SIAM J. Sci. Comput. © 2020 Society for Industrial and Applied Mathematics
Vol. 42, No. 5, pp. A3250–A3284

ON LEAST SQUARES PROBLEMS WITH CERTAIN VANDERMONDE–KHATRI–RAO STRUCTURE WITH APPLICATIONS TO DMD[*]

ZLATKO DRMAČ[†], IGOR MEZIĆ[‡], AND RYAN MOHR[§]

Abstract. This paper proposes a new computational method for solving the structured least squares problem that arises in the process of identification of coherent structures in dynamic processes, such as, e.g., fluid flows. It is deployed in combination with dynamic mode decomposition (DMD), which provides a nonorthogonal set of modes corresponding to particular temporal frequencies. A selection of these is used to represent time snapshots of the underlying dynamics. The coefficients of the representation are determined from a solution of a structured linear least squares problems with the matrix that involves the Khatri–Rao product of a triangular and a Vandermonde matrix. Such a structure allows for a very efficient normal equation based least squares solution, which is used in state-of-the-art computational fluid dynamics (CFD) tools, such as the sparsity promoting DMD (DMDSP). A new numerical analysis of the normal equations approach provides insights about its applicability and its limitations. Relevant condition numbers that determine numerical robustness are identified and discussed. Further, the paper offers a corrected seminormal solution and the QR factorization based algorithms. It shows how to use the Vandermonde–Khatri–Rao structure to efficiently compute the QR factorization of the least squares coefficient matrix, thus providing a new computational tool for the ill-conditioned cases where the normal equations may fail to compute a sufficiently accurate solution. Altogether, the presented material provides a firm numerical linear algebra framework for a class of structured least squares problems arising in a variety of applications besides the DMD, such as, e.g., multistatic antenna array processing.

Key words. antenna array processing, coherent structure, dynamic mode decomposition, Khatri–Rao product, Koopman operator, Krylov subspaces, proper orthogonal decomposition, Rayleigh–Ritz approximation, scattering coefficients, structured least squares, Vandermonde matrix

AMS subject classifications. 15A12, 15A23, 65F35, 65L05, 65M20, 65M22, 93A15, 93A30, 93B18, 93B40, 93B60, 93C05, 93C10, 93C15, 93C20, 93C57

DOI. 10.1137/19M1288474

1. Introduction.

Dynamic mode decomposition (DMD) [38] is a tool of the trade in computational fluid dynamics (CFD), both in high fidelity numerical simulations and in pure data driven scenarios; for a review see [42] and references therein. Its data driven framework and deep theoretical roots in Koopman operator theory [36, 51] make DMD a versatile tool in a plethora of applications beyond CFD, e.g., studying dynamics of infectious diseases [34] or revealing spontaneous subtle emotions on human faces [10] in the field of affective computing. For detailed study of DMD and its applications we refer the reader to [31].

A distinctive feature of DMD is that it is purely data driven. It does not assume knowledge of the solution of the governing, generally nonlinear, equations obeyed by

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section September 20, 2019; accepted for publication (in revised form) June 16, 2020; published electronically October 20, 2020.
https://doi.org/10.1137/19M1288474
Funding: This research is supported by the DARPA contract HR0011-16-C-0116 *On a Data-Driven, Operator-Theoretic Framework for Space-Time Analysis of Process Dynamics* and the DARPA contract HR0011-18-9-0033 *The Physics of Artificial Intelligence.*
[†]Faculty of Science, Department of Mathematics, University of Zagreb, 10000 Zagreb, Croatia (drmac@math.hr).
[‡]Department of Mechanical Engineering and Mathematics, University of California, Santa Barbara, Santa Barbara, CA 93106 USA (mezic@ucsb.edu) and AIMdyn, Inc., Santa Barbara, CA 93101 USA.
[§]AIMdyn, Inc., Santa Barbara, CA 93101 USA (mohrr@aimdyn.com).

the dynamics under study; it does assume that the supplied data snapshots (vectors of observables) $\mathbf{x}_i \in \mathbb{C}^n$ are generated by a linear operator $\mathbb{A}$ so that $\mathbf{x}_{i+1} = \mathbb{A}\mathbf{x}_i$, $i = 1, \ldots, m$, with some initial $\mathbf{x}_1$, and a constant time lag δt. One can think of $\mathbb{A}$ as, e.g., a black-box numerical simulation software, or as, e.g., a discretized physics taken by a high-speed camera such as in studying flame dynamics and combustion instabilities [18]. However, $\mathbb{A}$ is not accessible; only the snapshots $\mathbf{x}_1, \ldots, \mathbf{x}_{m+1}$ are available. The DMD computes approximate eigenpairs of $\mathbb{A}$ by a combination of the proper orthogonal decomposition (POD) and the Rayleigh–Ritz projection. Under certain conditions, DMD can serve as an approximation to the Koopman operator underlying the process evolution [36, 2, 29]. For a good approximation in the sense of the Koopman operator, the key is that the set of observables is well selected and rich enough—this is a separate issue not considered in this paper.

The Ritz pairs (λ_j, z_j), computed by the DMD as the approximate eigenpairs of $\mathbb{A}$, are used for spatial-temporal decomposition of the snapshots. More precisely, it can be shown that, generically, the snapshots can be decomposed using the Ritz vectors z_j (normalized in the ℓ_2 norm to $\|z_j\|_2 = 1$) as the spatial spanning set, and with the temporal coefficients provided by the Ritz values $\lambda_j = |\lambda_j| e^{\mathrm{i}\omega_j \delta t}$ (here assumed simple)

$$\mathbf{x}_i = \sum_{j=1}^{m} z_j \mathfrak{a}_j \lambda_j^{i-1} \equiv \sum_{j=1}^{m} z_j \mathfrak{a}_j |\lambda_j|^{i-1} e^{\mathrm{i}\omega_j (i-1)\delta t}, \quad i = 1, \ldots, m. \tag{1.1}$$

In order to identify the most important coherent structures in the dynamic process, the representations (1.1) are attempted with a smaller number of modes z_j; the task is to determine $\ell < m$, the indices $\varsigma_1 < \cdots < \varsigma_\ell$, and the coefficients α_j to achieve high fidelity of the snapshot representations

$$\mathbf{x}_i \approx \widehat{\mathbf{x}}_i \equiv \sum_{j=1}^{\ell} z_{\varsigma_j} \alpha_j \lambda_{\varsigma_j}^{i-1}, \quad i = 1, \ldots, m. \tag{1.2}$$

After selecting an appropriate subset $\{z_{\varsigma_j}\}$ of the modes, the key numerical step, and the central theme of this paper, is solution of a structured linear least squares (LS) problem $\sum_{i=1}^{m} \|\mathbf{x}_i - \widehat{\mathbf{x}}_i\|_2^2 \longrightarrow \min$ for the coefficients α_j in (1.2). We are interested in both algebraic and numerical aspects of this problem.

1.1. Contributions and overview.

To set the stage, in section 2 we briefly review the DMD, and in section 2.2 we state in detail the problem of snapshot reconstruction (1.1) and (1.2) using the selected modes; the sparsity promoting DMD (DMDSP) [26] solution of the snapshot reconstruction problem is reviewed in section 2.2.1.

Our contributions can be summarized as threefold and are presented in the rest of the paper as follows.

In section 3, we revisit the normal equations based solution, which is used in the DMDSP. In section 3.1, we formulate the reconstruction problem in more generality by allowing weighted reconstruction error—each snapshot carries a weight that determines its importance in the overall error. Also, the norm in the space of the observables can be an energy norm given by a user supplied positive definite matrix (e.g., inverse noise covariance, or a diagonal matrix that emphasizes snapshots of particular interest). Then, in section 3.2, we give explicit formulas for the normal equations solutions for this weighted LS problem, and we discuss the underlying structure, which is an interplay of the Kronecker, Hadamard, and Khatri–Rao matrix products. This provides an algebraic framework and a solution formula for the cases where noise, uncertainty, and different scales/physical nature of the observables must be taken into account. In section 3.3 we propose a new simple yet powerful modification of the implementation of the DMDSP; its polishing phase is considerably improved by replacing the variation of the Lagrangian technique with deployment of certain canonical projections. Using a CFD example, in section 3.4 we illustrate the efficacy of the presented techniques.

Next, in section 4 we examine the numerical aspects of the solution via the normal equations. From the numerical point of view, using normal equations for solving the LS problems is an ill-advised approach, because the condition number of the problem is squared; see, e.g., [7, section 2.1.4]. Moreover, the matrix of the LS coefficients depends on the notoriously ill-conditioned Vandermonde matrix defined by the Ritz values $\{\lambda_{\varsigma_j}\}$, and on the nonorthonormal set of the corresponding Ritz vectors. Nonetheless, the DMDSP is an important and widely used tool in the DMD computational framework. Furthermore, a similarly structured LS problem arises in multistatic antenna array processing, and the solution is again based on the normal equations [32]. To the best of our knowledge, the potential of numerical failure of this normal equations approach due to ill-conditioning has not been addressed in the literature. The key questions are when and why are normal equations safe to use? In section 4.2, we identify the relevant condition number and argue that the underlying Khatri–Rao and Hadamard products' structure of the normal equations usually ensures sufficient numerical accuracy despite potentially high condition numbers of both the Vandermonde matrix of the Ritz values and the matrix of the Ritz vectors. To the best of our knowledge, this aspect of the DMDSP has not been considered before, and our results provide a precise explanation of the condition under which the normal equation based method is safe to use. We use numerical examples (contrived small dimensional ones, Q2D Kolmogorov flow, the Chua's circuit, and the Ikeda map) to illustrate the applicability of our theory and to motivate and justify the development of new algorithms that avoid squaring the condition number of the LS matrix.

The third part of our contributions, in section 5, is new LS solution methods via the QR factorization—the corrected seminormal solution in section 5.1 and the pure QR factorization based procedure. The key technical difficulty is efficient computation of the QR factorization of a large and structured LS matrix; this is resolved in section 5.2 where we propose an efficient recursive scheme that exploits the Khatri–Rao product structure involving a Vandermonde matrix. In section 5.3 we provide an error and perturbation analysis of the QR factorization based solution and discuss the importance of pivoting. Fine details such as using only real arithmetic in the case of real snapshots are discussed in section 5.4. Together with detailed descriptions and numerical analysis of the new proposed algorithms, we provide detailed blueprints for numerical software development in the framework of the LAPACK library [1].

In the concluding remarks in section 6, we stress the importance of using the advanced numerical linear algebra and announce our work to develop high performance software implementations of the introduced algorithms.

2. Preliminaries.

For the reader's convenience, and to introduce the notation, we briefly review the DMD, the snapshot reconstruction task (1.1)–(1.2), and the DMDSP, which is a widely used snapshot reconstruction method in the DMD framework.

2.1. DMD and its variations. The DMD, introduced in [38], is outlined in Algorithm 1.

Algorithm 1. $[Z_k, \Lambda_k] = \mathrm{DMD}(\mathbf{X}_m, \mathbf{Y}_m)$.

```
Input:
    • $\mathbf{X}_m = (\mathbf{x}_1, \ldots, \mathbf{x}_m), \mathbf{Y}_m = (\mathbf{y}_1, \ldots, \mathbf{y}_m) = (\mathbf{x}_2, \ldots, \mathbf{x}_{m+1}) \in \mathbb{C}^{n\times m}$ that
      define a sequence of snapshots pairs $(\mathbf{x}_i, \mathbf{y}_i \equiv \mathbb{A}\mathbf{x}_i)$. (Tacit assumption is
      that $n$ is large and that $m \ll n$.)
1: $[U, \Sigma, \Phi] = svd(\mathbf{X}_m)$ ; {The thin SVD: $\mathbf{X}_m = U\Sigma\Phi^*$, $U \in \mathbb{C}^{n\times m}$, $\Sigma = \mathrm{diag}(\sigma_i)_{i=1}^m$}
2: Determine numerical rank $k$.
3: Set $U_k = U(:, 1:k)$, $\Phi_k = \Phi(:, 1:k)$, $\Sigma_k = \Sigma(1:k, 1:k)$
4: $S_k = ((U_k^*\mathbf{Y}_m)\Phi_k)\Sigma_k^{-1}$; {Schmid's formula for the Rayleigh quotient $U_k^*\mathbb{A}U_k$}
5: $[B_k, \Lambda_k] = \mathrm{eig}(S_k)$ {$\Lambda_k = \mathrm{diag}(\lambda_j)_{j=1}^k$; $S_kB_k(:,j) = \lambda_jB_k(:,j)$; $\|B_k(:,j)\|_2 = 1$}
   {We always assume the generic case that $S_k$ is diagonalizable, i.e., that in Line 5.
   the function eig() computes the full column rank matrix $B_k$ of the eigenvectors
   of $S_k$. In general, $B_k$ may be ill-conditioned.}
6: $Z_k = U_kB_k$ {Ritz vectors}
Output: $Z_k$, $\Lambda_k$
```

The more intuitive companion matrix based approach [36], which is considered numerically infeasible due to the computation with the notoriously ill-conditioned Vandermonde matrices, can be made competitive with Algorithm 1, as shown in [14]. Another approach, designated as Exact DMD [49], allows nonsequential data $\mathbf{Y}_m = \mathbb{A}\mathbf{X}_m$, and it implicitly uses the matrix $\mathbf{Y}_m\mathbf{X}_m^\dagger$, where $\mathbf{X}_m^\dagger$ denotes the Moore–Penrose generalized inverse. For a more extensive study of DMD and its variations and applications, see, e.g., [39, 11, 20, 12, 21, 46, 43, 45, 44, 33, 3]. Recently, in [15], we proposed an enhancement of Algorithm 1—Refined Rayleigh Ritz Data Driven Modal Decomposition. It follows the DMD scheme, but it further allows data driven refinement of the Ritz vectors, and it provides data driven computable residuals.

2.2. Sparse reconstruction using Ritz pairs. In matrix form, (1.1) is compactly written as

$$\mathbf{X}_m = \begin{pmatrix} z_1 & z_2 & \ldots & z_m \end{pmatrix} \begin{pmatrix} \mathfrak{a}_1 & & & \\ & \mathfrak{a}_2 & & \\ & & \ddots & \\ & & & \mathfrak{a}_m \end{pmatrix} \begin{pmatrix} 1 & \lambda_1 & \ldots & \lambda_1^{m-1} \\ 1 & \lambda_2 & \ldots & \lambda_2^{m-1} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & \lambda_m & \ldots & \lambda_m^{m-1} \end{pmatrix}, \tag{2.1}$$

where the coefficients $\mathfrak{a}_j$ are computed from the first column of $\mathbf{X}_m$, $\mathbf{X}_m(:,1)$ as

$$(\mathfrak{a}_j)_{j=1}^m = Z_m^\dagger \mathbf{X}_m(:,1). \tag{2.2}$$

In the framework of the Schmid's DMD, the amplitudes are usually determined by the formula (2.2). Since $Z_m = U_mB_m$ (see Line 6 in Algorithm 1) and $U_m^*U_m = \mathbb{I}_m$, instead of applying the pseudoinverse of the explicitly computed Z_m, one would use the more efficient formula

$$Z_m^\dagger \mathbf{X}_m(:,1) = B_m^{-1}(U_m^*\mathbf{X}_m(:,1)). \tag{2.3}$$

For a formal proof of the existence of the representation (2.1) and relation to the formulation via the Krylov decomposition and the Frobenius companion matrix, we refer the reader to [14].

The main goal of the representations (1.1), (2.1) is to reveal the underlying structure in the data. It is best achieved with an approximate representation using as few as possible Ritz pairs (λ_j, z_j). A DMD algorithm may return $k < m$ Ritz pairs, and the representation cannot be exact. Ideally, the reconstruction is successful with a small number of Ritz vectors that have small residuals (i.e., corresponding to some eigenvectors of the underlying operator accessible only through the sequence of snapshots $\mathbf{x}_i$).

If we desire to take only the $\ell < m$ most relevant Ritz pairs, then the question is how to determine ℓ, and what indices $\varsigma_1, \ldots, \varsigma_\ell$ should be selected to achieve good approximation:

$$\mathbf{X}_m \approx \begin{pmatrix} z_{\varsigma_1} & z_{\varsigma_2} & \ldots & z_{\varsigma_\ell} \end{pmatrix} \begin{pmatrix} \alpha_{\varsigma_1} & & & \\ & \alpha_{\varsigma_2} & & \\ & & \ddots & \\ & & & \alpha_{\varsigma_\ell} \end{pmatrix} \begin{pmatrix} 1 & \lambda_{\varsigma_1} & \ldots & \lambda_{\varsigma_1}^{m-1} \\ 1 & \lambda_{\varsigma_2} & \ldots & \lambda_{\varsigma_2}^{m-1} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & \lambda_{\varsigma_\ell} & \ldots & \lambda_{\varsigma_\ell}^{m-1} \end{pmatrix} \equiv Z_\varsigma D_{\boldsymbol{\alpha}} \mathbb{V}_\varsigma. \tag{2.4}$$

This seems a difficult task for practical computation. In general, here we assume the availability of a *reconstruction wizard* that selects $z_{\varsigma_1}, \ldots, z_{\varsigma_\ell}$ so that in (2.4) the reconstruction error $\|\mathbf{X}_m - Z_\varsigma D_{\boldsymbol{\alpha}} \mathbb{V}_\varsigma\|_F^2$ is as small as possible. An example of such a wizard is an optimizer with (relaxed) sparsity constraints, e.g., the ADMM (Alternating Directions Method of Multipliers) which is used in the DMDSP [26], which we briefly review in section 2.2.1. Another strategy, proposed in [30, section 3], is to choose modes that are dominant with respect to their influence over all snapshots. A similar approach is used in [5].

For any strategy, once the modes are selected, instead of using the coefficients $\mathfrak{a}_{\varsigma_i}$ from the full reconstruction, one can achieve higher fidelity of (2.4) by recomputing them by solving an LS problem for the new coefficients α_{ς_i}, with fixed $z_{\varsigma_1}, \ldots, z_{\varsigma_\ell}$. Here, optionally, we can use data driven refinements of the selected Ritz pairs $(\lambda_{\varsigma_j}, z_{\varsigma_j})$ and reduce the residuals[1]; see [15].

2.2.1. DMDSP: Sparsity promoting DMD. One way to set up a computational procedure is to formulate it as an LS approximation with sparsity constraints

$$\|\mathbf{X}_m - Z_k D_{\boldsymbol{\alpha}} \mathbb{V}_{k,m}\|_F^2 + \gamma \|\boldsymbol{\alpha}\|_0 \longrightarrow \min_{\boldsymbol{\alpha}}, \tag{2.5}$$

where $\|\boldsymbol{\alpha}\|_0$ denotes the number of nonzero entries in the vector $\boldsymbol{\alpha} = (\alpha_i)_{i=1}^k$ of the new coefficients; the parameter $\gamma \geq 0$ penalizes nonsparsity of the solution $\boldsymbol{\alpha}$. The measure of sparsity $\|\boldsymbol{\alpha}\|_0$ is in practical computations relaxed by using the ℓ_1 norm, $\|\boldsymbol{\alpha}\|_1$, which turns (2.5) into a convex optimization problem:

$$\|\mathbf{X}_m - Z_k D_{\boldsymbol{\alpha}} \mathbb{V}_{k,m}\|_F^2 + \gamma \|\boldsymbol{\alpha}\|_1 \longrightarrow \min_{\boldsymbol{\alpha}}. \tag{2.6}$$

In [26], for a given value of γ, the problem is solved in two steps:

1. Problem (2.6) is solved using the ADMM, and the optimal $\boldsymbol{\alpha}$ is sparsified by setting to zero its entries that are in modulus below a given threshold. Let $j_1, \ldots, j_s$ be the indices of the thus annihilated entries—they define the sparsity pattern. Define E to be the matrix whose columns are the columns of the identity with the indices $j_1, \ldots, j_s$. Then the obtained sparsity pattern of $\boldsymbol{\alpha}$ can be characterized by $E^T\boldsymbol{\alpha} = \mathbf{0}$.

