A Subspace-Conjugate Gradient Method for Linear Matrix Equations

A SUBSPACE-CONJUGATE GRADIENT METHOD FOR LINEAR MATRIX EQUATIONS*

DAVIDE PALITTA†, MARTINA IANNACITO† , AND VALERIA SIMONCINI†‡

Abstract. The efficient solution of large-scale multiterm linear matrix equations is a challenging task in numerical linear algebra, and it is a largely open problem. We propose a new iterative scheme for symmetric and positive definite operators, significantly advancing methods such as truncated matrix-oriented Conjugate Gradients (CG). The new algorithm capitalizes on the low-rank matrix format of its iterates by fully exploiting the subspace information of the factors as iterations proceed. The approach implicitly relies on orthogonality conditions imposed over much larger subspaces than in CG, unveiling insightful connections with subspace projection methods. The new method is also equipped with memory-saving strategies. In particular, we show that for a given matrix $\boldsymbol{Y}$, the action $\mathcal{L}(\boldsymbol{Y})$ in low rank format may not be evaluated exactly due to memory constraints. This problem is often underestimated, though it will eventually produce Out-of-Memory breakdowns for a sufficiently large number of terms. We propose an ad-hoc randomized range-finding strategy that appears to fully resolve this shortcoming.

Experimental results with typical application problems illustrate the potential of our approach over various methods developed in the recent literature.

Key words. Multiterm matrix equations, conjugate gradient, Galerkin condition.

MSC codes. 65F45, 65F25, 65F99

1. Introduction.

We are interested in the numerical solution of the problem

$$\boldsymbol{A}_1\boldsymbol{X}\boldsymbol{B}_1 + \ldots + \boldsymbol{A}_\ell\boldsymbol{X}\boldsymbol{B}_\ell = \boldsymbol{C}, \tag{1.1}$$

where $\boldsymbol{A}_i \in \mathbb{R}^{n_A\times n_A}$, $\boldsymbol{B}_i \in \mathbb{R}^{n_B\times n_B}$, $i = 1,\ldots,\ell$, are symmetric matrices, and $\boldsymbol{C} \in \mathbb{R}^{n_A\times n_B}$ has rank $s_C \ll \min\{n_A, n_B\}$ so that we can write $\boldsymbol{C} = C_1C_2^{\mathrm{T}}$, $C_1 \in \mathbb{R}^{n_A\times s_C}$, $C_2 \in \mathbb{R}^{n_B\times s_C}$. By introducing the linear operator $\mathcal{L}(\boldsymbol{X}) = \boldsymbol{A}_1\boldsymbol{X}\boldsymbol{B}_1 + \ldots + \boldsymbol{A}_\ell\boldsymbol{X}\boldsymbol{B}_\ell$, in the following we will use the more compact notation

$$\mathcal{L}(\boldsymbol{X}) = \boldsymbol{C}$$

for the matrix equation above. We assume that $\mathcal{L}$ is positive definite in the matrix inner product, that is it holds that $\langle \boldsymbol{X}, \mathcal{L}(\boldsymbol{X})\rangle > 0$ for any nonzero $\boldsymbol{X} \in \mathbb{R}^{n_A\times n_B}$. Given two $n \times m$ matrices $\boldsymbol{X}, \boldsymbol{Y}$, we define the matrix inner product as

$$\langle \boldsymbol{X}, \boldsymbol{Y}\rangle = \mathrm{trace}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}).$$

This inner product defines the Frobenius norm $\|\boldsymbol{X}\|_F^2 = \langle \boldsymbol{X}, \boldsymbol{X}\rangle$.

Encountered samples of $\mathcal{L}$ include for instance the generalized Lyapunov equation $\mathcal{L}(\boldsymbol{X}) = \boldsymbol{A}\boldsymbol{X}\boldsymbol{E}^{\mathrm{T}} + \boldsymbol{E}\boldsymbol{X}\boldsymbol{A}^{\mathrm{T}}$ and its multiterm counterpart $\mathcal{L}(\boldsymbol{X}) = \boldsymbol{A}\boldsymbol{X}\boldsymbol{E}^{\mathrm{T}} + \boldsymbol{E}\boldsymbol{X}\boldsymbol{A}^{\mathrm{T}} + \boldsymbol{M}\boldsymbol{X}\boldsymbol{M}^{\mathrm{T}}$ with $\boldsymbol{A}, \boldsymbol{E}, \boldsymbol{M} \in \mathbb{R}^{n\times n}$, as they occur in control theory [5],[4, Ch.6]; in our setting we have the additional hypothesis that all coefficient matrices are symmetric. In the following we will call a multiterm Lyapunov equation an equation as in (1.1) where $\boldsymbol{C}$ is symmetric and $\mathcal{L}$ is such that $(\mathcal{L}(\boldsymbol{X}))^{\mathrm{T}} = \mathcal{L}(\boldsymbol{X})$ for any symmetric $\boldsymbol{X}$. This implies that the solution to a multiterm Lyapunov equation is symmetric.

*Version of July 16, 2025

†Dipartimento di Matematica and (AM)^2, Alma Mater Studiorum Università di Bologna, Piazza di Porta San Donato 5, I-40127 Bologna, Italy, {davide.palitta}{martina.iannacito}{valeria.simoncini}@unibo.it

‡IMATI-CNR, Pavia, Italy.

More general forms typically arise whenever the terms on the left and right of the unknown have different meaning in the original application, such as geometric space vs time (see, e.g., [15],[20]), or geometric space vs parameter space (e.g., [27],[7]), or optimization (e.g., [10],[32]). We will refer to this general latter structure as multiterm Sylvester equation*.

Many different solid methods for the solution of equation (1.1) for $\ell = 2$ have been devised in the past two decades (see, e.g., the survey [30]). On the other hand, having a number of terms $\ell > 2$ in (1.1) makes the numerical treatment of this equation extremely challenging. Fewer options are available in the literature for medium up to large dimensions of the coefficient matrices. In particular, to the best of our knowledge, no decomposition-based method able to simultaneously triangularize $\ell > 2$ generic matrices $\boldsymbol{A}_i$'s, $\boldsymbol{B}_i$'s is available in the literature so that, up to date, recasting the problem in terms of its Kronecker form is the only option to get a direct solution. However, this strategy suffers from excessive memory constraints and computational cost even for moderate dimensions of the coefficient matrices, so that only iterative procedures for the solution of (1.1) are being explored. Among the classes of contributions in this direction are matrix-oriented Krylov methods with low-rank truncations [19],[32],[31],[23], projection methods tailored to the equation at hand [16],[31],[27], fixed-point iterations [9],[28], Riemannian optimization schemes [8], and greedy procedures [18].

In the following we focus our attention on short recurrences associated with matrix-oriented Krylov methods. These schemes amount to adapting standard Krylov schemes for linear systems to matrix equations by leveraging the equivalence between (1.1) and its Kronecker form. Thanks to our hypotheses on $\mathcal{L}$, the coefficient matrix of the linear system in Kronecker form is symmetric positive definite so that the Conjugate Gradient method (CG) can be applied; see, e.g., [19],[31],[3] and section 2 for more details. By building upon the low rank structure of the right-hand side, matrix-oriented CG generates matrix recurrences, rather than vector recurrences, in *factored* form, thus allowing high memory and computational savings, while retaining the optimality properties when brought back to the vectorized form. Unfortunately, as the iterations progress, recurrence factors may quickly increase their rank, losing the advantages of the whole matrix-oriented procedure. Rank truncation strategies of the factor iterates are usually enforced so as to keep memory allocations under control. As a side effect, however, convergence is often delayed, also possibly leading to stagnation [19],[17],[31].

By taking inspiration from this class of methods, we design a new iterative scheme for the solution of (1.1) that better exploits the rich subspace information obtained with the computed quantities, to define the next factorized iterates. More precisely, at each iteration the next approximate solution and direction are obtained by imposing a functional optimality with respect to the whole range of the low rank factors available in the current iteration. This should be compared with the approximate solution in matrix-oriented CG, obtained at each iteration by a functional optimality with respect to a single vector.

The idea appears to be new, as it goes far beyond the algorithmic developments usually associated with matrix-oriented approaches. While being closer to optimization procedures based on manifolds, it does not share the same complexity in the definition of the funding recurrences, as the new method is still fully derived from the original Conjugate Gradient method for linear systems. Truncation strategies are devised to maintain the computed matrix iterates low rank.

In designing the new approach we address a memory allocation issue that becomes crucial

*Often the term "generalized Sylvester" is used for the same equation. The term "generalized" is also employed for two-term equations, for rectangular problems, and some multi-variable contexts. To avoid ambiguity we prefer to use the name "multiterm Sylvester".

when the number ℓ of terms in (1.1) is significantly larger than two. More precisely, for a given matrix $\boldsymbol{Y}$, the action $\mathcal{L}(\boldsymbol{Y})$ in low rank format may not be evaluated exactly, making it impossible to generate quantities such as the residual matrix. This problem is often underestimated in the current literature, though it will eventually produce Out-of-Memory breakdowns in actual computations, for ℓ large enough. We propose an ad-hoc randomized range-finding strategy that appears to fully resolve this shortcoming, keeping the memory allocations under control.

We name the new method the preconditioned *subspace-conjugate gradient method* (SS–CG) to emphasize the role of subspaces in the recurrences. This novel point of view leads to remarkable computational gains making SS–CG a very competitive option for the solution of multiterm linear matrix equations of the form (1.1). Computational experiments with a selection of quite diverse problems illustrate the potential of the new strategy, when compared with other methods specifically designed for the considered matrix equations.

Here is a synopsis of the paper. After introducing the notation we use throughout the paper in section 1.1, in section 2 we recall the matrix-oriented CG method for (1.1). Section 3 sees the derivation of the subspace-conjugate gradient method, the main contribution of this paper, and in section 3.1 we illustrate a first pseudoalgorithm for multiterm Lyapunov equations. Some theoretical aspects of the novel procedure are studied in section 4. In section 5 we generalize our method to the solution of multiterm Sylvester equations. We then present several memory- and time-saving strategies: we discuss low-rank truncations and effective residual computation in section 6.1, and the employment of inexact coefficients in section 6.2. Section 7 dwells with the inclusion of preconditioning strategies. The resulting algorithm for multiterm Sylvester matrix equations is summarized in section 8. Numerical results illustrating the competitiveness of our new procedure in solving multiterm matrix equations are presented in section 9. Conclusions are depicted in section 10. The Appendix collects some of the discussed algorithms.

1.1. Notation. Throughout the paper, capital, bold letters ($\boldsymbol{X}$) will denote $n_A \times n_B$ matrices, with capital letters (X) denoting their possibly low-rank factors, e.g., $\boldsymbol{X} = X_1 X_2^{\mathrm{T}}$. We already mention here that, for the sake of the presentation, we will often use the notation $\boldsymbol{X}$ for our iterates, although we will operate with their low-rank factors only, without allocating these matrices as full. See section 3 for further details.

Greek letters (α) will denote scalars whereas bold Greek letters ($\boldsymbol{\alpha}$) will be used for small dimensional matrices. Moreover, $\mathrm{blkdiag}(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_s)$ denotes the block diagonal matrix having on the diagonal the matrices $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_s$. The symbol $\otimes$ denotes the Kronecker product whereas $\mathrm{vec}(\cdot)$ is the operator that stacks the columns of a matrix one below the other. For a matrix $\boldsymbol{X}$, $\mathrm{range}(\boldsymbol{X})$ is the space spanned by the columns of $\boldsymbol{X}$.

2. Truncated matrix-oriented CG. This section is devoted to surveying the well exercised matrix-oriented version of the Conjugate Gradient method, together with its truncated variant.

When addressing the solution of (1.1), the Kronecker formulation of the problem, namely

$$(\boldsymbol{B}_1^{\mathrm{T}} \otimes \boldsymbol{A}_1 + \ldots + \boldsymbol{B}_\ell^{\mathrm{T}} \otimes \boldsymbol{A}_\ell)\mathrm{vec}(\boldsymbol{X}) = \mathrm{vec}(\boldsymbol{C}) \quad \Leftrightarrow \quad \mathcal{A}x = c, \tag{2.1}$$

allows one to directly employ the classical Preconditioned Conjugate Gradient (PCG) method. A careful implementation should avoid the explicit construction of $\mathcal{A}$, so that matrix-vector products can be carried out in the original matrix form with the $\mathcal{L}$ operator. Even with this precaution, all vector iterates still have $n_A n_B$ components, so that whenever n_A, n_B are large, memory allocations may become prohibitive. A particularly convenient way out occurs when $\boldsymbol{C}$ is very low rank. In this case, under certain conditions, the exact solution $\boldsymbol{X}$ may also be well approximated by a low

rank matrix [2],[3],[13],[19]. To exploit this characterization, all vector iterates are transformed back to matrices and kept in low-rank matrix format. Unfortunately, although during the first few PCG iterations the iterates maintain low rank, the rank itself grows as the method proceeds. To control the memory requirements of the procedure after the first few iterations, truncation of the iterate factors are usually performed. We refer to, e.g., [19, Algorithm 2], [31] for the algorithmic description.

As long as no low-rank truncations are performed, truncated PCG is mathematically equivalent to applying the standard, *vectorized* CG method to the linear system stemming from (1.1) via the Kronecker form in (2.1). Implementing low-rank truncations may be viewed as a simple computational device to make the solution process affordable in terms of storage allocation and not as an algorithmic advance. The final attainable accuracy when truncation is in place depends on the truncation tolerance and on the decay of the singular values in the problem solution matrix; we refer to [2],[17],[31] for a detailed discussion on the effect of truncation.

3. The SS–CG method for the multiterm Lyapunov equation. In this section we derive our new method for the multiterm Lyapunov equation, that is we assume that $\mathcal{L}(\boldsymbol{X}) = \mathcal{L}(\boldsymbol{X})^{\mathrm{T}}$ for any symmetric $\boldsymbol{X}$, and that $\boldsymbol{C}$ is symmetric. In section 5 we will discuss the changes occurring when generalizing the procedure to the multiterm Sylvester case.

The key idea is to build a Conjugate Gradient type method remaining in $\mathbb{R}^n$, while generating quantities based on *subspaces* of $\mathbb{R}^n$, rather than on *vectors* of $\mathbb{R}^{n^2}$, the way the original truncated PCG does.

We follow the classical derivation of CG as a procedure to approximate the minimizer of a convex function. Let the function $\Phi : \mathbb{R}^{n\times n} \to \mathbb{R}$ be defined as

$$\Phi(\boldsymbol{X}) = \frac{1}{2}\langle \boldsymbol{X}, \mathcal{L}(\boldsymbol{X})\rangle - \langle \boldsymbol{X}, \boldsymbol{C}\rangle. \tag{3.1}$$

We then consider the following minimization problem: Find $\boldsymbol{X} \in \mathbb{R}^{n\times n}$ such that

$$\boldsymbol{X} = \arg\min_{\boldsymbol{X}\in\mathbb{R}^{n\times n}} \Phi(\boldsymbol{X}).$$

Starting with a zero initial guess $\boldsymbol{X}_0 \in \mathbb{R}^{n\times n}$ and $P_0 \in \mathbb{R}^{n\times s_C}$, where we recall that s_C is the rank of $\boldsymbol{C}$, we define the recurrence $\{\boldsymbol{X}_k\}_{k\geq 0}$ of approximate solutions by means of the following relation

$$\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + P_k \boldsymbol{\alpha}_k P_k^{\mathrm{T}}, \tag{3.2}$$

where $\boldsymbol{\alpha}_k \in \mathbb{R}^{s_k\times s_k}$ and $P_k \in \mathbb{R}^{n\times s_k}$, with corresponding recurrence for the residual $\boldsymbol{R}_{k+1} = \boldsymbol{C} - \mathcal{L}(\boldsymbol{X}_{k+1})$, that is $\boldsymbol{R}_{k+1} = \boldsymbol{R}_k - \mathcal{L}(P_k\boldsymbol{\alpha}_k P_k^{\mathrm{T}})$. We emphasize that s_k depends on the iteration index k, that is the number of columns of P_k may (and will) change as the iterations proceed, possibly growing up to a certain maximum value, corresponding to the maximum allowed rank of all iterates. Since we assume that the operator $\mathcal{L}$ is symmetric, that is $\mathcal{L}(\boldsymbol{X}) = (\mathcal{L}(\boldsymbol{X}))^{\mathrm{T}}$ for any symmetric matrix $\boldsymbol{X}$, and that $\boldsymbol{C} = CC^{\mathrm{T}}$, all iteration matrices are square and symmetric. In section 5 we will relax these assumptions, yielding possibly rectangular solution and iterates.

