TRUNCATED LSQR FOR MATRIX LEAST SQUARES PROBLEMS AND APPLICATION TO DICTIONARY LEARNING *

Valeria Simoncini, Lorenzo Piccinini

TRUNCATED LSQR FOR MATRIX LEAST SQUARES PROBLEMS AND APPLICATION TO DICTIONARY LEARNING *

LORENZO PICCININI† AND VALERIA SIMONCINI‡

Abstract. We are interested in the numerical solution of the matrix least squares problem

$$\min_{X\in\mathbb{R}^{m\times m}} \|AXB+CXD-F\|_{\mathcal{F}},$$

with A and C full column rank, B, D full row rank, F an $n\times n$ matrix of low rank, and $\|\cdot\|_{\mathcal{F}}$ the Frobenius norm. We derive a matrix-oriented implementation of LSQR, and devise an implementation of the truncation step that exploits the properties of the method. Experimental comparisons with the Conjugate Gradient method applied to the normal matrix equation and with a (new) sketched implementation of matrix LSQR illustrate the competitiveness of the proposed algorithm. We also explore the applicability of our method in the context of Kronecker-based Dictionary Learning, and devise a representation of the data that seems to be promising for classification purposes.

Key words. Matrix least squares, Kronecker products, rank truncation, large matrices, dictionary learning.

AMS subject classifications. 65F45, 65F55, 15A23.

1. Introduction.

We are interested in the numerical solution of the Generalized algebraic Sylvester equation

$$AXB+CXD=F, \tag{1.1}$$

for the unknown matrix X, with $A, C \in \mathbb{R}^{n\times m}$, $B, D \in \mathbb{R}^{m\times n}$ and $F \in \mathbb{R}^{n\times n}$, with $n > m$. In the following we assume that all coefficient matrices A, B^T, C and D^T have maximum rank, and that they have the same dimensions. The problem can be easily adapted to the case where X is rectangular. For general F the problem is overdetermined, hence a least squares formulation is more appropriate, that is

$$\min_{X\in\mathbb{R}^{m\times m}} \|AXB+CXD-F\|_{\mathcal{F}}, \tag{1.2}$$

where $\|\cdot\|_{\mathcal{F}}$ denotes the Frobenius norm. In the following, we will also assume that F has low rank, that is $F = F_1F_2^T$ with F_1, F_2 having few columns.

Least squares matrix formulations have recently emerged as good candidates for memory-saving coefficient representation strategies in ill-posed problems [19],[34], and in data science problems such as dictionary learning [8],[7], in addition to being building blocks for more complex procedures, see, e.g., [26].

More general forms that allow for distinct unknowns in the two matrix terms have been addressed in the recent literature. More precisely, the problem can be written as $\min_{X,Y\in\mathbb{R}^{m\times m}} \|AXB+CYD-F\|_{\mathcal{F}}$ (see, e.g., [43]), where X, Y could also have different dimensions. As a special case, the T-least squares problem

$$\min_{X\in\mathbb{R}^{m\times m}} \|AXB+CX^TD-F\|_{\mathcal{F}} \tag{1.3}$$

*Version of February 4, 2024. The authors are members of the INdAM Research Group GNCS that partially supported this work.

†Dipartimento di Matematica, Alma Mater Studiorum - Università di Bologna, Piazza di Porta San Donato 5, 40126 Bologna, Italia. Email: lorenzo.piccinini12@unibo.it

‡Dipartimento di Matematica, AM^2, Alma Mater Studiorum - Università di Bologna, Piazza di Porta San Donato 5, 40126 Bologna, Italy, and IMATI-CNR, Via Ferrata 5/A, Pavia, Italy. Email: valeria.simoncini@unibo.it

can be considered, as it also occurs for instance in certain semi-definite programming problems [4]. A different solution direction has recently been explored, solving these least squares problems via gradient-descent type methods and other optimization procedures; see, e.g., [20],[9] and references therein.

In this work we limit our attention to (1.2), while noticing that many of the strategies also apply to other contexts, such as (1.3); see, e.g., [33]. In particular, we stress that all proposed algorithms apply without significant changes to the multiterm setting, that is to problems of the form

$$\min_{X\in\mathbb{R}^{m\times m}} \|\sum_{i=1}^{\ell} A_i X B_i - F\|_{\mathcal{F}}, \tag{1.4}$$

with conforming dimensions for all given matrices.

A classical way to solve (1.2) consists of transforming the problem into vector form via the Kronecker product. More precisely, let us set $\boldsymbol{A} = B^T \otimes A + D^T \otimes C$, $\boldsymbol{x} = \mathrm{vec}(X)$ and $\boldsymbol{f} = \mathrm{vec}(F)$, where vec transforms a matrix into a vector by lining up its columns, and $M \otimes A$ is the Kronecker product of the two matrices A, M, defined as

$$M \otimes A = \begin{bmatrix} AM_{1,1} & \dots & AM_{1,m} \\ \vdots & \ddots & \vdots \\ AM_{n,m} & \dots & AM_{n,m} \end{bmatrix} \in \mathbb{R}^{nn\times mm}, \quad A, M \in \mathbb{R}^{n\times m}.$$

Then (1.2) is equivalent to

$$\min_{\boldsymbol{x}\in\mathbb{R}^{m^2}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{f}\|, \tag{1.5}$$

where $\|\cdot\|$ is the Euclidean vector norm. In the single term case, that is for C and D equal to zero, then direct methods that exploit the QR factorization can be efficiently adapted [11]. Direct QR-based methods can also be applied in the general case, as long as the involved matrices in (1.5) have small dimensions, so that $\boldsymbol{A}$ can be handled efficiently.

The problem becomes significantly more challenging for n, m large. Standard iterative methods can be used for (1.5), though $\boldsymbol{A}$ should never be explicitly computed as it is generally dense. In addition, it is crucial to take into account the structure more directly, so as to limit the use of memory allocations. In the past decade, several iterative system solvers and minimal residual methods have been adapted to directly exploit the matrix and tensorial structure of the given problem, so as to avoid the explicit use of matrix vectorizations, which destroy the problem structure [31],[6],[22],[23],[41]; this is particularly crucial in the case of fully tensorized problems, see, e.g., [5],[24]. A class of methods that aim at addressing these issues is that of truncated matrix-oriented iterative solvers. The idea is to take a classical iterative method for (1.5), in our case, recover the matrix form of all iterates, and then employ truncation procedures to store iterates as low-rank factors, rather than full matrices or very long vectors; all methods cited above share these properties.

We propose to employ the LSQR method ([29]) for the vectorized problem, by first deriving the matrix-oriented version of the method, and then using a new implementation of the truncation that exploits the algorithmic properties. To the best of our knowledge this variant of LSQR has not been explored in the past, for matrix least squares problems as that in (1.2).

In the vector case it is known that LSQR is preferable to the conjugate gradient method (CG) on the normal equation because of stability issues, although the two

methods are mathematically equivalent. In the truncated matrix setting, other differences arise. On the one hand, the generated space bases in LSQR are bound to quickly lose their orthogonality properties, so that the reduced least squares problem is no longer related to the original problem. On the other hand, CG requires explicitly working with the normal equation, which entails addressing a *multiterm* linear matrix equation. For these reasons, truncations of the matrix iterates may have very different effects on the two algorithms. Other issues are of concern, such as the computational cost of stopping criteria, and the role of the parameters employed during the truncation at different stages of the iteration. We will address all these issues, and in particular we will devise a new form for the truncation strategy that exploits the properties of LSQR. The new formulation allows us to decrease the cost of computing the recurrence coefficients, and lowers the cost of the orthogonalization step. As an alternative to this new formulation, we have also implemented a preliminary sketched version of the matrix-oriented LSQR method, in which the algorithmic coefficients are computed by embedding the bases in a fixed subspace [27]. Our experiments show that the sketched algorithm is not competitive, with respect to the method with the new truncation implementation. Finally, we explore the application of this methodology in the context of Dictionary Learning, with the problem formalized in [7, 8], towards its use in image classification. We propose a new way to set up the data and to analyze the resulting solution, that appear to successfully classify a new given image. These promising results will be deepened in future work on the topic.

The paper is organized as follows. Section 2 recalls the vector LSQR method. Section 3 introduces some properties of the matrix least squares problem, while section 4 devises the new matrix-oriented version of LSQR. Section 4.1 discusses the concerns associated with the stopping rule, and section 5 describes the new implementation of the truncation step, so as to take advantage of the existing orthogonality. A few considerations on the effect of truncation are detailed in section 6. Section 7 describes a possible implementation of the sketched matrix LSQR, while section 8 reports on our experimental study. Section 9 describes the application of LSQR to the dictionary learning problem, and derives a classification strategy. Finally, section 10 contains the conclusions.

Throughout the paper bold face letters refer to matrices and vectors used in the Kronecker formulation. All experiments were ran in Matlab [28].

2. Vector version of the LSQR method. For later derivations, in this section we recall the standard LSQR method first described in [29] to solve the least squares problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^{m^2}} \|\boldsymbol{A}x-\boldsymbol{b}\|_2 \tag{2.1}$$

with $\boldsymbol{A}\in\mathbb{R}^{n^2\times m^2}$ and $\boldsymbol{b}\in\mathbb{R}^{n^2}$, assuming $n>m$ and $\boldsymbol{A}$ full column rank. The procedure can be derived from the Golub-Kahan-Lanczos bidiagonalization [14]. This method allows us to compute the orthogonal matrices $\boldsymbol{U},\boldsymbol{V}$ satisfying $\boldsymbol{A}=\boldsymbol{U}\tilde{\boldsymbol{A}}\boldsymbol{V}^H$ with $\tilde{\boldsymbol{A}}$ lower bidiagonal.

