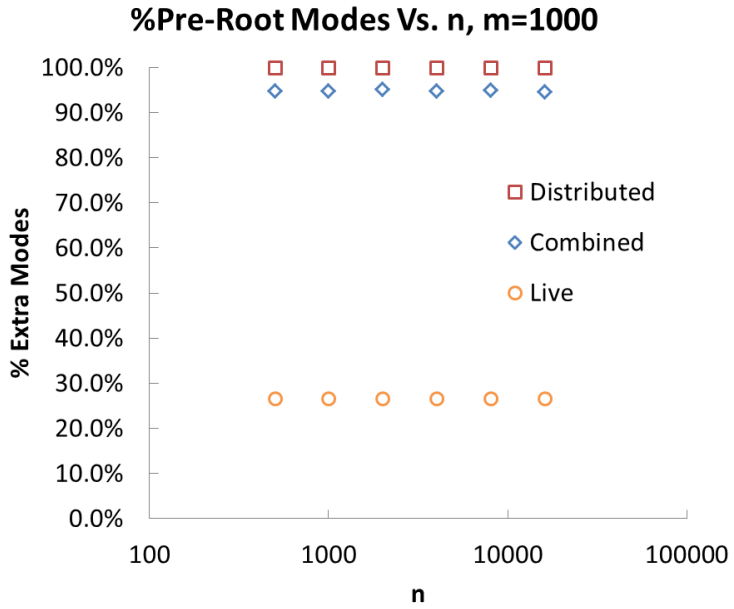


Figure 4.12: Percentage of modes (with respect to m) input into the final step of Truncated SVD vs. n with $m = 1000$, a weight of $\omega \approx 0.2$, a tolerance of $\varepsilon = 1e-6$, and $p = 1$.

4.2.3 Effects of the Singular Value Distribution

Finally, the effect of the *shape* of the singular value curve on the number of extra POD modes was explored at two different weights, corresponding to $k = 0.2$ and $k = 2$. The singular value shape parameters used were $p \in \{1, 2, ..., 16\}$ for both the rapidly-decaying and slow-decaying curve shapes. The tolerances used were $\varepsilon \in \{1e-8, 1e-6, 1e-4, 1e-2\}$. The $k = 2$ test shows that, at the corresponding weight, there are very few extra modes (no more than 4 modes out of the original 1000, and no more than 2 modes for the rapidly-decaying case).

The $k = 0.2$ test, on the other hand, was more revealing: For the fast-decaying cases (Figure 4.13), the tolerance seems to have relatively little effect on the number of extra modes observed, though the total number of modes decreases as the tolerance increases. This can be explained by the slopes of the singular value curves: From Figure 4.1, it can be seen that these slopes do not seem to change much between these tolerances, resulting in only small changes in the number of extra modes observed, even though the total number of modes changed significantly. (For the $p = 1$ case, a tolerance of $\varepsilon = 1e-8$ resulted in ~ 350 total POD modes, while a tolerance of $\varepsilon = 1e-2$ resulted in only ~ 50 total modes.)

The slow-decaying version emphasizes the notion that the logarithmic slope of the singular value curve controls the number of extra modes. Consider the case of $p = 2$, shown in Figure 4.15. From the figure, it can be seen that the magnitude of the slope of the singular value curve decreases significantly as the tolerance increases, corresponding with a significant increase in the number of extra modes observed in Figure 4.14, despite that fact that the *total* number of modes actually *decreases* significantly! Fortunately, the use of a higher weight can significantly reduce the number of extra modes: With a weight of approximately 0.9, and for the shapes and tolerances tested above, the number of extra modes never exceeded 2 in the fast-decaying case and 4 in the slow-decaying case (out of an original matrix size of $m = 1000$), as can be observed in Figures 4.16 and 4.17.

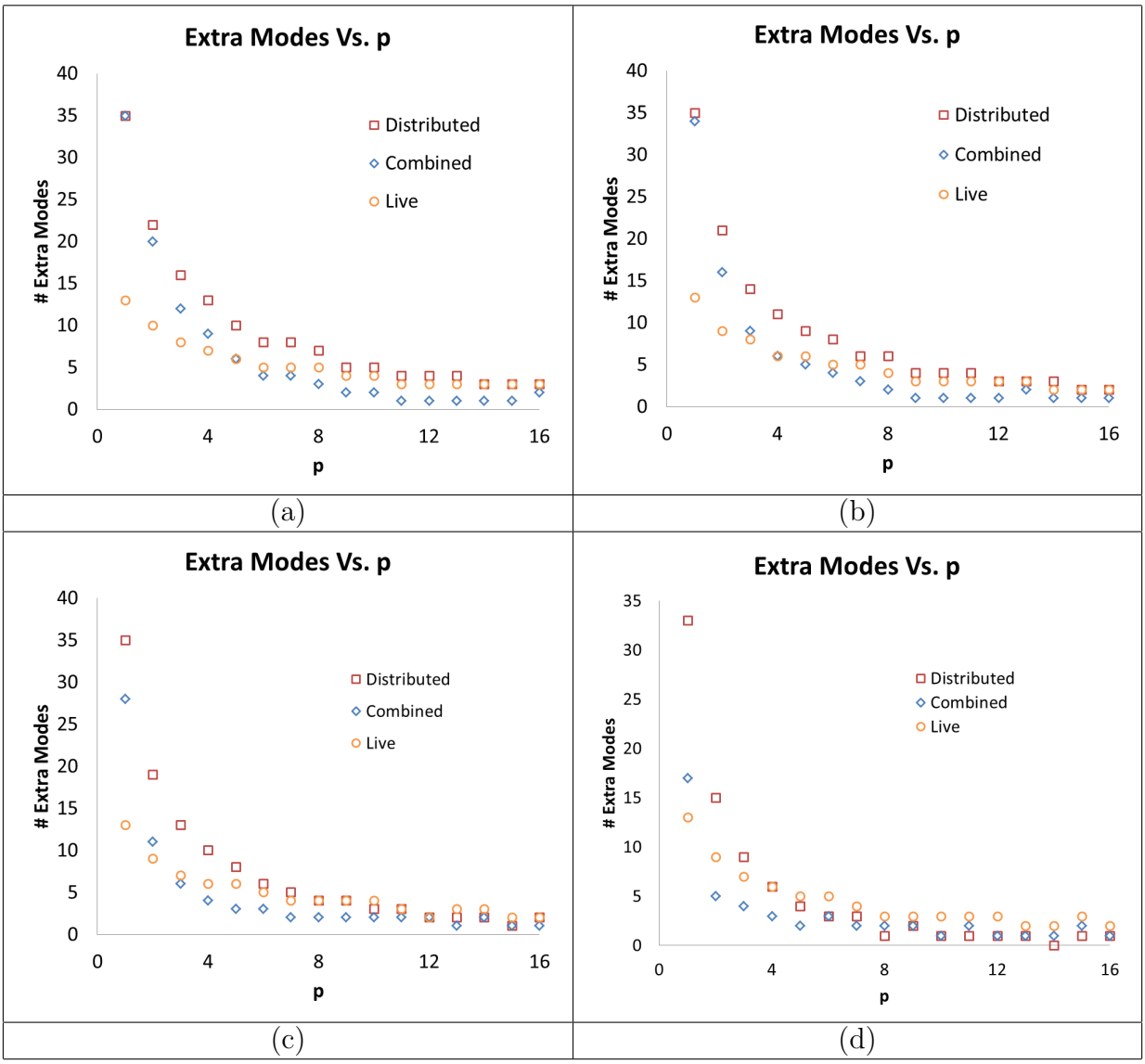


Figure 4.13: The number of extra HAPOD modes observed using various fast-decaying graph shapes with $n = 2000$, $m = 1000$, and a weight of $\omega \approx 0.2$, with (a) $\varepsilon = 1e-8$, (b) $\varepsilon = 1e-6$, (c) $\varepsilon = 1e-4$, and (d) $\varepsilon = 1e-2$.

