Dynamic Mode Decomposition for Wave Propagation

Yannis Voet
École polytechnique fédérale de Lausanne, Switzerland
April 22, 2024

Abstract

Dynamic Mode Decomposition (DMD) is an outstanding tool for data driven analyses and reduced order modeling of large scale dynamical systems. The method has been successfully applied in a variety of disciplines including fluid dynamics, image processing, epidemiology and neuroscience. However, relatively little attention has been paid to wave type propagation phenomena in structural dynamics. In this article, we investigate the applicability of the method to such problems for snapshot data produced by high fidelity tensorized finite element simulations. Moreover, we exploit the Khatri-Rao product structure sometimes featured in the data to enhance existing DMD algorithms. Several numerical experiments confirm our findings and support the application of DMD to wave type equations.

1 Introduction

Dynamic Mode Decomposition (DMD) [1] has recently emerged as a promising tool for the analysis of large scale dynamical systems and has quickly gained a strong foothold in fluid dynamics. DMD is a purely data driven approach for identifying and extracting coherent structure from experimental or numerical data. It assumes that data is generated from an unknown operator $\mathbb{A}$, whose spectral information provides an (approximate) description of the underlying dynamics, at least within a certain time window. DMD further connects to the Koopman theory for nonlinear dynamical systems, which provides the theoretical backbone of the method [2].

While sometimes conceived as an extension of the proper orthogonal decomposition method (POD), the selection of POD and DMD modes is fundamentally different. According to Schmid [1], POD is built around an "energy ranking" of the modes based entirely on the singular values of the snapshot matrix storing the data, while DMD, instead, detects the relevant frequencies of the flow, regardless of whether they are associated to low or high energy modes. As a matter of fact, as argued in [1, 3], high energy modes may not always be physically relevant, especially not when combining data with different physical units. Indeed, in such cases, important spectral information might be lost when truncating the singular value decomposition.

From a more mathematical perspective, the POD basis is an orthonormal basis for the range of the snapshots and its truncation to the kth basis vector best approximates (in the least squares sense) the range of the snapshots over all kth dimensional subspaces [3]. On the contrary, DMD forms a Rayleigh quotient matrix by projecting the operator $\mathbb{A}$ into the POD basis and, according to Schmid [1], contains more information on the temporal evolution of dynamical processes. However, one must bear in mind that DMD is not foolproof and sometimes fails to extract spectral information even for simple cases. In particular, it has difficulties detecting translational and rotational invariances and capturing transient phenomena [4].

Nevertheless, the method stood out in numerous difficult applications and is particularly useful for dynamical processes driven by just a few dominant frequencies and modes. While originally proposed for fluid dynamics, applications of DMD have been flourishing and now encompass image and video processing, epidemiology and neuroscience to name just a few [4]. However, to our knowledge, relatively little attention has been paid to wave propagation phenomena in structural dynamics. In this article, we initiate its application to dynamical systems stemming from high fidelity tensorized finite element discretizations of wave type equations. The tensorized nature of the finite element space often allows to approximate each (reshaped) snapshot by a low-rank matrix (or tensor), a property at the heart of dynamical low-rank methods [5, 6]. Therefore, the snapshot matrix, obtained by gathering the vectorized snapshots, possesses a Khatri-Rao product structure. To our knowledge, this structure has never been exploited within the DMD framework. Our article fills the gap by carefully examining its advantages.

The outline of the rest of the article is as follows: section 2 first presents the underlying application and provides a refresh on tensorized finite element discretizations, from which the snapshot matrix inherits its structure. Section 3 then recalls the fundamental ideas underpinning DMD and sets the scene for the developments of the rest of the section. In particular, we discuss the approximate computation of truncated SVDs, streaming QR factorizations and review the solution of structured least squares problems. All methods avoid forming the snapshot matrix explicitly, which is generally infeasible for large scale computations. We further highlight the difficulties of reconstructing the solution of second order (and more generally high order) dynamical systems and suggest a simple workaround. Section 4 presents a sample of numerical experiments ranging from academic examples to more realistic applications. Finally, section 5 summarizes our findings and advises on directions for future work.

2 Isogeometric analysis

The data used within this article stems from high fidelity numerical simulations of certain time-dependent partial differential equations (PDEs) governing wave type phenomena. The prototypical example of such problems is the wave equation. Let $\Omega \subset \mathbb{R}^d$ be an open connected domain with Lipschitz boundary and let $I = [0, T]$ be the time domain with $T > 0$ denoting the final time. We look for $u\colon \Omega \times [0, T] \to \mathbb{R}$ such that

$$\begin{aligned} \partial_{tt} u - \nabla \cdot (c^2 \nabla u) &= f && \text{in } \Omega \times (0, T], \\ u &= g_D && \text{on } \partial\Omega_D \times (0, T], \\ c^2 \nabla u \cdot \mathbf{n} &= g_N && \text{on } \partial\Omega_N \times (0, T], \\ u &= u_0 && \text{in } \Omega, \\ \partial_t u &= v_0 && \text{in } \Omega, \end{aligned} \tag{2.1}$$

where u_0 and v_0 are two initial conditions, g_D and g_N are Dirichlet and Neumann boundary conditions, respectively, and $\partial\Omega_D$ and $\partial\Omega_N$ form a disjoint partition of the boundary (i.e. $\overline{\partial\Omega_D} \cup \overline{\partial\Omega_N} = \overline{\partial\Omega}$ and $\partial\Omega_D \cap \partial\Omega_N = \emptyset$). The positive valued function c represents the wave velocity. Similar looking PDEs arise in structural dynamics, where the unknown is a vector valued displacement field and the differential operator is the divergence of a stress tensor (see e.g. [7]). Finite element discretization methods are based on the weak (or integral) form of the PDE and approximate the solution in a finite dimensional subspace V_h. Isogeometric analysis consists in choosing spline functions (i.e. piecewise polynomials) from computer-aided-design (CAD) such as B-splines both for representing the approximate solution and describing the geometry [8, 9]. Such functions follow a standardized construction in a so-called parametric domain $\hat{\Omega} = (0, 1)^d$ before being defined in the physical domain Ω. In dimension $d = 1$, the B-spline basis $\{\hat{B}_i\}_{i=1}^n$ is constructed recursively from the Cox-de Boor formula [10]. In dimension $d \geq 2$, the basis functions are defined by the tensor product

$$\hat{B}_i = \hat{B}_{\mathbf{i}} = \hat{B}_{1 i_1} \hat{B}_{2 i_2} \dots \hat{B}_{d i_d}$$

where $\hat{B}_{lj}$ denotes the jth function in the lth direction and $1 \leq i \leq n := \prod_{l=1}^d n_l$ is a global index which only depends on $\mathbf{i} = (i_1, \dots, i_d)$ and $\mathbf{n} = (n_1, \dots, n_d)$. The integers n_i are the space dimensions in each parametric direction. With a slight abuse of notation, we have identified the multi-index $\mathbf{i} = (i_1, i_2, \dots, i_d)$ with the "linear" index i in the global numbering.

In the isogeometric paradigm, the basis functions also describe the geometry via the spline parametrization $F\colon \hat{\Omega} \to \Omega$, which maps the parametric domain to the physical domain. Geometries described by such a map are called *single-patch*. The basis functions over the physical domain are then defined as $B_i = \hat{B}_i \circ F^{-1}$ and the spline spaces over the parametric and physical domains are

$$\hat{V}_h = \operatorname{span}\{\hat{B}_{\mathbf{i}}\colon \mathbf{1} \leq \mathbf{i} \leq \mathbf{n}\} \quad \text{and} \quad V_h = \operatorname{span}\{B_{\mathbf{i}}\colon \mathbf{1} \leq \mathbf{i} \leq \mathbf{n}\},$$

respectively, where $\mathbf{1}$ is the vector of all ones and vector inequalities are understood componentwise.

A Galerkin discretization of the spatial variables in the weak form (see for instance [11, 12]) now leads to solving the semi-discrete problem

$$\begin{aligned} M\ddot{\mathbf{u}}(t) + K\mathbf{u}(t) &= \mathbf{f}(t) \qquad \text{for } t \in [0, T], \\ \mathbf{u}(0) &= \mathbf{u}_0, \\ \dot{\mathbf{u}}(0) &= \mathbf{v}_0. \end{aligned} \tag{2.2}$$

where $K, M \in \mathbb{R}^{n \times n}$ are the stiffness and mass matrices, respectively. For single-patch geometries, their entries are

$$K_{ij} = \int_\Omega c^2 \, \nabla B_i \cdot \nabla B_j \quad \text{and} \quad M_{ij} = \int_\Omega B_i \, B_j \qquad 1 \leq i, j \leq n \tag{2.3}$$

As with any other standard Galerkin method, K and M are both symmetric and while M is positive definite, K is generally only positive semidefinite (unless Dirichlet boundary conditions are prescribed on some portion of the boundary). In isogeometric analysis, M is additionally nonnegative owing to the pointwise nonnegativity of the B-spline basis functions. The time-dependent right-hand side vector $\mathbf{f}(t)$ in (2.2) accounts for the function f and potential non-homogeneous Neumann and Dirichlet boundary conditions. The vectors $\mathbf{u}_0$ and $\mathbf{v}_0$ are the coefficient vectors of suitable projections of the initial conditions into the spline space and $\mathbf{u}(t)$ is the coefficient vector of the approximate solution $u_h(\mathbf{x}, t)$ in the spline basis.

The B-spline functions also have a rational counterpart, called Non-Uniform Rational B-splines (NURBS), which enable the exact representation of a broader class of geometries (including conic sections). However, the range of

geometries they may describe is still far too limited for most industrial applications. For complex geometries, it may be necessary to divide the physical domain into N_p subdomains (or patches); i.e.

$$\Omega = \bigcup_{r=1}^{N_p} \Omega_r$$

where each subdomain (or patch) Ω_r is described by its own map $F_r \colon \hat{\Omega} \to \Omega_r$. Thus, a *multi-patch* geometry is just a collection of patches. The construction of spline spaces over multipatch geometries is rather straightforward, though the notation becomes more cumbersome and falls outside the scope of this introduction. One must only remember that patches in isogeometric analysis are analogous to elements in classical finite element discretizations. Thus, the computed solution is only C^0 at patch interfaces.

Thanks to the interplay between analysis and design, isogeometric analysis enables the exact representation of many common geometries, thereby often eliminating geometry approximation errors induced by triangulations. Its strength also lies in its ability to construct smooth approximation spaces of regularity up to C^{p-1} for p degree splines, whereas classical Lagrange finite elements only produce C^0 spaces. The greater smoothness translates into superior approximation properties [13, 14, 15] and is very welcome for certain applications where the physical quantities of interest (e.g. strain, stress) depend on the derivatives of the solution.

For discretizing (2.2) in time, scores of methods have been proposed, including the Newmark, Wilson-θ and Generalized α methods to name just a few. Most of them are commonly included in textbooks [7, 11], which the reader may consult for details. These time stepping schemes produce a sequence of snapshots $\{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_m\}$, which are stored along the column of an $n \times (m+1)$ matrix U, called the *snapshot matrix*. Its entries are generally over a field F but given the scope of our application, we restrict ourselves to real data (i.e. $F = \mathbb{R}$) while noting that the extension to $F = \mathbb{C}$ is straightforward. Although this snapshot matrix is generally completely dense, it can sometimes be very well approximated in a data sparse format, a property directly inherited from the tensor product nature of the basis functions. As a matter of fact, for some idealized problems on trivial geometries (e.g. the hypercube $(0,1)^d$), the snapshot matrix S can be very well approximated by a Khatri-Rao product; i.e. $U \approx U_1 \odot \cdots \odot U_d$, where $U_i \in \mathbb{R}^{n_i \times m}$. More generally, for single-patch geometries, it may still be very well approximated by the sum

$$U \approx \sum_{k=1}^{q} U_1^{(k)} \odot \cdots \odot U_d^{(k)}. \tag{2.4}$$

In other words, each snapshot is the vectorization of a low-rank matrix (or tensor in dimension $d \geq 3$). Analogously to other notions of rank, a matrix is said to have Khatri-Rao rank q if it can be expressed by a sum of no less than q Khatri-Rao products. This value is simply the maximum of the Kronecker (or tensor) rank of the snapshots stored along the columns of the matrix. Dynamical low-rank methods, for instance, directly produce the snapshot data in this format as well as more general tensor formats [16, 17]. Such methods become essential for large scale applications, when forming the snapshot matrix explicitly becomes infeasible due to the "curse of dimensionality". Indeed, storing U explicitly requires $O(m \prod_{i=1}^{d} n_i)$ while storing all factor matrices $U_i^{(k)}$ only requires $O(qm \sum_{i=1}^{d} n_i)$. The savings are considerable in high dimensions and for moderate q, as shown in fig. 2.1. Unfortunately, low-rank solvers were not available for this work and we instead derived an algorithm for approximating U by a Khatri-Rao rank q approximation. This algorithm is presented in appendix A. The Khatri-Rao product approximation is computed once and for all during a costly offline phase.