Starting from problem (2.1), we compute the lower bidiagonalization as follows. Given the initial vectors $\boldsymbol{v}_0=\boldsymbol{0}$, $\boldsymbol{u}_1=\boldsymbol{b}/\beta_1$, where $\beta_1=\|\boldsymbol{b}\|\neq 0$, the algorithm computes for $i=1,2,\dots$

$$\alpha_i\boldsymbol{v}_i=\boldsymbol{A}^T\boldsymbol{u}_i-\beta_i\boldsymbol{v}_{i-1},\quad \|\boldsymbol{v}_i\|=1,\quad \beta_{i+1}\boldsymbol{u}_{i+1}=\boldsymbol{A}\boldsymbol{v}_i-\alpha_i\boldsymbol{u}_i,\quad \|\boldsymbol{u}_{i+1}\|=1, \tag{2.2}$$

until $\alpha_i=0$ or $\beta_{i+1}=0$ or $i=\min\{n,m\}$. Denote with $\boldsymbol{U}_k=[\boldsymbol{u}_1,\boldsymbol{u}_2,\dots,\boldsymbol{u}_k]$ and $\boldsymbol{V}_k=[\boldsymbol{v}_1,\boldsymbol{v}_2,\dots,\boldsymbol{v}_k]$ the matrices with orthonormal columns, $\boldsymbol{L}_k$ the square lower

bidiagonal matrix with main diagonal $[\alpha_1, \dots, \alpha_k]$ and subdiagonal $[\beta_2, \dots, \beta_k]$, and let $\boldsymbol{B}_k = [\boldsymbol{L}_k^T, \beta_{k+1}\boldsymbol{e}_k]^T$. Then the procedure stops at iteration p when either $\alpha_{p+1} = 0$ or $p = m$, giving

$$\boldsymbol{A}^T\boldsymbol{U}_p = \boldsymbol{V}_p\boldsymbol{L}_p^T, \qquad \boldsymbol{A}\boldsymbol{V}_p = \boldsymbol{U}_{p+1}\boldsymbol{B}_p.$$

Let $\boldsymbol{x}_k = \boldsymbol{V}_k\boldsymbol{y}_k$ for some vector $\boldsymbol{y}_k$ to be determined, and let $\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$ be the associated residual. Recalling that $\boldsymbol{b} = \boldsymbol{U}_{k+1}\boldsymbol{e}_1\beta_1$ and $\boldsymbol{A}\boldsymbol{V}_k = \boldsymbol{U}_{k+1}\boldsymbol{B}_k$, we can obtain

$$\boldsymbol{r}_k = \boldsymbol{U}_{k+1}\boldsymbol{e}_1\beta_1 - \boldsymbol{A}\boldsymbol{V}_k\boldsymbol{y}_k = \boldsymbol{U}_{k+1}(\boldsymbol{e}_1\beta_1 - \boldsymbol{B}_k\boldsymbol{y}_k) =: \boldsymbol{U}_{k+1}\boldsymbol{t}_{k+1}.$$

We determine $\boldsymbol{y}_k$ so as to minimize the residual norm. Using that $\boldsymbol{U}_{k+1}$ has orthonormal columns, this leads us to solve a reduced dimension least squares problem, that is

$$\min_{\boldsymbol{x}_k} \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k\| = \min_{\boldsymbol{y}_k} \|\boldsymbol{e}_1\beta_1 - \boldsymbol{B}_k\boldsymbol{y}_k\|. \tag{2.3}$$

Since $\boldsymbol{B}_k$ is bidiagonal, it is advantageous to solve (2.3) using Givens rotations, so that $\boldsymbol{B}_k = \boldsymbol{Q}_k \begin{bmatrix} \boldsymbol{R}_k \\ \boldsymbol{0} \end{bmatrix}$. We write:

$$\|\boldsymbol{e}_1\beta_1 - \boldsymbol{B}_k\boldsymbol{y}_k\| = \left\| \boldsymbol{Q}_k^T\boldsymbol{e}_1\beta_1 - \begin{bmatrix} \boldsymbol{R}_k \\ \boldsymbol{0} \end{bmatrix} \boldsymbol{y}_k \right\| = \left\| \begin{bmatrix} \phi_k - \boldsymbol{R}_k\boldsymbol{y}_k \\ \bar{\Phi}_{k+1} \end{bmatrix} \right\|, \tag{2.4}$$

where $\phi_k = [\Phi_1, \dots, \Phi_k]^T$. The vectors $\boldsymbol{y}_k$ and $\boldsymbol{t}_{k+1}$ can be found from

$$\boldsymbol{R}_k\boldsymbol{y}_k = \boldsymbol{f}_k, \qquad \boldsymbol{t}_{k+1} = \boldsymbol{Q}_k^T \begin{bmatrix} \boldsymbol{0} \\ \bar{\Phi}_{k+1} \end{bmatrix}.$$

Since $[\boldsymbol{R}_k, \boldsymbol{f}_k]$ is obtained from $[\boldsymbol{R}_{k-1}, \boldsymbol{f}_{k-1}]$ by adding a row and column, the iterate $\boldsymbol{x}_k$ can be updated as

$$\boldsymbol{x}_k = \boldsymbol{V}_k\boldsymbol{R}_k^{-1}\mathbf{f}_k = \boldsymbol{G}_k\boldsymbol{f}_k,$$

where $\boldsymbol{G}_k = [\boldsymbol{g}_1, \boldsymbol{g}_2, \dots, \boldsymbol{g}_k]$ is defined by solving the system $\boldsymbol{R}_k^T\boldsymbol{G}_k^T = \boldsymbol{V}_k^T$ by forward substitution. Setting $\boldsymbol{g}_0 = \boldsymbol{x}_0 = \boldsymbol{0}$, it holds that

$$\boldsymbol{g}_k = \frac{1}{\rho_k}(\boldsymbol{v}_k - \theta_k\boldsymbol{g}_{k-1}), \qquad \boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \Phi_k\boldsymbol{g}_k.$$

Finally, we recall that the QR decomposition at each iteration only requires the k-th plane rotation to act on the last two rows of $[\boldsymbol{B}_k, \boldsymbol{e}_1\beta_1]$ to annihilate β_{k+1}, that is

$$\begin{bmatrix} c_k & s_k \\ s_k & -c_k \end{bmatrix} \begin{bmatrix} \bar{\rho}_k & 0 & \bar{\Phi}_k \\ \beta_{k+1} & \alpha_{k+1} & 0 \end{bmatrix} = \begin{bmatrix} \rho_k & \theta_{k+1} & \Phi_k \\ 0 & \bar{\rho}_{k+1} & \bar{\Phi}_{k+1} \end{bmatrix},$$

where $\hat{\rho}_1 = \alpha_1$, $\bar{\Phi}_1 = \beta_1$ and the scalars c_k and s_k are the rotation coefficients. The complete procedure can be found in [29] and is reported for future reference in Algorithm 2.1.

The original stopping criterion fully relies on the orthonormality of the bases, and in particular on the equivalence (2.3); see [29]. In our setting, we will see that orthonormality of the two bases is lost, so that it will not be possible to cheaply evaluate the true least squares residual norm or, as it is common, the residual norm of the normal equation, through the computed scalars. If desired, these quantities must be evaluated directly.

Algorithm 2.1 LSQR ALGORITHM

Require: $\boldsymbol{A}$, $\boldsymbol{b}$, imax

$\beta_1 \boldsymbol{u}_1 = \boldsymbol{b}$, $\alpha_1 \boldsymbol{v}_1 = \boldsymbol{A}^T \boldsymbol{u}_1$, $\boldsymbol{g}_1 = \boldsymbol{v}_1$, $\mathbf{x}_0 = \mathbf{0}$, $\bar{\Phi}_1 = \beta_1$, $\bar{\rho}_1 = \alpha_1$

while $i <$ imax **do**

$i = i + 1$

$\beta_{i+1} \boldsymbol{u}_{i+1} = \boldsymbol{A} \boldsymbol{v}_i - \alpha_i \boldsymbol{u}_i$

$\alpha_{i+1} \boldsymbol{v}_{i+1} = \boldsymbol{A}^T \boldsymbol{u}_{i+1} - \beta_{i+1} \boldsymbol{v}_i$

$\rho_i = (\bar{\rho}_i^2 + \beta_{i+1}^2)^{1/2}$

$c_i = \bar{\rho}_i / \rho_i, \;\; s_i = \beta_{i+1} / \rho_i$

$\theta_{i+1} = s_i \alpha_{i+1}, \qquad \bar{\rho}_{i+1} = -c_i \alpha_{i+1}$

$\Phi_i = c_i \bar{\Phi}_i, \qquad \bar{\Phi}_{i+1} = s_i \bar{\Phi}_i$

$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + (\Phi_i / \rho_i) \boldsymbol{g}_i$

$\boldsymbol{g}_{i+1} = \boldsymbol{v}_{i+1} - (\theta_{i+1} / \rho_i) \boldsymbol{g}_i$

test for convergence

end while

3. The matrix least squares problem. We start by analyzing some algebraic properties of the problem, that can be used in its numerical solution when the dimensions are small.

We recall a general rank equivalence result, for the more general setting of two unknown matrices. The statement is adapted to our setting.

PROPOSITION 3.1. [16, Th. 4.4.25] *Let $A \in \mathbb{R}^{n \times m}$, $D \in \mathbb{R}^{m \times n}$ and $F \in \mathbb{R}^{n \times n}$. There are matrices $Z \in \mathbb{R}^{m \times n}$ and $Y \in \mathbb{R}^{n \times n}$ such that $AZ + YD = F$ if and only if*

$$\operatorname{rank} \begin{bmatrix} A & D \\ 0 & F \end{bmatrix} = \operatorname{rank} \begin{bmatrix} A & 0 \\ 0 & F \end{bmatrix}.$$

In our context, that is for $Z = Y$, we cannot in general use the if-and-only-if statement. However, one of the directions still holds. Indeed, we can use the theorem above to say that if (1.1) has a solution X, by setting $Z = XB$ and $Y = CX$, we can infer that the rank equality holds. On the other hand, even assuming that the rank condition holds, so that the two matrices Z and Y exist, then this does not necessarily imply that these two matrices are related as $Z = XB$ and $Y = CX$, so as to ensure X exists. Hence, posing the problem in terms of a least squares problem appears to be essential.

The least squares solution can be obtained by solving the vector oriented version of the problem, using the Kronecker product, as in (1.5). Nonetheless, the problem structure can be taken into account so as to avoid the QR factorization of the large Kronecker product matrix, as shown in the following proposition.

PROPOSITION 3.2. *Let*

$$[A, C] = Q \begin{bmatrix} R_{1,1} & R_{1,2} \\ & R_{2,2} \end{bmatrix}, \qquad [B^T, D^T] = U \begin{bmatrix} S_{1,1} & S_{1,2} \\ & S_{2,2} \end{bmatrix},$$

be the reduced QR decompositions of the given matrices. Then the problem (1.5) can be rewritten as

$$\min_{\boldsymbol{x}} \|\tilde{\boldsymbol{c}} - \left(\begin{bmatrix} S_{1,2} \\ S_{2,2} \end{bmatrix} \otimes R_{1,1} + S_{1,1} \otimes \begin{bmatrix} R_{1,2} \\ R_{2,2} \end{bmatrix} \right) \boldsymbol{x}\|$$

with $\tilde{c} = \operatorname{vec}(Q^T C U)$.

Proof. We just need to notice that

$$B^T \otimes A + D^T \otimes C = (Q \otimes U) \left(\begin{bmatrix} S_{1,2} \\ S_{2,2} \end{bmatrix} \otimes R_{1,1} + S_{1,1} \otimes \begin{bmatrix} R_{1,2} \\ R_{2,2} \end{bmatrix} \right),$$

from which the final least squares problem follows. ∎

The previous derivation does not make any assumption on the rank of $[A, C]$, hence either R factor of the considered QR factorizations could be rank deficient. In the extreme case where range(A)=range(C), the matrix problem simplifies. Let us write $C = AG_A$ and $B^T = D^T G_B$ for some nonsingular G_A, G_B (specifically, $G_A = (A^T A)^{-1} A^T C$, and similarly for G_B). Then (1.1) becomes

$$XG_B^T + G_A X = (A^T)^{-1} A^T F D^T (DD^T)^{-1}, \tag{3.1}$$

that is, X solves a standard Sylvester equation. In general, the following simple result holds.