Like in the vector case, we require that the matrix $\boldsymbol{P}_k = P_k P_k^{\mathrm{T}}$ satisfies a descent direction requirement completely conforming to the vector case, that is

$$\langle \nabla\Phi(\boldsymbol{X}_k), \boldsymbol{P}_k\rangle < 0. \tag{3.3}$$

To determine $\boldsymbol{\alpha}_k$, we let $\phi(\boldsymbol{\alpha}) = \Phi(\boldsymbol{X}_k + P_k\boldsymbol{\alpha}P_k^{\mathrm{T}})$. For given $\boldsymbol{X}_k, P_k$, at the kth iteration we construct $\boldsymbol{\alpha}_k$ so that

$$\phi(\boldsymbol{\alpha}_k) = \min_{\boldsymbol{\alpha}\in\mathbb{R}^{s_k\times s_k}} \phi(\boldsymbol{\alpha}). \tag{3.4}$$

The minimizer $\boldsymbol{\alpha}_k$ can be explicitly determined by solving a linear matrix equation of reduced dimensions, as described in the following result.

PROPOSITION 3.1. *Assume that $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ are symmetric, and that $\mathcal{L}$ is positive definite. The minimizer $\boldsymbol{\alpha}_k \in \mathbb{R}^{s_k\times s_k}$ of (3.4) is the unique solution of*

$$P_k^T\mathcal{L}(\boldsymbol{X}_k + P_k\boldsymbol{\alpha}P_k^T)P_k = P_k^T\boldsymbol{C}P_k, \tag{3.5}$$

or, equivalently, of $P_k^T\mathcal{L}(P_k\boldsymbol{\alpha}P_k^T)P_k = P_k^T\boldsymbol{R}_kP_k$.

Proof. We start by explicitly writing the function ϕ, that is

$$\phi(\boldsymbol{\alpha}) = \frac{1}{2}\langle \boldsymbol{X}_k + P_k\boldsymbol{\alpha}P_k^{\mathrm{T}}, \mathcal{L}(\boldsymbol{X}_k + P_k\boldsymbol{\alpha}P_k^{\mathrm{T}})\rangle - \langle \boldsymbol{X}_k + P_k\boldsymbol{\alpha}P_k^{\mathrm{T}}, \boldsymbol{C}\rangle.$$

To find the stationary points of ϕ, we compute the partial derivatives of ϕ with respect to $\boldsymbol{\alpha}$; this can be done in matrix compact form; see, e.g., [25]. We carry out this computation term by term,

$$\frac{\partial \mathrm{tr}(\boldsymbol{X}_k^{\mathrm{T}}\mathcal{L}(\boldsymbol{X}_k + P_k\boldsymbol{\alpha}P_k^{\mathrm{T}}))}{\partial\boldsymbol{\alpha}} = \frac{\partial \mathrm{tr}(\boldsymbol{X}_k^{\mathrm{T}}\mathcal{L}(P_k\boldsymbol{\alpha}P_k^{\mathrm{T}}))}{\partial\boldsymbol{\alpha}} = P_k^{\mathrm{T}}\mathcal{L}(\boldsymbol{X}_k)P_k,$$

and

$$\begin{aligned}\frac{\partial\,\mathrm{tr}\big((P_k\boldsymbol{\alpha}P_k^{\mathrm{T}})^{\mathrm{T}}\mathcal{L}\big(\boldsymbol{X}_k + P_k\boldsymbol{\alpha}P_k^{\mathrm{T}}\big)\big)}{\partial\boldsymbol{\alpha}} &= \frac{\partial\,\mathrm{tr}\left(\big(P_k\boldsymbol{\alpha}P_k^{\mathrm{T}}\big)^{\mathrm{T}}\mathcal{L}\big(\boldsymbol{X}_k\big)\right)}{\partial\boldsymbol{\alpha}} + \frac{\partial\mathrm{tr}\left(\big(P_k\boldsymbol{\alpha}P_k^{\mathrm{T}}\big)^{\mathrm{T}}\mathcal{L}(P_k\boldsymbol{\alpha}P_k^{\mathrm{T}})\right)}{\partial\boldsymbol{\alpha}}\\ &= P_k^{\mathrm{T}}\mathcal{L}\big(\boldsymbol{X}_k\big)P_k + 2{P_k}^{\mathrm{T}}\mathcal{L}(P_k\boldsymbol{\alpha}P_k^{\mathrm{T}})P_k.\end{aligned}$$

Moreover, it holds that $\frac{\partial\mathrm{tr}\left(\left(P_k\boldsymbol{\alpha}P_k^{\mathrm{T}}\right)^{\mathrm{T}}\boldsymbol{C}\right)}{\partial\boldsymbol{\alpha}} = P_k^{\mathrm{T}}\boldsymbol{C}P_k$, see, e.g., [25, Equations (101), (102), (108), (113)]. The final expression of the Jacobian of ϕ with respect to $\boldsymbol{\alpha}$ is thus

$$\frac{\partial\phi(\boldsymbol{\alpha})}{\partial\boldsymbol{\alpha}} = P_k^{\mathrm{T}}\mathcal{L}(P_k\boldsymbol{\alpha}P_k^{\mathrm{T}})P_k - P_k^{\mathrm{T}}\boldsymbol{R}_kP_k.$$

Consequently, the solution $\boldsymbol{\alpha}_k$ of (3.5) is a stationary point of ϕ. To ensure that $\boldsymbol{\alpha}_k$ is a minimizer, we show that the Hessian of ϕ is positive definite. To ease the reading, the Jacobian of ϕ is vectorized, resulting in

$$\mathrm{vec}\left(\frac{\partial\phi(\boldsymbol{\alpha})}{\partial\boldsymbol{\alpha}}\right) = \sum_{i=1}^{\ell}\big(P_k^{\mathrm{T}}\boldsymbol{B}_iP_k \otimes P_k^{\mathrm{T}}\boldsymbol{A}_iP_k\big)\mathrm{vec}\big(\boldsymbol{\alpha}\big) - \mathrm{vec}\big(P_k^{\mathrm{T}}\boldsymbol{R}_kP_k\big).$$

The Hessian $\boldsymbol{H}_k \in \mathbb{R}^{s_k^2\times s_k^2}$ is given by $\boldsymbol{H}_k = \sum_{i=1}^{\ell}\big(P_k^{\mathrm{T}}\boldsymbol{B}_iP_k \otimes P_k^{\mathrm{T}}\boldsymbol{A}_iP_k\big)$. Then, using the hypothesis on the operator $\mathcal{L}$, for any nonzero $\mathbf{y} \in \mathbb{R}^{s_k^2}$, it holds that $\mathbf{y}^{\mathrm{T}}\boldsymbol{H}_k\mathbf{y} > 0$, so that $\boldsymbol{H}_k$ is positive definite, and $\boldsymbol{\alpha}_k$ is a minimizer of ϕ. ∎

REMARK 3.2. For P_k full rank, the quantity $P_k\boldsymbol{\alpha}_kP_k^{\mathrm{T}}$ is invariant with respect to the basis of range(P_k) used to compute $\boldsymbol{\alpha}_k$. □

The minimization problem in (3.4) can also be recast in terms of an orthogonality condition. Indeed, solving (3.4) is equivalent to imposing the following *subspace orthogonality condition*

$$\mathrm{vec}(\boldsymbol{R}_{k+1}) \perp \mathrm{range}(P_k \otimes P_k). \tag{3.6}$$

More precisely, (3.6) is equivalent to $P_k^{\mathrm{T}} \boldsymbol{R}_{k+1} P_k = 0$ with $\boldsymbol{R}_{k+1} = \boldsymbol{C} - \mathcal{L}(\boldsymbol{X}_k + P_k \boldsymbol{\alpha} P_k^{\mathrm{T}})$. Hence, the computation of $\boldsymbol{\alpha}_k$ follows from a *local* matrix Galerkin projection of the original problem onto a space of dimension s_k^2 given by the current direction matrix factor P_k. We will return on this aspect in section 4.

REMARK 3.3. Thanks to the linearity of the matrix operator $\mathcal{L}$, products of the form $P_k^{\mathrm{T}}\mathcal{L}(\boldsymbol{X}_k + \boldsymbol{Y}_k)P_k$ for some matrix $\boldsymbol{Y}_k$, become

$$P_k^{\mathrm{T}}\mathcal{L}(\boldsymbol{X}_k + \boldsymbol{Y}_k)P_k = P_k^{\mathrm{T}}\mathcal{L}(\boldsymbol{X}_k)P_k + P_k^{\mathrm{T}}\mathcal{L}(\boldsymbol{Y}_k)P_k.$$

In addition, using the low rank factor form of the argument matrix, the left and right products act on the coefficient matrices as in reduction processes; see, e.g., [1, 30]. For instance, for $\boldsymbol{Y}_k = P_k \boldsymbol{\omega}_k P_k^{\mathrm{T}}$ and substituting the general operator $\mathcal{L}$ in (1.1), we obtain

$$P_k^{\mathrm{T}}\mathcal{L}(P_k \boldsymbol{\omega}_k P_k^{\mathrm{T}})P_k = \widetilde{\boldsymbol{A}}_1 \boldsymbol{\omega}_k \widetilde{\boldsymbol{B}}_1 + \ldots + \widetilde{\boldsymbol{A}}_\ell \boldsymbol{\omega}_k \widetilde{\boldsymbol{B}}_\ell,$$

with $\widetilde{\boldsymbol{A}}_i = P_k^{\mathrm{T}} \boldsymbol{A}_i P_k$, and $\widetilde{\boldsymbol{B}}_i = P_k^{\mathrm{T}} \boldsymbol{B}_i P_k$, for $i = 1, \ldots, \ell$. □

We define the recurrence for the directions $\boldsymbol{P}_k$ as

$$\boldsymbol{P}_{k+1} = \boldsymbol{R}_{k+1} + P_k \boldsymbol{\beta}_k P_k^{\mathrm{T}}.$$

The matrix $\boldsymbol{\beta}_k \in \mathbb{R}^{s_k \times s_k}$ is obtained by imposing that the new directions $\boldsymbol{P}_{k+1}$ are $\mathcal{L}$-orthogonal with respect to the previous ones. In particular, we write $\mathrm{vec}(\boldsymbol{P}_{k+1}) \perp_{\mathcal{L}} \mathrm{range}(P_k \otimes P_k)$, that is

$$(P_k \otimes P_k)^{\mathrm{T}} \mathrm{vec}(\mathcal{L}(\boldsymbol{P}_{k+1})) = 0. \tag{3.7}$$

Inserting the expression for $\boldsymbol{P}_{k+1}$, (3.7) becomes $P_k^{\mathrm{T}}\mathcal{L}(\boldsymbol{R}_{k+1} + P_k \boldsymbol{\beta}_k P_k^{\mathrm{T}})P_k = 0$, that is,

$$P_k^{\mathrm{T}}\mathcal{L}(\boldsymbol{R}_{k+1})P_k + P_k^{\mathrm{T}}\mathcal{L}(P_k \boldsymbol{\beta}_k P_k^{\mathrm{T}})P_k = 0.$$

This is a linear matrix equation in the unknown $\boldsymbol{\beta}_k$, with the same coefficient matrices of the linear matrix equation used to compute $\boldsymbol{\alpha}_k$. Once again, and using the considerations in Remark 3.3, $\boldsymbol{\beta}_k$ is obtained by solving a linear matrix equation of the same type as the original one, but with very small dimensions, by projecting the problem orthogonally onto $\mathrm{range}(P_k \otimes P_k)$, in a matrix sense.

Concerning the quality of the computed direction iterates, we next show that the descent direction property (3.3) is maintained.

PROPOSITION 3.4. *Let $\boldsymbol{P}_{k+1} = P_{k+1}\boldsymbol{\gamma}_{k+1}P_{k+1}^T$ for some matrix $\boldsymbol{\gamma}_{k+1}$. Then $\boldsymbol{P}_{k+1}$ is a descent direction.*

Proof. To prove that $\boldsymbol{P}_{k+1}$ is a descent direction matrix, we must show that

$$\langle \nabla\Phi(\boldsymbol{X}_{k+1}), \boldsymbol{P}_{k+1} \rangle < 0, \tag{3.8}$$

with Φ defined in (3.1). Following [25, Equations (101), (102), (108), (113)], we compute the terms of the Jacobian of Φ with respect to $\boldsymbol{X}_{k+1}$, yielding

$$\frac{\partial \mathrm{tr}(\boldsymbol{X}_{k+1}^{\mathrm{T}}\boldsymbol{C})}{\partial \boldsymbol{X}_{k+1}} = \boldsymbol{C}, \qquad \frac{\partial \mathrm{tr}\big(\boldsymbol{X}_{k+1}^{\mathrm{T}}\mathcal{L}(\boldsymbol{X}_{k+1})\big)}{\partial \boldsymbol{X}_{k+1}} = 2\mathcal{L}(\boldsymbol{X}_{k+1});$$

Algorithm 3.1 SS–CG - vanilla version for multiterm Lyapunov equations.

Input: Operator $\mathcal{L} : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$, right-hand side $\boldsymbol{C}$, initial guess $\boldsymbol{X}_0$, maximum number of iterations `maxit`, tolerance `tol`.
Output: Approximate solution $\boldsymbol{X}_k$ such that $\|\mathcal{L}(\boldsymbol{X}_k) - \boldsymbol{C}\| \le \|\boldsymbol{C}\| \cdot$ `tol`

Set $\boldsymbol{R}_0 = \boldsymbol{C} - \mathcal{L}(\boldsymbol{X}_0)$, $\boldsymbol{P}_0 = \boldsymbol{R}_0 = P_0 P_0^{\mathrm{T}}$
for $k = 0, \dots,$ `maxit` **do**
  Compute $\boldsymbol{\alpha}_k$ by solving $P_k^{\mathrm{T}} \mathcal{L}(P_k \boldsymbol{\alpha}_k P_k^{\mathrm{T}}) P_k = P_k^{\mathrm{T}} \boldsymbol{R}_k P_k$
  Set $\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + P_k \boldsymbol{\alpha}_k P_k^{\mathrm{T}}$ in a factorized fashion $X_{k+1}\boldsymbol{\tau}_{k+1}X_{k+1}^{\mathrm{T}} = \boldsymbol{X}_{k+1}$
    optional: low-rank truncation of $\boldsymbol{X}_{k+1}$
  Set $\boldsymbol{R}_{k+1} = \boldsymbol{C} - \mathcal{L}(X_{k+1}\boldsymbol{\tau}_{k+1}X_{k+1}^{\mathrm{T}})$ in a factorized fashion $R_{k+1}\boldsymbol{\rho}_{k+1}R_{k+1}^{\mathrm{T}} = \boldsymbol{R}_{k+1}$
    optional: low-rank truncation of $\boldsymbol{R}_{k+1}$
  if $\|\boldsymbol{R}_{k+1}\| \le \|\boldsymbol{C}\| \cdot$ `tol` **then**
    Return $\boldsymbol{X}_{k+1}$
  end if
  Compute $\boldsymbol{\beta}_k$ by solving $P_k^{\mathrm{T}} \mathcal{L}(P_k \boldsymbol{\beta}_k P_k^{\mathrm{T}}) P_k = -P_k^{\mathrm{T}} \mathcal{L}(\boldsymbol{R}_{k+1}) P_k$
  Set $\boldsymbol{P}_{k+1} = \boldsymbol{R}_{k+1} + P_k \boldsymbol{\beta}_k P_k^{\mathrm{T}}$ in a factorized fashion $P_{k+1}\boldsymbol{\gamma}_{k+1}P_{k+1}^{\mathrm{T}} = \boldsymbol{P}_{k+1}$
    optional: low-rank truncation of $\boldsymbol{P}_{k+1}$
end for
Return $\boldsymbol{X}_{k+1}$

Here we used the fact that $\boldsymbol{A}_i, \boldsymbol{B}_i$ are symmetric. Hence, $\nabla\Phi(\boldsymbol{X}_{k+1}) = \mathcal{L}(\boldsymbol{X}_{k+1}) - \boldsymbol{C} = -\boldsymbol{R}_{k+1}$. The inner product of (3.8) can be written as

$$\begin{aligned}\langle \nabla\Phi(\boldsymbol{X}_{k+1}), \boldsymbol{P}_{k+1}\rangle &= \langle -\boldsymbol{R}_{k+1}, \boldsymbol{R}_{k+1} + P_k\boldsymbol{\beta}_k P_k^{\mathrm{T}}\rangle \\ &= -\|\boldsymbol{R}_{k+1}\|_F^2 - \langle \boldsymbol{R}_{k+1}, P_k\boldsymbol{\beta}_k P_k^{\mathrm{T}}\rangle = -\|\boldsymbol{R}_{k+1}\|_F^2 < 0,\end{aligned}$$

where, by using (3.6), we have that $\langle \boldsymbol{R}_{k+1}, P_k\boldsymbol{\beta}_k P_k^{\mathrm{T}}\rangle = 0$. ∎

The proof above relies on the property $\langle \boldsymbol{R}_{k+1}, \boldsymbol{P}_k\rangle = 0$. In our setting, this annihilation is ensured in a stronger sense than in the matrix-oriented CG algorithm. More precisely, not only $\mathrm{vec}(\boldsymbol{P}_k)^{\mathrm{T}}\mathrm{vec}(\boldsymbol{R}_{k+1}) = 0$ holds, which would be enough to show Proposition 3.4, but the stronger constraint $P_k^{\mathrm{T}}\boldsymbol{R}_{k+1}P_k = 0$ holds. This *block* orthogonality is reminiscent of block methods for multiple right-hand side systems [22], though in practice there are no further connections.