[1]Smaller residuals improve the performance of (1.2) when used for prediction.

2. Once (2.6) has identified the sparsity structure, the coefficients α_i are computed by solving the LS problem with sparsity constraint:

$$\|\mathbf{X}_m - Z_k D_{\boldsymbol{\alpha}} \mathbb{V}_{k,m}\|_F^2 \longrightarrow \min_{\boldsymbol{\alpha}}, \text{ with the constraint } E^T\boldsymbol{\alpha} = \mathbf{0}. \tag{2.7}$$

In DMDSP terminology [26], this is the *polishing* part of the computation. Since an optimal value of γ may be problem dependent, the above two-step procedure is repeated over a grid of (dozens or hundreds of) values of γ.

An advantage of DMDSP is that it is a black-box procedure; the wizard is simply a convex optimization solver. However, it requires suitable range for the parameter γ, which, to the best of our understanding, is determined experimentally for each particular problem.[2] Further, ADMM uses the augmented Lagrangian, which requires an additional penalty parameter ρ, which means the user must input two parameters (see [26, section A]).

The optimization problem (2.7) is solved by variation of the Lagrangian (see [26, Appendix C]). This can be done more efficiently, and we discuss it in section 3.3. Further, in the case of real data, the approximant needs to be real as well; this is naturally achieved if the selected modes appear in complex conjugate pairs. It is not clear whether the optimizer in DMDSP is so tuned to respect this structure. The issue of computations with real data is discussed in detail in section 5.4.

We complete this review with an important observation.

Remark 2.1. The minimizations (2.5), (2.6) look analogous to the compressed sensing problems; in fact [26] motivated the development of DMDSP as a combination of compressed sensing and convex optimization techniques. It should be noted, however, that the snapshot reconstruction problem (2.4) may be heavily overdetermined and generically with unique solution, while in the compressed sensing framework one has an underestimated LS problem and the sparsity constraint is imposed over a manifold of solutions. These are two fundamentally different situations. We further comment on this issue in section 5.3.1.

3. Snapshot reconstruction: Weighted LS formulation and algebraic structure.

In this section, we discuss the algebraic structure of the LS problem that defines the coefficients α_j in (1.2). In section 3.1, we first introduce a more general weighted form of the problem by assigning nonnegative weight $\mathfrak{w}_i$ to each snapshot $\mathbf{x}_i$, thus determining its importance in the reconstruction. We believe that this added functionality will prove useful in applications of the DMD (in particular in non-autonomous setting), as well as in other applications such as, e.g., multistatic antenna array processing, which is briefly discussed in section 3.2.2. An explicit formula for the solution of the corresponding normal equation for this general weighted problem is given in section 3.2, and in section 3.2.1 we show that the Hadamard product structure of the normal equations originates in the Khatri–Rao product structure of the coefficients of the LS problem. Finally, in section 3.3 we revisit the polishing phase of the DMDSP and show that it can be implemented more efficiently.

3.1. Setting the scene: The weighted LS reconstruction problem.

Suppose we have selected ℓ modes, reindexed so that they are the leading ones, $\varsigma_j = j$, $j = 1, \ldots, \ell$, and that we seek an approximate snapshot reconstruction (1.2). The pairs (λ_j, z_j) are approximate eigenpairs of $\mathbb{A}$, e.g., the Ritz pairs. In terms of Algorithm 1, $Z_\ell \equiv \begin{pmatrix} z_1 & \ldots & z_\ell \end{pmatrix} = U_k \widetilde{B}_\ell$ with some $k \times \ell$ matrix $\widetilde{B}_\ell$, and $\ell \leq k \leq \min(m, n)$.

[2]See, e.g., the software [25] for test examples in [26].

The task is to determine, for given (λ_j, z_j)'s and nonnegative weights $\mathfrak{w}_i$, the α_j's to achieve

$$\sum_{i=1}^{m} \mathfrak{w}_i^2 \left\| \mathbf{x}_i - \sum_{j=1}^{\ell} z_j \alpha_j \lambda_j^{i-1} \right\|_2^2 \longrightarrow \min_{\alpha_j}. \tag{3.1}$$

The weights $\mathfrak{w}_i > 0$ are used to emphasize snapshots whose reconstruction is more important; with suitable choices of $\mathbf{W} \equiv \mathrm{diag}(\mathfrak{w}_i)_{i=1}^m$ we can target discrete time subintervals of particular interest or, e.g., introduce forgetting factors that give less importance to older, or more noisy (less reliable), information.

Remark 3.1. Here we use the $\|\cdot\|_2$ norm as the most common choice; if one wants to distinguish the importance of different observables (that might be of a different physical nature, expressed in different units, on different scales, and with different levels of uncertainty), then $\|\cdot\|_2$ can be replaced with an energy norm $\|\sqrt{\mathbf{M}}\cdot\|_2$, where $\mathbf{M}$ is positive definite and $\sqrt{\mathbf{M}}$ stands for the upper triangular Cholesky factor, or the positive definite square root of $\mathbf{M}$. This important modification adds one new level of technical details, and it is omitted in this paper. For a detailed analysis of DMD in the $\mathbf{M}$-induced inner product we refer the reader to [15].

3.1.1. A matrix formulation and dimension reduction. The special structure of (3.1) is best revealed in its matrix formulation. To that end, define $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_j)_{j=1}^{\ell}$,

$$\boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \cdot \\ \alpha_\ell \end{pmatrix}, \quad \Delta_{\boldsymbol{\alpha}} = \begin{pmatrix} \alpha_1 & 0 & \cdot & 0 \\ 0 & \alpha_2 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 0 \\ 0 & \cdot & 0 & \alpha_\ell \end{pmatrix}, \quad \Lambda_i = \mathbb{V}_{\ell,m}(:,i) \equiv \begin{pmatrix} \lambda_1^{i-1} \\ \lambda_2^{i-1} \\ \cdot \\ \lambda_\ell^{i-1} \end{pmatrix},$$

$$\Delta_{\Lambda_i} = \begin{pmatrix} \lambda_1^{i-1} & 0 & \cdot & 0 \\ 0 & \lambda_2^{i-1} & \cdot & \cdot \\ \cdot & \cdot & \cdot & 0 \\ 0 & \cdot & 0 & \lambda_\ell^{i-1} \end{pmatrix} \equiv \boldsymbol{\Lambda}^{i-1},$$

and write the objective (3.1) as the function of $\boldsymbol{\alpha}$,

$$\Omega^2(\boldsymbol{\alpha}) \equiv \| \left[\mathbf{X}_m - Z_\ell \Delta_{\boldsymbol{\alpha}} \begin{pmatrix} \Lambda_1 & \Lambda_2 & \ldots & \Lambda_m \end{pmatrix} \right] \mathbf{W} \|_F^2 \longrightarrow \min, \tag{3.2}$$

where

$$\begin{pmatrix} \Lambda_1 & \Lambda_2 & \ldots & \Lambda_m \end{pmatrix} = \begin{pmatrix} 1 & \lambda_1 & \ldots & \lambda_1^{m-1} \\ 1 & \lambda_2 & \ldots & \lambda_2^{m-1} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & \lambda_\ell & \ldots & \lambda_\ell^{m-1} \end{pmatrix} \equiv \mathbb{V}_{\ell,m} \in \mathbb{C}^{\ell \times m}. \tag{3.3}$$

Schematically, we have, assuming $n > m$,

$$\underbrace{\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}}_{\mathbf{X}_m} \approx \underbrace{\begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \end{pmatrix}}_{Z_\ell} \overbrace{\begin{pmatrix} \star & 0 \\ 0 & \star \end{pmatrix}}^{\Delta_{\boldsymbol{\alpha}}} \underbrace{\begin{pmatrix} + & + & + & + & + \\ + & + & + & + & + \end{pmatrix}}_{(\Lambda_1\ \Lambda_2\ \ldots\ \Lambda_m)}.$$

Since $\mathbf{X}_m = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$, we can rewrite the above expression as follows:

$$\Omega^2(\boldsymbol{\alpha}) = \| \left[\begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_m \end{pmatrix} - Z_\ell \Delta_{\boldsymbol{\alpha}} \begin{pmatrix} \Lambda_1 & \Lambda_2 & \ldots & \Lambda_m \end{pmatrix} \right] \mathbf{W} \|_F^2$$

(3.4)

$$= \|(\mathbf{W} \otimes \mathbb{I}_n) \left[\begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_m \end{pmatrix} - \begin{pmatrix} Z_\ell \Delta_{\boldsymbol{\alpha}} \Lambda_1 \\ \vdots \\ Z_\ell \Delta_{\boldsymbol{\alpha}} \Lambda_m \end{pmatrix} \right] \|_2^2 = \| \begin{pmatrix} \mathfrak{w}_1 \mathbf{x}_1 \\ \vdots \\ \mathfrak{w}_m \mathbf{x}_m \end{pmatrix} - \begin{pmatrix} \mathfrak{w}_1 Z_\ell \Delta_{\Lambda_1} \\ \vdots \\ \mathfrak{w}_m Z_\ell \Delta_{\Lambda_m} \end{pmatrix} \boldsymbol{\alpha} \|_2^2.$$

In principle, (3.4) is a standard linear LS problem that can be solved using an off-the-shelf solver. This is certainly not optimal (here we assume $n \gg m > \ell$) because it ignores the particular structure of the coefficient matrix, which is of potentially large dimensions $mn \times \ell$, not sparse, and the direct solver complexity is $O(mn\ell^2)$ flops. Further, to understand the numerical accuracy of the computed approximations, and to identify relevant condition numbers, we need to take into account the structure that involves the Ritz pars (z_j, λ_j). These issues are important, and in this section we provide the details.

The first step is to remove the ambient space dimension n and to replace it with ℓ. To that end, let $Z_\ell = Q \left(\begin{smallmatrix} R \\ \mathbf{0} \end{smallmatrix} \right)$ be the QR factorization. Note that R is of full rank.

Then, using the unitary invariance of the Euclidean norm, the above can be written as

$$\Omega^2(\boldsymbol{\alpha}) = \left\| \begin{pmatrix} \mathfrak{w}_1 Q^* \mathbf{x}_1 \\ \vdots \\ \mathfrak{w}_m Q^* \mathbf{x}_m \end{pmatrix} - \begin{pmatrix} \mathfrak{w}_1 \left(\begin{smallmatrix} R \\ \mathbf{0} \end{smallmatrix} \right) \Delta_{\Lambda_1} \\ \vdots \\ \mathfrak{w}_m \left(\begin{smallmatrix} R \\ \mathbf{0} \end{smallmatrix} \right) \Delta_{\Lambda_m} \end{pmatrix} \boldsymbol{\alpha} \right\|_2^2 . \tag{3.5}$$

Now partition each $Q^*\mathbf{x}_i$ as $Q^*\mathbf{x}_i = \left(\begin{smallmatrix} \mathbf{g}_i \\ \mathbf{h}_i \end{smallmatrix} \right)$ where $\mathbf{g}_i$ is $\ell \times 1$ and $\mathbf{h}_i$ s $(n - \ell) \times 1$. The objective function becomes

$$\Omega^2(\boldsymbol{\alpha}) = \|(\mathbf{W} \otimes \mathbb{I}_\ell) \left[\begin{pmatrix} \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_m \end{pmatrix} - \begin{pmatrix} R\Delta_{\Lambda_1} \\ \vdots \\ R\Delta_{\Lambda_m} \end{pmatrix} \boldsymbol{\alpha} \right] \|_2^2 + \sum_{i=1}^m \mathfrak{w}_i^2 \|\mathbf{h}_i\|_2^2 \tag{3.6}$$

$$\equiv \|(\mathbf{W} \otimes \mathbb{I}_\ell) \left[\begin{pmatrix} \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_m \end{pmatrix} - (\mathbb{I}_m \otimes R) \begin{pmatrix} \Delta_{\Lambda_1} \\ \vdots \\ \Delta_{\Lambda_m} \end{pmatrix} \boldsymbol{\alpha} \right] \|_2^2 + \sum_{i=1}^m \mathfrak{w}_i^2 \|\mathbf{h}_i\|_2^2 \tag{3.7}$$

$$\equiv \| \begin{pmatrix} \mathfrak{w}_1 \mathbf{g}_1 \\ \vdots \\ \mathfrak{w}_m \mathbf{g}_m \end{pmatrix} - (\mathbf{W} \otimes R) \begin{pmatrix} \Delta_{\Lambda_1} \\ \vdots \\ \Delta_{\Lambda_m} \end{pmatrix} \boldsymbol{\alpha} \|_2^2 + \sum_{i=1}^m \mathfrak{w}_i^2 \|\mathbf{h}_i\|_2^2, \tag{3.8}$$

where $\otimes$ denotes the Kronecker product. Note that $\mathbf{h}_i$ is the component of the corresponding $\mathbf{x}_i$ that is orthogonal to the range of Z_ℓ and thus beyond the reach of optimization with respect to $\boldsymbol{\alpha}$. Hence, we actually need the economy-size QR factorization $Z_\ell = Q(:, 1:\ell)R$; $\mathbf{g}_i = Q(:, 1:\ell)^* \mathbf{x}_i$.

Remark 3.2. Since in a DMD framework $Z_\ell = U_k \widetilde{B}_\ell$, $\widetilde{B}_\ell \in \mathbb{C}^{k \times \ell}$, the factorization of Z_ℓ is obtained from the QR factorization $\widetilde{B}_\ell = Q'R$, where by the essential uniqueness $Q(:, 1:\ell) = U_k Q'$. (Even if the columns of Z_ℓ are the refined Ritz vectors, they have a representation of the form $Z_\ell = U_k \widetilde{B}_\ell$, only with a different matrix $\widetilde{B}_\ell$.) Hence, the QR factorization of Z_ℓ is not necessarily computed explicitly from the explicitly computed Z_ℓ; the more economic way is using the QR factorization of $\widetilde{B}_\ell$.

Further, if $\mathbf{X}_m = U_m\Sigma_m V_m^*$ is the thin SVD of $\mathbf{X}_m$, then

$$\begin{aligned}\Omega^2(\boldsymbol{\alpha}) &= \left\| \left[U_m\Sigma_m V_m^* - U_k\widetilde{B}_\ell\Delta_{\boldsymbol{\alpha}}\begin{pmatrix}\Lambda_1 & \Lambda_2 & \dots & \Lambda_m\end{pmatrix}\right]\mathbf{W}\right\|_F^2 \quad (\text{since } U_k = U_m U_m^* U_k)\\ &= \left\| \left[\Sigma_m V_m^* - \begin{pmatrix}\widetilde{B}_\ell \\ \mathbf{0}_{m-k,\ell}\end{pmatrix}\Delta_{\boldsymbol{\alpha}}\begin{pmatrix}\Lambda_1 & \Lambda_2 & \dots & \Lambda_m\end{pmatrix}\right]\mathbf{W}\right\|_F^2 \\ &= \left\| \left[\Sigma_k V_k^* - \widetilde{B}_\ell\Delta_{\boldsymbol{\alpha}}\begin{pmatrix}\Lambda_1 & \Lambda_2 & \dots & \Lambda_m\end{pmatrix}\right]\mathbf{W}\right\|_F^2 + \sum_{j=k+1}^{m}\sigma_j^2\|V_m^*(j,:)\mathbf{W}\|_2^2,\end{aligned}$$

and (as in the unweighted case considered in [26]) the reconstruction is done with the projections of the snapshots onto the span of the leading k left singular vectors (principal components). In the corresponding basis, the snapshots are the columns of $\Sigma_k V_k^*$ ($\mathbf{x}_i \equiv (\Sigma_k V_k^*)(:,i)$), the Ritz vectors are in $\widetilde{B}_\ell$ ($Z_\ell \equiv \widetilde{B}_\ell$), and we may proceed with the QR factorization $\widetilde{B}_\ell = Q'R$, thus removing the large dimension n in the very first step. This is included as a special case in the generic description (3.4)–(3.6) which appears in other applications (outside the DMD framework; see, e.g., [32]), and it is thus preferred as a general form of the structured LS problem.

Remark 3.3. An efficient alternative approach to reducing the dimension is the QR-compressed scheme [15, section 5.2.1], which replaces the ambient space dimension n with $m+1$ even before the DMD computation. In this projected framework, the QR factorization of Z_ℓ then reduces the dimension from $m+1$ to ℓ. This has been successfully used in [14], where Z_ℓ is computed directly (without the DMD) from the eigenvectors of the companion matrix (inverse of the Vandermonde matrix from (2.1)). For large scale problems, parallel implementation of the QR factorization is used in [37] as an efficient preprocessing step for the SVD in Line 1 of Algorithm 1.

3.2. Structure of the LS problem and normal equations solution. The objective function $\Omega^2(\boldsymbol{\alpha})$ has a very particular structure that allows a rather elegant explicit formula [26, section II.B] for the optimal $\boldsymbol{\alpha}$ via the normal equations in the unweighted case ($\mathbf{W} = \mathbb{I}_m$). Here, we generalize the formula to the weighted case. Then, we discuss an additional structure.