PROPOSITION 3.3. *Assume that $\boldsymbol{A}$ is full column rank, so that the unique solution $\boldsymbol{x}$ to (2.1) exists, and let X be such that $\boldsymbol{x} = \mathrm{vec}(X)$. Assume also that A and D are full column rank. Then it holds that the spectra of $G_A = (A^T A)^{-1} A^T C$ and $G_B = (DD^T)^{-1} DB^T$ do not intersect, except possibly in zero.*

Proof. Let $R = AXB + CXD - F$, so that X solves the matrix equation $AXB + CXD = F + R$. Multiplying from the left by A^T and from the right by D^T we obtain $A^T AXBD^T + A^T CXDD^T = A^T (F + R) D^T$. Dividing by $A^T A$ from the left and by DD^T from the right, we obtain the square matrix equation in (3.1).

If at least one of the matrices G_A, G_B is nonsingular, the solution X is unique, and it must hold that G_A and G_B have disjoint spectra [16]. ∎

Matrix oriented algorithms aim at approximating the solution X as a low-rank factorization. It is important to recognize that using low-rank approximation makes sense if the exact solution can be well approximated by a low-rank matrix. While singular value decay of the solution for Sylvester equations has been investigated in the literature, with very insightful results especially in the symmetric case [1],[36],[15][32],[37], we are not aware of corresponding results for the overdetermined equation. Hence, in the following we proceed with the low-rank approximation assuming that it is appropriate. Whenever the problem dimensions allowed, the singular value decay of the least squares solution was verified, by using the solution of the vector formulation. We leave this fundamental analysis to later research.

4. Matrix-oriented LSQR with standard truncation implementation. We start by writing the matrix-oriented version of the LSQR method described in section 2, which merely corresponds to replacing vectors with their counterpart matrices, whenever possible, using the vec operator. For example:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{\phi_1}{\rho_1} \boldsymbol{g}_k \qquad \longrightarrow \qquad X_{k+1} = X_k + \frac{\phi_1}{\rho_1} G_k,$$

and

$$\boldsymbol{x}^T \boldsymbol{y} = \mathrm{trace}(X^T Y), \qquad \|\boldsymbol{x}\|_2 = \|X\|_{\mathcal{F}}.$$

With these modifications, the algorithm computes the same quantities of the method in vector form, with the only advantage taking place if dense matrix-matrix operations can be performed efficiently. The "vanilla" matrix version of LSQR is

reproduced in Algorithm 4.1, where $\mathcal{L} : \mathbb{R}^{m\times m} \to \mathbb{R}^{n\times n}$ is the operator associated to the matrix equation, that is $\mathcal{L}(X) = AXB + CXD$, while its transpose is $\mathcal{L}^T(Y) = A^TYB^T + C^TYD^T$. The algorithm solves the original least squares problem without ever explicitly building $\mathcal{L}^T(\mathcal{L}(\cdot))$, that from now on will be called the normal operator.

Algorithm 4.1 LSQR ALGORITHM (MATRIX VERSION)

Require: $\mathcal{L}$, $\mathcal{L}^T$, F, tol, imax
  $\beta = \|F\|_{\mathcal{F}}, \quad U = \frac{F}{\beta}, \quad r = \beta, V = \mathcal{L}^T(U), \quad \alpha = \|V\|_{\mathcal{F}}, \quad V = \frac{V}{\alpha}$
  $\phi = \beta, \quad \rho = \alpha, \quad i = 0, \quad X = 0, G = V, \quad r_0 = \beta$
  while $i <$ imax, $\frac{r}{r_0} >$ tol **do**
    $i = i + 1$
    $U = \mathcal{L}(V) - \alpha U, \quad \beta = \|U\|_{\mathcal{F}}, \quad U = \frac{U}{\beta}$
    $V = \mathcal{L}^T(U) - \beta V, \quad \alpha = \|V\|_{\mathcal{F}}, \quad V = \frac{V}{\alpha}$
    $\rho_1 = \sqrt{\rho^2 + \beta^2}, \quad c = \frac{\rho}{\rho_1}, \quad s = \frac{\beta}{\rho_1}$
    $\theta = s\alpha, \quad \rho = -c\alpha, \quad \phi_1 = c\phi, \quad \phi = s\phi$
    $X = X + \frac{\phi_1}{\rho_1}G, \quad G = V - \frac{\theta}{\rho_1}G$
    $r = \phi$
    check convergence
  end while

To take full advantage of the matrix form in terms of both memory consumption and computational costs, it has become standard practice to perform some kinds of truncation to maintain iterates in factored form, whenever the known matrix F can be approximated well by a low-rank matrix, with known factors – in our setting F is in fact always given as a low-rank matrix; we refer to, e.g., [23],[21],[30],[38] and their references. The implementation is then modified in such a way that each updating step is done only on the low-rank factors. To compute the low-rank factors of a given matrix, we can proceed as follows; see [23]. Let $Y_j = Q_jR_j$, $j = 1, 2$ be the reduced QR decomposition of each factor of $Y = Y_1Y_2^T$. Then

$$\begin{aligned} Y &= Q_1(R_1R_2^T)Q_2^T = Q_1USV^TQ_2^T \\ &\approx (Q_1U_{:,1:r}S^{\frac{1}{2}}_{1:r,1:r})S^{\frac{1}{2}}_{1:r,1:r}(Q_2V_{:,1:r})^T =: Y_1Y_2^T. \end{aligned} \tag{4.1}$$

The final matrices Y_1, Y_2 replace the original factors, and have hopefully lower column dimension. The new rank is obtained with the following truncation criterion

$$r = \min_{\ell\le k}\left(\frac{\sum_{i=1}^{\ell} S_{1:i,1:i}}{\sum_{i=1}^{k} S_{1:i,1:i}} > 1 - \texttt{tol}\right),$$

where k is the rank of the original matrix Y. The generic update $V^{(k+1)} = V^{(k)} + P^{(k)}$ is performed by using the addend factors: assuming that the two matrices $V^{(k)}, P^{(k)}$ are in factored form, then

$$V^{(k)} + P^{(k)} = V_1^{(k)}(V_2^{(k)})^T + P_1^{(k)}(P_2^{(k)})^T = [V_1^{(k)}, P_1^{(k)}][V_2^{(k)}, P_2^{(k)}]^T.$$

Hence, by rank reducing the two factors $[V_1^{(k)}, P_1^{(k)}]$, $[V_2^{(k)}, P_2^{(k)}]$ as in (4.1), we obtain

$$V^{(k+1)} := V_1^{(k+1)}(V_2^{(k+1)})^T.$$

We stress that in computing $V^{(k+1)}$ we accumulate two types of truncation errors: the one stemming from the already performed truncation of both $V^{(k)}$ and $P^{(k)}$, and that incurred in truncating the factors of $V^{(k+1)}$. The truncation procedure is applied to every iterate update inside the LSQR algorithm, so that all matrices are kept in factored form. Two parameters are used: the truncation tolerance (`tol` in the truncation criterion above), and the maximum allowed rank $r_{\max}$. In our experiments we focused on the latter value, while keeping `tol` small. A specific comment deserves the factorization step of the matrix after the operators are applied. More precisely,

$$\mathcal{L}(V_1 V_2^T) = \begin{bmatrix} AV_1 & CV_1 \end{bmatrix} \begin{bmatrix} B^T V_2 & D^T V_2 \end{bmatrix}^T = L_1 L_2^T;$$

similarly for $\mathcal{L}^T$. At each iteration k, by denoting $L_1^{(k)} = \left[AV_1^{(k)}, CV_1^{(k)}\right]$ and $L_2^{(k)} = \left[B^T V_2^{(k)}, D^T V_2^{(k)}\right]$, the update for the U iterate becomes:

$$U_1^{(k+1)} = \left[L_1^{(k)},\, -\alpha_k^{1/2} U_1^{(k)}\right], \qquad U_2^{(k+1)} = \left[L_2^{(k)},\, \alpha_k^{1/2} U_2^{(k)}\right].$$

The truncation strategy can be applied first to $L_1^{(k)}, L_2^{(k)}$ and subsequently to $U_1^{(k+1)}, U_2^{(k+1)}$, hence acting on two blocks at the time, or directly to the three block matrix $U_j^{(k+1)} = \left[AV_1^{(k)},\, CV_1^{(k)},\, \pm\alpha_k^{1/2} U_1^{(k)}\right]$, for $j = 1, 2$. The latter requires more temporary storage.

We conclude this section by recalling an important property of the trace, that makes the overall computation more accessible for low-rank matrices.

REMARK 4.1. *Given $Y, Z \in \mathbb{R}^{n\times m}$, $Y = Y_1 Y_2^T$, $Z = Z_1 Z_2^T$ with $Y_1, Z_1 \in \mathbb{R}^{n\times r}$ and $Y_2, Z_2 \in \mathbb{R}^{m\times r}$ and $r \ll n, m$, we have that*

$$\operatorname{trace}(YZ^T) = \operatorname{trace}((Y_1 Y_2^T)(Z_2 Z_1^T)) = \operatorname{trace}((Z_1^T Y_1)(Y_2^T Z_2)).$$

Clearly this is more efficient because both $Z_1^T Y_1$ and $Y_2^T Z_2$ are $r \times r$. This will be used in all instances in our experiments.

With this remark, we observe that $\beta = \|U\|_{\mathcal{F}}$ with $U = U_1 U_2^T$ changes into

$$\beta = (\operatorname{trace}((U_1^T U_1)(U_2^T U_2)))^{1/2}.$$

4.1. Stopping criteria. As already mentioned, truncation destroys the equivalence between the true residual norm and the reduced residual norm. As a consequence, if the true residual norm is meant to be used to check convergence, it requires an explicit computation, with a significant impact on the overall computational effort. The fact that $X^{(k)}$ is in factored form allows us to avoid the explicit computation of the large dense residual matrix, so that we can proceed as follows. First the residual factors are computed as

$$R_1^{(k+1)} = [F_1, -AX_1^{(k+1)}, -CX_1^{(k+1)}], \qquad R_2^{(k+1)} = [F_2, BX_2^{(k+1)}, DX_2^{(k+1)}],$$

then $\|R_k^{true}\|^2 = \operatorname{trace}(((R_1^{(k+1)})^T R_1^{(k+1)})((R_2^{(k+1)})^T R_2^{(k+1)}))$.

In the vector LSQR algorithm, the residual norm of the normal equation is also used as stopping criterion. Computing such a norm can be performed also in our setting, though the computational and memory costs of dealing with the corresponding residual factors can be significantly large, due to the presence of several terms in the equation (see the discussion for truncated CG in section 8).

An additional issue that needs to be taken into account is that the minimization of the true residual norm is no longer ensured. Hence, in the truncated case the residual norm can actually have a non-monotonic behavior. We have experienced this phenomenon in case the truncation rank was chosen too small. Although the residual norm can still decrease at later iterations, this behavior is a clear indication that the procedure has significantly departed from the full rank regime, so that a norm increase may be taken as a stopping criterion.