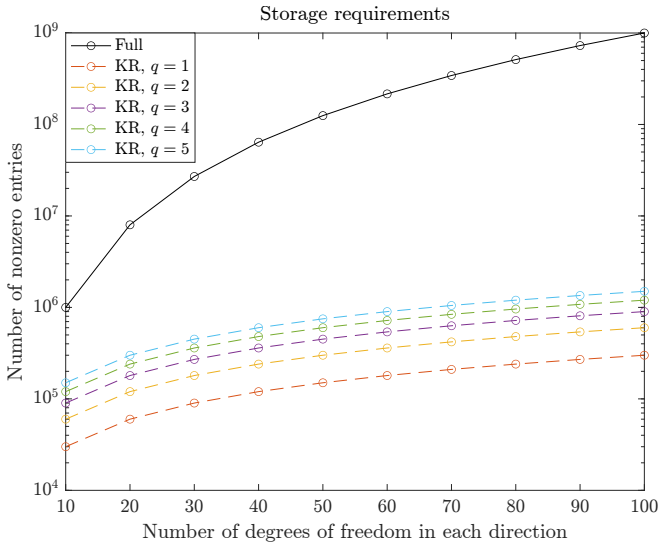

Figure 2.1: Comparison of the full storage of the snapshot matrix and its structured Khatri-Rao (KR) expression (2.4) for $d = 3$, $m = 1000$ and various values of q.

Although it arises very naturally in our context, we are not aware of any prior work related to Khatri-Rao structured snapshot matrices. In the rest of this article, we will show how this structure may be exploited to great advantages. For notational convenience, we sometimes only consider the case $d = 2$ or $d = 3$ and note that the extension to arbitrary dimensions d is straightforward.

3 Dynamic Mode Decomposition

In this section, we first recall some fundamental principles underlying DMD as well as the algorithms they led to and later discuss how to best exploit the structure of the snapshot matrix to speed them up.

3.1 Background

The Dynamic Mode Decomposition is based on the assumption that the snapshots, stored along the columns of $X_m \in \mathbb{R}^{n\times m}$ form a Krylov sequence

$$\{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{m-1}\} = \{\mathbf{u}_0, \mathbb{A}\mathbf{u}_0, \ldots, \mathbb{A}^{m-1}\mathbf{u}_0\}.$$

for some unknown linear operator $\mathbb{A}$ and a starting vector (or initial condition) $\mathbf{u}_0$. Thus, if X_m has full rank, it forms a basis for the Krylov subspace $\mathcal{K}_m(\mathbb{A}, \mathbf{u}_0)$. In practice, many of the snapshots may be nearly linearly dependent and only produce redundant information. Assuming the first m snapshots are linearly independent, we decompose $\mathbf{u}_m$ as

$$\mathbf{u}_m = X_m\mathbf{c} + \mathbf{r}$$

where $\mathbf{c}$ is the coefficient vector of the best approximation of $\mathbf{u}_m$ in $\mathcal{X}_m = \text{span}(X_m)$ and is obtained by solving the least squares problem

$$\min_{\mathbf{v}\in\mathbb{R}^m} \|X_m\mathbf{v} - \mathbf{u}_m\|_2.$$

Recalling that $\mathbb{A}\mathbf{u}_i = \mathbf{u}_{i+1}$ for $i = 0, \ldots, m-1$, we obtain the Krylov decomposition

$$AX_m = Y_m = X_mC_m + \mathbf{r}\mathbf{e}_m^T \tag{3.1}$$

where $\mathbf{e}_m \in \mathbb{R}^m$ is the mth canonical basis vector, $Y_m = [\mathbf{u}_1, \ldots, \mathbf{u}_m]$ is a shifted snapshot matrix and

$$C_m = \begin{pmatrix} 0 & 0 & \ldots & 0 & c_0 \\ 1 & 0 & \ldots & 0 & c_1 \\ 0 & 1 & \ldots & 0 & c_2 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & c_{m-1} \end{pmatrix}$$

is a companion matrix whose last column is the vector $\mathbf{c}$. Equation (3.1) is at the heart of the Krylov DMD method and the eigenpairs of C_m provide approximations to some of the eigenpairs of $\mathbb{A}$. Contrary to the Arnoldi method that constructs an orthonormal basis for the Krylov subspace by a sequence of projections, DMD only assumes availability of data, while the underlying operator that generated the data is unknown. Unfortunately, the Krylov DMD method is numerically unstable and Schmid [1] instead proposed computing the thin singular value decomposition (SVD) of the snapshots $X_m = U\Sigma V^T$ to form the Rayleigh quotient $S_m = U^T\mathbb{A}U = U^T Y_m V\Sigma^{-1}$, which also allowed for truncating the smallest singular values. As noted in [1], this Rayleigh quotient is formed by the projection of the linear operator $\mathbb{A}$ onto the POD basis U and is the core of Schmid's DMD method. Without truncating the POD basis, it is similar to C_m and therefore yields the same eigenvalues [3, Proposition 3.1]. Since Schmid's groundbreaking work, several variants and enhancements have been proposed in the literature. In [3], residual estimates were provided to only retain good approximations to the eigenpairs of $\mathbb{A}$ and a refined Rayleigh-Ritz (RRR) DMD algorithm was proposed by minimizing the residuals in span(U). In the same article [3], the authors presented a QR compressed DMD method that takes advantage of highly optimized QR factorization algorithms for compressing the problem to a lower dimensional space before employing SVD based DMD algorithms (e.g. Schmid or RRR). A streaming version of the same algorithm was later proposed by adding or removing snapshots and subsequently updating the QR factorization. Other noteworthy contributions include sparsity promoting and weighted DMD [3, 18]. The main algorithms proposed in the DMD literature are listed in table 3.1. A complete description of these algorithms is beyond the scope of this introduction and the interested reader may consult the references provided.

Algorithms	Brief description	References
DMD - Schmid	Schmid's DMD algorithm	[1] [3, Algorithm 1]
DMD - QR	QR compressed DMD	[3, Algorithm 3]
DMD - Krylov	Krylov DMD with Vandermonde matrix	[1]
DMD - RRR	Refined Rayleigh-Ritz DMD	[3, Algorithm 2]

Table 3.1: Main DMD algorithms and references

As we have seen, many DMD algorithms either require solving least squares problems or computing a truncated SVD or a (thin) QR decomposition of the snapshot matrix. However, forming the snapshot matrix itself is generally infeasible in our context, let alone its factorization. Thus, computing a few of its dominant singular values and vectors is the best one can hope for. Interestingly, most of the algorithms listed in table 3.1 support implicitly defined snapshot matrices, only available through matrix-vector products, provided there exists suitable alternatives to the truncated SVD or QR factorizations. In this section, we explore such alternatives and explain how to exploit the Khatri-Rao structure of the snapshot matrix to perform matrix-vector products implicitly without ever forming it. This constraint naturally leads to considering iterative methods for structured matrix computations.

3.2 Matrix products and properties

In this section, we recall the definitions of some basic matrix products and operators used throughout the article.

- *Kronecker product*: Given two matrices $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{p\times q}$, their Kronecker product is defined as

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1m}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \dots & a_{nm}B \end{pmatrix} \in \mathbb{R}^{np\times mq}.$$

- *Khatri-Rao product*: Given two matrices $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{p\times m}$, partitioned in columns as

$$A = [\mathbf{a}_1 \dots \mathbf{a}_m] \quad \text{and} \quad B = [\mathbf{b}_1 \dots \mathbf{b}_m],$$

  their Khatri-Rao product is defined as

$$A \odot B = [\mathbf{a}_1 \otimes \mathbf{b}_1 \dots \mathbf{a}_m \otimes \mathbf{b}_m] \in \mathbb{R}^{np\times m}.$$

- *Hadamard product*: Given two matrices $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{n\times m}$, their Hadamard (or elementwise) product is defined as

$$A * B = \begin{pmatrix} a_{11}b_{11} & \dots & a_{1m}b_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1}b_{n1} & \dots & a_{nm}b_{nm} \end{pmatrix} \in \mathbb{R}^{n\times m}.$$

- *Outer product*: Given two vector $\mathbf{a} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$, their outer product is defined as

$$\mathbf{a} \circ \mathbf{b} = \mathbf{a}\mathbf{b}^T = \begin{pmatrix} a_1 b_1 & \dots & a_1 b_m \\ \vdots & \ddots & \vdots \\ a_n b_1 & \dots & a_n b_m \end{pmatrix}.$$

- *Vectorization*: Given a matrix $A \in \mathbb{R}^{n \times m}$, partitioned as $A = [\mathbf{a}_1 \dots \mathbf{a}_m]$, the vectorization operator $\mathrm{vec}\colon \mathbb{R}^{n \times m} \to \mathbb{R}^{nm}$ is defined as

$$\mathrm{vec}(A) = \begin{pmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_m \end{pmatrix} := \mathbf{a}$$

  and $\mathrm{vec}^{-1}(\mathbf{a}) = A$ (provided m, n are known).

- *Diagonal*: Given a vector $\mathbf{x} = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$, the diagonal operator $\mathrm{diag}\colon \mathbb{R}^n \to \mathbb{R}^{n \times n}$ is defined as

$$\mathrm{diag}(\mathbf{x}) = \begin{pmatrix} x_1 & 0 & \dots & 0 \\ 0 & x_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & x_n \end{pmatrix}$$

  Moreover, it is understood that given $A \in \mathbb{R}^{n \times n}$, $\mathrm{diag}(A) = (a_{11}, \dots, a_{nn})^T$ extracts the diagonal of A.

3.3 Krylov based SVD

There exist several Krylov subspace methods for computing a few of the dominant singular values and vectors of a large sparse matrix. The Golub-Kahan Lanczos bidiagonalization method [19, 20], summarized in algorithm 3.1, is one of them and is the workhorse behind MATLAB's `svds` function [21][1].

Algorithm 3.1 Golub-Kahan Lanczos bidiagonalization

Input: Matrix $A \in \mathbb{R}^{n \times m}$, unit norm starting vector $\mathbf{p}_1 \in \mathbb{R}^m$, number of iterations $s \in \mathbb{N}$.
Output: Approximate truncated SVD $A \approx U_s \Sigma_s V_s^T$.

Set $P_1 = \mathbf{p}_1$, $\mathbf{q}_1 = A\mathbf{p}_1$, $\alpha_1 = \|\mathbf{q}_1\|$, $\mathbf{q}_1 = \mathbf{q}_1/\alpha_1$, $Q_1 = \mathbf{q}_1$.
for $j = 1, \dots, s$ **do**
  $\mathbf{r}_j = A^T\mathbf{q}_j - \alpha_j \mathbf{p}_j$
  $\mathbf{r}_j = \mathbf{r}_j - P_j P_j^T \mathbf{r}_j$ ▷ Full reorthogonalization
  $\beta_j = \|\mathbf{r}_j\|$
  $\mathbf{p}_{j+1} = \mathbf{r}_j/\beta_j$
  $P_{j+1} = [P_j \ \mathbf{p}_{j+1}]$
  $\mathbf{q}_{j+1} = A\mathbf{p}_{j+1} - \beta_j \mathbf{q}_j$
  $\mathbf{q}_{j+1} = \mathbf{q}_{j+1} - Q_j Q_j^T \mathbf{q}_{j+1}$ ▷ Full reorthogonalization
  $\alpha_{j+1} = \|\mathbf{q}_{j+1}\|$
  $\mathbf{q}_{j+1} = \mathbf{q}_{j+1}/\alpha_{j+1}$
  $Q_{j+1} = [Q_j \ \mathbf{q}_{j+1}]$
end for
Set $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_s)$
Set $\boldsymbol{\beta} = (\beta_1, \dots, \beta_{s-1})$
Set $B_s = \mathrm{diag}(\boldsymbol{\alpha}, 0) + \mathrm{diag}(\boldsymbol{\beta}, 1)$ ▷ Upper bidiagonal matrix
$B_s = F_s \Sigma_s G_s$ ▷ SVD of B_s
Set $U_s = Q_s F_s$
Set $V_s = P_s G_s$

Remark 3.1. Similarly to the Arnoldi method, algorithm 3.1 is always supplemented with restarting procedures to avoid the prohibitive growth of Krylov subspaces [21]. Moreover, it is sometimes possible to use partial reorthogonalization schemes to reduce the computational cost. However, these topics fall outside the scope of this contribution and we instead refer to the dedicated literature [20].