THEOREM 3.4. *With the notation as above, the unique solution $\boldsymbol{\alpha}$ of the LS problem* (3.1) *is*

$$\boldsymbol{\alpha} = [(R^*R)\circ(\overline{\mathbb{V}_{\ell,m}\mathbf{W}^2\mathbb{V}_{\ell,m}^*})]^{-1}[(\overline{\mathbb{V}_{\ell,m}\mathbf{W}}\circ(R^*G\mathbf{W}))\mathbf{e}], \tag{3.9}$$

where $G = \begin{pmatrix}\mathbf{g}_1 & \dots & \mathbf{g}_m\end{pmatrix}$, $\mathbf{e} = \begin{pmatrix}1 & \dots & 1\end{pmatrix}^T$. *In terms of the original data,*

$$\boldsymbol{\alpha} = [(Z_\ell^*Z_\ell)\circ(\overline{\mathbb{V}_{\ell,m}\mathbf{W}^2\mathbb{V}_{\ell,m}^*})]^{-1}[(\overline{\mathbb{V}_{\ell,m}\mathbf{W}}\circ(Z_\ell^*\mathbf{X}_m\mathbf{W}))\mathbf{e}]. \tag{3.10}$$

Proof. Note that $\boldsymbol{\alpha}$ actually solves

$$\|(\mathbf{W}\otimes\mathbb{I}_\ell)\left[\vec{\mathbf{g}} - S\boldsymbol{\alpha}\right]\|_2 \longrightarrow \min, \quad \text{where } \vec{\mathbf{g}} = \begin{pmatrix}\mathbf{g}_1\\ \vdots \\ \mathbf{g}_m\end{pmatrix}, \quad S = (\mathbb{I}_m\otimes R)\begin{pmatrix}\Delta_{\Lambda_1}\\ \vdots \\ \Delta_{\Lambda_m}\end{pmatrix} \equiv \begin{pmatrix}R\Delta_{\Lambda_1}\\ \vdots \\ R\Delta_{\Lambda_m}\end{pmatrix}. \tag{3.11}$$

Hence, the optimal $\boldsymbol{\alpha}$ is given by the Moore–Penrose generalized inverse, $\boldsymbol{\alpha} = S_{\mathfrak{w}}^\dagger\vec{\mathbf{g}}_{\mathfrak{w}}$, where $\vec{\mathbf{g}}_{\mathfrak{w}} = (\mathbf{W}\otimes\mathbb{I}_\ell)\vec{\mathbf{g}}$, $S_{\mathfrak{w}} = (\mathbf{W}\otimes\mathbb{I}_\ell)S$, $S_{\mathfrak{w}}^\dagger = (S_{\mathfrak{w}}^*S_{\mathfrak{w}})^{-1}S_{\mathfrak{w}}^*$, i.e.,

$$\boldsymbol{\alpha} = S_{\mathfrak{w}}^\dagger\vec{\mathbf{g}}_{\mathfrak{w}} = (S_{\mathfrak{w}}^*S_{\mathfrak{w}})^{-1}S_{\mathfrak{w}}^*\vec{\mathbf{g}}_{\mathfrak{w}} = \left(\sum_{k=1}^{m}\mathfrak{w}_k^2\Delta_{\Lambda_k}^*R^*R\Delta_{\Lambda_k}\right)^{-1}\sum_{i=1}^{m}\mathfrak{w}_i\Delta_{\Lambda_i}^*(\mathfrak{w}_iR^*\mathbf{g}_i). \tag{3.12}$$

Further, using the Hadamard matrix product $\circ$ and the elementwise conjugation $\overline{\cdot}$, we can write

$$\sum_{k=1}^{m} \mathfrak{w}_k^2 \Delta_{\Lambda_k}^* R^* R \Delta_{\Lambda_k} = \sum_{k=1}^{m} \mathfrak{w}_k^2 (R^*R) \circ (\overline{\Lambda_k} \Lambda_k^T) = (R^*R) \circ \sum_{k=1}^{m} \mathfrak{w}_k^2 \overline{\Lambda_k} \Lambda_k^T \tag{3.13}$$

$$= (R^*R) \circ (\overline{\mathbb{V}}_{\ell,m} \mathbf{W}^2 \mathbb{V}_{\ell,m}^T) = (R^*R) \circ (\overline{\mathbb{V}_{\ell,m} \mathbf{W}^2 \mathbb{V}_{\ell,m}^*}), \quad \text{and} \tag{3.14}$$

$$\sum_{i=1}^{m} \mathfrak{w}_i \Delta_{\Lambda_i}^* (\mathfrak{w}_i R^* \mathbf{g}_i) = (\overline{\mathbb{V}_{\ell,m} \mathbf{W}} \circ (R^* G \mathbf{W}))\mathbf{e}. \tag{3.15}$$

Note that Theorem 3.4 does not use the Vandermonde structure of $\mathbb{V}_{\ell,m}$; it only requires full row rank; similarly the triangular structure of R is not important, and we only require R to have full column rank. Hence, its result can be stated in a more general form. ◻

The formula (3.9), (3.10), which appears to be new, contains the formula from [26, section II.B] as a special unweighted case. Since the Hadamard product of two positive definite matrices remains positive definite, the solution of (3.9) is obtained using the Cholesky factorization of $(R^*R) \circ (\overline{\mathbb{V}_{\ell,m} \mathbf{W}^2 \mathbb{V}_{\ell,m}^*})$ followed by forward and backward substitutions. The complexity of (3.9) in terms of arithmetic operations is dominated by $O(m\ell^2) + O(\ell^3)$. Typically, $m \gg \ell$.

Relations (3.11)–(3.15), which imply (3.9), hide an elegant structure that we discuss next in section 3.2.1.

3.2.1. On Khatri–Rao structure of the snapshot reconstruction problem. Recall that, for two column partitioned matrices $A = (a_1, a_2, \ldots, a_n) \in \mathbb{C}^{p \times n}$ and $B = (b_1, b_2, \ldots, b_n) \in \mathbb{C}^{q \times n}$, their Khatri–Rao product is defined as the columnwise Kronecker product:

$$A \odot B = \begin{pmatrix} a_1 \otimes b_1 & a_2 \otimes b_2 & \ldots & a_n \otimes b_n \end{pmatrix} \in \mathbb{C}^{pq \times n}. \tag{3.16}$$

The following proposition contains well-known facts; it is included solely for the reader's convenience. For a more detailed study of matrix products involved in this section we refer the reader to [22, Chapters 4 and 5].

Proposition 3.5. *The Khatri–Rao product satisfies the following relations:*

- *For any matrices A, B, C, D of appropriate dimensions,*

$$(AB) \odot (CD) = (A \otimes C)(B \odot D). \tag{3.17}$$

- *For any three matrices A, B, C of appropriate dimensions and with B diagonal,*

$$\mathrm{vec}(ABC) = (C^T \odot A)\,\mathrm{diag}(B). \tag{3.18}$$

(Recall that in general we have, for any B, $\mathrm{vec}(ABC) = (C^T \otimes A)\mathrm{vec}(B)$.)

- *For any two matrices A and B with the same number of columns,*

$$(A \odot B)^T (A \odot B) = (B \odot A)^T (B \odot A) = (A^T A) \circ (B^T B), \tag{3.19}$$

$$(A \odot B)^* (A \odot B) = (B \odot A)^* (B \odot A) = (A^* A) \circ (B^* B). \tag{3.20}$$

Let Π be a permutation matrix whose columns are the columns of the $(m\ell \times m\ell)$ identity taken in the order of the following permutation $\varpi_{\ell,m}$:

$$\varpi_{\ell,m} = \begin{pmatrix} 1 & 2 & 3 & \dots & m & m{+}1 & m{+}2 & m{+}3 & \dots & 2m & \dots & m(\ell{-}1){+}1 & \dots & m\ell \\ 1 & \ell{+}1 & 2\ell{+}1 & \dots & (m{-}1)\ell{+}1 & 2 & \ell{+}2 & 2\ell{+}2 & \dots & (m{-}1)\ell{+}2 & \dots & \ell & \dots & \ell{+}(m{-}1)\ell \end{pmatrix}. \tag{3.21}$$

(In MATLAB, we can generate $\varpi_{\ell,m}$ as

$$\varpi_{\ell,m} = \texttt{reshape(reshape(}1:\ell*m\texttt{,}\ell,m\texttt{)',}\ell*m\texttt{,1)},$$

which intuitively describes $\varpi_{\ell,m}$ as transformation of indices between a column and a row major ordering of a two dimensional array.)

PROPOSITION 3.6. *The LS problem* (3.11) *has the following Khatri–Rao structure:*

(i) $S = \Pi(R \odot \mathbb{V}_{\ell,m}^T) = \mathbb{V}_{\ell,m}^T \odot R$, *or, equivalently,* $S(\varpi_{\ell,m},:) = R \odot \mathbb{V}_{\ell,m}^T$.

(ii) $(R^*R)\circ(\overline{\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*}) = (R\odot\mathbb{V}_{\ell,m}^T)^*(R\odot\mathbb{V}_{\ell,m}^T) = (\mathbb{V}_{\ell,m}^T\odot R)^*(\mathbb{V}_{\ell,m}^T\odot R) = S^*S$.

(iii) *For any* $m \times m$ *matrix* $\mathbf{W}$, $(\mathbf{W}\otimes\mathbb{I}_m)S = (\mathbf{W}\mathbb{V}_{\ell,m}^T)\odot R$.

Proof. The proof is straightforward. ∎

Since Π is orthogonal, the problem reduces to computing the QR factorization of $R \odot \mathbb{V}_{\ell,m}^T$, and the objective (3.11) can be written as

$$\|\vec{\mathbf{g}} - S\boldsymbol{\alpha}\|_2 \equiv \|\vec{\mathbf{g}} - \Pi(R\odot\mathbb{V}_{\ell,m}^T)\boldsymbol{\alpha}\|_2 \equiv \|\Pi^T\vec{\mathbf{g}} - (R\odot\mathbb{V}_{\ell,m}^T)\boldsymbol{\alpha}\|_2 \longrightarrow \min. \tag{3.22}$$

Note that $\vec{\mathbf{g}} = \mathrm{vec}(G)$ and that $\Pi^T\vec{\mathbf{g}} = \mathrm{vec}(G^T)$, where G is from (3.15), and vec$(\cdot)$ denotes the operator that reshapes a matrix by stacking its columns in a tall vector. Normal equations are attractive in this setting because of avoiding the row dimension $m\ell$ of the Khatri–Rao product.

3.2.2. Application in antenna array processing. The matrix LS approximation with the Khatri–Rao structure also appears in other applications. An example is multistatic antenna array processing, where one determines the scattering coefficients α_i by solving $\|H - G_{rec}\mathrm{diag}(\alpha_i)_{i=1}^{\ell}G_{tr}^T\|_F \longrightarrow \min_{\alpha_i}$. Here H stands for the multistatic data matrix, and the columns $G_{rec}(:,i)$ and $G_{tr}(:,i)$ are the steering vectors associated with wave propagation between the receiving and transmitting array, respectively, and the ith scatterer. We refer the reader to [32], where the unweighted (i.e., $\mathbf{W} = \mathbb{I}$) version of (3.9) is derived as the normal equations solution based on the properties of the Khatri–Rao product. In fact, Theorem 3.4 provides a generalization of [32] to weighted LS. (The formulas in Theorem 3.4 do not use the fact that $\mathbb{V}_{\ell,m}$ is a Vandermonde matrix.)

3.3. An improvement of the sparsity promoting DMDSP. In the notation of [26], with $\ell = k$ (i.e., using k modes, where k is the dimension of the POD basis), we can write $\Omega^2(\boldsymbol{\alpha})$ in the unweighted case ($\mathbf{W} = \mathbb{I}_m$) as

$$\Omega^2(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^*\mathbb{P}\boldsymbol{\alpha} - \mathbf{q}^*\boldsymbol{\alpha} - \boldsymbol{\alpha}^*\mathbf{q} + \|\Sigma_k\|_F^2, \text{ where } \mathbb{P} = (Z_k^*Z_k)\circ(\overline{\mathbb{V}_{k,m}\mathbb{V}_{k,m}^*}), \tag{3.23}$$

and $\mathbf{q} = \mathrm{diag}(\overline{\mathbb{V}_{k,m}V_k\Sigma_kB_k})$, where $Z_k = U_kB_k$ and B_k is the matrix of the Ritz vectors computed in Line 5 of Algorithm 1.

Remark 3.7. Let us compare this $\mathbf{q}$ and the corresponding $\mathbf{q_0} = [(\overline{\mathbb{V}_{k,m}}\circ(Z_k^*\mathbf{X}_m))\mathbf{e}]$ in relation (3.10) (with $\ell = k$ and $W = \mathbb{I}_m$). Since $Z_k^*\mathbf{X}_m = B_k^*\Sigma_kV_k^*$, by [22, Lemma 5.1.3], we have

$$\mathbf{q_0} = [(\overline{\mathbb{V}_{k,m}}\circ(B_k^*\Sigma_kV_k^*))\mathbf{e}] = \mathrm{diag}(\overline{\mathbb{V}_{k,m}}(B_k^*\Sigma_kV_k^*)^T = \mathrm{diag}(\overline{\mathbb{V}_{k,m}V_k}\Sigma_k\overline{B_k}) = \mathbf{q}.$$

The constrained problem (2.7) ($\Omega^2(\boldsymbol{\alpha}) \longrightarrow \min$, $E^T\boldsymbol{\alpha} = \mathbf{0}$, where the $k \times s$ matrix E is described in section 2.2.1) is in the polishing phase of DMDSP solved via the variation of the Lagrangian (see [26, Appendix C]), which yields the augmented $(k+s) \times (k+s)$ system of equations

$$\begin{pmatrix} \mathbb{P} & E \\ E^T & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\nu} \end{pmatrix} = \begin{pmatrix} \mathbf{q} \\ \mathbf{0} \end{pmatrix} \tag{3.24}$$

for the amplitudes $\boldsymbol{\alpha}$ and the vector of Lagrange multipliers $\boldsymbol{\nu}$. The amplitudes are then explicitly expressed as (see [26, Appendix C] and [25])

$$\boldsymbol{\alpha} = \begin{pmatrix} \mathbb{I}_k & \mathbf{0} \end{pmatrix} \left[\begin{pmatrix} \mathbb{P} & E \\ E^T & \mathbf{0} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{q} \\ \mathbf{0} \end{pmatrix} \right]. \tag{3.25}$$

A more efficient procedure is to write $\boldsymbol{\alpha}$ as $\boldsymbol{\alpha} = E_\perp \boldsymbol{\beta}$, where $E_\perp \in \mathbb{C}^{k\times(k-s)}$ contains the remaining columns of the identity (complementary to the columns of E), and $\boldsymbol{\beta} \in \mathbb{C}^{k-s}$ is a new (unconstrained) variable. The objective function now reads $\Omega^2(\boldsymbol{\alpha})_{E^T\boldsymbol{\alpha}=\mathbf{0}} \equiv \Omega^2(\boldsymbol{\beta}) = \boldsymbol{\beta}^*(E_\perp^*\mathbb{P}E_\perp)\boldsymbol{\beta} - (E_\perp^*\mathbf{q})^*\boldsymbol{\beta} - \boldsymbol{\beta}^*(E_\perp^*\mathbf{q}) + \|\Sigma_k\|_F^2$, where $E_\perp^*\mathbb{P}E_\perp$ is a $(k-s)\times(k-s)$ main submatrix of $\mathbb{P}$, and the explicit solution is (cf. Theorem 3.4)

$$\boldsymbol{\alpha} = E_\perp \left[(E_\perp^*\mathbb{P}E_\perp)^{-1}(E_\perp^*\mathbf{q}) \right]. \tag{3.26}$$

Schematically, the coefficient matrices of the linear systems in (3.25) and (3.26) can be illustrated as follows:

$$\begin{pmatrix} \mathbb{P} & E \\ E^T & \mathbf{0} \end{pmatrix} = \begin{pmatrix}
* & * & * & * & * & * & * & * & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\
* & \circledast & * & * & \circledast & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 \\
* & * & * & * & * & * & * & * & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
* & * & * & * & * & * & * & * & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
* & \circledast & * & * & \circledast & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 \\
* & * & * & * & * & * & * & * & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
* & * & * & * & * & * & * & * & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\
* & * & * & * & * & * & * & * & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}
\end{pmatrix}, \quad E_\perp^*\mathbb{P}E_\perp = \begin{pmatrix} \circledast & \circledast \\ \circledast & \circledast \end{pmatrix}, \quad E_\perp = \begin{pmatrix} 0 & 0 \\ \mathbf{1} & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \mathbf{1} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{3.27}$$

Remark 3.8. To appreciate the difference between (3.25) and (3.26), consider for example $k = m = 1200$ and only $\ell = 30$ modes. The augmented system in (3.25) is of dimension $1200 + 1170 = 2370$, while the same result is obtained from the system (3.26) of dimension 30. Given the cubic complexity of the solution of linear systems, the speedup of (3.26) over (3.25) is considerable. It should be noted that (3.24) is a particular case of the saddle point problem that is solved by specially tailored methods; for an overview we refer the reader to [4]; see also, e.g., [19]. However, our proposed alternative (3.26) is simpler and much more efficient.

Remark 3.9. The same scheme applies in the more general case with arbitrary full column rank matrix E. The only technical difference is in constructing the complement $E_\perp$. Let $E = Q\left(\begin{smallmatrix} R \\ \mathbf{0} \end{smallmatrix}\right) \equiv \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}\left(\begin{smallmatrix} R \\ \mathbf{0} \end{smallmatrix}\right)$ be the full QR factorization. It is then clear that we can take $E_\perp = Q_2$, and the above procedure applies verbatim.

3.4. An example: Q2D Kolmogorov flow. We illustrate the discussion in this section using a concrete numerical example. The goals are twofold:

(i) to show the importance of recomputing the coefficients from the full reconstruction (1.1) to achieve an improved approximation (1.2) in the LS sense, and the performance of the normal equations;

(ii) to illustrate the benefits of using weights (3.1) to emphasize the importance of a particular subset of the data snapshots, and to test the solution formula from Theorem 3.4.

For a reconstruction $\widehat{\mathbf{x}}_i$ of $\mathbf{x}_i$, we use the relative error $\epsilon_i = \|\widehat{\mathbf{x}}_i - \mathbf{x}_i\|_2/\|\mathbf{x}_i\|_2$ as an accuracy measure.[3] Recall that the LS procedure minimizes the sum of squares $\sum_i \|\widehat{\mathbf{x}}_i - \mathbf{x}_i\|_2^2$, so that in the case of large variations in the norms $\|\mathbf{x}_i\|_2$, the smallest $\mathbf{x}_i$'s might be poorly reconstructed; this can be improved by suitable weighting coefficients $\mathfrak{w}_i$.

Further, we might be interested in a particular time interval, and the data snapshots with the corresponding discrete indices can be given larger weights in (3.1). We illustrate this with a CFD example.

Example 3.10. We use the simulation data of a 2D model obtained by depth-averaging the Navier–Stokes equations for a shear flow in a thin layer of electrolyte suspended on a thin lubricating layer of a dielectric fluid; see [48, 41] for more detailed description of the experimental setup and numerical simulations.[4] The data represent 401 snapshots of the vorticity field on a rectangular domain, discretized using a 359×279 uniform grid. The DMD is computed using the Schmid DMD (Algorithm 1), and the companion matrix based version (designated as Krylov+DFT) that works with the original Krylov sequence and uses the discrete Fourier transformation (DFT) (see [14]). We selected ℓ modes by taking the indices of the ℓ absolutely dominant Ritz values, and with suitable weights $\mathfrak{w}_i$ we enforced higher accuracy on selected subsets of the snapshots. Sample outputs are given in Figure 3.1.

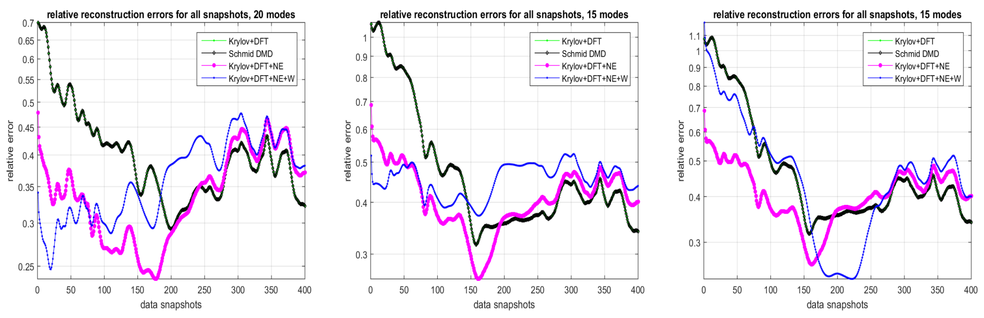

FIG. 3.1. *(Example 3.10) The relative errors ϵ_i of the reconstruction of the $m = 400$ snapshots using $\ell = 20$ and $\ell = 15$ modes. In all cases, the errors of the Schmid DMD and the Krylov+DFT algorithm (with the coefficients from the full reconstruction (1.1)) are nearly the same, so the graphs overlap. Recomputing the coefficients using the normal equation significantly reduces the error (-., Krylov+DFT+NE). The blue curves show the effects of weighting to the reconstruction error. The $\mathfrak{w}_i$'s are set to one except* (i) *the left and the central panels: $\mathfrak{w}_1 = \cdots = \mathfrak{w}_{50} = 10$;* (ii) *the rightmost panel: $\mathfrak{w}_{200} = \cdots = \mathfrak{w}_{250} = 10$. (Color available online.)*

In Example 3.10, the coefficients computed by sparsifying the solution of (3.25) and, more efficiently,[5] by solving the normal equations (i.e., solving (3.26)) agree up to $O(10^{-14})$. In the next section, we analyze the subtle numerical details of the solution of the normal equations.

[3]In a more general setting, the error would be measured in an appropriate energy norm; see Remark 3.1.

[4]We thank Michael Schatz, Balachandra Suri, Roman Grigoriev, and Logan Kageorge from the Georgia Institute of Technology for providing us with the data.

[5]cf. Remark 3.8.