3.1. A first version of the algorithm for the multiterm Lyapunov equation. Summarizing the previous derivation, the iteration of the SS–CG scheme is given in Algorithm 3.1. This algorithm includes extra commands with respect to our initial presentation, which require more detailed explanation. Following standard procedures, the next iterates $\boldsymbol{X}_{k+1}, \boldsymbol{P}_{k+1}$, and $\boldsymbol{R}_{k+1}$ are not explicitly computed, as this would lead to storing large dense matrices. Each of these matrices is kept in factored form, whose rank is truncated if necessary. The updating step is linked to the subsequent factorization step as follows. Consider the approximate solution update, starting from $\boldsymbol{X}_k = X_k\boldsymbol{\tau}_k X_k^{\mathrm{T}}$. We write

$$\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + P_k\boldsymbol{\alpha}_k P_k^{\mathrm{T}} = [X_k, P_k]\mathrm{blkdiag}(\boldsymbol{\tau}_k, \boldsymbol{\alpha}_k)[X_k, P_k]^{\mathrm{T}} = X_{k+1}\boldsymbol{\tau}_{k+1}X_{k+1}^{\mathrm{T}},$$

where X_{k+1} is obtained as the reduced orthonormal factor of the QR decomposition of $[X_k, P_k]$, that is $[X_k, P_k] = Q\boldsymbol{r}$, and $\boldsymbol{\tau}_{k+1} = \boldsymbol{r}\mathrm{blkdiag}(\boldsymbol{\tau}_k, \boldsymbol{\alpha}_k)\boldsymbol{r}^{\mathrm{T}}$. A more precise implementation ensures that $\boldsymbol{\tau}_{k+1}$ has full rank via an eigenvalue decomposition, that may lower the rank of the factor X_{k+1}. From a memory point of view, none of the full matrices in bold is stored, as factors are immediately created and saved. More details on this rank reduction will be given in section 6.1.

We stress that the updated terms X_{k+1}, P_{k+1}, and R_{k+1} in Algorithm 3.1 each have orthonormal columns, thus simplifying some of the computations. We also observe that the factor $\boldsymbol{\gamma}_{k+1}$ in $P_{k+1}\boldsymbol{\gamma}_{k+1}P_{k+1}^{\mathrm{T}}$ does not play a role in later computations, as only the subspace basis $P_{k+1} \otimes P_{k+1}$ is employed; see Remark 3.2. We postpone the complete implementation of the method to section 8, after the description of several advanced implementation strategies.

4. Discussion on the developed procedure. It is natural to compare the new SS–CG with the standard matrix-oriented CG. The subspaces acting in the SS–CG method are significantly larger than in matrix-oriented CG, as explained next.

REMARK 4.1. In (3.6), orthogonality is imposed with respect to a subspace of $\mathbb{R}^{n^2}$ of dimension s_k^2. On the other hand, in the matrix-oriented CG condition (3.6) is replaced by $\mathrm{vec}(\boldsymbol{P}_k)^{\mathrm{T}}\mathrm{vec}\big(\boldsymbol{R}_k - \alpha_k\mathcal{L}(P_kP_k^{\mathrm{T}})\big) = 0$, with $\alpha_k \in \mathbb{R}$, so that the orthogonality is imposed with respect to a subspace of $\mathbb{R}^{n^2}$ of dimension 1. Analogously, in (3.7), orthogonality is imposed with respect to a subspace of $\mathbb{R}^{n^2}$ of size s_k^2, whereas in the matrix-oriented CG, the orthogonality condition is instead given by $\mathrm{vec}(\boldsymbol{P}_k)^{\mathrm{T}}\mathrm{vec}\big(\mathcal{L}(\boldsymbol{P}_{k+1})\big) = 0$, that is, with respect to a subspace of $\mathbb{R}^{n^2}$ of dimension 1. □

The orthogonality conditions imposed in deriving the coefficient matrices $\boldsymbol{\alpha}_k$, $\boldsymbol{\beta}_k$ allow us to extend orthogonality properties to other iterates, and to derive optimality results later in the section. To this end, we introduce some notation for operations with the generalized Lyapunov operator.

For $R \in \mathbb{R}^{n\times s}$ we will denote with $\boldsymbol{A}_\star \bullet R$ the matrix

$$\boldsymbol{A}_\star \bullet R = [\boldsymbol{A}_1 R, \dots, \boldsymbol{A}_\ell R],$$

and analogously for $\boldsymbol{B}_\star \bullet R$. Moreover, for $k \geq 0$ we define

$$\boldsymbol{A}_\star^{k+1} \bullet R = \boldsymbol{A}_\star \bullet (\boldsymbol{A}_\star^{k} \bullet R).$$

We are going to characterize the spaces generated by the factors P_k, R_k of $\boldsymbol{P}_k, \boldsymbol{R}_k$, respectively, with the next proposition. To this end, with the new notation we define the approximation space

$$\mathscr{K}_k(\boldsymbol{A}_\star, R_0) = \mathrm{range}([R_0, \boldsymbol{A}_\star \bullet R_0, \dots, \boldsymbol{A}_\star^k \bullet R_0]);$$

Note that the spaces are nested, that is $\mathscr{K}_k(\boldsymbol{A}_\star, R_0) \subseteq \mathscr{K}_{k+1}(\boldsymbol{A}_\star, R_0)$. The notation above is reminiscent of a block Krylov subspace. However, the space is in general very different. Indeed, it involves all matrices associated with the operation $\bullet$, that is $\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell$. Although the space dimension grows very quickly, it can be significantly smaller than the sum of the number of terms; for instance, if one of the $\boldsymbol{A}_i$'s is the identity matrix, then the product $\boldsymbol{A}_\star \bullet R_0$ will surely contribute at most $(\ell - 1)\cdot s$ vectors to the space $\mathrm{range}([R_0, \boldsymbol{A}_\star \bullet R_0])$, for $R_0 \in \mathbb{R}^{n\times s}$, due to the redundancy of R_0. We also observe that for the Lyapunov operator, $\mathscr{K}_k(\boldsymbol{A}_\star, R_0) = \mathscr{K}_k(\boldsymbol{B}_\star, R_0)$. Note that the fact that the left and right spaces are the same justifies our use of iterates in the form $P_k\boldsymbol{\omega}P_k^{\mathrm{T}}$ for some $\boldsymbol{\omega}$.

PROPOSITION 4.2. *Assume* $\boldsymbol{X}_0 = 0$ *so that* $R_0 = C$. *Then* $\mathrm{range}(R_k), \mathrm{range}(P_k) \subseteq \mathscr{K}_k(\boldsymbol{A}_\star, R_0)$.

Proof. For brevity, we denote $\mathrm{range}(Y)$ as $r(Y)$. We also recall that the updates have the form

$$\boldsymbol{X}_{k+1} = X_k\boldsymbol{\tau}_k X_k^{\mathrm{T}} + P_k\boldsymbol{\alpha}_k P_k^{\mathrm{T}} = [X_k, P_k]\boldsymbol{\tau}_{k+1}[X_k, P_k]^{\mathrm{T}},$$
$$\boldsymbol{R}_{k+1} = [R_0, \boldsymbol{A}_\star \bullet X_{k+1}]\boldsymbol{\rho}_{k+1}[R_0, \boldsymbol{B}_\star \bullet X_{k+1}]^{\mathrm{T}},$$

for some $\boldsymbol{\rho}_{k+1}$, and $\boldsymbol{P}_{k+1} = R_{k+1}\boldsymbol{\rho}_{k+1}R_{k+1}^{\mathrm{T}} + P_k\boldsymbol{\beta}_k P_k^{\mathrm{T}} = [R_{k+1}, P_k]\mathrm{blkdiag}(\boldsymbol{\rho}_{k+1}, \boldsymbol{\beta}_k)[R_{k+1}, P_k]^{\mathrm{T}}$. It suffices to collect and write down the block components for the first few iterations. The result then follows by induction. Indeed,

$$
\begin{aligned}
&P_0 = R_0, \quad X_1 = P_0, \quad r(R_1) \subset r([R_0, \boldsymbol{A}_\star \bullet R_0]) = \mathscr{K}_1 \\
&r(P_1) \subset r([R_1, R_0]) \subset \mathscr{K}_1, \quad X_2 \subset r([X_1, P_1]) \subset r([P_0, P_1]) \subset r([R_0, R_1]) \\
&\qquad r(R_2) = r([R_0, \boldsymbol{A}_\star \bullet X_2]) \subset r([R_0, \boldsymbol{A}_\star \bullet R_0, \boldsymbol{A}_\star \bullet P_1]) \subset r([R_0, \boldsymbol{A}_\star \bullet R_0, \boldsymbol{A}_\star^2 \bullet R_0]) = \mathscr{K}_2 \\
&r(P_2) \subset r([R_2, P_1]) \subset r([R_0, R_1, R_2]) \subset \mathscr{K}_2,
\end{aligned}
$$

and so on. ∎

We proceed with a result ensuring that subsequent residual matrices are block orthogonal to each other. In the following we say that a matrix with blocks has maximum possible rank if rank reduction is only due to linear dependence in exact arithmetic. For instance, $[v, \boldsymbol{A}_1 v, v]$ and $[v, \boldsymbol{A}_1 v]$ have the same maximum possible rank two. As a related concept, we shall talk about maximum possible dimension for the subspaces generated by matrices with the same maximum possible rank.

PROPOSITION 4.3. *For any $k > 0$, let $\boldsymbol{R}_k = R_k\boldsymbol{\rho}_k R_k^T$. Assume that all updates have maximum possible rank, so that* range$(\boldsymbol{P}_k)$, range$(\boldsymbol{R}_k)$, *and $\mathscr{K}_k(\boldsymbol{A}_\star, R_0)$ have the same dimension. Then $R_k^T\boldsymbol{R}_{k+1}R_k = 0$.*

Proof. Let the columns of U_k form an orthonormal basis for $\mathscr{K}_k$. Using the stated hypotheses, we have that $P_k = U_k G_1$ and $R_k = U_k G_2$ with G_1, G_2 having full row rank. From (3.6) we have that $0 = P_k^{\mathrm{T}}\boldsymbol{R}_{k+1}P_k = G_1^{\mathrm{T}}U_k^{\mathrm{T}}\boldsymbol{R}_{k+1}U_k G_1$, and since G_1 has full row rank, it holds that $U_k^{\mathrm{T}}\boldsymbol{R}_{k+1}U_k = 0$. Since $R_k^{\mathrm{T}}\boldsymbol{R}_{k+1}R_k = G_2^{\mathrm{T}}U_k^{\mathrm{T}}\boldsymbol{R}_{k+1}U_k G_2$, the result follows. ∎

From the proof above, and under the same hypothesis of equal maximum possible dimension of range$(\boldsymbol{P}_k)$, range$(\boldsymbol{R}_k)$ and $\mathscr{K}_k(\boldsymbol{A}_\star, R_0)$, it also follows that $R_j^{\mathrm{T}}\boldsymbol{R}_{k+1}R_j = 0$, $j = 1, \ldots, k$.

We can next state a finite termination result.

PROPOSITION 4.4. *Assume that* range(R_k) = range(P_k) *and have maximum possible dimension. If $\mathcal{L}$ is the multiterm Lyapunov operator and it holds that*

$$\mathit{range}(\mathcal{L}(P_k\boldsymbol{\alpha}_k P_k^T)) \subseteq \mathrm{range}(P_k),$$

then the space $\mathit{range}(P_k \otimes P_k)$ contains the exact solution.

Proof. Under the stated hypothesis, $\mathcal{L}(P_k\boldsymbol{\alpha}_k P_k^{\mathrm{T}}) = P_k\boldsymbol{\omega}_k P_k^{\mathrm{T}}$ for some matrix $\boldsymbol{\omega}_k$. Hence, $\boldsymbol{R}_{k+1} = \boldsymbol{R}_k - P_k\boldsymbol{\omega}_k P_k^{\mathrm{T}}$ so that range$(R_{k+1}) \subset$ range(P_k). From (3.6) we have that $\boldsymbol{R}_{k+1} \perp \mathit{range}(P_k)$, hence it must be $\boldsymbol{R}_{k+1} = 0$. ∎

The formalization in terms of the space $\mathscr{K}_k$ allows us to characterize the new method with respect to less close but still known approaches. Unless truncation takes place, it holds that range$(P_{k-1}) \subseteq$ range(P_k), so that the iterate $\boldsymbol{X}_{k+1}$ could be written as $\boldsymbol{X}_{k+1} = P_k\boldsymbol{\tau}_k P_k^{\mathrm{T}}$, for some $\boldsymbol{\tau}_k$. Moreover, the residual matrix $\boldsymbol{R}_{k+1}$ is orthogonal, in the matrix inner product, to the space $\mathscr{K}_k \otimes \mathscr{K}_k$. These two properties together show that under maximum possible rank of the iterates, the new algorithm is mathematically equivalent to the Galerkin method for Lyapunov equations on the subspace $\mathscr{K}_k(\boldsymbol{A}_\star, R_0)$ [30]. For the operator $\mathcal{L}(\boldsymbol{X}) = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{X}\boldsymbol{A}^{\mathrm{T}} + \boldsymbol{M}\boldsymbol{X}\boldsymbol{M}^{\mathrm{T}}$, it is interesting to observe that $\mathscr{K}_k$ is the same as the space introduced in [31, Section 4], although in there the space was generated one vector at the time. Moreover, the approach we are taking here allows us to update the iterates, rather than solving the projected system from scratch at each iteration. The two approaches significantly deviate when truncation takes place.

By following the discussion in [24, Section 2.2], thanks to the orthogonality condition (3.6) imposed to compute $\boldsymbol{\alpha}_k$, we can also state the following optimality result.

PROPOSITION 4.5. *Let $\boldsymbol{X}$ be the exact solution to (1.1) and $\|\boldsymbol{Y}\|_{\mathcal{L}}^2 = \langle \boldsymbol{Y}, \mathcal{L}(\boldsymbol{Y})\rangle$. Assume that $P_k \in \mathbb{R}^{n\times s_k}$ is computed by Algorithm 3.1 with no low-rank truncation and that range(P_k) has the maximum possible dimension. Then, $\boldsymbol{X}_k = P_k\boldsymbol{\tau}_k P_k^T$ is such that*

$$\boldsymbol{X}_k = \arg \min_{\substack{\boldsymbol{Z}=P_k\boldsymbol{\tau}P_k^T \\ \boldsymbol{\tau}\in\mathbb{R}^{s_k\times s_k}}} \|\boldsymbol{X}-\boldsymbol{Z}\|_{\mathcal{L}}.$$

Proof. The proof follows the same lines as the proof of [24, Proposition 1]. ∎

We end this section with a consideration on the numerical rank of the approximate solution iterate. Numerical experiments with matrix-oriented CG have shown that without truncation, the approximate solution rank tends to significantly increase before decreasing towards its final value, corresponding to the rank of the exact solution; see, e.g., [19]. Allowing for a richer linear combination of the generated space columns, we expect that the approximate solution of SS–CG, with no truncation, will reach the final rank without an intermediate growth. Numerical experiments seemed to confirm this fact, although a rigorous analysis remains an open problem.