[1] According to the author's webpage: `https://www.math.uri.edu/meet/james-baglama/`

Algorithm 3.1 only requires matrix-vector multiplications with A and A^T. We explain in this section how to perform these operations implicitly for Khatri-Rao products. Without loss of generally, we assume $q = 1$ since the same operations can be performed individually on each term of the sum in (2.4). The case $d = 2$ is well-known and recalled below. Its proof is instructive and will help us generalize the properties to $d \geq 3$.

Proposition 3.2. Let $A \in \mathbb{R}^{n_1 \times m}$, $B \in \mathbb{R}^{n_2 \times m}$, $\mathbf{x} \in \mathbb{R}^m$, $Y \in \mathbb{R}^{n_2 \times n_1}$ and $\mathbf{y} = \operatorname{vec}(Y)$. Then,

1. $(A \odot B)\mathbf{x} = \operatorname{vec}(B \operatorname{diag}(\mathbf{x}) A^T)$,
2. $(A \odot B)^T \mathbf{y} = \operatorname{diag}(B^T Y A)$.

Proof. We prove each property below.

1. Since $B \operatorname{diag}(\mathbf{x}) A^T = \sum_{j=1}^m x_i \mathbf{b}_j \mathbf{a}_j^T = \sum_{j=1}^m x_i (\mathbf{b}_j \circ \mathbf{a}_j)$, we immediately obtain

$$\operatorname{vec}(B \operatorname{diag}(\mathbf{x}) A^T) = \sum_{j=1}^m x_i (\mathbf{a}_j \otimes \mathbf{b}_j) = (A \odot B)\mathbf{x}.$$

2. Firstly, $\operatorname{diag}(B^T Y A) = (\mathbf{b}_1^T Y \mathbf{a}_1, \ldots, \mathbf{b}_m^T Y \mathbf{a}_m)^T$. Secondly, $\mathbf{b}_j^T Y \mathbf{a}_j = \operatorname{vec}(\mathbf{b}_j^T Y \mathbf{a}_j) = (\mathbf{a}_j^T \otimes \mathbf{b}_j^T) \operatorname{vec}(Y)$. Thus,

$$\operatorname{diag}(B^T Y A) = \begin{pmatrix} (\mathbf{a}_1^T \otimes \mathbf{b}_1^T)\mathbf{y} \\ \vdots \\ (\mathbf{a}_m^T \otimes \mathbf{b}_m^T)\mathbf{y} \end{pmatrix} = (A \odot B)^T \mathbf{y}.$$

□

For the second property, needless to say that the matrix $B^T Y A$ is never formed explicitly. Only its diagonal entries are computed. Even so, we must emphasize that proposition 3.2 merely provides equivalent expressions to avoid forming the Khatri-Rao product explicitly. While they certainly save up on storage, they do not save up on operations. Before generalizing these properties to $d \geq 3$, we need a suitable generalization of the vectorization and diagonal operators as well as the outer product of several vectors. These notions are well-known (see e.g. [22]) but recalled below for the sake of completeness.

Definition 3.3 (vectorization)**.** Let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ be a dth order tensor. The vectorization operator $\operatorname{vec}\colon \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d} \to \mathbb{R}^{n_1 n_2 \ldots n_d}$ stacks the entries of $\mathcal{X}$ in reverse lexicographical ordering in a column vector; i.e.

$$\operatorname{vec}(\mathcal{X}) = \begin{pmatrix} x_{11\ldots1} \\ x_{21\ldots1} \\ \vdots \\ x_{n_1 1\ldots1} \\ x_{12\ldots1} \\ \vdots \\ x_{n_1 n_2 \ldots n_d} \end{pmatrix}$$

Although the lexicographical ordering produces more elegant expression in our context (see appendix A), the reverse lexicographical ordering is commonly adopted and we stick to this convention.

Definition 3.4 (diagonal)**.** Let $\mathcal{X} \in \mathbb{R}^{n \times n \times \cdots \times n}$ be a dth order tensor. The diagonal operator $\operatorname{diag}\colon \mathbb{R}^{n \times n \times \cdots \times n} \to \mathbb{R}^n$ stacks the diagonal entries of $\mathcal{X}$ in a column vector; i.e.

$$\operatorname{diag}(\mathcal{X}) = \begin{pmatrix} x_{11\ldots1} \\ x_{22\ldots2} \\ \vdots \\ x_{nn\ldots n} \end{pmatrix}$$

The next definitions generalize the notions of outer and inner products.

Definition 3.5 (outer product)**.** Let $\mathbf{a}_i \in \mathbb{R}^{n_i}$ for $i = 1, \ldots, d$. Their outer product $\mathcal{X} = \mathbf{a}_1 \circ \cdots \circ \mathbf{a}_d$ is the $n_1 \times \cdots \times n_d$ tensor such that

$$x_{i_1 \ldots i_d} = a_{1 i_1} \ldots a_{d i_d}.$$

Definition 3.6 (inner product)**.** Let $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ be two dth order tensors. Their inner product is defined as

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} x_{i_1 \ldots i_d} y_{i_1 \ldots i_d}.$$

The Khatri-Rao product of d matrices is closely connected to a special type of tensor, which we define next.

Definition 3.7 (CP format)**.** Let $A_i = [\mathbf{a}_1^{(i)} \ \ldots \ \mathbf{a}_r^{(i)}] \in \mathbb{R}^{n_i \times r}$ for $i = 1, \ldots, d$ and $\boldsymbol{\lambda} \in \mathbb{R}^r$. A tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ in Canonical Polyadic (CP) format is defined as

$$\mathcal{X} = [\![\boldsymbol{\lambda}; A_1, \ldots, A_d]\!] = \sum_{j=1}^{r} \lambda_j (\mathbf{a}_j^{(1)} \circ \cdots \circ \mathbf{a}_j^{(d)}).$$

If r is the smallest number of terms in the sum, the tensor is said to have CP rank r.

Finally, a generalization of the standard matrix product is useful.

Definition 3.8 (μ-mode multiplication)**.** The μ-mode multiplication of a tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ with a matrix $A \in \mathbb{R}^{n_\mu \times m}$ is a $n_1 \times \cdots \times n_{\mu-1} \times m \times n_{\mu+1} \times \cdots \times n_d$ tensor defined as

$$(\mathcal{X} \circ_\mu A)_{i_1 \ldots i_d} = \sum_{k=1}^{n_\mu} x_{i_1 \ldots i_{\mu-1} k i_{\mu+1} \ldots i_d} a_{k i_\mu}.$$

We are now ready to state the generalization of proposition 3.2 to arbitrary dimensions.

Proposition 3.9. Let $A_i \in \mathbb{R}^{n_i \times m}$ for $i = 1, \ldots, d$, $\mathbf{x} \in \mathbb{R}^m$, $\mathcal{Y} \in \mathbb{R}^{n_d \times \cdots \times n_1}$ and $\mathbf{y} = \mathrm{vec}(\mathcal{Y})$. Then,

1. $(A_1 \odot \cdots \odot A_d)\mathbf{x} = \mathrm{vec}([\![\mathbf{x}; A_d, \ldots, A_1]\!])$,
2. $(A_1 \odot \cdots \odot A_d)^T \mathbf{y} = \mathrm{diag}(\mathcal{Y} \circ_1 A_d \cdots \circ_d A_1)$.

Proof. We prove each property below with the same arguments as in proposition 3.2

1. By combining definitions 3.3 and 3.7, we immediately obtain

$$\mathrm{vec}([\![\mathbf{x}; A_d, \ldots, A_1]\!]) = \mathrm{vec}(\sum_{j=1}^{m} x_j (\mathbf{a}_j^{(d)} \circ \cdots \circ \mathbf{a}_j^{(1)})) = \sum_{j=1}^{m} x_j (\mathbf{a}_j^{(1)} \otimes \cdots \otimes \mathbf{a}_j^{(d)}) = (A_1 \odot \cdots \odot A_d)\mathbf{x}.$$

2. By combining definitions 3.4, 3.6 and 3.8, we obtain

$$\begin{aligned}
(\mathcal{Y} \circ_1 A_d \cdots \circ_d A_1)_{jj\ldots j} &= \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} y_{i_1 \ldots i_n} a_{i_1 j}^{(d)} \ldots a_{i_d j}^{(1)} \\
&= \langle \mathcal{Y}, \mathbf{a}_j^{(d)} \circ \cdots \circ \mathbf{a}_j^{(1)} \rangle \\
&= \langle \mathrm{vec}(\mathcal{Y}), \mathrm{vec}(\mathbf{a}_j^{(d)} \circ \cdots \circ \mathbf{a}_j^{(1)}) \rangle \\
&= \langle \mathbf{y}, \mathbf{a}_j^{(1)} \otimes \cdots \otimes \mathbf{a}_j^{(d)} \rangle \\
&= ((A_1 \odot \cdots \odot A_d)^T \mathbf{y})_j.
\end{aligned}$$

□

Remark 3.10. Although the relations in proposition 3.9 are mathematically elegant, our algorithms do not directly exploit them for $d \geq 3$. The main reason is that they involve tensor operations and all our implementations performed rather poorly in this setting. Actually, the Tensor Toolbox [23] itself uses the left-hand side of the first identity for forming the right-hand side, which we are precisely trying to avoid. In practice, both $(A_1 \odot \cdots \odot A_d)\mathbf{x}$ and $(A_1 \odot \cdots \odot A_d)^T \mathbf{y}$ were computed with a single `for` loop over m. In the second case, parallel computing capabilities may be leveraged.

It is well-know that the Lanczos bidiagonalization method converges quickly to the dominant, well-separated, singular values [20]. Thus, good approximations may be computed after only a few steps. The next section discusses another class of methods for achieving this goal.

3.4 Randomized SVD

Randomized algorithms are a valuable alternative when the snapshot matrix is known to be low-rank. This may be expected for certain classes of problems. The randomized SVD, presented in [24] and summarized in algorithm 3.2, may yield significant speedups without undermining the accuracy [24]. It is based on sketching and effectively reduces the cost for computing the SVD from $\mathcal{O}(nm^2)$ to $\mathcal{O}(nmr)$, where r is a user supplied target rank and should be slightly larger than the rank of A.

Algorithm 3.2 Randomized SVD

Input Matrix $A \in \mathbb{R}^{n\times m}$ where $n > m$ and $r \in \mathbb{N}$.
Output $U \in \mathbb{R}^{n\times r}, \Sigma \in \mathbb{R}^{r\times r}, V \in \mathbb{R}^{r\times m}$ such that $A \approx U\Sigma V^T$

1: Generate Gaussian random matrix $\Omega \in \mathbb{R}^{m\times r}$
2: Set $Y = A\Omega$
3: Compute $Y = QR$ ▷ QR factorization
4: Compute $Q^T Y = \widetilde{U}\Sigma V$ ▷ Economy size SVD
5: Set $U = Q\widetilde{U}$

3.5 Streaming QR factorization

Forming the full snapshot matrix explicitly is generally infeasible and obviously so is its QR factorization. However, it is possible to form a few columns at a time and update those columns if need be. If the QR factorization is computed for those initial columns, it must also be updated (see fig. 3.1). Streaming procedures are specifically designed for this purpose. Several variants have been proposed, including householder QR [25] and (modified) Gram-Schmidt. In this work, we have opted for Householder reflections for improved numerical stability and adapted the algorithm proposed in [25] to the streaming setting. The result, summarized in algorithm 3.3 keeps track of the Householder reflectors computed at each step for future compression. In comparison to naively recomputing the QR factorization each time snapshots are appended, the streaming QR factorization reduces the complexity from $\mathcal{O}(n^2(m + m_{\text{new}}))$ to $\mathcal{O}(nm_{\text{new}}(m + m_{\text{new}}))$.

