On least squares problems with certain Vandermonde–Khatri–Rao structure with applications to DMD

Zlatko Drmač^{1,*}and Igor Mezić² and Ryan Mohr³

¹ Faculty of Science, Department of Mathematics, University of Zagreb, 10000 Zagreb, Croatia.

² Department of Mechanical Engineering and Mathematics, University of California, Santa Barbara, CA 93106, USA

³ AIMdyn, Inc., Santa Barbara, CA 93101, USA

Abstract. This paper proposes a new computational method for solving structured least squares problems that arise in the process of identification of coherent structures in 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 subset of which 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 problem with the matrix that involves the Khatri-Rao product of a triangular and a Vandermonde matrix. Such a structure allows a very efficient normal equation based least squares solution, which is used in state of the art 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 semi-normal solution and QR factorization based algorithms. It is shown 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.

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

Key words: coherent structure, dynamic mode decomposition, Khatri–Rao product, Koopman operator, Krylov subspaces, proper orthogonal decomposition, Rayleigh-Ritz approximation, structured least squares, Vandermonde matrix

1. Introduction

Dynamic mode decomposition (DMD) [26] has become a tool of trade in computational fluid dynamics (CFD), both in the high fidelity numerical simulations and in the pure data driven scenarios; for a review see [30] and references therein. Its data driven framework and deep theoretical roots in the Koopman operator theory [25], [33], make DMD a versatile tool in a plethora of applications beyond CFD, e.g. for studying dynamics of infectious diseases [24] or for revealing spontaneous subtle emotions on human faces [5] in the field of affective computing.

^{*}Corresponding author. *Email addresses:* drmac@math.hr (Z. Drmač), mezic@ucsb.edu (I. Mezić), mohrr@aimdyn.com (R. Mohr)

A distinctive feature of the DMD is that it is purely data driven. It does not assume knowledge of the solution of the governing, generally nonlinear, equations obeyed by the dynamics under study; it does assume that the supplied data snapshots (vectors of observables) $\mathbf{f}_i \in \mathbb{C}^n$, are generated by a linear operator \mathbb{A} so that $\mathbf{f}_{i+1} = \mathbb{A}\mathbf{f}_i$, $i = 1, \ldots, m$, with some initial \mathbf{f}_1 , and a constant time lag δt . One can think of \mathbb{A} as e.g. a black-boxed numerical simulation software, or e.g. as a discretized physics taken by a high-speed camera such as in studying flame dynamics and combustion instabilities [14]. The spectral data for \mathbb{A} are then inferred 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 [25, 1, 21]. For a good approximation in the sense of 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 computed Ritz pairs (λ_j, z_j) 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 $\lambda_j = |\lambda_j| e^{i\omega_j \delta t}$:

$$\mathbf{f}_i = \sum_{j=1}^m z_j \mathfrak{a}_j \lambda_j^{i-1} \equiv \sum_{j=1}^m z_j \mathfrak{a}_j |\lambda_j|^{i-1} e^{\mathbf{i}\omega_j(i-1)\delta t}, \quad i = 1, \dots, m.$$
(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 – the task is to determine $\ell < m$, indices $\varsigma_1 < \cdots < \varsigma_\ell$ and the coefficients α_i to achieve high fidelity of the snapshot representations

$$\mathbf{f}_i \approx \sum_{j=1}^{\ell} z_{\varsigma_j} \alpha_j \lambda_{\varsigma_j}^{i-1}, \quad i = 1, \dots, m.$$
(1.2)

A solution of this sparse reconstruction problem is proposed in [19] as a combination of compressed sensing and convex optimization – the resulting sparsity promoting DMD (DMDSP) has become an important addition to the DMD technology. One of the key numerical steps in DMDSP is solution of a structured least squares problem for the coefficients α_i in (1.2) via the normal equations.

From the numerical point of view, using normal equations for least squares solution is an illadvised approach because the classical spectral condition number of the problem is squared. We examine this issue, provide a floating point error analysis and analyze condition numbers that determine sensitivity of the problem. This allows us to explain good results obtained by DMDSP, and to determine the limits for applicability of the normal equations approach. To ensure robustness of the least squares representation (1.2), we develop a QR factorization based algorithm. Together with detailed descriptions and numerical analysis of the new proposed algorithms, we provide guidelines for numerical software development.

The material is organized as follows. In §2, we briefly review the DMD and in §2.2 we state the problem of snapshot reconstruction using the selected modes; the sparsity promoting DMD (DMDSP) is reviewed in §2.2.1. In §3 we first 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. Then, in §3.2, the explicit formulas for the normal equations solutions are given for this weighted error. In §4 we propose a modification of the DMDSP – its polishing phase can be considerably improved by replacing the variation of the Lagrangian technique with deployment of certain canonical projections. Numerical properties of the normal equations approach are studied in detail in §5. We identify the relevant condition number and argue that the underlying Khatri–Rao product structure of the normal equations usually ensures numerical accuracy despite potentially high condition numbers of both the Vandermonde matrix of Ritz values and of the Ritz vectors. To the best of our knowledge, this aspect of the DMDSP has not been considered before.

However, normal equations have its limitations, and we illustrate this by contrived small dimensional examples. This motivates the development in §6, where we consider solution methods via the QR factorization – the corrected seminormal solution in §6.1 and the pure QR based procedure. The key technical difficulty is the computation of the QR factorization of the least squares matrix; this is resolved in §6.2 where we propose an efficient recursive scheme that exploits the Khatri–Rao product structure involving a Vandermonde matrix. Fine details such as using only real arithmetic in the case of real snapshots are discussed in §6.4. Finally, in §8 we provide Matlab source codes for some of the presented 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. For a more extensive study of DMD and its variations and applications, see e.g. [27], [6], [16], [7], [17].

2.1. DMD and its variations

The DMD [27] is outlined in Algorithm 1.

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

Input:

- $\mathbf{X}_m = (\mathbf{x}_1, \dots, \mathbf{x}_m), \mathbf{Y}_m = (\mathbf{y}_1, \dots, \mathbf{y}_m) \in \mathbb{C}^{n \times m}$ 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: $\mathbf{X}_m = U\Sigma\Phi^*, U \in \mathbb{C}^{n \times m}, \Sigma = diag(\sigma_i)_{i=1}^m$ }
- 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: $[B_k, \Lambda_k] = \operatorname{eig}(S_k)$ { $\Lambda_k = \operatorname{diag}(\lambda_j)_{j=1}^k$; $S_k B_k(:, j) = \lambda_j B_k(:, j)$; $||B_k(:, j)||_2 = 1$ } {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: $Z_k = U_k B_k \{Ritz \ vectors\}$ **Output:** Z_k, Λ_k

2.1.1. DDMD RRR. Recently, in [10], we proposed an enhancement of the 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. For the reader's convenience, the method is outlined in Algorithm 2; for detailed description and analysis, and a Matlab implementation we refer to [10].

Algorithm 2 $[Z_k, \Lambda_k, r_k, \rho_k] = \text{DDMD}_RRR(\mathbf{X}_m, \mathbf{Y}_m; \epsilon)$ {Refined Rayleigh-Ritz DDMD}

Input:

- $\mathbf{X}_m = (\mathbf{x}_1, \dots, \mathbf{x}_m), \mathbf{Y}_m = (\mathbf{y}_1, \dots, \mathbf{y}_m) \in \mathbb{C}^{n \times m}$ 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 $m \ll n$.)
- Tolerance level ϵ for numerical rank determination.

1:
$$D_x = \operatorname{diag}(\|\mathbf{X}_m(:,i)\|_2)_{i=1}^m; \mathbf{X}_m^{(1)} = \mathbf{X}_m D_x^{\dagger}; \mathbf{Y}_m^{(1)} = \mathbf{Y}_m D_x^{\dagger} \{ Optional \, scaling. \}$$

2: $[U, \Sigma, V] = svd(\mathbf{X}_m^{(1)}); \{ The \, thin \, SVD: \, \mathbf{X}_m^{(1)} = U\Sigma V^*, \, U \in \mathbb{C}^{n \times m}, \, \Sigma = \operatorname{diag}(\sigma_i)_{i=1}^m \}$
3: Determine numerical rank k , with the threshold $\epsilon: k = \max\{i : \sigma_i \geq \sigma_1 \epsilon\}$.
4: Set $U_k = U(:, 1:k), \, V_k = V(:, 1:k), \, \Sigma_k = \Sigma(1:k, 1:k)$
5: $B_k = \mathbf{Y}_m^{(1)}(V_k \Sigma_k^{-1}); \{ Schmid's \, data \, driven \, formula \, for \, AU_k \}$
6: $[Q, R] = qr((U_k, B_k)); \{ The \, thin \, QR \, factorization: \, (U_k, B_k) = QR; \, Q \, not \, computed \}$
7: $S_k = \operatorname{diag}(\overline{R_{ii}})_{i=1}^k R(1:k,k+1:2k) \{ S_k = U_k^* AU_k \, is \, the \, Rayleigh \, quotient \}$
8: $\Lambda_k = \operatorname{eig}(S_k) \, \{ \Lambda_k = \operatorname{diag}(\lambda_i)_{i=1}^k; \, Ritz \, values, \, i.e. \, eigenvalues \, of \, S_k \}$
9: for $i = 1, \dots, k$ do
10: $R_{\lambda_i} = \begin{pmatrix} R^{(1:k,k+1:2k)-\lambda_i}R^{(1:k,1:k)} \\ R^{(k+1:2k,k+1:2k)} \end{pmatrix}$
11: $[\sigma_{\lambda_i}, w_{\lambda_i}] = svd_{\min}(R_{\lambda_i}); \{ Mnimal \, singular \, value \, of \, R_{\lambda_i} \, and \, the \, corresponding \, right \, singular \, vector \}$
12: $W_k(:,i) = w_{\lambda_i}; r_k(i) = \sigma_{\lambda_i} \, \{ Optimal \, residual, \, \sigma_{\lambda_i} = \|R_{\lambda_i}w_{\lambda_i}\|_2 \}$
13: $\rho_k(i) = w_{\lambda_i}^* S_k w_{\lambda_i} \, \{ Rayleigh \, quotient, \, \rho_k(i) = (U_k w_{\lambda_i})^* \mathbb{A}(U_k w_{\lambda_i}) \}$
14: end for
15: $Z_k = U_k W_k \, \{ Refined \, Ritz \, vectors \}$
Output: $Z_k, \, \Lambda_k, \, r_k, \, \rho_k$

2.2. Sparse reconstruction using selected Ritz pairs

In matrix form, (1.1) is compactly written as

$$\mathbf{X}_{m} = \begin{pmatrix} z_{1} & z_{2} & \dots & z_{m} \end{pmatrix} \begin{pmatrix} \mathfrak{a}_{1} & & & \\ & \mathfrak{a}_{2} & & \\ & & \ddots & \\ & & & \mathfrak{a}_{m} \end{pmatrix} \begin{pmatrix} 1 & \lambda_{1} & \dots & \lambda_{1}^{m-1} \\ 1 & \lambda_{2} & \dots & \lambda_{2}^{m-1} \\ \vdots & \vdots & \dots & \vdots \\ 1 & \lambda_{m} & \dots & \lambda_{m}^{m-1} \end{pmatrix},$$
(2.3)

where the coefficients \mathfrak{a}_j are computed as

$$(\mathfrak{a}_{j})_{j=1}^{m} = Z_{m}^{\dagger} \mathbf{X}_{m}(:,1).$$
(2.4)

In the framework of the Schmid's DMD, the amplitudes are usually determined by the formula (2.4). Since $Z_m = U_m B_m$ (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

$$Z_m^{\dagger} \mathbf{X}_m(:,1) = B_m^{-1}(U_m^* \mathbf{X}_m(:,1)).$$
(2.5)

For a formal proof of the existence of the representation (2.3) and relation to the formulation via the Krylov decomposition and the Frobenius companion matrix, we refer to [13].

The main goal of the representations (1.1), (2.3) is revealing the underlying structure in the snapshots. It is best achieved with an approximate representation using as few as possible Ritz pairs (i.e. approximate eigenpairs). Ideally, the reconstruction is successful with small number of

Ritz vectors that have small residuals (i.e. corresponding to selected eigenvectors of the underlying operator that is accessible only through the sequence of snapshots \mathbf{f}_i).