4. Sensitivity analysis of normal equations method. Although elegant and efficient, the formula (3.9) has a drawback typical of normal equation based LS solutions; it squares the condition number ($\kappa_2(R^*R) = \kappa_2(R)^2$ and $\kappa_2(\overline{\mathbb{V}_{\ell,m}\mathbf{W}^2\mathbb{V}_{\ell,m}^*}) = \kappa_2(\mathbb{V}_{\ell,m}\mathbf{W})^2$) and uses a Cholesky factorization based solver with the coefficient matrix $C \equiv (R^*R) \circ (\overline{\mathbb{V}_{\ell,m}\mathbf{W}^2\mathbb{V}_{\ell,m}^*})$. In theory, by a Schur theorem [22, Theorem 5.2.1], the Hadamard product of a positive definite matrix and a positive semidefinite matrix with positive diagonal is positive definite, and thus C possesses the Cholesky factor. In section 4.1 we show by example that this is not necessarily true in finite precision (floating point) computation. Using perturbation analysis, in section 4.2 we identify the relevant condition numbers, which behave (due to their scaling invariance) better than the standardly used condition numbers. Further, we explain how the Hadamard product structure of C improves this condition number and thus allows accurate computation in cases deemed ill-conditioned by the classical perturbation theory. We conclude this section in section 4.3, with a simple example that illustrates the need for an alternative to the normal equations, and with a decision tree for switching to a new proposed QR factorization based algorithm.

4.1. Limitations of normal equations. In extremely ill-conditioned cases, however, it can happen that both computed and stored matrices R^*R and $\mathbb{V}_{\ell,m}\mathbf{W}^2\mathbb{V}_{\ell,m}^*$ are exactly singular (or even indefinite) so that the Cholesky factorization of C might fail. Recall that, in finite precision computation, the Cholesky factorization may fail even if the matrix stored in the machine memory is exactly positive definite. Moreover, the factorization may even succeed with an indefinite matrix on input. We refer the reader to [13] for details on computing the Cholesky factorization in floating point arithmetic.

Hence, the formula (3.9) should be deployed with great care, and its sensitivity must be well understood as it is used in a variety of applications.

The following example, although contrived, illustrates the above discussion and should be enough to call into question the scope of applicability of the normal equations.

Example 4.1. (All computations done in MATLAB R2015.a, with the double precision roundoff unit ε = `eps=2.220446049250313e-16`.) Let $\ell = 3$, $m = 4$, $\xi = \sqrt{\varepsilon}$, $\lambda_1 = \xi$, $\lambda_2 = 2\xi$, and $\lambda_3 = 0.2$, so that the Vandermonde section $\mathbb{V}_{\ell,m}$ equals

$$\mathbb{V}_{\ell,m} = \begin{pmatrix} 1.000000000000000e{+}00 & 1.490116119384766e{-}08 & 2.220446049250313e{-}16 & 3.308722450212111e{-}24 \\ 1.000000000000000e{+}00 & 2.980232238769531e{-}08 & 8.881784197001252e{-}16 & 2.646977960169689e{-}23 \\ 1.000000000000000e{+}00 & 2.000000000000000e{-}01 & 4.000000000000001e{-}02 & 8.000000000000002e{-}03 \end{pmatrix},$$

and let the triangular factor R be

$$R = \begin{pmatrix} 1 & 1 & 1 \\ 0 & \xi/2 & \xi \\ 0 & 0 & \xi \end{pmatrix} = \begin{pmatrix} 1.000000000000000e{+}00 & 1.000000000000000e{+}00 & 1.000000000000000e{+}00 \\ 0 & 7.450580596923828e{-}09 & 1.490116119384766e{-}08 \\ 0 & 0 & 1.490116119384766e{-}08 \end{pmatrix}.$$

Set $\mathbf{W} = \mathbb{I}_m$. From the singular values of $\mathbb{V}_{\ell,m}$ and R, computed as

	$\sigma_i(\mathbb{V}_{\ell,m})$	$\sigma_i(R)$
1	$1.736092504099537e+00$	$1.732050807568878e+00$
2	$1.662733207986230e-01$	$1.555891180151553e-08$
3	$2.105723035894610e-09$	$4.119745457168918e-09$

,

we see that their condition numbers are of the order of $1/\sqrt{\varepsilon}$, which leaves the possibility of computation (involving R and $\mathbb{V}_{\ell,m}$) with an $O(\sqrt{\varepsilon})$ accuracy; in the IEEE 64 bit arithmetic this means seven to eight decimal places. Moreover, both $\mathbb{V}_{\ell,m}$ and R are of full rank and the closest rank deficient matrices are at distances $\sigma_3(\mathbb{V}_{\ell,m}) > 10^{-9}$ and $\sigma_3(R) > 10^{-9}$, respectively. Further, since both matrices are entrywise nonnegative, both $\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*$ and R^*R are computed to nearly full machine precision[6]; the same holds for $C = (R^*R) \circ (\overline{\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*})$. Although all three matrices are by design positive definite and computed in each entry to full machine precision, none of them is numerically positive definite; the Cholesky factorization fails on each of them:

```
>> chol(Vlm*Vlm')
Error using chol
Matrix must be positive definite.
>> chol(R'*R)
Error using chol
Matrix must be positive definite.
```

```
>> chol((R'*R).*(Vlm*Vlm'))
Error using chol
Matrix must be positive definite.
```

Hence, the normal equation solver (which assumes positive definite linear system) might fail; and even if it succeeded the result might be unacceptably inaccurate.

On the other hand, the unweighted form ($\mathbf{W} = \mathbb{I}_m$) of the formula (3.9) has been successfully used in the computational DMD framework, despite the fact that it is based on normal equation (an ill-advised approach) that involves potentially ill-conditioned matrices. We offer an explanation and provide a way to estimate a priori whether this method can produce a sufficiently accurate result.

4.2. Condition number estimates. In some cases, the structure of the perturbations (e.g., backward error in matrix operations in finite precision arithmetic) allows condition numbers that are invariant under certain scalings by diagonal matrices and thus potentially much smaller than the traditional condition number. This is true, in particular, for computations with positive definite matrices, and in this subsection we prove that scaling invariance of the relevant condition number is the key for understanding numerical properties of the normal equations based solution of the reconstruction problem (1.2).

4.2.1. Review of perturbation theory for the Cholesky factorization and positive definite systems. Based on [13], we know that floating point Cholesky factorization $C = LL^*$ (L lower triangular with positive diagonal) of C is feasible if[7] the matrix $C_s = (c_{ij}/\sqrt{c_{ii}c_{jj}})_{i,j=1}^{\ell}$ is well conditioned, i.e., if (roughly) $\|C_s^{-1}\|_2 < 1/\boldsymbol{\varepsilon}$. Further, if $\widetilde{L}$ is the computed Cholesky factor, then $\widetilde{L}\widetilde{L}^* = C+\delta C$, where the backward error $\delta C = (\delta c_{ij})_{i,j=1}^{\ell}$ of the computed factorization is such that

$$\max_{i,j} \frac{|\delta c_{ij}|}{\sqrt{c_{ii}c_{jj}}} \leq O(\ell)\boldsymbol{\varepsilon}. \tag{4.1}$$

If we set $D_C = \operatorname{diag}(\sqrt{c_{ii}})_{i=1}^{\ell}$, then the relation above can be written as

$$\widetilde{L}\widetilde{L}^* = C + \delta C = D_C(C_s + D_C^{-1}\delta C D_C^{-1})D_C, \;\; C_s = D_C^{-1} C D_C^{-1}, \tag{4.2}$$

[6]In this example, there are no proper subtractions, and each entry in both matrices is approximated with the corresponding floating point number up to an $O(\boldsymbol{\varepsilon})$ relative error—practically the best one can hope for.

[7]Actually, if no additional structure is assumed, here we have an "if and only if." This means that, in the absence of additional properties such as sparsity or sign distribution of matrix entries, the failure of the Cholesky decomposition means that the matrix cannot be considered numerically positive definite.

where the entries of $D_C^{-1}\delta C D_C^{-1}$ are estimated by (4.1). Note that C_s has unit diagonal and that all its off-diagonal entries are in absolute value below one.

By [17, Theorem 3.1], we can write $\widetilde{L} = L(\mathbb{I}+\Gamma)$, where $\mathbb{I}+\Gamma$ is lower triangular with positive diagonal and the size of the multiplicative error (note: $\delta L \equiv \widetilde{L} - L = L\Gamma$) can be bounded by

$$\|\Gamma\|_2 \leq O(\ell \log_2 \ell)\boldsymbol{\varepsilon}\|C_s^{-1}\|_2 \leq O(\ell \log_2 \ell)\boldsymbol{\varepsilon}\kappa_2(C_s). \tag{4.3}$$

Further, if we solve the linear system $Cx = b \neq \mathbf{0}$ using the Cholesky factor in the forward and backward substitutions, then the computed solution $\widetilde{x}$ satisfies (see [13, Theorem 2.1])

$$\frac{\|D_C(\widetilde{x} - C^{-1}b)\|_2}{\|D_C\widetilde{x}\|_2} \leq g(\ell)\boldsymbol{\varepsilon}\kappa_2(C_s), \tag{4.4}$$

where $g(\ell)$ is a modest function of the dimension. Note that this implies componentwise error bound for each $\widetilde{x}_i \neq 0$:

$$\frac{|\widetilde{x}_i - (C^{-1}b)_i|}{|\widetilde{x}_i|} \leq \left[\frac{\|D_C\widetilde{x}\|_2}{\sqrt{c_{ii}}|\widetilde{x}_i|}\right] g(\ell)\boldsymbol{\varepsilon}\kappa_2(C_s), \quad \text{where} \quad \left[\frac{\|D_C\widetilde{x}\|_2}{\sqrt{c_{ii}}|\widetilde{x}_i|}\right] \geq 1. \tag{4.5}$$

For further discussion we refer the reader to [13].

Hence, it is the scaled condition number $\kappa_2(C_s)$ that determines the sensitivity to perturbations and the accuracy of the computed amplitudes, and not $\kappa_2(C)$. This is important because of the following theorem by Van der Sluis [50].

THEOREM 4.2. *Let $H \in \mathbb{C}^{n\times n}$ be a positive definite Hermitian matrix, $D_H = \mathrm{diag}(\sqrt{H_{ii}})$, and $H_s = D_H^{-1}HD_H^{-1}$. Then $\kappa_2(H_s) \leq n \min_D \kappa_2(DHD)$, where the minimum is taken over all diagonal matrices D.*

Remark 4.3. $\kappa_2(C_s)$ can be at most ℓ times larger than $\kappa_2(C)$, and it can be much smaller.

4.2.2. An estimate of the scaled condition number $\boldsymbol{\kappa_2(C_s)}$. In our case, $C = A \circ B$, with $A = R^*R$, $B = \overline{\mathbb{V}_{\ell,m}\mathbf{W}^2\mathbb{V}_{\ell,m}^*}$, and it is important to understand how $\kappa_2(C_s)$ depends on A and B.

THEOREM 4.4. *Let A and B be Hermitian positive semidefinite matrices with positive diagonal entries, and let $C = A \circ B$. If $A_s = (a_{ij}/\sqrt{a_{ii}a_{jj}})$, $B_s = (b_{ij}/\sqrt{b_{ii}b_{jj}})$, and $C_s = (c_{ij}/\sqrt{c_{ii}c_{jj}})$, then*

$$\max(\lambda_{\min}(A_s), \lambda_{\min}(B_s)) \leq \lambda_i(C_s) \leq \min(\lambda_{\max}(A_s), \lambda_{\max}(B_s)). \tag{4.6}$$

In particular, $\|C_s^{-1}\|_2 \leq \min(\|A_s^{-1}\|_2, \|B_s^{-1}\|_2)$ and $\kappa_2(C_s) \leq \min(\kappa_2(A_s), \kappa_2(B_s))$. If A or B is diagonal, all inequalities in this theorem become equalities.

Proof. The key observation is that $C_s = A_s \circ B_s$, and the proof completes by invoking [22, Theorem 5.3.4], which states the following general property of the Hadamard product:

$$\begin{aligned}\lambda_{\min}(A_s)\lambda_{\min}(B_s) \leq \min_i (A_s)_{ii}\lambda_{\min}(B_s) \leq \lambda_i(C_s) \leq \max_i (A_s)_{ii}\lambda_{\max}(B_s)\\ \leq \lambda_{\max}(A_s)\lambda_{\max}(B_s),\end{aligned}$$

in which we can also swap the roles of A_s and B_s. Finally, note that $(A_s)_{ii} = 1$ for all i. ∎

Remark 4.5. It should be intuitively clear that the Hadamard product $C = A \circ B$ of a positive definite and a positive semidefinite matrix with nonzero diagonal should not worsen the scaled condition number, i.e., that $\kappa_2(C_s)$ is expected to be better than both $\kappa_2(A_s)$ and $\kappa_2(B_s)$. Indeed, C_s has unit diagonal and the off-diagonal entries are $(C_s)_{ij} = \frac{c_{ij}}{\sqrt{c_{ii}c_{jj}}} = \frac{a_{ij}}{\sqrt{a_{ii}a_{jj}}}\frac{b_{ij}}{\sqrt{b_{ii}b_{jj}}} = (A_s)_{ij}(B_s)_{ij}$, where by the (semi)definiteness of A and B both factors on the right-hand side are in modulus below one. That is, the Hadamard product increases the dominance of the diagonal of C_s over any off-diagonal position, as compared to the dominance of the corresponding diagonal entries in A_s and B_s. Hence, it is possible that $\kappa_2(C_s) \ll \min(\kappa_2(A_s), \kappa_2(B_s))$.

In the following example we illustrate such a situation, where R^*R and $\overline{\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*}$ are so close to the boundary of the cone of positive definite matrices that they numerically appear as indefinite, but their Hadamard product possesses a numerical Cholesky factor.

Example 4.6. With the notation of Example 4.1, we use the same $\mathbb{V}_{\ell,m}$ but change the matrix R to

$$R = \begin{pmatrix} 1 & 1 & 1 \\ 0 & \xi & \xi \\ 0 & 0 & \xi/2 \end{pmatrix} = \begin{pmatrix} 1.000000000000000e{+}00 & 1.000000000000000e{+}00 & 1.000000000000000e{+}00 \\ 0 & 1.490116119384766e{-}08 & 1.490116119384766e{-}08 \\ 0 & 0 & 7.450580596923828e{-}09 \end{pmatrix}.$$

If we repeat the experiment with the Cholesky factorizations, we obtain

```
    >> chol(Vlm*Vlm')                    >> chol(R'*R)
    Error using chol                     Error using chol
    Matrix must be positive definite.    Matrix must be positive definite.
>> TC = chol((R'*R).*(Vlm*Vlm'))
TC =
1.000000000000000e+00   1.000000000000000e+00     1.000000002980232e+00
                    0   1.490116119384765e-08     1.999999880790710e-01
                    0                       0     4.079214149695062e-02
```

The condition number of `TC` is estimated to $8.2 \cdot 10^8$, and its inverse is used in backward and forward substitutions when solving the normal equations; see (3.9). Note, however, that in this case $\kappa_2(C_s)\boldsymbol{\varepsilon} > 2$, and perturbation theory [13, 17] provides no guarantee for the accuracy of the computed Cholesky factor. (In fact, in Example 4.12 below, we argue that `TC` is a pretty bad approximation.)

COROLLARY 4.7. *Let* $C \equiv (R^*R) \circ (\overline{\mathbb{V}_{\ell,m}\mathbf{W}^2\mathbb{V}_{\ell,m}^*})$, $C_s = (c_{ij}/\sqrt{c_{ii}c_{jj}})$. *Further, let* $R = R_c\Delta_r$ *and* $\mathbb{V}_{\ell m}\mathbf{W} = \Delta_v(\mathbb{V}_{\ell m}\mathbf{W})_r$ *with diagonal scaling matrices* Δ_r *and* Δ_v *such that* R_c *has unit columns and* $(\mathbb{V}_{\ell m}\mathbf{W})_r$ *has unit rows (in Euclidean norm). Then* $\kappa_2(C_s) \le \min(\kappa_2(R_c)^2, \kappa_2((\mathbb{V}_{\ell,m}\mathbf{W})_r)^2)$.

Proof. Note that, with the notation $A = R^*R$, $B = \overline{\mathbb{V}_{\ell,m}\mathbf{W}^2\mathbb{V}_{\ell,m}^*}$, we can apply Theorem 4.4, where we have $A_s = R_c^*R_c$, $B_s = (\mathbb{V}_{\ell m}\mathbf{W})_r(\mathbb{V}_{\ell,m}\mathbf{W})_r^*$. ∎

4.2.3. Error analysis of computing C in floating point arithmetic.

The condition number $\kappa_2(C_s)$ from Corollary 4.7 will determine the accuracy if we can compute C so that the error $\delta_0 C$ in computing C explicitly is such that $|\delta_0 c_{ij}|/\sqrt{c_{ii}c_{jj}}$ is small for all i, j.

PROPOSITION 4.8. *Let* $A = X^*X$, $B = Y^*Y$ *with* $X \in \mathbb{C}^{m_x\times\ell}$, $Y \in \mathbb{C}^{m_y\times\ell}$, *and let* $C = A \circ B$, *and* $\widetilde{C} = C + \delta_0 C = computed(computed(X^*X) \circ computed(Y^*Y))$.

Then, for all i, j,

$$|\delta_0 c_{ij}| \leq (O(m_x\boldsymbol{\varepsilon}) + O(m_y\boldsymbol{\varepsilon}) + O(m_x m_y \boldsymbol{\varepsilon}^2))\sqrt{c_{ii}c_{jj}}.$$

Proof. We follow the standard steps of floating point error analysis:

$$computed(X^*X) = A + \delta A, \;\; |\delta A| \leq O(m_x\boldsymbol{\varepsilon})|X^*||X|; \;\; |\delta a_{ii}| \leq O(m_x\boldsymbol{\varepsilon})a_{ii},$$

and, by the Cauchy–Schwarz inequality, $|\delta a_{ij}| \leq O(m_x\boldsymbol{\varepsilon})\sqrt{a_{ii}a_{jj}}$. An analogous claim holds for the matrix $computed(Y^*Y) = B + \delta B$. Altogether,

$$\widetilde{c}_{ij} = (a_{ij} + \delta a_{ij})(b_{ij} + \delta b_{ij})(1 + \epsilon_{ij}) = (c_{ij} + \delta a_{ij}b_{ij} + a_{ij}\delta b_{ij} + \delta a_{ij}\delta b_{ij})(1 + \epsilon_{ij}),$$

where (since $c_{ii} = a_{ii}b_{ii}$ and $|b_{ij}| \leq \sqrt{b_{ii}b_{jj}}$) $|\delta a_{ij} b_{ij}| \leq O(m_x\boldsymbol{\varepsilon})\sqrt{a_{ii}a_{jj}}\sqrt{b_{ii}b_{jj}} = O(m_x\boldsymbol{\varepsilon})\sqrt{c_{ii}c_{jj}}$. The remaining error terms are bounded in the same way. ◻

In finite precision computation, we work with $\widetilde{C} = C + \delta_0 C$, so that the backward error (4.1), (4.2) applies to $\widetilde{C}$. The scaled condition number $\kappa_2(\widetilde{C}_s)$ ($\widetilde{C}_s = (\widetilde{c}_{ij}/\sqrt{\widetilde{c}_{ii}\widetilde{c}_{jj}})$) will be moderate if $\kappa_2(C_s)$ is moderate, so all conclusions apply to $\widetilde{C}$ as well. Since both $\delta_0 C$ and δC are of the same type, the overall perturbation in C and its Cholesky factor is bounded in the same way. We omit the details for the sake of brevity.

4.2.4. Numerical examples.

According to Theorem 4.2 and Remark 4.3, $\kappa_2(C_s)$ is potentially much smaller than $\kappa_2(C)$, and, by the discussion in section 4.2, the results may be much better than predicted by the classical perturbation theory based on $\kappa_2(C)$. And, we can very precisely determine whether the normal equation solution will succeed by computing/estimating $\kappa_2(C_s)$. In addition to the contrived Example 4.1 and Example 4.6, we use concrete dynamical systems to show how our theoretical analysis applies in nontrivial computations. We first use the same type of turbulence data as in Example 3.10.