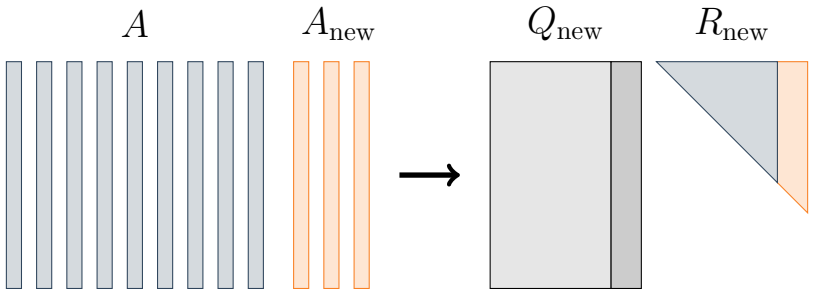

Figure 3.1: Streaming QR factorization. The matrix A_{new} is appended to A and the QR factorization is updated accordingly. The first part of the upper triangular matrix R_{new} remains unchanged.

Algorithm 3.3 Additive Householder QR

Input: Matrix $A_{\text{new}} \in \mathbb{C}^{n \times m_{\text{new}}}$ where $n > m_{\text{new}}$
Optional input: Matrices $E \in \mathbb{C}^{n \times m}$, $V \in \mathbb{C}^{n \times m}$, $Q \in \mathbb{C}^{n \times m}$, and $R \in \mathbb{C}^{m \times m}$ from previous executions of the algorithm on parts of the matrix A
Output: $Q_{\text{new}} \in \mathbb{C}^{n \times (m+m_{\text{new}})}$, and $R_{\text{new}} \in \mathbb{C}^{(m+m_{\text{new}}) \times (m+m_{\text{new}})}$ such that $Q_{\text{new}} R_{\text{new}} = [A, A_{\text{new}}]$

Add m_{new} columns to Q and V, and m_{new} columns and rows to R
Extend E with m_{new} orthogonal columns
for $k = m+1, m+2, \ldots, m+m_{\text{new}}$ **do**
    $a = A(:, k-m)$
    for $j = 1, 2, \ldots, k-1$ **do** ▷ Apply reflectors from previous iterations
        $a \leftarrow a - 2V(:,j)(V(:,j)^* a)$
        $R(j,k) = E(:,j)^* a$
        $a = a - R(j,k)E(:,j)$
    end for
    $R(k,k) \leftarrow \|a\|$
    $\alpha \leftarrow E(:,k)^* a$
    if $\alpha \neq 0$ **then**
        $E(:,k) \leftarrow E(:,k)(-\alpha/|\alpha|)$
    end if
    $V(:,k) \leftarrow R(k,k)E(:,k) - a$
    $V(:,k) \leftarrow V(:,k) - E(:,1:k-1)(E(:,1:k-1)^* V(:,k))$ ▷ Execute twice to reorthogonalize
    $\sigma \leftarrow \|V(:,k)\|$
    if $\sigma \neq 0$ **then**
        $V(:,k) \leftarrow V(:,k)/\sigma$
    else
        $V(:,k) \leftarrow E(:,k)$
    end if
    for $j = k, k-1, \ldots, 1$ **do** ▷ Apply reflections to Q
        $Q(:,k) = Q(:,k) - 2V(:,j)(V(:,j)^* Q(:,k))$
    end for
end for

3.6 Khatri-Rao least squares problems

After addressing SVD computations for structured matrices and streaming QR factorizations, we now move to the solution of least squares problems, another central topic within DMD (see section 3.1). Quite some attention has been devoted to Khatri-Rao structured least squares problems as they typically arise when computing the CP approximation of a tensor [20, 22] (see also [26] for other occurrences). More recently they have also surfaced within DMD for (weighted) snapshots reconstruction [18]. The next proposition states a well-know result, whose short proof will be very instructive for explaining the difficulties. The identities are stated here for $d = 2$ but their extension to arbitrary dimensions is straightforward.

Proposition 3.11. For arbitrary matrices A, B, C and D such that the products AB and CD are compatible

$$AB \odot CD = (A \otimes C)(B \odot D).$$

Proof. From the mixed-product property of the Kronecker product [27, Lemma 4.2.10], it is well-known that

$$AB \otimes CD = (A \otimes C)(B \otimes D). \tag{3.2}$$

Moreover, by definition, $AB \odot CD$ is a submatrix obtained by extracting a subset of the columns of $AB \otimes CD$. Thus, a factorization of $AB \odot CD$ is obtained from (3.2) by extracting the same subset from the columns of $(B \otimes D)$, that is $(B \odot D)$. □

Similarly to the Kronecker product, the Khatri-Rao product also has several elegant "SVD like" and "QR like" factorizations. Indeed, a direct application of proposition 3.11 shows that

$$A_1 \odot A_2 = (U_1 \otimes U_2)(\Sigma_1 \otimes \Sigma_2)(V_1^T \odot V_2^T), \tag{3.3}$$
$$A_1 \odot A_2 = (Q_1 \otimes Q_2)(R_1 \odot R_2), \tag{3.4}$$

where $U_i\Sigma_i V_i$ and $Q_i R_i$ are the (thin) SVD and QR decompositions of A_i, respectively. However, a quick inspection of the formula reveals that $\Sigma = \Sigma_1 \otimes \Sigma_2$ does not contain the singular values of $A_1 \odot A_2$ and $R_1 \odot R_2$ is not upper triangular. Equations (3.3) and (3.4) are merely matrix factorizations but are not the actual SVD or QR factorizations of $A_1 \odot A_2$. As a matter of fact, the entries of Σ are the singular values of $A_1 \otimes A_2$, of which $A_1 \odot A_2$ is a submatrix. It remains unclear if/how eqs. (3.3) and (3.4) may be exploited (if at all).

Instead, least squares solvers for Khatri-Rao products commonly rely on the normal equations and exploit the useful identity [18, Proposition 3.5]

$$(A_1 \odot A_2)^T (A_1 \odot A_2) = (A_1^T A_1 * A_2^T A_2). \tag{3.5}$$

One of the major drawbacks of the normal equations is the squaring of the condition number of $A_1 \odot A_2$ and the subsequent reduction in digits of accuracy. For Khatri-Rao products, this problem was extensively studied in [18]. Therein, the authors explain that the conditioning of the Hadamard product matrix may be significantly better than the conditioning of the individual matrices involved. Moreover, the suitability of the normal equations as a solver most critically depends on the conditioning of a certain correlation matrix, obtained by scaling the rows and columns by the square root of the (positive) diagonal entries. Provided that condition number is moderate, the solution is computable with a satisfactory level of accuracy. In case the condition number is too large, the authors suggest alternatives including a seminormal approach and a QR factorization based algorithm. That being said, in practice, (3.5) remains by far the most popular method for solving Khatri-Rao structured least squares problems. In our work, forming (3.5) became quite expensive for large Khatri-Rao ranks and we opted instead for an approximate truncated SVD (computed with algorithm 3.1).

Remark 3.12. If the snapshot matrix has Khatri-Rao rank 1, it might be worthwhile directly approximating the spectral decomposition of (3.5) with a Krylov subspace method, as was already investigated in [26] in the more general tensor setting.

3.7 Reconstruction: issues and remedies

DMD is built around the assumption that all snapshots are generated from successive applications of an operator $\mathbb{A}$. In that respect, a reconstruction based on the initial snapshot, as originally proposed, seems reasonable. However, in our context, that underlying assumption is often severely flawed. Indeed, the general solution of the semi-discrete problem (2.2) is given by

$$\mathbf{u}(t) = \cos(\sqrt{A}t)\mathbf{u}_0 + t\,\mathrm{sinc}(\sqrt{A}t)\mathbf{v}_0 + \int_0^t (t-\tau)\,\mathrm{sinc}(\sqrt{A}(t-\tau))M^{-1}\mathbf{f}(\tau)\,d\tau \tag{3.6}$$

where $A = M^{-1}K$. A detailed derivation is provided in appendix B (corollary B.2). Clearly, the general solution not only depends on the initial condition $\mathbf{u}_0$, but also on $\mathbf{v}_0$ and the right-hand side data. Thus, a reconstruction based only on the snapshot data for $U = [\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_m]$ is generally hopeless. As a matter of fact, if the initial condition $\mathbf{u}_0$ is zero, the standard reconstruction method will only return the zero solution and is practically useless (see examples 4.1 and 4.2 below). Even more advanced reconstruction methods employing all snapshot data might fail to produce useful results.

Fortunately, this issue is easily resolved by rewriting the second order system as an extended first order system, thereby edging closer toward the linear assumption underpinning DMD. This process, commonly known as linearization, consists in introducing additional variables related to the derivatives of the original unknown. In our case, denoting $\mathbf{v}(t) = \dot{\mathbf{u}}(t)$, system (2.2) is rewritten as

$$\begin{aligned} \begin{pmatrix} M & 0 \\ 0 & M \end{pmatrix} \begin{pmatrix} \dot{\mathbf{v}}(t) \\ \dot{\mathbf{u}}(t) \end{pmatrix} + \begin{pmatrix} 0 & K \\ -M & 0 \end{pmatrix} \begin{pmatrix} \mathbf{v}(t) \\ \mathbf{u}(t) \end{pmatrix} &= \begin{pmatrix} \mathbf{f}(t) \\ 0 \end{pmatrix} \qquad \text{for } t \in [0,T], \\ \begin{pmatrix} \mathbf{v}(0) \\ \mathbf{u}(0) \end{pmatrix} &= \begin{pmatrix} \mathbf{v}_0 \\ \mathbf{u}_0 \end{pmatrix} \end{aligned} \tag{3.7}$$

and is more compactly expressed as

$$\begin{aligned} \mathsf{M}\dot{\mathbf{w}}(t) + \mathsf{K}\mathbf{w}(t) &= \mathbf{f}(t) \qquad \text{for } t \in [0,T], \\ \mathbf{w}(0) &= \mathbf{w}_0, \end{aligned} \tag{3.8}$$

where

$$\mathsf{M} = I_2 \otimes M = \begin{pmatrix} M & 0 \\ 0 & M \end{pmatrix}, \quad \mathsf{K} = \begin{pmatrix} 0 & K \\ -M & 0 \end{pmatrix}, \quad \mathbf{f}(t) = \begin{pmatrix} \mathbf{f}(t) \\ 0 \end{pmatrix} \quad \mathbf{w}(t) = \begin{pmatrix} \mathbf{v}(t) \\ \mathbf{u}(t) \end{pmatrix} \quad \text{and} \quad \mathbf{w}_0 = \begin{pmatrix} \mathbf{v}_0 \\ \mathbf{u}_0 \end{pmatrix}.$$

Clearly, (3.8) may be recast to standard form $\dot{\mathbf{w}}(t) + \mathsf{A}\mathbf{w}(t) = \mathbf{g}(t)$ by pre-multiplying by M^{-1} but we will prefer (3.8) as it retains the structure inherited from finite element discretizations. Note that

$$\mathsf{A} = \mathsf{M}^{-1}\mathsf{K} = \begin{pmatrix} 0 & M^{-1}K \\ -I & 0 \end{pmatrix}$$

and thus, its spectrum $\Lambda(\mathsf{A}) = \Lambda(\mathsf{K}, \mathsf{M}) = \pm i\Lambda(K, M)$ is purely imaginary since $\Lambda(K, M)$ is real positive. Thus, the exact solution of (3.8) involving the matrix exponential does not diverge and is obviously equivalent to (3.6). System (3.8) is a first order system of ODEs and in this framework, the success of DMD seems much more likely. Obviously, the above argument can be easily generalized to any kth order linear system of ODEs. In practice, we run the DMD algorithms on the augmented snapshot matrix $[U;\ V]$ formed by appending to U the snapshot data $V = [\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_m]$. This is somewhat similar to using both the pressure and velocity for Navier-Stokes equations, as in [3]. However, this technique requires computing a data sparse representation for the augmented matrix, whose columns now combine data with different physical units. For this reason, we increment the dimension for the Khatri-Rao product approximation by one but this, unfortunately, also typically increases its rank. Nevertheless, snapshot augmentation is straightforwardly incorporated into existing DMD codes since it only requires changes to the input data.