If we desire to take only $\ell < m$ 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

$$\mathbf{X}_{m} \approx \begin{pmatrix} z_{\varsigma_{1}} & z_{\varsigma_{2}} & \dots & z_{\varsigma_{\ell}} \end{pmatrix} \begin{pmatrix} \mathfrak{a}_{\varsigma_{1}} & & & \\ & \mathfrak{a}_{\varsigma_{2}} & & \\ & & \ddots & \\ & & & \mathfrak{a}_{\varsigma_{\ell}} \end{pmatrix} \begin{pmatrix} 1 & \lambda_{\varsigma_{1}} & \dots & \lambda_{\varsigma_{1}}^{m-1} \\ 1 & \lambda_{\varsigma_{2}} & \dots & \lambda_{\varsigma_{2}}^{m-1} \\ \vdots & \vdots & \dots & \vdots \\ 1 & \lambda_{\varsigma_{\ell}} & \dots & \lambda_{\varsigma_{\ell}}^{m-1} \end{pmatrix} \equiv Z_{\varsigma} D_{\mathfrak{a}} \mathbb{V}_{\varsigma}.$$
(2.6)

This seems 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 (2.6) the reconstruction error $\|\mathbf{X}_m - Z_{\varsigma} D_{\mathfrak{a}} \mathbb{V}_{\varsigma}\|_F^2$ 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 sparsity promoting DMD [19], that we briefly review in §2.2.1. Another strategy, proposed in [22, §3], is to choose modes that are dominant with respect to their influence over all snapshots.

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 (2.6) by recomputing them by solving a least squares problem with fixed $z_{\varsigma_1}, \ldots, z_{\varsigma_{\ell}}$. Here, optionally, we can use data driven refinements of the selected Ritz pairs $(z_{\varsigma_i}, \lambda_{\varsigma_i})$, see [10].

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

$$\|\mathbf{X}_m - Z_k D_{\mathfrak{a}} \mathbb{V}_{k,m}\|_F^2 + \gamma \|\mathfrak{a}\|_0 \longrightarrow \min_{\widetilde{\mathfrak{a}}},$$
(2.7)

where $\|\mathbf{a}\|_0$ denotes the number of nonzero entries in the vector $\mathbf{a} = (\mathbf{a}_i)_{i=1}^k$ of the new coefficients; the parameter $\gamma \geq 0$ penalizes non-sparsity of the solution \mathbf{a} . The measure of sparsity $\|\mathbf{a}\|_0$ is in practical computations relaxed by using the ℓ_1 norm, $\|\mathbf{a}\|_1$, which turns (2.7) into a convex optimization problem

$$\|\mathbf{X}_m - Z_k D_{\mathfrak{a}} \mathbb{V}_{k,m}\|_F^2 + \gamma \|\mathfrak{a}\|_1 \longrightarrow \min_{\mathfrak{a}}.$$
 (2.8)

In [19], for a given value of γ , the problem is solved in two steps:

- 1. (2.8) is solved using the ADMM, and the optimal \mathfrak{a} 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 thus annihilated entries – they define 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 \mathfrak{a} can be characterized by $E^T \mathfrak{a} = \mathbf{0}$.
- 2. Once (2.8) has identified the sparsity structure, the coefficients a_i are computed by solving the least squares problem with sparsity constraint:

$$\|\mathbf{X}_m - Z_k D_{\mathfrak{a}} \mathbb{V}_{k,m}\|_F^2 \longrightarrow \min_{\mathfrak{a}, E^T \mathfrak{a} = \mathbf{0}}$$
(2.9)

In DMDSP terminology, 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) 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 our best understanding, is determined experimentally for each particular problem.[†] Further, ADMM uses

[†]See e.g. the software [20] for test examples in [19].

the augmented Lagrangian that requires an additional penalty parameter ρ , which means that the user must supply two parameters (see [19, §A.]).

The optimization problem (2.9) is solved by variation of the Lagrangian (see [19, Appendix C]). This can be done more efficiently and we discuss it in §4. 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 §6.4.

We complete this review with an important observation.

Remark 2.1. The minimizations (2.7), (2.8), with the sparsity constraint, look analogous to the compressed sensing problems; in fact [19] 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 (2.6) is heavily overdetermined and generically with unique solution, while in the compressed sensing framework one has an underestimated least squares problem and the sparsity constraint is imposed over a manifold of solutions. These are two fundamentally different situations. We further comment this issue in §6.3.1.

3. Snapshot reconstruction: structure, sensitivity and condition numbers

Suppose we have selected ℓ modes, re-indexed so that they are the leading ones, $1, \ldots, \ell$, and that we seek an approximate snapshot reconstruction

$$\mathbf{f}_i \approx \sum_{j=1}^{\ell} z_j \alpha_j \lambda_j^{i-1}, \quad i = 1, \dots, m.$$
(3.10)

The pairs (λ_j, z_j) are approximate eigenpairs of \mathbb{A} , e.g. the Ritz pairs. In terms of Algorithm 1 or Algorithm 2, $Z_{\ell} \equiv \begin{pmatrix} z_1 & \dots & z_{\ell} \end{pmatrix} = U_k \widetilde{B}_{\ell}$ with some $k \times \ell$ matrix \widetilde{B}_{ℓ} , and $\ell \leq k \leq \min(m, n)$.

In general, selection of modes will be determined by a wizard, aimed at achieving certain goals; an example is the DMDSP, reviewed in §2.2.1. But once the (λ_j, z_j) 's have been selected, we may ask whether the coefficients $\alpha_j = \mathfrak{a}_j$ can be recomputed so that the reconstruction (3.10) is improved over all snapshots. This is e.g. the polishing phase (2.9) in DMDSP.

3.1. Setting the scene

The task is to determine, for given (λ_i, z_i) 's and nonnegative weights \mathfrak{w}_i , the α_i 's to achieve

$$\sum_{i=1}^{m} \mathfrak{w}_i^2 \|\mathbf{f}_i - \sum_{j=1}^{\ell} z_j \alpha_j \lambda_j^{i-1} \|_2^2 \longrightarrow \min.$$
(3.11)

Set $\mathbf{W} = \text{diag}(\mathbf{w}_i)_{i=1}^m$. The weights $\mathbf{w}_i > 0$ are used to emphasize snapshots whose reconstruction is more important; with suitable choices of \mathbf{W} we can target discrete time subintervals of particular interest, or e.g. introduce forgetting factors that give less importance to older information. We believe that this added functionality will prove useful in applications and the non-autonomous setting.

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 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 $\|\sqrt{\Omega}\cdot\|_2$, where Ω is positive definite. (If Ω is not diagonal, then $\sqrt{\Omega}$ stands for the upper triangular Cholesky factor.) 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 Ω -induced inner product we refer the reader to [10].

To introduce all necessary notation, define $\mathbf{\Lambda} = \operatorname{diag}(\lambda_j)_{j=1}^{\ell}$,

$$\vec{\boldsymbol{\alpha}} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_\ell \end{pmatrix}, \quad \Delta_{\boldsymbol{\alpha}} = \begin{pmatrix} \alpha_1 & 0 & \cdot & 0 \\ 0 & \alpha_2 & \cdot & \cdot \\ \vdots & \ddots & \cdot & 0 \\ 0 & \cdot & 0 & \alpha_\ell \end{pmatrix}, \quad \Lambda_i = \begin{pmatrix} \lambda_1^{i-1} \\ \lambda_2^{i-1} \\ \vdots \\ \lambda_\ell^{i-1} \end{pmatrix}, \quad \Delta_{\Lambda_i} = \begin{pmatrix} \lambda_1^{i-1} & 0 & \cdot & 0 \\ 0 & \lambda_2^{i-1} & \cdot & \cdot \\ \vdots & \ddots & \cdot & 0 \\ 0 & \cdot & 0 & \lambda_\ell^{i-1} \end{pmatrix} \equiv \boldsymbol{\Lambda}^{i-1},$$

and write the objective (3.11) as the function of $\vec{\alpha}$,

$$\Omega^{2}(\vec{\boldsymbol{\alpha}}) \equiv \| [\mathbf{X}_{m} - Z_{\ell} \Delta_{\boldsymbol{\alpha}} (\Lambda_{1} \quad \Lambda_{2} \quad \dots \quad \Lambda_{m})] \mathbf{W} \|_{F}^{2} \longrightarrow \min, \qquad (3.12)$$

4.

where

$$(\Lambda_1 \quad \Lambda_2 \quad \dots \quad \Lambda_m) = \begin{pmatrix} 1 \quad \lambda_1 \quad \dots \quad \lambda_1^{m-1} \\ 1 \quad \lambda_2 \quad \dots \quad \lambda_2^{m-1} \\ \vdots \quad \vdots \quad \dots \quad \vdots \\ 1 \quad \lambda_\ell \quad \dots \quad \lambda_\ell^{m-1} \end{pmatrix} \equiv \mathbb{V}_{\ell,m} \in \mathbb{C}^{\ell \times m}.$$
 (3.13)

Schematically, we have, assuming n > m,

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

$$\Omega^{2}(\vec{\boldsymbol{\alpha}}) = \| \begin{bmatrix} (\mathbf{f}_{1} \quad \mathbf{f}_{2} \quad \dots \quad \mathbf{f}_{m}) - Z_{\ell} \Delta_{\boldsymbol{\alpha}} (\Lambda_{1} \quad \Lambda_{2} \quad \dots \quad \Lambda_{m}) \end{bmatrix} \mathbf{W} \|_{F}^{2}$$

$$= \| (\mathbf{W} \otimes \mathbb{I}_{n}) \begin{bmatrix} \mathbf{f}_{1} \\ \vdots \\ \mathbf{f}_{m} \end{bmatrix} - \begin{pmatrix} Z_{\ell} \Delta_{\boldsymbol{\alpha}} \Lambda_{1} \\ \vdots \\ Z_{\ell} \Delta_{\boldsymbol{\alpha}} \Lambda_{m} \end{pmatrix} \end{bmatrix} \|_{2}^{2} = \| \begin{pmatrix} \mathfrak{w}_{1} \mathbf{f}_{1} \\ \vdots \\ \mathfrak{w}_{m} \mathbf{f}_{m} \end{pmatrix} - \begin{pmatrix} \mathfrak{w}_{1} Z_{\ell} \Delta_{\Lambda_{1}} \\ \vdots \\ \mathfrak{w}_{m} Z_{\ell} \Delta_{\Lambda_{m}} \end{pmatrix} \vec{\boldsymbol{\alpha}} \|_{2}^{2} (3.14)$$

In principle, (3.14) is a standard linear least squares problem that can be solved using an off-theshelf solver for the generic " $||Ax - b||_2 \rightarrow \min$ " problem. 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 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 $Z_{\ell} = Q\begin{pmatrix} R \\ 0 \end{pmatrix}$ 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

$$\Omega^{2}(\vec{\boldsymbol{\alpha}}) = \| \begin{pmatrix} \boldsymbol{\mathfrak{w}}_{1}Q^{*}\mathbf{f}_{1} \\ \vdots \\ \boldsymbol{\mathfrak{w}}_{m}Q^{*}\mathbf{f}_{m} \end{pmatrix} - \begin{pmatrix} \boldsymbol{\mathfrak{w}}_{1}\begin{pmatrix} R \\ 0 \end{pmatrix} \Delta_{\Lambda_{1}} \\ \vdots \\ \boldsymbol{\mathfrak{w}}_{m}\begin{pmatrix} R \\ 0 \end{pmatrix} \Delta_{\Lambda_{m}} \end{pmatrix} \vec{\boldsymbol{\alpha}} \|_{2}^{2}.$$
(3.15)

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

$$\Omega^{2}(\vec{\boldsymbol{\alpha}}) = \|(\mathbf{W} \otimes \mathbb{I}_{\ell}) \begin{bmatrix} \begin{pmatrix} \mathbf{g}_{1} \\ \vdots \\ \mathbf{g}_{m} \end{pmatrix} - \begin{pmatrix} R\Delta_{\Lambda_{1}} \\ \vdots \\ R\Delta_{\Lambda_{m}} \end{pmatrix} \vec{\boldsymbol{\alpha}} \|_{2}^{2} + \sum_{i=1}^{m} \mathfrak{w}_{i}^{2} \|\mathbf{h}_{i}\|_{2}^{2}$$
(3.16)

$$\equiv \| (\mathbf{W} \otimes \mathbb{I}_{\ell}) \left[\begin{pmatrix} \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_m \end{pmatrix} - (\mathbb{I}_m \otimes R) \begin{pmatrix} \Delta_{\Lambda_1} \\ \vdots \\ \Delta_{\Lambda_m} \end{pmatrix} \vec{\alpha} \right] \|_2^2 + \sum_{i=1}^m \mathfrak{w}_i^2 \| \mathbf{h}_i \|_2^2$$
(3.17)

$$\equiv \| \begin{pmatrix} \mathbf{w}_1 \mathbf{g}_1 \\ \vdots \\ \mathbf{w}_m \mathbf{g}_m \end{pmatrix} - (\mathbf{W} \otimes R) \begin{pmatrix} \Delta_{\Lambda_1} \\ \vdots \\ \Delta_{\Lambda_m} \end{pmatrix} \vec{\alpha} \|_2^2 + \sum_{i=1}^m \mathbf{w}_i^2 \| \mathbf{h}_i \|_2^2, \qquad (3.18)$$

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

Remark 3.1. 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 column 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}_{ℓ} .) 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}_{ℓ} .