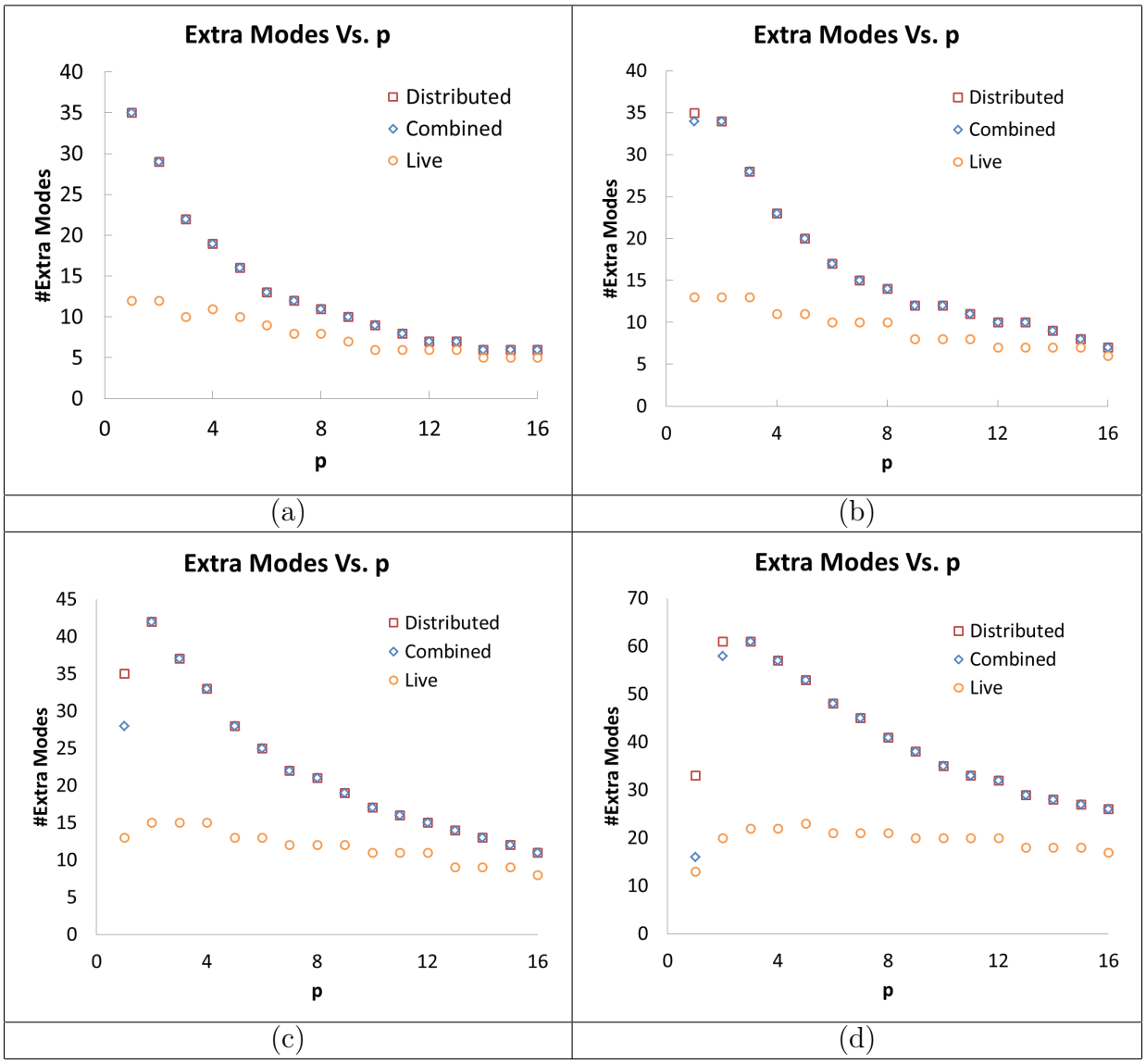


Figure 4.14: The number of extra HAPOD modes observe using various slow-decaying graph shapes with $n = 2000$, $m = 1000$, and a weight of $\omega \approx 0.2$, with (a) $\varepsilon = 1e-8$, (b) $\varepsilon = 1e-6$, (c) $\varepsilon = 1e-4$, and (d) $\varepsilon = 1e-2$.

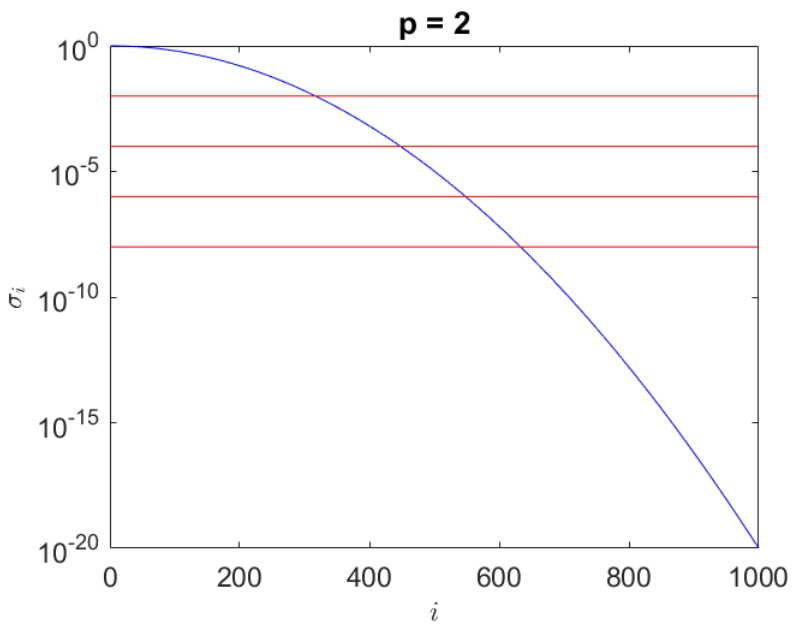


Figure 4.15: Slow-decaying Singular Value Curve Shape with $p = 2$, showing the tolerances used in Figure 4.14.

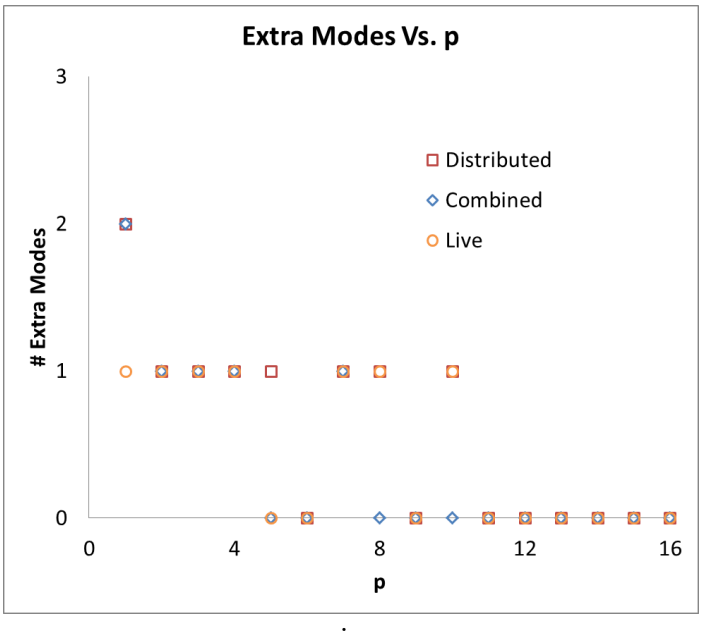


Figure 4.16: The number of extra HAPOD modes observed (as a fraction of the total number of POD modes) using various fast-decaying graph shapes with $n = 2000$, $m = 1000$, and a weight of $\omega \approx 0.894$, with $\varepsilon = 1e - 8$

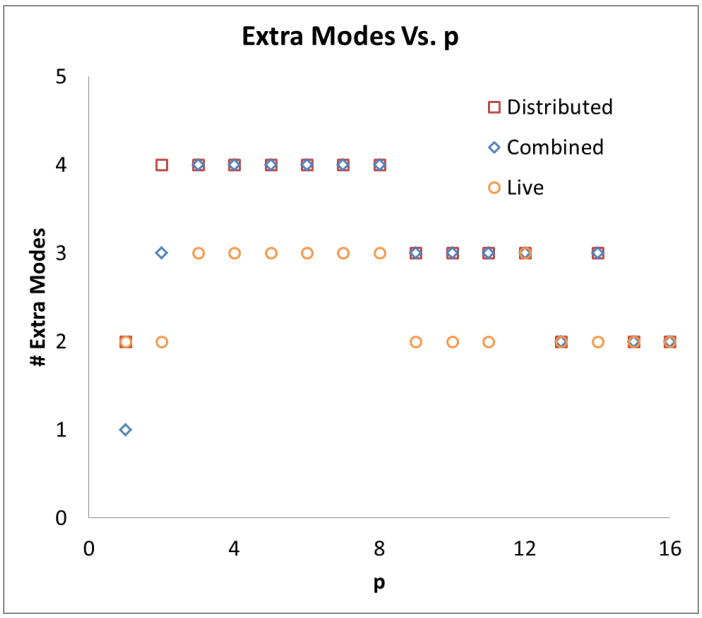


Figure 4.17: The number of extra HAPOD modes observed (as a fraction of the total number of POD modes) using various slow-decaying graph shapes with $n = 2000$, $m = 1000$, and a weight of $\omega \approx 0.894$, with $\varepsilon = 1e - 2$.

4.3 Benchmarking

HAPOD was benchmarked using both the synthetic data in the previous section and 2-D simulation data of strongly heated flows using a Boussinesq approximation for fluid buoyancy [25, 44]:

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} = -\mathbf{u} \cdot \nabla \mathbf{u} - \nabla p + \nu \nabla^2 \mathbf{u} - \beta \mathbf{g}(T - T_\infty) \\ 0 = \nabla \cdot \mathbf{u} \\ \dfrac{\partial T}{\partial t} = -\mathbf{u} \cdot \nabla T + \alpha \nabla^2 T. \end{cases} \tag{4.10}$$

In the study with synthetic data, the effects of the matrix characteristics and algorithm inputs on the simulation time are investigated. To this end, benchmarking was performed while varying the singular value distribution of the snapshot matrix, the size of the matrix, and the choice of HAPOD weights and POD tolerances. For the study with simulation data, as the singular value distribution can no longer be controlled explicitly, only the effects of matrix width and input tolerance are studied. In addition, HAPOD is benchmarked against Arrighi et al.'s implementation [4] of Brand's incremental SVD algorithm [2], and against HAPOD on a randomized matrix with the same singular values. In all cases, the

benchmarking was performed using C++ on a remote computing cluster, parallelized using 4 MPI processors.