Example 4.9. The data[8] consists of $n_t = 1201$ snapshots of dimensions $n_x \times n_y$ ($n_x = n_y = 128$), representing the (scalar) vorticity field. The $n_x \times n_y \times n_t$ tensor is unfolded into $n_x \cdot n_y \times n_t$ matrix $(\mathbf{x}_1, \ldots, \mathbf{x}_{n_t})$, and $\mathbf{X}_m$ is of dimensions 16384×1200.

After performing a DMD analysis, we have computed the values of the condition numbers of $C \equiv (R^*R) \circ (\overline{\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*})$ and C_s as follows: $\kappa_2(C) > 10^{87} \gg \kappa_2(C_s) \approx$ `8.50e+01`. Using (4.3), we can conclude that the Cholesky factor of C can be computed to high accuracy, despite the fact that $\kappa_2(C) \gg 1/\boldsymbol{\varepsilon}$. A closer look reveals that the high condition number is due to grading; i.e., it is in the diagonal D_C. More precisely, as a result of the Ritz values being from both sides of the unit circle, the diagonal entries of C vary over several orders of magnitude, from (roughly) `1.6e+00` to `4.5e+87`. The absolute values of the computed Ritz values go (roughly) from `0.63` to `1.08`.

Following the discussion in Remark 4.5 and Examples 4.1 and 4.6, the bound of Corollary 4.7 is in this example (luckily) an extreme overestimate because $\kappa_2(R_c)^2 \approx$ `3.05e+13`, $\kappa_2((\mathbb{V}_{\ell,m})_r)^2 \approx$ `9.62e+14`. However, Example 4.1 illustrates the danger of potential failure of the formula (3.9).

Example 4.9 illustrates the theory in this section—high spectral condition numbers of R^*R and $\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*$ do not preclude accurate solution of the normal equations. What matters is that both $\kappa_2(R_c)^2 \approx$ `3.05e+13` and $\kappa_2((\mathbb{V}_{\ell,m})_r)^2 \approx$ `9.62e+14` are

[8]We used this dataset in [14] for testing a new algorithm for representing the snapshots using the Koopman modes.

below $1/\varepsilon \approx$ `4.50e+15`, so both factors in the Hadamard product are positive definite. Note how close these condition numbers are to the critical value at which the (numerical) positive definiteness cannot be guaranteed. In the next example, the condition number of $\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*$ crosses the critical threshold, but normal equations can be used thanks to the preconditioning mechanism discussed in section 4.2.2 and the error estimates in sections 4.2.1 and 4.2.3.

Example 4.10. Chua's circuit is an electronic circuit that exhibits chaos, and it is an excellent example for mathematical study of the chaos-producing mechanism [27, 28]. We ran a simulation of the circuit for $t \in [0, 50]$, with the initial condition $(-0.5, 0.5, 0)$ and rearranged the time snapshots (taken with time step $\delta t = 10^{-3}$) in a tall 147903×700 Hankel matrix H, thus obtaining an approximate Krylov sequence (the columns of H, $\mathbf{x}_i = H(:, i)$) of the Koopman operator as in [2, 8]. The MATLAB function `chol()` failed to compute the Cholesky factor of $\overline{\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*}$; the condition number of the scaled matrix (as defined in section 4.2.1) was estimated as `6.9e+16`. Nonetheless, since $\kappa_2(R_c^* R_c) \approx$ `1.4e+07`, using Corollary 4.7 and Proposition 4.8, we know that the normal equations approach is feasible; in fact, $\kappa_2(C_s) < 100$.

Example 4.11. The Ikeda discrete dynamical system [23, 24], which models the evolution of laser light across a nonlinear optical resonator, provides another nontrivial example that illustrates the applicability of our theory, the limitations of the normal equations approach,[9] and the need for a more robust method. We used the real (two-dimensional) form of the Ikeda map (see, e.g., [35])

$$\begin{array}{ll} x_{n+1} = \boldsymbol{\phi} + \boldsymbol{\psi}(x_n \cos(\boldsymbol{\rho} - \frac{\boldsymbol{\omega}}{1+x_n^2+y_n^2}) - y_n \sin(\boldsymbol{\rho} - \frac{\boldsymbol{\omega}}{1+x_n^2+y_n^2})) \\ y_{n+1} = \qquad \boldsymbol{\psi}(x_n \sin(\boldsymbol{\rho} - \frac{\boldsymbol{\omega}}{1+x_n^2+y_n^2}) - y_n \cos(\boldsymbol{\rho} - \frac{\boldsymbol{\omega}}{1+x_n^2+y_n^2})) \end{array}, \quad n = 0, 1, \ldots,$$

with $\boldsymbol{\phi} = 1$, $\boldsymbol{\psi} = 0.6$, $\boldsymbol{\rho} = 0.4$, $\boldsymbol{\omega} = 6$, and initial condition (x_0, y_0). We generated 3500 snapshots and arranged them in the 5802×600 Hankel matrix, similarly as in Example 4.10. Keeping these parameters fixed, any of the scenarios illustrated in Examples 4.1 and 4.6 can be realized with suitable choices of initial conditions. So, for instance, with $(x_0, y_0) = (1, 2)$, both $\overline{\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*}$ and R^*R are so ill-conditioned that Cholesky factorizations fail, but the Cholesky factorization of $C = (R^*R) \circ (\overline{\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*})$ completed without error. However, the condition number of the scaled matrix C_s is $\kappa_2(C_s) \approx 1.3{\cdot}10^{15}$ ($\kappa_2(C) \approx 3.5{\cdot}10^{16}$) and no accuracy can be expected in the standard sixteen digit arithmetic. On the other hand, the accuracy of the QR factorization based methods is governed by $\sqrt{\kappa_2(C_s)}$, which means that (roughly) eight digits of accuracy are feasible.[10] If we start with $(x_0, y_0) = (-2000, -2500)$, then none of $\overline{\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*}$, R^*R and $C = (R^*R) \circ (\overline{\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*})$ can be considered numerically definite; finite precision Cholesky factorizations fail, and, in particular, the normal equations method based on the Cholesky factorization of C fails. In this example, $\kappa_2(C_s) \approx 10^{17}$. It should be noted that these results are highly sensitive to implementation details[11] and initial conditions; if we change y_0 to -2501, the normal equations have the condition number $\kappa_2(C_s) \approx 7.2 \cdot 10^8$, which allows reasonable accuracy in double precision, but none in the single precision (eight digit) arithmetic.

[9] See [7, section 2.1.4].

[10] The advantage of this smaller condition number is even more important in single precision arithmetic, which is an attractive option in large scale computation using massively parallel GPU architectures; see, e.g., [47].

[11] We used a QR-compressed implementation of the DMD; see Remark 3.3.

4.3. A decision tree. Equipped with the results from section 4.2, we can devise a strategy that chooses the most efficient procedure to deliver the output to the accuracy warranted by the data on the input. Our goal is to develop a reliable software tool that is capable of computing to reasonable accuracy even in the most extreme cases. Since the scaled condition number $\kappa_2(C_s)$ has been identified as the key parameter; we can safely proceed with solving the normal equations if we know a priori that it is moderate. One way to establish that is to use the upper triangular factor R in the QR factorization $Z_\ell = Q\left(\begin{smallmatrix} R \\ \mathbf{0} \end{smallmatrix}\right)$ of the approximate eigenvectors (e.g., Ritz vectors or the refined Ritz vectors) that are usually computed as normalized, i.e., $\|Z_\ell(:,i)\|_2 = 1$ and $R_c \equiv R$. Independent of that normalization, by Corollary 4.7, the condition number that matters is $\kappa_2(R_c) \equiv \kappa_2((Z_\ell)_c)$, where $(Z_\ell)_c$ denotes the matrix Z_ℓ after normalizing its columns.

Recall that the condition number of an $\ell \times \ell$ triangular matrix can be efficiently estimated at an $O(\ell^2)$ cost; see, for instance, the subroutine `XPOCON()` in LAPACK. Moreover, since $Z_\ell = U_k \widetilde{B}_\ell$ (see Remark 3.2) and since R can be computed directly from the $O(k\ell^2)$ QR factorization of the $k \times \ell$ matrix $\widetilde{B}_\ell$, we can estimate $\kappa_2(R_c)$ at the cost of $O(k\ell^2)$. If R is already available (computed for some other use), then the cost of estimating $\kappa_2(R_c)$ is only $O(\ell^2)$, and in that case we consider the estimate available as well. Hence, if one estimates that $\kappa_2(R_c)^2 \ll 1/\boldsymbol{\varepsilon}$, then one can safely use the normal equation solution (3.9).

If an estimate for $\kappa_2(R_c)^2$ is not available (e.g., R not computed) or if one concludes that $\kappa_2(R_c)^2 \gtrapprox 1/\boldsymbol{\varepsilon}$, then one goes on and computes C and an estimate for $\kappa_2(C_s)$, which is then tested against the threshold $1/\boldsymbol{\varepsilon}$. We can organize this in a decision tree as in Algorithm 2. It is designed for the case when the dimensions are sufficiently large and the efficiency of the formula (3.9) is desirable, but not at any price—it is deployed only if it can warrant a certain level of accuracy.

Algorithm 2. Decision tree for selecting a solution method for (3.11).

```
Input:
      • $R$, $\Lambda$, $\ell$, $m$
      • Tolerance level $tol \in (1, 1/\boldsymbol{\varepsilon})$ for acceptable condition number estimate.
 1: if  $R$ available and $\kappa_2(R_c)^2 \le tol$  then
 2:    Solve (3.11) by normal equations, $\boldsymbol{\alpha} = [(R^*R) \circ (\overline{\mathbb{V}_{\ell,m}\mathbf{W}\mathbb{V}_{\ell,m}^*})]^{-1}[(\overline{\mathbb{V}_{\ell,m}\mathbf{W}} \circ (Z_\ell^*\mathbf{X}_m\mathbf{W}))\mathbf{e}]$
 3: else
 4:    Compute $C = (Z_\ell^* Z_\ell) \circ (\overline{\mathbb{V}_{\ell,m}\mathbf{W}^2\mathbb{V}_{\ell,m}^*})$ and estimate $\kappa_2(C_s)$
 5:    if $\kappa_2(C_s) \le tol$ then
 6:       Solve (3.11) by normal equations, $\boldsymbol{\alpha} = [(Z_\ell^* Z_\ell) \circ (\overline{\mathbb{V}_{\ell,m}\mathbf{W}^2\mathbb{V}_{\ell,m}^*})]^{-1}[(\overline{\mathbb{V}_{\ell,m}\mathbf{W}} \circ (Z_\ell^*\mathbf{X}_m\mathbf{W}))\mathbf{e}]$
 7:    else
 8:       $\boxed{\text{Solve (3.11) using QR factorization based solver, without squaring the condition number.}}$
 9:    end if
10: end if
```

In the next section, we discuss $\boxed{\text{Line 8}}$ in Algorithm 2. The next example illustrates the problem and motivates the need for an efficient implementation of the QR factorization of S.

Example 4.12. (Continuation of Example 4.6.) From Proposition 3.6, the (upper triangular) Cholesky factor of $(R^*R) \circ (\overline{\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*})$ can be equivalently obtained from the QR factorization of $S = \mathbb{V}_{\ell,m}^T \odot R$. If we compute S explicitly and compute its QR factorization, normalized to have positive diagonal,[12] then

```
>> [Q,TQR] = qr(S,0) ; TQRn = diag(sign(diag(TQR)))*TQR
TQRn =
     1.000000000000000e+00     1.000000000000000e+00     1.000000002980232e+00
                         0     2.107342425544703e-08     1.414213575017149e-01
                         0                         0     1.471869344809795e-01
```

Note that the difference between `TQRn` and the theoretically identical matrix `TC`, computed in Example 4.6, is significant. Since $\kappa_2(S) \approx 1.6 \cdot 10^8$, we know that `TQR` is provably more accurate than `TC`.

The consequence of this difference in applications of the computed triangular factor is easily illustrated by the following simple computation (that is used for solving (3.9)). We set $\vec{\mathbf{g}} \equiv$ `g = 1./(1:ell*m)'`, `G = reshape(g,ell,m)`, and we set `b=(conj(Vlm).*(R'*G))*ones(m,1)`. After solving for $\boldsymbol{\alpha}$ with `TC` and `TQR`, the computed solutions and the corresponding relative (with respect to $\|\vec{\mathbf{g}}\|_2$) residuals show striking differences:

```
>> [alpha_TC alpha_TQR]=[TC\(TC'\b) TQR\(TQR'\b)]   >> [rez_TC ; rez_TQR]
ans =                                                ans =
-8.043848500219065e+08   -3.089216637627713e+07      9.404744232190256e+00
 8.043849072032555e+08    3.089216822956897e+07      4.137855517443713e-01
-5.618134924384464e+01   -8.532918510394528e-01
```

Although small dimensional and entirely synthetic, Examples 4.6 and 4.12 should be convincing enough to call for the development of an algorithm that solves the LS problem without the explicitly computed $(R^*R) \circ (\overline{\mathbb{V}_{\ell,m}\mathbb{V}_{\ell,m}^*})$. We tackle this problem in the next section. For the sake of brevity, we will consider the case $\mathbf{W} = \mathbb{I}_m$. The more involved general weighted case is left for future work.

5. QR factorization based solvers.

A numerically more robust procedure for solving the LS problem (3.11) needs the QR factorization of the $\ell m \times \ell$ matrix S, with the complexity of $O(m\ell^3)$ if an off-the-shelf procedure is deployed, e.g., `qr` from MATLAB or `xGEQRF`, `xGEQP3` from LAPACK. This $O(m\ell^3)$ factor will then dominate the overall cost of the LS solution, even if one decides to solve (3.11) using the SVD of S. (Since S is tall and skinny, the SVD of S is computed more efficiently if the computation starts with the QR factorization and proceeds with the SVD of the triangular factor.)

Squaring the condition number of S in the explicit formula (3.9) representing (3.12) is avoided if we use the tall QR factorization $S = Q_S R_S$; then[13]

$$\boldsymbol{\alpha} = R_S^{-1}(Q_S^* \vec{\mathbf{g}}). \tag{5.1}$$

In section 5.1, we first adopt the corrected seminormal approach and propose its structure exploiting implementation, assuming we have R_S. Then, in section 5.2 we present a novel recursive algorithm for computing the QR factorization $S = Q_S R_S$, with error analysis in section 5.3 and details of implementation in real arithmetic in section 5.4.

Note that for computing $Q_S^* \vec{\mathbf{g}}$ we do not need the explicitly formed factor Q_S. Indeed, since Q_S^* is built from a sequence of elementary unitary matrices (Householder

[12]This normalization is done only for easier comparison with the Cholesky factor computed in Example 4.6.

[13]In this section we use the notation from section 3.2.

reflectors or Givens rotations) so that $Q_S^* S = R_S$, it suffices to apply the same matrices to $\vec{\mathbf{g}}$ as if it were the $(\ell+1)$st column of S; a detailed algorithm is discussed in Remark 5.2. If, at a later stage in an application, the problem needs to be solved with a new value of $\vec{\mathbf{g}}$, then Q_S must be stored for later use.

5.1. Corrected seminormal approach. If we want to avoid the additional cost for computing $Q_S^* \vec{\mathbf{g}}$, in particular in view of the efficient formula (3.15) for $S^* \vec{\mathbf{g}}$, then we can settle for using the QR factorization of S only for implicit computing of the Cholesky factor of $(R^* R) \circ (\overline{\mathbb{V}_{\ell,m} \mathbb{V}_{\ell,m}^*})$ in (3.9), to avoid the problems illustrated in Examples 4.1 and 4.6. Then, the seminormal solution [6] is

$$\boldsymbol{\alpha} = R_S^{-1}(R_S^{-*}(S^* \vec{\mathbf{g}})). \tag{5.2}$$

This method only partially alleviates the problem of ill-conditioning. It computes a more accurate triangular factor than the pure normal equations approach (using the Cholesky factor of $S^* S$, which may even fail), but in general it is not guaranteed to be much better than the normal equations solver. Note, however, that in Example 4.12 the formula (5.2) gives an approximate solution `alpha_TQR` with better residual that the normal equation based solution `alpha_TC`.

However, if supplemented by a correction step, the seminormal solution becomes nearly as good as (sometimes even better than) the QR factorization based solver [6]. The correction procedure, done in the same working precision, first computes the residual $r = \vec{\mathbf{g}} - S\boldsymbol{\alpha}$, and then computes the correction $\delta\boldsymbol{\alpha}$ as

$$\delta\boldsymbol{\alpha} = R_S^{-1}(R_S^{-*}(S^* r)) \tag{5.3}$$

and the corrected solution $\boldsymbol{\alpha}_* = \boldsymbol{\alpha} + \delta\boldsymbol{\alpha}$. This process can be repeated. For an error analysis in favor of this scheme, see [6].

In an efficient software implementation, we use the structure of S and organize the data to increase locality; i.e., we prefer matrix multiplications. A prototype of the computational scheme is given in Algorithm 3. (In Lines 2, 3, 5, and 6 we give hints to an implementer, regarding high performance implementation based on the libraries BLAS and LAPACK.)

Algorithm 3. Corrected seminormal solution of (3.11).

```
Input: $R$, $\Lambda$, $G$, $S$ (Here the notation from section 3.2 applies.)
Output: Corrected solution $\boldsymbol{\alpha}_*$
1: Compute the triangular factor $R_S$ in the QR factorization of $S$.
2: $g_S = [(\overline{\mathbb{V}_{\ell,m}} \circ (R^* G))\mathbf{e}]$ {Note that $g_S = S^* \vec{\mathbf{g}}$. Use xTRMM from BLAS 3.}
3: $\boldsymbol{\alpha} = R_S^{-1}(R_S^{-*} g_S)$ {Use xTRSM or xTRTRS or xTRSV from LAPACK.}
4: $r_\square = G - R\begin{pmatrix}\boldsymbol{\alpha} & \Lambda\boldsymbol{\alpha} & \Lambda^2\boldsymbol{\alpha} & \ldots & \Lambda^{m-1}\boldsymbol{\alpha}\end{pmatrix} \equiv G - R\mathrm{diag}(\boldsymbol{\alpha})\mathbb{V}_{\ell,m}$
5: $r_S = [(\overline{\mathbb{V}_{\ell,m}} \circ (R^* r_\square))\mathbf{e}]$ {Note that $r_S = S^* r$. Use xTRMM from BLAS 3.}
6: $\delta\boldsymbol{\alpha} = R_S^{-1}(R_S^{-*} r_S)$ {Use xTRSM or xTRTRS or xTRSV from LAPACK.}
7: $\boldsymbol{\alpha}_* = \boldsymbol{\alpha} + \delta\boldsymbol{\alpha}$
```

Note that Algorithm 3 neither computes nor stores any information on the $\ell m \times \ell$ orthonormal Q_S, and that, except Line 1, all computation is in the framework of the original data matrices of dimensions $\ell \times \ell$ and $\ell \times m$, involving matrix operations already available in high performance libraries BLAS and LAPACK.

An illustration of the theoretically confirmed fact that the seminormal approach nearly matches the QR factorization based solution is given in the following example.

Example 5.1. (Continuation of Example 4.12.) Using the same data as in Example 4.12, we compute the approximate solutions `alpha_QR` using the formula (5.1) and `alpha_CSN` using Algorithm 3. The results are self-explanatory.

```
>> [alpha_TC                alpha_TQR                alpha_QR                alpha_CSN]
ans =
-8.04384850..e+08  -3.089216637627713e+07  -3.089216717302755e+07  -3.089216717302752e+07
 8.04384907..e+08   3.089216822956897e+07   3.089216902631945e+07   3.089216902631943e+07
-5.61813492..e+01  -8.532918510394528e-01  -8.532919080311419e-01  -8.532919080311413e-01
```

Hence, for the full benefit of the corrected seminormal equation approach, it is desirable to have an efficient QR factorization algorithm that can exploit the particular structure of S and thus lower the $O(m\ell^3)$ cost of a structure-oblivious straightforward factorization. With such a factorization, one can solve the LS problem (3.11) using (5.1) or the corrected seminormal approach as in Algorithm 3.