Remark 3.13. A closer look at the Newmark time stepping scheme reveals that it may be possible to represent both U and V in a common basis formed by appending to the initial conditions $\mathbf{u}_0$ and $\mathbf{v}_0$ the snapshot data for the accelerations A also generated during the course of the iterations; i.e. $U = BS_1$ and $V = BS_2$, where $B = [\mathbf{u}_0\ \mathbf{v}_0\ A]$ and S_1 and S_2 are square upper triangular matrices. We then obtain the data sparse representation

$$\begin{pmatrix} U \\ V \end{pmatrix} = (I_2 \otimes B) \begin{pmatrix} S_1 \\ S_2 \end{pmatrix},$$

which yields significant savings if $B = B_1 \odot \cdots \odot B_d$. This format is currently under investigation by the authors.

Moreover, one must distinguish residual estimates returned by DMD algorithms from the quality of the reconstructed solution. Albeit simple, the following example is insightful. Consider the snapshot sequence $\{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_m\}$ produced by an operator $\mathbb{A}$ such that $\mathbf{u}_i = \mathbf{z}\phi(t_i)$ for some fixed vector $\mathbf{z} \in \mathbb{R}^n$ and for a function $\phi : [t_0, t_m] \to \mathbb{R}$. DMD algorithms will identify the only mode $\mathbf{z}$ along with an eigenvalue $\lambda \in \mathbb{R}$ with a zero residual. Yet, the reconstructed solutions read

$$\hat{\mathbf{u}}_i = \mathbf{z}\alpha\lambda^i \quad i = 0, \ldots, m,$$

and might arbitrarily depart from the original snapshots, regardless of how well the coefficient α has been computed. This simple example shows the discrepancy between residual estimates and the reconstructed solution, which is the true quantity of interest.

4 Numerical experiments

This section gathers a few numerical experiments comparing the standard DMD algorithms to their data sparse counterparts. The problems span various levels of difficulty and range from academic level to real world applications. All examples were generated with GeoPDEs, a MATLAB-Octave software package for isogeometric analysis [28]. We test and compare several algorithms proposed in the DMD literature and listed in table 3.1. The standard (truncated) SVD in Schmid's DMD and RRR algorithms may be substituted for cheaper variants discussed in section 3, including the Lanczos bidiagonalization method (algorithm 3.1) and the randomized SVD (algorithm 3.2). This will always be indicated in parentheses after the method's name. All algorithms were adapted to support implicitly provided snapshot data (i.e. only available through matrix-vector or matrix-matrix multiplications), except for the QR DMD algorithm. Moreover, since zero or near zero columns are a real possibility in our setting, we avoid rescaling the columns in the RRR algorithm, although originally advocated in [3]. All algorithms were implemented in MATLAB R2023a and run on MacOS with an M1 chip and 32 GB of RAM.

4.1 Single-patch geometries

This section is devoted to single-patch geometries (see section 2), for which our data sparse framework is applicable. We first provide a short description of the examples used in this section. They all arise from isogeometric discretizations of wave type equations and the snapshot matrix is generated from an implicit Newmark time stepping scheme.

Example 4.1 (2D standing wave)**.** The first example is the academic example of a standing wave on a unit square, described by $u(x, y, t) = \sin(2\pi x)\sin(2\pi y)\sin(2\pi t)$ for $t \in [0, 1]$. The initial and boundary conditions of the PDE are computed accordingly. Two snapshots of the discretized solution are shown in fig. 4.1. The snapshot matrix in this example is 1089×201 but we will consider throughout this section different levels of refinement in space and time to illustrate different properties of the method.

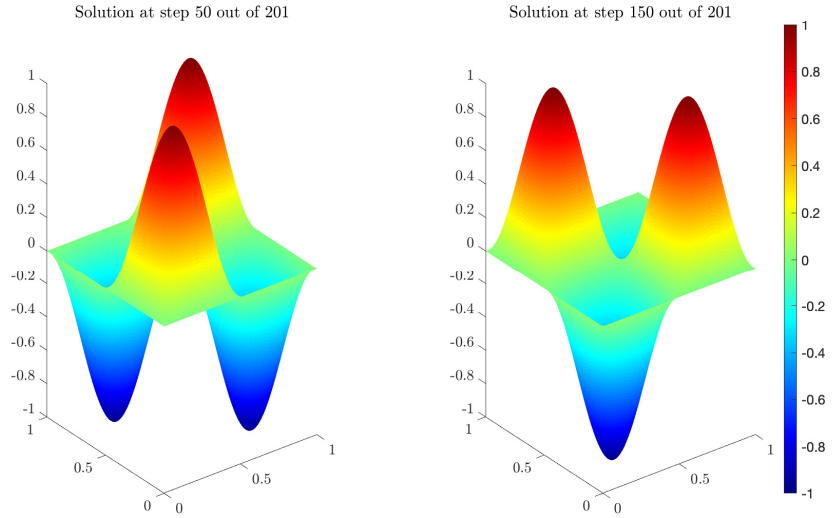

Figure 4.1: Standing wave on the unit square, discretized with cubic B-splines and 30 subdivisions in each direction

Example 4.2 (3D standing wave)**.** The second example is simply the 3D counterpart of the first one and provides a simple and yet conclusive example of the "curse of dimensionality"; i.e. the prohibitive (exponential) growth of memory and operations with the dimension. The exact solution is $u(x, y, z, t) = \sin(2\pi x)\sin(2\pi y)\sin(2\pi z)\sin(2\pi t)$ for $t \in [0, 1]$. Two snapshots of the discretized solution, shown along slices of the unit cube, are displayed in fig. 4.2. The snapshot matrix for this problem is 35937×201; i.e. 33 times larger than its 2D counterpart.

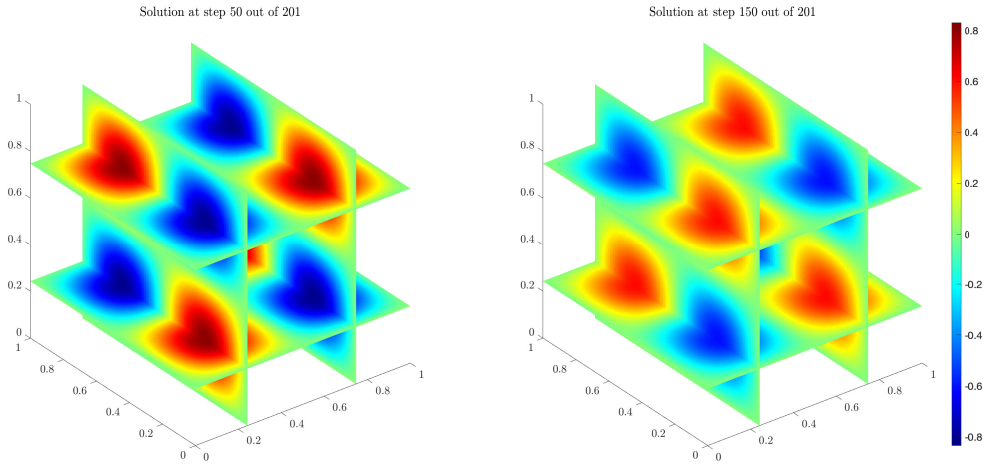

Figure 4.2: Standing wave on the unit cube, discretized with cubic B-splines and 30 subdivisions in each direction

Example 4.3. Our third example, inspired from [29], is more realistic and models the acoustic wave equation (2.1) in a non-homogeneous medium. The wave speed of the medium is $c(x, y) = 1 + y$. We consider homogeneous Neumann boundary conditions over the entire boundary and initial conditions $u_0(x, y) = e^{-124x^2}$ and $v_0(x, y) = 0$. The Gaussian pulse travels from left to right in the medium before hitting the right boundary and rebounding. Figure 4.3 shows a few snapshots of the solution. The size of the snapshot matrix is 62025×1001. The geometry was discretized with C^4 quintic splines and 250 subdivisions in each direction.

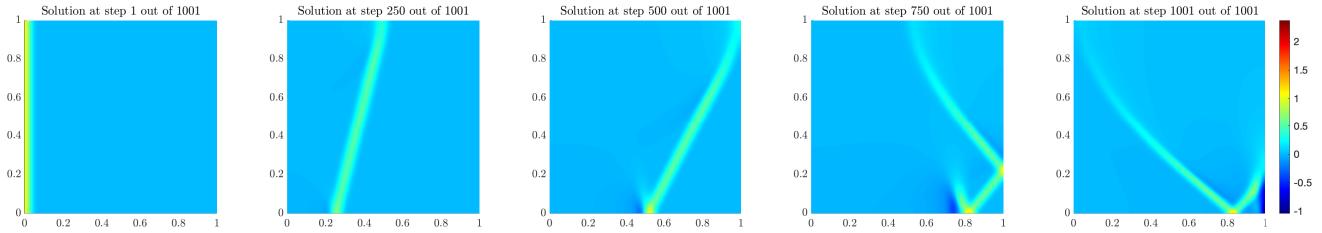

Figure 4.3: Solution snapshots

The ability to approximate the snapshot matrix by (sums of) Khatri-Rao products depends primarily on the solution itself and the features of the dynamics. The standing waves described in examples 4.1 and 4.2 are just a single wave modulated by a varying amplitude. Thus, tensorized finite element methods produce a nearly perfect Khatri-Rao rank 1 approximation; i.e. each snapshot is almost perfectly approximated by a Kronecker product of vectors and the approximation further improves for finer meshes. On the contrary, the Kronecker rank of the snapshots for the third example gradually increases as the simulation advances and so does the Khatri-Rao rank of the snapshot matrix, which in this example can be approximated by a Khatri-Rao rank 40 approximation with a relative error of 2.8×10^{-4}. This error is perfectly acceptable for our applications. However, the Khatri-Rao rank of the augmented snapshots increases to 80 for a similar accuracy level. Hence, alternative formats for the augmented snapshots might be valuable (see remark 3.13). In practice, a low-rank solver would produce the snapshots directly in low-rank format and approximating it by a Khatri-Rao product is unnecessary. Thus, we do not provide any timings for this operation. For completeness, the size of the (non-augmented) snapshot matrix of the different variants and its Khatri-Rao rank are summarized in table 4.1. The problem sizes still allow forming the snapshot matrix explicitly for the sake of comparison.

We compare the performance of the different methods, both in terms of accuracy and computing time. Our experiments include the Lanczos SVD and randomized SVD as alternatives to the truncated SVD and incorporated within Schmid's DMD. For the two first examples, the maximum number of iterations for the Lanczos SVD method and the target rank for the randomized SVD were both 10. They were increased to 100 for the last more difficult example, based on the errors obtained for the reconstructed solutions (see table 4.4 below). Both algorithms only require matrix-vector multiplications with the (augmented) snapshot matrix, which is provided implicitly. When comparing computing times, we distinguish the cases where the snapshot matrix is provided explicitly and implicitly. Moreover, since the Khatri-Rao product approximation of the augmented snapshot matrix artificially increments the dimension by one, we have run the experiments separately for U (table 4.2) and $[U;\ V]$ (table 4.3). For the latter, the values of n reported in table 4.1 must be doubled, while the Khatri-Rao rank q was capped at 40 just for the sake of this experiment. In practice, it should be chosen slightly larger to ensure an accurate data sparse approximation. However, we are not concerned with this step here and simply assume the data is provided in this format.