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

$$\Omega^{2}(\vec{\boldsymbol{\alpha}}) = \| \begin{bmatrix} U_{m} \Sigma_{m} V_{m}^{*} - U_{k} \widetilde{B}_{\ell} \Delta_{\boldsymbol{\alpha}} \left(\Lambda_{1} \quad \Lambda_{2} \quad \dots \quad \Lambda_{m} \right) \end{bmatrix} \mathbf{W} \|_{F}^{2} \quad (\text{since } U_{k} = U_{m} U_{m}^{*} U_{k})$$
$$= \| \begin{bmatrix} \Sigma_{m} V_{m}^{*} - \begin{pmatrix} \widetilde{B}_{\ell} \\ \mathbf{0}_{m-k,\ell} \end{pmatrix} \Delta_{\boldsymbol{\alpha}} \left(\Lambda_{1} \quad \Lambda_{2} \quad \dots \quad \Lambda_{m} \right) \end{bmatrix} \mathbf{W} \|_{F}^{2}$$
$$= \| \begin{bmatrix} \Sigma_{k} V_{k}^{*} - \widetilde{B}_{\ell} \Delta_{\boldsymbol{\alpha}} \left(\Lambda_{1} \quad \Lambda_{2} \quad \dots \quad \Lambda_{m} \right) \end{bmatrix} \mathbf{W} \|_{F}^{2} + \sum_{j=k+1}^{m} \sigma_{j}^{2} \| V_{m}^{*}(j, :) \mathbf{W} \|_{2}^{2},$$

and (as in the unweighted case considered in [19]) 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 $\Sigma_k V_k^*$ ($\mathbf{f}_i \equiv (\Sigma_k V_k^*)(:,i)$), the Ritz vectors are in \tilde{B}_{ℓ} ($Z_{\ell} \equiv \tilde{B}_{\ell}$), and we may proceed with the QR factorization $\tilde{B}_{\ell} = Q'R$, thus removing the large dimension n in the very first step. This is included as a special case in the generic description (3.14 - 3.16) which appears in other applications (outside the DMD framework; see e.g. [23]) and it is thus preferred as a general form of the structured LS problem.

Remark 3.2. An efficient alternative approach to reducing the dimension is the QR-compressed scheme [10, §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 [13], where Z_{ℓ} is computed directly (without the DMD) from the eigenvectors of the companion matrix (inverse of the Vandermonde matrix from (2.3)).

3.2. Structure of the LS problem and normal equations solution

The objective function $\Omega(\vec{\alpha})$ has a very particular structure that allows a rather elegant explicit formula [19, §II.B] for the optimal $\vec{\alpha}$ 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 3.3. With the notation as above, the unique solution $\vec{\alpha}$ of the LS problem (3.11) is

$$\vec{\boldsymbol{\alpha}} = [(R^*R) \circ (\overline{\mathbb{V}_{\ell,m} \mathbf{W}^2 \mathbb{V}_{\ell,m}^*})]^{-1} [(\overline{\mathbb{V}_{\ell,m} \mathbf{W}} \circ (R^*G \mathbf{W}))\mathbf{e}],$$
(3.19)

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

$$\vec{\boldsymbol{\alpha}} = [(Z_{\ell}^* Z_{\ell}) \circ (\overline{\mathbb{V}_{\ell,m} \mathbf{W}^2 \mathbb{V}_{\ell,m}^*})]^{-1} [(\overline{\mathbb{V}_{\ell,m} \mathbf{W}} \circ (Z_{\ell}^* \mathbf{X}_m \mathbf{W}))\mathbf{e}].$$
(3.20)

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

$$\|(\mathbf{W}\otimes\mathbb{I}_{\ell})\left[\vec{\mathbf{g}}-S\vec{\alpha}\right]\|_{2} \longrightarrow \min, \text{ where } \vec{\mathbf{g}} = \begin{pmatrix} \mathbf{g}_{1} \\ \vdots \\ \mathbf{g}_{m} \end{pmatrix}, \quad S = (\mathbb{I}_{m}\otimes R)\begin{pmatrix} \Delta_{\Lambda_{1}} \\ \vdots \\ \Delta_{\Lambda_{m}} \end{pmatrix} \equiv \begin{pmatrix} R\Delta_{\Lambda_{1}} \\ \vdots \\ R\Delta_{\Lambda_{m}} \end{pmatrix}. \quad (3.21)$$

Hence, the optimal $\vec{\alpha}$ is given by the Moore–Penrose generalized inverse, $\vec{\alpha} = S_{\mathfrak{w}}^{\dagger} \vec{\mathbf{g}}_{\mathfrak{w}}$, 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.

$$\vec{\boldsymbol{\alpha}} = S_{\boldsymbol{\vartheta}}^{\dagger} \vec{\mathbf{g}}_{\boldsymbol{\vartheta}} = (S_{\boldsymbol{\vartheta}}^* S_{\boldsymbol{\vartheta}})^{-1} S_{\boldsymbol{\vartheta}}^* \vec{\mathbf{g}}_{\boldsymbol{\vartheta}} = (\sum_{k=1}^m \boldsymbol{\vartheta}_k^2 \Delta_{\Lambda_k}^* R^* R \Delta_{\Lambda_k})^{-1} \sum_{i=1}^m \boldsymbol{\vartheta}_i \Delta_{\Lambda_i}^* (\boldsymbol{\vartheta}_i R^* \mathbf{g}_i).$$
(3.22)

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

$$\sum_{k=1}^{m} \mathfrak{w}_k^2 \Delta_{\Lambda_k}^* R^* R \Delta_{\Lambda_k} = \sum_{k=1}^{m} \mathfrak{w}_k^2 (R^* R) \circ (\overline{\Lambda_k} \Lambda_k^T) = (R^* R) \circ \sum_{k=1}^{m} \mathfrak{w}_k^2 \overline{\Lambda_k} \Lambda_k^T$$
(3.23)

$$= (R^*R) \circ (\overline{\mathbb{V}}_{\ell,m} \mathbf{W}^2 \mathbb{V}_{\ell,m}^T) = (R^*R) \circ (\overline{\mathbb{V}_{\ell,m} \mathbf{W}^2 \mathbb{V}_{\ell,m}^*}), \qquad (3.24)$$

$$\sum_{i=1}^{m} \mathfrak{w}_i \Delta^*_{\Lambda_i}(\mathfrak{w}_i R^* \mathbf{g}_i) = (\overline{\mathbb{V}_{\ell,m}} \mathbf{W} \circ (R^* G \mathbf{W})) \mathbf{e}.$$
(3.25)

Note that Theorem 3.3 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 (3.19, 3.20), which appears to be new, contains the formula from [19, §II.B] as a special unweighted case. Since the Hadamard product of two positive definite matrices remains positive definite, the solution of (3.19) 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 (3.19) in terms of arithmetic operations is dominated by $O(m\ell^2) + O(\ell^3)$. Typically, $m \gg \ell$.

In fact, the relations (3.21 - 3.25), which implied (3.19), hide an elegant structure that we discuss in §3.2.1.

3.2.1. On Khatri–Rao structure of the snapshot reconstruction problem. Recall, 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

$$A \odot B = \begin{pmatrix} a_1 \otimes b_1 & a_2 \otimes b_2 & \dots & a_n \otimes b_n \end{pmatrix} \in \mathbb{C}^{pq \times n}.$$
(3.26)

The following proposition contains well known facts; it is included solely for the reader's convenience. For more detailed study of matrix products involved in this section we refer to [18, Chapters 4 and 5].

Proposition 3.4. The Khatri–Rao product satisfies the following relations:

• For any matrices A, B, C, D of appropriate dimensions

$$(AB) \odot (CD) = (A \otimes C)(B \odot D). \tag{3.27}$$

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

$$\operatorname{vec}(ABC) = (C^T \odot A) \operatorname{diag}(B).$$
(3.28)

(Recall, if B is not diagonal, $vec(ABC) = (C^T \otimes A)vec(B)$.)

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

$$(A \odot B)^T (A \odot B) = (B \odot A)^T (B \odot A) = (A^T A) \circ (B^T B),$$
(3.29)

$$(A \odot B)^* (A \odot B) = (B \odot A)^* (B \odot A) = (A^* A) \circ (B^* B).$$
(3.30)

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

$$\varpi_{\ell,m} = \begin{pmatrix} 1 & 2 & 3 & \dots & m & m+1 & m+2 & m+3 & \dots & 2m & \dots & m(\ell-1)+1 & \dots & m\ell \\ 1 & \ell+1 & 2\ell+1 & \dots & (m-1)\ell+1 & 2 & \ell+2 & 2\ell+2 & \dots & (m-1)\ell+2 & \dots & \ell & \dots & \ell+(m-1)\ell \end{pmatrix}.$$
(3.31)

(In Matlab, we can generate $\varpi_{\ell,m}$ as $\varpi_{\ell,m} = \text{reshape}(\text{reshape}(1 : \ell * m, \ell, m)', \ell * m, 1)$, which intuitively describes $\varpi_{\ell,m}$ as transformation of indices between a column and a row major ordering of a two dimensional array.)

Proposition 3.5. The LS problem (3.21) 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, $S(\varpi_{\ell,m}, :) = R \odot \mathbb{V}_{\ell,m}^T$.

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

(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. Brute-force.

Since Π is orthogonal, the problem reduces to computing the QR factorization of $R \odot \mathbb{V}_{\ell,m}^T$, and the objective (3.21) can be written as

$$\|\vec{\mathbf{g}} - S\vec{\boldsymbol{\alpha}}\|_2 \equiv \|\vec{\mathbf{g}} - \Pi(R \odot \mathbb{V}_{\ell,m}^T)\vec{\boldsymbol{\alpha}}\|_2 \equiv \|\Pi^T\vec{\mathbf{g}} - (R \odot \mathbb{V}_{\ell,m}^T)\vec{\boldsymbol{\alpha}}\|_2 \longrightarrow \min.$$
(3.32)

Note that $\vec{\mathbf{g}} = \operatorname{vec}(G)$ and that $\Pi^T \vec{\mathbf{g}} = \operatorname{vec}(G^T)$, where G is from (3.25), and $\operatorname{vec}(\cdot)$ 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 least squares 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

$$||H - G_{rec} \operatorname{diag}(\alpha_i)_{i=1}^{\ell} G_{tr}^T ||_F \longrightarrow \min_{\alpha_i} A_{tr}^{\ell}$$

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 *i*th scatterer. We refer the reader to [23], where the unweighted (i.e. $\mathbf{W} = \mathbb{I}$) version of (3.19) is derived as normal equations solution based on the properties of the Khatri–Rao product. In fact, Theorem 3.3 provides a generalization of [23] to weighted least squares, and the material of this section supplies the supporting numerical analysis. (The formulas in Theorem 3.3 do not use the fact that $\mathbb{V}_{\ell,m}$ is Vandermonde matrix.)

4. An improvement of the sparsity promoting DMDSP

In the notation of [19], with $\ell = k$ (i.e. using k modes, where k is the dimension of the POD basis), we can write $\Omega^2(\vec{\alpha})$ in the unweighted case ($\mathbf{W} = \mathbb{I}_m$) as

$$\Omega^{2}(\vec{\boldsymbol{\alpha}}) = \vec{\boldsymbol{\alpha}}^{*} \mathbb{P} \vec{\boldsymbol{\alpha}} - \mathbf{q}^{*} \vec{\boldsymbol{\alpha}} - \vec{\boldsymbol{\alpha}}^{*} \mathbf{q} + \|\Sigma_{k}\|_{F}^{2}, \text{ where } \mathbb{P} = (Z_{k}^{*} Z_{k}) \circ (\overline{\mathbb{V}_{k,m} \mathbb{V}_{k,m}^{*}}),$$
(4.33)

and

$$\mathbf{q} = \operatorname{diag}(\overline{\mathbb{V}_{k,m}}V_k\Sigma_kB_k),\tag{4.34}$$

where $Z_k = U_k B_k$ and B_k is the matrix of the Ritz vectors computed in Line 5 of Algorithm 1. (If the refined Ritz vectors are used, then B_k is replaced with W_k that is computed in Line 12 of Algorithm 2.)