In section 5.2, we achieve that goal and show how the recursive structure of the block rows of S can be exploited to compute its QR factorization at the cost of $O(\ell^3 \log_2 m)$.

5.2. QR factorization of S.

We now provide the details of the new algorithm for computing the QR factorization of S. The recursive structure of the algorithm is first described in section 5.2.1 for the simplest case of $m = 2^p$, with the details of the kernel routine given in section 5.2.2. Section 5.2.3 provides the scheme with an arbitrary number of snapshots m.

5.2.1. The case $m = 2^p$.

The basic idea, using a binary elimination tree, is illustrated in (5.4) for $m = 16$, and it is, in a sense, analogous to the FFT divide and conquer scheme.

Algorithm 4. Recursive QR factorization of S in (3.11) for $m = 2^p$.

Input: Upper triangular $R \in \mathbb{C}^{\ell\times\ell}$; diagonal $\mathbf{\Lambda} \in \mathbb{C}^{\ell\times\ell}$; number of snapshots $m = 2^p$.
Output: The $2^p\ell \times \ell$ unitary $Q_S = \widehat{Q}_p$ and the upper triangular QR factor $R_S = T_p$ of $S \in \mathbb{C}^{2^p\ell\times\ell}$ in (3.11).

(5.4)

table of the elimination process for $m = 2^4$

$$\begin{array}{c|r|r|r||r}
\boxed{\boxed{T_4}} & \longleftarrow \boxed{T_3} & \longleftarrow \boxed{T_2} & \longleftarrow \boxed{T_1} & \longleftarrow R\mathbf{\Lambda}^0 \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \nwarrow R\mathbf{\Lambda}^1 \\ \hline
\mathbf{0} & \mathbf{0} & \mathbf{0} & \nwarrow T_1\mathbf{\Lambda}^2 & R\mathbf{\Lambda}^2 \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & R\mathbf{\Lambda}^3 \\ \hline\hline
\mathbf{0} & \mathbf{0} & \nwarrow T_2\mathbf{\Lambda}^4 & T_1\mathbf{\Lambda}^4 & R\mathbf{\Lambda}^4 \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & R\mathbf{\Lambda}^5 \\ \hline
\mathbf{0} & \mathbf{0} & \mathbf{0} & T_1\mathbf{\Lambda}^6 & R\mathbf{\Lambda}^6 \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & R\mathbf{\Lambda}^7 \\ \hline\hline\hline
\mathbf{0} & \nwarrow T_3\mathbf{\Lambda}^8 & T_2\mathbf{\Lambda}^8 & T_1\mathbf{\Lambda}^8 & R\mathbf{\Lambda}^8 \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & R\mathbf{\Lambda}^9 \\ \hline
\mathbf{0} & \mathbf{0} & \mathbf{0} & T_1\mathbf{\Lambda}^{10} & R\mathbf{\Lambda}^{10} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & R\mathbf{\Lambda}^{11} \\ \hline\hline
\mathbf{0} & \mathbf{0} & T_2\mathbf{\Lambda}^{12} & T_1\mathbf{\Lambda}^{12} & R\mathbf{\Lambda}^{12} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & R\mathbf{\Lambda}^{13} \\ \hline
\mathbf{0} & \mathbf{0} & \mathbf{0} & T_1\mathbf{\Lambda}^{14} & R\mathbf{\Lambda}^{14} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & R\mathbf{\Lambda}^{15}
\end{array}
\quad \longleftrightarrow$$

algorithm

```
1 :   $T_0 = R$ ;  $\widehat{Q}_0 = \mathbb{I}_\ell$
2 :   for  $i = 1 : p$  do
3 :   $[Q_i, \boxed{T_i}] = \mathtt{qr}(\begin{pmatrix} \boxed{T_{i-1}} \\ T_{i-1}\mathbf{\Lambda}^{2^{i-1}} \end{pmatrix})$
%     economy size QRF, $Q_i \in \mathbb{C}^{2\ell\times\ell}$
4 :   $\boxed{\widehat{Q}_i} = \begin{pmatrix} \boxed{\widehat{Q}_{i-1}} & \mathbf{0} \\ \mathbf{0} & \boxed{\widehat{Q}_{i-1}} \end{pmatrix} Q_i$
5 :   end for
```

Note that the core operation in an ith step of the above scheme is the QR factorization of the $2\ell \times \ell$ structured matrix (in (5.4), Line 3 of the algorithm, and marked by a pair of arrows $\overset{\leftarrow}{\nwarrow}$ in the table)

$$\begin{pmatrix} T_{i-1} \\ T_{i-1}\mathbf{\Lambda}^{2^{i-1}} \end{pmatrix} = \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \\ \star & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & 0 & \star \end{pmatrix}, \text{ i.e., implicitly of } \begin{pmatrix} T_{i-1} \\ \mathbf{0}_{(2^{i-1}-1)\ell,\ell} \\ T_{i-1}\mathbf{\Lambda}^{2^{i-1}} \\ \mathbf{0}_{(2^{i-1}-1)\ell,\ell} \end{pmatrix}, \tag{5.5}$$

and, thus, it can be computed more efficiently than in the general case of a $2\ell \times \ell$ dense matrix. The QR factorizations of the remaining pairs in an ith column are obtained simply by scaling the triangular factor of this prototype QR factorization of (5.5) by a corresponding power of $\mathbf{\Lambda}$.

Remark 5.2. If we want to solve the LS problem using (5.1), then in Algorithm 4, instead of accumulating Q_S, we include applying Q_i^* to the corresponding parts of $\vec{\mathbf{g}}$. The algorithm then reads

```
$T_0 = R$
**for** $i = 1 : p$ **do**
  $[Q_i, \boxed{T_i}] = \mathtt{qr}\left(\begin{pmatrix} \boxed{T_{i-1}} \\ T_{i-1}\mathbf{\Lambda}^{2^{i-1}} \end{pmatrix}\right)$ {Thin QR factorization. $Q_i$ is $2\ell \times \ell$.}
  **for** $j = 0 : 2^i : 2^p - 2^i$ **do**
    $\vec{\mathbf{g}}(j\ell + 1 : (j+1)\ell) = Q_i^* \begin{pmatrix} \vec{\mathbf{g}}(j\ell + 1 : (j+1)\ell) \\ \vec{\mathbf{g}}((j + 2^{i-1})\ell + 1 : (j + 2^{i-1} + 1)\ell) \end{pmatrix}$
  **end for**
**end for**
```

Note that after this sequence of updates, the array $\vec{\mathbf{g}}(1 : \ell)$ contains the vector $Q_S^*\vec{\mathbf{g}}$ from (5.1); the LS solution is then computed as the solution of the triangular system $T_p\boldsymbol{\alpha} = \vec{\mathbf{g}}(1 : \ell)$.

5.2.2. Details of the QR factorization of (5.5).

The seemingly simple task to compute the QR factorization of two stacked triangular matrices (5.5) is an excellent case study for turning an algorithm into an efficient software implementation. For that reason, we give detailed descriptions of the main phases of the development. However, we omit the details of assembling Q_S, which follow easily from Line 4 in (5.4).

Algorithm 5 illustrates a straightforward scheme to annihilate the $\ell(\ell+1)/2$ entries in the lower block in (5.5); it works for the general case of two independent upper triangular matrices stacked on top of each other, and its cost is $\ell^3 + O(\ell^2)$ *flops*. The nested loops in Algorithm 5 access the matrix B columnwise. Changing the loops into

$$\textbf{for } \ i = 1 : \ell \ \ \{\textbf{for } \ j = i : \ell \ \ \{4 : \ \ldots ; 5 : \ \ldots \textbf{end for}\} \textbf{ end for } \}$$

will change the access to row-wise. The proper choice of loop ordering depends on the layout of the data matrices in the computer memory.

To enhance data locality, the annihilation strategy in Algorithm 5 can be modified so that the communication between the arrays A and B is reduced to once per column; it suffices to concentrate the mass of a column of B into a single entry (e.g., using a single Householder reflector) and then move it up into the corresponding diagonal entry of A using a single Givens rotation. This yields Algorithm 6.

Algorithm 5. Givens QR factorization of the type (5.5).

Input: Upper triangular matrices $A, B \in \mathbb{C}^{\ell\times\ell}$

Output: The upper triangular factor in the QR factorization of $\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} \ast & \ast & \ast & \ast \\ 0 & \ast & \ast & \ast \\ 0 & 0 & \ast & \ast \\ 0 & 0 & 0 & \ast \\ \star & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & 0 & \star \end{pmatrix}$

```
1: $Q = \mathbb{I}_{2\ell}$
2: for $j = 1 : \ell$ do
3:   for $i = 1 : j$ do
4:     Compute $2 \times 2$ Givens rotation $\mathcal{G}$ such that $\mathcal{G}\begin{pmatrix} A(j,j) \\ B(i,j) \end{pmatrix} = \begin{pmatrix} \sqrt{|A(j,j)|^2 + |B(i,j)|^2} \\ 0 \end{pmatrix}$
5:     $\begin{pmatrix} A(j,j:\ell) \\ B(i,j:\ell) \end{pmatrix} := \mathcal{G}\begin{pmatrix} A(j,j:\ell) \\ B(i,j:\ell) \end{pmatrix}$; $(Q(:,i), Q(:,j)) := (Q(:,i), Q(:,j))\mathcal{G}^*$
6:   end for
7: end for
```

Algorithm 6. Householder+Givens QR factorization of the type (5.5).

Input: Upper triangular matrices $A, B \in \mathbb{C}^{\ell\times\ell}$

Output: The upper triangular factor in the QR factorization of $\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} \ast & \ast & \ast & \ast \\ 0 & \ast & \ast & \ast \\ 0 & 0 & \ast & \ast \\ 0 & 0 & 0 & \ast \\ \star & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & 0 & \star \end{pmatrix}$

```
1: for $j = 1 : \ell$ do
2:   if $j > 1$ then
3:     Compute Householder reflector $\mathcal{H} = \mathbb{I} - \beta w w^*$ such that $\mathcal{H}B(1:j,j) = \pm\|B(1:j,j)\|_2 e_1$
4:     $B(1,j) = \pm\|B(1:j,j)\|_2$
5:     if $j < \ell$ then
6:       Update $B$: $B(1:j,j+1:\ell) = B(1:j,j+1:\ell) - \beta w(w^* B(1:j,j+1:\ell))$
7:     end if
8:   end if
9:   Compute $2 \times 2$ Givens rotation $\mathcal{G}$ such that $\mathcal{G}\begin{pmatrix} A(j,j) \\ B(1,j) \end{pmatrix} = \begin{pmatrix} \sqrt{|A(j,j)|^2 + |B(1,j)|^2} \\ 0 \end{pmatrix}$
10:  $\begin{pmatrix} A(j,j:\ell) \\ B(1,j:\ell) \end{pmatrix} := \mathcal{G}\begin{pmatrix} A(j,j:\ell) \\ B(1,j:\ell) \end{pmatrix}$;
11: end for
```

The operations in Algorithm 6 are illustrated in the scheme (5.6).

$$\left(\begin{array}{cccc} \ast & \ast & \ast & \ast \\ 0 & \ast & \ast & \ast \\ 0 & 0 & \ast & \ast \\ 0 & 0 & 0 & \ast \\ \hline\hline \star & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & 0 & \star \end{array}\right) \xrightarrow{1,1} \left(\begin{array}{cccc} \bullet & \ast & \ast & \ast \\ 0 & \ast & \ast & \ast \\ 0 & 0 & \ast & \ast \\ 0 & 0 & 0 & \ast \\ \hline\hline \mathbf{0} & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & 0 & \star \end{array}\right) \overset{1:2,2}{\Longrightarrow} \left(\begin{array}{cccc} \bullet & \ast & \ast & \ast \\ 0 & \ast & \ast & \ast \\ 0 & 0 & \ast & \ast \\ 0 & 0 & 0 & \ast \\ \hline\hline 0 & \divideontimes & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & 0 & \star \end{array}\right) \xrightarrow{1,2} \left(\begin{array}{cccc} \bullet & \ast & \ast & \ast \\ 0 & \bullet & \ast & \ast \\ 0 & 0 & \ast & \ast \\ 0 & 0 & 0 & \ast \\ \hline\hline 0 & 0 & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & 0 & \star \end{array}\right) \overset{1:3,3}{\Longrightarrow} \left(\begin{array}{cccc} \bullet & \ast & \ast & \ast \\ 0 & \bullet & \ast & \ast \\ 0 & 0 & \ast & \ast \\ 0 & 0 & 0 & \ast \\ \hline\hline 0 & 0 & \divideontimes & \star \\ 0 & 0 & 0 & \star \\ 0 & 0 & 0 & \star \\ 0 & 0 & 0 & \star \end{array}\right) \xrightarrow{1,3} \left(\begin{array}{cccc} \bullet & \ast & \ast & \ast \\ 0 & \bullet & \ast & \ast \\ 0 & 0 & \bullet & \ast \\ 0 & 0 & 0 & \ast \\ \hline\hline 0 & 0 & 0 & \star \\ 0 & 0 & 0 & \star \\ 0 & 0 & 0 & \star \\ 0 & 0 & 0 & \star \end{array}\right) \overset{1:4,4}{\Longrightarrow}$$

$$\begin{pmatrix} \bullet & * & * & * \\ 0 & \bullet & * & * \\ 0 & 0 & \bullet & * \\ 0 & 0 & 0 & * \\ \hline 0 & 0 & 0 & \divideontimes \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{1,4} \begin{pmatrix} \bullet & * & * & * \\ 0 & \bullet & * & * \\ 0 & 0 & \bullet & * \\ 0 & 0 & 0 & \bullet \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \quad \boxed{\text{Legend : } \begin{array}{l} \xrightarrow{1,j} \text{Givens rotation; Lines 9–10.} \\ \xRightarrow{1:j,j} \text{Householder reflector; Lines 3–8.} \end{array}} \tag{5.6}$$

The above scheme is now a starting point for a block-oriented algorithm that delivers true high performance computation. Suppose our matrices are block partitioned with block size b so that the total number of blocks is $\wp = \lceil \ell/b \rceil$; the leading $\wp - 1$ diagonal blocks are $b \times b$, and the size of the last block is $(\ell - (\wp - 1)b) \times (\ell - (\wp - 1)b)$. Then we can simply imagine that, e.g., in (5.6) each $*, \star, \bullet, \divideontimes, 0$ represents a $b \times b$ matrix (instead of being a scalar); the blocks in the last row and column may have one or both dimensions $(\ell - (\wp - 1)b)$. For such a block partition, we use the notation $A[i, j]$, $B[i, j]$ to denote the submatrices (blocks) at the position (i, j), and $A[i_1 : i_2, j_1 : j_2]$ is defined analogously to the scalar case.

Remark 5.3. In high performance libraries such as LAPACK, Householder reflectors are aggregated so that several of them can be applied more efficiently, with a better *flop*-to-memory-reference ratio. If in Line 4 of Algorithm 7 the matrix $B[1 : j, j]$ has b columns, then the QR factorization is achieved by a sequence of left multiplications $\mathcal{H}_b \cdots \mathcal{H}_2 \mathcal{H}_1 B[1 : j, j]$, where $\mathcal{H}_i = \mathbb{I} - \beta_i w_i w_i^*$ with $w_i(1 : i - 1) = \mathbf{0}$, $w_i(i) = 1$. The compact form $\mathcal{H} = \mathbb{I} - \mathcal{W}_b \mathcal{T}_b \mathcal{W}_b^*$ of $\mathcal{H} = \mathcal{H}_1 \mathcal{H}_2 \cdots \mathcal{H}_b$ is then computed recursively as

$\mathcal{W}_1 = (w_1)$, $\mathcal{T}_1 = \beta_1$; $\mathcal{W}_j = (\mathcal{W}_{j-1} \; w_j)$,

$$\mathcal{T}_j = \begin{pmatrix} \mathcal{T}_{j-1} & -\beta_j \mathcal{T}_{j-1} \mathcal{W}_{j-1}^* w_j \\ \mathbf{0} & \beta_j \end{pmatrix}, \quad j = 2, \ldots, b.$$

For more details we refer the reader to [40], and for a guidelines for an efficient implementation we suggest studying the structure of the subroutine `xGEQRF` in LAPACK; essentially, the computation in Lines 4 and 6 of Algorithm 7 is already contained as a part of `xGEQRF`.

5.2.3. The case of general m. We now generalize the recursive scheme of Algorithm 4 to general dimension $m \neq 2^p$. First, introduce simple notation: S will be considered as a block row partitioned with the ith block $S_{[i]} = R\Lambda^{i-1}$. The submatrix of S consisting of consecutive blocks from the i_1th to the i_2th will be denoted by $S_{[i_1:i_2]}$. Note that $S_{[i_1:i_2]} = S_{[1:i_2-i_1+1]}\Lambda^{i_1-1}$.

As a motivation, note that the QR factorization of $S_{[1:16]}$ in the scheme (5.4) contains among its intermediate results the QR factorization also, e.g., of$S_{[1:2]}$ (the factor is T_1), of $S_{[1:4]}$ (the factor is T_2), of $S_{[1:8]}$ (the factor is T_3), and of $S_{[9:12]}$ (the factor is $T_2\Lambda^8$ as $S_{[9:12]} = S_{[1:4]}\Lambda^8$). Also, T_i is the triangular factor of the leading 2^i block rows of S.

To exploit this, write m as a binary number $m \equiv \mathfrak{b} = (\mathfrak{b}_{\lfloor \log_2 m \rfloor}, \ldots, \mathfrak{b}_1, \mathfrak{b}_0)_2$, i.e.,

$$m = \sum_{i=0}^{\lfloor \log_2 m \rfloor} \mathfrak{b}_i 2^i \equiv \sum_{j=1}^{j^*} 2^{i_j}, \quad \lfloor \log_2 m \rfloor = i_{j^*} > i_{j^*-1} > \cdots > i_2 > i_1 \geq 0, \tag{5.7}$$

and introduce the block partition of S as follows:

$$S^T = \begin{pmatrix} S^T_{[1:2^{i_{j^*}}]} & S^T_{[2^{i_{j^*}}+1:2^{i_{j^*}}+2^{i_{j^*-1}}]} & \cdots & S^T_{[2^{i_{j^*}}+\cdots+2^{i_2}+1:2^{i_{j^*}}+\cdots+2^{i_1}]} \end{pmatrix}. \tag{5.8}$$

Algorithm 7. Block-oriented Householder+Givens QR factorization of the type (5.5).

```
Input: Upper triangular matrices $A, B \in \mathbb{C}^{\ell\times\ell}$ and the block parameter $b$.
Output: The upper triangular factor in the QR factorization of $\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \\ \star & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & 0 & \star \end{pmatrix}$
1: $\wp = \lceil \ell/b \rceil$; $b' = \ell - (\wp - 1)b$. Introduce block partitions in $A$ and $B$.
2: for $j = 1 : \wp$ do
3:   if $j > 1$ then
4:     Compute the QR factorization $B[1:j,j] = \mathcal{H}\binom{\mathcal{R}}{0}$ of $B[1:j,j]$ as follows:
         4.1: The upper triangular factor $\mathcal{R}$ overwrites the leading submatrix of $B[1,j]$.
         4.2: Write the accumulated product of Householder reflectors $\mathcal{H}$ in the compact form $\mathcal{H} = \mathbb{I} - \mathcal{W}\mathcal{T}\mathcal{W}^*$. (See Remark 5.3.)
5:     if $j < \wp$ then
6:       Update $B$: $B[1:j, j+1:\wp] = B[1:j, j+1:\wp] - \mathcal{W}\mathcal{T}^*(\mathcal{W}^* B[1:j, j+1:\wp])$
7:     end if
8:   end if
9:   Compute the QR factorization $\begin{pmatrix} A[j,j] \\ B[1,j] \end{pmatrix} = \mathcal{Q}\begin{pmatrix} \widehat{\mathcal{R}} \\ \mathbf{0} \end{pmatrix}$ so that the upper triangular factor $\widehat{\mathcal{R}}$ overwrites $A[j,j]$. Use, e.g., Algorithm 6.
10:  if $j < \wp$ then
11:    Update $A$ and $B$: $\begin{pmatrix} A[j,j+1:\wp] \\ B[1,j+1:\wp] \end{pmatrix} := \mathcal{Q}^* \begin{pmatrix} A[j,j+1:\wp] \\ B[1,j+1:\wp] \end{pmatrix}$;
12:  end if
13: end for
```

The QR factorization of the largest block $S_{[1:2^{i_{j^*}}]}$ can be computed by the $O(\lfloor \log_2 m \rfloor \ell^3)$ Algorithm 4, and the triangular factor of each of the subsequent blocks in (5.8) is, up to column scaling by an appropriate power of Λ, available among the intermediate results, as discussed above. These are designated as *local triangular factors*. This consists of the first reduction step, which results in at most $\lfloor \log_2 m \rfloor + 1$ local $\ell \times \ell$ upper triangular factors. In the second step, these are reduced, by building *global triangular factors* to a single triangular matrix at the cost of at most $O(\lfloor \log_2 m \rfloor \ell^3)$.