Example	n	m	q
Example 4.1 I	1089	200	1
Example 4.1 II	41209	1500	1
Example 4.2 I	79507	200	1
Example 4.2 II	79507	1000	1
Example 4.3	65025	1000	40

Table 4.1: Examples and problem sizes

Example	SVD	Lanczos SVD		Randomized SVD	
		Explicit	Implicit	Explicit	Implicit
Example 4.1 I	1.27×10^{-2}	6.81×10^{-4}	9.36×10^{-4}	1.02×10^{-3}	2.93×10^{-3}
Example 4.1 II	5.71×10^{0}	3.85×10^{-2}	2.11×10^{-2}	5.13×10^{-2}	2.71×10^{-2}
Example 4.2 I	4.01×10^{-1}	1.70×10^{-2}	1.73×10^{-1}	1.98×10^{-2}	1.52×10^{-1}
Example 4.2 II	5.37×10^{0}	4.61×10^{-2}	8.22×10^{-1}	6.40×10^{-2}	7.29×10^{-1}
Example 4.3	4.80×10^{0}	1.77×10^{0}	1.44×10^{1}	4.98×10^{-1}	1.44×10^{1}

Table 4.2: Timings (in seconds) for Schmid's DMD for the snapshot data U. Green and red cells identify the smallest and largest timings, respectively, for each example.

Example	SVD	Lanczos SVD		Randomized SVD	
		Explicit	Implicit	Explicit	Implicit
Example 4.1 I	2.12×10^{-2}	2.10×10^{-3}	1.50×10^{-1}	2.13×10^{-3}	3.58×10^{-2}
Example 4.1 II	1.09×10^{1}	9.27×10^{-2}	2.48×10^{0}	9.65×10^{-2}	1.80×10^{0}
Example 4.2 I	7.01×10^{-1}	8.50×10^{-2}	1.10×10^{0}	3.85×10^{-2}	2.97×10^{-1}
Example 4.2 II	9.83×10^{0}	2.99×10^{-1}	5.47×10^{0}	1.18×10^{-1}	1.55×10^{0}
Example 4.3	9.03×10^{0}	5.51×10^{0}	3.80×10^{2}	1.09×10^{0}	1.32×10^{1}

Table 4.3: Timings (in seconds) for Schmid's DMD for the snapshot data $[U;\ V]$. Green and red cells identify the smallest and largest timings, respectively, for each example.

Tables 4.2 and 4.3 indicate that our matrix-vector multiplication algorithms perform poorly in higher dimensions and are even sometimes slower than the truncated SVD. Due to MATLAB's highly optimized BLAS 2 and 3 operations, the algorithms perform much better with the explicitly formed Khatri-Rao product. Moreover, comparing the two last lines of the tables also indicates that the Khatri-Rao rank q heavily impacts the performance of the method. Nevertheless, we must stress that forming the snapshot matrix is generally infeasible in high dimensional settings.

We will now draw our attention to the reconstruction of the solutions. Table 4.4 compares the reconstruction errors for the same examples using the methods from table 3.1. We employ the standard reconstruction method based on the first snapshot and compute the reconstruction error as the mean elementwise error. In this experiment, the relative truncation tolerance for SVD based algorithms was set to 10^{-4}; i.e. the singular values and vectors retained satisfy $\frac{\sigma_i}{\sigma_1} \leq 10^{-4}$, where σ_1 is the largest singular value.

Examples	Schmid	QR	Krylov	RRR	Schmid (RSVD)	Schmid (Lanczos)
Example 4.1 I	6.46×10^{-5}	6.46×10^{-5}	8.54×10^{-5}	2.18×10^{-1}	6.46×10^{-5}	6.46×10^{-5}
Example 4.1 II	1.32×10^{-6}	1.32×10^{-6}	3.15×10^{-6}	2.50×10^{-1}	1.32×10^{-6}	1.32×10^{-6}
Example 4.2 I	9.23×10^{-2}	9.02×10^{-2}	1.47×10^{-1}	2.05×10^{-1}	9.23×10^{-2}	9.23×10^{-2}
Example 4.2 II	9.34×10^{-2}	9.29×10^{-2}	1.85×10^{-1}	2.06×10^{-1}	9.34×10^{-2}	9.34×10^{-2}
Example 4.3	1.20×10^{-2}	6.36×10^{-3}	6.88×10^{-11}	2.53×10^{0}	9.58×10^{-2}	1.38×10^{-1}

Table 4.4: Reconstruction errors for each example. Red cells identify the largest reconstruction errors.

Overall, Schmid's DMD and the QR method often yield similar results and so do cheaper variants based on the randomized and Lanczos SVD. Combined with tables 4.2 and 4.3, these results indicate that alternatives to the truncated SVD often yield similar accuracy at a reduced cost. Krylov's DMD is less robust over the range of examples tested and the reconstruction error fluctuates considerably. Surprisingly, the RRR method always resulted in the largest reconstruction error. Similar trends were obtained with the normal reconstruction method proposed in [18], although the results were slightly more accurate. Figure 4.4 compares a snapshot of the reconstructed solutions for Schmid's DMD and the RRR method to the original data (in Khatri-Rao format). While the reconstructed solution with Schmid's DMD is visually identical to the original, the one for the RRR method is utterly inaccurate and confirm our suspicions from table 4.4. However, inspecting fig. 4.5 reveals that the RRR method produced smaller residuals. Although we could not precisely pinpoint the reason for the method's failure, section 3.7 provides plausible causes. In particular, small residuals do not guarantee an accurate reconstruction. That being said, the RRR method still produced reasonably accurate results for the first two examples, despite the significantly larger reconstruction error. Nevertheless, we must emphasize that examples 4.1 and 4.2 are academic examples and are

exceedingly well described with only a couple of modes, which are precisely the standing waves depicted in figs. 4.1 and 4.2.

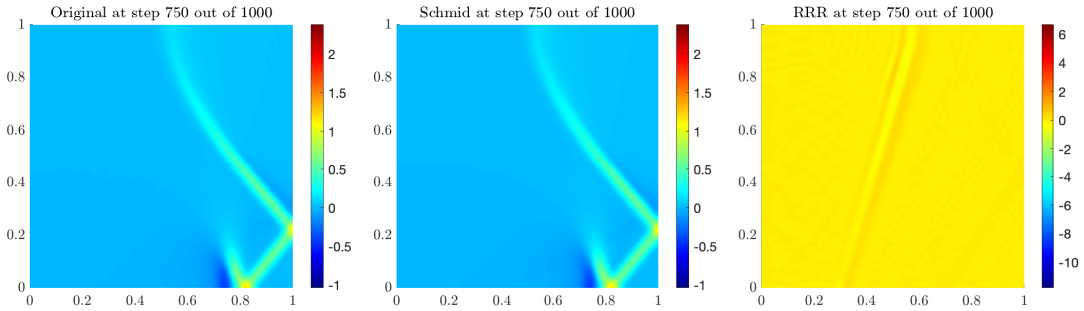

Figure 4.4: Reconstruction comparison for example 4.3

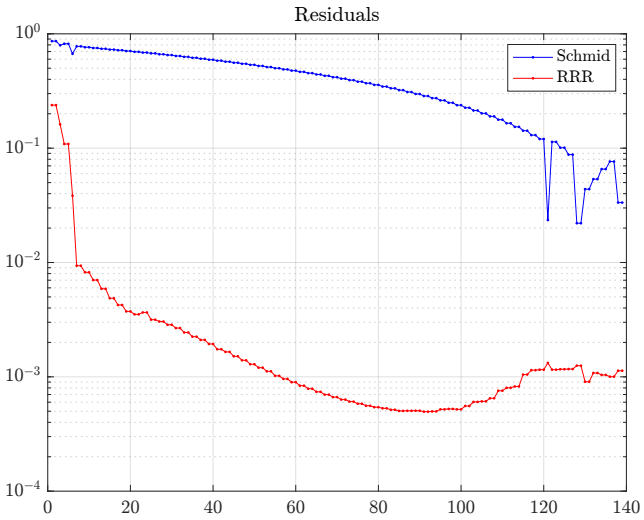

Figure 4.5: Residuals comparison for example 4.3

For example 4.3, the reconstructed solution obtained only from the snapshot data U was in all aspects similar to the one obtained from the augmented snapshot data $[U;\ V]$. As a matter of fact, the data in V probably does not add any information owing to the zero initial velocity in this example. Thus, in some cases, snapshot augmentation is not necessary for an accurate reconstruction.

4.2 Multi-patch geometries

We consider in this section multi-patch geometries (i.e. geometries described by a collection of patches). In such cases, the snapshot matrix cannot be approximated by (sums of) Khatri-Rao products and to our knowledge, data sparse formats for multi-patch geometries are nontrivial but are currently being investigated (see e.g. [17] for very recent work in that direction). Most problems of practical interest fall in this category and we only investigate in this section the applicability of DMD, independently of the availability of data sparse representations.

Example 4.4. Our first example is a slight variation of example 4.3, where a small cavity is added in the middle of the square. The homogeneous Dirichlet boundary conditions prescribed over its boundary create multiple reflections, which further complicate the dynamics. The initial conditions are $u_0(x,y) = e^{-30^2(x+0.6)^2}$ and $v_0(x,y) = 0$. The wave velocity is constant ($c = 1$) and all other data are set to zero. Snapshots of the solution are shown in fig. 4.6. The snapshot matrix for this problem has size 62000×1001.

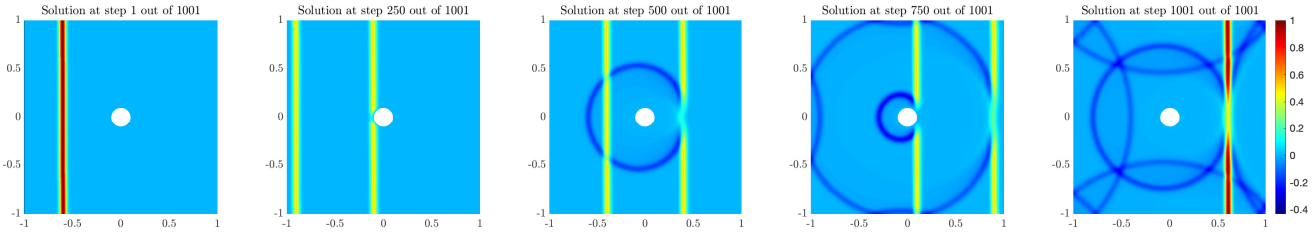

Figure 4.6: Solution snapshots

Example 4.5. Our second example is simpler and models a traction test, commonly employed in material science for measuring the resistance of rods. Non-homogeneous Dirichlet boundary conditions are imposed at both ends of the rod and in the initial phase of the test, the slowly varying displacement field follows a linear elastic model. Figure 4.7 shows the horizontal displacement field at different times. The snapshot matrix for this problem has size 3096×1501.

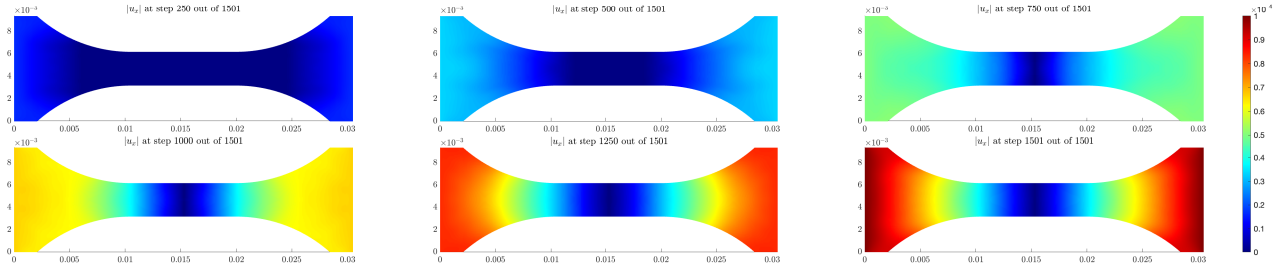

Figure 4.7: Solution snapshots

The size of the snapshot matrix for both examples is recalled in table 4.5. Although sparse data formats are not immediately available for those examples, alternatives to the (truncated) SVD are still valuable. In table 4.6, we provide timings for the augmented snapshot matrix $[U;\ V]$. As expected, both the Lanczos SVD and randomized SVD methods outperform the truncated SVD, for a maximum number of iterations and target rank of 100.