Remark 4.1. Let us compare this **q** and the corresponding $\mathbf{q}_0 = [(\overline{\mathbb{V}_{k,m}} \circ (Z_k^* \mathbf{X}_m))\mathbf{e}]$ in relation (3.20) (with $\ell = k$ and $W = \mathbb{I}_m$). Since $Z_k^* \mathbf{X}_m = B_k^* \Sigma_k V_k^*$, by [18, Lemma 5.1.3], we have

$$\mathbf{q}_{\mathbf{0}} = [(\overline{\mathbb{V}_{k,m}} \circ (B_k^* \Sigma_k V_k^*))\mathbf{e}] = \operatorname{diag}(\overline{\mathbb{V}_{k,m}} (B_k^* \Sigma_k V_k^*)^T = \operatorname{diag}(\overline{\mathbb{V}_{k,m}} V_k \Sigma_k \overline{B_k}) = \mathbf{q}_{\mathbf{0}}$$

The constrained problem (2.9) $(\Omega^2(\vec{\alpha}) \longrightarrow \min, E^T \vec{\alpha} = 0)$, where the $k \times s$ matrix E is described in §2.2.1) is in the polishing phase of DMDSP solved via the variation of the Lagrangian (see [19, Appendix C]), which yields the augmented $(k + s) \times (k + s)$ system of equations

$$\begin{pmatrix} \mathbb{P} & E \\ E^T & \mathbf{0} \end{pmatrix} \begin{pmatrix} \vec{\alpha} \\ \nu \end{pmatrix} = \begin{pmatrix} \mathbf{q} \\ \mathbf{0} \end{pmatrix}$$
(4.35)

for the amplitudes $\vec{\alpha}$ and the vector of Lagrange multipliers ν . The amplitudes are then explicitly expressed as (see [19, Appendix C] and [20])

$$\vec{\boldsymbol{\alpha}} = \begin{pmatrix} \mathbb{I}_k & \mathbf{0} \end{pmatrix} \begin{bmatrix} \begin{pmatrix} \mathbb{P} & E \\ E^T & \mathbf{0} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{q} \\ \mathbf{0} \end{pmatrix} \end{bmatrix}.$$
(4.36)

More efficient procedure is to write $\vec{\alpha}$ as $\vec{\alpha} = E_{\perp}\vec{\beta}$, where $E_{\perp} \in \mathbb{C}^{k \times (k-s)}$ contains the remaining columns of the identity (complementary to the columns of E), and $\vec{\beta} \in \mathbb{C}^{k-s}$ is a new (unconstrained) variable. The objective function now reads

$$\Omega^{2}(\vec{\boldsymbol{\alpha}})_{E^{T}\vec{\boldsymbol{\alpha}}=\mathbf{0}} \equiv \Omega^{2}(\vec{\boldsymbol{\beta}}) = \vec{\boldsymbol{\beta}}^{*}(E_{\perp}^{*}\mathbb{P}E_{\perp})\vec{\boldsymbol{\beta}} - (E_{\perp}^{*}\mathbf{q})^{*}\vec{\boldsymbol{\beta}} - \vec{\boldsymbol{\beta}}^{*}(E_{\perp}^{*}\mathbf{q}) + \|\boldsymbol{\Sigma}_{k}\|_{F}^{2},$$
(4.37)

aimdyn: 2018 Nov. 001v1

AIMDYN INC.

where $E_{\perp}^* \mathbb{P} E_{\perp}$ is a $(k-s) \times (k-s)$ main submatrix of \mathbb{P} , and the explicit solution is

$$\vec{\boldsymbol{\alpha}} = E_{\perp} \left[(E_{\perp}^* \mathbb{P} E_{\perp})^{-1} (E_{\perp}^* \mathbf{q}) \right].$$
(4.38)

Schematically, the coefficient matrices of the linear systems in (4.36) and (4.38) can be illustrated as follows:

Remark 4.2. To appreciate the difference between (4.36) and (4.38), consider for example k = m = 1200, and only $\ell = 30$ modes. The augmented system in (4.36) is of dimension 1200 + 1170 = 2370, while the same result is obtained from the system (4.38) of dimension 30. Given the cubic complexity of the solution of linear systems, the speedup of (4.38) over (4.36) is considerable. It should be noted that (4.35) is a particular case of the saddle point problem that is solved by specially tailored methods; for an overview we refer to [2]; see also e.g. [15]. However, our proposed alternative (4.38) is simpler and much more efficient.

Remark 4.3. 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 $E = Q\begin{pmatrix} R \\ 0 \end{pmatrix} \equiv \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix}$ be the full QR factorization. It is then clear that we can take $E_{\perp} = Q_2$, and the above procedure applies verbatim.

5. Sensitivity analysis of normal equations method

Although elegant and efficient, the formula (3.19) has a drawback, typical for normal equation based LS solution – it squares the condition number $(\kappa_2(R^*R) = \kappa_2(R)^2 \text{ and } \kappa_2(\overline{\mathbb{V}_{\ell,m}}\mathbf{W}^2\mathbb{V}_{\ell,m}^*) = \kappa_2(\mathbb{V}_{\ell,m}\mathbf{W})^2)$ 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 [18, 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 possesses the Cholesky factor.

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 to [8] for details on computing the Cholesky factorization in floating point arithmetic.

Hence, the formula (3.19) 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 is contrived, to illustrate the above discussion.

Example 5.1. (All computations done in Matlab R2015.a, with the double precision roundoff unit $\varepsilon = eps=2.220446049250313e-16$.) Let $\ell = 3$, m = 4, $\xi = \sqrt{\varepsilon}$, $\lambda_1 = \xi$, $\lambda_2 = 2\xi$, $\lambda_3 = 0.2$, so that the Vandermonde section $\mathbb{V}_{\ell,m}$ equals

aimdyn:2018Nov.001v1

and let the triangular factor R be

1.

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

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 entry-wise nonnegative, both $\mathbb{V}_{\ell,m}\mathbb{V}^*_{\ell,m}$ and R^*R are computed to nearly full machine precision[‡]; the same holds for $C = (R^*R) \circ (\overline{\mathbb{V}_{\ell,m}}\mathbb{V}^*_{\ell,m})$. Although all these 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')	>> chol((R'*R).*(Vlm*Vlm'))
Error using chol	Error using chol
Matrix must be positive definite.	Matrix must be positive definite.

>> chol(R'*R)
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 ($\mathbf{W} = \mathbb{I}$) of the formula (3.19) 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 sufficiently accurate result.

5.1. Condition number estimates

Based on [8], we know that floating point Cholesky factorization $C = LL^*$ (*L* lower triangular with positive diagonal) of *C* is feasible if[§] the matrix $C_s = (c_{ij}/\sqrt{c_{ii}c_{jj}})_{i,j=1}^{\ell}$ is well conditioned. 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

$$\max_{i,j} \frac{|\delta c_{ij}|}{\sqrt{c_{ii}c_{jj}}} \le O(\ell)\varepsilon.$$
(5.40)

[‡]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 $O(\varepsilon)$ relative error – practically the best one can hope for.

[§]Actually, if no additional structure is assumed, here we have an "if and only if". This means that, in 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.

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

$$\widetilde{L}\widetilde{L}^* = C + \delta C = D_C(C_s + D_C^{-1}\delta C D_C^{-1})D_C, \quad C_s = D_C^{-1}C D_C^{-1}, \quad (5.41)$$

where the entries of $D_C^{-1} \delta C D_C^{-1}$ are estimated by (5.40). Note that C_s has unit diagonal and that all its off-diagonal entries are in absolute value below one.

By [12, 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

$$\|\Gamma\|_2 \le O(\ell \log_2 \ell) \varepsilon \|C_s^{-1}\|_2 \le O(\ell \log_2 \ell) \varepsilon \kappa_2(C_s).$$
(5.42)

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 \tilde{x} satisfies (see [8, Theorem 2.1])

$$\frac{\|D_C(\widetilde{x} - C^{-1}b)\|_2}{\|D_C\widetilde{x}\|_2} \le g(\ell)\varepsilon\kappa_2(C_s),\tag{5.43}$$

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

$$\frac{\widetilde{x}_i - (C^{-1}b)_i|}{|\widetilde{x}_i|} \le \underbrace{\left[\frac{\|D_C \widetilde{x}\|_2}{\sqrt{c_{ii}}|\widetilde{x}_i|}\right]}_{\ge 1} g(\ell) \varepsilon \kappa_2(C_s).$$
(5.44)

For further discussion we refer to [8].

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 [32]:

Theorem 5.2. Let $H \in \mathbb{C}^{n \times n}$ be positive definite Hermitian matrix, $D_H = \operatorname{diag}(\sqrt{H_{ii}})$ and $H_s = D_H^{-1}HD_H^{-1}$. Then $\kappa_2(H_s) \leq n \min_{D=diag} \kappa_2(DHD)$.

Remark 5.3. Note that $\kappa_2(C_s)$ can be at most ℓ times larger, and that it can be much smaller than $\kappa_2(C)$.

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

$$\max(\lambda_{\min}(A_s), \lambda_{\min}(B_s)) \le \lambda_i(C_s) \le \min(\lambda_{\max}(A_s), \lambda_{\max}(B_s)).$$
(5.45)

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 [18, Theorem 5.3.4], which states the following general property of the Hadamard product

$$\lambda_{\min}(A_s)\lambda_{\min}(B_s) \le \min_i(A_s)_{ii}\lambda_{\min}(B_s) \le \lambda_i(C_s) \le \max_i(A_s)_{ii}\lambda_{\max}(B_s) \le \lambda_{\max}(A_s)\lambda_{\max}(B_s),$$

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 5.5. 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 products 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 numerical Cholesky factor.

Example 5.6. With the notation of Example 5.1, we use the same $\mathbb{V}_{\ell,m}$ but change the definition of R to

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

>> chol(Vlm*Vlm')		>> cho	l(R'*R)		
Error using chol		Error	using ch	ol	
Matrix must be positi	ive definite.	Matrix	must be	positive	definite.
>> TO = chol((R'*R).*(V) TO =	lm*Vlm'))				
1.0000000000000000e+00	1.0000000000000000000000000000000000000)e+00	1.00000	0000298023	32e+00
0	1.490116119384765	je-08	1.99999	9988079071	0e-01
0		0	4.07921	1414969506	32e-02

The condition number of T0 is estimated to $8.2 \cdot 10^8$, and its inverse is used in backward and forward substitutions when solving the normal equations, see (3.19).

Note, however, that in this case $\kappa_2(C_s)\varepsilon > 2$, and perturbation theory [8], [12] provides no guarantee for the accuracy of the computed Cholesky factor. (In fact, in Example 5.11 below, we argue that T0 is a pretty bad approximation.)

Corollary 5.7. 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 5.4, where we have $A_s = R_c^*R_c$, $B_s = (\mathbb{V}_{\ell m}\mathbf{W})_r(\mathbb{V}_{\ell,m}\mathbf{W})_r^*$.

Example 5.8. According to Theorem 5.2 and Remark 5.3, $\kappa_2(C_s)$ is potentially much smaller than $\kappa_2(C)$, and, by the discussion in §5.1, 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 $\kappa_2(C_s)$.

This can be illustrated e.g. with 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 [31], [29] for more detailed description of the experimental setup and numerical simulations.[¶]

The data^{\parallel} 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 ($\mathbf{f}_1, \ldots, \mathbf{f}_{n_t}$), 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:

$$\kappa_2(C) > 10^{87} \gg \kappa_2(C_s) \approx 8.504071414461372e+01.$$

Using (5.42), we can conclude that the Cholesky factor of C can be computed to high accuracy, despite the fact that $\kappa_2(C) \gg 1/\varepsilon$. 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, see Figure 1.



Figure 1: The diagonal entries of C vary from 1.682014829394577e+00 to 4.574179501782303e+87. The absolute values of the

computed Ritz values of the computed Ritz values go (roughly) from 0.63 to 1.08.

Following the discussion in Remark 5.5 and Examples 5.1 and 5.6, the bound of Corollary 5.7 is in this example (luckily) an extreme overestimate because $\kappa_2(R_c)^2 = 3.052938825507679e+13$, $\kappa_2((\mathbb{V}_{\ell,m})_r)^2 = 9.620903374928321e+14$. However, Example 5.1 illustrates the danger of potential failure of the formula (3.19).

Remark 5.9. In the DMDSP, the augmented Lagrangian method works with $C + \rho I$, with suitable parameter $\rho > 0$. The Cholesky factor of $C + \rho I$ is used to repeatedly solve linear systems of equations. With any moderate value of ρ , the diagonal of C in Example 5.8 will behave similarly as shown in Figure 1.

The condition number $\kappa_2(C_s)$ from Corollary 5.7 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.

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

^{||}We used this dataset in [9] for testing a new algorithm for representing the snapshots using thr Koopman modes.

Proposition 5.10. 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,

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

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

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

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

$$|\delta a_{ij}b_{ij}| \le O(m_x \varepsilon) \sqrt{a_{ii}a_{jj}} \sqrt{b_{ii}b_{jj}} = O(m_x \varepsilon) \sqrt{c_{ii}c_{jj}}$$

The remaining error terms are bounded in the same way.

Hence, in a concrete finite precision computation, we will work with $\tilde{C} = C + \delta_0 C$, so that the backward error (5.40, 5.41) applies to \tilde{C} . The scaled condition number $\kappa_2(\tilde{C}_s)$ ($\tilde{C}_s = (\tilde{c}_{ij}/\sqrt{\tilde{c}_{ii}\tilde{c}_{jj}})$) will be moderate if $\kappa_2(C_s)$ is moderate, so all conclusions apply to \tilde{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.