Algorithm 5.1 Fully three-term Truncation function

Require: Q_{Y_1}, S_Y, Q_{Y_2}, Q_{Z_1}, S_Z, Q_{Z_2}, tol, r

$F_{1,2} = Q_{Y_1}^T Q_{Z_1}$

$W = Q_{Z_1} - Q_{Y_1} F_{1,2}$

$[Q, F_{2,2}] = \text{QR}(W, 0)$

$Q_1 = \begin{bmatrix} Q_{Y_1}, Q \end{bmatrix}$ % Q matrix

$R_1 = \begin{bmatrix} I & F_{1,2} \\ 0 & F_{2,2} \end{bmatrix} \begin{bmatrix} S_Y & 0 \\ 0 & S_Z \end{bmatrix}$ % R matrix

$P_{1,2} = Q_{Y_2}^T Q_{Z_2}$

$W = Q_{Z_2} - Q_{Y_2} P_{1,2}$

$[\tilde{Q}, P_{2,2}] = \text{QR}(W, 0)$

$Q_2 = \begin{bmatrix} Q_{Y_2}, \tilde{Q} \end{bmatrix}$ % Q matrix

$R_2 = \begin{bmatrix} I & P_{1,2} \\ 0 & P_{2,2} \end{bmatrix}$ % R matrix

$[U, S, V] = \text{SVD}(R_1 R_2^T, 0)$ %thin SVD

$t = \text{find}(\text{cumsum}((\text{diag}(S))./\text{sum}(\text{diag}(S)) > 1 - \text{tol}, 1)$

$t = \min(r, t)$

$Q_{Y_1} = Q_1 U(:, 1:r), \quad Q_{Y_2} = Q_2 V(:, 1:r), \quad S = S(1:r, 1:r)$

5. A new three-term truncated factorized form. The truncation step is time consuming, and in the standard algorithm it is supposed to be performed at least four times per iteration. Even worse, it does not seem to take full advantage of the previous factorizations, except for appreciating the factors presence. The truncation step in fact generates two terms with orthonormal columns, and a diagonal term, that is, in the algorithm notation,

$$Y_1 Y_2^T = (Q_1 U_{:,1:r}) S (Q_2 V_{:,1:r})^T =: Q_{Y_1} S_Y Q_{Y_2}^T.$$

If the orthonormality of the two matrices is acknowledged, then it can be exploited in later computations. First of all, $\|Y_1 Y_2^T\|_{\mathcal{F}} = \|S_Y\|_{\mathcal{F}} = \|\text{diag}(S_Y)\|$, since S_Y is diagonal; this can be exploited for instance in the computation of β. Moreover, if two terms have this factored form, then their sum can be obtained more cheaply. Indeed,

$$\begin{aligned} V^{(k)} + P^{(k)} &= Q_{V_1}^{(k)} S_V (Q_{V_2}^{(k)})^T + Q_{P_1}^{(k)} S_P (Q_{P_2}^{(k)})^T \\ &= [Q_{V_1}^{(k)}, Q_{P_1}^{(k)}] \begin{bmatrix} S_V & \\ & S_P \end{bmatrix} [Q_{V_2}^{(k)}, Q_{P_2}^{(k)}]^T. \end{aligned}$$

The two tall blocks each have block-orthonormal columns, so that the orthogonalization step can be limited to one of the blocks, the smallest one whenever possible. This procedure is summarized in Algorithm 5.1. Note that the algorithm assumes that the factors of the first matrix, that is Q_{Y_1}, Q_{Y_2}, have orthonormal columns. We refer to Algorithm 5.2 for the actual call.

Algorithm 5.2 LSQR ALGORITHM (TRUNCATED VERSION)

Require: $\mathcal{L}$, $\mathcal{L}^T$, F_1, F_2, $\texttt{tol}$, $\texttt{tol\_tr}$, $\texttt{r}_{\max}$, imax

$\beta = \|F_1 F_2^T\|_F, \quad U_1 = F_1/\sqrt{\beta}, \quad U_2 = F_2/\sqrt{\beta}$

$[Q_{U_1}, R_{U_1}] = \text{QR}(U_1, 0), \quad [Q_{U_2}, R_{U_2}] = \text{QR}(U_2, 0), \quad r = \beta$

$[W_1, S_U, W_2] = \text{SVD}(R_{U_1} R_{U_2}^T, 0)$

$[V_1, V_2] = \mathcal{L}^T(U_1 W_1, U_2 W_2, S_U)$

$[Q_{V_1}, Q_{V_2}, S_V] = \text{Alg. 5.1}([\,], [\,], V_1, I, [\,], V_2, \texttt{tol\_tr}, \texttt{r}_{\max})$

$\alpha = \|\text{diag}(S_V)\|, \quad S_V = S_V/\alpha$

$\phi = \beta, \quad \rho = \alpha, \quad i = 0, \quad X_1 = 0, \quad X_2 = 0$

$Q_{D_1} = Q_{V_1}, \quad Q_{D_2} = Q_{V_2}, \quad S_D = S_V, \quad r_0 = \beta$

while $i < \text{imax}$, $\frac{r}{r_0} > \texttt{tol}$ **do**

$i = i + 1$

$[\tilde{U}_1, \tilde{U}_2] = \mathcal{L}(Q_{V_1}, Q_{V_2}, S_V)$

$[Q_{U_1}, Q_{U_2}, S_U] = \text{Alg. 5.1}(Q_{U_1}, -\alpha S_U, \tilde{U}_1, I, Q_{U_2}, \tilde{U}_2, \texttt{tol\_tr}, \texttt{r}_{\max})$

$\beta = \|\text{diag}(S_U)\|, \quad S_U = S_U/\beta$

$[\tilde{V}_1, \tilde{V}_2] = \mathcal{L}^T(Q_{U_1}, Q_{U_2}, S_U)$

$[Q_{V_1}, Q_{V_2}, S_V] = \text{Alg. 5.1}(Q_{V_1}, -\beta S_V, \tilde{V}_1, I, Q_{V_2}, \tilde{V}_2, \texttt{tol\_tr}, \texttt{r}_{\max})$

$\alpha = \|\text{diag}(S_V)\|, \quad S_V = S_V/\alpha$

$\rho_1 = (\rho^2 + \beta^2)^{1/2}, \quad c = \frac{\rho}{\rho_1}, \quad s = \frac{\beta}{\rho 1}$

$\theta = s\alpha, \quad \rho = -c\alpha, \quad \phi_1 = c\phi, \quad \phi = s\phi, \quad \xi_1 = \frac{\phi_1}{\rho_1}, \quad \xi_2 = \frac{\theta}{\rho_1}$

$[Q_{X_1}, Q_{X_2}, S_X] = \text{Alg.5.1}(Q_{X_1}, S_X, Q_{D_1}, \xi_1 S_D, Q_{X_2}, Q_{D_2}, \texttt{tol\_tr}, \texttt{r}_{\max})$

$[Q_{D_1}, Q_{D_2}, S_D] = \text{Alg. 5.1}(Q_{V_1}, S_V, Q_{D_1}, -\xi_2 S_D, Q_{V_2}, I, Q_{D_2}, I, \texttt{tol\_tr}, \texttt{r}_{\max})$

$r = \phi\alpha|c|$

check convergence

end while

I stands for an identity matrix of conforming size

$\mathcal{L}(P, Q, S)$ stands for $\mathcal{L}(PSQ^T)$, and $[\,]$ stands for an empty matrix

To ensure better orthogonality properties, the two Gram-Schmidt steps leading to the working matrix W are each done twice in the practical implementation [12]. Possible rank deficiency of the resulting block (matrix W in Algorithm 5.2) may be treated via rank revealing or SVD. In general, these procedures are more expensive, while rank deficiency is also taken care of in the subsequent thin SVD of $R_1 R_2^T$. The implementation of the truncation in the two bases deserves an explicit description. For instance, given a basis iterate $V^{(k)} = Q_{V_1} S_V Q_{V_2}^T$, $U^{(k)} = Q_{U_1} S_U Q_{U_2}^T$ in factored form, the next iterate $U^{(k+1)}$ is given by

$$
\begin{aligned}
U^{(k+1)} &= \mathcal{L}(Q_{V_1} S_V Q_{V_2}^T) - \alpha U^{(k)} \\
&= \begin{bmatrix} AQ_{V_1} & CQ_{V_1} \end{bmatrix} \begin{bmatrix} S_V & 0 \\ 0 & S_V \end{bmatrix} \begin{bmatrix} B^T Q_{V_2} & D^T Q_{V_2} \end{bmatrix}^T - \alpha Q_{U_1} S_U Q_{U_2}^T \\
&= [Q_{U_1},\, AQ_{V_1},\, CQ_{V_1}] \begin{bmatrix} -\alpha S_U & & \\ & S_V & \\ & & S_V \end{bmatrix} [Q_{U_2},\, AQ_{V_2},\, CQ_{V_2}]^T. \qquad (5.1)
\end{aligned}
$$

Hence, Algorithm 5.1 can be applied to the two blocks Q_{U_1} (with orthonormal columns) and $[AQ_{V_1}, CQ_{V_1}]$ and corresponding terms. A similar form can be deduced for the V recurrence. Note that Algorithm 5.1 is thus applied to larger blocks, where the second block no longer has orthonormal columns. Nonetheless, orthogonalization

is taken care of by Algorithm 5.1.

The final complete method with three-term factorized iterates is reported in Algorithm 5.2. The algorithm stores block iterates of size at most $n \times r$, where r is the maximum allowed rank. However, the iterate updates, the residual computation and the application of the operators necessitate temporary storage for at most $3r$ vectors at the time, see, e.g., the representation in (5.1). We also notice that the orthogonality associated with the U-recurrence involves vectors of length n, while that with the V-recurrence mostly uses m-length vectors. In case $m \ll n$, the two recurrences entail quite different costs.

6. Effects of truncation. In this section the effects of truncation in the LSQR algorithm are analyzed. The recurrences in the algorithm have different roles, and one could consider using different truncation rank and threshold for the different recurrences.

We recall that the recurrences $\{U^{(k)}\}$ and $\{V^{(k)}\}$ construct the biorthogonal approximation bases (in matrix form). Hence, their truncation modifies the space where the approximation is sought, yielding a "perturbed" space. Orthogonality among the basis vectors is also lost. However, as long as linear independence is maintained, the approximation process seems to continue nonetheless. This will be shown in Example 6.1. Similar properties have been shown in other "inexact" computations associated with the construction of Krylov subspace bases [39].

Due to truncation, the iterate $G^{(j)}$, $j = 1, \ldots, k$ no longer precisely accounts for each column of $\boldsymbol{V}_k \boldsymbol{R}_k^{-1}$, since on the one hand, the matrix version of $\boldsymbol{V}_k$ is truncated, and on the other hand the term $G^{(k)}$ is itself truncated. To cope with this double inaccuracy, one may want to enforce larger rank on this term, or a different update strategy. Finally, truncation strongly impacts the solution iterate; though this step does not influence any other computation, it possibly suffers from the most visible truncation side effect.