Example	n	m
Example 4.4	62000	1000
Example 4.5	3096	1500

Table 4.5: Examples and problem sizes

Examples	SVD	Lanczos SVD	Randomized SVD
Example 4.4	8.63×10^{0}	3.25×10^{0}	1.01×10^{0}
Example 4.5	2.55×10^{0}	3.05×10^{-1}	1.13×10^{-1}

Table 4.6: Timings (in seconds) for Schmid's DMD for the snapshot data $[U;\ V]$. Green and red cells identify the smallest and largest timings, respectively, for each example.

The reconstruction errors provided in table 4.7 once again indicate that the RRR method performs rather poorly, despite the small residuals shown in fig. 4.9. This finding is confirmed in fig. 4.8. The normal reconstruction led to far better results, especially for the Krylov method.

Examples	Schmid	QR	Krylov	RRR	Schmid (RSVD)	Schmid (Lanczos)
Example 4.4	4.27×10^{-4}	4.28×10^{-4}	1.16×10^{0}	9.20×10^{0}	2.53×10^{-1}	1.87×10^{-1}
Example 4.5	1.71×10^{-3}	1.69×10^{-3}	5.66×10^{-3}	3.55×10^{-1}	1.74×10^{-2}	1.55×10^{-2}

Table 4.7: Reconstruction errors for each example. Red cells identify the largest reconstruction errors.

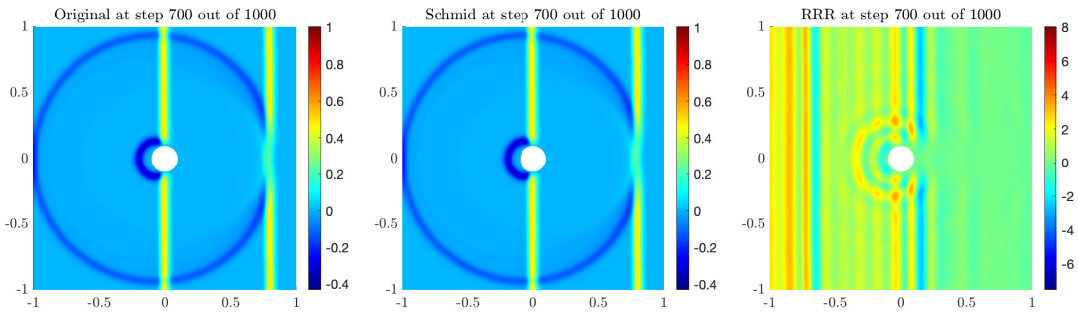

Figure 4.8: Reconstruction comparison for example 4.4

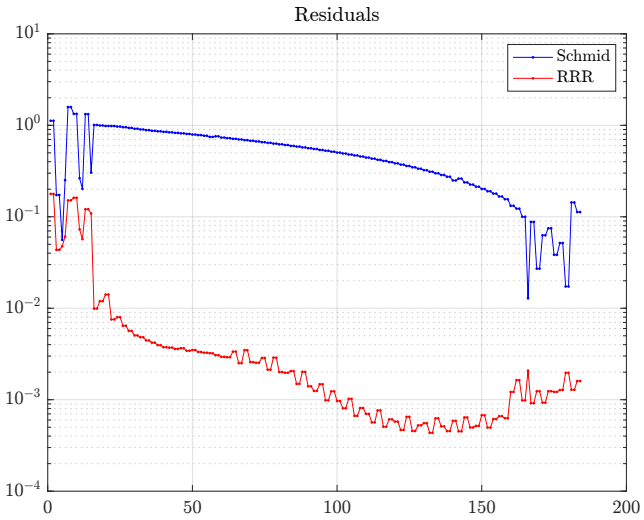

Figure 4.9: Residuals comparison for example 4.4

4.3 Down-sampling

A fine time discretization (i.e. a small time-step) may generate nearly identical snapshots and sub-sampling the snapshot matrix may sometimes be advantageous to reduce its size without compromising the accuracy of the subsequent computations. More formally, this process, referred to as *down-sampling*, consists in extracting a subset $\{\mathbf{u}_s, \mathbf{u}_{2s}, \ldots, \mathbf{u}_{ks}\}$ of the columns of U, where $s \in \mathbb{N}$ is the sampling step. The sampling step is problem dependent and if suitably chosen, it may speedup the algorithms by a factor s. For instance, fig. 4.10 reveals that the reconstruction error for example 4.3 remains at an acceptable level for a sampling step of about 10 for most methods. Unfortunately, we are not aware of any method for estimating s and postpone this issue to future work.

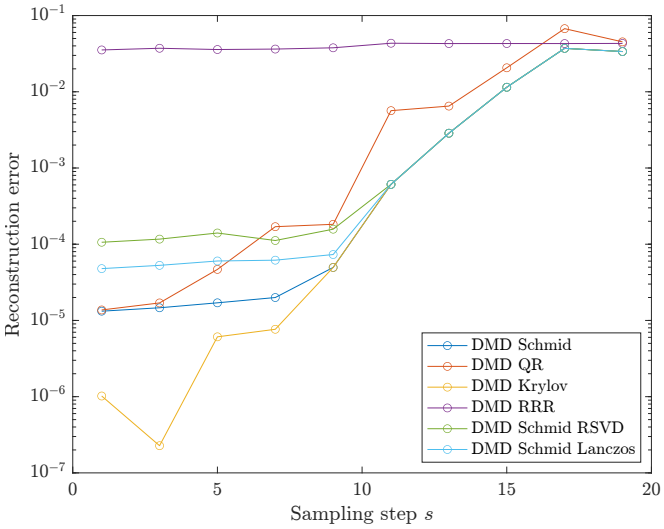

Figure 4.10: Reconstruction error for multiple DMD algorithms

4.4 Steaming QR

We have also tested the streaming QR factorization on example 4.3 and compared it to recomputing the QR factorization at each step. We restrict experimentation to the most relevant case in our setting, which is the addition of snapshots. Figure 4.11 shows the evolution of the reconstruction error for streaming QR and the standard QR always recomputed from scratch. The curves closely match and indicate that streaming QR does not have detrimental effects on the accuracy of the reconstructed solution. However, both curves dramatically increase after 25 snapshots, suggesting that an accurate reconstruction is only feasible for small time windows. Indeed, the dynamics become quite involved as the time window enlarges. Thus, an increase of the reconstruction error is expected.

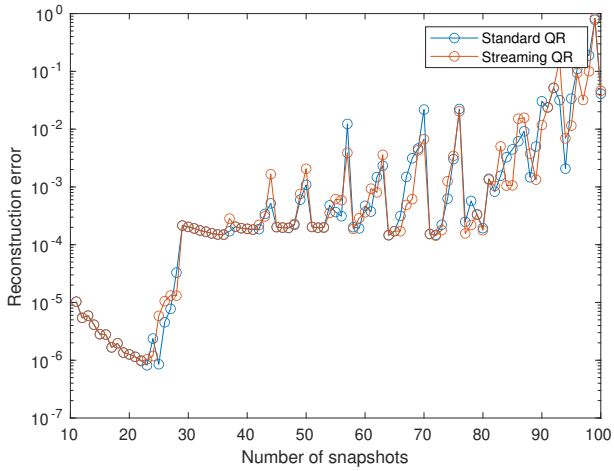

Figure 4.11: Reconstruction error for sequential snapshot additions

5 Conclusion

In this work, we have investigated the application of Dynamic Mode Decomposition methods to wave propagation problems, discretized by high fidelity tensorized finite element methods. Not only have we provided a positive answer, but we have also shown how such methods could be combined with Khatri-Rao structured snapshot matrices produced by (dynamical) low-rank solvers to mitigate the "curse of dimensionality". In this matrix-free setting, the snapshot data is only available through matrix-vector or matrix-matrix multiplications and we have shown how to perform those operations implicitly for arbitrary dimensions. Moreover, we have suggested alternatives to the truncated SVD and initiated the possibility of down-sampling the snapshot matrix prior to any computations. However, for tackling relevant problems, the methods also require data sparse formats for multi-patch geometries,

which are sadly too limited. Moreover, it is worthwhile investigating whether other data sparse formats (e.g. Tucker formats as in [16, 17]) yield similar savings.

A Nearest Khatri-Rao product approximation

This appendix describes how to best approximate a matrix by a sum of Khatri-Rao products. Structured matrix approximations have already been considered for a variety of matrix products, including Kronecker products [30, 20] and of course the standard matrix product [31]. Actually, some of these problems are closely related and can be reformulated into one another. To our knowledge, the nearest Khatri-Rao product approximation has not yet surfaced in the literature and is the object of this appendix. For simplicity, we again consider the case $d = 3$ to avoid multiple cumbersome subscripts. Given $M \in \mathbb{R}^{n_1 n_2 n_3 \times m}$, we look for factor matrices $A^{(k)}$, $B^{(k)}$ and $C^{(k)}$ that minimize

$$\left\| M - \sum_{k=1}^{q} A^{(k)} \odot B^{(k)} \odot C^{(k)} \right\|_F^2 .$$

The minimization in the Frobenius norm allows to solve separate minimization problems for each column of M. Indeed,

$$\left\| M - \sum_{k=1}^{q} A^{(k)} \odot B^{(k)} \odot C^{(k)} \right\|_F^2 = \sum_{j=1}^{m} \left\| \mathbf{m}_j - \sum_{k=1}^{q} \mathbf{a}_j^{(k)} \otimes \mathbf{b}_j^{(k)} \otimes \mathbf{c}_j^{(k)} \right\|_2^2$$

where $\mathbf{m}_j$ denotes the jth column of M and similarly for the factor matrices $A^{(k)}$, $B^{(k)}$ and $C^{(k)}$. Thus, their columns are computed by solving m independent minimization problems for each column of M. Moreover,

$$\left\| \mathbf{m}_j - \sum_{k=1}^{q} \mathbf{a}_j^{(k)} \otimes \mathbf{b}_j^{(k)} \otimes \mathbf{c}_j^{(k)} \right\|_2^2 = \left\| \mathcal{M}_j - \sum_{k=1}^{q} \mathbf{a}_j^{(k)} \circ \mathbf{b}_j^{(k)} \circ \mathbf{c}_j^{(k)} \right\|_F^2$$

where $\circ$ denotes the outer product and $\mathcal{M}_j \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is obtained by reshaping (and permuting) $\mathbf{m}_j$ into a 3rd (and generally dth) order tensor (see algorithm A.1). Equivalently, $\mathbf{m}_j$ is obtained through the vectorization of $\mathcal{M}_j$ in lexicographical ordering. Finally, the vectors $\mathbf{a}_j^{(k)}$, $\mathbf{b}_j^{(k)}$ and $\mathbf{c}_j^{(k)}$ are obtained by computing a CP rank q approximation of $\mathcal{M}_j$. This last problem is rather standard in tensor calculus and several software packages are available for this purpose; e.g. the Tensor Toolbox [23, 22]. Moreover, the method offers great potential for parallelism given that the m minimization problems are independent and do not share any data: the columns of M are loaded into separate processors. The size of the tensors obtained by reshaping those columns is determined by the integers n_i, which are generally always clear from the context of the application. For instance, for (single-patch) tensorized finite element discretizations, they are the number of degrees of freedom in each parametric direction. In practice, the factor matrices $A^{(k)}$, $B^{(k)}$ and $C^{(k)}$ are conveniently stored along the pages of third order tensors $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$. Algorithm A.1 provides a MATLAB style pseudo-code for arbitrary dimension d.

Algorithm A.1 Khatri-Rao rank q approximation

Input: Matrix $M \in \mathbb{R}^{n_1 n_2 \ldots n_d \times m}$, partitioned as $M = [\mathbf{m}_1, \ldots, \mathbf{m}_m]$, rank q.
Output: Tensors $\mathcal{A}_i \in \mathbb{R}^{n_i \times m \times q}$ for $i = 1, \ldots, d$.
1: Initialize $\mathcal{A}_i$ for $i = 1, \ldots, d$.
2: **for** $j = 1, \ldots, m$ **do**
3: $\quad \mathcal{M}_j \leftarrow \texttt{reshape}(\mathbf{m}_j, (n_d, \ldots, n_1))$
4: $\quad \mathcal{M}_j \leftarrow \texttt{permute}(\mathcal{M}_j, (d, \ldots, 1))$
5: $\quad [\mathcal{A}_1(:, j, :), \ldots, \mathcal{A}_d(:, j, :)] = \texttt{cp}(\mathcal{M}_j, rank = q)$
6: **end for**

Effectively, algorithm A.1 computes each slice of $\mathcal{A}_i$ individually. In line 5, essentially any off-the-shelf algorithm for computing a CP approximation may be used. In our implementation, we have used the `cp_als` function from the Tensor Toolbox, which employs an alternating least squares solver.