4.3.1 Synthetic Data

Effect of the Weights

The benchmarking with synthetic data was performed with snapshot matrices having fast-decaying singular value curves with $p \in \{1, 3, 9, 19\}$, using 1, 2, 4, and 8 MPI processes and a tolerance of $\varepsilon = 1e-6$. The results for 1 process and 4 processes are shown in Figures 4.18 and 4.19, respectively. With only one process, note that the combined method is identical to the live method, and so the results are nearly identical (up to some statistical variance in run-times). For the $p = 1$ case, none of the HAPOD methods yielded any speed advantage over direct SVD. This can be explained by the number of modes remaining before the final stage (Figure 4.20): For the distributed case, HAPOD yielded no reduction of modes at all in intermediate stages except in the extreme case of $p = 19$, resulting in no possible gains in simulation time. On the other hand, for the live case, a relatively large mode count in the intermediate stages resulted in a significantly increased simulation time, despite the roughly 70% reduction in mode count before the final step (as indicated in Figure 4.20). In contrast, as the SVD curve steepened, HAPOD began to yield significant gains over full SVD, even with no parallelism. These gains were especially pronounced for the combined method, as the other methods gained little from this parallelism: The live approach is run in serial, while the computational cost of the distributed case was dominated by the final step, which, even for the most rapidly-decaying singular value curve tested, resulted in only a 60% reduction in the number of modes before the final step. For the combined and live methods, the computational gains were already pronounced at all tested weights for $p = 3$, and the combined method in particular was especially fast. All computational gains were strongly correlated with a large reduction in the number of modes before the final step, as can be seen in Figure 4.20.

It is worth noting that, for these tests, the computational cost for most of these methods did not change very significantly as the weights increased, especially for the combined method. However, note that the singular value curves are extremely steep near the target cutoff, as can be seen in Figure 4.1. Even at the largest weight of roughly $\omega = 0.98$, the tolerance is reduced by less than an order of magnitude at the intermediate stages, resulting a very small increase in the number of modes before the final step (Figure 4.20). This effect may be much more pronounced if the singular value curve is more shallow. For example, consider a case where HAPOD is applied on a sample $10^6 \times 73$ matrix arising from time-integration of a PDE, whose singular value decay is given in 4.21. Using a tolerance of $0.01 \cdot \sigma_m ax$, the computational cost begins to escalate rapidly as the weights increase (as shown in Figure 4.22) due to the very shallow singular value decay near the tolerance. This escalation corroborates the

need for a balanced choice of weight. At this point, we suggest a weight of $\omega = \frac{1}{\sqrt{2}}$, due to both the marginal number of extra modes observed ($<\sim 4\%$ for the tested matrices and tolerances) and to the marginal increase in simulation time. This weight solves $\omega = \sqrt{1-\omega^2}$, corresponding to a choice of $k = 1$, and results in an equal weight multiplier to the tolerance for both the final and intermediate SVD steps.

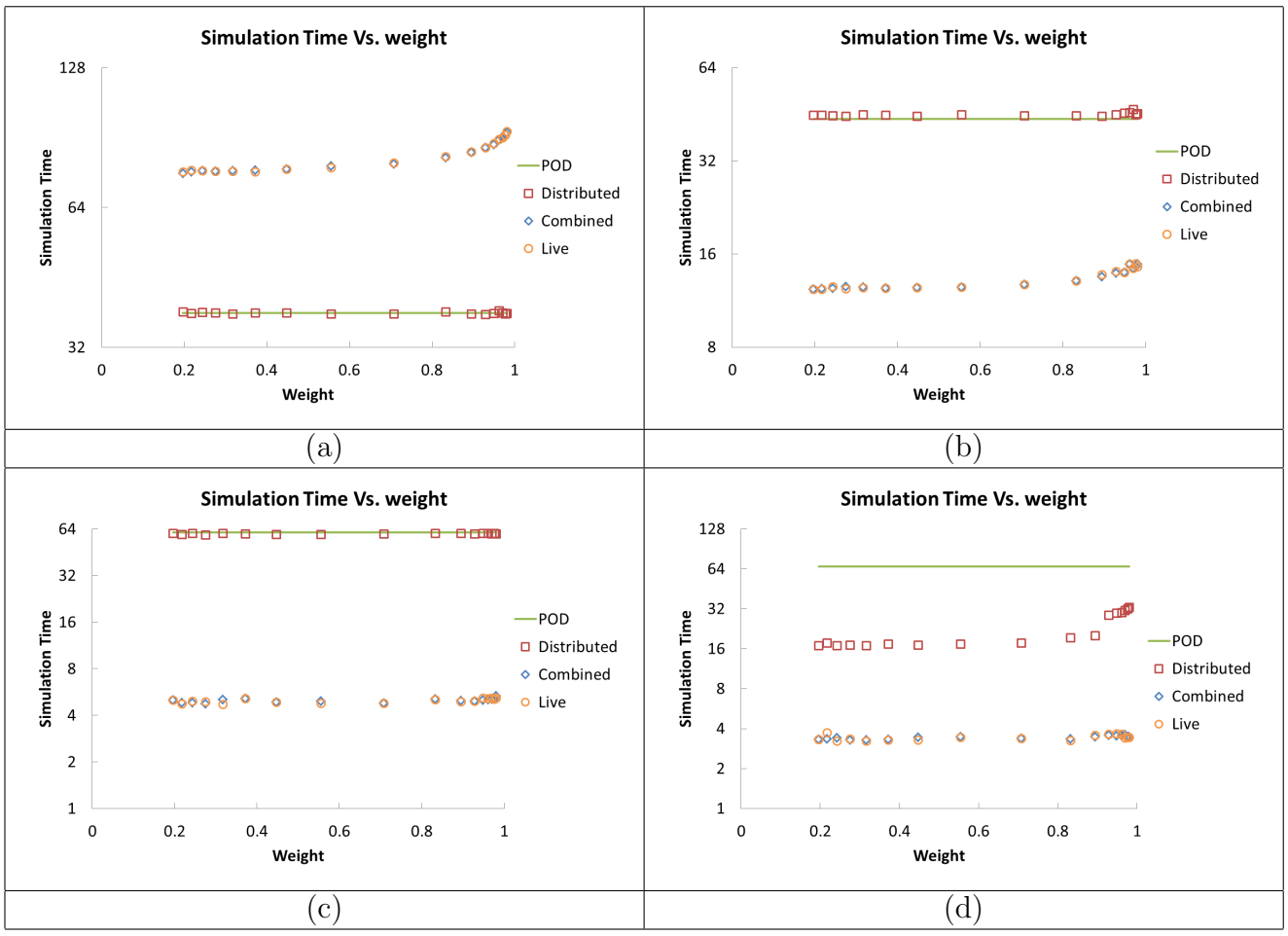


Figure 4.18: Fast-decay benchmarking results with $n = m = 2000$, a tolerance of $\varepsilon = 1e-6$, and 1 MPI process, with (a) $p = 1$, (b) $p = 3$, (c) $p = 9$, and (d) $p = 19$.