Let us linger over the loss of accuracy in the vector biorthogonalization relations (2.2). At each iteration, truncation of the matrix form can be thought of as a modification of the exact vector $\boldsymbol{u}_{i+1}^{(ex)}$ to give an approximation $\boldsymbol{u}_{i+1}$, that is

$$\boldsymbol{u}_{i+1} = \boldsymbol{A}\boldsymbol{v}_i - \alpha_i \boldsymbol{u}_i + \boldsymbol{\epsilon}_i.$$

Collecting these iterates, we obtain the inexact relation

$$\boldsymbol{A}\boldsymbol{V}_p = \boldsymbol{U}_{p+1}\boldsymbol{B}_p + \boldsymbol{E}_p, \qquad \boldsymbol{E}_p = [\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_p].$$

Note that none of the computed matrices is the same as if $\boldsymbol{E}_p$ were zero. Assume now truncation is only performed in the two bases, so that the vectors $\boldsymbol{d}_i$ and $\boldsymbol{x}_i$ are computed exactly with these bases vectors. Hence, the residual norm is written as

$$\begin{aligned} \|\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b}\| &= \|\boldsymbol{A}\boldsymbol{V}_k\boldsymbol{y}_k - \boldsymbol{U}_{k+1}\boldsymbol{e}_1\beta_0\| \\ &= \|\boldsymbol{U}_{k+1}\boldsymbol{B}_k\boldsymbol{y}_k - \boldsymbol{U}_{k+1}\boldsymbol{e}_1\beta_0 + \boldsymbol{E}_p\boldsymbol{y}_k\| \\ &\leq \|\boldsymbol{U}_{k+1}(\boldsymbol{B}_k\boldsymbol{y}_k - \boldsymbol{e}_1\beta_0)\| + \|\boldsymbol{E}_p\boldsymbol{y}_k\| \\ &\leq \|\boldsymbol{U}_{k+1}\|\,\|\boldsymbol{B}_k\boldsymbol{y}_k - \boldsymbol{e}_1\beta_0\| + \|\boldsymbol{E}_p\boldsymbol{y}_k\|. \end{aligned} \tag{6.1}$$

These inequalities show that the reduced problem solved by LSQR, that is

$$\min_{\boldsymbol{y}^{(k)}} \|\beta_1\boldsymbol{e}_1 - \boldsymbol{B}^{(k)}\boldsymbol{y}^{(k)}\|,$$

is no longer equivalent to solving the original least squares problem. The columns of $\boldsymbol{U}_{p+1}$ cease to be orthonormal, though the bound $\|\boldsymbol{U}_{p+1}\| \leq \sqrt{p+1}$ holds if the columns have unit norm. Moreover, the inexact biorthogonal relation adds a nonzero term involving $\boldsymbol{E}_k \boldsymbol{y}_k$, where the norm of each column of $\boldsymbol{E}_k$ is associated with the truncation threshold. In this case, and referring to previous related works (see, e.g., [39]), we can observe that

$$\|\boldsymbol{E}_k \boldsymbol{y}_k\| = \|\sum_{i=1}^{k} \boldsymbol{\epsilon}_i (\boldsymbol{y}_k)_i\| \leq \sum_{i=1}^{k} \|\boldsymbol{\epsilon}_i\| \, \|(\boldsymbol{y}_k)_i\|,$$

that is, the extra term $\|\boldsymbol{E}_k \boldsymbol{y}_k\|$ has low perturbation effect as long as each product $\|\boldsymbol{\epsilon}_i\| \, \|(\boldsymbol{y}_k)_i\|$ is small, and not both factors. In other words, if we expect the final residual to be small, we know that the components $\|(\boldsymbol{y}_k)_i\|$ will decrease with i, then the corresponding $\|\boldsymbol{\epsilon}_i\|$ are allowed to be large, thus maintaining the same magnitude of $\|\boldsymbol{E}_k \boldsymbol{y}_k\|$. We should keep in mind, however, that if the final residual is not small, $\|\boldsymbol{E}_k \boldsymbol{y}_k\| \approx \|\boldsymbol{E}_k\|$, with corresponding effects on the true residual norm after truncation. Denoting with $\boldsymbol{r}_k$ the true residual, (6.1) yields

$$\|\boldsymbol{r}_k\| \leq \sqrt{k+1} |\rho_k| + \|\boldsymbol{E}_p \boldsymbol{y}_k\|.$$

This bound shows that the true residual norm can actually increase, if the extra term has sizable magnitude. This property seems to be confirmed by our experiments.

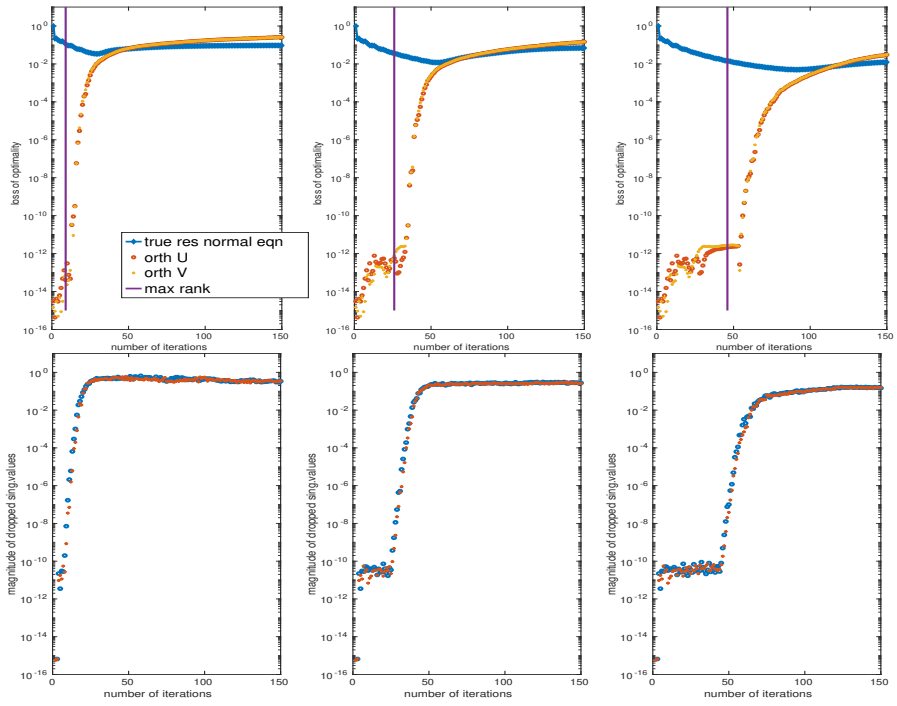

FIG. 6.1. *Example 6.1. Top: for truncation parameter $r = 50, 100, 150$ (left to right), true normal equation residual, angles $|\boldsymbol{u}_k^T \boldsymbol{u}_{k-1}|$, $|\boldsymbol{v}_k^T \boldsymbol{v}_{k-1}|$; the vertical line corresponds to the iteration at which the maximum rank is reached for either basis. Bottom: magnitude of first discarded singular value in the biorthogonal bases.*

To help our intuition on the loss of orthogonality we proceed with an experiment.

EXAMPLE 6.1. We consider the following Matlab reproducible data

```
n=300;   m=200;
A=toeplitz([3,-1,-1/2,zeros(1,n-3)],[3,1,zeros(1,m-2)]);   B=A';
C=toeplitz([-1,3,zeros(1,n-2)],[-1,1/2,-1, zeros(1,m-3)]);  D=C';
F1=ones(n,1);F2=ones(n,1);
```

We are interested in analyzing the loss of orthogonality in the two bases $\{\boldsymbol{u}_k\}$ and $\{\boldsymbol{v}_k\}$, caused by the rank truncation of the corresponding matrices. That is, at each iteration of Algorithm 5.2 we define $\boldsymbol{u}_k = Q_{U_1} S_U Q_{U_1}^T$ and similarly for $\boldsymbol{v}_k$, and compute their inner product. Results are reported in Figure 6.1. Both the residual norm of the normal equation and the local orthogonality of the bases are reported in the plots (Top), for different values of the truncation parameters ($r = 50, 100, 150$). The truncation threshold was set to 10^{-12}, so as to play a minimal role. We observe that the residual convergence degrades - the norm even increases - as soon as loss of orthogonality is so severe to reach the same magnitude as the residual norm. On the other hand, we notice that convergence is not significantly affected as long as the basis vectors remain well linearly independent, though no longer orthogonal. In the bottom plots the magnitude of the neglected singular values at truncation is also reported. We notice that this closely follows the pattern of the loss of orthogonality, as expected, however it sets in to a larger value, and consistently earlier than for the monitored basis inner products.

Finally, we report that we experimented with more ill conditioned matrices in the same family, as it is sufficient to increase the entry $1/2$ in C to make the Kronecker sum increasingly more badly behaved. However, the patterns shown in Figure 6.1 remained unvaried.

7. Sketched-LSQR. Randomization algorithms have become ubiquitous in numerical linear algebra (NLA) [27]. A way to incorporate randomized NLA strategies consists of embedding the iterates onto a significantly smaller space by means of a possibly *oblivious subspace*, see, e.g., [44]. In the context of minimum residual methods, and in particular in least squares problems, this procedure can be interpreted as using a semidefinite norm, see, e.g., [10],[35],[42]. This can be defined as minimizing the vector or matrix residual in the given semidefinite inner product, defined as $x^T W x$, with W symmetric and semidefinite [3]. Indefinite inner products have been analyzed in detail as a source of special properties [13], while semi-definite inner products have recently been used in signal processing, data science, eigenvalue computations, see, e.g., [18],[40], in addition to sketching-based projections [2].

Given a matrix A, whose columns span a low-dimensional subspace $\mathcal{V}$ of $\mathbb{R}^n$, an oblivious subspace embedding allows us to embed $\mathcal{V}$ into $\mathbb{R}^s$ with $s \ll n$, such that the norms are distorted as follows

$$\langle u, v\rangle - \epsilon\|u\|\|v\| \leq \langle \mathcal{S}(u), \mathcal{S}(v)\rangle \leq \langle u, v\rangle + \epsilon\|u\|\|v\|,$$

for all vectors $u, v \in \mathcal{V}$, where $\epsilon \in [0, 1)$ and $\mathcal{S} : \mathbb{R}^n \longrightarrow \mathbb{R}^s$. Clearly, if $u = v$ the condition becomes

$$(1-\epsilon)\|v\|^2 \leq \|\mathcal{S}(v)\|^2 \leq (1+\epsilon)\|v\|^2,$$

for all $v \in \mathcal{V}$. We will refer to $\mathcal{S}(\cdot)$ as the *sketching operator*. The term *oblivious* comes from the fact that the subspace $\mathcal{V}$ is not known a-priori. In this case the sketching operator is built using probabilistic methods requiring just the dimension of the subspace we want to embed and the target dimension s of the embedding space.

We are interested in applying sketching to make the computation of the iterate coefficients faster. Indeed, we cannot use sketching in order to embed the matrix iterates used at each iteration since we would not be able to retrieve the solution in the original space. In the context of iterative Krylov subspace methods for linear systems, this idea has been used by Babalanov, Grigori and collaborators, for instance.