5.1.1. A decision tree. Equipped with the results from §5.1, 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 the upper triangular factor R in the QR factorization $Z_{\ell} = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$ 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 5.7, 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 $Z_{\ell} = U_k \tilde{B}_{\ell}$ (see Remark 3.1) and since R can be computed directly from the $O(k\ell^2)$ QR factorization of the $k \times \ell$ matrix \tilde{B}_{ℓ} , 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 $\kappa_2(R_c)^2 \ll 1/\varepsilon$, then one can safely use the normal equation solution (3.19).

If an estimate for $\kappa_2(R_c)^2$ is not available (e.g. R not computed) or if one concludes that $\kappa_2(R_c)^2 \gtrsim 1/\varepsilon$, then one goes on and computes C and an estimate for $\kappa_2(C_s)$, which is then tested against the threshold $1/\varepsilon$. We can organize this in a decision tree as in Algorithm 3. It is designed for the case when the dimensions are sufficiently large and the efficiency of the formula (3.19) is desirable, but not at any price – it is deployed only if it can warrants certain level of accuracy.

Algorithm 3 Decision tree for selecting a solution method for (3.21)

Input:

- R, Λ, ℓ, m
- Tolerance level $tol \in (1, 1/\varepsilon)$ for acceptable condition number estimate.

1: if R available and $\kappa_2(R_c)^2 \leq tol$ then Solve (3.21) by normal equations, $\vec{\alpha} = [(R^*R) \circ (\overline{\mathbb{V}_{\ell,m} \mathbf{W} \mathbb{V}_{\ell,m}^*})]^{-1} [(\overline{\mathbb{V}_{\ell,m} \mathbf{W}} \circ (Z_{\ell}^* \mathbf{X}_m \mathbf{W}))\mathbf{e}]$ 2: 3: else Compute $C = (Z_{\ell}^* Z_{\ell}) \circ (\overline{\mathbb{V}_{\ell,m} \mathbf{W}^2 \mathbb{V}_{\ell,m}^*})$ and estimate $\kappa_2(C_s)$ 4: if $\kappa_2(C_s) \leq tol$ then 5: Solve (3.21) by normal equations, $\vec{\boldsymbol{\alpha}} = [(Z_{\ell}^* Z_{\ell}) \circ (\overline{\mathbb{V}_{\ell,m} \mathbf{W}^2 \mathbb{V}_{\ell,m}^*})]^{-1} [(\overline{\mathbb{V}_{\ell,m} \mathbf{W}} \circ (Z_{\ell}^* \mathbf{X}_m \mathbf{W}))\mathbf{e}]$ 6: 7: else Solve (3.21) using QR factorization based solver, without squaring the condition number. 8: 9: end if 10: end if

In the next section, we discuss Line 8 in Algorithm 3.

Example 5.11. (Continuation of Example 5.6) From Proposition 3.5, 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, then

>> [Q,T] = qr(S,0) ; T	' = diag(sign(diag(T)))*T	
T =		
1.000000000000000e+00	1.0000000000000000e+00	1.00000002980232e+00
0	2.107342425544703e-08	1.414213575017149e-01
0	0	1.471869344809795e-01

Note that the difference between (theoretically identical matrices) T and T0, computed in Example 5.6, is significant. Since the condition number of S is $\kappa_2(S) \approx 1.6 \cdot 10^8$, we know that T is provably more accurate than T0.

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 (3.19))

```
>> [T0\(T0'\ones(3,1)) T\(T'\ones(3,1))]
ans =
2.703847231205138e+01 2.00000063280420e+00
-2.603847410305737e+01 -1.00000063280429e+00
1.791005994071227e-06 9.429518722743198e-15
```

Although small dimensional and entirely synthetic, Examples 5.6 and 5.11 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 the future work.

6. QR factorization based solvers

A numerically more robust procedure for solving the LS problem (3.21) 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 (3.21) 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 (3.19) representing (3.22) is avoided if we use the tall QR factorization $S = Q_S R_S$; then^{**}

$$\vec{\boldsymbol{\alpha}} = R_S^{-1}(Q_S^*\vec{\mathbf{g}}). \tag{6.46}$$

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 reflectors or Givens rotations) so that $Q_S^* S = R_S$, it suffices to apply the same matrices to $\vec{\mathbf{g}}$ as it was the $(\ell + 1)$ st column of S.

6.1. Corrected semi-normal approach

If we want to avoid the additional cost for computing $Q_S^* \vec{\mathbf{g}}$, in particular in view of the efficient formula (3.25) 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 (3.19), to avoid the problems illustrated in Examples 5.1 and 5.6. Then, the seminormal solution [3] is

$$\vec{\alpha} = R_S^{-1}(R_S^{-*}(S^*\vec{\mathbf{g}})). \tag{6.47}$$

This method only partially alleviates the problem of ill-conditioning. It computes 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 much better than the normal equations solver.

However, if supplemented by a correction step, the seminormal solution becomes nearly as good as (sometimes even better than) the QR factorization based solver. The correction procedure, done in the same working precision, first computes the residual

$$r = \vec{\mathbf{g}} - S\vec{\alpha} \tag{6.48}$$

and then computes the correction $\delta \vec{\alpha}$ as

$$\delta \vec{\alpha} = R_S^{-1}(R_S^{-*}(S^*r)) \tag{6.49}$$

and the corrected solution $\vec{\alpha}_* = \vec{\alpha} + \delta \vec{\alpha}$. This process can be repeated. For an error analysis in favor of this scheme see [3].

In an efficient software implementation, we use the structure of S and organize the data to increase locality, i.e. prefer matrix multiplications. A prototype of the computational scheme is given in Algorithm 4. (In lines 2. and 3. we give hints to an implementer, regarding high performance implementation based on the libraries BLAS and LAPACK.)

^{**}In this section we use the notation from $\S3.2$.

Algorithm 4 Corrected semi-normal solution of (3.21)

Input: R, Λ , G, S (Here applies the notation from §3.2.) **Output:** Corrected solution $\vec{\alpha}_*$

1: Compute the triangular factor R_S in the QR factorization of S. 2: $S^*\vec{\mathbf{g}} = [(\overline{\mathbb{V}_{\ell,m}} \circ (R^*G))\mathbf{e}]$ {Use xTRMM from BLAS 3.} 3: $\vec{\alpha} = R_S^{-1}(R_S^{-*}(S^*\vec{\mathbf{g}}))$ {Use xTRSM or xTRTRS or xTRSV from LAPACK.} 4: $r_{\Box} = G - R(\vec{\alpha} \quad \Lambda \vec{\alpha} \quad \Lambda^2 \vec{\alpha} \quad \dots \quad \Lambda^{m-1} \vec{\alpha}) \equiv G - R \text{diag}(\vec{\alpha}) \mathbb{V}_{\ell,m}$ 5: $S^*r = [(\overline{\mathbb{V}_{\ell,m}} \circ (R^*r_{\Box}))\mathbf{e}]$ {Use xTRMM from BLAS 3.} 6: $\delta \vec{\alpha} = R_S^{-1}(R_S^{-*}(S^*r))$ {Use xTRSM or xTRTRS or xTRSV from LAPACK.} 7: $\vec{\alpha}_* = \vec{\alpha} + \delta \vec{\alpha}$

It is certainly desirable to have an efficient QR factorization based LS solver 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 using (6.46) or the corrected semi-normal approach as in Algorithm 4.

The recursive structure of the block rows of S can indeed be exploited to compute its QR factorization at the cost of $O(\ell^3 \log_2 m)$. In the next section we provide the details.

6.2. 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 §6.2.1 for the simplest case of $m = 2^p$, with the details of the kernel routine given in §6.2.2. Section 6.2.3 provides the scheme with an arbitrary number of snapshots m.

6.2.1. The case $m = 2^p$. The basic idea is illustrated in (6.50) for m = 16, and it is, in a sense, analogous to the FFT divide and conquer scheme.

Algorithm 5 Recursive QR factorization of S in (3.21) for $m = 2^p$

Input: Upper triangular $R \in \mathbb{C}^{\ell \times \ell}$; diagonal $\Lambda \in \mathbb{C}^{\ell \times \ell}$; number of snapshots $m = 2^p$ **Output:** Upper triangular QR factor $R_S = T_p$ of $S \in \mathbb{C}^{2^p \ell \times \ell}$ in (3.21)

T_4	$\leftarrow T_3$	$\leftarrow T_2$	$\leftarrow T_1$	$\leftarrow R \mathbf{\Lambda}^0$	
0	0	0	0	$ \leftarrow R \mathbf{\Lambda}^1 $	
0	0	0	$\leftarrow T_1 \mathbf{\Lambda}^2$	$R\Lambda^2$	
0	0	0	0	$R\mathbf{\Lambda}^3$	
0	0	$\leftarrow T_2 \Lambda^4$	$T_1 \Lambda^4$	$R\Lambda^4$	
0	0	0	0	$R \mathbf{\Lambda}^5$	$1: T_0 = R$
0	0	0	$T_1 \Lambda^6$	$R \Lambda^6$	2: for i = 1:p do
0	0	0	0	$R\mathbf{\Lambda}^7$	$\left\ \cdot \right\ _{2} = \left(\overline{T_{i}} \right)^{-1} \left(\overline{T_{i-1}} \right)^{-1}$
0	$\leftarrow T_3 \Lambda^8$	$T_2 \Lambda^8$	$T_1 \Lambda^8$	$R\Lambda^8$	$ \begin{vmatrix} 3 \\ 0 \end{vmatrix} = \operatorname{qr}\left(\begin{pmatrix} -1 \\ T_{i-1} \Lambda^{2^i} \end{vmatrix} \right) $
0	0	0	0	$R \Lambda^9$	4 : end for
0	0	0	$T_1 \mathbf{\Lambda}^{10}$	$R \mathbf{\Lambda}^{10}$	
0	0	0	0	$R \mathbf{\Lambda}^{11}$	
0	0	$T_2 \mathbf{\Lambda}^{12}$	$T_1 \mathbf{\Lambda}^{12}$	$R \Lambda^{12}$	
0	0	0	0	$R \mathbf{\Lambda}^{13}$	
0	0	0	$T_1 \mathbf{\Lambda}^{14}$	$R \Lambda^{14}$	
0	0	0	0	$R \Lambda^{15}$	
L					

{For a simple Matlab implementation see §8.1.}

Note that the core operation in the *i*th step of the above scheme is the QR factorization of the $2\ell \times \ell$ structured matrix

$$\begin{pmatrix}
T_{i-1} \\
T_{i-1}\boldsymbol{\Lambda}^{2^{i-1}}
\end{pmatrix} = \begin{pmatrix}
* * * * * \\
0 * * * \\
0 & 0 & * \\
0 & 0 & * \\
0 & 0 & * \\
0 & 0 & * \\
0 & 0 & * \\
0 & 0 & *
\end{pmatrix}, \text{ i.e. implicitly of } \begin{pmatrix}
T_{i-1} \\
\mathbf{0}_{(2^{i-1}-1)\ell,\ell} \\
T_{i-1}\boldsymbol{\Lambda}^{2^{i-1}} \\
\mathbf{0}_{(2^{i-1}-1)\ell,\ell}
\end{pmatrix}$$
(6.51)

and, thus, it can be computed more efficiently than in the general case of a $2\ell \times \ell$ dense matrix.

Remark 6.1. If we want to solve the LS problem using (6.46), then in Algorithm 5 we need to include updating of $\vec{\mathbf{g}}$ for each $i = 1, \ldots, p$. The algorithm then reads as follows:

$$\begin{split} T_0 &= R\\ \text{for } i &= 1:p \text{ do}\\ \begin{pmatrix} \boxed{T_{i-1}}\\ T_{i-1}\Lambda^{2^{i-1}} \end{pmatrix} &= Q_i \boxed{T_i} \text{ {Thin QR factorization. }} Q_i \text{ is } 2l \times l. \text{ }\\ \text{for } j &= 0:2^i:2^p - 2^i \text{ do}\\ \vec{\mathbf{g}}(j\ell+1:(j+1)\ell) &= Q_i^* \begin{pmatrix} \vec{\mathbf{g}}(j\ell+1:(j+1)\ell)\\ \vec{\mathbf{g}}((j+2^{i-1})\ell+1:(j+2^{i-1}+1)\ell) \end{pmatrix}\\ \text{end for}\\ \text{end for}\\ \text{end for} \end{split}$$

aimdyn: 2018 Nov. 001v1

(6.50)

Note that after this sequence of updates, the array $\vec{\mathbf{g}}(1:\ell)$ contains the vector $Q_S^* \vec{\mathbf{g}}$ from (6.46); the least squares solution is then computed as the solution of the triangular system $T_p \vec{\alpha} = \vec{\mathbf{g}}(1:\ell)$.

6.2.2. Details of the QR factorization of (6.51). The seemingly simple task to compute the QR factorization of two stacked triangular matrices (6.51) is an excellent case study for turning an algorithm into an efficient software implementation.

Algorithm 6 illustrates a straightforward scheme to annihilate the $\ell(\ell+1)/2$ entries in the lower block in (6.51); 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.