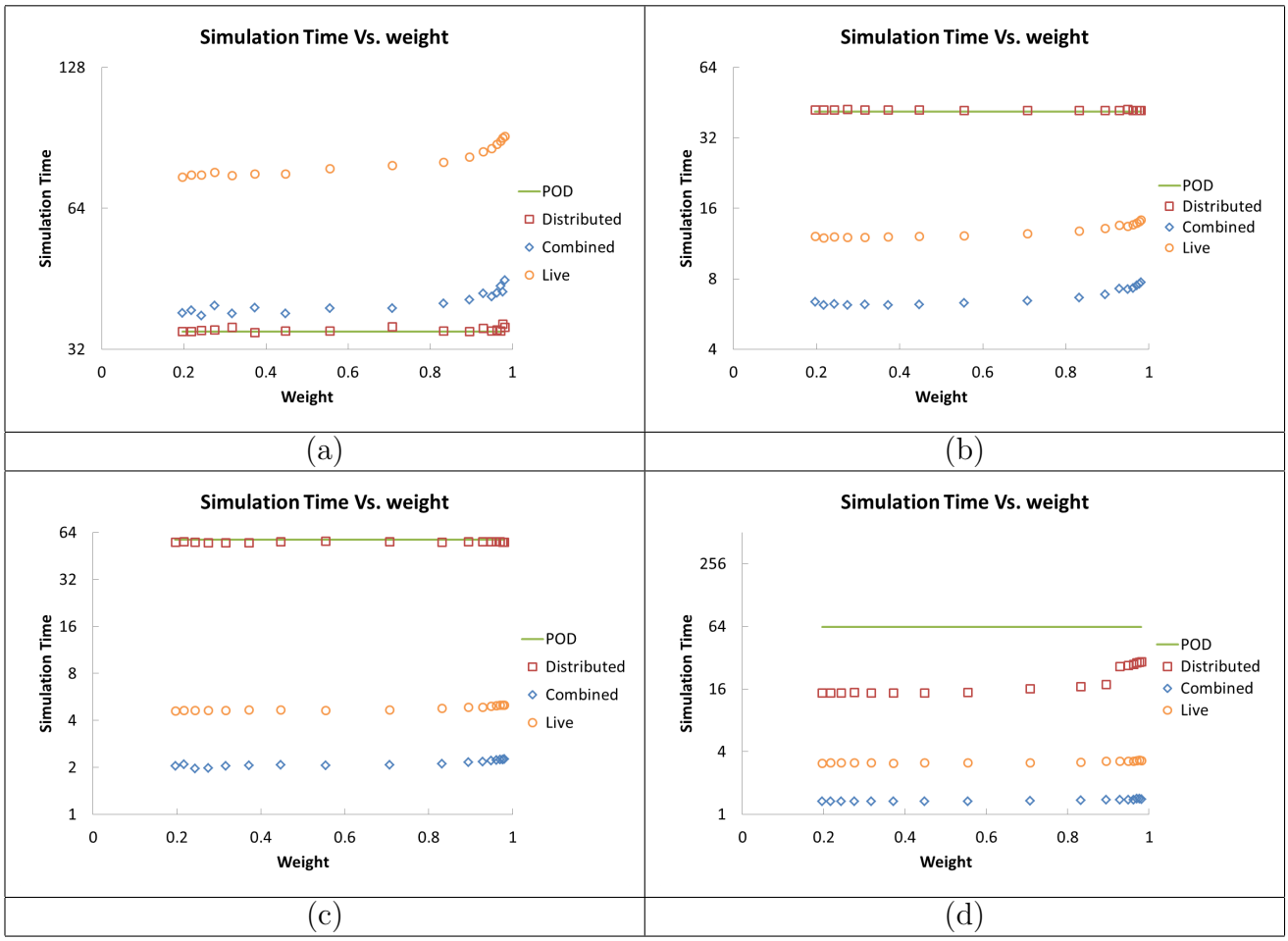


Figure 4.19: Fast-decay benchmarking results with $n = m = 2000$, a tolerance of $\varepsilon = 1e-6$, and 4 MPI processes, with (a) $p = 1$, (b) $p = 3$, (c) $p = 9$, and (d) $p = 19$.

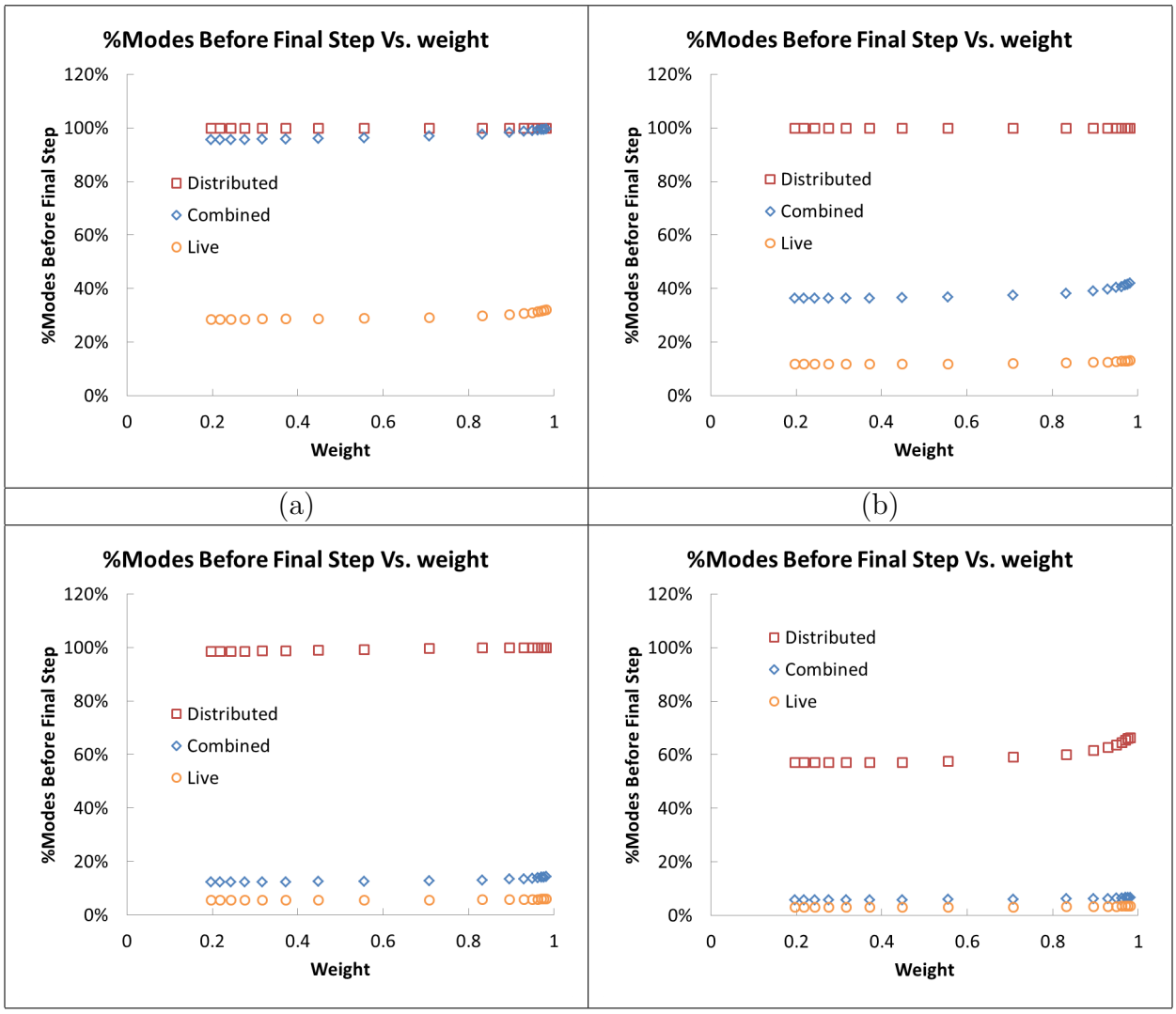


Figure 4.20: Percentage of the total modes remaining in input to the final step with $n = m = 2000$, a tolerance of $\varepsilon = 1e-6$, and 4 MPI processes, with (a) $p = 1$, (b) $p = 3$, (c) $p = 9$, and (d) $p = 19$.

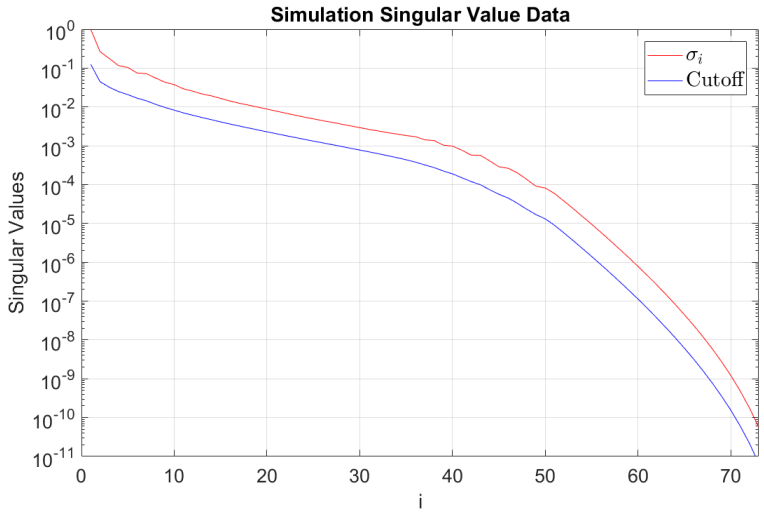


Figure 4.21: Normalized singular values of a snapshot matrix resulting from the simulation of a PDE.

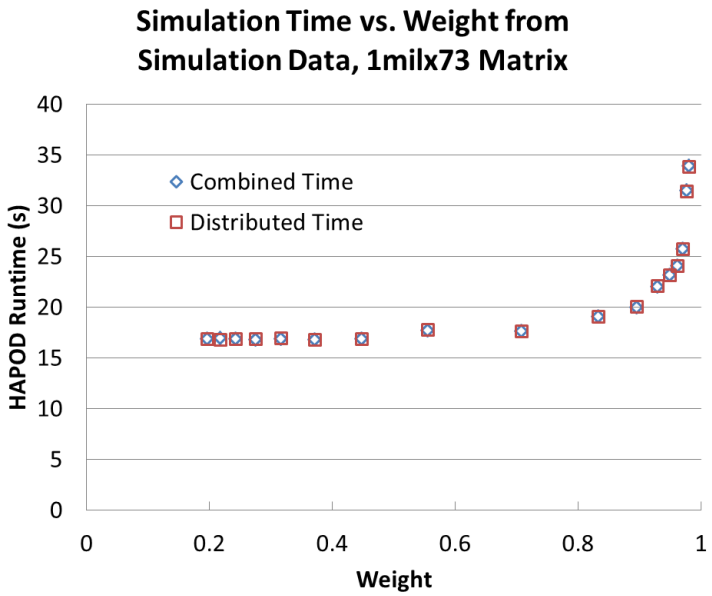
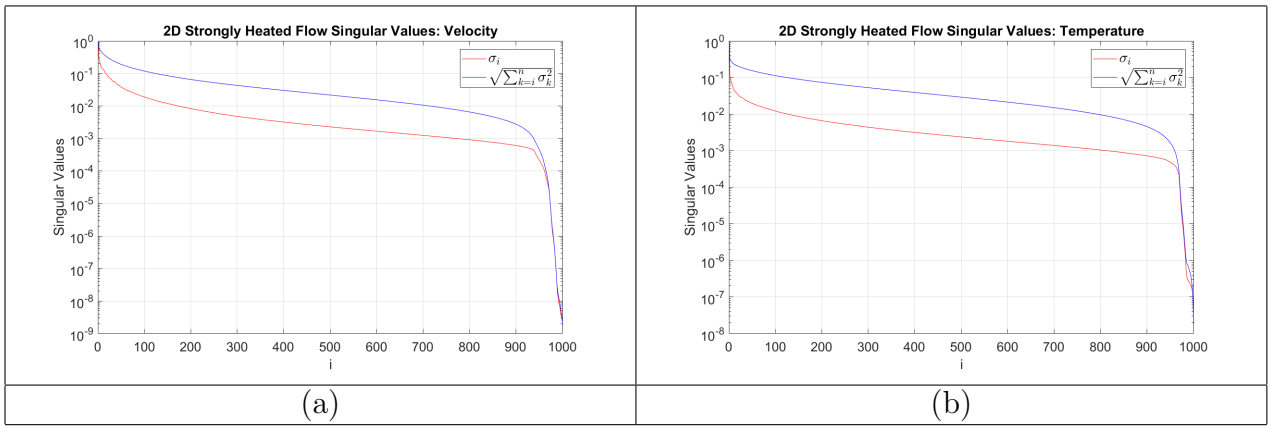