For an implementation of this procedure, it will be convenient to process the powers 2^{i_j} in an increasing order, i.e., to scan the binary representation $\mathfrak{b}$ from the right to the left. The local triangular factors of the blocks

(5.9)
$$S_{[1:2^{i_1}]}, S_{[2^{i_1}+1:2^{i_1}+2^{i_2}]}, S_{[2^{i_1}+2^{i_2}+1:2^{i_1}+2^{i_2}+2^{i_3}]}, \ldots, S_{[2^{i_1}+2^{i_2}+\cdots+2^{i_{j^*-1}}+1:2^{i_1}+2^{i_2}+2^{i_3}+\cdots+2^{i_{j^*}}]}$$

are computed by scaling the computed triangular factors of, respectively,

$$S_{[1:2^{i_1}]}, S_{[1:2^{i_2}]}, S_{[1:2^{i_3}]}, \ldots, S_{[1:2^{i_{j^*}}]} \tag{5.10}$$

with, respectively, $\Lambda^0, \Lambda^{2^{i_1}}, \Lambda^{2^{i_1}+2^{i_2}}, \ldots, \Lambda^{2^{i_1}+2^{i_2}+\cdots+2^{i_{j^*-1}}}$, and are built into the global triangular factor by a sequence of updates. The procedure is summarized in Algorithm 8. For the sake of simplicity, we did not include computation of $Q_S^* \vec{\mathbf{g}}$, which can be easily added following the procedure outlined in Remark 5.2. The details of assembling Q_S (explicitly, or implicitly by storing Householder vectors as in `xGEQRF`, xGEQP3, xORGQR, xORMQR in LAPACK) are omitted because they are technical details in software development which are beyond the scope of this paper and are a subject of our separate work.

Remark 5.4. The scheme of Algorithm 8 can be easily adapted to work, e.g., in base 3, i.e., $m = 3^p$ and, in general, using the representation of m in base 3. Such details/variations become important for a custom-made implementation on a particular hardware.

Algorithm 8. Recursive QR factorization of S in (3.11).

```
Input: Upper triangular $R \in \mathbb{C}^{\ell\times\ell}$; diagonal $\mathbf{\Lambda} \in \mathbb{C}^{\ell\times\ell}$; number of snapshots $m$
Output: Upper triangular QR factor $R_S = \mathbb{T}_{j-1}$ of $S$ in (3.11)
1: Compute the binary representation (5.7) of $m$: $m \equiv \mathfrak{b} = (\mathfrak{b}_{\lfloor\log_2 m\rfloor}, \ldots, \mathfrak{b}_1, \mathfrak{b}_0)_2$
2: Let $\lfloor\log_2 m\rfloor = i_{j^*} > i_{j^*-1} > \cdots > i_2 > i_1 \geq 0$ be as in (5.7)
3: $T_0 = R$
4: if $i_1 = 0$ then
5:    $\mathbb{T}_1 = T_0$; $j = 2$; $\wp = 1$
6: else
7:    $\mathbb{T}_0 = []$; $j = 1$; $\wp = 0$
8: end if
9: for $k = 1 : i_{j^*}$ do
10:    $\begin{pmatrix} \boxed{T_k} \\ \mathbf{0} \end{pmatrix} = \mathtt{qr}\left(\begin{pmatrix} \boxed{T_{k-1}} \\ T_{k-1}\mathbf{\Lambda}^{2^{k-1}} \end{pmatrix}\right)$ {Local triangular factor. Use algorithms from section 5.2.2.}
11:    if $k = i_j$ then
12:       if $\mathbb{T}_{j-1} \neq []$ then
13:          $\begin{pmatrix} \boxed{\mathbb{T}_j} \\ \mathbf{0} \end{pmatrix} = \mathtt{qr}\left(\begin{pmatrix} \boxed{\mathbb{T}_{j-1}} \\ T_k\mathbf{\Lambda}^{\wp} \end{pmatrix}\right)$ {Global triangular factor. Use algorithms from section 5.2.2.}
14:       else
15:          $\boxed{\mathbb{T}_j} = \boxed{T_k}$
16:       end if
17:       $j := j + 1$; $\wp := \wp + 2^k$
18:    end if
19: end for
```

5.3. Numerical stability. For an efficient implementation, the computational scheme, e.g., in Algorithm 5 will be modified to enhance spatial and temporal locality of data, e.g., using tiling or blocking techniques as in Algorithm 7, or parallelized for multicore hardware. It can be shown that with any such modification, the above computation is backward stable in the sense that the computed triangular factor is an exact factor of $S + \delta S$, where the backward error δS is columnwise small, i.e.,

$$\|\delta S(:,j)\|_2 \leq \eta \|S(:,j)\|_2, \quad j = 1, \ldots, \ell; \quad \eta \leq f(\ell, m)\boldsymbol{\varepsilon}, \tag{5.11}$$

where $f(\ell, m)$ is a modest polynomial that depends on the details of a particular implementation.[14]

[14]Note that (5.11) is a much stronger statement than the usually used backward error bound $\|\delta S\|_F \leq g(\ell, m)\boldsymbol{\varepsilon}\|S\|_F$.

This follows from the simple fact that in Algorithms 4 and 8 (using Algorithm 5 with any ordering of Givens rotations, or Algorithms 6 and 7 as a kernel computational routine) we actually multiply the initial S from the left by a sequence of elementary unitary matrices; this is nothing else but independent unitary transformations (backward stable) of the columns of S. Hence, each column of S has backward error that is small relative to that same column; in this way even the tiniest columns are preserved, independently of the remaining possibly much larger ones.

Due to (5.11), the condition number that determines the accuracy of the decomposition is $\kappa_2(S_c)$, where S_c is obtained from S by scaling its columns[15] so that $\|S_c(:,j)\|_2 = 1$ for all j. More precisely, if $\widetilde{R}_S = R_S + \delta R_S$ is the computed factor, then $\delta R_S = \Gamma R_S$ (i.e., $\delta R_S(:,j) = \Gamma(:,1:j)R_S(:,j)$, for all j) and, following [16, section 6.],

$$\|\Gamma\|_F \leq \frac{\sqrt{8}\ell\eta}{1-\eta}\|S_c^\dagger\|_2 + O((\eta\|S_c^\dagger\|_2)^2) \leq \sqrt{8}\ell\eta\kappa_2(S_c) + O((\eta\|S_c^\dagger\|_2)^2).$$

In terms of the initial data, the condition number of column-equilibrated S is estimated as follows.

Corollary 5.5. *With the notation of Corollary* 4.7*, it holds that*

$$\begin{aligned}\kappa_2(S_c) = \sqrt{\kappa_2(C_s)} &\leq \min(\kappa_2(R_c), \kappa_2((\mathbb{V}_{\ell,m})_r)) \\ &\leq \sqrt{\ell}\min(\min_{D=\mathrm{diag}}\kappa_2(RD), \min_{D=\mathrm{diag}}\kappa_2(D\mathbb{V}_{\ell,m})).\end{aligned} \tag{5.12}$$

Proof. It follows from Proposition 3.6 that $C_s = S_c^* S_c$; hence $\kappa_2(S_c) = \sqrt{\kappa_2(C_s)}$, and the first inequality follows from Corollary 4.7. The second inequality follows from the classical result [50]. ∎

This can be used to estimate the accuracy of Line 8 in Algorithm 2: if $\min(\kappa_2(R_c), \kappa_2((\mathbb{V}_{\ell,m})_r))$ in (5.12) is below an appropriate threshold, a certain level of accuracy can be guaranteed a priori.

5.3.1. Importance of pivoting.

For better accuracy of the solutions of triangular equations (forward and backward substitutions in (5.2)), it would be advantageous to have column pivoted (rank revealing) QR factorization of S, i.e., that R_S has strong diagonal dominance. Further, if S is ill-conditioned, then the rank revealing QR factorization can be used to determine the numerical rank of S and, by truncating R_S, to compute an approximate LS solution with certain level of sparsity. For the reader's convenience, we briefly review this procedure.

Let $\mathrm{rank}(S) = r_S < \ell$, so that in the column pivoted QR factorization

$$SP = Q_S R_S = \begin{pmatrix} R_{[11]} & R_{[12]} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} = Q_{S,r}\begin{pmatrix} R_{[11]} & R_{[12]} \end{pmatrix}, \quad Q_{S,r} = Q_S(:,1:r). \tag{5.13}$$

Then the LS problem $\|S\boldsymbol{\alpha} - \vec{\mathbf{g}}\|_2 \to \min$ can be written as

$$\begin{aligned}\|S\boldsymbol{\alpha} - \vec{\mathbf{g}}\|_2^2 &= \|Q_{S,r}\begin{pmatrix} R_{[11]} & R_{[12]} \end{pmatrix}P^T\boldsymbol{\alpha} - Q_{S,r}Q_{S,r}^*\vec{\mathbf{g}} - (\mathbb{I} - Q_{S,r}Q_{S,r}^*)\vec{\mathbf{g}}\|_2^2 \\ &= \|\begin{pmatrix} R_{[11]} & R_{[12]} \end{pmatrix}P^T\boldsymbol{\alpha} - Q_{S,r}^*\vec{\mathbf{g}}\|_2^2 + \|(\mathbb{I} - Q_{S,r}Q_{S,r}^*)\vec{\mathbf{g}}\|_2^2 \to \min,\end{aligned}$$

and the problem reduces to

$$\|\begin{pmatrix} R_{[11]} & R_{[12]} \end{pmatrix}P^T\boldsymbol{\alpha} - Q_{S,r}^*\vec{\mathbf{g}}\|_2 \longrightarrow \min_{\boldsymbol{\alpha}}. \tag{5.14}$$

[15] This scaling is only a theoretical device; the algorithm does not perform it.

One particular vector in the solution manifold is

$$\boldsymbol{\alpha}_\diamond = P \begin{pmatrix} R_{[11]}^{-1} Q_{S,r}^* \vec{\mathbf{g}} \\ \mathbf{0} \end{pmatrix}. \tag{5.15}$$

Note that $\boldsymbol{\alpha}_\diamond$ has at least $\ell - r_S$ zero entries and that it is different from the shortest solution $\boldsymbol{\alpha}_* = S^\dagger \vec{\mathbf{g}}$. Hence, if the additional criterion is sparsity, the rank deficient LS problem is best solved by (5.13), (5.15). In the full rank case $\boldsymbol{\alpha}_\diamond = \boldsymbol{\alpha}_*$.

Remark 5.6. Note that (5.14) is underdetermined and that we can add a sparsity constraint analogously to (2.5), which is accordance with Remark 2.1. The sparsity of the explicit solution (5.15) is a good starting point for a quest for sparse solution.

Remark 5.7. The MATLAB backslash operator (e.g., `alpha=S\g`) solves LS problems using (5.13), (5.15); the pivoted QR factorization ($[Q_S, R_S, P] = \mathtt{qr}(S)$) is computed by the LAPACK subroutine `xGEQP3`. If sparsity is a desirable property of the solution in the rank deficient case, then (5.13), (5.15) should be preferred over using the Moore–Penrose pseudoinverse based minimum norm solution (e.g., `alpha=pinv(S)*g` or `alpha=lsqminnorm(S,g)` in MATLAB, or calling the `xGELSX` and `xGELSY` LS solvers in LAPACK).

The numerical rank $\widetilde{r}_S$ of S is detected as follows: The column pivoted QR factorization computes $SP = Q_S R_S$ and finds the smallest index $\widetilde{r}_S$ such that $|(R_S)_{\widetilde{r}_S+1,\widetilde{r}_S+1}| \leq \xi |(R_S)_{\widetilde{r}_S,\widetilde{r}_S}|$ where the threshold ξ is usually $O(\ell\boldsymbol{\varepsilon})$. Then, in the block partition

$$SP = Q_S R_S = \begin{pmatrix} R_{[11]} & R_{[12]} \\ \mathbf{0} & R_{[22]} \end{pmatrix}, \quad R_{[11]} \in \mathbb{C}^{\widetilde{r}_S \times \widetilde{r}_S} \tag{5.16}$$

it holds that $\|R_{[22]}\|_F \leq \sqrt{\ell - \widetilde{r}_S}\,\xi |R_{\widetilde{r}_S,\widetilde{r}_S}|$, and $R_{[22]}$ is set to zero, thus yielding a rank revealing decomposition of the form (5.13). The truncation of $R_{[22]}$ is justified by a backward perturbation of S. The choice of the threshold ξ can be determined by taking into account the noise level on input, or it can be used to aggressively enforce low numerical rank (by allowing larger backward error) to obtain a faster solver or a sparser LS solution. Indubitably, a QR factorization LS algorithm should use pivoting, and it remains to see how to mount the pivoting device in Algorithms 4 and 8. The simplest way is to take the final upper triangular matrix R_S on exit and recompute its QR factorization with the Businger–Golub [9] column pivoting; the more elegant way is to build the pivoting in the algorithm. It can be easily seen that the pivoting can be turned on at any (or every) stage in both algorithms; the permutations are accumulated/composed and pushed backward in the original matrix S. If the overhead due to pivoting at all stages is not acceptable, then we can settle for only the last step when computing the last upper triangular matrix.

Remark 5.8. Both the normal equations and the QR factorization based approaches (including the corrected seminormal equations) can be enhanced with Tikhonov regularization either using the QR (using the inverse of $R_S^* R_S + \mu\mathbb{I}$, where μ is a regularization parameter) or using the SVD of S. As we mentioned at the beginning of section 5, the QR factorization of S also allows efficient SVD of S (via the SVD of its triangular factor): If $S = Q_S R_S$ and if the SVD of R_S is $R_S = U_R \Sigma_S V_R^*$, then the regularized solution is $\boldsymbol{\alpha}(\mu) = V_R \mathrm{diag}(\sigma_i/(\sigma_i^2 + \mu^2))_{i=1}^{\ell} U_R^* (Q_S^* \vec{\mathbf{g}})$, where σ_i is the ith largest singular value of S and $Q_S^* \vec{\mathbf{g}}$ is computed during the QR factorization of S, as explained in Remark 5.2. The regularization parameter is tuned based on the information on the noise. For the DMD analysis in the presence of noise see [12], and for the corresponding numerical linear algebra framework see [15, sections 5.4 and 5.5].

5.4. On real data and closedness under complex conjugacy. If the data $\mathbf{x}_i$ in (1.2) and the operator $\mathbb{A}$ are real, it is desirable that the reconstruction ansatz is a priori structurally real, even if the selected Ritz pairs (z_j, λ_j) are in general complex. Further, in the real case, it is desirable to do the entire computation in real arithmetic, which would substantially improve the performances of the software.

Keeping real arithmetic for real input (e.g., real matrices) and complex output that is a priori known to have complex conjugacy symmetry (eigenvalues and eigenvectors of real matrices) has important benefits with respect to numerical robustness (structure preserving that is important from the point of view of the perturbation theory) and computational efficiency (real data structures and real arithmetic). These are exploited in the state-of-the-art software for matrix computations. So, for instance, in the LAPACK library, the driver routines for computing eigenvalues and eigenvectors `xGEEV`, `xGEEVX` ($\mathtt{x} \in \{\mathtt{S},\mathtt{D}\}$ for single and double precision real matrices) use only real arithmetic, and complex eigenvalues and eigenvectors are returned as ordered pairs of their real and imaginary parts.

Hence, in a high performance LAPACK-based implementation of a minimizer for (3.4), the computation with (λ_j, z_j) should be in terms of $(\Re(\lambda_j), \Im(\lambda_j); \Re(z_j), \Im(z_j))$. Here $\Re(\cdot)$ and $\Im(\cdot)$ denote, respectively, the real and the imaginary part of a scalar, vector, or matrix.

5.4.1. Real reconstruction scheme of real data. The following technical proposition provides all the details needed for obtaining in real arithmetic an LS solution closed under complex conjugation, and, consequently, real approximants to the snapshots $\mathbf{x}_i$.

Proposition 5.9. *Let all $\mathbf{x}_i$'s be real, i.e., $\mathbf{x}_i \in \mathbb{R}^n$, $\mathbb{A} \in \mathbb{R}^{n\times n}$. If the wizard has selected the Ritz pairs so that with each complex pair (z_j, λ_j) the sum on the right-hand side of (1.2) contains also the contribution of its conjugate $(\overline{z_j}, \overline{\lambda_j})$, then the corresponding coefficients are α_j and $\overline{\alpha_j}$, respectively. If (z_j, λ_j) is real, then α_j is real as well. As a result, the computed approximation is real.*

Proof. Assume that the eigenvalues are ordered so that $\lambda_1, \ldots, \lambda_{\ell_1}$ are purely real, and the remaining complex eigenvalues are listed in groups of complex conjugate pairs $\lambda_j, \lambda_{j+1} = \overline{\lambda_j}$, with $\mathrm{Im}(\lambda_j) > 0$. Let ℓ_2 be the number of complex conjugate pairs. Hence $\ell = \ell_1 + 2\cdot\ell_2$, $\ell_1, \ell_2 \geq 0$.