5. The iteration for the multiterm Sylvester equation. When the matrix operator $\mathcal{L}$ is nonsymmetric, that is $\mathcal{L}(\boldsymbol{X}) \neq (\mathcal{L}(\boldsymbol{X}))^{\mathrm{T}}$ for symmetric $\boldsymbol{X}$, or $\boldsymbol{C}$ is indefinite or even nonsymmetric, the iteration obtained with the new algorithm needs to be revised to address the general Sylvester problem in (1.1). For general multiterm Sylvester equations, we still assume that all coefficient matrices are symmetric, although the $\boldsymbol{A}_i$'s and the $\boldsymbol{B}_i$'s matrices are different. This is in fact the more common situation for the problem (1.1), as the coefficient matrices $\boldsymbol{A}_i, \boldsymbol{B}_i$, $i = 1,\dots,\ell$ may even have different dimensions, leading to a rectangular solution matrix $\boldsymbol{X}$. Fortunately, the algorithmic differences are only technical, mostly affecting the notation. Given two matrices P_k^l, P_k^r, the iterates are computed by means of the relation $\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + P_k^l\boldsymbol{\alpha}_k(P_k^r)^{\mathrm{T}}$, with $\boldsymbol{X}_k \in \mathbb{R}^{n_A\times n_B}$, and $\boldsymbol{\alpha}_k$ is computed by solving the reduced equation

$$(P_k^l)^{\mathrm{T}}\mathcal{L}(P_k^l\boldsymbol{\alpha}(P_k^r)^{\mathrm{T}})P_k^r = (P_k^l)^{\mathrm{T}}\boldsymbol{R}_k P_k^r.$$

Analogously, $\boldsymbol{P}_{k+1} = \boldsymbol{R}_{k+1} + P_k^l\boldsymbol{\beta}_k(P_k^r)^{\mathrm{T}}$, so that $P_{k+1}^l\boldsymbol{\gamma}_{k+1}(P_{k+1}^r)^{\mathrm{T}} = \boldsymbol{P}_{k+1}$, where $\boldsymbol{\beta}_k$ solves $(P_k^l)^{\mathrm{T}}\mathcal{L}(\boldsymbol{R}_{k+1})P_k^r + (P_k^l)^{\mathrm{T}}\mathcal{L}(P_k^l\boldsymbol{\beta}(P_k^r)^{\mathrm{T}})P_k^r = 0$. This construction of $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$ ensures that the orthogonality conditions discussed in the previous sections continue to hold, with respect to the left and right factors.

Like for the direction matrix $\boldsymbol{P}_k$, also the iterates $\boldsymbol{X}_k$ and $\boldsymbol{R}_k$ will be nonsymmetric and possibly rectangular, and they need to be factorized accordingly, namely $\boldsymbol{X}_k = X_k^l\boldsymbol{\tau}_k(X_k^r)^{\mathrm{T}}$ and $\boldsymbol{R}_k = R_k^l\boldsymbol{\rho}_k(R_k^r)^{\mathrm{T}}$. All truncation strategies will have to keep into account the nonsymmetry of the iterate, and in particular, we will see that all computations need to be performed in a mirrored fashion for the left and right spaces. The overall algorithm for the resulting multiterm Sylvester equation will be given in section 8, Algorithm 8.1.

6. Advanced implementation devices. In this section we discuss some advanced devices to make Algorithm 3.1 competitive and robust in terms of memory requirements, running time, and final attainable accuracy for the given chosen truncation thresholds. We start by analyzing in detail the low-rank iterate truncation, including the residual matrix computation, then we discuss the computation of the (matrix) coefficients $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$.

6.1. Low-rank truncations. Solvers for large-scale matrix equations often require a low-rank truncation step to make the overall solution process affordable in terms of storage allocation. This is usually carried out by performing a thin QR decomposition and a subsequent truncated singular value decomposition (SVD) of small dimensional objects; see, e.g., [19]. For the sake of brevity, we will refer to this procedure as a QR-SVD (low-rank) truncation in the following. In matrix-oriented constructions of CG type methods, the truncation step is a crucial part of the implementation, because it determines the actual success of the whole procedure. A well designed truncation strategy, balancing the low-rank requirement and the singular value accuracy, may allow the algorithm to deliver a sufficiently accurate solution. In the past few years the effects of truncation have been analyzed – both experimentally and theoretically – in several articles, see, e.g., [19],[17],[31],[32]. In our setting the space truncation is completely analogous to that encountered in truncated CG, so that a similar sensitivity to truncation is expected; this was confirmed by our extensive computational experience; we refer to Example 9.1 for a sample.

In our setting, if we consider $\boldsymbol{X}_k = X_k^l \boldsymbol{\tau}_k (X_k^r)^{\mathrm{T}}$, this is updated as

$$\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + P_k^l \boldsymbol{\alpha}_k (P_k^r)^{\mathrm{T}} = [X_k^l, P_k^l]\mathrm{blkdiag}(\boldsymbol{\tau}_k, \boldsymbol{\alpha}_k)[X_k^r, P_k^r]^{\mathrm{T}}.$$

Let $Q^l \boldsymbol{r}^l = [X_k^l, P_k^l]$, $Q^r \boldsymbol{r}^r = [X_k^r, P_k^r]$ be the thin QR decompositions of the two matrices, and compute the singular value decomposition $\boldsymbol{r}^l \mathrm{blkdiag}(\boldsymbol{\tau}_k, \boldsymbol{\alpha}_k)(\boldsymbol{r}^r)^{\mathrm{T}} = U\Sigma V^{\mathrm{T}}$, with $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_s)$. The low-rank truncation then takes place following two different criteria. The first one uses a threshold `tolrank` and the other one a maximum rank `maxrank`. In the first case, the number of columns $\widehat{j}_{k+1}$ of the low-rank terms X_{k+1}^l and X_{k+1}^r defining $\boldsymbol{X}_{k+1}$ will be given by $\widehat{j}_{k+1} = \arg\max_j \{\sigma_j \, : \, (\sigma_j/\sigma_1) \leq \texttt{tolrank}\}$, where the σ_js are the singular values just computed. In the second case, for Σ of size $s \times s$, we have $\widehat{j}_{k+1} = \min\{\texttt{maxrank}, s\}$. These two selections of $\widehat{j}_{k+1}$ are often performed in different moments of the iterative solver. The `tolrank` criterion is preferable at an initial stage, when the iterates rank is still moderate by construction. In later iterations, memory constraints generally force the application of the more aggressive truncation based on `maxrank`. An automatic switch between the two truncation policies is obtained as follows

$$\widehat{j}_{k+1} = \min\{\texttt{maxrank}, s, \arg\max_j \{\sigma_j \, : \, (\sigma_j/\sigma_1) \leq \texttt{tolrank}\}\}.$$

Once $\widehat{j}_{k+1}$ is selected, we define $X_{k+1}^l = Q^l U_{1:\widehat{j}_{k+1}}$, $X_{k+1}^r = Q^r V_{1:\widehat{j}_{k+1}}$, and $\boldsymbol{\tau}_{k+1} = \Sigma_{1:\widehat{j}_{k+1}}$, with the previous notation. In the rest of the paper, we will adopt the notation (see, e.g., [19])

$$[X_{k+1}^l, \boldsymbol{\tau}_{k+1}, X_{k+1}^r] = \mathcal{T}([X_k^l, P_k^l], \mathrm{blkdiag}(\boldsymbol{\tau}_k, \boldsymbol{\alpha}_k), [X_k^r, P_k^r], \texttt{params}), \tag{6.1}$$

for the computation of the QR-SVD truncated updating. In (6.1), `params` is a shorthand notation that indicates that all the necessary parameters are given in input. In particular, our QR-SVD requires the values `tolrank` and `maxrank`. A QR-SVD truncation can be applied to compute $\boldsymbol{P}_{k+1} = P_{k+1}^l \boldsymbol{\gamma}_{k+1} (P_{k+1}^r)^{\mathrm{T}}$ as well.

In principle, the term $\boldsymbol{R}_{k+1} = R_{k+1}^l \boldsymbol{\rho}_{k+1} (R_{k+1}^r)^{\mathrm{T}}$ could be computed in the same way. However, we would like to bring to the reader's attention an aspect that is often overlooked. More precisely,

by following the same procedure as above, we would have

$$\begin{aligned}\boldsymbol{R}_{k+1} &= \boldsymbol{C} - \mathcal{L}(X^l_{k+1}\boldsymbol{\tau_{k+1}}(X^r_{k+1})^{\mathrm{T}}) \\ &= [C_1, A_1X^l_{k+1}, \dots, A_\ell X^l_{k+1}] \begin{bmatrix} I & & & \\ & -\boldsymbol{\tau}_{k+1} & & \\ & & \ddots & \\ & & & -\boldsymbol{\tau}_{k+1} \end{bmatrix} [C_2, B_1X^r_{k+1}, \dots, B_\ell X^r_{k+1}]^{\mathrm{T}}. \end{aligned} \tag{6.2}$$

For a large number of terms ℓ in (1.1), explicitly allocating the matrices $[C_1, A_1X^l_{k+1}, \dots, A_\ell X^l_{k+1}]$ and $[C_2, B_1X^r_{k+1}, \dots, B_\ell X^r_{k+1}]$ may not be affordable[†]. This drawback is not a peculiarity of Algorithm 3.1, as it often plagues solvers for (1.1) as, e.g., the algorithms in [31],[23],[8].

For ℓ larger than three or four, say, we consider the use of two possible strategies to overcome this issue. The first one computes a number of thin QR factorizations, and the second one relies on randomized numerical linear algebra tools. In the following discussion, we employ an ad-hoc memory allocation threshold, namely `maxrankR`, which we set to be equal to 2·`maxrank` in our implementation. Note that this value does not increase the actual requested memory allocations, as the strategy employed for the iterates X_{k+1}, P_{k+1} above requires storing two blocks of size `maxrank` each; see Algorithm 8.1.

Dynamic truncated QR update. In place of computing the thin QR of the whole matrices $[C_1, \boldsymbol{A}_1X^l_{k+1}\boldsymbol{\tau_{k+1}}, \dots, \boldsymbol{A}_\ell X^l_{k+1}\boldsymbol{\tau_{k+1}}]$ and $[C_2, -\boldsymbol{B}_1X^r_{k+1}, \dots, -\boldsymbol{B}_\ell X^r_{k+1}]$ and then use their triangular factors in the truncation, we sequentially combine 2ℓ thin QR factorizations one after the other, detecting a possible linear dependency in the factors after each QR. More in detail, we collect the subsequent products as follows:

$Q^l\boldsymbol{r}^l = [C_1, \boldsymbol{A}_1X^l_{k+1}\boldsymbol{\tau_{k+1}}]$, $\quad Q^r\boldsymbol{r}^r = [C_2, -\boldsymbol{B}_1X^r_{k+1}]$ (QR factors of the two matrices)

For $j = 2, \dots, \ell$

$Q^l\boldsymbol{r}^l_1 = [Q^l, \boldsymbol{A}_jX^l_{k+1}\boldsymbol{\tau_{k+1}}]$, $\quad Q^r\boldsymbol{r}^r_1 = [Q^r, -\boldsymbol{B}_jX^r_{k+1}]$ (QR factors of the two matrices)

$$\boldsymbol{r}^l = \boldsymbol{r}^l_1 \begin{bmatrix} \boldsymbol{r}^l & 0 \\ 0 & I \end{bmatrix}, \qquad \boldsymbol{r}^r = \boldsymbol{r}^r_1 \begin{bmatrix} \boldsymbol{r}^r & 0 \\ 0 & I \end{bmatrix}.$$

This procedure does not yet control the rank. To do so, the triangular matrices $\boldsymbol{r}^l_1$ and $\boldsymbol{r}^r_1$ are decomposed by means of the SVD, so as to truncate the rank down to the maximum admittable value `maxrankR`. More precisely, if $\boldsymbol{r}^l_1 = U\Sigma V^{\mathrm{T}}$ is the singular value decomposition of $\boldsymbol{r}^l_1$, then the factors are truncated to the most $i \le$ `maxrankR` leading diagonal elements in Σ, so that $\boldsymbol{r}^l_1 \approx U_{:,1:i}\Sigma_{1:i,1:i}V^{\mathrm{T}}_{:,1:i}$. Then, the matrices Q^l and $\boldsymbol{r}^l$ are updated accordingly[‡], that is

$$Q^l = Q^lU_{:,1:i}, \qquad \boldsymbol{r}^l = \Sigma_{1:i,1:i}V^{\mathrm{T}}_{:,1:i} \begin{bmatrix} \boldsymbol{r}^l & 0 \\ 0 & I \end{bmatrix}.$$

The same is done for Q^r and $\boldsymbol{r}^r$. At the end of the j-cycle, setting $R^l_{k+1} = Q^l$, $R^r_{k+1} = Q^r$, and $\boldsymbol{\rho}_{k+1} = \boldsymbol{r}^l(\boldsymbol{r}^r)^{\mathrm{T}}$, we (re)define $\boldsymbol{R}_{k+1} := R^l_{k+1}\boldsymbol{\rho}_{k+1}(R^r_{k+1})^{\mathrm{T}}$, (not explicitly computed) which is now an approximation to the true quantity in (6.2).

[†]Notice that in the multiterm Lyapunov case, only the factor $[C, A_1X_{k+1}, \dots, A_\ell X_{k+1}]$ needs to be stored, by possibly rearranging the terms in the middle matrix containing the $\boldsymbol{\tau}_i$s; see, e.g., [31].

[‡]A rank revealing QR decomposition could also be employed.

Randomized approximate QR update. The main goal is to compute tall matrices Q and W with orthonormal columns, whose range is able to well represent the column- and row-space of $\boldsymbol{R}_{k+1}$, respectively, i.e., range$(Q) \approx$ range$(\boldsymbol{R}_{k+1})$ and range$(W) \approx$ range$(\boldsymbol{R}_{k+1}^{\mathrm{T}})$. To this end, we apply the randomized range finder algorithm that can be found in, e.g., [14, Algorithm 4.1]. The first step in [14, Algorithm 4.1] consists in multiplying the matrix at hand ($\boldsymbol{R}_{k+1}$ and $\boldsymbol{R}_{k+1}^{\mathrm{T}}$ in our case) by a so-called *sketching* matrix G^l of conforming dimensions. The randomized nature of this sketching matrix is key for the success of the entire procedure. We will employ only Gaussian sketching matrices, though other options are available in the literature; see, e.g., [14] for a discussion.

In our setting, given a target rank `maxrankR`, we generate a Gaussian matrix $G^l \in \mathbb{R}^{n_B \times \texttt{maxrankR}}$, and using (6.2) we then compute

$$\boldsymbol{R}_{k+1}G^l = C_1(C_2^{\mathrm{T}}G^l) - \sum_{i=1}^{\ell} \boldsymbol{A}_i(X_{k+1}^l \boldsymbol{\tau}_{k+1}((X_{k+1}^r)^{\mathrm{T}}(\boldsymbol{B}_i G^l))). \tag{6.3}$$

The algorithm proceeds with computing the $Q \in \mathbb{R}^{n_A \times \texttt{maxrankR}}$ matrix of the thin QR decomposition for the right-hand side matrix in (6.3), whose range is used as an approximation of the column-space of $\boldsymbol{R}_{k+1}$. The quality of this approximation strongly depends on the choice of the target rank `maxrankR` and on the decay rate of the singular values of $\boldsymbol{R}_{k+1}$; see, e.g., [14, Theorem 9.1].

The same procedure is adopted to compute $W \in \mathbb{R}^{n_B \times \texttt{maxrankR}}$, with $\boldsymbol{R}_{k+1}$ replaced by $\boldsymbol{R}_{k+1}^{\mathrm{T}}$.