Figure 4.22: Benchmarking results from a snapshot matrix resulting from the simulation of a PDE, using a tolerance of $\varepsilon = 1e-2 \cdot \sigma_{max}$.

4.3.2 Simulation Data

The purpose of this section is threefold: To investigate the effectiveness of HAPOD when using simulated data with respect to the number of included bases, to investigate the effects of matrix structure on performance, and to compare the performance of HAPOD with that of a parallelized implementation of the incremental SVD. In all cases, the simulation data used for testing arises from 1001 time-steps (20 seconds total) of a 2-D finite element simulation of a strongly heated flow using the Boussinesq formulation (4.10).

Separate bases were constructed using both the temperature data and the velocity data. The singular value distributions for this data are shown again for reference in Figure 4.23.



(a) (b)

Figure 4.23: Singular value distributions of mine fire simulation data, including singular values and the theoretical energy at the cutoff points for each order n. These singular values and cutoff energies are scaled by the maximum observed singular value and cutoff energy, respectively.

Effectiveness of HAPOD

In order to investigate the effectiveness of HAPOD using real simulation data, the time-performance of HAPOD was recorded while varying the tolerance and number of included snapshots for the LES simulation data. As can be observed in Figure 4.24, the live structure tends to require much more time than the combined and distributed structures, while the distributed structure tends to be the fastest. Curiously, this is the case even when 10% of the energy is truncated, resulting in quite a significant reduction in the number of modes, as shown in Figure 4.25. Comparing Figures 4.23 (d) and 4.27, it seems that most of the less significant modes (with energy less than 0.1% of the maximum energy) show up in only the latter portion of the simulation, where the maximum temperature behavior seems to have stabilized (see Figure 4.26). Indeed, the smallest singular value observed in this portion is significantly larger than the smallest values in the simulation as a whole, while the small

singular values show up in the decomposition of the first 200 modes. This is a result of the nature of the simulation, as once the heat source at the bottom of the domain is activated, a warm-up phase ensues. Naturally, the dominant modes from this phase are largely absent from the remainder of the simulation due to their transient nature. This transient phase results in a slightly faster time for HAPOD when most of the columns are used, since these modes can be quickly eliminated from the early matrix slices. Note that, when using the full snapshot matrix, the computational effectiveness of HAPOD is relatively limited with a tolerance of even 1%, since the reduction in the number of modes is not very significant (roughly 760 modes were required out of the 1001 total modes, compared to the theoretical cutoff of 713 modes).

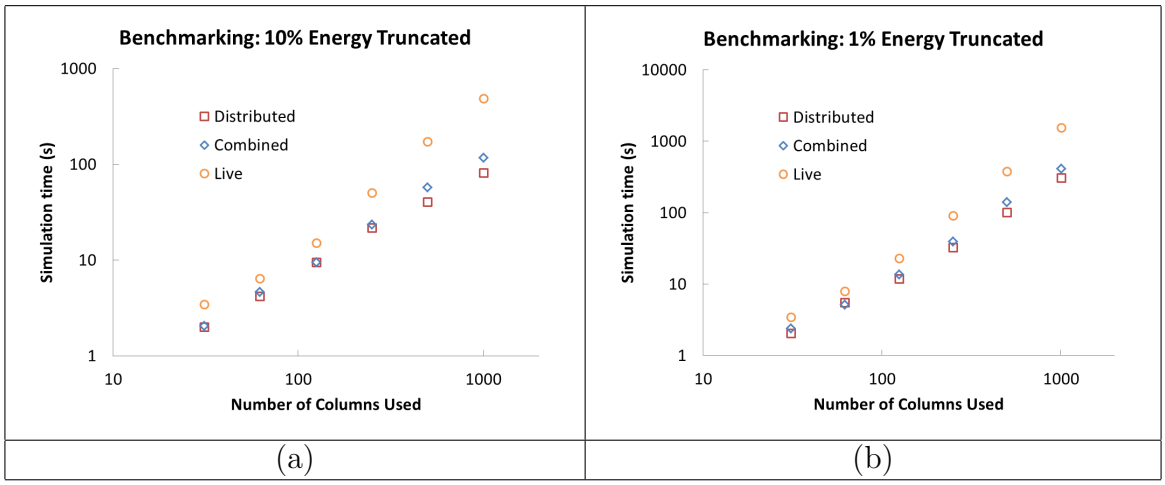


Figure 4.24: Benchmarking of HAPOD using velocity data from a 2-D mine fire simulation and a relative tolerance of (a) 10% and (b) 1%.

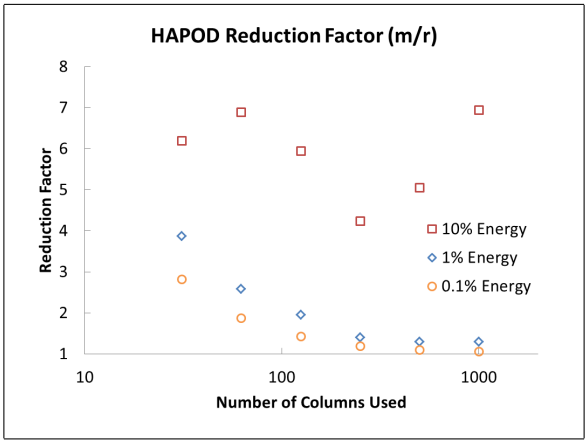


Figure 4.25: Factor of order reduction $\frac{m}{r}$ resulting from HAPOD using velocity data from a 2-D mine fire simulation, using relative tolerances of 10%, 1%, and 0.1%.

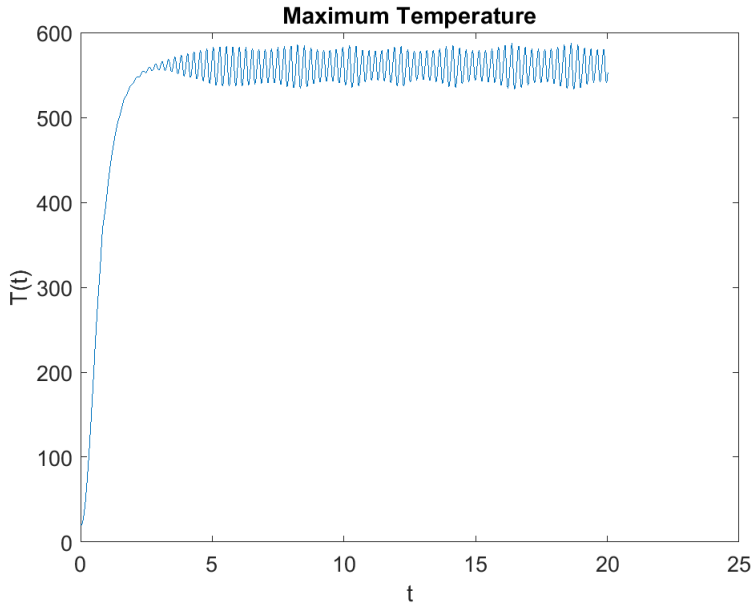


Figure 4.26: Maximum temperature for the LES simulation data at each time step.