Algorithm 6 Givens QR factorization of the type (6.51);

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

Output: The upper triangular factor in the QR factorization of $\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & * \\ 0 & 0 & * \\ 0 & 0 & * \\ 0 & 0 & * \\ 0 & 0 & * \\ 0 & 0 & * \end{pmatrix}$ 1: $Q = \mathbb{I}_{2\ell}$

2: for $j = 1 : \ell$ do 3: for i = 1 : j do 4: Compute 2×2 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

The nested loops in Algorithm 6 access the matrix B column-wise. Changing the loops into

for $i = 1 : \ell$ {for $j = i : \ell$ {4 : ...; 5 : ... end for } 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 6 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 7. Algorithm 7 Householder+Givens QR factorization of the type (6.51);

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

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

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

The operations in Algorithm 7 are illustrated in the scheme (6.52).

$$\begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0$$

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 (6.52) each $*, \star, \bullet, *, 0$ 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.

Algorithm 8 Block-oriented Householder+Givens QR factorization of the type (6.51);

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 $\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} 0 & * & * & * \\ 0 & 0 & 0 & * \\ 0 & 0 & * & * & \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 &$

1: $\wp = \lfloor \ell/b \rfloor$; $b' = \ell - (\wp - 1)b$. Introduce block partitions in A and B.

- 2: for $j = 1 : \wp$ do
- if j > 1 then 3:

Compute the QR factorization $B[1:j,j] = \mathcal{H}\begin{pmatrix} \mathcal{R} \\ 0 \end{pmatrix}$ of B[1:j,j] as follows: 4:

- The upper triangular factor \mathcal{R} overwrites the leading submatrix of B[1, j]. 4.1:
- 4.2: Write the accumulated product of Householder reflectors \mathcal{H} in the compact form $\mathcal{H} = \mathbb{I} - \mathcal{WTW}^*$. (See Remark 6.2.)
- if $j < \wp$ then 5:

Update B:
$$B[1:j, j+1:\wp] = B[1:j, j+1:\wp] - \mathcal{WT}^*(\mathcal{W}^*B[1:j, j+1:\wp])$$

end if 7:

6:

end if 8:

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 $\widehat{\mathcal{R}}$ 9: overwrites A[j, j]. Use e.g. Algorithm 7.

if $j < \wp$ then 10:

11: Update A and B:
$$\begin{pmatrix} A[j, j+1:\wp]\\ B[1, j+1:\wp] \end{pmatrix} := \mathcal{Q}^* \begin{pmatrix} A[j, j+1:\wp]\\ B[1, j+1:\wp] \end{pmatrix};$$

- end if 12:
- end for 13:

Remark 6.2. 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 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 of $\mathcal{H} = \mathcal{H}_1 \mathcal{H}_2 \cdots \mathcal{H}_b$ is then computed recursively as: $\mathcal{W}_1 = (w_1)$, $\mathcal{T}_1 = \beta_1; \, \mathcal{W}_j = (\mathcal{W}_{j-1} \, w_j),$

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

For more details we refer to [28] and for a guidelines for an efficient implementation we suggest studying the structure of the subroutine xGEQRF in LAPACK – essentially, the computation in Line 4 and Line 6 is already contained as a part of xGEQRF.

6.2.3. The case of general *m*. We now generalize the recursive scheme of the Algorithm 5 to general dimension $m \neq 2^p$. First, introduce simple notation: S will be considered as block-row partitioned with the *i*th block $S_{[i]} = R\Lambda^{i-1}$. The submatrix of S consisting of consecutive blocks from the i_1 th to the i_2 th 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 (6.50) contains among its intermediate results the QR factorization also of e.g. $S_{[1:2]}$ (the factor is T_1), of $S_{[1:4]}$ (the factor is T_2), of $S_{[1:8]}$ (the factor is T_3), 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 binary number $m \equiv \mathfrak{b} = (\mathfrak{b}_{\lfloor \log_2 m \rfloor}, \dots, \mathfrak{b}_1, \mathfrak{b}_0)_2$, i.e.

$$m = \sum_{i=0}^{\lfloor \log_2 m \rfloor} \mathfrak{b}_i 2^i \equiv \sum_{j=1}^{j^*} 2^{i_j}, \ \lfloor \log_2 m \rfloor = i_{j^*} > i_{j^*-1} > \dots > i_2 > i_1 \ge 0,$$
(6.53)

and introduce the block partition of S as follows:

$$S^{T} = \left(S^{T}_{[1:2^{i_{j^{*}}}]} \quad S^{T}_{[2^{i_{j^{*}}}+1:2^{i_{j^{*}}}+2^{i_{j^{*}}-1}]} \quad \dots \quad S^{T}_{[2^{i_{j^{*}}}+\dots+2^{i_{2}}+1:2^{i_{j^{*}}}+\dots+2^{i_{1}}]}\right).$$
(6.54)

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 5, and the triangular factor of each of the subsequent blocks in (6.54) 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 the first reduction step, that 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

$$S_{[1:2^{i_1}]}, S_{[2^{i_1}+1:2^{i_1}+2^{i_2}]}, S_{[2^{i_1}+2^{i_2}+1:2^{i_1}+2^{i_2}+2^{i_3}]}, \dots, S_{[2^{i_1}+2^{i_2}+\dots+2^{i_{j^*}-1}+1:2^{i_1}+2^{i_2}+2^{i_3}+\dots+2^{i_{j^*}}]}$$
(6.55)

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

$$S_{[1:2^{i_1}]}, S_{[1:2^{i_2}]}, S_{[1:2^{i_3}]}, \dots, S_{[1:2^{i_j^*}]}$$

$$(6.56)$$

with, respectively,

$$\Lambda^{0}, \Lambda^{2^{i_{1}}}, \Lambda^{2^{i_{1}}+2^{i_{2}}}, \dots, \Lambda^{2^{i_{1}}+2^{i_{2}}+\dots+2^{i_{j}*-1}}$$
(6.57)

and are built in into the global triangular factor by a sequence of updates. The procedure is summarized in Algorithm 9.

Algorithm 9 Recursive QR factorization of S in (3.21)

Input: Upper triangular $R \in \mathbb{C}^{\ell \times \ell}$; diagonal $\Lambda \in \mathbb{C}^{\ell \times \ell}$; number of snapshots m**Output:** Upper triangular QR factor $R_S = \mathbb{T}_{i-1}$ of S in (3.21) 1: Compute the binary representation (6.53) of m: $m \equiv \mathfrak{b} = (\mathfrak{b}_{|\log_2 m|}, \dots, \mathfrak{b}_1, \mathfrak{b}_0)_2$ 2: Let $\lfloor \log_2 m \rfloor = i_{j^*} > i_{j^*-1} > \dots > i_2 > i_1 \ge 0$ be as in (6.53) 3: $T_0 = R$ 4: if $i_1 = 0$ then 5: $\mathbb{T}_1 = T_0; \ j = 2; \ \wp = 1$ 6: else $\mathbb{T}_0 = []; j = 1; \wp = 0$ 7:8: end if 9: for $k = 1 : i_{i^*}$ do $\frac{\overline{T_k}}{\mathbf{0}} = \operatorname{qr}\left(\begin{pmatrix} \overline{T_{k-1}} \\ T_{k-1} \mathbf{\Lambda}^{2^{k-1}} \end{pmatrix} \right)$ {Local triangular factor. Use algorithms from §6.2.2.} 10: if $k = i_i$ then 11: if $\mathbb{T}_{j-1} \neq []$ then 12: $\begin{pmatrix} \mathbb{T}_j \\ \mathbf{0} \end{pmatrix} = \operatorname{qr}\left(\begin{pmatrix} \mathbb{T}_{j-1} \\ T_k \mathbf{\Lambda}^{\wp} \end{pmatrix}\right) \{\text{Global triangular factor. Use algorithms from } \S6.2.2.\}$ 13:else 14: $\overline{\mathbb{T}}_{j}$ $| = |T_k|$ 15:16: $j := j + 1; \ \wp := \wp + 2^k$ 17:end if 18:19: end for 20: {For a simple Matlab implementation see $\S8.1.$ }

Remark 6.3. The above scheme 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 implementations on a particular hardware.

6.3. Numerical stability

For an efficient implementation, the computational scheme e.g. in Algorithm 6 will be modified to enhance spatial and temporal locality of data, e.g. using tiling or blocking techniques as in Algorithm 8, 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 column-wise small, i.e.

$$\|\delta S(:,j)\|_{2} \le \eta \|S(:,j)\|_{2}, \quad j = 1, \dots, \ell; \quad \eta \le f(\ell,m)\varepsilon,$$
(6.58)

where $f(\ell, m)$ is modest polynomial that depends on the details of a particular implementation.^{††}

This follows from the simple fact that in Algorithms 5 and 9 (using Algorithm 6 with any ordering of Givens rotations, or Algorithms 7 and 8 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

^{††}Note that (6.58) is much stronger statement than the usually used backward error bound $\|\delta S\|_F \leq g(\ell, m)\varepsilon \|S\|_F$.

column of S has backward error that is small relative to that same column; in this way even the tiniest columns are preserved, independent of the remaining possibly much larger ones.

Due to (6.58), the corresponding 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 so that $||S_c(:,j)||_2 = 1$ for all j. More precisely, if $\tilde{R}_S = R_S + \delta R_S$ 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 [11, §6.],

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

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

Corollary 6.4. With the notation of Corollary 5.7, it holds that

$$\kappa_2(S_c) = \sqrt{\kappa_2(C_s)} \le \min(\kappa_2(R_c), \kappa_2((\mathbb{V}_{\ell,m})_r)) \le \sqrt{\ell} \min(\min_{D=\text{diag}} \kappa_2(RD), \min_{D=\text{diag}} \kappa_2(D\mathbb{V}_{\ell,m})).$$
(6.59)

Proof. It follows from Proposition 3.5 that $C_s = S_c^* S_c$; hence $\kappa_2(S_c) = \sqrt{\kappa_2(C_s)}$, and the first inequality follows from Corollary 5.7. The second inequality follows from the classical result [32].

This can be used to estimate the accuracy of Line 8 in Algorithm 3: if $\min(\kappa_2(R_c), \kappa_2((\mathbb{V}_{\ell,m})_r))$ in (6.59) is below an appropriate threshold, certain level of accuracy can be guaranteed a priori.

6.3.1. Importance of pivoting. For better accuracy of the solutions of triangular equations (forward and backward substitutions in (6.47)), 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.

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

$$SP = Q_S R_S = \begin{pmatrix} R_{[11]} & R_{[12]} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} = Q_{S,r} \begin{pmatrix} R_{[11]} & R_{[12]} \end{pmatrix}, \quad Q_{S,r} = Q_S(:, 1:r).$$
(6.60)

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

$$\begin{split} \|S\vec{\alpha} - \vec{\mathbf{g}}\|_{2}^{2} &= \|Q_{S,r} \left(R_{[11]} \quad R_{[12]}\right) P^{T}\vec{\alpha} - Q_{S,r}Q_{S,r}^{*}\vec{\mathbf{g}} - (\mathbb{I} - Q_{S,r}Q_{S,r}^{*})\vec{\mathbf{g}}\|_{2}^{2} \\ &= \|\left(R_{[11]} \quad R_{[12]}\right) P^{T}\vec{\alpha} - Q_{S,r}^{*}\vec{\mathbf{g}}\|_{2}^{2} + \|(\mathbb{I} - Q_{S,r}Q_{S,r}^{*})\vec{\mathbf{g}}\|_{2}^{2} \to \min, \end{split}$$

and the problem reduces to

$$\| \begin{pmatrix} R_{[11]} & R_{[12]} \end{pmatrix} P^T \vec{\alpha} - Q^*_{S,r} \vec{\mathbf{g}} \|_2 \longrightarrow \min_{\vec{\alpha}}.$$
(6.61)

One particular vector in the solution manifold is

$$\vec{\boldsymbol{\alpha}}_{\diamond} = P \begin{pmatrix} R_{[11]}^{-1} Q_{S,r}^* \vec{\mathbf{g}} \\ \mathbf{0} \end{pmatrix}.$$
(6.62)

Note that $\vec{\alpha}_{\diamond}$ has at least $\ell - r_S$ zero entries, and that it is different from the shortest solution $\vec{\alpha}_* = S^{\dagger} \vec{\mathbf{g}}$. Hence, if the additional criterion is sparsity, the rank deficient least squares problem is best solved by (6.60, 6.62). In the full rank case $\vec{\alpha}_{\diamond} = \vec{\alpha}_*$.

Remark 6.5. Note that (6.61) is under-determined and that we can add sparsity constraint analogously to (2.7), which is accordance with Remark 2.1. The sparsity of the explicit solution (6.62) is a good starting point for a quest for sparse solution.