Once Q and W are computed, we use (6.2) to perform

$$Q^{\mathrm{T}}\boldsymbol{R}_{k+1}W = Q^{\mathrm{T}}C_1C_2^{\mathrm{T}}W - \sum_{i=1}^{\ell}(Q^{\mathrm{T}}\boldsymbol{A}_i X_{k+1})\boldsymbol{\tau}_{k+1}(X_{k+1}^{\mathrm{T}}\boldsymbol{B}_i W) \in \mathbb{R}^{\texttt{maxrankR}\times\texttt{maxrankR}}.$$

The procedure concludes by computing a truncated SVD of $Q^{\mathrm{T}}\boldsymbol{R}_{k+1}W$, namely $Q^{\mathrm{T}}\boldsymbol{R}_{k+1}W \approx \widehat{U}\widehat{\Sigma}\widehat{V}^{\mathrm{T}}$ providing the terms $R_{k+1}^l = Q\widehat{U}$, $\boldsymbol{\rho}_{k+1} = \widehat{\Sigma}$, and $R_{k+1}^r = W\widehat{V}$.

We would like to stress that the same sketching matrices G^l, G^r can be used throughout the process, so that they can be generated once for all at the beginning of the iterative procedure.

As already mentioned, in principle one could use non-Gaussian sketching matrices G^l and G^r and adopt, e.g., subsampled randomized trigonometric transformations (SRTT) for this task. However, we believe that employing Gaussian matrices is more appropriate in our context. Indeed, due to memory restrictions, we are able to allocate (at most) `maxrankR` columns for the sketching matrices, and Gaussian matrices provide better approximations than SRTTs for a fixed target rank, in general; see, e.g., [14, Section 11.2]. Achieving good rank-`maxrankR` approximations to the residual matrix $\boldsymbol{R}_k$ is crucial for the convergence of the overall SS–CG method. Therefore, we always employ Gaussian matrices in spite of the slightly larger, yet negligible, cost in their application.

Numerical results reported in Table 6 for Example 9.4 for which $\ell = 10$, show that the randomized update is superior to the dynamic truncated procedure. Hence, in all other tests we either report results with explicit computations of the products with $\boldsymbol{A}_\star\bullet$, $\boldsymbol{B}_\star\bullet$ or with the randomized strategy. The corresponding procedure is summarized in Algorithm 10.1 in the Appendix, and it is named $\mathcal{T}_{res}$.

6.2. Inaccurate coefficients. The computation of $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$ requires the solution of an algebraic equation with a linear operator having the same nature of the original $\mathcal{L}$, but with smaller dimension. Up to a certain column dimension of P_k, the matrices $\boldsymbol{\alpha}_k$, $\boldsymbol{\beta}_k$ can be computed by solving the related linear systems in Kronecker form by means of a direct solver. Notice that the coefficient

matrix of such linear systems has to be assembled only once to compute both $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$. However, it may be the case, either because of the not-so-small dimension, or because of the complexity of the operator $\mathcal{L}$, the computation of $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$ may have to be done via an iterative procedure such as the truncated matrix-oriented CG. If this is the case, these quantities are computed inexactly, that is, the exact matrices are replaced by approximate quantities. We denote them as $\widetilde{\boldsymbol{\alpha}}_k = \boldsymbol{\alpha}_k + \boldsymbol{\epsilon}_k$, $\widetilde{\boldsymbol{\beta}}_k = \boldsymbol{\beta}_k + \boldsymbol{\eta}_k$. To ease the connection with the derivations of section 2 and section 4, in the following we use the symmetric notation for the factors. The inexact solutions $\widetilde{\boldsymbol{\alpha}}_k$ and $\widetilde{\boldsymbol{\beta}}_k$ no longer grant the orthogonality properties associated with the exact quantities $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$. In particular, by defining $\widetilde{\boldsymbol{R}}_{k+1} = \boldsymbol{R}_k - \mathcal{L}(P_k\widetilde{\boldsymbol{\alpha}}_k P_k^{\mathrm{T}})$, the orthogonality property $P_k^{\mathrm{T}}\widetilde{\boldsymbol{R}}_{k+1}P_k = 0$ is lost. Nonetheless, loss of local orthogonality can be tracked at each iteration, and directly related to the accuracy with which the small problems are solved. This is described in the following proposition.

PROPOSITION 6.1. *Let $\widetilde{\boldsymbol{\alpha}}_k$ be the approximate solution to $P_k^T\mathcal{L}(P_k\boldsymbol{\alpha}_k P_k^T)P_k = P_k^T\boldsymbol{R}_k P_k$, and let $\boldsymbol{\varrho}_k$ be the associated residual matrix. Then*

$$P_k^T\widetilde{\boldsymbol{R}}_{k+1}P_k = \boldsymbol{\varrho}_k,$$

where it also holds that $P_k^T\widetilde{\boldsymbol{R}}_{k+1}P_k = P_k^T(\widetilde{\boldsymbol{R}}_{k+1} - \boldsymbol{R}_{k+1})P_k$.

Proof. We notice that $\boldsymbol{\varrho}_k = P_k^{\mathrm{T}}\mathcal{L}(P_k\boldsymbol{\epsilon}_k P_k^{\mathrm{T}})P_k$. The residual recurrence gives $\widetilde{\boldsymbol{R}}_{k+1} = \boldsymbol{R}_k - \mathcal{L}(P_k\widetilde{\boldsymbol{\alpha}}_k P_k^{\mathrm{T}}) = \boldsymbol{R}_{k+1} - \mathcal{L}(P_k\boldsymbol{\epsilon}_k P_k^{\mathrm{T}})$. Hence, $P_k^{\mathrm{T}}\widetilde{\boldsymbol{R}}_{k+1}P_k = P_k^{\mathrm{T}}\boldsymbol{R}_{k+1}P_k - P_k^{\mathrm{T}}\mathcal{L}(P_k\boldsymbol{\epsilon}_k P_k^{\mathrm{T}})P_k = \boldsymbol{\varrho}_k$. ∎

A similar relation holds for the residual matrix associated with the reduced matrix equation yielding the coefficient $\widetilde{\boldsymbol{\beta}}_k$.

Inexactness also implies loss of orthogonality with respect to the previous iterates. However, in a context where all iteration factors are anyway truncated, we do not expect this truncation to have particularly strong implications. In most of our numerical experiments we compute $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$ *exactly*, by solving the related linear systems by a direct solver. Indeed, thanks to the rather moderate caps on the rank of the iterates we adopt, this operation does not remarkably affect the computational cost of the overall iterative solver.

7. Preconditioning. As the number of iterations increases, the iterates rank grows, possibly compelling a systematic use of truncation. Since information may be lost during truncation, strategies to accelerate convergence so as to decrease the number of iterations need to be devised. As in standard CG, preconditioning the coefficient operator is a natural strategy. Equipping Algorithm 3.1 with a preconditioner does not follow the same exact lines as what is done for matrix-oriented CG. Indeed, the different computation of $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$ leads to a different handling of the preconditioned quantities. We derive the transformed recurrence closely following the procedure in [12, Section 11.5.2] employed for vector CG.

Consider a preconditioning operator $\mathcal{P} : \mathbb{R}^{n_A\times n_B} \to \mathbb{R}^{n_A\times n_B}$ defined by symmetric matrices, and positive definite with respect to the matrix inner product. We restrict our attention to invertible operators such that

$$\mathcal{P}^{-1}(Y^l(Y^r)^{\mathrm{T}}) = [\boldsymbol{H}_1^l Y^l, \dots, \boldsymbol{H}_t^l Y^l]\boldsymbol{\kappa}[\boldsymbol{H}_1^r Y^r, \dots, \boldsymbol{H}_t^r Y^r]^{\mathrm{T}} = [\boldsymbol{H}_\star^l \bullet Y^l]\boldsymbol{\kappa}[\boldsymbol{H}_\star^r \bullet Y^r]^{\mathrm{T}}, \tag{7.1}$$

for some matrix $\boldsymbol{\kappa}$ of conforming dimensions. Under these hypotheses, there exists a nonsingular symmetric operator $\mathcal{G} : \mathbb{R}^{n_A\times n_B} \to \mathbb{R}^{n_A\times n_B}$ such that[§] $\mathcal{P}(\boldsymbol{X}) = \mathcal{G}(\mathcal{G}(\boldsymbol{X}))$. In place of $\mathcal{L}(\boldsymbol{X}) = \boldsymbol{C}$,

[§]In Kronecker form, $\mathcal{G}$ corresponds to the square root of the positive definite Kronecker form of $\mathcal{P}$.

we can thus solve the equivalent equation

$$\mathcal{G}^{-1}(\mathcal{L}(\mathcal{G}^{-1}(\mathcal{G}(\boldsymbol{X})))) = \mathcal{G}^{-1}(\boldsymbol{C}).$$

By setting $\widetilde{\mathcal{L}} = \mathcal{G}^{-1}\mathcal{L}\mathcal{G}^{-1} : \mathbb{R}^{n_A \times n_B} \to \mathbb{R}^{n_A \times n_B}$, $\widetilde{\boldsymbol{X}} = \mathcal{G}(\boldsymbol{X})$, and $\widetilde{\boldsymbol{C}} = \mathcal{G}^{-1}(\boldsymbol{C})$, we now apply the new scheme to the equation $\widetilde{\mathcal{L}}(\widetilde{\boldsymbol{X}}) = \widetilde{\boldsymbol{C}}$.

By starting with $\widetilde{\boldsymbol{R}}_0 = \widetilde{\boldsymbol{C}} - \widetilde{\mathcal{L}}(\widetilde{\boldsymbol{X}}_0) = \mathcal{G}^{-1}(\boldsymbol{C} - \mathcal{L}(\boldsymbol{X}_0))$ for a given $\widetilde{\boldsymbol{X}}_0$, and setting $\widetilde{\boldsymbol{P}}_0 = \widetilde{\boldsymbol{R}}_0$, the SS–CG iteration becomes

$$\widetilde{\boldsymbol{X}}_{k+1} = \widetilde{\boldsymbol{X}}_k + \widetilde{P}_k^l \widetilde{\boldsymbol{\alpha}}_k (\widetilde{P}_k^r)^{\mathrm{T}}, \quad \widetilde{\boldsymbol{R}}_{k+1} = \widetilde{\boldsymbol{C}} - \widetilde{\mathcal{L}}(\widetilde{\boldsymbol{X}}_{k+1}), \quad \widetilde{\boldsymbol{P}}_{k+1} = \widetilde{\boldsymbol{R}}_{k+1} + \widetilde{P}_k^l \widetilde{\boldsymbol{\beta}}_k (\widetilde{P}_k^r)^{\mathrm{T}}.$$

This can be rewritten as $\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + \mathcal{G}^{-1}(\mathcal{G}^{-1}(P_k^l \boldsymbol{\alpha}_k (P_k^r)^{\mathrm{T}})) = \boldsymbol{X}_k + \mathcal{P}^{-1}(P_k^l \boldsymbol{\alpha}_k (P_k^r)^{\mathrm{T}}) = \boldsymbol{X}_k + \widehat{P}_k^l \widehat{\boldsymbol{\alpha}}_k (\widehat{P}_k^r)^{\mathrm{T}}$, where $\widehat{P}_k^l \widehat{\boldsymbol{\alpha}}_k (\widehat{P}_k^r)^{\mathrm{T}}$ is the (possibly low rank) factorization of the result of applying $\mathcal{P}^{-1}$. Moreover, $\boldsymbol{R}_{k+1} = \boldsymbol{C} - \mathcal{L}(\boldsymbol{X}_{k+1})$ and

$$\mathcal{G}^{-1}(\boldsymbol{P}_{k+1}) = \mathcal{G}^{-1}(\boldsymbol{R}_{k+1}) + \mathcal{G}^{-1}(P_k^l \boldsymbol{\beta}_k (P_k^r)^{\mathrm{T}}).$$

By applying $\mathcal{G}^{-1}$ to the last relation we get $\mathcal{P}^{-1}(\boldsymbol{P}_{k+1}) = \mathcal{P}^{-1}(\boldsymbol{R}_{k+1}) + \mathcal{P}^{-1}(P_k^l \boldsymbol{\beta}_k (P_k^r)^{\mathrm{T}})$, namely

$$\widehat{\boldsymbol{P}}_{k+1} = \boldsymbol{Z}_{k+1} + \widehat{P}_k^l \widehat{\boldsymbol{\beta}}_k (\widehat{P}_k^r)^{\mathrm{T}}, \quad \text{where } \boldsymbol{Z}_{k+1} := \mathcal{P}^{-1}(\boldsymbol{R}_{k+1}).$$

REMARK 7.1. The computation $\mathcal{P}^{-1}(\boldsymbol{P}_{k+1}) = \mathcal{P}^{-1}(\boldsymbol{R}_{k+1}) + \mathcal{P}^{-1}(P_k^l \boldsymbol{\beta}_k (P_k^r)^{\mathrm{T}})$ above should in fact be understood as that the range of $\mathcal{P}^{-1}(\boldsymbol{P}_{k+1})$ equals the range of $\mathcal{P}^{-1}(\boldsymbol{R}_{k+1} + P_k^l \boldsymbol{\beta}_k (P_k^r)^{\mathrm{T}})$. In particular, by using (7.1), the range of $\mathcal{P}^{-1}(\boldsymbol{P}_{k+1})$ is obtained by using the range of $[\boldsymbol{H}_\star^l \bullet R_{k+1}, \boldsymbol{H}_\star^l \bullet P_k^l]$. The same holds for the transpose of $\mathcal{P}^{-1}(\boldsymbol{P}_{k+1})$. □

To sum up, the preconditioned variant of the subspace conjugate-gradient method (SS–CG) starts by setting $\boldsymbol{Z}_0 := \mathcal{P}^{-1}(\boldsymbol{R}_0)$ and $\boldsymbol{P}_0 = \boldsymbol{Z}_0$ and then at the kth iteration it proceeds as (in the final implementation each of these assignments will undergo a proper truncation step for generating the low-rank factors)

$$\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + P_k^l \boldsymbol{\alpha}_k (P_k^r)^{\mathrm{T}}, \quad \boldsymbol{R}_{k+1} = \boldsymbol{C} - \mathcal{L}(\boldsymbol{X}_{k+1})$$
$$\boldsymbol{Z}_{k+1} = \mathcal{P}^{-1}(\boldsymbol{R}_{k+1}), \quad \boldsymbol{P}_{k+1} = \boldsymbol{Z}_{k+1} + P_k^l \boldsymbol{\beta}_k (P_k^r)^{\mathrm{T}}.$$

The matrix $\boldsymbol{\alpha}_k$ is again the minimizer of $\phi(\boldsymbol{\alpha})$ in (3.4), with the newly "preconditioned" directions P_k^l, P_k^r. To maintain the $\mathcal{L}$-orthogonality of the directions $\boldsymbol{P}_i$'s, we compute $\boldsymbol{\beta}_k$ as the solution to the projected equation

$$(P_k^l)^{\mathrm{T}} \mathcal{L}(P_k^l \boldsymbol{\beta}_k (P_k^r)^{\mathrm{T}}) P_k^r = -(P_k^l)^{\mathrm{T}} \mathcal{L}(\boldsymbol{Z}_{k+1}) P_k^r.$$

We are left to select the actual preconditioning operator to perform $\boldsymbol{Z}_{k+1} = \mathcal{P}^{-1}(\boldsymbol{R}_{k+1}) = \mathcal{P}^{-1}(R_{k+1}^l \boldsymbol{\rho}_{k+1} (R_{k+1}^r)^{\mathrm{T}})$. Most preconditioning operators in the relevant literature are of the form

$$\mathcal{P}_1(\boldsymbol{X}) = \boldsymbol{E}\boldsymbol{X}\boldsymbol{D}, \quad \text{or} \quad \mathcal{P}_2(\boldsymbol{X}) = \boldsymbol{E}\boldsymbol{X}\boldsymbol{D} + \boldsymbol{F}\boldsymbol{X}\boldsymbol{G}.$$

Indeed, applying $\mathcal{P}^{-1}$ turns out to be affordable only when $\mathcal{P}$ is itself a linear operator with at most two terms; see, e.g., [8],[34],[26],[33],[32]. The operation $\mathcal{P}_1^{-1}$ corresponds to inverting $\boldsymbol{E}$ and $\boldsymbol{D}$, namely $\mathcal{P}_1^{-1}(\boldsymbol{X}) = \boldsymbol{E}^{-1}\boldsymbol{X}\boldsymbol{D}^{-1}$, thus perfectly matching the condition (7.1). On the other hand, to

comply with (7.1), the application of $\mathcal{P}_2^{-1}$ can be performed by using a method like low-rank ADI (LR-ADI) for Sylvester equations (see [6]), whose approximate solution can be shown to satisfy this expression; see [11, Proposition 3.1]. For completeness we mention that to the best of our knowledge, the only option where $\mathcal{P}$ has more than two terms is proposed in [34, Section 4] where $\mathcal{P}^{-1}$ is computed as an approximation to $\mathcal{L}^{-1}$ in Kronecker form with q (possibly sparse) terms.