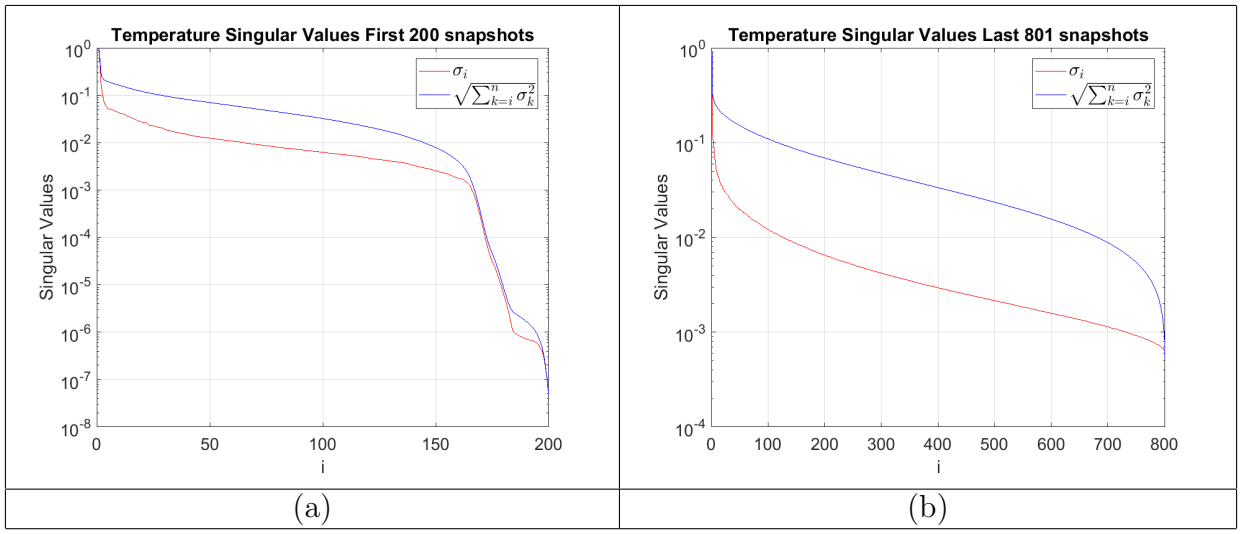


Figure 4.27: Singular value distribution for (a) the first 200 snapshots and (b) the last 801 temperature snapshots from a 2-D strongly heated flow simulation.

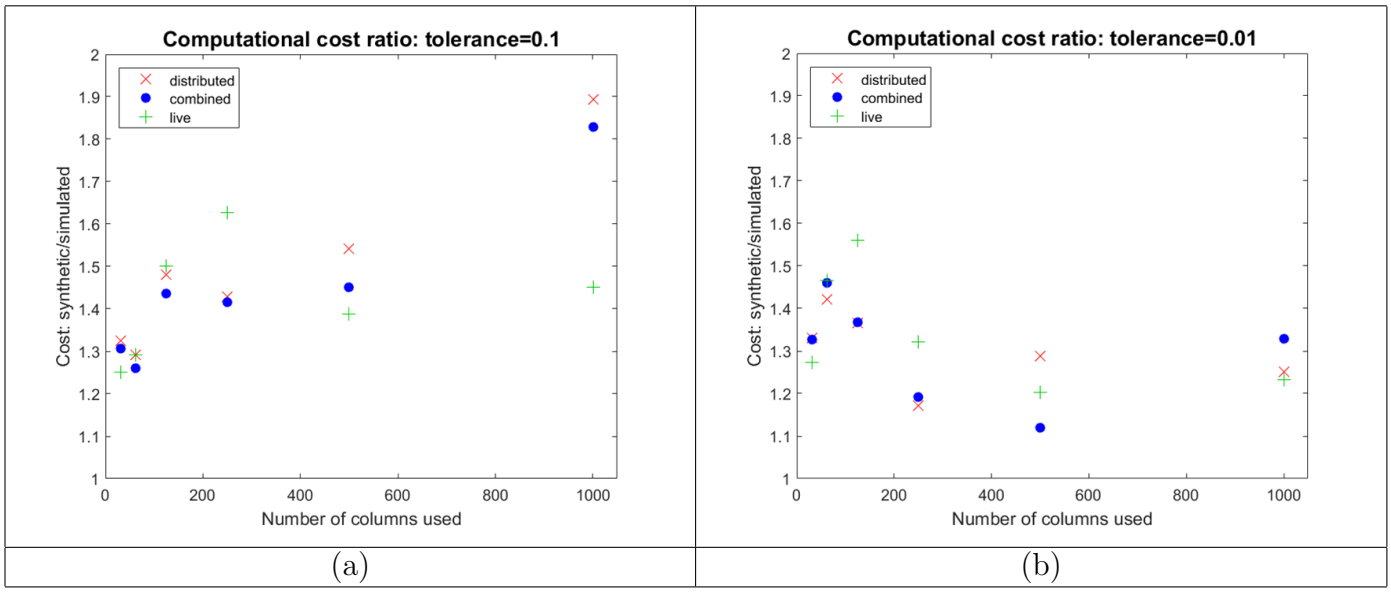
The Effects of Matrix Structure

To study the effects of matrix structure on the performance of HAPOD, the full SVD of the snapshot matrix for Temperature was computed, and randomized synthetic data was constructed with the same singular value distribution. This study was performed using the LES

simulation data, with relative tolerances of 0.1 and 0.01. As it turns out, the experiments using synthetic data required anywhere between 10% and 100% more time to compute than their counterparts using the simulated data, as shown in Figure 4.28. This is most likely due to the fact that the random nature of the modes from the synthetic data results in behavior that fluctuates very quickly in time, while the structures arising from the simulated data display result in solutions that vary more slowly in time. This slow time-variance can result in a more pronounced reduction in the number of modes for the intermediate stages, and thus a cheaper simulation time.

This potential for structure-based speedup can be corroborated using another sort of synthetic data: If every row in the snapshot matrix were to oscillate with its own frequency, with a bound on the maximum frequency allowed, far more pronounced gains can be observed. To show how such gains may occur, a synthetic snapshot matrix $S \in \mathbb{R}^{512\times512}$ was generated with a maximum frequency of 1 full oscillation per 16 time steps (using randomly, but logarithmically, distributed time-frequencies at each point in the domain). (Maximum periods ranging from 1 to 512 time steps were tested, but 16 time steps was the frequency cap at which the structure made the largest difference.) A completely random matrix with the same singular values was then generated, and the computation time of HAPOD was then tested in MATLAB. The truncated SVD of each matrix was then approximated via distributed HAPOD using a relative tolerance of 1% and a weight of $\omega = \frac{1}{\sqrt{2}}$. This was repeated 100 times per generated matrix, and the runtimes averaged to compute the speedup attained from using the original matrices. This process was repeated 20 times, and the ratios averaged.

On average, the HAPOD required a factor of 5.81 more time to run in MATLAB for the fully randomized matrix (running in serial), though the singular values of the two matrices were identical. This speed-up was due to a reduction in the number of modes in the intermediate stages: HAPOD using the fully random matrix resulted in no reduction of modes before the final step, and so a full-order SVD was performed in the final step. On the other hand, HAPOD using the original matrix reduced the order to just 87 modes before the final step, saving a significant amount of time. (For reference, the final truncated bases contained 34 modes, while the theoretical cutoff was 33 modes). These savings likely owe largely to the slow rate of change of the modes: Fewer key modes show up in each local slice of the matrix, resulting in less repeated data, and more pronounced reduction, in the modes before the final step.



(a) (b)

Figure 4.28: Ratio of costs for synthetic vs. simulated data (LES), using a relative tolerance of (a) 10% and (b) 1% of the total energy.

Comparison with Incremental SVD

Finally, to compare the effectiveness of HAPOD with that of an implementation of the incremental SVD solver, the two were benchmarked using the Temperature data from the LES simulation. However, on the first attempt to compare the methods, the incremental solver performed far worse than HAPOD in most cases, and much worse than expected (nearly three orders of magnitude slower than HAPOD in the worst cases tested), as can be seen in Figure 4.29. Upon further inspection, the version of the algorithm used in [4] was computing $U_r^{(j)} = Q^{(j)}Q_r^{(j)}$ at every iteration, which was unnecessary since the full basis was used only for matrix-vector multiplication. The cost of this extra computation was a time-complexity of $O(nmr^2)$ (worst-case $O(nm^3)$ if no reduction of order is attained). This step was the bottleneck for the majority of cases tested; once this step was bypassed, the computational cost of the method was reduced significantly (with a worst-case time-complexity of $O(nmr)$ for the matrix-vector multiplication).

In fact, as can be seen in Figure 4.30, the computation time after switching to the version of the algorithm given in 2.2.3 was reduced by more than an order of magnitude in the worst case tested, and was comparable to that of HAPOD with a 10% tolerance. However, when the reduction in the number of modes (see Figure 4.31) was less significant, as with the larger sample sets with 1% tolerance, there was still a significant increase in computational cost when compared to the distributed version of HAPOD. The live version of HAPOD saw a similar (though less pronounced) increase in cost. These increases stem from the fact that

the distributed version performs this truncated SVD only once on a matrix with a number of columns $m_i \approx m$, while the live HAPOD and incremental methods must perform many computation steps on smaller, but still quite large, systems. The cumulative cost of these smaller steps quickly surpasses that of the full-order computation when the truncated order is large (especially when more than half the modes are required to reach the required cutoff tolerance). In terms of computational complexity, the $O(mr^3)$ cost of the order $r \times r$ SVD at each step begins to dominate the $O(mnr)$ cost when all columns of S are incorporated, since r becomes comparable to m and $r^2 \approx 10n$ ($r = 791,\ n = 80401$).