The coefficients used in LSQR that can take advantage of this procedure are $\alpha = \|V\|_{\mathcal{F}}$ and $\beta = \|U\|_{\mathcal{F}}$. For U and V in factored form, that is $U = U_1 U_2^T$ and $V = V_1 V_2^T$, we have $\alpha = \text{trace}((V_1^T V_1)(V_2^T V_2))^{1/2}$ and $\beta = \text{trace}((U_1^T U_1)(U_2^T U_2))^{1/2}$. By defining the sketching operators $\mathcal{S}_1(\cdot) : \mathbb{R}^n \to \mathbb{R}^s$ and $\mathcal{S}_2(\cdot) : \mathbb{R}^m \to \mathbb{R}^s$, we can write

$$\begin{aligned} \alpha &= \text{trace}\left(\mathcal{S}_2(V_1)^T \mathcal{S}_2(V_1)\ \mathcal{S}_2(V_2)^T \mathcal{S}_2(V_2)\right), \\ \beta &= \text{trace}\left(\mathcal{S}_1(U_1)^T \mathcal{S}_1(U_1)\ \mathcal{S}_1(U_2)^T \mathcal{S}_1(U_2)\right). \end{aligned}$$

We aim to consider this procedure as an attempt to lower the computational costs of the version of LSQR with standard truncation. Indeed, given the matrices $U_1, U_2 \in \mathbb{R}^{n\times r}$ and $V_1, V_2 \in \mathbb{R}^{m\times r}$ with $r \ll n, m$, after applying the sketching operators we have $\mathcal{S}_1(U_1), \mathcal{S}_1(U_2) \in \mathbb{R}^{s_1\times r}$ and $\mathcal{S}_2(V_1), \mathcal{S}_2(V_2) \in \mathbb{R}^{s_2\times r}$, implying a cheaper matrix product operations, for $s_1, s_2 \ll n, m$. We expect this procedure not to be competitive with respect to the three-term form devised in section 5, which reduces the inner product to a vector norm of size n. In the next section we will experimentally show that this is indeed the case. A sketching strategy that involves more parts of the code may provide more significant benefits to the overall computational costs.

8. Numerical experiments. In this section we present some computational experiments, focusing on the comparison between the truncated versions of the LSQR and the CG algorithms, the latter applied to the normal equation in matrix-oriented form. The truncated CG method is taken from [23],[38]. We recall that in our context, truncated CG is applied to a matrix equation with operator $\mathcal{L}^T(\mathcal{L}(\cdot))$, which gives rise to a multiple term matrix equation. The residual associated with an approximate solution $X_1^{(k)}(X_2^{(k)})^T$ of rank r is given by $R^{(k)} = \mathcal{L}^T(F_1 F_2^T) - \mathcal{L}^T(\mathcal{L}(X_1^{(k)}(X_2^{(k)})^T))$, and it can still be factorized, although the number of vector allocations exceeds $4r$, which is more than the (at most) $3r$ required by LSQR. More precisely, we have

$$\begin{aligned} R^{(k)} &= A^T F_1 F_2^T D^T - E^T F_1 F_2^T B^T - A^T (A X_1^{(k)} (X_2^{(k)})^T D - E X_1^{(k)} (X_2^{(k)})^T B) D^T \\ &\quad - E^T (A X_1^{(k)} (X_2^{(k)})^T D + E X_1^{(k)} (X_2^{(k)})^T B) B^T \\ &= \begin{bmatrix} A^T F_1 & E^T F_1 & -A^T A X_1^{(k)} & -A^T E X_1^{(k)} & -E^T A X_1^{(k)} & -E^T E X_1^{(k)} \end{bmatrix} \\ &\quad \begin{bmatrix} D F_2 & B F_2 & D D^T X_2^{(k)} & D B^T X_2^{(k)} & B D^T X_2^{(k)} & B B^T X_2^{(k)} \end{bmatrix}^T \\ &= R_1^{(k)} (R_2^{(k)})^T. \end{aligned}$$

We notice that no preconditioning is used for CG. Although preconditioning may enhance convergence in terms of number of iterations, its effect on the iterates rank may be too negative. In addition, it is hard to devise a successful preconditioner due to the presence of several terms in the coefficient matrix; see, e.g., the analysis in [26].

The three-term factorization cannot be adapted to the CG algorithm. The idea of partitioning the QR decomposition is based on the fact that when we update a matrix, at least a block of it remains orthogonal. This is not happening in CG except for the updating of the matrix P. Furthermore, in LSQR this procedure makes sense

	Trunc CG			Trunc LSQR			Trunc S-LSQR		
m	# iter	CPU time	Final norm	# iter	CPU time	Final norm	# iter	CPU time	Final norm
1000	97	7.46	0.86	77	4.45	0.86	*550	-	-
1200	102	8.25	0.80	77	4.93	0.80	268	21.8	0.80
1400	107	9.36	0.71	81	5.48	0.71	130	10.3	0.71
1600	118	11.99	0.60	84	5.94	0.60	164	14.7	0.60
1800	168	37.01	0.43	82	6.34	0.43	145	16.2	0.43

* max allowed # iterations

TABLE 8.1

Example 8.1, $n = 2001$. Matrix C as in (8.1).

because in most cases the updated matrix is made of two blocks, consequently we have to orthogonalize just one block. Due to the use of the normal operator $\mathcal{L}^T(\mathcal{L}(\cdot))$, the updates in CG involve several terms, leading to a higher computational cost when orthogonalizing each block.

EXAMPLE 8.1. We consider an enlarged version of the problem in Example 6.1, with `A=toeplitz([3,-1,-1/2,zeros(1,n-3)],[3,1,zeros(1,m-2)]);`, $B = A^T$, $D = C^T$ and two different selections of C, that is

$$\texttt{C} = \texttt{toeplitz}([-1, 3, \texttt{zeros}(1, \texttt{n}-2)], [-1, 1/2, -1, \texttt{zeros}(1, \texttt{m}-3)]); \tag{8.1}$$

and

$$\texttt{C} = \texttt{toeplitz}([-1, 3, \texttt{zeros}(1, \texttt{n}-2)], [-1, 2, -1, \texttt{zeros}(1, \texttt{m}-3)]); \tag{8.2}$$

We consider $n = 2001$ and $m = 1000, 1200, \ldots, 1800$, leading to a coefficient matrix in Kronecker form of size $4 \cdot 10^6 \times 3.2 \cdot 10^6$. The maximum rank for truncation is set to $r = 100$, and the truncation threshold is equal to 10^{-12}. All methods use the explicitly computed *true* matrix residual norm, and the stopping criterion is

$$\frac{\left| \|R_k^{true}\| - \|R_{k-1}^{true}\| \right|}{\|R_k^{true}\|} \leq 10^{-9},$$

and we remark that monotonic behavior of the residual norm was observed for all methods on this problem. We compare the performance of truncated CG applied to the normal equation, as implemented in [24],[38], with that of truncated LSQR (Algorithm 5.2). Results are shown in Table 8.1 (in Table 8.2) for C defined in (8.1) (in (8.2)). For completeness we also report results with the sketched LSQR method described in section 7, with sketched space of dimension $s = 200$. Both tables report the number of iterations required to satisfy the criterion, the CPU time in seconds, and the achieved final residual norm. Although CG and LSQR are mathematically equivalent, they are not so in their truncated variant, and the number of iterations may change significantly, with Truncated CG paying a high toll. The CPU time changes accordingly. Nonetheless, the final residual remains the same for both methods. Also the sketched approach reaches the final residual in all instances, with CPU time lower than for CG, but the number of iterations is affected. We explicitly observe the non-convergence anomaly of the sketched method for $m = 1000$ with C as in (8.1). Somewhat similar comments apply for the second choice of C, although on the one hand, number of iterations and CPU times are significantly lower, and on the other

	Trunc CG			Trunc LSQR			Trunc S-LSQR		
m	# iter	CPU time	Final norm	# iter	CPU time	Final norm	# iter	CPU time	Final norm
1000	41	2.10	0.86	42	1.87	0.86	47	2.64	0.86
1200	43	2.39	0.80	44	2.05	0.80	138	8.30	0.80
1400	46	2.89	0.71	47	2.49	0.71	58	3.78	0.71
1600	68	5.66	0.60	65	4.32	0.60	130	10.10	0.60
1800	79	7.42	0.43	70	4.72	0.43	110	9.21	0.43

TABLE 8.2
Example 8.1, $n = 2001$. Matrix C as in (8.2).

hand, sketched LSQR is less competitive. Notice that the final residual norm has the same first digits as for the other case, although the dynamics of the methods is quite different for the two choices of C. To deepen our understanding of this phenomenon, we considered a problem of smaller size: the solutions to the two vectorized problems were explicitly computed, and we noticed that the distance between the solutions was sizable, and that the singular values displayed similar decays in the two cases. Hence the different performance for the two different choices of C does not seem to be due to the rank properties of the two solutions, but on how the approximation process proceeds.

EXAMPLE 8.2. The second example is a building block in the solution of large scale Lyapunov equations by means of Petrov-Galerkin projection methods. The computational treatment of the least squares problem resulting after projection was first discussed in [17] and then further analyzed in detail in [26]. The problem comes from approximating the solution to the Lyapunov equation $AX + XA^T - F_1F_1^T = 0$ as $X \approx X_m = \mathcal{V}_m Y_m \mathcal{V}_m^T$, where the columns of $\mathcal{V}_m$ are an orthonormal basis for the block Krylov subspace $\mathcal{K}_m(A, F_1)$ and Y_m is to be determined. Here and in the following A is square and we assume that F_1 has column rank p. A Petrov-Galerkin method with test space $A\mathcal{K}_m(A, F_1)$ corresponds to a minimization problem in the Frobenius norm [17], so that the solution Y_m is obtained as

$$Y_m = \arg \min_{Y_m \in \mathbb{R}^{m\times m}} \|A\mathcal{V}_m Y_m \mathcal{V}_m^T + \mathcal{V}_m Y_m \mathcal{V}_m^T A^T - F_1F_1^T\|_{\mathcal{F}}. \tag{8.3}$$

The problem (8.3) can be simplified by using the standard Arnoldi relation

$$A\mathcal{V}_m = \tilde{\mathcal{V}}_{m+1}\underline{H}_m, \qquad \text{with} \qquad \tilde{\mathcal{V}}_{m+1} = [V_1, \ldots, V_m, \tilde{V}_{m+1}],$$

where $\tilde{V}_{m+1} \in \mathbb{R}^{n\times p}$, has orthonormal columns, and $\underline{H}_m \in \mathbb{R}^{(m+1)p+\times mp}$. Then the minimization problem (8.3) can be written as

$$Y_{mp} = \arg \min_{Y\in\mathbb{R}^{mp\times mp}} \|\underline{H}_m Y \underline{I}^T + \underline{I} Y \underline{H}_m^T - \begin{bmatrix} \widetilde{F}_1\widetilde{F}_1^T & 0 \\ 0 & 0 \end{bmatrix} \|_{\mathcal{F}}, \tag{8.4}$$

where $\underline{I} = \begin{bmatrix} I_{mp} \\ 0 \end{bmatrix} \in \mathbb{R}^{(m+1)p\times mp}$ and $F_1 = V_1\widetilde{F}_1$. This formulation precisely corresponds to our setting in (1.2).