Consider now a complex conjugate pair (λ_j, z_j), $(\lambda_{j+1}, z_{j+1}) \equiv (\overline{\lambda_j}, \overline{z_j})$. Since z_j and $\overline{z_j}$ are linearly independent, the contribution of the directions of z_j and z_{j+1} can be replaced by the span of the two purely real vectors $\Re(z_j)$ and $\Im(z_j)$. Let

$$\Phi = \frac{1}{2}\begin{pmatrix} 1 & -\mathfrak{i} \\ 1 & \mathfrak{i} \end{pmatrix}, \text{ with } \Phi^{-1} = \begin{pmatrix} 1 & 1 \\ \mathfrak{i} & -\mathfrak{i} \end{pmatrix}.$$

Then $\begin{pmatrix}\Re(z_j) & \Im(z_j)\end{pmatrix} = \begin{pmatrix} z_j & z_{j+1}\end{pmatrix}\Phi$. Note that $\sqrt{2}\Phi$ is unitary and that for $j = \ell_1+1, \ell_1+3, \ldots, \ell-1$,

$$\Phi^{-1}\begin{pmatrix}\lambda_j & 0 \\ 0 & \overline{\lambda_j}\end{pmatrix}\Phi = \begin{pmatrix}\Re(\lambda_j) & \Im(\lambda_j) \\ -\Im(\lambda_j) & \Re(\lambda_j)\end{pmatrix} \equiv \hat{\Lambda}_j. \text{ (Here } \Im(\lambda_j) > 0.)$$

Define block–diagonal matrix Φ_λ with unit diagonal 1×1 blocks (1) for each real Ritz value λ_j and 2×2 matrix Φ for each complex conjugate pair λ_j, λ_{j+1}. Note that $Z_\ell \Phi_\lambda$

is now a real matrix,

$$Z_\ell \Phi_\lambda = \left(z_1, \quad \ldots, \quad z_{\ell_1}, \quad \Re(z_{\ell_1+1}), \quad \Im(z_{\ell_1+1}), \quad \ldots, \quad \Re(z_{\ell-1}), \quad \Im(z_{\ell-1})\right), \tag{5.17}$$

and the real versions of the powers of $\mathbf{\Lambda}$ contain 2×2 blocks for each pair $\lambda_j, \lambda_{j+1} = \overline{\lambda_j}$,

$$\Phi_\lambda^{-1}\mathbf{\Lambda}^{i-1}\Phi_\lambda = \left[\bigoplus_{\mathrm{Im}(\lambda_j)=0} (\lambda_j^{i-1})\right] \bigoplus \left[\bigoplus_{\mathrm{Im}(\lambda_j)>0} \hat{\Lambda}_j^{i-1}\right] = \begin{pmatrix} \bullet & & & \\ & \bullet & & \\ & & \bullet & \\ & & & \begin{smallmatrix}\bullet&\bullet\\\bullet&\bullet\end{smallmatrix} \\ & & & & \begin{smallmatrix}\bullet&\bullet\\\bullet&\bullet\end{smallmatrix} \end{pmatrix} \equiv \widehat{\mathbf{\Lambda}}^{i-1} \in \mathbb{R}^{\ell\times\ell}. \tag{5.18}$$

When used in the objective function, these transformations yield an equivalent formula

$$\begin{aligned} &\left\| (\mathbf{W}\otimes \mathbb{I}_n) \left[\begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_m \end{pmatrix} - \begin{pmatrix} Z_\ell \Delta_{\Lambda_1} \\ \vdots \\ Z_\ell \Delta_{\Lambda_m} \end{pmatrix} \boldsymbol{\alpha} \right] \right\|_2 \\ &= \left\| \begin{pmatrix} \mathfrak{w}_1 \mathbf{x}_1 \\ \vdots \\ \mathfrak{w}_m \mathbf{x}_m \end{pmatrix} - \begin{pmatrix} \mathfrak{w}_1 (Z_\ell\Phi_\lambda)(\Phi_\lambda^{-1}\mathbf{\Lambda}^0\Phi_\lambda) \\ \vdots \\ \mathfrak{w}_m (Z_\ell\Phi_\lambda)(\Phi_\lambda^{-1}\mathbf{\Lambda}^{m-1}\Phi_\lambda) \end{pmatrix} \underbrace{(\Phi_\lambda^{-1}\boldsymbol{\alpha})}_{\boldsymbol{\rho}} \right\|_2 . \end{aligned} \tag{5.19}$$

If we introduce a change of variables in (5.19) by letting $\boldsymbol{\rho} = \Phi_\lambda^{-1}\boldsymbol{\alpha}$, then the optimal $\boldsymbol{\rho}$ in (5.19) must be real, and then the optimal $\boldsymbol{\alpha} = \Phi_\lambda \boldsymbol{\rho}$ reads

$$\boldsymbol{\alpha} = \left(\rho_1 \quad \ldots \quad \rho_{\ell_1}, \quad \rho_{\ell_1+1} + \mathfrak{i}\rho_{\ell_1+2}, \quad \rho_{\ell_1+1} - \mathfrak{i}\rho_{\ell_1+2}, \quad \ldots, \quad \rho_{\ell-1} + \mathfrak{i}\rho_\ell, \quad \rho_{\ell-1} - \mathfrak{i}\rho_\ell\right)^T . \tag{5.20}$$

□

The difference between (5.19) and the original formulation (3.4) is in replacing the diagonal scaling matrices $\mathbf{\Lambda}^{i-1}$ with block diagonal matrices $\widehat{\mathbf{\Lambda}}^{i-1} = \Phi_\lambda^{-1}\mathbf{\Lambda}^{i-1}\Phi_\lambda \equiv (\Phi_\lambda^{-1}\mathbf{\Lambda}\Phi_\lambda)^{i-1}$ containing 1×1 or 2×2 diagonal blocks. A similar statement holds for (3.5), where now we have real QR factorization $Z_\ell\Phi_\lambda = Q\left(\begin{smallmatrix} R \\ \mathbf{0}\end{smallmatrix}\right)$. Note that $Z_\ell\Phi_\lambda$ is of the same dimensions as Z_ℓ, but it is a real matrix, thus occupying half the storage needed for Z_ℓ. In a software implementation based on, e.g., LAPACK, $Z_\ell\Phi_\lambda$ will be actually computed in the form (5.17), so that this switching to real arithmetic is simple.

Needless to say, computing the QR factorization in real arithmetic is (this is estimated, but can be easily demonstrated in numerical experiments with sufficiently large dimensions) at least twice faster than in complex arithmetic. Moreover, as discussed above, in a high performance computing environment, we initially have $Z_\ell\Phi_\lambda$, and not its complexification Z_ℓ. After the real QR factorization of $Z_\ell\Phi_\lambda$, the steps (3.5), (3.2) are performed in real arithmetic. And, finally, the fact that the solution is guaranteed to have complex conjugacy structure (5.20) analogous to that of the selected eigenpairs, thus yielding real approximation, makes the final argument in favor of this formulation.

5.4.2. Algorithms of section 5.2 for real data. After rewriting the steps (3.4)–(3.11) over $\mathbb{R}$ using (5.18), (5.19), it remains to adapt the algorithms described in section 5.2 to the case of real R and with the block-diagonal matrices $\widehat{\mathbf{\Lambda}}^{i-1}$ instead of the diagonal $\mathbf{\Lambda}^{i-1}$. Clearly, the global structure of Algorithms 4 and 8 remains unchanged. The difference is in the structure (5.5), which is changed as illustrated below (three real eigenvalues ($\bullet$) and two complex conjugate pairs ($\blacklozenge$)):

$$T_{i-1}\mathbf{\Lambda}^{2^{i-1}} = \begin{pmatrix} * & * & * & * & * & * & * \\ & * & * & * & * & * & * \\ & & * & * & * & * & * \\ & & & * & * & * & * \\ & & & & * & * & * \\ & & & & & * & * \\ & & & & & & * \end{pmatrix} \begin{pmatrix} \bullet & & & & & & \\ & \bullet & & & & & \\ & & \bullet & & & & \\ & & & \blacklozenge & \blacklozenge & & \\ & & & \blacklozenge & \blacklozenge & & \\ & & & & & \blacklozenge & \blacklozenge \\ & & & & & \blacklozenge & \blacklozenge \end{pmatrix} = \begin{pmatrix} * & * & * & * & * & * & * \\ & * & * & * & * & * & * \\ & & * & * & * & * & * \\ & & & * & * & * & * \\ & & & \bigstar & * & * & * \\ & & & & & * & * \\ & & & & & \bigstar & * \end{pmatrix}, \quad \bullet, *, \blacklozenge, \bigstar \in \mathbb{R}.$$

To put it simply, each complex conjugate pair of Ritz values creates a bulge ($\bigstar$). Hence, before running algorithms described in section 5.2.2, it suffices to apply Givens rotations to first annihilate the bulges ($\bigstar$), whose number equals the total number of complex conjugate pairs. It is easily seen that the rotations can be applied independently; there can be at most $\ell/2$ rotations, so the total cost of this correction is $O(\ell^2)$). Such a correction is needed only in the $(2, 1)$ block, designated as B in section 5.2.2.

6. Concluding remarks.

In this paper we have provided a new firm numerical linear algebra framework for solving structured least squares problems that arise in applications of the Koopman/DMD analysis of dynamical systems (e.g., in computational fluid dynamics). Although the DMD was our main motivation that triggered this development, the applicability of our results extends to many other computational tasks such as, e.g., multistatic antenna array processing. Using error and perturbation analysis, we have explained the accuracy and the limitations of the normal equations based solution. Further, we have proposed new algorithms, based on a structure exploiting QR factorization, specific to the DMD framework. The new algorithm for computing the structured QR factorization has been presented with detailed numerical analysis and implementation details that can serve as blueprints for developing a high performance software. A LAPACK based implementation of all introduced algorithms is under development.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., Software Environ. Tools 9, SIAM, Philadelphia, 1999, https://doi.org/10.1137/1.9780898719604.

[2] H. Arbabi and I. Mezić, *Ergodic theory, dynamic mode decomposition, and computation of spectral properties of the Koopman operator*, SIAM J. Appl. Dyn. Syst., 16 (2017), pp. 2096–2126, https://doi.org/10.1137/17M1125236.

[3] T. Askham and J. N. Kutz, *Variable projection methods for an optimized dynamic mode decomposition*, SIAM J. Appl. Dyn. Syst., 17 (2018), pp. 380–416, https://doi.org/10.1137/M1124176.

[4] M. Benzi, G. H. Golub, and J. Liesen, *Numerical solution of saddle point problems*, Acta Numer., 14 (2005), pp. 1–137, https://doi.org/10.1017/S0962492904000212.

[5] D. A. Bistrian and I. M. Navon, *Efficiency of randomised dynamic mode decomposition for reduced order modelling*, Int. J. Comput. Fluid Dyn., 32 (2018), pp. 88–103, https://doi.org/10.1080/10618562.2018.1511049.

[6] Å. Björck, *Stability analysis of the method of seminormal equations for linear least squares problems*, Linear Algebra Appl., 88–89 (1987), pp. 31–48, https://doi.org/10.1016/0024-3795(87)90101-7.

[7] Å. Björck, *Numerical Methods in Matrix Computations*, Texts Appl. Math. 59, Springer, Cham, 2015.

[8] S. L. Brunton, B. W. Brunton, J. L. Proctor, E. Kaiser, and J. N. Kutz, *Chaos as an intermittently forced linear system*, Nature Commun., 8 (2017), 19.

[9] P. A. Businger and G. H. Golub, *Linear least squares solutions by Householder transformations*, Numer. Math., 7 (1965), pp. 269–276.

[10] A. Cat Le Ngo, J. See, and R. C.-W. Phan, *Sparsity in Dynamics of Spontaneous Subtle Emotions: Analysis & Application*, preprint, https://arxiv.org/abs/1601.04805, 2016.

[11] K. K. Chen, J. H. Tu, and C. W. Rowley, *Variants of dynamic mode decomposition: Boundary condition, Koopman, and Fourier analyses*, J. Nonlinear Sci., 22 (2012), pp. 887–915.

[12] S. T. M. Dawson, M. S. Hemati, M. O. Williams, and C. W. Rowley, *Characterizing and correcting for the effect of sensor noise in the dynamic mode decomposition*, Experiments in Fluids, 57 (2016), 42, https://doi.org/10.1007/s00348-016-2127-7.

[13] J. Demmel, *On Floating Point Errors in Cholesky*, LAPACK Working Note 14, Computer Science Department, University of Tennessee, Knoxville, TN, 1989.

[14] Z. Drmač, I. Mezić, and R. Mohr, *Data driven Koopman spectral analysis in Vandermonde–Cauchy form via the DFT: Numerical method and theoretical insights*, SIAM J. Sci. Comput., 41 (2019), pp. 3118–3151 https://doi.org/10.1137/18M1227688.

[15] Z. Drmač, I. Mezić, and R. Mohr, *Data driven modal decompositions: Analysis and enhancements*, SIAM J. Sci. Comput., 40 (2018), pp. A2253–A2285, https://doi.org/10.1137/17M1144155.

[16] Z. Drmač and Z. Bujanović, *On the failure of rank revealing QR factorization software—a case study*, ACM Trans. Math. Softw., 35 (2008), pp. 1–28.

[17] Z. Drmač, M. Omladič, and K. Veselić, *On the perturbation of the Cholesky factorization*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 1319–1332, https://doi.org/10.1137/S0895479893244717.

[18] S. Ghosal, V. Ramanan, S. Sarkar, S. Chakravarthy, and S. Sarkar, *Detection and analysis of combustion instability from hi-speed flame images using dynamic mode decomposition*, in ASME Dynamic Systems and Control Conference, Volume 1, ASME, New York, 2016, DSCC2016-9907, https://doi.org/10.1115/DSCC2016-9907.

[19] N. Gould, D. Orban, and T. Rees, *Projected Krylov methods for saddle-point systems*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 1329–1343, https://doi.org/10.1137/130916394.

[20] M. S. Hemati, C. W. Rowley, E. A. Deem, and L. N. Cattafesta, *De-Biasing the Dynamic Mode Decomposition for Applied Koopman Spectral Analysis*, preprint, https://arxiv.org/abs/1502.03854, 2015.

[21] M. S. Hemati, M. O. Williams, and C. W. Rowley, *Dynamic mode decomposition for large and streaming datasets*, Phys. Fluids, 26 (2014), 111701.

[22] R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1991.

[23] K. Ikeda, *Multiple-valued stationary state and its instability of the transmitted light by a ring cavity system*, Opt. Comm, 30 (1979), pp. 257–261.

[24] K. Ikeda, H. Daido, and O. Akimoto, *Optical turbulence: Chaotic behavior of transmitted light from a ring cavity*, Phys. Rev. Lett., 45 (1980), pp. 709–712, https://doi.org/10.1103/PhysRevLett.45.709.

[25] M. R. Jovanović, P. J. Schmid, and J. W. Nichols, *DMDSP—Sparsity–Promoting Dynamic Mode Decomposition*, http://people.ece.umn.edu/users/mihailo/software/dmdsp/, 2013–2014.

[26] M. R. Jovanović, P. J. Schmid, and J. W. Nichols, *Sparsity-promoting dynamic mode decomposition*, Phys. Fluids, 26 (2014), 024103.

[27] M. Kennedy, *Three steps to chaos.* I: *Evolution*, IEEE Trans. Circuits Systems, 40 (1993), pp. 640–656, https://doi.org/10.1109/81.246140.

[28] M. P. Kennedy, *Three steps to chaos.* II: *A Chua's circuit primer*, IEEE Trans. Circuits Systems, 40 (1993), pp. 657–674, https://doi.org/10.1109/81.246141.

[29] M. Korda and I. Mezić, *On convergence of extended dynamic mode decomposition to the Koopman operator*, J. Nonlinear Sci., 28 (2017), pp. 687–710.

[30] J. Kou and W. Zhang, *An improved criterion to select dominant modes from dynamic mode decomposition*, European J. Mech. B/Fluids, 62 (2017), pp. 109–129, https://doi.org/10.1016/j.euromechflu.2016.11.015.

[31] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor, *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*, SIAM, Philadelphia, 2016, https://doi.org/10.1137/1.9781611974508.

[32] H. Lev-Ari, *Efficient solution of linear matrix equations with application to multistatic antenna array processing*, Commun. Inf. Syst., 5 (2005), pp. 123–130, https://projecteuclid.org:443/euclid.cis/1149698475.

[33] J. Proctor, S. L. Brunton, and J. N. Kutz, *Dynamic mode decomposition with control*, SIAM J. Appl. Dyn. Syst., 15 (2016), pp. 142–161, https://doi.org/10.1137/15M1013857.

[34] J. L. Proctor and P. A. Eckhoff, *Discovering dynamic patterns from infectious disease data using dynamic mode decomposition*, Internat. Health, 7 (2015), pp. 139–145, https:

//doi.org/10.1093/inthealth/ihv009.

[35] A. Ray, S. Rakshit, D. Ghosh, and S. K. Dana, *Intermittent large deviation of chaotic trajectory in Ikeda map: Signature of extreme events*, Chaos, 29 (2019), 043131, https://doi.org/10.1063/1.5092741.

[36] C. W. Rowley, I. Mezić, S. Bagheri, P. Schlatter, and D. S. Henningson, *Spectral analysis of nonlinear flows*, J. Fluid Mech., 641 (2009), pp. 115–127.

[37] T. Sayadi and P. J. Schmid, *Parallel data-driven decomposition algorithm for large-scale datasets: With application to transitional boundary layers*, Theor. Comput. Fluid Dyn., 30 (2016), pp. 415–428.

[38] P. J. Schmid, *Dynamic mode decomposition of numerical and experimental data*, J. Fluid Mech., 656 (2010), pp. 5–28, https://doi.org/10.1017/S0022112010001217.

[39] P. J. Schmid, L. Li, M. P. Juniper, and O. Pust, *Applications of the dynamic mode decomposition*, Theor. Comput. Fluid Dyn., 25 (2011), pp. 249–259, https://doi.org/10.1007/s00162-010-0203-9.

[40] R. Schreiber and C. van Loan, *A storage-efficient WY representation for products of Householder transformations*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 53–57, https://doi.org/10.1137/0910005.

[41] B. Suri, J. Tithof, R. Mitchell, Jr., R. O. Grigoriev, and M. F. Schatz, *Velocity profile in a two-layer Kolmogorov-like flow*, Phys. Fluids, 26 (2014), 053601, https://doi.org/10.1063/1.4873417.

[42] K. Taira, S. Brunton, S. Dawson, C. Rowley, T. Colonius, J. B. McKeon, O. Schmidt, S. Gordeyev, V. Theofilis, and L. Ukeiley, *Modal analysis of fluid flows: An overview*, AIAA J., 55 (2017), pp. 4013–4041.

[43] N. Takeishi, Y. Kawahara, Y. Tabei, and T. Yairi, *Bayesian dynamic mode decomposition*, in Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 2814–2821, https://doi.org/10.24963/ijcai.2017/392.

[44] N. Takeishi, Y. Kawahara, and T. Yairi, *Learning Koopman invariant subspaces for dynamic mode decomposition*, in Advances in Neural Information Processing Systems 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., Curran Associates, Red Hook, NY, 2017, pp. 1130–1140, http://papers.nips.cc/paper/6713-learning-koopman-invariant-subspaces-for-dynamic-mode-decomposition.pdf.

[45] N. Takeishi, Y. Kawahara, and T. Yairi, *Sparse nonnegative dynamic mode decomposition*, in Proceeings of the 2017 IEEE International Conference on Image Processing (ICIP), IEEE, Washington, DC, 2017, pp. 2682–2686, https://doi.org/10.1109/ICIP.2017.8296769.

[46] N. Takeishi, Y. Kawahara, and T. Yairi, *Subspace dynamic mode decomposition for stochastic Koopman analysis*, Phys. Rev. E, 96 (2017), 033310, https://doi.org/10.1103/PhysRevE.96.033310.

[47] M. Taufer, O. Padron, P. Saponaro, and S. Patel, *Improving numerical reproducibility and stability in large-scale numerical simulations on GPUs*, in Proceedings of the 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS), IEEE, Washington, DC, 2010, pp. 1–9, https://doi.org/10.1109/IPDPS.2010.5470481.

[48] J. Tithof, B. Suri, R. K. Pallantla, R. O. Grigoriev, and M. F. Schatz, *Bifurcations in a quasi-two-dimensional Kolmogorov-like flow*, J. Fluid Mech., 828 (2017), pp. 837–866, https://doi.org/10.1017/jfm.2017.553.

[49] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz, *On dynamic mode decomposition: Theory and applications*, J. Comput. Dyn., 1 (2014), pp. 391–421, https://doi.org/10.3934/jcd.2014.1.391.

[50] A. van der Sluis, *Condition numbers and equilibration of matrices*, Numer. Math., 14 (1969), pp. 14–23.

[51] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, *A data–driven approximation of the Koopman operator: Extending dynamic mode decomposition*, J. Nonlinear Sci., 25 (2015), pp. 1307–1346, https://doi.org/10.1007/s00332-015-9258-5.