Remark 6.6. The Matlab backslash operator (e.g. $alpha=S\backslash g$) solves least squares problems using (6.60, 6.62); the same procedure is used in xGELSX and xGELSY least squares solvers in LAPACK. The pivoted QR factorization is computed by the LAPACK subroutine xGEQP3 (or xGEQPF). If sparsity is a desirable property of the solution in rank deficient case, then (6.60, 6.62) should be preferred over using the Moore–Penrose pseudoinverse (e.g. alpha=pinv(S)*g, in Matlab).

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

$$SP = Q_S R_S = \begin{pmatrix} R_{[11]} & R_{[12]} \\ \mathbf{0} & R_{[22]} \end{pmatrix}, \quad R_{[11]} \in \mathbb{C}^{\widetilde{r}_S \times \widetilde{r}_S}$$
(6.63)

it holds that $||R_{[22]}||_F \leq \sqrt{\ell - \tilde{r}_S} \xi |R_{\tilde{r}_S,\tilde{r}_S}|$, and $R_{[22]}$ it is set to zero, thus yielding a rank revealing decomposition of the form (6.60). 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 faster solver, or sparser least squares solution.

Indubitably, a QR factorization last squares algorithm should use pivoting, and it remains to see how to mount the pivoting device in Algorithms 5 and 9. The simplest way is to take the final upper triangular matrix R_S on exit, and recompute its QR factorization wit the Businger-Golub [4] column pivoting; the more elegant one is to build the pivoting in the algorithm. It can be easily seen than 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 with only the last step, when computing the last upper triangular matrix.

6.4. On real data and closedness under complex conjugacy

If the data \mathbf{f}_i in (3.10) 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 ($x \in {S, D}$ 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 (3.14), 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.

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

Proposition 6.7. Let all \mathbf{f}_i 's be real, i.e. $\mathbf{f}_i \in \mathbb{R}^n$, $\mathbb{A} \in \mathbb{R}^{n \times n}$. 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 (3.10) 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 so ordered 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 $\operatorname{Im}(\lambda_j) > 0$. Let ℓ_2 be the number of complex conjugate pairs. Hence $\ell = \ell_1 + 2 \cdot \ell_2, \ell_1, \ell_2 \ge 0$.

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} to 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 $(\Re(z_j) \Im(z_j)) = (z_j z_{j+1}) \Phi$. Note that $\sqrt{2}\Phi$ is unitary and that for $j = \ell_1 + 1, \ell_1 + 3, \dots, \ell - 1$

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

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 real matrix,

$$Z_{\ell}\Phi_{\lambda} = \begin{pmatrix} z_1, & \dots, & z_{\ell_1}, & \Re(z_{\ell_1+1}), & \Im(z_{\ell_1+1}), & \dots, & \Re(z_{\ell-1}), & \Im(z_{\ell-1}) \end{pmatrix}$$
(6.64)

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

$$\Phi_{\lambda}^{-1} \mathbf{\Lambda}^{i-1} \Phi_{\lambda} = \left[\bigoplus_{\mathrm{Im}(\lambda_j)=0} (\lambda_j^{i-1}) \right] \bigoplus \left[\bigoplus_{\mathrm{Im}(\lambda_j)>0} \hat{\Lambda}_j^{i-1} \right] = \left(\begin{array}{c} \bullet \bullet \\ \bullet \\ \bullet \bullet \\ \bullet \\ \bullet \bullet \\ \bullet \\$$

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

$$\|(\mathbf{W}\otimes\mathbb{I}_n)\left[\begin{pmatrix}\mathbf{f}_1\\\vdots\\\mathbf{f}_m\end{pmatrix}-\begin{pmatrix}Z_{\ell}\Delta_{\Lambda_1}\\\vdots\\Z_{\ell}\Delta_{\Lambda_m}\end{pmatrix}\vec{\boldsymbol{\alpha}}\right]\|_2=\|\begin{pmatrix}\mathbf{w}_1\mathbf{f}_1\\\vdots\\\mathbf{w}_m\mathbf{f}_m\end{pmatrix}-\begin{pmatrix}\mathbf{w}_1(Z_{\ell}\Phi_{\lambda})(\Phi_{\lambda}^{-1}\mathbf{\Lambda}^0\Phi_{\lambda})\\\vdots\\\mathbf{w}_m(Z_{\ell}\Phi_{\lambda})(\Phi_{\lambda}^{-1}\mathbf{\Lambda}^{m-1}\Phi_{\lambda})\end{pmatrix}\underbrace{(\Phi_{\lambda}^{-1}\vec{\boldsymbol{\alpha}})}_{\vec{\boldsymbol{\rho}}}\|_2.$$
(6.66)

If we introduce a change of variables in (6.66) by letting $\vec{\rho} = \Phi_{\lambda}^{-1}\vec{\alpha}$, then the optimal $\vec{\rho}$ in (6.66) must be real, and then the optimal $\vec{\alpha} = \Phi_{\lambda}\vec{\rho}$ reads

$$\vec{\boldsymbol{\alpha}} = \left(\rho_1 \quad \dots \quad \rho_{\ell_1}, \quad \rho_{\ell_1+1} + i\rho_{\ell_1+2}, \quad \rho_{\ell_1+1} - i\rho_{\ell_1+2}, \quad \dots, \quad \rho_{\ell-1} + i\rho_{\ell}, \quad \rho_{\ell-1} - i\rho_{\ell}\right)^T.$$
(6.67)

The difference between (6.66) and the original formulation (3.14) is in replacing the diagonal scaling matrices Λ^{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. Similar holds for (3.15), where now we have real QR factorization $Z_{\ell}\Phi_{\lambda} = Q\begin{pmatrix} R\\ 0 \end{pmatrix}$. Note that $Z_{\ell}\Phi_{\lambda}$ is of 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 e.g. on LAPACK, $Z_{\ell}\Phi_{\lambda}$ will be actually computed in the form (6.64), so that this switching to real arithmetic is simple.

Needles to say, computing the QR factorization in real arithmetic is (estimated, but can be easily demonstrated in numerical experiments with sufficiently large dimensions) at least twice faster than in complex. 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 (3.15), (3.16) are performed in real arithmetic.

And, finally, the fact that the solution is guaranteed to have complex conjugacy structure (6.67) analogous to the one of the selected eigenpairs, thus yielding real approximation, makes the final argument in favor of this formulation.

6.4.2. Algorithms of §6.2 for real data. After rewriting the steps (3.14 - 3.21) over \mathbb{R} using (6.65), (6.66), it remains to adapt the algorithms described in §6.2 to the case of real R and with the block-diagonal matrices $\widehat{\Lambda}^{i-1}$ instead of the diagonal Λ^{i-1} .

Clearly, the global structure of Algorithm 5 and Algorithm 9 remains unchanged. The difference is in the structure (6.51), which is changed as illustrated below (three real eigenvalues (\bullet) and two complex conjugate pairs (\blacklozenge)):

To put it simply, each complex conjugate pair of Ritz values creates a bulge (\bigstar). Hence, before running algorithms described in §6.2.2, 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 correction is needed only in the (2, 1) block, designated as *B* in §6.2.2.

7. Concluding Remarks

In this paper we have provided a new firm numerical linear algebra framework for solving structured least squares problem that arises in applications in e.g. computational fluid dynamic and multistatic antenna array processing. The numerical analysis explains the accuracy and the limitations of the normal equations based solution. New algorithm, based on a structure exploiting QR factorization has been presented with detailed numerical analysis and implementation details that should be the starting point for a high performance implementations on modern computing platforms. In the Appendix §8, we list few sample codes in Matlab; a LAPACK based implementation of all presented algorithms is under development.

8. Matlab source codes

For the reader's convenience, in this section we provide Matlab implementations of some of the algorithms described in the paper.

8.1. QR_Khatri_Rao_VTR

```
Algorithm 10 QR factorization of the Khatri-Rao product \mathbb{V}_{\ell m}^T \odot R
```

```
function T = QR_Khatri_Rao_VTR_2p( R, Lambda, p )
% QR_Khatri_Rao_VTR_2p computes the upper triangular factor in the QR factorization
\% of the Khatri-Rao product S=Khatri_Rao(Vlm.',R), where R is an <ell x ell>
\% upper triangular matrix, and Vlm is an <\!\!ell x m\!> Vandermonde matrix V, whose
 columns are defined as V(:,i) = Lambda.(i-1), i = 1,...,m, and m=2^p.
% This code is written only to illustrate the global recursive structure of the
% algorithm. It has not been optimized for run-time efficiency.
8.....
% Coded by Zlatko Drmac, Department of Mathematics, University of Zagreb.
% drmac@math.hr
% April 2018
8.....
% Input:
% R
       < ell x ell > (upper triangular) real or complex matrix
 Lambda < ell x 1 > real or complex vector, defines Vlm = Vandermonde marix
% p integer >=0
                 defines m = 2^p
% Output:
      < ell x ell > upper triangular QR fator of Khatri_Rao(Vlm.',R)
8 T
e
8
T = R; D = Lambda;
00
for i = 1 : p
   [~, T] = qr([T; T*diag(D)], 0);
   D = D.^{2};
end
end
```

31

Algorithm 11 QR factorization of the Khatri-Rao product $\mathbb{V}_{\ell,m}^T \odot R$

```
function [ T, b1 ] = QR_Khatri_Rao_VTR_2p_b( R, Lambda, p, b )
% QR_Khatri_Rao_VTR computes the upper triangular factor in the QR factorization
\% of the Khatri-Rao product S=Khatri.Rao(Vlm.',R), where R is an <ell x ell>
\% upper triangular matrix, and Vlm is an \ <ell x m> Vandermonde matrix V, whose
  columns are defined as V(:,i) = Lambda.^(i-1), i = 1,...,m, and m=2^p. 
% This code is written only to illustrate the global recursive structure of the
% algorithm. It has not been optimized for run-time efficiency.
ş.....
% Coded by Zlatko Drmac, Department of Mathematics, University of Zagreb.
% drmac@math.hr
% April 2018
e.....
% Input:
% R < ell x ell > (upper triangular) real or complex matrix
% Lambda < ell x 1 > real or complex vector, defines Vlm = Vandermonde marix
% p integer >=0 defines m = 2^p
% Output:
% T < ell x ell > upper triangular QR fator of Khatri_Rao(Vlm.',R)
8
2
T = R; D = Lambda; ell = length(Lambda); b1 = b;
00
for i = 1 : p
[Q, T] = qr( [ T ; T*diag(D)], 0 ) ; D = D.^2 ; s = 2^(i-1) ;
for j = 0 : 2^i : 2^p - 2^i
k = j + s ;
b1( j*ell+1 : (j+1)*ell ) = Q' * [ b1( j*ell+1 : (j+1)*ell ) ; ...
b1( k*ell+1 : (k+1)*ell ) ] ;
end
end
end
```

Algorithm 12 QR factorization of the Khatri-Rao product $\mathbb{V}_{\ell,m}^T \odot R$

```
function TT = QR_Khatri_Rao_VTR( R, Lambda, m )
% QR_Khatri_Rao_VTR computes the upper triangular factor in the QR
% factorization of the Khatri-Rao product S=Khatri_Rao(Vlm.',R), where
\% R is an <ell x ell> upper triangular matrix, and Vlm is an \, <ell x m> \,
% Vandermonde matrix V, whose columns are defined as V(:,i) = Lambda.<sup>1</sup>
\% i = 0, ..., m-1.
% This code has been written to illustrate the global recursive structure of the
% computation. It has not been optimized for speed.
8.....
% Coded by Zlatko Drmac, Department of Mathematics, University of Zagreb.
% drmac@math.hr
% June 2018
g.....
% Input:
% R <ell x ell> (upper triangular) matrix
\ Lambda <ell x 1 >\, vector, defines Vlm = Vandermonde marix
% m < 1 x 1 > integer > 0
8.....
% Output:
% T
       <ell x ell> upper triangular QR fator of Khatri_Rao(Vlm.',R)
8.....
00
J = floor(log2(m))+1; B = zeros(1, J);
for k = J : -1 : 1
   B(k) = mod(m, 2); m = (m-B(k))/2;
end
% B = (m)_2 is the binary representation of m
B = fliplr(B) ; % reverse the string and process it later from the left on
BB = find(B) - 1; % these are the increasingly ordered powers of two that
               % correspond to nonzero binary digits in (m)_2
lBB = length(BB) ; % this is the number of nonzero binary digits in (m)_2
% initialize the local and the global triangular factors
T = R; D = Lambda; D0 = D;
if BB(1) == 0
  TT = T ; k0 = 2 ; pow_shift = 1 ;
else
  TT = [] ; k0 = 1 ; pow_shift = 0 ;
end
% run over the powers of two and assemble the global factor
for k = 1 : BB(1BB) % the main loop runs up to the highest power
   [~, T] = qr( [ T ; T*diag(D)], 0 ) ; % this is a local triangular factor
                                   % built for each binary digit in (m)_2
   if (k == BB(k0))
      % interrupt for an update of the global triangular factor
      if ~isempty(TT)
         [~, TT] = qr( [ TT ; T*diag(D0.^ (pow_shift)) ], 0 );
      else % avoid trivial computation
         TT = T;
     end
     k0 = k0 + 1; pow_shift = pow_shift + 2^k;
   end
   D = D.^{2};
end
end
```