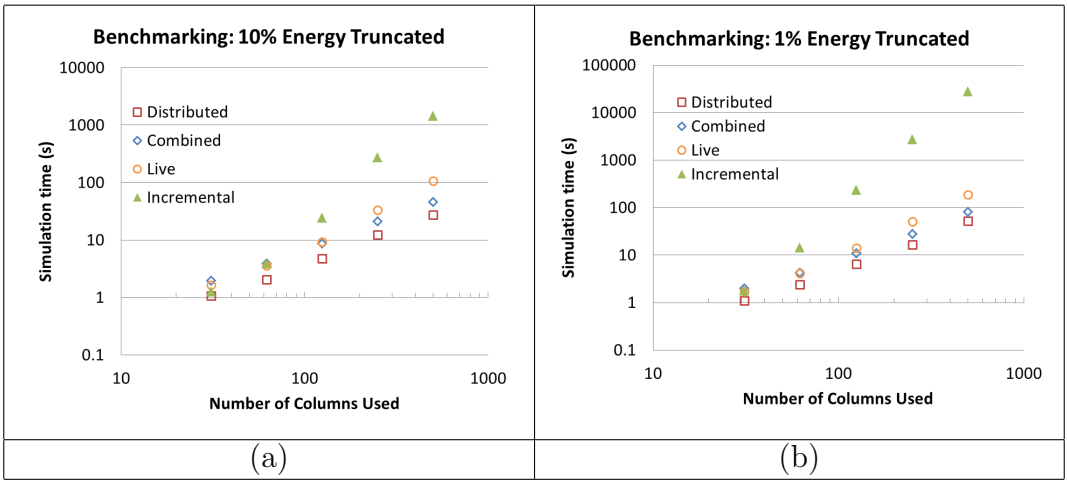


Figure 4.29: Benchmarking comparison of HAPOD to the original implementation of the Incremental SVD method [2, 4] using the first m columns of the snapshot matrix and truncating (a) 10% and (b) 1% of the total energy. The temperature simulation data was used for this comparison.

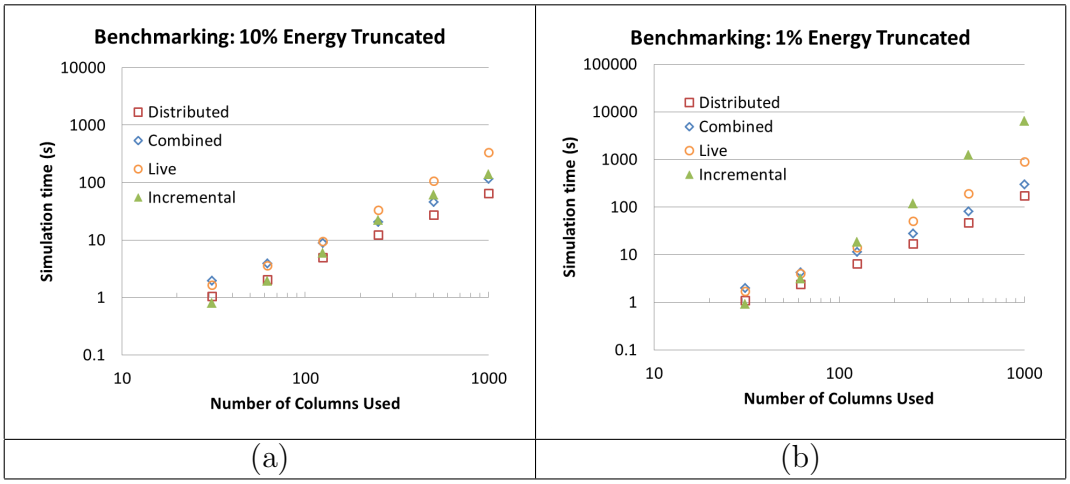


Figure 4.30: Benchmarking comparison of HAPOD to a modified implementation of the Incremental SVD method [2,4] using the first m columns of the snapshot matrix and truncating (a) 10% and (b) 1% of the total energy. The temperature simulation data was used for this comparison.

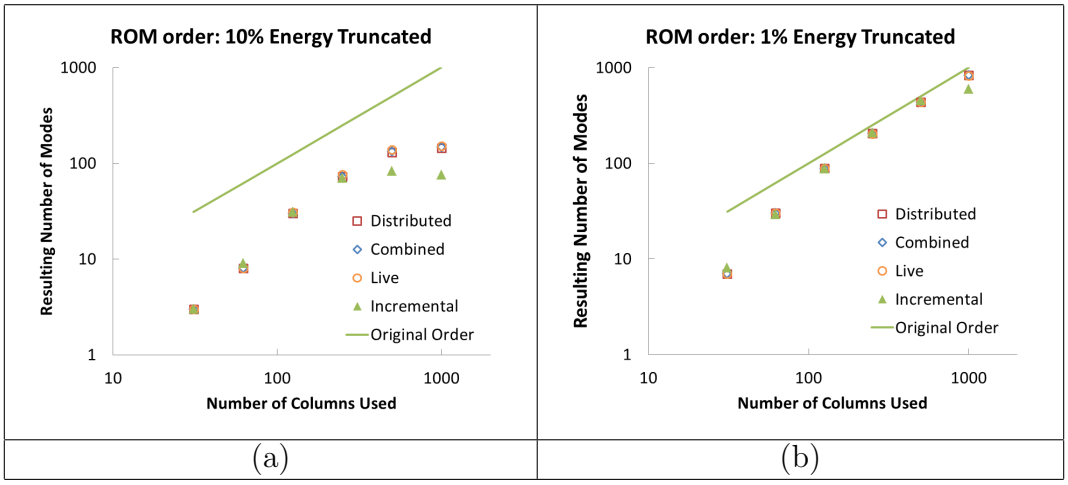


Figure 4.31: Reduced orders for the benchmarking comparison of HAPOD to an implementation of the Incremental SVD method [2,4] using the first m columns of the snapshot matrix and truncating (a) 10% and (b) 1% of the total energy. The temperature simulation data was used for this comparison.

Chapter 5

Conclusions

HAPOD is a method for the computation of the scaled left singular vectors of a snapshot matrix that is highly general, easily parallelizable, and can result in significant savings in terms of both simulation time and required memory usage. The potential for computational savings is corroborated by the numerical study of HAPOD using randomized synthetic data. These savings were found to depend heavily on the decay of singular values of the snapshot matrix, and to a lesser degree on how the expressions of the dominant modes change in time. Slowly-varying changes resulted, to an extent, in a more pronounced reduction in the number of modes at intermediate stages of HAPOD, leading to a cheaper overall simulation in terms of both simulation time and memory usage. Both the accuracy and computational savings of HAPOD were affected little (in a relative sense) by the dimensions of the snapshot matrix, all else equal.

In addition, from the benchmarking and accuracy studies using randomized synthetic data, it was found that a HAPOD weight of around $\omega = \frac{1}{\sqrt{2}}$ provided a good balance between the computational savings gained from HAPOD and the number of resulting extra modes. The user-selected tolerance was also crucial, as a tolerance that is too tight will result in little reduction in the number of modes of the snapshot matrix. The sensitivity of the computational cost and the final reduced order to the chosen tolerance depended mainly on the decay of singular values of the snapshot matrix: roughly speaking, a steep decay of singular values near a target tolerance resulted in a low sensitivity, while a shallow decay resulted in a much higher sensitivity. (Nonlocal information also plays a role in this sensitivity, as the cutoff is based on the sum of squares of singular values, so that (for example) a sustained flat region results in a much higher sensitivity than a brief flat region.)

Finally, in all cases, the performance of the combined HAPOD structure in all aspects was either between those of the live and distributed structures or better than both. However, given random matrices with an extremely steep singular value decay, the combined method

strongly outpaced both the live and distributed structures.

Bibliography

[1] Christian Himpe, Tobias Leibner, and Stephan Rave. Hierarchical Approximate Proper Orthogonal Decomposition. pages 1–28, 2016.

[2] M. Brand. Incremental singular value decomposition of uncertain data with missing values. In Anders; Heyden, Gunnar; Sparr, Mads; Nielsen, and Peter Johansen, editors, *Computer Vision - ECCV 2002*, volume 2350, pages 707–720. 2002.

[3] Matthew Brand. Fast online SVD revisions for lightweight recommender systems. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 37–46, San Francisco, CA, USA, 2003.

[4] William Arrighi, Geoffrey Oxberry, Tanya Vassilevska, and Kyle Chand. libROM User Guide and Design. Technical report, Lawrence Livermore National Laboratory, 2015.

[5] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[6] L. Mirsky. Symmetric gauge functions and unitarily invariant norms. *The Quarterly Journal of Mathematics*, 11(1):50–59, 1960.

[7] G H Golub and C F V Loan. *Matrix Computations*. Third edition, 1996.

[8] Gw Stewart. *Matrix Algorithms Volume 1: Basic Decompositions*. 1998.

[9] M. S. Tombs and I. Postlethwaite. Truncated balanced realization of a stable non-minimal state-space system. *International Journal of Control*, 46(4):1319–1330, 1987.

[10] Athanasios C Antoulas and Dan C Sorensen. Approximation of Large-Scale Dynamical Systems: An Overview. *International Journal of Applied Math and Computer Science*, 11(5):1093–1121, 2001.

[11] Serkan Gugercin and Athanasios C. Antoulas. A survey of model reduction by balanced truncation and some new results. *International Journal of Control*, 77(8):748–766, 2004.

[12] Serkan Gugercin, A. C. Antoulas, and Christopher Beattie. H2 Model Reduction for Large-Scale Linear Dynamical Systems. *SIAM Journal on Matrix Analysis and Applications*, 30(2):609–638, 2008.

[13] Christopher A. Beattie and Serkan Gugercin. A trust region method for optimal H2 model reduction. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 5370–5375, Shanghai, China, 2009.