For this example, A stems from a five-point stencil finite difference discretization of the operator $-\Delta u + 10xu_x + 10yu_y$ on the unit square, and the zero boundary conditions of the associated partial differential equation. The right-hand side is obtained

		Trunc CG		Trunc LSQR	
m	r	Final norm	CPU time	Final norm	CPU time
600	50	6.5241e-01	7.46	1.1832e+00	4.60
600	75	6.5089e-01	14.00	7.7435e-01	8.50
600	100	6.4893e-01	23.17	6.5080e-01	14.64
600	125	6.4806e-01	33.96	6.4781e-01	23.39
600	150	6.4736e-01	66.20	6.4733e-01	33.26
800	50	6.4857e-01	11.47	1.2665e+00	10.34
800	75	6.5082e-01	32.88	7.6861e-01	12.74
800	100	6.4922e-01	39.83	6.5026e-01	27.25
800	125	6.4811e-01	60.30	6.4785e-01	31.37
800	150	6.4741e-01	78.03	6.4731e-01	60.68
1000	50	6.5178e-01	15.19	1.2738e+00	10.02
1000	75	6.5111e-01	39.77	7.7703e-01	16.61
1000	100	6.4939e-01	44.11	6.5106e-01	30.11
1000	125	6.4808e-01	69.24	6.4781e-01	33.06
1000	150	6.4763e-01	93.23	6.4732e-01	56.54

TABLE 8.3

Example 8.2, $n = m + 1$. Truncated CG uses matrices with at most $4r$ columns; Truncated LSQR at most $2r$.

with F_1 a vector (so that $p = 1$) with random entries uniformly distributed in the interval $(0, 1)$. We compare truncated CG applied to the matrix normal equation and truncated LSQR for solving the reduced problem (8.4) as the Krylov subspace grows. More precisely, we fix a discretization dimension so that A has size 14400×14400, and then generate the Krylov space of dimension up to $m_{\max} = 1000$. We report the performance of the considered methods in solving the inner least squares problem for $m = 600, 800, 1000$ and with varying truncation rank (and truncation threshold 10^{-10}), while we fix the maximum number of iterations to 300. Results are reported in Table 8.3, where we display the final true residual norm and the total CPU time. This norm is computed for both methods at each iteration. We remark that convergence is very slow, and that truncation takes place quite early in the iteration for both methods. However, as already mentioned, truncated CG uses twice the amount of memory as LSQR for the explicit computation of the residual matrix factors. This memory unbalance shows that we should in general compare the performance of CG with that of LSQR using a larger truncation rank - though not necessarily twice the size to limit computational costs. Disregarding this possibility, Table 8.3 shows that LSQR outperforms CG in terms of CPU time in all cases, for the same number of iterations, and less memory. On the other hand, we observe that if the truncation parameter r is too small ($r = 50$), the LSQR final residual norm is unsatisfactory. What happens in practice is that at some point the residual norm starts to slowly increase again. An informed stopping criterion could take into account this behavior to determine the final attainable accuracy.

9. Application to Dictionary Learning. Dictionary Learning is a branch of machine learning and signal processing that, given input data $\boldsymbol{f} \in \mathbb{R}^n$, aims to find a dictionary $\boldsymbol{\mathcal{D}}^{n\times m}$ and a sparse representation $\boldsymbol{x} \in \mathbb{R}^m$, such that $\boldsymbol{f} \approx \boldsymbol{\mathcal{D}x}$. The dictionary $\boldsymbol{\mathcal{D}}$ can be either underdetermined (large) or overdetermined (tall), and

collects samples of objects from different classes. The problem can be represented as

$$(\boldsymbol{\mathcal{D}}, \boldsymbol{x}) = \arg\min_{\boldsymbol{\mathcal{D}}, \boldsymbol{x}} \|\boldsymbol{f} - \boldsymbol{\mathcal{D}}\boldsymbol{x}\|_{\mathcal{F}}^2.$$

We are interested in exploring the dictionary learning framework in the case that $\boldsymbol{\mathcal{D}}$ is a *given* Kronecker sum of sub-dictionaries, as proposed in [7],[8]. More precisely, we consider

$$\boldsymbol{\mathcal{D}} = B^T \otimes A + D^T \otimes C,$$

with A, B^T, C and D^T tall matrices. The case of more than two addends can be treated similarly. We readily recognize that by setting $\boldsymbol{x} = \mathrm{vec}(X)$ and $\boldsymbol{f} = \mathrm{vec}(F)$, the problem fits our setting with

$$F \approx AXB + CXD.$$

We use this formulation for classification purposes: given a fixed dictionary $\boldsymbol{\mathcal{D}}$ and a new image $\boldsymbol{f}$, we look for a solution $\boldsymbol{x}$ that allows us to recognize the class the new image belongs to. To this end, we switch to the more convenient matrix form, due to the Kronecker structure of the dictionary matrix. We will see that the matrix formulation makes the classification step very natural.

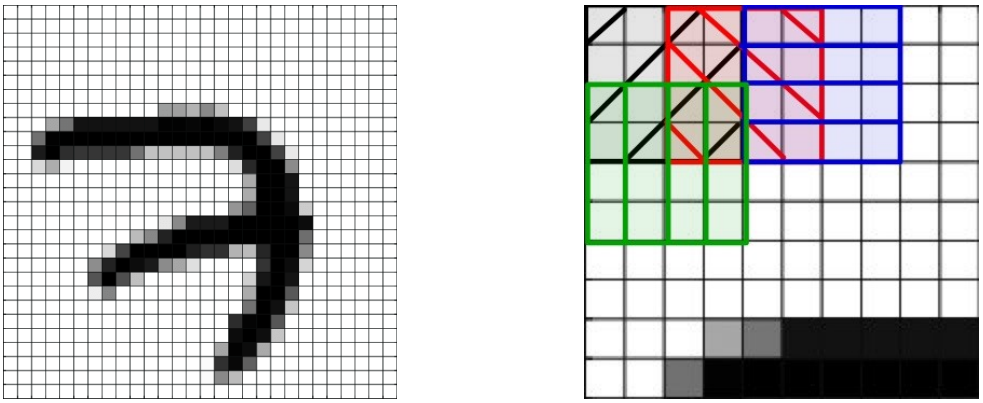

FIG. 9.1. *Left: example of an image in MNIST. Right: example of how patches are built.*

We experiment with this data science framework by using the benchmark image dataset MNIST [25] of handwritten digits. The dataset is made of a training sample set of about 60000 images and a test set of about 10000 images. Each image has size 28×28 pixels.

In the following we introduce a (possibly new) way to generate the multiterm dictionary by using a rich sample of images from the training set of all digits. Each matrix A to D^T collects a randomly chosen sample of 200 images for each considered digit from 0 to 9, so that all four (different) coefficient matrices have $m = 2000$ columns. To determine the content of each matrix column, we split each 28×28 training image into 4×4 overlapping patches, starting from the top-left corner moving towards the right with step two, and then using lexicographic ordering so as to cover the whole image. An instance of the patching scheme is displayed in Figure 9.1. Overall, this gives 169 patches for the processed image.

We then collect the vectorization of all these 169 image patches as a single vector. Since each patch has a total of 16 pixels, the resulting column has dimension 2704.

The right-hand side is built in the same way, by patching and then vectorizing a single image to be classified.

Summarizing, $A, B^T, C, D^T \in \mathbb{R}^{2704\times 2000}$ and $X \in \mathbb{R}^{2000\times 2000}$. By defining the right-hand side $F = ff^T$ with $f \in \mathbb{R}^{2704}$ built as explained, we need to solve

$$\min_X \|AXB + CXD - ff^T\|_{\mathcal{F}}^2.$$

We have run our new method to classify three different test images, corresponding to the digits 1, 2 and 7 (see Figure 9.2-9.4). In the experiments we have used a truncation tolerance $\texttt{tol\_tr} = 10^{-12}$ and maximum rank after truncation $r = 100$, while for the stopping criteria we have used maximum number of iterations max_it = 350 and tolerance tol $= 10^{-8}$, as described in section 4.1. The solution obtained with the methods discussed in this paper is not generally sparse. However, as it is common in other sparsity-promoting formulations, threshold-based sparsification can be performed a-posteriori. In all figures (left plots) we report the largest elements in X in absolute value. The sparsity threshold depends on the choice of the right-hand side; experimentally a good choice that allows the user to appreciate the result is usually of the order $\mathcal{O}(10^{-4} - 10^{-3})$. Recalling that X is multiplied by the left and by the right by dictionaries, e.g., AXB, we realize that each block of rows in X is multiplied by the column block (specific digit sample) in the matrix A. The same for the columns of X. We readily see that the largest elements in X capture the correct portion of the dictionary partition.

Digit to be classified	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
Test '1'	1.5	**12.6**	2.4	1.9	2.5	1.1	2.4	1.7	3.2	1.2
Test '2'	5.4	3.3	**48.5**	10.7	5.2	5.6	8.4	3.5	4.9	2.7
Test '7'	4.3	1.8	4.6	3.7	5.4	5.3	3.0	**24.9**	4.8	7.8

TABLE 9.1
Norm$\times 10^{-3}$ of each diagonal block of X, associated with a digit from 0 to 9.

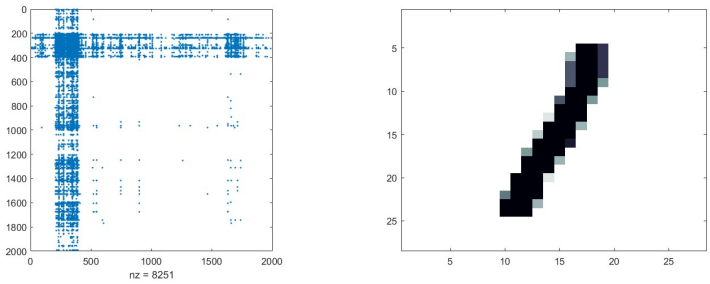

FIG. 9.2. *Left: most significant elements of the solution. Right: image of the digit* 1 *used to build the right-hand side.*

To provide a more automatic classification procedure, we propose to evaluate the norm of the diagonal 200×200 blocks in X. The largest norm will provide the best-fit class among the digits. The left plots in Figure 9.2-9.4 already illustrate that the highest density of largest values is indeed on the diagonal blocks. We report in Table 9.1 the Frobenius norm of the solution blocks for each considered image to be

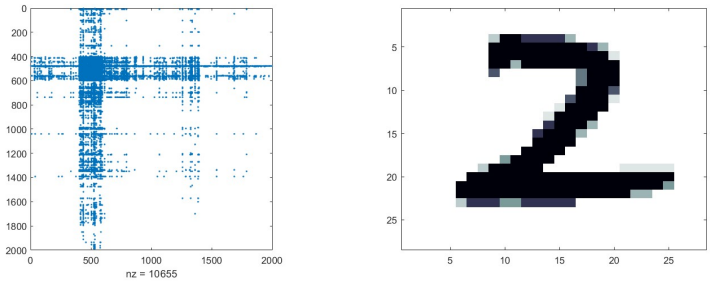

FIG. 9.3. *Left: most significant elements of the solution. Right: image of the digit* 2 *used to build the right-hand side.*

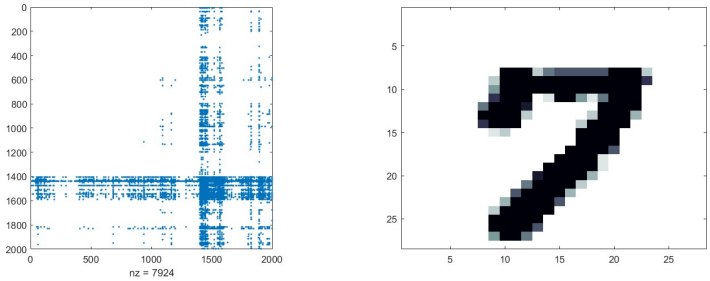

FIG. 9.4. *Left: most significant elements of the solution* X*. Right: image of the digit* 7 *used to build the right-hand side.*

classified. The norm of the "correct" digit is in most cases one order of magnitude larger than for the other ones, resulting in a particularly satisfactory strategy.