B Exact solution of the semi-discrete problem

In this appendix, we derive the exact solution of the semi-discrete problem recalled below:

$$\begin{cases} M\ddot{\mathbf{u}}(t) + K\mathbf{u}(t) = \mathbf{f}(t) \qquad \text{for } t \in [0,T], \\ \mathbf{u}(t=0) = \mathbf{u}_0 \\ \dot{\mathbf{u}}(t=0) = \mathbf{v}_0 \end{cases} \tag{B.1}$$

where $K, M \in \mathbb{R}^{n\times n}$ are symmetric and M is positive definite with initial conditions $\mathbf{u}_0, \mathbf{v}_0 \in \mathbb{R}^n$. The exact solution may be expressed in terms of matrix functions. This can be easily seen by considering an analogous scalar problem.

Lemma B.1. The solution of the ODE

$$\begin{cases} \ddot{u}(t) + \lambda u(t) = f(t) \\ u(t=0) = u_0 \\ \dot{u}(t=0) = v_0 \end{cases} \tag{B.2}$$

with $\lambda \in \mathbb{R}_+^*$ and $u_0, v_0 \in \mathbb{R}$ is given by

$$u(t) = u_0 \cos(\sqrt{\lambda}t) + t v_0 \operatorname{sinc}(\sqrt{\lambda}t) + \int_0^t (t-\tau)\operatorname{sinc}(\sqrt{\lambda}(t-\tau)) f(\tau)\, d\tau.$$

Proof. The general solution of (B.2) is given by $u(t) = u_h(t) + u_p(t)$, where $u_h(t)$ is the solution to the corresponding homogeneous problem and $u_p(t)$ is a particular solution. For the homogeneous solution, we immediately obtain

$$u_h(t) = \alpha\cos(\omega t) + \beta \sin(\omega t).$$

where we denoted $\omega = \sqrt{\lambda}$. Moreover, one can easily verify that

$$u_p(t) = \frac{1}{\omega}\int_0^t \sin(\omega(t-\tau)) f(\tau)\, d\tau = \int_0^t (t-\tau)\operatorname{sinc}(\omega(t-\tau)) f(\tau)\, d\tau$$

is a particular solution, where $\operatorname{sinc}(x) = \sin(x)/x$. Indeed,

$$\begin{aligned} \dot{u}_p(t) &= \int_0^t \cos(\omega(t-\tau)) f(\tau)\, d\tau \\ \ddot{u}_p(t) &= f(t) - \omega \int_0^t \sin(\omega(t-\tau)) f(\tau)\, d\tau = f(t) - \omega^2 u_p(t). \end{aligned}$$

Consequently, $\ddot{u}_p(t) + \omega^2 u_p(t) = f(t)$ as wanted. Finally, the constants α and β are determined by imposing the initial conditions:

$$\begin{aligned} u(t=0) &= \alpha = u_0, \\ \dot{u}(t=0) &= \beta\omega = v_0, \end{aligned}$$

from which we deduce $\alpha = u_0$ and $\beta = v_0/\omega$. Thus, the general solution is

$$u(t) = u_0\cos(\omega t) + \frac{v_0}{\omega}\sin(\omega t) + \frac{1}{\omega}\int_0^t \sin(\omega(t-\tau)) f(\tau)\, d\tau$$

and may equivalently be expressed in terms of sinc functions. □

The solution of (B.1) is now readily expressed in terms of matrix functions.

Corollary B.2. The exact solution of (B.1) is given by

$$\mathbf{u}(t) = \cos(\sqrt{A}t)\mathbf{u}_0 + t\operatorname{sinc}(\sqrt{A}t)\mathbf{v}_0 + \int_0^t (t-\tau)\operatorname{sinc}(\sqrt{A}(t-\tau)) M^{-1}\mathbf{f}(\tau)\, d\tau \tag{B.3}$$

where $A = M^{-1}K$.

Proof. There exist multiple equivalent ways of deducing the result, e.g. via the spectral decomposition of $M^{-1}K$, or in a more symmetric way using the eigenbasis of (K, M) or the Cholesky decomposition of M. We will employ the eigenbasis approach. It is well-known that if (K, M) is a symmetric matrix pair and K and M are positive definite, there exists an invertible matrix U of eigenvectors such that

$$U^T K U = D, \qquad U^T M U = I, \tag{B.4}$$

where $D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ is the diagonal matrix of positive eigenvalues (see e.g. [32, Theorem VI.1.15]) and $KU = MUD$ is the associated eigendecomposition. In particular, the matrix U forms an M-orthonormal basis of $\mathbb{R}^n$. We change basis and set $\mathbf{u}(t) = U\mathbf{x}(t)$ (i.e. $\mathbf{x}(t) = U^{-1}\mathbf{u}(t)$. Substituting this expression in (B.1), pre-multiplying by U^T and using (B.4), we obtain the set of uncoupled equations

$$\begin{cases} \ddot{\mathbf{x}}(t) + D\mathbf{x}(t) = U^T\mathbf{f}(t) \\ \mathbf{x}(t=0) = U^{-1}\mathbf{u}_0 \\ \dot{\mathbf{x}}(t=0) = U^{-1}\mathbf{v}_0 \end{cases}$$

whose exact solution, thanks to lemma B.1, is given by

$$\mathbf{x}(t) = \cos(\sqrt{D}t)U^{-1}\mathbf{u}_0 + t\,\operatorname{sinc}(\sqrt{D}t)U^{-1}\mathbf{v}_0 + \int_0^t (t-\tau)\operatorname{sinc}(\sqrt{D}(t-\tau))U^T\mathbf{f}(\tau)\,d\tau.$$

The result follows after back-transforming by pre-multiplying by U

$$\mathbf{u}(t) = U\mathbf{x}(t) = U\cos(\sqrt{D}t)U^{-1}\mathbf{u}_0 + tU\operatorname{sinc}(\sqrt{D}t)U^{-1}\mathbf{v}_0 + \int_0^t (t-\tau)U\operatorname{sinc}(\sqrt{D}(t-\tau))U^{-1}UU^T\mathbf{f}(\tau)\,d\tau$$

and noting that $M^{-1}K = UDU^{-1}$ is the spectral decomposition of $M^{-1}K$ and $M^{-1} = UU^T$. □

Remark B.3. It is sometimes possible to give more explicit expressions for specific choices of right-hand sides (e.g. $\mathbf{f}(t) = \mathbf{b}$ or $\mathbf{f}(t) = \sin(\tilde{\omega}t)\mathbf{b}$, where $\mathbf{b}$ is a constant vector). See [33] for a derivation.

References

[1] P. J. Schmid, Dynamic mode decomposition of numerical and experimental data, Journal of fluid mechanics 656 (2010) 5–28.

[2] C. W. Rowley, I. Mezić, S. Bagheri, P. Schlatter, D. S. Henningson, Spectral analysis of nonlinear flows, Journal of fluid mechanics 641 (2009) 115–127.

[3] Z. Drmac, I. Mezic, R. Mohr, Data driven modal decompositions: analysis and enhancements, SIAM Journal on Scientific Computing 40 (4) (2018) A2253–A2285.

[4] J. N. Kutz, S. L. Brunton, B. W. Brunton, J. L. Proctor, Dynamic mode decomposition: data-driven modeling of complex systems, SIAM, 2016.

[5] O. Koch, C. Lubich, Dynamical low-rank approximation, SIAM Journal on Matrix Analysis and Applications 29 (2) (2007) 434–454.

[6] M. Hochbruck, M. Neher, S. Schrammer, Rank-adaptive dynamical low-rank integrators for first-order and second-order matrix differential equations, BIT Numerical Mathematics 63 (1) (2023) 9.

[7] K.-J. Bathe, Finite element procedures, Klaus-Jurgen Bathe, 2006.

[8] T. J. Hughes, J. A. Cottrell, Y. Bazilevs, Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement, Computer methods in applied mechanics and engineering 194 (39-41) (2005) 4135–4195.

[9] J. A. Cottrell, T. J. Hughes, Y. Bazilevs, Isogeometric analysis: toward integration of CAD and FEA, John Wiley & Sons, 2009.

[10] C. De Boor, A practical guide to splines, Vol. 27, springer-verlag New York, 1978.

[11] T. J. Hughes, The finite element method: linear static and dynamic finite element analysis, Courier Corporation, 2012.

[12] A. Quarteroni, Numerical models for differential problems, Vol. 2, Springer, 2009.

[13] Y. Bazilevs, L. Beirao da Veiga, J. A. Cottrell, T. J. Hughes, G. Sangalli, Isogeometric analysis: approximation, stability and error estimates for h-refined meshes, Mathematical Models and Methods in Applied Sciences 16 (07) (2006) 1031–1090.

[14] A. Bressan, E. Sande, Approximation in fem, dg and iga: a theoretical comparison, Numerische Mathematik 143 (2019) 923–942.

[15] E. Sande, C. Manni, H. Speleers, Explicit error estimates for spline approximation of arbitrary smoothness in isogeometric analysis, Numerische Mathematik 144 (4) (2020) 889–929.

[16] M. Montardini, G. Sangalli, M. Tani, A low-rank isogeometric solver based on tucker tensors, Computer Methods in Applied Mechanics and Engineering 417 (2023) 116472.

[17] M. Montardini, G. Sangalli, M. Tani, A low-rank multipatch isogeometric method based on tucker tensors, arXiv preprint arXiv:2312.08736 (2023).

[18] Z. Drmac, I. Mezic, R. Mohr, On least squares problems with certain vandermonde–khatri–rao structure with applications to dmd, SIAM journal on scientific computing 42 (5) (2020) A3250–A3284.

[19] G. Golub, W. Kahan, Calculating the singular values and pseudo-inverse of a matrix, Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis 2 (2) (1965) 205–224.

[20] G. H. Golub, C. F. Van Loan, Matrix computations, JHU press, 2013.

[21] J. Baglama, L. Reichel, Augmented implicitly restarted lanczos bidiagonalization methods, SIAM Journal on Scientific Computing 27 (1) (2005) 19–42.

[22] T. G. Kolda, B. W. Bader, Tensor decompositions and applications, SIAM review 51 (3) (2009) 455–500.

[23] B. W. Bader, T. G. Kolda, Tensor toolbox for matlab, version 3.6 (2023).
URL `www.tensortoolbox.org`

[24] N. Halko, P.-G. Martinsson, J. A. Tropp, Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, SIAM review 53 (2) (2011) 217–288.

[25] L. N. Trefethen, Householder triangularization of a quasimatrix, IMA journal of numerical analysis 30 (4) (2010) 887–897.

[26] D. Kressner, L. Perisa, Recompression of hadamard products of tensors in tucker format, SIAM Journal on Scientific Computing 39 (5) (2017) A1879–A1902.

[27] R. A. Horn, C. R. Johnson, Topics in Matrix Analysis, Cambridge University Press, 1991.

[28] R. Vázquez, A new design for the implementation of isogeometric analysis in Octave and Matlab: GeoPDEs 3.0, Computers & Mathematics with Applications 72 (3) (2016) 523–554.

[29] S. Fraschini, G. Loli, A. Moiola, G. Sangalli, An unconditionally stable space-time isogeometric method for the acoustic wave equation, arXiv preprint arXiv:2303.07268 (2023).

[30] C. F. Van Loan, N. Pitsianis, Approximation with Kronecker products, in: Linear algebra for large scale and real-time applications, Springer, 1993, pp. 293–314.

[31] R. A. Horn, C. R. Johnson, Matrix analysis, Cambridge university press, 2012.

[32] G. Stewart, J. Sun, Matrix Perturbation Theory, Computer Science and Scientific Computing, ACADEMIC Press, INC, 1990.

[33] Y. Voet, On the computation of matrix functions for the finite element solution of linear elasticity problems in dynamics, Tech. rep., École polytechnique fédérale de Lausanne (2020).