In all our experiments, whenever $\mathcal{P}_2$ is employed we apply it by running t_{ADI} iterations of LR-ADI for Sylvester equations, where we use t_{ADI} (sub)optimal Wachspress' shifts [35]. According to (7.1), the resulting left and right factors will each have $t_{ADI} \cdot r_{k+1}$ columns whenever the input factor R_{k+1}^l has r_{k+1} columns. We stress that the algorithm in [35] uses the same shifts for the left and right sequences. Our implementation of LR-ADI is the same as the one proposed in [8], except for the matrix sparsification[¶]. We have not further modified the code, since we used $\mathcal{P}_2$ in a context when the left and right shifts could be the same.

To limit memory allocations, and according to what was already described for the application of the operator $\mathcal{L}$, we perform a *dynamic* low-rank truncation of the current iterate factors *at each* LR-ADI iteration. If this truncation is based on a `maxrank` policy (cf. section 6.1), this implementation of LR-ADI allocates at most $4 \cdot$`maxrank` columns (half of which for either the right or left factor) regardless of the number of iterations. Since this procedure can significantly worsen the preconditioner quality, it should only be adopted under severe storage constraints.

8. The complete algorithm. The new preconditioned subspace-conjugate gradient method (SS–CG) for multiterm Sylvester equations is summarized in Algorithm 8.1, equipped with the computational advances described in the previous sections.

Special attention deserves the stopping criterion. While the randomized procedure is able to remarkably reduce the memory requirements of the overall solution process, it does not construct an approximation of the norm of the residual matrix $\boldsymbol{C} - \mathcal{L}(\boldsymbol{X}_{k+1})$ that can be used to assess the accuracy of $\boldsymbol{X}_{k+1}$. As an alternative common choice (see, e.g., [27]), we monitor the difference between two consecutive approximate solutions as stopping criterion:

$$\|\boldsymbol{X}_{k+1} - \boldsymbol{X}_k\|_F / \|\boldsymbol{X}_{k+1}\|_F \leq \texttt{tol}. \tag{8.1}$$

The computation of the Frobenius norms exploits the fact that the columns X_j^l and X_j^r are kept orthonormal for every j, so that $\|X_{k+1}^l \boldsymbol{\tau}_{k+1} (X_{k+1}^r)^\mathrm{T}\|_F = \|\boldsymbol{\tau}_{k+1}\|_F$, and

$$\|X_{k+1}^l \boldsymbol{\tau}_{k+1} (X_{k+1}^r)^\mathrm{T} - X_k^l \boldsymbol{\tau}_k (X_k^r)^\mathrm{T}\|_F^2 = \|\boldsymbol{\tau}_{k+1}\|_F^2 + \|\boldsymbol{\tau}_k\|_F^2 - 2\mathrm{trace}(\boldsymbol{\tau}_{k+1}(X_{k+1}^r)^\mathrm{T} X_k^l \boldsymbol{\tau}_k (X_k^r)^\mathrm{T} X_{k+1}^l).$$

REMARK 8.1. *It is known that when used with iterative methods, the stopping criterion in (8.1) may be sensitive to the ill-conditioning of the problem, as a small relative difference does not necessarily correspond to a small residual. In our setting, however, we recall that stagnation of the approximate solution is more likely to occur as an intrinsic effect of the truncation step. The criterion (8.1) was chosen because computing the true residual norm is expensive, as previously discussed. Nonetheless, if one is interested in monitoring the residual, a careful implementation may consider including an estimation of the true residual norm once the criterion (8.1) is satisfied. In case such estimate is not satisfactorily small, the iteration will continue a few more steps, until the residual norm either converges or stagnates, or the maximum number of allowed iterations*

[¶]The code made available by the authors in the repository cited in [8] used matrices in full format, instead of sparse format.

Algorithm 8.1 Preconditioned subspace-conjugate gradient method (SS–CG)

Input: Operator $\mathcal{L} : \mathbb{R}^{n_A \times n_B} \to \mathbb{R}^{n_A \times n_B}$, preconditioner $\mathcal{P} : \mathbb{R}^{n_A \times n_B} \to \mathbb{R}^{n_A \times n_B}$, right-hand side factors C_1, C_2 ($\boldsymbol{C} = C_1 C_2^{\mathrm{T}}$), initial guess factors X_0^l, X_0^r, $\boldsymbol{\tau}_0$ ($\boldsymbol{X}_0 = X_0^l \boldsymbol{\tau}_0 (X_0^r)^{\mathrm{T}}$), max no. iterations `maxit`, tolerance `tol`, low-rank truncation tolerances `tolrank, maxrank, maxrankR`, flag `flag_rsvd`.
Output: Approximate solution factors X_k^l, X_k^r, $\boldsymbol{\tau}_k$ ($\boldsymbol{X}_k = X_k^l \boldsymbol{\tau}_k (X_k^r)^{\mathrm{T}}$) such that $\|\boldsymbol{X}_k - \boldsymbol{X}_{k-1}\| \leq \|\boldsymbol{X}_k\| \cdot$ `tol`

1: **if** `flag_rsvd` **then**
2: Create random Gaussians $G^l \in \mathbb{R}^{n_B \times \texttt{maxrankR}}$, $G^r \in \mathbb{R}^{n_A \times \texttt{maxrankR}}$ **else** Set $G^l = G^r = \emptyset$
3: **end if**
4: Set $[R_0^l, \boldsymbol{\rho}_0, R_0^r] = \mathcal{T}_{res}(C_1, C_2, X_0^l, \boldsymbol{\tau}_0, X_0^r, \texttt{params\_res})$ ▷ $(\ell\texttt{maxrank} + s_C)(n_A + n_B)$ or $\texttt{maxrankR}(n_A + n_B)$
5: Compute $[Z_0^l, \boldsymbol{\zeta}_0, Z_0^r] = \mathcal{T}(\mathcal{P}^{-1}(R_0^l \boldsymbol{\rho}_0 (R_0^r)^{\mathrm{T}}), \texttt{params})$ ▷ $\texttt{maxrank}(n_A + n_B)$ (at least)
6: Set $P_0^l = Z_0^l$, $P_0^r = Z_0^r$
7: **for** $k = 0, \ldots,$ `maxit` **do**
8: Compute $\boldsymbol{\alpha}_k$ by solving ▷ s_k^4

$$(P_k^l)^{\mathrm{T}} \mathcal{L}(P_k^l \boldsymbol{\alpha}_k (P_k^r)^{\mathrm{T}}) P_k^r = (P_k^l)^{\mathrm{T}} R_k^l \boldsymbol{\rho}_k (R_k^r)^{\mathrm{T}} P_k^r$$

9: Set $[X_{k+1}^l, \boldsymbol{\tau}_{k+1}, X_{k+1}^r] = \mathcal{T}([X_k^l, P_k^l], \mathrm{blkdiag}(\boldsymbol{\tau}_k, \boldsymbol{\alpha}_k), [X_k^r, P_k^r], \texttt{params})$ ▷ $2(n_A + n_B)\texttt{maxrank}$
10: **if** (8.1) holds **then**
11: Return $X_{k+1}^l, \boldsymbol{\tau}_{k+1}, X_{k+1}^r$
12: **end if**
13: Set $[R_{k+1}^l, \boldsymbol{\rho}_{k+1}, R_{k+1}^r] = \mathcal{T}_{res}(C_1, C_2, X_{k+1}^l, \boldsymbol{\tau}_{k+1}, X_{k+1}^r, \texttt{params\_res})$ ▷ $(\ell\texttt{maxrank} + s_C)(n_A + n_B)$ or ▷ $\texttt{maxrankR}(n_A + n_B)$
14: Compute $[Z_{k+1}^l, \boldsymbol{\zeta}_{k+1}, Z_{k+1}^r] = \mathcal{T}(\mathcal{P}^{-1}(R_{k+1}^l \boldsymbol{\rho}_{k+1} (R_{k+1}^r)^{\mathrm{T}}), \texttt{params})$ ▷ $\texttt{maxrank}(n_A + n_B)$ (at least)
15: Compute $\boldsymbol{\beta}_k$ by solving

$$(P_k^l)^{\mathrm{T}} \mathcal{L}(P_k^l \boldsymbol{\beta}_k (P_k^r)^{\mathrm{T}}) P_k^r = (P_k^l)^{\mathrm{T}} Z_{k+1}^l \boldsymbol{\zeta}_{k+1} (Z_{k+1}^r)^{\mathrm{T}} P_k^r$$

16: Set $[P_{k+1}^l, \boldsymbol{\gamma}_{k+1}, P_{k+1}^r] = \mathcal{T}([Z_{k+1}^l, P_k^l], \mathrm{blkdiag}(\boldsymbol{\zeta}_{k+1}, \boldsymbol{\beta}_k), [Z_{k+1}^r, P_k^r], \texttt{params})$ ▷ $2(n_A + n_B)\texttt{maxrank}$
17: **end for**

is reached. The estimation of the true residual 2-norm can be obtained – not inexpensively – by running a few iterations of a sparse SVD iterative method using the factorized version of $\boldsymbol{R}_{k+1}$. This variant is not included in the experiments below. However, the true residual norm was computed at completion for all considered methods.

Concerning the preconditioning step, when writing $\mathcal{P}^{-1}(R_{k+1}^l \boldsymbol{\rho}_{k+1} (R_{k+1}^r)^{\mathrm{T}})$ in lines 5 and 14 we mean that the preconditioner is applied as in (7.1), without explicitly assembling $R_{k+1}^l \boldsymbol{\rho}_{k+1} (R_{k+1}^r)^{\mathrm{T}}$. The result of this operation can be further truncated by the QR-SVD operator $\mathcal{T}$ in (6.1). The shorthand notations `params` and `params_res` for $\mathcal{T}$ and $\mathcal{T}_{res}$ resp., indicate the inclusion of all the necessary input parameters.

Finally, concerning memory requirements, in addition to all iterates' factors – each requiring (at most) $\texttt{maxrank}(n_A + n_B) + \texttt{maxrank}^2$ allocations – the method uses working storage, reported in Algorithm 8.1. In particular, the storage allocation needed by $\mathcal{T}_{res}$ (lines 4 and 13) depends on the adopted procedure: $(\ell\texttt{maxrank} + s_C)(n_A + n_B)$ for the QR-SVD truncation or $\texttt{maxrankR}(n_A + n_B)$ for the randomized strategy illustrated in section 6.1. Similarly, in line 5 and line 14 memory requirements for $\boldsymbol{Z}_k$ correspond to $\texttt{maxrankR}(n_A + n_B)$ for $\mathcal{P}_1$, and they will be higher when using $\mathcal{P}_2$. Memory requirements are comparable with those of TCG, except for s_k^4 in line 8 that is due to the allocation of the coefficient matrix of the Kronecker form of (3.5).

9. Numerical results. This section serves as introduction to the upcoming numerical experiments. All data are either publicly available or can be easily constructed. Our computational

analysis has two goals: first, we illustrate the properties of the new method. We explore how the choice of the maximum rank influences the performance. To this end, in all results we always set `tolrank`=10^{-12}. Moreover, we report the convergence behavior when the different strategies of section 6.1 are adopted to deal with memory requirements associated to the construction of the residual matrix.

Second, we compare the performance of our new method with that of state-of-the-art algorithms for the same problems. More precisely:

TPCG: truncated preconditioned CG, as recalled in[‖] section 2. The maximum allocated memory corresponds to twice `maxrank` long vectors for each recurrence, while for the residual computation we need to allocate ℓ`maxrank` $+ s_C$ long vectors.

SSS: Fixed-point iteration method for multiterm Lyapunov equations, with inexact solves with the leading portion of the operator, given by the first two terms[**] [28]. The allocated memory cannot be predicted a priori, as it depends on the memory required by the inner iterative projection solver. In our experiments this is reported a-posteriori, and it is generally unrelated to `maxrank`. Moreover, the final rank of the approximate solution is not fixed a-priori, and it is the result of a truncation to the small threshold 10^{-14}, as suggested in the original code.

R-NLCG: Optimization approach approximating the solution on manifolds of maximum-rank matrices through Riemannian Conjugate Gradient, with incorporation of preconditioners [8]. It will be used for multiterm Sylvester equations[††]. The algorithm terminates if the gradient norm drops below `tolgradnorm` $= 10^{-6}\|\boldsymbol{C}\|$ or if the norm of the displacement vector (to be retracted) is smaller than `minstepsize` $= 10^{-6}$. Memory allocations are not declared in the article. However scrutiny of the code seems to show that the memory employed is actually significant. For instance, the residual factors $\boldsymbol{A}_\star \bullet X_1$ and $\boldsymbol{B}_\star \bullet X_2$ are computed explicitly.

MultiRB: Projection method specifically designed for finite element discretizations of differential equations with stochastic inputs[‡‡] [27]. The method enforces a Galerkin condition for the spatial variables, with respect to a rational Krylov subspace specifically tailored to the problem, while the random space is not reduced. As the stopping criterion is based on tolerance, memory allocations and final rank can only be monitored a posteriori.

Unless explicitly stated, for SS–CG we consider both truncation variants of the residual matrix, that is the deterministic version with the factor fully allocated ($\mathcal{T}_{res}$ with $G^l = G^r = \emptyset$, labeled SS–CG *determ*) and the randomization-based one ($\mathcal{T}_{res}$ with $G^l = G^r \neq \emptyset$, labeled SS–CG *rand'zed*). In all instances, the solution of the matrix equations to determine $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$ was carried out with the problem in Kronecker form up to dimension 4000 of the Kronecker matrix. For all CG-type methods and MultiRB, the stopping criterion was based on (8.1) while a cheap bound is used for SSS; except for Example 9.3, the stopping tolerance was set to 10^{-6}. Algorithm R-NLCG used multiple stopping criteria, with values set above. This parameter tuning ensured that all the different solvers attain similar solutions, in general. In particular, the final true residual norm was computed at completion (but excluded in the total costs), to double check that all solutions have comparable accuracy. The running time is marked whenever the residual norm was smaller (over-solving) or larger (under-solving) by at least one order of magnitude with respect to those of the other methods.

[‖] A possible implementation is available at `http://www.dm.unibo.it/~simoncin/tcg.tar.gz`

[**] The Matlab code is publicly available at `https://www.dm.unibo.it/~simoncin/software.html`.

[††] The Matlab code is publicly available at `https://github.com/IvanBioli/riemannian-spdmatrixeq`

[‡‡] The Matlab code is publicly available at `https://www.dm.unibo.it/~simoncin/software.html`

All the experiments have been run using Matlab (version 2024b) on a machine with a 4.4GHz Intel 10-core CPU, including two high-performance cores and eight high-efficiency cores, equipped with i5 processor with 16GB RAM on an Ubuntu 2020.04.2 LTS operating system.

9.1. Numerical experiments for the multiterm Lyapunov equation. In this section we report a selection of results from our computational experience with Algorithm 8.1 applied to the multiterm Lyapunov equation

$$\boldsymbol{A}\boldsymbol{X} + \boldsymbol{X}\boldsymbol{A} + \boldsymbol{M}\boldsymbol{X}\boldsymbol{M} = \boldsymbol{C}. \tag{9.1}$$

For this problem, preconditioning is naturally two-term (cf. $\mathcal{P}_2$ in section 7), especially if the operator $\mathcal{L}_0 : \boldsymbol{X} \to \boldsymbol{A}\boldsymbol{X} + \boldsymbol{X}\boldsymbol{A}$ is, in some sense, the dominant part of the whole $\mathcal{L}$. Both examples below thus use $\mathcal{L}_0$ as preconditioner, and the action of its inverse is approximated as described in section 7.