Although X is in general not sparse, a-posterior sparsification does provide the sought after structure. Moreover, the obtained X has low rank, which is another common way to enforce sparsity. The norm of the diagonal blocks in the solution appears to contain the key information. In particular, we notice that to compute these norms, the solution X does not need to be explicitly formed, since this information can be obtained by simply multiplying the corresponding blocks of the two factors.

For completeness, in Table 9.2 we report the performance of the two analyzed methods. For this example a fixed number of iterations was used, and a maximum rank $r = 100$ was enforced. The results are consistent with those in the previous examples.

10. Conclusions and outlook. We have presented a new matrix oriented version of LSQR, with a new implementation of the truncation strategy for enforcing low rank in all iterates. The computational results show that the method maintains a good performance when compared to matrix-oriented and truncated CG on the associated normal matrix equation. The derived sketched LSQR procedure, although well behaved compared to CG, does not compete with the effectiveness of the new version of LSQR.

Several problems remain open. Among them, is the generalization to problems with more than to terms, and the implementation of a tensorized version for tensor least squares problems. In terms of further enhancing strategies, different randomized techniques could be implemented, such as using a randomized SVD approach during

			Trunc CG		Trunc LSQR	
Digit to be classified	r	iters	Final norm	CPU time	Final norm	CPU time
Test '1'	100	350	5.6031e-02	165.07	4.4297e-02	97.17
Test '2'	100	350	9.1144e-02	170.59	7.6902e-02	94.35
Test '7'	100	350	6.2586e-02	162.47	5.1792e-02	95.15

TABLE 9.2
Final true residual norms and CPU times for the examples in the Figures 9.2-9.4

the truncation steps at each iteration.

Acknowledgements. Part this work was funded by the European Union - NextGenerationEU under the National Recovery and Resilience Plan (PNRR) - Mission 4 Education and research - Component 2 From research to business - Investment 1.1 Notice Prin 2022 - DD N. 104 of 2/2/2022, entitled "Low-rank Structures and Numerical Methods in Matrix and Tensor Computations and their Application", code 20227PCCKZ – CUP J53D23003620006. The work of the first author is funded by the European Union under the National Recovery and Resilience Plan (PNRR) - Mission 4 - Component 2 Investment 1.4 "Strengthening research structures and creation of "National R&D Champions" on some Key Enabling Technologies" DD N. 3138 of 12/16/2021 rectified with DD N. 3175 of 18/12/2021, code CN00000013 - CUP J33C22001170001.

REFERENCES

[1] J. Baker, M. Embree, and J. Sabino, *Fast singular value decay for Lyapunov solutions with nonnormal coefficients*, SIAM J. Matrix Analysis and Applications, 36 (2015), p. 656–668.
[2] O. Balabanov and L. Grigori, *Randomized block Gram–Schmidt process for solution of linear systems and eigenvalue problems*, arXiv preprint arXiv:2111.14641, (2021).
[3] ———, *Randomized Gram–Schmidt process with application to GMRES*, SIAM J. Sci. Comput., 44 (2022), pp. A1450–A1474.
[4] S. Bellavia, J. Gondzio, and M. Porcelli, *A relaxed interior point method for low-rank semidefinite programming problems with applications to matrix completion*, J. of Scientific Computing, 89 (2021).
[5] A. H. Bentbib, A. Khouia, and H. Sadok, *The LSQR method for solving tensor least-squares problems*, Electron. Trans. Numer. Anal., 55 (2022), pp. 92–111.
[6] M. Bollhöfer and A. K. Eppler, *A structure preserving FGMRES method for solving large Lyapunov equations*, in Progress in Industrial Mathematics at ECMI 2010, Mathematics in Industry, M. Günther, A. Bartel, M. Brunk, S. Schoeps, and M. Striebel, eds., vol. 17, 2012, pp. 131–136.
[7] C. F. Dantas, J. E. Cohen, and R. Gribonval, *Learning tensor-structured dictionaries with application to hyperspectral image denoising*, in 27th European Signal Processing Conference, EUSIPCO 2019, A Coruña, Spain, September 2-6, 2019, IEEE, 2019, pp. 1–5.
[8] C. F. Dantas, M. N. Da Costa, and R. da Rocha Lopes, *Learning dictionaries as a sum of Kronecker products*, IEEE Signal Processing Letters, 24 (2017), pp. 559–563.
[9] W. Deng, X. Zeng, and Y. Hong, *Distributed optimisation approach to least-squares solution of Sylvester equations*, IET Control Theory & Applications, 14 (2020), pp. 2968–2976.
[10] A. Eshragh, O. D. Pietro, and M. A. Saunders, *Toeplitz least squares problems, fast algorithms and big data*, tech. rep., arXiv: 2112.12994, 2021.
[11] D. W. Fausett and H. Hashish, *Overview of QR Methods for Large Least Squares Problems Involving Kronecker Products*, De Gruyter, Berlin, Boston, 1995, pp. 71–80.
[12] L. Giraud, J. Langou, and M. Rozloznik, *The loss of orthogonality in the Gram-Schmidt orthogonalization process*, Computers and Mathematics with Applications, 50 (2005), pp. 1069–1075.
[13] I. Gohberg, P. Lancaster, and L. Rodman, *Matrices and indefinite scalar products*, in Operator theory: advances and applications, vol. 8, Birkhäuser, 1983.
[14] G. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, J. Society for Industrial and Applied Mathematics, Series B: Numerical Analysis, 2 (1965),

pp. 205–224.
[15] L. Grubišić and D. Kressner, *On the eigenvalue decay of solutions to operator Lyapunov equations*, Systems & Control Letters, 73 (2014), pp. 42–47.
[16] R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*, Cambridge University Press, Cambridge, 1991.
[17] D. Y. Hu and L. Reichel, *Krylov Subspace Methods for the Sylvester Equation*, Linear Algebra and Appl., 172 (1992), pp. 283–313.
[18] M. Ishteva, K. Usevich, and A. Markovsky, *Factorization approach to structured low-rank approximation with applications*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 1180–1204.
[19] J. Kamm and J. Nagy, *Optimal Kronecker product approximation of block toeplitz matrices*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 155–172.
[20] A. Kittisopaporn and P. Chansangiam, *Approximated least-squares solutions of a generalized sylvester-transpose matrix equation via gradient-descent iterative algorithm*, Advances in Difference equations, 2021 (2021).
[21] D. Kressner, M. Plešinger, and C. Tobler, *A preconditioned low-rank CG method for parameter-dependent Lyapunov equations*, Num. Lin. Alg. Appl., 21 (2014), pp. 666–684.
[22] D. Kressner and P. Sirković, *Truncated low-rank methods for solving general linear matrix equations*, Num. Lin. Alg. Appl., 22 (2015), p. 564–583.
[23] D. Kressner and C. Tobler, *Krylov subspace methods for linear systems with tensor product structure*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 1688–1714.
[24] D. Kressner and C. Tobler, *Low-rank tensor Krylov subspace methods for parametrized linear systems*, SIAM. J. Matrix Anal. & Appl., 32 (2011), pp. 1288–1316.
[25] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.
[26] Y. Lin and V. Simoncini, *Minimal residual methods for large scale Lyapunov equations*, Applied Numerical Mathematics, 72 (2013), pp. 52–71.
[27] P.-G. Martinsson and J. A. Tropp, *Randomized numerical linear algebra: Foundations and algorithms*, Acta Numerica, 29 (2020), p. 403–572.
[28] The MathWorks, Inc., *MATLAB 7*, r2020b ed., 2020.
[29] C. C. Paige and M. A. Saunders, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Transactions on Mathematical Software (TOMS), 8 (1982), pp. 43–71.
[30] D. Palitta and P. Kürschner, *On the convergence of Krylov methods with low-rank truncations*, Numerical Algorithms, 88 (2021), pp. 1383–1417.
[31] Z.-Y. Peng, *A matrix LSQR iterative method to solve matrix equation AXB=C*, Int'l J. of Computer Mathematics, 87 (2010), pp. 1820–1830.
[32] T. Penzl, *Eigenvalue decay bounds for solutions of Lyapunov equations: the symmetric case*, Systems and Control Letters, 40 (2000), pp. 139–144.
[33] L. Piccinini, *Least squares methods for Sylvester-like linear matrix equations*, 2023. Master Thesis.
[34] R. E. Platero, *Least squares updating for Kronecker products*, 2017. Bachelor Thesis.
[35] V. Rokhlin and M. Tygert, *A fast randomized algorithm for overdetermined linear least-squares regression*, Proc. Natl. Acad. Sci. USA, 105 (2008), pp. 13212–13217.
[36] J. Sabino, *Solution of Large-Scale Lyapunov Equations via the Block Modified Smith Method*, PhD thesis, Rice University, 2006.
[37] V. Simoncini, *Computational methods for linear matrix equations*, SIAM Review, 58 (2016), pp. 377–441.
[38] V. Simoncini and Y. Hao, *Analysis of the truncated conjugate gradient method for linear matrix equations*, SIAM J. Matrix Anal. and Appl., 44 (2023), pp. 359–381.
[39] V. Simoncini and D. B. Szyld, *Theory of inexact Krylov subspace methods and applications to scientific computing*, SIAM J. Sci. Comput., 25 (2003), pp. 454–477.
[40] G. W. Stewart, *On the semidefinite B-Arnoldi method*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1458–1468.
[41] M. Stoll and T. Breiten, *A low-rank in time approach to PDE-constrained optimization*, SIAM J. Sci. Comput., 37 (2015), pp. B1–B29.
[42] E. Timsit, L. Grigori, and O. Balabanov, *Randomized orthogonal projection methods for Krylov subspace solvers*, arXiv preprint arXiv:2302.07466, (2023).
[43] M. Wang, M. Wei, and Y. Feng, *An iterative algorithm for a least squares solution of a matrix equation*, International Journal of Computer Mathematics, 87 (2010), pp. 1289–1298.
[44] D. P. Woodruff, *Computational advertising: Techniques for targeting relevant ads*, Foundations and Trends® in Theoretical Computer Science, 10 (2014), p. 1–157.