[14] Athanasios C Antoulas, Christopher A Beattie, and Serkan Gugercin. Model reduction of large-scale dynamical systems. In Javad Mohammadpour and Karolos M. Grigoriadis, editors, *Efficient Modeling and Control of Large-Scale Systems*, pages 3–58. 2010.

[15] Peter Benner and A Schneider. Balanced truncation model order reduction for LTI systems with many inputs or outputs. In *Proceedings of the 19th International Symposium on Mathematical Theory of Networks and Systems*, pages 1971–1974, 2010.

[16] Peter; Benner and Tobias Breiten. Interpolation-Based H2-Model Reduction of Bilinear Control Systems. *SIAM Journal on Matrix Analysis and Applications*, 33(3):859–885, 2011.

[17] Peter Benner and Tobias Damm. Lyapunov Equations, Energy Functionals, and Model Order Reduction of Bilinear and Stochastic Systems. *SIAM Journal on Control and Optimization*, 49(2):686–711, 2011.

[18] Peter; Benner, Tobias; Breiten, and Tobias Damm. Generalized Tangential Interpolation for Model Reduction of Discrete-Time MIMO Bilinear Systems. *International Journal of Control*, 84(8):1398–1407, 2011.

[19] Carsten Hartmann, Boris Schafer-Bung, and Anastasia Thöns-Zueva. Balanced Averaging of Bilinear Systems with Applications to Stochastic Control. *SIAM Journal on Control and Optimization*, 51(3):2356–2378, 2013.

[20] Bjorn Gustavsen and Adam Semlyen. Rational approximation of frequency domain responses by vector fitting. *IEEE Transactions on Power Delivery*, 14(3):1052–1061, 1999.

[21] Christopher Beattie and Serkan Gugercin. Realization-independent H2-approximation. In *51st IEEE conference on Decision and Control*, pages 4953–4958, Maui, Hawaii, USA, 2012.

[22] Zlatko Drmac, Serkan Gugercin, and Christopher Beattie. Vector Fitting for Matrix-Valued Rational Approximation. *SIAM Journal on Scientific Computing*, 37(5):A2346–A2379, 2015.

[23] Zlatko Drmac, Serkan Gugercin, and Christopher Beattie. Quadrature-Based Vector Fitting for Discretized H2 Approximation. *SIAM Journal on Scientific Computing*, 37(2):A625–A652, 2015.

[24] Christopher Beattie and Serkan Gugercin. Interpolatory projection methods for structure-preserving model reduction. *Systems and Control Letters*, 58(3):225–232, 2009.

[25] Alan Martin Lattimer. *Model Reduction of Nonlinear Fire Dynamics Models*. PhD thesis, Virginia Polytechnic Institute and State University, 2016.

[26] Martin Redmann and Peter Benner. Model Reduction for Stochastic Systems. *Stochastic Partial Differential Equations: Analysis and Computations*, 3(3):291–338, 2014.

[27] René Pinnau. Model Reduction via Proper Orthogonal Decomposition. In Joost Schilders, Wilhelmus H. A.; van der Vorst, Henk A. ; Rommes, editor, *Model Order Reduction: Theory, Research Aspects and Applications*, pages 95–109. 2008.

[28] Saifon Chaturantabut and Danny C. Sorensen. Nonlinear Model Reduction via Discrete Empirical Interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010.

[29] B. Noack, M Morzy'nski, and G Tadmor. *Reduced-Order Modelling for Flow Control*. 2011.

[30] I Kalashnikova and M. F. Barone. Efficient non-linear proper orthogonal decomposition/Galerkin reduced order models with stable penalty enforcement of boundary conditions. *International Journal for Numerical Methods in Engineering*, 90(11):1337–1362, 2012.

[31] Jan S Hesthaven, Gianluigi Rozza, and Benjamin Stamm. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*. 2016.

[32] Geoffrey M. Oxberry, Tanya Kostova-Vassilevska, William Arrighi, and Kyle Chand. Limited-memory adaptive snapshot selection for proper orthogonal decomposition. *International Journal for Numerical Methods in Engineering*, 109(2):198–217, 2017.

[33] Christopher A. Beattie, Jeff Borggaard, Serkan Gugercin, and Traian Iliescu. A Domain Decomposition Approach to POD. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 6750–6756, San Diego, CA, USA, 2006.

[34] Vladimir Rokhlin, Arthur Szlam, and Mark Tygert. A randomized algorithm for principal component analysis. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1–26, 2009.

[35] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):1–74, 2011.

[36] Zlatko Drmac and Serkan Gugercin. A New Selection Operator for the Discrete Empirical Interpolation Method – improved a priori error bound and extensions. *SIAM Journal on Scientific Computing*, 38(2):A631–A648, 2016.

[37] B.S. Manjunath, S. Chandrasekaran, and Y.F. Wang. An eigenspace update algorithm for image analysis. In *Proceedings of International Symposium on Computer Vision - ISCV*, pages 551–556, 1995.

[38] S. Chandrasekaran, B.S. Manjunath, Y.F. Wang, J. Winkeler, and H. Zhang. An eigenspace update algorithm for image analysis. *Graphical models and image processing*, 59(5):321–332, 1997.

[39] Younes Chahlaoui, Kyle a Gallivan, and Paul Van Dooren. An Incremental Method for Computing Dominant Singular Spaces. In Michael W. Berry, editor, *SIAM Computational Information Retrieval*, pages 53–62. 2001.

[40] Avraham Levy and Michael Lindenbaum. Sequential Karhunen-Loeve basis extraction and ite Application to Images. *IEEE Transactions on Image Processing*, 9(8):1371–1374, 2000.

[41] Y. Chahlaoui, K. Gallivan, and P. Van Dooren. Recursive Calculation of Dominant Singular Subspaces. *SIAM Journal on Matrix Analysis and Applications*, 25(2):445–463, 2003.

[42] Christopher G. Baker. *A block incremental algorithm for computing dominant singular subspaces*. Master's thesis, Florida State University, 2004.

[43] Christopher G Baker, Kyle A Gallivan, and Paul Van Dooren. Low-rank incremental methods for computing dominant singular subspace. *Linear Algebra and its Applications*, 436(8):2866–2888, 2012.

[44] Joseph Boussinesq. *Théorie de l'écoulement tourbillonnant et tumultueux des liquides dans les lits rectilignes a grande section*. Paris, France, 1897.

Appendix A

Reproduction Details

The random matrices for the numerical studies were generated in two different ways and orders: one way in MATLAB, and another way in C. The non-benchmarking synthetic tests were performed in MATLAB, and the rand() function was used to generate the random matrices. Unfortunately, no random seed was set at the start of each set of trials, which were not run on a fresh MATLAB session, so the results from Section 4.2 are not exactly reproducible. However, the there was very little fluctuation in the behavior of HAPOD from trial to trial with the same parameters but different randomizations.

In C, the matrices were generated using the rand() function from the stdlib library in C in the following manner, which results in something very close to a uniform distribution of positive numbers between 0 and 1:

```
void generate_matrix(int n, int m, double* Sigma, double* S){
for(int i=0; i<n*m; ++i)
S[i] = (double) rand() / (RAND_MAX + 1.);
//Compute the SVD of the matrix S: S = Urand Srand Vrand^T.
        //This SVD is performed using the LAPACK (specify) function
//S = Urand*diag(Sigma)*Vrand^T
}
```

where $S \in \mathbb{R}^{m \times n}$ is stored in column-major format. Note that the computations were performed using the default random number seed in C. For the synthetic data tests, the parameters are iterated over in the following order to generate the random matrices:

```
dfloat *S = (dfloat) calloc(n*m, sizeof(dfloat*));
//Define integer array of p-values of size nps. Call it p.
for(p_i=0; p_i<nps; p_i++)
//Generate a vector of singular values Sigma based on p[p_i]
generate_matrix(n, m, Sigma, S);
//for each weight and tolerance, benchmark each
```

```
//        structure of HAPOD using this matrix S.
}
```

For the tests comparing the simulated data with the synthetic data, the parameters are iterated over in the following order:

```
dfloat *S = (dfloat) calloc(n*m, sizeof(dfloat*));
//Define integer array of m-values of size nms. Call it ms.
//Sigma contains the singular values computed from the
//        first m columns of the snapshot matrix S.
for(m_i=0; m_i<nms; m_i++)
generate_matrix(n, m, Sigma, S);
//for each tolerance, benchmark each structure of
//        HAPOD using this matrix S.
}
```

Finally, for the distributed and combined HAPOD structures, the slices of S are disseminated according to the structure defined below (in C-code):

```
//The snapshot matrix S is an n-by-m matrix, and NmpiProcs
        //contains the number of MPI processors.
//Note that the first slice is stored as column 0, due to
        //the indexing paradigm in C.

 int Nslices = ceil(sqrt(m))

 int colsPerSlice = m/Nslices;
 int extraCols = m%Nslices;

 int blocksPerMpiProc = Nslices/NmpiProcs;
 int extraBlocks = Nslices%NmpiProcs;
```

The first *extraCols* slices will then contain *colsPerSlice+1* consecutive columns of the snapshot matrix, and the rest will contain *colsPerSlice* columns. These columns are disseminated in order (so that the first block of columns is stored in slice 1, the next block is stored in slice 2, etc.) *blocksPerMpiProc+1* of these slices are then stored in the first *extraBlocks* MPI processors, and *ExtraBlocks* slices are stored in the rest. Again, these slices are disseminated in order.