8.2. SLS_Khatri_Rao_NE

Algorithm 13 Normal equations solution to the LS problem $\|(\mathbb{V}_{\ell_m}^T \odot R) x - \vec{\mathbf{g}}\|_2 \longrightarrow \min$

```
function x = SLS_Khatri_Rao_NE( R, Lambda, G )
% SLS_Khatri_Rao_NE uses the normal equations to solve structured least squares
 problem || S * x - vec(G) ||_2 -> min, where S is the Khatri-Rao product,
\ S=Khatri_Rao(Vlm.',R), R is an <ell x ell> upper triangular matrix, and
% Vlm is an <ell x m> Vandermonde matrix, whose columns are defined as
% Vlm(:,i) = Lambda.^i, i = 0,...,m-1. Lambda is complex vector.
% The code is writen for clarity, not optimality.
8.....
% Coded by Zlatko Drmac, Department of Mathematics, University of Zagreb.
% drmac@math.hr
8.....
% Input:
% R <ell x ell> (upper triangular) matrix
% Lambda <ell x 1> vector, defines Vlm = Vandermonde marix
% G <ell x m> real or complex data
% Output:
% X
      <ell x 1> the solution of the LS problem.
8
[ell,m] = size(G);
Vlm = ones( ell, m ) ;
for j = 2 : m
Vlm(:,j) = Lambda.*Vlm(:,j-1) ;
end
K = (R'*R).*conj(Vlm*Vlm')
                             ;
b = (conj(Vlm) \cdot (R' \cdot G)) \cdot ones(m, 1);
x = K \setminus b
                             ;
end
```

8.3. SLS_W_Khatri_Rao_NE

```
Algorithm 14 Normal equations solution to the LS problem
\|(W \otimes I_{\ell}) \left[ (\mathbb{V}_{\ell,m}^T \odot R) x - \vec{\mathbf{g}} \right] \|_2 \longrightarrow \min
```

```
function x = SLS_W_Khatri_Rao_NE( R, Lambda, G, W )
% SLS_Khatri_Rao_NE uses the normal equations to solve structured least squares
\frac{1}{2} problem || kron(diag(W), eye(ell))(S * x - vec(G)) || 2 --> min, where S is
% the Khatri-Rao product, S=Khatri_Rao(Vlm.',R), R is an <ell x ell> upper
% triangular matrix, and Vlm is an <ell x m> Vandermonde matrix, whose columns
% are defined as Vlm(:,i) = Lambda.^i, i = 0,...,m-1. Lambda is complex vector.
% The code is writen for clarity, not optimality.
8.....
% Coded by Zlatko Drmac, Department of Mathematics, University of Zagreb.
% drmac@math.hr
%.....
% Input:
% R
      <ell x ell> (upper triangular) matrix
% Lambda <ell x 1> vector, defines Vlm = Vandermonde marix
% G <ell x m> real or complex data
                 real vector of positive weights
8 W
      <m x 1>
% Output:
% X
      <ell x 1> the solution of the LS problem.
2
[ell,m] = size(G);
Vlm = ones( ell, m ) ;
for j = 2 : m
Vlm(:,j) = Lambda.*Vlm(:,j-1) ;
end
K = (R'*R).*conj(Vlm*diag(W.^2)*Vlm')
b = (conj(Vlm*diag(W)).*(R'*G*diag(W)))*ones(m,1);
x = K \setminus b
end
```

8.4. $SLS_Spectral_Reconstruct_W_NE$

```
Algorithm 15 Normal equations solution to the LS problem \| [(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) - Z_\ell \Delta_{\boldsymbol{\alpha}}(\Lambda_1, \Lambda_2, \dots, \Lambda_m)] \mathbf{W} \|_F \longrightarrow \min
```

```
function x = SLS_Spectral_Reconstruct_W_NE(Z, Lambda, F, W)
% SLS_Spectral_Reconstruct_W_NE uses the normal equations to solve structured
% least squares problem
% || ( F-Z*diag(x)*(Lambda^0,Lambda^1,...,Lambda^(m-1))*diag(W) || F --> min,
% where Lambda and x are complex vectors, W is real vector with positive
% entries, and F and Z are real or complex matrices.
% The code is writen for clarity, not optimality.
8.....
% Coded by Zlatko Drmac, Department of Mathematics, University of Zagreb.
% drmac@math.hr
% April 2018
8.....
% Input:
       <n x ell> real or complex matrix
8 7
                 real or complex vector, defines Vlm = Vandermonde marix
% Lambda <ell x 1>
    <ell x m> real or complex data
<m x 1> real vector of positive weights
% F
    <m x 1>
8 W
% Output:
       <ell x 1> the solution of the LS problem.
% X
8
[Q,R] = qr(Z,0);
   = Q'*F
G
[ell,m] = size(G)
Vlm = ones(ell, m);
for j = 2 : m
  Vlm(:, j) = Lambda. * Vlm(:, j-1);
end
if nargin == 3
  K = (R'*R).*conj(Vlm*Vlm')
  b = (conj(Vlm) \cdot (R' * G)) * ones(m, 1);
else
  K = (R' * R) . * conj(Vlm * diag(W.^2) * Vlm')
  b = (conj(Vlm*diag(W)).*(R'*G*diag(W)))*ones(m,1);
end
x = K \setminus b;
end
```

8.5. Software license

The following license applies to the software in this section.

% % Redistribution and use in source and binary forms, with or without % modification, are permitted provided that the following conditions are met: % % 1. Redistributions of source code must retain the above copyright notice, % this list of conditions and the following disclaimer. % 2. Redistributions in binary form must reproduce the above copyright notice, % this list of conditions and the following disclaimer in the documentation % and/or other materials provided with the distribution. % 3. Neither the name of the copyright holder nor the names of its contributors % may be used to endorse or promote products derived from this software % without specific prior written permission. % % The copyright holder provides no reassurances that the source code % provided does not infringe any patent, copyright, or any other % intellectual property rights of third parties. The copyright holder % disclaims any liability to any recipient for claims brought against % recipient by any third party for infringement of that parties % intellectual property rights. % % THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS "AS IS" AND % ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED % WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE % DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE % FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL % DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR % SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER % CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, % OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE % OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. %

References

- Hassan Arbabi and Igor Mezić. Ergodic Theory, Dynamic Mode Decomposition, and Computation of Spectral Properties of the Koopman Operator. SIAM Journal on Applied Dynamical Systems, 16(4):2096–2126, November 2017.
- [2] Michele Benzi, Gene H. Golub, and Jörg Liesen. Numerical solution of saddle point problems. Acta Numerica, 14:1137, 2005.
- [3] Å. Björck. Stability analysis of the method of seminormal equations for linear least squares problems. Linear Algebra and its Applications, 88-89:31–48, 1987.
- [4] P. A. Businger and G. H. Golub. Linear least squares solutions by Householder transformations. *Numerische Mathematik*, 7:269–276, 1965.
- [5] A. Cat Le Ngo, J. See, and R. Chung-Wei Phan. Sparsity in Dynamics of Spontaneous Subtle Emotions: Analysis & Application. *ArXiv e-prints*, January 2016.
- [6] K. K. Chen, J. H. Tu, and C. W. Rowley. Variants of Dynamic Mode Decomposition: Boundary Condition, Koopman, and Fourier Analyses. *Journal of Nonlinear Science*, 22(6):887–915, April 2012.

- [7] Scott T. M. Dawson, Maziar S. Hemati, Matthew O. Williams, and Clarence W. Rowley. Characterizing and correcting for the effect of sensor noise in the dynamic mode decomposition. *Experiments in Fluids*, 57(3):42, 2016.
- [8] J. Demmel. On floating point errors in Cholesky. LAPACK Working Note 14, Computer Science Department, University of Tennessee, October 1989.
- [9] Z. Drmač, I. Mezić, and R. Mohr. Data driven Koopman spectral analysis in Vandermonde-Cauchy form via the DFT: numerical method and theoretical insights. *ArXiv e-prints*, August 2018.
- [10] Z. Drmač, I. Mezić, and R. Mohr. Data driven modal decompositions: analysis and enhancements. SIAM Journal on Scientific Computing, 40(4):A2253–A2285, 2018.
- [11] Z. Drmač and Z. Bujanović. On the failure of rank revealing QR factorization software a case study. ACM Trans. Math. Softw., 35(2):1–28, 2008.
- [12] Z. Drmač, M. Omladič, and K. Veselić. On the perturbation of the Cholesky factorization. SIAM J. Matrix Anal. Appl., 15(4):1319–1332, 1994.
- [13] Zlatko Drmač, Igor Mezić, and Ryan Mohr. Data driven Koopman spectral analysis in Vandermonde– Cauchy form via the DFT: numerical methods and theoretical insights. AIMDyn Tech Report, July 2018.
- [14] Sambuddha Ghosal, Vikram Ramanan, Soumalya Sarkar, Satyanarayanan Chakravarthy, and Soumik 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, 2016.
- [15] Nick Gould, Dominique Orban, and Tyrone Rees. Projected krylov methods for saddle-point systems. SIAM Journal on Matrix Analysis and Applications, 35(4):1329–1343, 2014.
- [16] M. S. Hemati, C. W. Rowley, E. A. Deem, and L. N. Cattafesta. De-Biasing the Dynamic Mode Decomposition for Applied Koopman Spectral Analysis. ArXiv e-prints, February 2015.
- [17] M. S. Hemati, M. O. Williams, and C. W. Rowley. Dynamic mode decomposition for large and streaming datasets. *Physics of Fluids*, 26(11):111701, November 2014.
- [18] R. A. Horn and C. R. Johnson. Topics in Matrix Analysis. Cambridge University Press, 1991.
- [19] M. R. Jovanović, P. J. Schmid, and J. W. Nichols. Sparsity-promoting dynamic mode decomposition. *Phys. Fluids*, 26(2):024103 (22 pages), 2014.
- [20] Mihailo R. Jovanović, Peter J. Schmid, and Joseph W. Nichols. Dmdsp Sparsity–Promoting Dynamic Mode Decomposition. http://people.ece.umn.edu/users/mihailo/software/dmdsp/, 2013-2014.
- [21] Milan Korda and Igor Mezić. On Convergence of Extended Dynamic Mode Decomposition to the Koopman Operator. Journal of Nonlinear Science, 28(2):687–710, November 2017.
- [22] Jiaqing Kou and Weiwei Zhang. An improved criterion to select dominant modes from dynamic mode decomposition. European Journal of Mechanics - B/Fluids, 62:109 – 129, 2017.
- [23] Hanoch Lev-Ari. Efficient solution of linear matrix equations with application to multistatic antenna array processing. *Commun. Inf. Syst.*, 05(1):123–130, 2005.
- [24] Joshua L. Proctor and Philip A. Eckhoff. Discovering dynamic patterns from infectious disease data using dynamic mode decomposition. *International Health*, 7(2):139–145, 2015.
- [25] Clarence W Rowley, Igor Mezić, Shervin Bagheri, Philipp Schlatter, and Dan S Henningson. Spectral analysis of nonlinear flows. *Journal of fluid mechanics*, 641:115–127, 2009.
- [26] P. J. Schmid, L. Li, M. P. Juniper, and O. Pust. Applications of the dynamic mode decomposition. *Theoretical and Computational Fluid Dynamics*, 25(1):249–259, 2011.
- [27] Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. Journal of Fluid Mechanics, 656:5–28, 008 2010.
- [28] R. Schreiber and C. van Loan. A storage-efficient WY representation for products of Householder transformations. SIAM J. Sci. Stat. Comput., 10(1):53–57, 1989.
- [29] B. Suri, J. Tithof, R. Mitchell, R. O. Grigoriev, and M. F. Schatz. Velocity profile in a two-layer Kolmogorov-like flow. *Physics of Fluids*, 26(5):053601, May 2014.
- [30] Kunihiko Taira, Steven Brunton, Scott Dawson, Clarence Rowley, Tim Colonius, Beverley J. McKeon, Oliver Schmidt, Stanislav Gordeyev, V Theofilis, and Lawrence Ukeiley. Modal analysis of fluid flows: An overview. AIAA Journal, 55(12):4013–4041, 2017.
- [31] Jeffrey Tithof, Balachandra Suri, Ravi Kumar Pallantla, Roman O. Grigoriev, and Michael F. Schatz. Bifurcations in a quasi-two-dimensional kolmogorov-like flow. *Journal of Fluid Mechanics*, 828:837866, 2017.

- [32] A. van der Sluis. Condition numbers and equilibration of matrices. *Numerische Mathematik*, 14:14–23, 1969.
- [33] Matthew O. Williams, Ioannis G. Kevrekidis, and Clarence W. Rowley. A data-driven approximation of the Koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.