EXAMPLE 9.1. This first example aims at illustrating the convergence behavior of the new method with respect to the chosen values of the parameters `maxrank` and `tol`. We thus focus on number of iterations as quality measure, postponing to subsequent experiments the use of running time. We consider the discretization by centered finite differences of the partial differential equation

$$(\theta(x)u_x)_x + (\theta(y)u_y)_y + \gamma(x,y)u = f, \quad \text{with } (x,y) \in (0,1)^2,$$

and Dirichlet zero boundary conditions. Here f is constant and equal to one in the whole domain, while $\theta(z) = -\frac{1}{10}\exp(-z)$. Each factor in the second order term can be discretized by a three-point stencil that deals with one-dimensional second order derivatives with nonconstant coefficients. By doing so, we obtain a matrix $\boldsymbol{A}$ which is symmetric, since we are working on the unit square discretized with a uniform mesh, and tridiagonal having components

$$\boldsymbol{A} = \frac{1}{h^2}\text{tridiag}(A_{i,i-1}, A_{i,i}, A_{i,i+1}), \qquad A_{i,i\pm1} = \theta(x_{i\pm\frac{1}{2}}),\ A_{i,i} = -(\theta(x_{i-\frac{1}{2}}) + \theta(x_{i+\frac{1}{2}})),$$

where the x_js are the discretization nodes in each direction, and $x_{j\pm\frac{1}{2}}$ are values at the midpoint of each discretization subinterval. The reactive coefficient $\gamma(x,y)$ is separable, that is $\gamma(x,y) = \gamma_0(x)\gamma_0(y)$, for two different settings:

i) $\gamma_0(z) = \sin(z\pi)$, with $z \in (0,1)$; ii) $\gamma_0(z) = \exp(z\pi)$, with $z \in (0,1)$.

At the discrete level, the reactive term can thus be represented by $\boldsymbol{M}\boldsymbol{X}\boldsymbol{M}$ with $\boldsymbol{M}$ diagonal and having on its diagonal the nodal values of γ_0. The discretization employs $n_A = 8000$ interior nodes in each direction so that $\boldsymbol{A}, \boldsymbol{M} \in \mathbb{R}^{n_A \times n_A}$.

The left plot In Figure 1 reports the leading singular value distribution of the solution $\boldsymbol{X}$ (obtained with higher accuracy than in the following experiments) for $\gamma_0(z) = \sin(z\pi)$ and $\gamma_0(z) = \exp(z\pi)$. The different singular value decay for the two cases is clearly visible, providing insight into what to expect when running truncation-based methods: the slower decay for $\gamma_0(z) = \exp(z\pi)$ suggests that the method will require higher rank iterates to be able to compute an accurate solution in this case. In other words, the stopping tolerance and the maximum rank cannot be selected disjointly.

Table 1 shows the performance of SS–CG in terms of number of iterations, for different choices of `maxrank` and the final `tol`. The two-term preconditioner $\mathcal{P}_2 \equiv \mathcal{L}_0$ was used, running $t_{ADI} = 8$ LR-ADI iterations. For this test case, no randomization is adopted, so as to analyze the fully

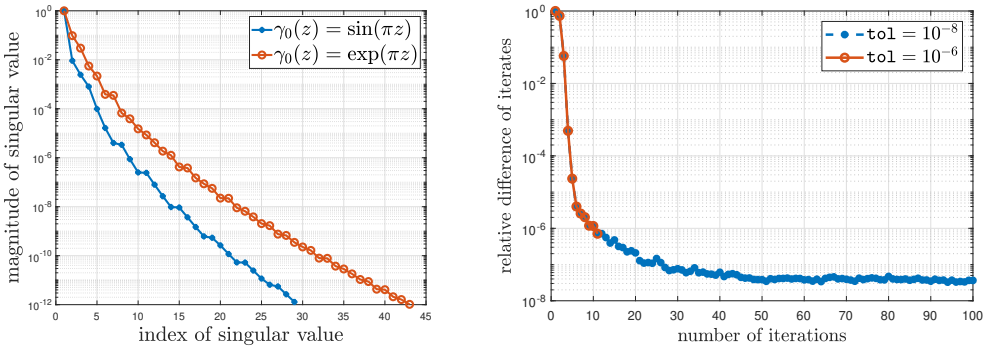

Fig. 1: Example 9.1. Left: Singular value distribution of the approximate solution matrix $\boldsymbol{X}$ for $\gamma_0(z) = \sin(z\pi)$ and $\gamma_0(z) = \exp(z\pi)$. Right: Convergence history of SS–CG for $\gamma_0(z) = \exp(z\pi)$, different stopping tolerances `tol`, and fixed `maxrank`=20.

γ_0	`maxrank`	`tol`	# iter
$\sin(z\pi)$	20	10^{-6}	5
	20	10^{-8}	7

γ_0	`maxrank`	`tol`	# iter
$\exp(z\pi)$	20	10^{-6}	10
	20	10^{-8}	–
	30	10^{-8}	17
	40	10^{-8}	5

Table 1: Example 9.1. Number of iterations for SS–CG as `maxrank` and final tolerance `tol` vary, for different reactive term coefficient γ_0. " – " stands for no convergence in 100 iterations.

deterministic setting. Truncation is driven by the `maxrank` parameter. For the reactive coefficient $\gamma_0(z) = \sin(z\pi)$, the fast singular value decay ensures convergence in few iterations for both tested tolerances for the chosen very small maximum rank. As suggested by the discussion above on the decay of the singular values of $\boldsymbol{X}$, the scenario changes for $\gamma_0(z) = \exp(z\pi)$: for `maxrank`= 20, SS–CG convergences rather fast if `tol`= 10^{-6}, but it is not able to meet the prescribed accuracy in 100 iterations if `tol`= 10^{-8}. The right plot of Figure 1 reports the convergence detail for `maxrank`=20 and the two different values of `tol`: for `tol`= 10^{-8} stagnation can be observed around the threshold, as this choice of `maxrank` is too small to capture the first `maxrank` singular values of $\boldsymbol{X}$ sufficiently well. As can be seen in Table 1, this issue gets fixed by increasing the value of `maxrank` with a remarkable reduction in the iteration count by increasing the maximum rank further. This pattern is typical in all our subsequent experiments, giving as feedback that if stagnation occurs, the value of `maxrank` should be increased or the tolerance `tol` relaxed. The joint selection of these two parameters is distinctive of truncation-based strategies.

EXAMPLE 9.2. We consider an example stemming from the control of dynamical systems, first discussed in [3] and then used in [28] for comparison purposes. The matrices correspond to the discretization of a heat model problem in the spatial domain $(0,1)^2$, so that $\boldsymbol{A}$ is the discretization of the 2D Laplace operator, and $\boldsymbol{M} = \boldsymbol{N}\boldsymbol{N}^{\mathrm{T}}$ is a low rank matrix (with rank the square root of the dimension of $\boldsymbol{A}$), allocating Robin conditions $\bar{\boldsymbol{n}} \cdot \nabla(x) = \delta u(x-1)$ on one of the domain boundaries, while zero Dirichlet conditions are imposed on the rest of the boundary. In our experiments we

Example	n_A	maxrank	SSS (iter/alloc/rank)	TPCG	SS–CG determ.	SS–CG rand'zed
HEAT1(0.5)	102400	20		61.35 (16)	– (100)	– (100)
		30	**6.15** (7/126/31)	22.78 (4)	17.93 (3)	18.17 (3)
	250000	20		– (100)	– (100)	– (100)
		30	**18.35** (7/139/31)	60.84 (4)	64.46 (4)	64.45 (4)
HEAT1(0.9)	102400	40		– (100)	– (100)	– (100)
		50		310.72 (26)	**58.11** (5)	58.52 (5)
		–	– (50/ /)			
	250000	50		2401.90 (93)	– (100)	– (100)
		60		936.39 (30)	**119.55** (4)	120.43 (4)
		–	– (50/ /)			

– no conv.

Table 2: Example 9.2. For each method, running time in seconds, and in parenthesis the number of iterations. Stopping tolerance 10^{-6}. For SSS, number of iterations, the subspace total memory allocation for length n_A vectors and the solution rank are reported.

consider $\delta \in \{0.5, 0.9\}$. The Lyapunov problem for $\delta = 0.5$ was called HEAT1 in [28]. We name HEAT1(δ) the two settings where $\delta = 0.5, 0.9$. We consider two discretization levels leading to a matrix problem of dimension $n_A = 320^2 = 102400$ as in [28], and $n_A = 500^2 = 250000$. Both SS–CG and TPCG are preconditioned by $\mathcal{L}_0 : \boldsymbol{X} \to \boldsymbol{AX} + \boldsymbol{XA}$ by running $t_{ADI} = 8$ LR-ADI iterations. For all considered methods, the accuracy tolerance is $\texttt{tol} = 10^{-6}$.

In Table 2 we report the results for both HEAT1(0.5) and HEAT1(0.9), and different values of n_A and maxrank. For HEAT1(0.5), the (standard) Lyapunov operator $\mathcal{L}_0$ is the dominant part of the whole $\mathcal{L}$. This is the scenario SSS has been designed for. Indeed, from the results in Table 2 we can see that SSS converges very fast for both values of n_A.

The operator $\mathcal{L}_0$ is less dominant in HEAT1(0.9). Indeed, SSS has convergence problems: after 50 fixed-point iterations its residual estimate is still above 10^{-3}. On the other hand, both TPCG and SS–CG converge for sufficiently large values of maxrank. In particular, SS–CG (both determ. and rand'zed) requires way fewer iterations than TPCG with consequent remarkable benefits in terms of computational timings, being about one order of magnitude faster than TPCG for this problem. Due to the small number of terms in $\mathcal{L}$ ($\ell = 3$), no notable performance difference of SS–CG *determ.* and *rand'zed* is observed in terms of either computational timings or memory allocations.

9.2. Numerical experiments for the multiterm Sylvester equation. In this section we consider the more general multiterm Sylvester equation in (1.1), with one-term or two-term preconditioning (cf. $\mathcal{P}_1$ and $\mathcal{P}_2$ in section 7, respectively).

EXAMPLE 9.3. We consider a problem used in [8, section 5.1], consisting of the parameterized diffusion equation $-\nabla \cdot (k\nabla u) = 0$ in $(0,1)^2$, with homogeneous boundary conditions and semi-separable diffusion coefficient

$$k(x,y) = \delta_1 k_{1,x}(x)k_{1,y}(y) + \ldots + \delta_{\ell_k} k_{\ell_k,x}(x)k_{\ell_k,y}(y), \quad \text{where} \quad k(x,y) = 1 + \sum_{j=1}^{\ell_k - 1} \frac{10^j}{j!} x^j y^j,$$

with $\ell_k = 4$. We insert $k(x,y)$ into the equation, and discretize the second order operator using

n	Precond type	maxrank	R-NLCG	TPCG	SS–CG determ.	SS–CG rand'zed
10000	$\mathcal{P}_1$	20	– (100)	– (100)	– (100)	– (100)
	$\mathcal{P}_1$	40	– (100)	– (100)	1.08 (5)	**0.92** (5)
	$\mathcal{P}_1$	60	– (100)	– (100)	2.47 (5)	**2.34** (5)
	$\mathcal{P}_2$	20	**11.25 (36)**	11.42 (38)	– (100)	– (100)
	$\mathcal{P}_2$	40	*42.97 (36)	**15.54** (33)	– (100)	– (100)
	$\mathcal{P}_2$	60	*98.62 (35)	32.39 (28)	9.59 (5)	**8.37** (5)
102400	$\mathcal{P}_1$	20	– (100)	– (100)	– (100)	– (100)
	$\mathcal{P}_1$	40	†	– (100)	18.17 (6)	**8.74** (6)
	$\mathcal{P}_1$	60	†	– (100)	23.50 (5)	**16.93** (5)
	$\mathcal{P}_2$	20	**183.44** (41)	– (100)	– (100)	– (100)
	$\mathcal{P}_2$	40	†	446.94 (47)	– (100)	– (100)
	$\mathcal{P}_2$	60	†	884.20 (26)	115.73 (3)	**101.91** (3)

– no conv. * Lower final residual norm than other methods † Out of Memory

Table 3: Example 9.3. For each method, running time in seconds, and in parenthesis the number of iterations. Stopping tolerance $\texttt{tol} = 5 \cdot 10^{-6}$.

standard finite differences (see Example 9.1). We then obtain the matrix equation

$$\sum_{j=1}^{\ell_k} \delta_j (A_{j,x} \boldsymbol{X} \boldsymbol{D}_{j,y} + \boldsymbol{D}_{j,y} \boldsymbol{X} A_{j,y}) = \boldsymbol{C}$$

where $\boldsymbol{C}$ is a rank-four matrix accounting for the boundary conditions; see [8]. For $\ell_k = 4$ a total of $\ell = 8$ terms appear in the matrix equation. We observe that the operator is a Lyapunov operator, however the overall equation is nonsymmetric, due to the nonsymmetry of the right-hand side matrix $\boldsymbol{C}$. In [8, Section 5.1], two-term preconditioning of the form $\mathcal{P}_2$ (cf. section 7) was employed, with the specific selection of the third and forth terms in $\mathcal{L}$. Following [8, Section 3.4], the preconditioning step in R-NLCG operates within a metric-related matrix inner product. For both SS–CG and TPCG, $t_{ADI} = 8$ LR-ADI iterations were employed for $n_A = 10000$ whereas $t_{ADI} = 15$ iterations were necessary for $n_A = 102400$. The stopping criterion used $\texttt{tol} = 5 \cdot 10^{-6}$ as threshold.

According to the discussion in section 7, for a large number ℓ of terms the use of one-term preconditioning $\mathcal{P}_1$ may be beneficial. For both SS–CG and TPCG, the left third and right forth terms in $\mathcal{L}$ were used to define $\mathcal{P}_1$. The results in Table 3 show that $\mathcal{P}_1$ is extremely effective for both problem dimensions when associated with SS–CG, giving at least one order of magnitude lower timings than with R-NLCG. The systematic low number of iterations for small maxrank is also remarkable. The use of $\mathcal{P}_1$ is thus recommended.

EXAMPLE 9.4. We consider a multiterm Sylvester equation arising from the Galerkin approximation of the following two dimensional elliptic PDE problem with correlated random inputs,

$$-\nabla \cdot (a(x,\omega)\nabla u) = f \quad \text{in } D, \quad u(x,\omega) = 0, \quad \text{on } \partial D. \tag{9.2}$$

Here $\omega \in \Omega$, where Ω is a sample space associated with a proper probability space; see, e.g., [21, Chapter 9], and $D \subset \mathbb{R}^2$ is the space domain. The diffusion coefficient is assumed to be a random field, with expansion in terms of a finite number of real-valued independent random variables $\{\xi_j\}_{j \le \ell-1}$ defined in Ω. Following the derivation in [27], we consider a truncated Karhunen-Loève expansion, giving $a(x,\omega) = \mu(x) + \sigma \sum_{j=1}^{\ell-1} \sqrt{\lambda_j} \phi_j(x) \xi_j(\omega)$, where μ corresponds to the diffusion

Example ($\ell = 10$)	n_A, n_B	`maxrank`	R-NLCG	MultiRB (spacedim/rank)	TPCG	SS–CG determ.	SS–CG rand'zed
[27, Ex.5.2]	16129, 1287	60	– (100)		– (100)	– (100)	– (100)
		125	67.54 (38)		53.50 (18)	19.55 (12)	**11.83** (13)
		150	57.31 (24)		66.88 (14)	23.73 (11)	12.58 (11)
				12.76 (312/306)			
[27, Ex.5.5]	16129, 2002	25	⋆5.58 (28)		6.55 (24)	– (100)	– (100)
		50	14.63 (26)		13.03 (16)	4.89 (8)	**3.20** (8)
		100	35.98 (25)		37.11 (16)	6.39 (6)	3.82 (6)
				6.89 (158/66)			

– no conv. ⋆ Final residual norm is *larger* than for other methods

Table 4: Example 9.4, stochastic problem. For each method, running time in seconds, and in parenthesis the number of iterations. Stopping tolerance `tol` $= 10^{-6}$. Best running times are in bold. For MultiRB the the final approximation space dimension and the final solution rank are reported.

coefficient expected value, σ is the standard deviation, while (λ_j, ϕ_j) are the leading eigenpairs of the associated covariance matrix. Under proper hypotheses on the coefficients, the problem is well posed, and its Galerkin finite element discretization on a tensor space (see [27]) gives an algebraic problem of type (1.1) with ℓ terms: the matrices $\boldsymbol{A}_i$ account for the spatial discretization terms, while the matrices $\boldsymbol{B}_i$ contain the discretized weighted moments in the random basis. The right-hand side is a rank-one matrix $\boldsymbol{C} = f_0 e_1^{\mathrm{T}}$, where f_0 is the finite element discretization of the forcing term in (9.2).

In our experiments we first consider the data corresponding to Example 5.2 and Example 5.1 in [27]; the second example was also used in [8]. The $\boldsymbol{A}_i$s have size $n_A = 16\,129$, while the $\boldsymbol{B}_i$s have size $n_B = 1287$ and $n_B = 2002$, respectively. The problems have $\ell = 9$ and $\ell = 10$ terms, resp. Explicit inspection (not reported here) shows that the solution of Example 5.2 in [27] has about 200 singular values with magnitude above 10^{-6}, from which we deduce that we cannot expect a very accurate approximate solution of small rank.

For this problem we also compare the performance with that of the algorithm MultiRB from [27], briefly recalled at the beginning of section 9. For this method, the final subspace dimension and the final solution rank are reported; both are recorded a-posteriori. Our experimental results are shown in Table 4, with stopping tolerance `tol` $= 10^{-6}$ and $\mathcal{P}_1$ preconditioning with $\boldsymbol{A}_1, \boldsymbol{B}_1$, as used in the literature for this problem. As expected, SS–CG requires a large value of `maxrank` to converge smoothly for the first problem. For smaller values, say `maxrank`=100, the convergence curve reached a plateau right above the requested tolerance, so that a slightly larger value of `tol` would have ensured a successful completion. Low memory allocations of the randomized strategy show that our new approach is also superior to MultiRB, in addition to R-NLCG, while being quite effective in terms of running time, for both cases. We should mention, however, that for the first problem, `maxrank`=125 produced a post-computed true residual norm larger than that obtained with MultiRB. Using `maxrank`=150 overcame the problem. Comparisons with the most direct competitor TPCG are consistent with the previous results. We highlight the particularly fast convergence of SS–CG for the second dataset, both in terms of number of iterations and running time.

We also created larger datasets for the setting of Example 5.1 in [27], using the S-IFISS package [29]; unless explicitly stated, all default values were used to create the problem data. We employed

n_A, n_B	decay	`maxrank`	R-NLCG	MultiRB (spacedim/rank)	TPCG	SS–CG determ.	SS–CG rand'zed
65025, 2002	fast	20	⋆50.39 (29)		– (100)	– (100)	– (100)
		30	82.81 (27)		27.65 (16)	10.61 (10)	**8.29** (11)
		40	111.98 (27)		38.02 (16)	13.54 (9)	9.30 (9)
				21.02 (159/66)			
65025, 2002	slow	40	⋆197.97 (45)		59.22 (24)	– (100)	– (100)
		50	194.77 (33)		41.26 (12)	21.25 (9)	13.05 (9)
		60	187.93 (24)		48.27 (11)	22.89 (8)	15.34 (8)
				10.04 (124/101)			
65025, 5005	fast	20	⋆29.21 (37)		– (100)	– (100)	– (100)
		30	33.06 (25)		38.09 (19)	14.79 (13)	11.02 (14)
		40	44.27 (25)		42.38 (17)	15.70 (10)	**10.66** (10)
				29.79 (159/72)			
65025, 5005	slow	40	⋆39.17 (22)		162.436 (56)	– (100)	– (100)
		50	45.52 (19)		50.15 (13)	24.41 (10)	14.79 (10)
		60	59.57 (18)		55.02 (12)	33.15 (9)	21.39 (9)
				13.32 (133/107)			

– no conv. ⋆ Final residual norm is *larger* than for other methods

Table 5: Example 9.4, stochastic problem, large dimensions. For each method, running time in seconds, and in parenthesis the number of iterations. For MultiRB, in parenthesis are approximation space dimension and final solution rank. Stopping tolerance `tol` $= 10^{-6}$. Best running times are in bold.

a finer spatial discretization so that $n_A = 65\,025$ (level 8), and used 9 random variables with polynomial degree 5, so that $n_B = 2002$, as above. For this setting we used both fast and slow decay of the expansion coefficients; we refer to [27] for a discussion. Finally, we consider the 'fast' and 'slow' decay problems with 9 random variables and polynomial degree 6, yielding $n_B = 5005$. All results with these newly created data are reported in Table 5.

In spite of the broader scenario, the results are consistent with the previous ones, with the new method largely surpassing its more direct competitors in all instances. The comparison with respect to MultiRB is less clear cut, if only running time is considered, while overall the new method requires less memory. We conclude with a comment on the expected behavior for larger n_B on this problem. Since MultiRB provides no reduction in the stochastic variable, the costs of the method will significantly increase with n_B, whereas we expect less dramatic effects on the new method.

Finally, in Table 6 we test the performance of the memory-saving dynamic residual computation described in section 6.1 with respect to the full computation of the residual factor, on this last dataset. Although the number of iterations seems to not have been affected, this is not so for the running time, which in many cases more than doubles. Similar results were obtained with the previous examples whenever ℓ was large. Summarizing, and given the especially good behavior of the randomized memory-saving strategy we have devised, we do not advocate using the dynamic strategy, at least for the classes of problems we have tested it.

10. Conclusions. We have proposed a new iterative method for solving multiterm matrix equations with symmetric and positive definite operator. The method generates a sequence of approximate solutions by locally minimizing a functional over a subspace that is allowed to grow up to a desired threshold. The derivation closely follows that of matrix-oriented Conjugate Gradients on the Kronecker form, without being affected by the same dramatic loss of optimality. By using

n_A, n_B	decay	maxrank	SS–CG determ.	SS–CG dyn.	n_A, n_B	decay	maxrank	SS–CG determ.	SS–CG dyn.
65025, 2002	fast	30	10.61 (10)	30.76 (10)	65025, 5005	fast	30	14.79 (13)	47.33 (13)
		40	13.54 (9)	36.31 (9)			40	15.70 (10)	47.63 (10)
	slow	50	21.25 (9)	61.41 (9)		slow	50	24.41 (10)	66.93 (10)
		60	22.89 (8)	65.44 (8)			60	33.15 (9)	70.77 (9)

Table 6: Example 9.4, stochastic problem, large dimensions. Comparison between storing the whole residual factor and dynamically updating its truncated QR factorization. For each method, running time in seconds, and in parenthesis the number of iterations. Stopping tolerance $\texttt{tol} = 10^{-6}$.

particularly convenient randomized range-finding strategies, the method is able to ensure low memory requirements. Our numerical experiments have shown that the new method is computationally robust and competitive with respect to state-of-the-art methods in addition to TCG, on quite diverse application problems.

The matlab code of SS–CG is available at `https://github.com/palittaUniBO`.

Acknowledgments. The authors thank the reviewers for their careful reading of the manuscript and their insightful comments. All authors are members of the INdAM Research Group GNCS. Moreover, their work was partially supported 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.

REFERENCES

[1] A. C. Antoulas, *Approximation of large-scale Dynamical Systems*, Advances in Design and Control, SIAM, Philadelphia, 2005.
[2] B. Beckermann, D. Kressner, and C. Tobler, *An error analysis of Galerkin projection methods for linear systems with tensor product structure*, SIAM J. Numer. Anal., 51 (2013), pp. 3307–3326.
[3] P. Benner and T. Breiten, *Low rank methods for a class of generalized Lyapunov equations and related issues*, Numer. Math., 124 (2013), pp. 441–470, https://doi.org/10.1007/s00211-013-0521-0.
[4] P. Benner, A. Cohen, M. Ohlberger, and K. Willcox, eds., *Model reduction and approximation: theory and Algorithms*, Computational Science & Engineering, SIAM, PA, 2017.
[5] P. Benner and T. Damm, *Lyapunov equations, energy functionals, and model order reduction of bilinear and stochastic systems*, SIAM J. Control Optim., 49 (2011), pp. 686–711.
[6] P. Benner, R.-C. Li, and N. Truhar, *On the ADI method for Sylvester equations*, Journal of Computational and Applied Mathematics, 233 (2009), pp. 1035–1045, https://doi.org//10.1016/j.cam.2009.08.108.
[7] P. Benner, A. Onwunta, and M. Stoll, *Low-rank solution of unsteady diffusion equations with stochastic coefficients*, SIAM/ASA J. Uncertainty Quantification, 3 (2015), pp. 622–649.
[8] I. Bioli, D. Kressner, and L. Robol, *Preconditioned low-rank riemannian optimization for symmetric positive definite linear matrix equations*, SIAM J. Sci. Comput., 47 (2025), pp. A1091–A1116, https://doi.org/10.1137/24M1688540.
[9] T. Damm, *Direct methods and ADI-preconditioned Krylov subspace methods for generalized Lyapunov equations*, Num. Lin. Alg. with Appl., 15 (2008), pp. 853–871. Special issue on Matrix equations.
[10] S. Dolgov and M. Stoll, *Low-rank solution to an optimization problem constrained by the Navier–Stokes equations*, SIAM J. Sci. Comput., 39 (2017), pp. A255–A280.
[11] V. Druskin, L. Knizhnerman, and V. Simoncini, *Analysis of the rational Krylov subspace and ADI methods for solving the Lyapunov equation*, SIAM J. Numer. Anal., 49 (2011), pp. 1875–1898.
[12] G. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore,

4th ed., 2013.
[13] L. Grubišić and D. Kressner, *On the eigenvalue decay of solutions to operator Lyapunov equations*, Systems & Control Letters, 73 (2014), pp. 42–47.
[14] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*, SIAM Review, 53 (2011), pp. 217–288, https://doi.org/10.1137/090771806.
[15] J. Henning, D. Palitta, V. Simoncini, and K. Urban, *An ultraweak space-time variational formulation for the wave equation: Analysis and efficient numerical solution*, ESAIM: M2AN, 56 (2022), pp. 1173–1198, https://doi.org/10.1051/m2an/2022035.
[16] E. Jarlebring, G. Mele, D. Palitta, and E. Ringh, *Krylov methods for low-rank commuting generalized Sylvester equations*, Num. Lin. Alg. Appl, 25 (2018), https://doi.org/10.1002/nla.2176.
[17] 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.
[18] D. Kressner and P. Sirković, *Truncated low-rank methods for solving general linear matrix equations*, Numerical Linear Algebra with Applications, 22 (2015), pp. 564–583, https://doi.org/10.1002/nla.1973.
[19] 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, https://doi.org/10.1137/100799010.
[20] G. Loli, M. Montardini, G. Sangalli, and M. Tani, *An efficient solver for space–time isogeometric Galerkin methods for parabolic problems*, Computers & Mathematics with Applications, 80 (2020), pp. 2586–2603, https://doi.org/https://doi.org/10.1016/j.camwa.2020.09.014.
[21] G. J. Lord, C. E. Powell, and T. Shardlow, *An introduction to computational stochastic PDEs*, Cambridge University Press, 2014.
[22] D. P. O'Leary, *The block conjugate gradient algorithm and related methods*, Linear Algebra and its Applications, 29 (1980), pp. 293–322, https://doi.org/10.1016/0024-3795(80)90247-5.
[23] D. Palitta and P. Kürschner, *On the convergence of Krylov methods with low-rank truncations*, Numer Algor, 88 (2021), pp. 1383–1417, https://doi.org/10.1007/s11075-021-01080-2.
[24] D. Palitta and V. Simoncini, *Optimality Properties of Galerkin and Petrov–Galerkin Methods for Linear Matrix Equations*, Vietnam J. Math., 48 (2020), p. 791–807, https://doi.org/10.1007/s10013-020-00390-7.
[25] K. B. Petersen and M. S. Pedersen, *The matrix cookbook*, Oct. 2008, http://www2.imm.dtu.dk/pubdb/p.php?3274. Version 20081110.
[26] C. E. Powell and H. C. Elman, *Block-diagonal preconditioning for spectral stochastic finite-element systems*, IMA J. Numer. Anal., 29 (2009), pp. 350–375, https://doi.org/10.1093/imanum/drn014.
[27] C. E. Powell, D. Silvester, and V. Simoncini, *An Efficient Reduced Basis Solver for Stochastic Galerkin Matrix Equations*, SIAM J. Sci. Comput., 39 (2017), pp. A141–A163, https://doi.org/10.1137/15M1032399.
[28] S. D. Shank, V. Simoncini, and D. B. Szyld, *Efficient low-rank solutions of generalized Lyapunov equations*, Numerische Mathematik, 134 (2016), pp. 327–342.
[29] D. Silvester, A. Bespalov, and C. Powell, *S-IFISS version 1.0*, 2014, https://personalpages.manchester.ac.uk/staff/david.silvester/ifiss/sifiss.html.
[30] V. Simoncini, *Computational Methods for Linear Matrix Equations*, SIAM Review, 58 (2016), pp. 377–441, https://doi.org/10.1137/130912839.
[31] V. Simoncini and Y. Hao, *Analysis of the truncated conjugate gradient method for linear matrix equations*, SIAM. J. Matrix Anal. & Appl., 44 (2023), pp. 359–381, https://doi.org/10.1137/22M147880X.
[32] M. Stoll and T. Breiten, *A low-rank in time approach to PDE-constrained optimization*, SIAM J. Sci. Comput., 37 (2015), pp. B1–B29.
[33] E. Ullmann, *A Kronecker product preconditioner for stochastic Galerkin finite element discretizations*, SIAM J. Sci. Comput., 32 (2010), pp. 923–946, https://doi.org/10.1137/080742853.
[34] Y. Voet, *Preconditioning Techniques for Generalized Sylvester Matrix Equations*, Numerical Linear Algebra with Applications, 32 (2025), p. e70020, https://doi.org/10.1002/nla.70020.
[35] E. Wachspress, *The ADI Model Problem*, Springer, 2013, https://doi.org/10.1007/978-1-4614-5122-8.

Appendix. In this Appendix we report in Algorithm 10.1 the low-rank truncation scheme for the residual matrix presented in section 6.1.

Algorithm 10.1 $\mathcal{T}_{res}$ - truncation procedure for the residual matrix

Input: Operator $\mathcal{L} : \mathbb{R}^{n_A \times n_B} \to \mathbb{R}^{n_A \times n_B}$, factors C_1, C_2 ($\boldsymbol{C} = C_1 C_2^{\mathrm{T}}$), factors of current approx sol'n X^l, X^r, $\boldsymbol{\tau}$ ($\boldsymbol{X} = X^l \boldsymbol{\tau} (X^r)^{\mathrm{T}}$), matrices Ω and Π, truncation tolerances `tolrank, maxrank`.
Output: Approximate factors R^l, R^r, $\boldsymbol{\rho}$ to the residual matrix $\boldsymbol{C} - \mathcal{L}(\boldsymbol{X})$

(Short-hand not'n: $A_\star \bullet X^l = [A_1 X^l, \ldots, A_\ell X^l]$, $B_\star \bullet X^r = [B_1 X^r, \ldots, B_\ell X^r]$, $\boldsymbol{D} = \mathrm{blkdiag}(I_{s_C}, -\boldsymbol{\tau}, \ldots, -\boldsymbol{\tau})$)
if $\Omega = \Pi = \emptyset$ **then**
    $[R^l, \boldsymbol{\rho}, R^r] = \mathcal{T}([C_1, A_\star \bullet X^l], \boldsymbol{D}, [C_2, B_\star \bullet X^r], \mathtt{tolrank}, \mathtt{maxrank})$
else
    Compute skinny QRs(†)

$$[Q, *] = \mathrm{QR}([C_1, A_\star \bullet X^l]\,(\boldsymbol{D}([C_1, B_\star \bullet X^r]^{\mathrm{T}}\Omega))), \quad [G, *] = \mathrm{QR}([C_2, B_\star \bullet X^r]\,\Big(\boldsymbol{D}([C_1, A_\star \bullet X^l]^{\mathrm{T}}\Pi)\Big))$$

    Compute truncated SVD based on `tolrank` and `maxrank`: $U\Sigma V \approx Q^{\mathrm{T}}[C_1, A_\star \bullet X^l]\boldsymbol{D}[C_2, B_\star \bullet X^r]^{\mathrm{T}} G$
    Set $R^l = QU$, $\boldsymbol{\rho} = \Sigma$, $R^r = GV$
end if

(†) The product $[C_2, B_\star \bullet X^r]^{\mathrm{T}}\Omega$ is performed one block at the time, so as not to explicitly form $B_\star \bullet X^r$. The same for all other similar products in the algorithm.